REXYGEN www.rexygen.com

Function Blocks of REXYGEN
Reference manual

REX Controls s.r.o.

Version 3.0
2023-12-15
Plzeň (Pilsen), Czech Republic

Contents

Note: Only a partial documentation is available in blocks marked by * .

Chapter 1

Introduction

The manual "REXYGEN system function blocks" is a reference manual for the REXYGEN system function block library RexLib. It includes description and detailed information about all function blocks RexLib consists of.

1.1 How to use this manual

The extensive function block library RexLib, which is a standard part of the REXYGEN system, is divided into smaller sets of logically related blocks, the so-called *categories* (sublibraries). A separate chapter is devoted to each category, introducing the general properties of the whole category and its blocks followed by a detailed description of individual function blocks.

The content of individual chapters of this manual is following:

1 **Introduction**
This introductory chapter familiarizing the readers with the content and ordering of the manual. A convention used for individual function blocks description is presented.

2 **EXEC – Runtime executive configuration**
Blocks used mainly for configuration of the structure, priorities and timing of individual objects linked to the real-time subsystem of the REXYGEN system (the RexCore program) are described in this chapter.

3 **INOUT – Input and output blocks**
This sublibrary consists of the blocks used mainly for the REXYGEN system. These blocks provide the connection between the control tasks and input/output drivers.

4 **MATH – Mathematic blocks**
The blocks for arithmetic operations and basic math functions.

21 **MC_MULTI – Multi-axes motion control**
Function blocks of this sublibrary were developed according to the PLCopen Motion Control standard for motion control in multiple axes.

22 **MC_COORD – Coordinated motion control**
Function blocks of this sublibrary were developed according to the PLCopen Motion Control standard for coordinated motion control.

16 **SPEC – Special blocks**
The most interesting blocks of this sublibrary are the `REXLANG` and `RDC` blocks. It is possible to compile and interpret user algorithms using the `REXLANG` block, whose programming language is very similar to the C language (the syntax of the `REXLANG` commands is mostly the same as in the C language). The `RDC` block can be used for real-time communication between two REXYGEN-enabled target devices.

The individual chapters of this reference guide are not much interconnected, which means they can be read in almost any order or even only the necessary information for specific block can be read for understanding the function of that block. The electronic version of this manual (in the `.pdf` format) is well-suited for such case as it is equipped with hypertext bookmarks and contents, which makes the look-up of individual blocks very easy.

Despite of that it is recommended to read the following subchapter, which describes the conventions used for description of individual blocks in the rest of this manual.

1.2 The function block description format

The description of each function block consists of several sections (in the following order):

Block Symbol – displays the graphical symbol of the block

Function Description – brief description of the block function, omitting too detailed information.

Inputs – detailed description of all inputs of the block

Outputs – detailed description of all outputs of the block

Parameters – detailed description of all parameters of the block

Examples – a simple example of the use of the block in the context of other blocks and optional graph with input and output signals for better understanding of the block function.

If the block function is obvious, the section Examples is omitted. In case of block with no input or no output the corresponding section is omitted as well.

The inputs, outputs and parameters description has a tabular form:

`<name>`	[*nam*] Detailed description of the input (output, parameter) `<name>`. Mathematical symbol *nam* on the right side of the first column is used in the equations in the Function Description section. It is listed only if it differs from the name more than typographically. If the variable value is limited to only enumerated values, the meaning of these values is explained in this column. [⊙`<def>`] [↓`<min>`] [↑`<max>`]	`<type>`

The meaning of the three columns is quite obvious. The third column contains the item `<type>`. The REXYGEN control system supports the types listed in table 1.1. But the most frequently used types are `Bool` for Boolean variables, `Long (I32)` for integer variables and `Double (F64)` for real variables (in floating point arithmetics).

Each described variable (input, output or parameter) has a default value `<def>` in the REXYGEN system, which is preceded by the ⊙ symbol. Also it has upper and lower limits, preceded by the symbols ↓ and ↑ respectively. All these three values are optional (marked by [ ]). If the value ⊙`<def>` is not listed in the second column, it is equal to zero. If the values of ↓`<min>` and/or ↑`<max>` are missing, the limits are given by the the minimum and/or maximum of the corresponding type, see table 1.1[1].

Type	Meaning	Minimum	Maximum
`Bool`	Boolean value 0 or 1	0	1
`Byte (U8)`	8-bit integer number without the sign	0	255
`Short (I16)`	16-bit integer number with the sign	-32768	32767
`Long (I32)`	32-bit integer number with the sign	-2147483648	2147483647
`Large (I64)`	64-bit integer number with the sign	$-9.2234 \cdot 10^{18}$	$9.2234 \cdot 10^{18}$
`Word (U16)`	16-bit integer number without the sign	0	65535
`DWord (U32)`	32-bit integer number without the sign	0	4294967295
`Float (F32)`	32-bit real number in floating point arithmetics	$-3.4 \cdot 10^{38}$	$3.4 \cdot 10^{38}$
`Double (F64)`	64-bit real number in floating point arithmetics	$-1.7 \cdot 10^{308}$	$1.7 \cdot 10^{308}$
`String`	character string		

Table 1.1: Types of variables in the REXYGEN system.

1.3 Conventions for variables, blocks and subsystems naming

Several conventions are used to simplify the use of the REXYGEN control system. All used variable types were defined in the preceding chapter. The term variable refers to function block inputs, outputs and parameters in this chapter. The majority of the blocks uses only the following three types:

[1]Precise range of the `Large` data type is -9223372036854775808 to 9223372036854775807.

`Bool` – for two-state logic variables, e.g. on/off, yes/no or true/false. The logic one (yes, true, on, 1) is referred to as `on` in this manual. Similarly the logic zero (no, false, off, 0) is represented by `off`. This holds also for REXYGEN Studio. Other tools and 3rd party software may display these values as `1` for `on` and `0` for `off`. The names of logic variables consist of uppercase letters, e.g. `RUN`, `YCN`, `R1`, `UP`, etc.

`Long (I32)` – for integer values, e.g. set of parameters ID, length of trend buffer, type of generated signal, error code, counter output, etc. The names of integer variables use usually lowercase letters and the initial character (always lowercase) is in most cases {`i`, `k`, `l`, `m`, `n`, or `o`}, e.g. `ips`, `l`, `isig`, `iE`, etc. But several exceptions to this rule exist, e.g. `cnt` in the `COUNT` block, `btype`, `ptype1`, `pfac` and `afac` in the `TRND` block, etc.

`Double (F64)` – for floating point values (real numbers), e.g. gain, saturation limits, results of the majority of math functions, PID controller parameters, time interval lengths in seconds, etc. The names of floating point variables use only lowercase letters, e.g. `hilim`, `y`, `ti`, `tt`.

The function block names in the REXYGEN system use uppercase letters, numbers and the `'_'` (underscore) character. It is recommended to append a lowercase user-defined string to the standard block name when creating user instances of function blocks.

It is explicitly not recommended to use diacritic and special characters like spaces, CR (end of line), punctuation, operators, etc. in the user-defined names. The use of such characters limits the transferability to various platforms and it can lead to incomprehension. The names are checked by the REXYGEN Compiler compiler which generates warnings if inappropriate characters are found.

1.4 The signal quality corresponding with OPC

Every signal (input, output, parameter) in the REXYGEN system has the so-called *quality flags* in addition to its own value of corresponding type (table 1.1). The quality flags in the REXYGEN system correspond with the OPC (OLE for Process Control) specification [1]. They can be represented by one byte, whose structure is explained in the table 1.2.

Bit number	7	6	5	4	3	2	1	0
Bit weight	128	64	32	16	8	4	2	1
Bit field	**Quality**		**Substatus**				**Limits**	
	Q	Q	S	S	S	S	L	L
`BAD`	0	0	S	S	S	S	L	L
`UNCERTAIN`	0	1	S	S	S	S	L	L
not used in OPC	1	0	S	S	S	S	L	L
`GOOD`	1	1	S	S	S	S	L	L

Table 1.2: The quality flags structure

The basic quality type is determined by the QQ flags in the two most important bits. Based on these the quality is distinguished between `GOOD`, `UNCERTAIN` and `BAD`. The four SSSS bits provide more detailed information about the signal. They have different meaning for each basic quality. The two least significant bits LL inform whether the value exceeded its limits or if it is constant. Additional details and the meaning of all bits can be found in [1], chapter 6.8.

Chapter 2

EXEC – Real-time executive configuration

Contents

ALARMS – Alarms Definition Configuration

Block Symbol

Licence: STANDARD

Function Description

The ALARMS is placed in the *main project file* and allows the user to configure list of alarms. Alarms are activated by the ALM or ALMI blocks. Alarms are defined in a .csv (Comma separated variable) file. The afile parameter contains the file name of the .csv file. An alarm could be activated also by the ALB, ALBI, ALN, ALNI blocks, but these blocks not use definitions in the ALARMS block.

The configuration file has the following columns:

- id ... Unique alarm reference number. The number is used in the ALM block, in archive records, etc.
- level ... The value stored into an archive record (in the level field).
- archives ... Bit field – identifies archives for recording events associated with the alarm (alarm starts, ends, acknowledges). E.g. 0 = not stored in the archive, 1 = stored in the 1st archive, 2 = stored in the 2nd archive, 4 = stored in the 3rd archive, 3 = stored in the 1st and 2nd archives, etc.
- group ... Reserved for future use, now some number (or bitfield up to 64 bits) to filter alarm's list in HMI.
- name ... Name of the alarm; can be used as the alarm identifier, so it should be unique.
- description ... Text description of the alarm. It is possible to insert formatting characters for multilingual texts and to insert values associated with the alarm associated values) in the text, see below.

Multilingual support

REXYGEN supports multilingual alarm description. The description field must be in the form:

<lang1_ID>:<lang1 text>|<lang2_ID>:<lang2 text>|<lang3_ID>:<lang3 text>

Number of languages is not limited, but total size of the field is limited to 32765 bytes (english characters). The lang1 (language 1) is used if the user sets unsupported language. Example: let's expect the description field in the form: cz:Přepětí|en:High voltage alarm. The user will see High voltage alarm if the language is set to en. The user will see Přepětí if the language is set to cz. The user will see Přepětí in all other cases (for example if the language is set to de, cze, EN, en-us, etc.).

Associated values

The desription field can contain special marks that is replaced by values from control algorithm – so-called associated values. The mark has the form:

`%<value number>[<format>][:<number of characters>[:<precision>]]`

where the format is one of the following characters:

- `b`, `B` ... binary value (string `on` or `off` is shown)
- `d`, `D` ... integer number shown as decimal string, the default value for integer types
- `x`, `X` ... integer number shown as hexadecimal string
- `f`, `F` ... real number in fix point form, the precision of it is a number of digits behind decimal point (if precision is specified)
- `e`, `E` ... real number in exponential (scientific) form
- `g`, `G` ... the same as `F` or `E` (depends on actual value), the default format for real number types
- `s`, `S` ... text string

The default type is used if the format is not specified or if the type of the value is not compatible with the specified format. More characters than it is specified is used if it is necessary to show the correct value.

Format Examples:

- `%2` ... value of 2nd variable (e.g. `av2` in the `ALM` block)
- `%1:8:2` ... value of 1st variable (e.g. `av1` in the `ALM` block), 2 characters behind decimal point, total 8 characters (leading spaces are used if necessary)
- `%1e` ... value of 1st variable (e.g. `av1` in the `ALM` block) in exponential form

The `ALB, ALBI` blocks not use associated values. The `ALN` and `ALNI` maps it this way:

- 1 ... value of the `u` input
- 2 ... value of the `h` parameter (input)
- 3 ... value of the `hh` parameter (input)
- 4 ... value of the `l` parameter (input)
- 5 ... value of the `ll` parameter (input)
- 6 ... value of the `tout` parameter (input)

Remarks:

- It is possible to use comma or semicolon as a separator in the `.csv` file. The first row with column names is optional.
- Alarms (lines) in the file must be in the ascending order respect to the `id`.
- The `id` must be unique including other alarming/archiving blocks (`TRND`, `ALB`, `ALN`, ...).
- It is possible to use the internal editor (the `Configure` button in parametric dialog) or external tool. Internal editor generates a correct example if the `.csv` file does not exist.

- The blocks ALB, ALBI, ALN and ALNI regard lvl > 127 as an event, where only its begin (nor end nor acknoledge) is stored into archives. The blocks ALM, ALMI do not implement this event function.
- Alarm's associated values are stored into alarm's value when alarm is triggered (begin). Later changes of the associated values are not updated in an alarm window in HMI.
- Alarm window in HMI can show also alarm name. It is the name of the block (without block type if it prefixes the block name) that is connected to the alarm.
- The whole description string is displayed, if client sets the empty language (e.g. "").

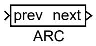
Parameters

afile	file name of an alarm's definition .csv file	String

ARC – The REXYGEN system archive

Block Symbol

Licence: STANDARD



Function Description

The ARC block is intended for archives configuration in the REXYGEN control system. The archives can be used for continuous recording of alarms, events and history trends directly on the target platform. The output Archives of the EXEC block must be connected to the prev input of the first archive. The following archives can be added by connecting the input prev with the preceding archive's output next. Only one archive block can be connected to each next output, the output of the last archive remains unconnected. The resulting archives sequence determines the order of allocation and initialization of individual archives in the REXYGEN system and also the index of the archive, which is used in the arc parameter of the archiving blocks (see chapter 10). The archives are numbered from 1 and the maximum number of archives is limited to 15 (archive no. 0 is the internal system log).

The atype parameter determines the type of archive from the data-available-after-restarting point of view. The admissible types depend on the target platform properties, which can be inspected in the Diagnostics section of the REXYGEN Studio program after successful connecting to the target device.

Archive consists of sequenced variable-length items (memory and disk space optimization) with a timestamp. Therefore the other parameters are the total archive size in bytes asize and maximum number of timestamps nmarks for speeding-up the sequential seeking in the archive.

The frequency of writing values to disk can be influenced by the period parameter. For devices using flash memory or SD cards as a disk, it is not suitable to write values too often, therefore it is appropriate to set this parameter to a value in the order of minutes. Furthermore, it is possible to select a suitable source of time stamps with the timesrc parameter.

Input

prev	Input for connecting with the next output of the preceding archive or with the Archives output of the EXEC block in the case of the first archive	Long (I32)

Output

next	Output for creating sequences of archives by connecting to the prev input of the following archive	Long (I32)

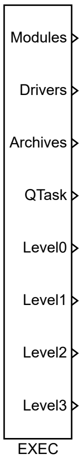
Parameters

`atype`	Archive type ⊙1 1 ..... archive is allocated in the RAM memory (data is irreversibly lost after restarting the target device) 2 ..... archive is allocated in backed-up memory, e.g. CMOS (data remains available after restarting the target device) 3 ..... archive is allocated on a drive (data remains available in the file after restarting)	`Long (I32)`
`asize`	Size of the archive in bytes ↓256 ⊙102400	`Long (I32)`
`nmarks`	Number of time stamps for speeding-up sequential seeking in the archive ↓2 ⊙720	`Long (I32)`
`ldaymax`	Maximum size of archive per day [bytes] ↓1000 ↑2147480000 ⊙1048576	`Large (I64)`
`period`	Period of writing data to disk [s] ⊙60.0	`Double (F64)`
`timesrc`	Source of timestamps ⊙1 1 ..... `CORETIMER` – technological time – at current tick 2 ..... `CORETIMER-PRECISE` – technological time – at block execution 3 ..... `RTC` – real time clock (wallclock) from operating system – at current tick 4 ..... `RTC-PRECISE` – real time clock (wallclock) from operating system – at block execution 4 ..... `PFC` – raw high precision time (PerFormanceCounter)	`Long (I32)`

EXEC – Real-time executive

Block Symbol

Licence: STANDARD



Function Description

The EXEC block is a cornerstone of the so-called *project main file* in the .mdl format, which configures individual subsystems of the REXYGEN system. No similar block can be found in the Matlab-Simulink system. The EXEC block and all connected configuration blocks do not implement any mathematic algorithm. Such configuration structure is used by the REXYGEN Compiler compiler during building of the overall REXYGEN control system application.

The REXYGEN system configuration consists of modules (Modules), input/output drivers (Drivers), archive subsystem (Archives) and real-time subsystem, which includes quick computation tasks (see the QTASK function block description for details) and four priority levels (Level0 to Level3) for inserting computation tasks (see the TASK function block description for details).

The base (shortest) period of the application is determined by the tick parameter. This value is checked by the REXYGEN Compiler compiler as its limits vary by selected target platform. Generally speaking, the lower period is used, the higher computational requirements of the REXYGEN system runtime core (RexCore) are.

The periods of individual computation levels (Level0 to Level3) are determined by multiplying the base period tick by the parameters ntick0 to ntick3. Parameters pri0 to pri3 are the logical priorities of corresponding computation levels in the REXYGEN system. The REXYGEN system uses 32 logical priorities, which are internally mapped to the target platform operating system dependent priorities. The highest logical priority of the REXYGEN system is 0, the value 31 means the lowest. Should two tasks with different priorities run at the same time, the lower priority (higher value) task would be interrupted by the higher priority (lower value) task.

The default priorities `pri0` to `pri3` reflect the commonly accepted idea that the "fast" tasks (short sampling period) should have higher priority than the "slow" ones (the so-called *Rate monotonic scheduling*). This means that the default priorities need not to be changed in most cases. Impetuous changes can lead to unpredictable effects!

In devices with multiple CPUs, it is possible to assign different levels to various CPUs. The assignment of CPUs is managed using the parameters `cpu0` to `cpu3`. The CPUs are numbered starting from 0, where -1 denotes the default setting.

Outputs

`Modules`	Output for connecting the REXYGEN system expansion modules, see the `MODULE` function block description for details	`Long (I32)`
`Drivers`	Output for connecting the REXYGEN system input/output drivers, see the `IODRV` and `TIODRV` function block descriptions for details	`Long (I32)`
`Archives`	Output for archives configuration, see the `ARC` block	`Long (I32)`
`QTask`	Output for connecting quick tasks with the highest priority and the shortest period, see the `QTASK` block	`Long (I32)`
`Level0`	Computation level for inserting tasks (see the `TASK` block) with high priority `pri0` and short period determined by the `ntick0` parameter	`Long (I32)`
`Level1`	Computation level for inserting tasks with medium priority `pri1` and medium-length period determined by the `ntick1` parameter	`Long (I32)`
`Level2`	Computation level for inserting tasks with low priority `pri2` and long period determined by the `ntick2` parameter	`Long (I32)`
`Level3`	Computation level for inserting tasks with the lowest priority `pri3` and the longest period determined by the `ntick3` parameter	`Long (I32)`

Parameters

`target`	Target device `Generic target device` ⊙`Generic target device`	`String`
`tick`	The base period (tick) of the REXYGEN system core and also the quick task (`QTASK`) period (in seconds) ⊙`0.05`	`Double (F64)`
`ntick0`	The multiplication `tick*ntick0` determines the period of tasks connected to `Level0` ↓`1` ⊙`10`	`Long (I32)`
`ntick1`	The multiplication `tick*ntick1` determines the period of tasks connected to `Level1` ↓`ntick0+1` ⊙`50`	`Long (I32)`
`ntick2`	The multiplication `tick*ntick2` determines the period of tasks connected to `Level2` ↓`ntick1+1` ⊙`100`	`Long (I32)`
`ntick3`	The multiplication `tick*ntick3` determines the period of tasks connected to `Level3` ↓`ntick2+1` ⊙`1200`	`Long (I32)`
`pri0`	Priority of all `Level0` tasks ↓`3` ↑`31` ⊙`5`	`Long (I32)`
`pri1`	Priority of all `Level1` tasks ↓`pri0+1` ↑`31` ⊙`9`	`Long (I32)`

`pri2`	Priority of all `Level2` tasks	↓`pri1+1` ↑31 ⊙13	`Long (I32)`
`pri3`	Priority of all `Level3` tasks	↓`pri2+1` ↑31 ⊙18	`Long (I32)`
`cpu0`	Level0 tasks CPU core (-1=default, 0=core 0, 1=core 1, ...)	↓-1 ↑127 ⊙-1	`Long (I32)`
`cpu1`	Level1 tasks CPU core (-1=default, 0=core 0, 1=core 1, ...)	↓-1 ↑127 ⊙-1	`Long (I32)`
`cpu2`	Level2 tasks CPU core (-1=default, 0=core 0, 1=core 1, ...)	↓-1 ↑127 ⊙-1	`Long (I32)`
`cpu3`	Level3 tasks CPU core (-1=default, 0=core 0, 1=core 1, ...)	↓-1 ↑127 ⊙-1	`Long (I32)`

HMI – Human-Machine Interface Configuration

Block Symbol

Licence: STANDARD

Function Description

The HMI block is a so-called "pseudo-block" which stores additional settings and parameters related to the Human-Machine Interface (HMI) and the contents of the internal web server. The only file where the block can be placed is the main project file with a single EXEC block.

The REXYGEN system currently provides three straightforward methods of how to create Human-Machine Interface:

- **WebWatch** is an auto-generated HMI from the REXYGEN Studio development tool during project compilation. It has similar look, attributes and functions as the online mode of the REXYGEN Studio development tool. The main difference is that **WebWatch** is stored on the target device, is available from the integrated web server and may be viewed with any modern web browser or any application that is compatible with HTML, SVG and JavaScript. The **WebWatch** is a perfect tool for instant creation of HMI that is suitable for system developers or integrators. It provides a graphical interaction with almost all signals in the control algorithm.

- **WebBuDi**, which is an acronym for Web Buttons and Displays, is a simple JavaScript file with several declarative blocks that describe data points which the HMI is connected to and assemble a table in which all the data is presented. It provides a textual interaction with selected signals and is suitable for system developers and integrators or may serve as a fall-back mode HMI for non-standard situations.

- **RexHMI** is a standard SVG file that is edited using REXYGEN HMI Designer. The REXYGEN HMI Designer is a great tool for creating graphical HMI that is suitable for operators and other end users.

The IncludeHMI parameter includes or excludes the HMI files from the final binary form of the project. The HmiDir specifies a path to a directory where the final HMI is located and from where it is inserted into the binary file during project compilation. The path may be absolute or relative to the project. The GenerateWebWatch specifies whether a **WebWatch** HMI should be generated into HmiDir during compilation. The GenerateRexHMI specifies whether a **RexHMI** and **WebBuDi** should be generated into HmiDir during compilation.

The logic of generating and including HMI during project compilation is as follows:

1. Delete all contents from `HmiDir` when `GenerateWebWatch` or `GenerateRexHMI` is specified.

2. Generate RexHMI and WebBuDi from `SourceDir` into `HmiDir` if `GenerateRexHMI` is enabled. All **WebBuDi** source files should be named in a `*.hmi.js` format and all **RexHMI** source files should be named in a `*.hmi.svg` format. The generated files are then named `*.html`.

3. Copy all contents from `SourceDir` except **WebBuDi** or **RexHMI** source files into `HmiDir` if `IncludeHMI` is enabled.

4. Insert HMI from `HmiDir` into binary configuration if `IncludeHMI` is enabled.

The block does not have any inputs or outputs. The `HMI` block itself does not become a part of the final binary configuration, only the files it points to do. Be careful when inserting big files or directories as the integrated web server is not designed for massive data transfers. It is possible to shrink the data by enabling gzip compression. The compression also reduces amount of data transferred to the client, but decompression must be performed by the server when a client does not support gzip compression, which brings additional load on the target device.

For a proper operation of the `HMI` block the compilation must be launched from the REXYGEN Studio development tool and the REXYGEN HMI Designer must be installed.

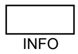
Parameters

`IncludeHMI`	Include HMI files in the project	⊙on	Bool
`HmiDir`	Output folder for HMI files	⊙hmi	String
`SourceDir`	Source directory	⊙hmisrc	String
`GenerateWebWatch`	Generate WebWatch HMI files	⊙on	Bool
`GenerateRexHMI`	Generate HMI from SVG and JS files	⊙on	Bool
`RedirectToHMI`	Web server will automatically redirect to HMI webpage if enabled otherwise it will serve a standard home page as a starting page.	⊙on	Bool
`Compression`	Enables data compression in gzip format.		Bool

INFO – Description of Algorithm

Block Symbol

Licence: STANDARD



Function Description

The INFO block is a so-called "pseudo-block" which stores textual information about a real-time executive. The only file where the block can be placed is a main project file with a single EXEC block an so it belongs to the EXEC category. The block does not have any inputs or outputs. The information specified with this block becomes a part of the final configuration, is stored on the target device and may be seen on different diagnostics screens but does not have any impact on execution of the control algorithm or target's behavior.

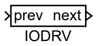
Parameters

Title	Project title	String
Author	Project author	String
Description	Brief description of the project	String
Customer	Information about a customer	String

IODRV – The REXYGEN system input/output driver

Block Symbol

Licence: STANDARD



Function Description

The input/output drivers of the REXYGEN system are implemented as extension modules (see the MODULE block). A module can contain several drivers, which are added to the REXYGEN system configuration by using the IODRV blocks. The prev input of the block must be connected with the Drivers output of the EXEC block or with the next output of a IODRV block which is already included in the configuration. There can be only one driver connected to the next output of the IODRV block. The next output of the last driver in the configuration remains unconnected. This means that the drivers create a unidirectional chain which defines the order of initialization and execution of the individual drivers.

Each driver of the REXYGEN system is identified by its name, which is defined by the classname parameter (beware, the name is case-sensitive!). If the name of the driver differs from the name of the module containing the given driver, the module name must be specified by the module parameter, it is left blank otherwise. Details about these two parameters can be found in the documentation of the corresponding REXYGEN system driver.

The majority of drivers stores its own configuration data in files with .rio extension (REXYGEN Input/Output), whose name is specified by the cfgname parameter. The .rio files are created in the same directory where the project main file is located (.mdl file with the EXEC block). Driver is configured (e.g. names of the input/output signals, connection to physical inputs/outputs, parameters of communication with the input/output device, etc.) in an embedded editor provided by the driver itself. The editor is opened when the Configure button is pressed in the parameter dialog of the IODRV block in the REXYGEN Studio program of the REXYGEN control system. In Matlab/Simulink the editor is opened upon ticking the "Tick this checkbox to call IOdrv EDIT dialog" checkbox.

The remaining parameters are useful only when the driver implements its own computational task (see the corresponding driver documentation). The factor parameter defines the driver's task execution period by multiplying the EXEC block's tick parameter factor times (factor*tick). The stack parameter defines the stack size in bytes. It is recommended to keep the default setting unless stated otherwise in the driver documentation. The parameter pri defines the logical priority of the driver's task. Inappropriate priority can influence the overall performance of the control system critically so it is highly recommended to check the driver documentation and the load of the control system (drivers, levels and tasks) in the Diagnostics section of the REXYGEN Studio program. The cpu parameter can be used to specify where the driver thread should run on multi-CPU devices.

Input

prev	Input for connecting the driver with the Drivers output of the EXEC block or with the next output of the preceding driver	Long (I32)

Output

next	Output for connecting to the prev input of the succeeding driver	Long (I32)

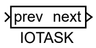
Parameters

module	Name of the module, which includes the input/output driver (mandatory only if module name differs from classname)	String
classname	I/O driver class name; case sensitive! ⊙DrvClass	String
cfgname	Name of the driver configuration file ⊙iodrv.rio	String
factor	Multiple of the EXEC block's tick parameter defining the driver's task execution period ↓1 ⊙10	Long (I32)
stack	Stack size of the driver's task in bytes ↓1024 ⊙10240	Long (I32)
pri	Logical priority of the driver's task ↓1 ↑31 ⊙3	Long (I32)
cpu	CPU core assigned to driver thread (-1=default, 0=core 0, 1=core 1, ...) ↓-1 ↑127 ⊙-1	Long (I32)

IOTASK – Driver-triggered task of the REXYGEN system

Block Symbol

Licence: STANDARD



Function Description

Standard tasks of the REXYGEN system are integrated into the configuration using the TASK or QTASK blocks. Such tasks are executed by the system timer, whose **tick** is configured by the EXEC block.

But the system timer can be unsuitable in some cases, e.g. when the shortest execution period is too long or when the task should be executed by an external event (input signal interrupt) etc. In such a case the IOTASK can be executed directly by the I/O driver configured by the TIODRV block. The user manual of the given driver provides more details about the possibility and conditions of using the above mentioned approach.

Input

prev	Input for connecting the first task to the Tasks output of the TIODRV block or for connecting to the previous task's next output	Long (I32)

Output

next	Output for sequencing the tasks by connecting to the prev input of the following task	Long (I32)

Parameters

factor	Execution factor which can be used to determine the task execution period, see the user guide of the corresponding I/O driver ⊙1	Long (I32)
stack	Stack size [bytes] ⊙10240	Long (I32)
filename	Name of the file with the .mdl extension which contains the task algorithm; in the case filename is not specified, the filename is given by the name of the IOTASK block in the project main file (the .mdl extension is attached automatically)	String

LPBRK – Loop break

Block Symbol

Licence: STANDARD

Function Description

The LPBRK block is an auxiliary block often used in the control schemes consisting of the REXYGEN system function blocks. The block is usually placed in all feedback loops in the scheme. Its behavior differs in the REXYGEN system and the Simulink system.

The LPBRK block creates a one-sample delay in the Simulink system. If there exists a feedback loop without the LPBRK block, the Simulink system detects an algebraic loop and issues a warning (Matlab version 6.1 and above). The simulation fails after some time.

The REXYGEN Compiler omits the LPBRK block, the only effect of this block is the breaking of the feedback loop at the block's position. If there exists a loop without the LPBRK block, the REXYGEN Compiler compiler issues a warning and breaks the loop at an automatically determined position. It is recommended to use the LPBRK block in all loops to achieve the maximum compatibility between the REXYGEN system and the Simulink system.

Note: Behavior of the LPBRK block has been changed since the version 3.0. The block is not removed by the REXYGEN Compiler but is present in the algorithm and clears the quality flag of the y output. This change is useful and necessary due to the quality propagation in function blocks. Original behaviour (e.g. the block is removed from the algorithm) can be forced by the RB = on parameter. The main function of the block (indication of the feedback signal) remains unchanged in all cases.

Input

u	Input signal	Double (F64)

Output

y	Output signal	Double (F64)

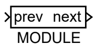
Parameters

RB	Removing block flag	Bool

MODULE – Extension module of the REXYGEN system

Block Symbol

Licence: STANDARD



Function Description

The REXYGEN system has an open architecture thus its functionality can be extended. Such extension is provided by modules. Each module is identified by its name placed below the block symbol. The individual modules are added to the project main file by connecting the **prev** input with the **Modules** output of the EXEC block or with the **next** output of a **MODULE** which is already included in the project. There can be only one module connected to the **next** output of the **MODULE** block. The **next** output of the last module in the project remains unconnected. This means that the modules create a unidirectional chain which defines the order of initialization of individual modules.

Input

prev	Input for connecting the module with the **Modules** output of the EXEC block or with the **next** output of the preceding module	**Long (I32)**

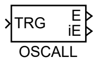
Output

next	Output for connecting to the **prev** input of the succeeding module	**Long (I32)**

OSCALL – Operating system calls

Block Symbol

Licence: STANDARD



Function Description

The OSCALL block is intended for executing operating system functions from within the REXYGEN system. The chosen action is performed upon a rising edge (off→on) at the TRG input. However, not all actions are supported on individual platforms. The result of the operation and the possible error code are displayed by the E and iE outputs.

Note that there is also the EPC block available, which allows execution of external programs.

Input

TRG	Trigger of the selected action	Bool

Outputs

E	Error flag	Bool
iE	Error code i ..... REXYGEN general error	Long (I32)

Parameter

action	System function to perform 1 ..... Reboot system 2 ..... System shutdown 3 ..... System halt 4 ..... Flush disc caches 5 ..... Lock system partition 6 ..... Unlock system partition 7 ..... Disable internal webserver 8 ..... Enable internal webserver	⊙1	Long (I32)

PROJECT – Additional Project Settings

Block Symbol

Licence: STANDARD

Function Description

The PROJECT block is a so-called "pseudo-block" which stores additional settings and parameters related to a project and a real-time executive. The only file where the block can be placed is a main project file with a single EXEC block an so it belongs to the EXEC category.

The block does not have any inputs or outputs. The block does not become a part of the final binary configuration.

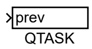
Parameters

CompileParams	Command-line options that are passed to REXYGEN Compiler during project compilation. To display the documentation for the available parameters, run REXYGEN Compiler from the command line with the parameter -?: RexComp -?.		String
SourcesOnTarget	Store source files on target device	⊙on	Bool
TargetURL	URL address of a target on which the configuration should be run. The address is inserted into all connection dialogs automatically.		String
LibraryPath	Path to libraries referenced in the project. Can be absolute or relative to project folder.		String

QTASK – Quick task of the REXYGEN system

Block Symbol

Licence: STANDARD



Function Description

The QTASK block is used for including the so-called quick task with high priority into the executive of the REXYGEN system. This task is used where the fastest processing of the input signals is necessary, e.g. digital filtering of input signals corrupted with noise or immediate processing of switches connected via digital inputs. The quick task is added into the configuration by connecting the prev input with the EXEC block's QTask output. The quick task is initialized before the initialization of the Level0 computation level (see the TASK block).

There can be only one QTASK block in the REXYGEN control system. It runs with the logical priority no. 2. The algorithm of the quick task is configured the same way as the standard TASK, it is a separate .mdl file.

The execution period of the task is given by a multiple of the factor parameter and the tick of the EXEC block. The task is executed with the shortest period of tick seconds for factor=1. In that case the system load is the highest. Under all circumstances the QTASK must be executed within tick seconds, otherwise a real-time executive fatal error occurs and no other tasks are executed. Therefore the QTASK block must be used with consideration. The block's execution time is shown in the Diagnostics section of the REXYGEN Studio program.

Input

prev	Input for connecting the task with the QTask output of the EXEC block	Long (I32)

Parameters

factor	Multiple of the EXEC block's tick parameter defining the quick task execution period ⊙1	Long (I32)
stack	Stack size [bytes] ⊙10240	Long (I32)
filename	Name of the file with the .mdl extension which contains the quick task algorithm; in the case filename is not specified, the filename is given by the name of the QTASK block in the project main file (the .mdl extension is attached automatically)	String

SLEEP – Timing in Simulink

Block Symbol

Licence: STANDARD

Function Description

The Matlab/Simulink system works natively in simulation time, which can run faster or slower than real time, depending on the complexity of the algorithm and the computing power available. Therefore the SLEEP block must be used when accurate timing and execution of the algorithm in the Matlab/Simulink system is required. In the REXYGEN system, timing and execution is provided by system resources (see the EXEC block) and the SLEEP block is ignored.

In order to perform real-time simulation of the algorithm, the SLEEP block must be included. It guarantees that the algorithm is executed with the period given by the ts parameter unless the execution time is longer than the requested period.

The SLEEP block is implemented for Matlab/Simulink running in Microsoft Windows operating system. It is recommended to use periods of 100 ms and above. For the proper functionality the 'Solver type' must be set to `fixed-step` and `discrete (no continuous states)` in the 'Solver' tab of the 'Simulation parameters' dialog. Further the `Fixed step size` parameter must be equal to the ts parameter of the SLEEP block. There should be at most one SLEEP block in the whole simulation scheme (including all subsystems).

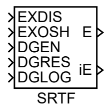
Parameter

ts	Simulation scheme execution period (in seconds)	⊙0.1	Double (F64)

SRTF – Set run-time flags

Block Symbol

Licence: ADVANCED



Function Description

The SRTF block (Set Run-Time Flags) can be used to influence the execution of tasks , subsystems (sequences) and blocks of the REXYGEN system. This block is not meant for use in Matlab-Simulink. When describing this block, the term object refers to a REXYGEN system object running in real-time, i.e. input/output driver, one of the tasks, subsystem or a simple function block of the REXYGEN system.

All the operations described below affect the object, whose full path is given by the bname parameter. Should the parameter be left blank (empty string), the operation applies to the nearest owner of the SRTF object, i.e. the subsystem in which the block is directly included or the task containing the block.

The run-time flags allow the following operations:

- **Disable execution** of the object by setting the EXDIS input to on. The execution can be enabled again by using the input signal EXDIS = off. The EXDIS input sets the same run-time flag as the Halt/Run button in the upper right corner of the Workspace tab in the Diagnostics of the REXYGEN Studio program.

- **One-shot execution** of the object. If the object execution is disabled by the EXDIS = on input or by the Diagnostics section of the REXYGEN Studio program, it is possible to trigger one-shot execution by EXOSH = on.

- **Enable diagnostics** for the given object by DGEN = on. The result is equivalent to ticking the Enable checkbox in the Diagnostics section of the corresponding tab (I/O Driver, Level, Quick Task, Task, I/O Task, Sequence) of the REXYGEN Studio program.

- **Reset diagnostic data** of the given object by DGRES = on. The same flag can be set by the Reset button in the Diagnostics section of the corresponding tab in the REXYGEN Studio program. The flag is automatically set back to 0 when the data reset is performed.

The following table shows the flags available for various objects in the REXYGEN system.

Object type	EXDIS	EXOSH	DGEN	DGRES
I/O Driver	√	√	√	√
Level	√	×	√	√
Task	√	√	√	√
Quick Task	√	√	√	√
I/O Task	√	√	√	√
Sequence, subsystem	√	×	√	√
Block	√	×	×	×

Inputs

EXDIS	Disable execution	Bool
EXOSH	One-shot execution	Bool
DGEN	Enable diagnostics	Bool
DGRES	Reset diagnostic data	Bool
DLOG	Enable more verbose logging	Bool

Outputs

E — Error flag — Bool
- off ... No error
- on .... An error occurred

iE — Error code (for E = on) — Long (I32)
- 0 ..... No error
- 1 ..... The object specified by the bname parameter was not found
- 2 ..... REXYGEN system internal error (invalid pointers)
- 3 ..... Flag could not be set (timeout)

Parameter

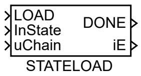
bname — Full path to the block/object. Case sensitive. Individual layers are separated by dots, the object names excluding tasks (TASK, QTASK) start with the following special characters: — String
- ^ ..... Computational level, e.g. ^0 for Level0
- & ..... Input/Output Driver, e.g. &WcnDrv

Name of the task triggered by input/output driver (IOTASK) has the form &<driver_name>.<task_name>.

STATELOAD – Load multiple block states and parameters

Block Symbol

Licence: ADVANCED



Function Description

The STATELOAD block reloads values of state and parameters from a file or string. The file is specified by the filename parameter and must be in JSON format, usually stored by the STATESAVE block. It is also possible to read data from the InState input, which is a JSON string in the same format as the input file. The InState input is used if the filename parameter is empty.

All values configured by the parameters blocks, depth, and mask that are stored in the file are loaded. If the parameter Strict is set to on, the block checks if the configured blocks and values match those stored in the file, and the file is refused if there is a mismatch.

Inputs

LOAD	Trigger to load the state	Bool
InState	JSON string to load if the filename parameter is empty	String
uChain	This input is not used by the block but is useful for placing the block in the correct execution order	Long (I32)

Parameters

filename	Filename from which to load	String
blocks	List of blocks to load. The block name must be a relative connection string (e.g. beginning with a dot) and they are separated by semicolons. All blocks (in the current subsystem) are loaded if this parameter is empty	String
depth	If the loading block is a subsystem, this parameter specifies how many levels are also loaded. 0 = current level only, 1 = current level and blocks directly in the current level subsystems, etc. ↓0 ↑65535	Long (I32)

`mask`	Select which objects are loaded. Each bit of the number signifies: • 1 ... inputs • 2 ... outputs • 4 ... parameters • 8 ... internal states • 16 ... array parameters • 32 ... array states • 64 ... cyclic (trend) buffers • 256 ... metadata (STATESAVE only) ↓0 ↑65535 ⊙65535	`Long (I32)`
`LoadOnInit`	The file is loaded during the configuration initialization ⊙on	`Bool`
`STRICT`	If set, the file is checked against the current configuration and data are refused if there is a mismatch ⊙on	`Bool`

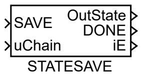
Outputs

`DONE`	State has been loaded	`Bool`
`iE`	Error code if block execution fails	`Error`

STATESAVE – Save multiple block states and parameters

Block Symbol

Licence: ADVANCED



Function Description

The STATESAVE block stores the values of states and parameters in a file. The file is specified by the **filename** parameter and is in JSON format, which can usually be reloaded by the STATELOAD block. It is also possible to store data in the OutState output, which is a JSON string in the same format as the file output. The OutState output is used if the **filename** parameter is empty.

All values configured by the parameters **blocks**, **depth**, and **mask** are stored.

Inputs

SAVE	Trigger to save the state	Bool
uChain	This input is not used by the block but is useful for placing the block in the correct execution order.	Long (I32)

Parameters

filename	Filename where to store	String
blocks	List of blocks to store. Block names must be relative connection strings (e.g., begin with a dot) and are separated by semicolons. All blocks (within the current subsystem) are stored if the parameter is empty.	String
depth	If the saved block is a subsystem, this parameter specifies the number of levels to save. 0 = current level only, 1 = current level and blocks directly in the current level subsystems, etc. ↓0 ↑65535	Long (I32)

mask	Select which objects are saved. Each bit of the number represents: • 1 ... inputs • 2 ... outputs • 4 ... parameters • 8 ... internal states • 16 ... array parameters • 32 ... array states • 64 ... cyclic (trend) buffers • 256 ... metadata (STATESAVE only) ↓0 ↑65535 ⊙65535	Long (I32)
SaveOnExit	If set, the file is stored when the entire configuration is terminated. ⊙on	Bool

Outputs

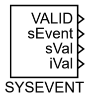
OutState	JSON string where values are stored (only if the filename parameter is empty)	String
DONE	Indicates whether the state has been saved	Bool
iE	Error code if block execution fails	Error

"

SYSEVENT – * Read system log

Block Symbol

Licence: STANDARD



Function Description

The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

Parameters

arc	Archive to read (0=system log)	↓0 ↑16	Long (I32)
filter	String that item must contain		String
idfrom	Minimum item ID to show	↓0 ↑65535	Long (I32)
idto	Maximum item ID to show	↓0 ↑65535 ⊙65655	Long (I32)
lvlfrom	Minimum item level to show	↓0 ↑255	Long (I32)
lvlto	Maximum item level to show	↓0 ↑255 ⊙255	Long (I32)

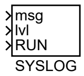
Outputs

VALID	Output data are valid (actual)	Bool
sEvent	Whole archive item in JSOM	String
sVal	Archive item value (string)	String
iVal	Archive item value (integer)	Long (I32)

SYSLOG – Write system log

Block Symbol

Licence: STANDARD



Function Description

The SYSLOG block is intended for writing any messages to the REXYGEN system log. It can be used for basic logging of user events. To write, it is necessary to have messages of the given level enabled in the System Logs Configuration (Target -> System Logs Configuration -> Function block messages).

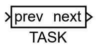
Inputs

`msg`	The message you want to save to the log (max. 512 znaků)	`String`
`lvl`	Level of logged message: `0` ..... Error `1` ..... Warning `2` ..... Info `3` ..... Verbose	`Long (I32)`
`RUN`	Writing enable. Writing to the log continues as long as the `RUN` input is `ON`	`Bool`

TASK – Standard task of the REXYGEN system

Block Symbol

Licence: STANDARD



Function Description

The overall control algorithm of the REXYGEN system consists of individual tasks. These are included by using the TASK block. There can be one or more tasks in the control algorithm. The REXYGEN system contains four main computational levels represented by the Level0 to Level3 outputs of the EXEC block. The individual tasks are added to the given computational level <i> by connecting the prev input with the corresponding Level<i> output or with the next output of a TASK, which is already included in the given level <i>. There can be only one task connected to the next output of the TASK block. The next output of the last task in the given level remains unconnected. This means that the tasks in one level create a unidirectional chain which defines the order of initialization and execution of the individual tasks of the given level in the REXYGEN system. The individual levels are ordered from Level0 to Level3 (the QTASK block precedes Level0).

All tasks at the given level <i> are executed with the same priority, which is determined by the pri<i> parameter of the EXEC block. The execution period of the task is calculated as a multiple of the factor parameter and the base tick of the ntick<i>*tick in the EXEC block.

The time allocated for task execution starts at the start tick and ends at the stop tick. The start and stop values can be fixed or left to be controlled by the RexCore. For RexCore control, the parameters can be filled in as follows:

- start = -1: The execution begins as soon as the previous Task ends.
- start = -2: The execution starts on the next tick after the completion of the previous task.
- stop = -1: The task execution must finish before the end of ntick<i>*tick.
- stop = -2: The task execution must finish in the next tick.

For fixed execution times, start and stop should be a non-negative integer.

The REXYGEN Compiler compiler additionally verifies that the stop parameter of the preceding task is less than or equal to the stop parameter of the succeeding task. This ensures that the allocated time intervals for individual tasks do not overlap. If the timing of individual levels is inappropriate, tasks may be interrupted by tasks and other events with higher priority. In such cases, execution is not aborted but delayed (in contrast to the QTASK block). The Diagnostics section of the REXYGEN Studio program assesses whether the execution delay is occasional or permanent (the Level and Task tabs).

Input

`prev`	Input for connecting the task with the corresponding `Level<i>` output of the EXEC block or with the `next` output of the preceding task of the given level	`Long (I32)`

Output

`next`	Output for connecting to the `prev` input of the succeeding task in the given level	`Long (I32)`

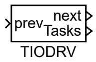
Parameters

`factor`	Execution factor; multiple of the execution period of the `i`-th level of the EXEC block defining the execution period of the task: `factor` ∗ `tick` ∗ `ntick<i>` ⊙1	`Long (I32)`
`start`	Number of tick of the given computational level which should trigger the task execution ↓0 ↑`ntick<i>` ⊙0	`Long (I32)`
`stop`	Number of tick of the given computational level by which the task execution should finish ↓`start+1` ↑`ntick<i>` ⊙1	`Long (I32)`
`stack`	Stack size [bytes] ⊙10240	`Long (I32)`
`filename`	Name of the file with the `.mdl` extension which contains the task algorithm. In the case `filename` is not specified, the filename is given by the name of the `TASK` block in the project main file (the `.mdl` extension is attached automatically)	`String`

TIODRV – The REXYGEN system input/output driver with tasks

Block Symbol

Licence: STANDARD



Function Description

The TIODRV block is used for configuration of special drivers of the REXYGEN system which are able to execute tasks defined by the IOTASK blocks. See the corresponding driver documentation.

The prev input of the IOTASK block must be connected with the Tasks output of the TIODRV block. If the driver allows so, the next output of a TIODRV block which is already included in the configuration can be used to add more tasks. The next output of the last task remains unconnected. On the contrary to standard tasks, the number and order of the driver's tasks are not checked by the REXYGEN Compiler compiler but by the input-output driver itself.

If the driver cannot guarantee periodic execution of some task (e.g. task is triggered by an external event), a corresponding flag is set for the given task. Such a task cannot contain blocks which require constant sampling period (e.g. the majority of controllers). If some of these restricted blocks are used, the executive issues a task execution error, which can be traced using the Diagnostics section of the REXYGEN Studio program. The cpu parameter can be used to specify where the driver thread should run on multi-CPU devices.

Input

prev	Input for connecting the driver with the Drivers output of the EXEC block or with the next output of the preceding driver		Long (I32)

Outputs

next	Output for connecting to the prev input of the succeeding driver		Long (I32)
Tasks	The IOTASK blocks executed by the driver are connected to this output using the prev input		Long (I32)

Parameters

module	Name of the module, which includes the input/output driver (mandatory only if module name differs from classname)		String
classname	Name of the driver class; case sensitive!	⊙DrvClass	String

`cfgname`	Name of the driver configuration file ⊙`iodrv.rio`	`String`
`factor`	Multiple of the `EXEC` block's `tick` parameter defining the driver's task execution period ↓1 ⊙10	`Long (I32)`
`stack`	Stack size of the driver's task in bytes ↓1024 ⊙10240	`Long (I32)`
`pri`	Logical priority of the driver's task ↓1 ↑31 ⊙3	`Long (I32)`
`cpu`	CPU core assigned to driver thread (-1=default, 0=core 0, 1=core 1, ...) ↓-1 ↑127 ⊙-1	`Long (I32)`

`WWW` – Internal Web Server Content

Block Symbol

Licence: STANDARD

Function Description

The `WWW` block is a so-called "pseudo-block" which stores additional information about a contents of an internal web server. The only file where the block can be placed is a main project file with a single `EXEC` block an so it belongs to the `EXEC` category.

The block does not have any inputs or outputs. The block itself does not become a part of a final binary configuration but the data it points to does. Be careful when inserting big files or directories as the integrated web server is not optimized for a large data. It is possible to shrink the data by enabling gzip compression. The compression also reduces amount of data transferred to the client, but decompression must be performed on the server side when a client does not support gzip compression which brings additional load on the target device.

Parameters

`Source`	Specifies a source directory or a file name that should be placed on the target and should be available via integrated web server using standard HTTP and/or HTTPS protocol. The path may be absolute or relative to path of a main project file.	`String`
`Target`	Specifies a target directory or a file name on the integrated web server.	`String`
`Compression`	Enables data compression in gzip format.	`Bool`

Chapter 3

INOUT – Input and output blocks

Contents

Display – Numeric display of input values

Block Symbol

Licence: STANDARD

Function Description

The DISPLAY block shows input value in a selected format. A suffix may be appended to the value. An actual value is shown immediately in REXYGEN Studio even without turning on *Watch* mode for the block, and the same in *WebWatch*. Actual conversion of input into its textual representation is performed on the target device in each Decimation period so the value displayed may be also read via the *REST* interface or used in visualization.

Input

u	Input signal	Any

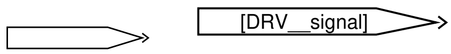
Parameters

`Format`	Format of displayed value	⊙1	`Long (I32)`
	`Best fit` same as long, but for extremly small or big numbers same as long_e; this is default format for real numbers `short` fixed point, no more than 3 places after the decimal point; default format is used for not-real numbers `long` .. fixed point, full precision (up to 16 digits); default format for is used not-real numbers `short_e` exponencial (scientific) format, no more than 3 places after the decimal point; default format is used for not-real numbers `long_e` , exponencial (scientific) format, full precision (up to 16 digits); default format is used for not-real numbers `bank` .. fixed point, 2 places after the decimal point; default format is used for not-real numbers `dec` ... integer number in decimal format (standard number); this is default format for integer numbers `hex` ... integer number in hexadecimal format (used by programmers); default format is used for not-integer numbers `bin` ... integer number in binary format (used by programmers); default format is used for not-integer numbers `oct` ... integer number in octal format (used by programmers); default format is used for not-integer numbers		
`Decimation`	Value is evaluated in each `Decimation` period	↓1 ↑100000 ⊙1	`Long (I32)`
`Suffix`	A string to be appended to the value		`String`

From, INSTD – Signal connection or input

Block Symbols

Licence: STANDARD



Function Description

The two blocks `From` (signal connection) and `INSTD` (standard input) share the same symbol. They are used for referring to another signal, either internal or external.

In the function block library, you can only find the `From` block. It is converted to the `INSTD` block at compile time if necessary. The following rules define how the REXYGEN Compiler compiler distinguishes between the two block types:

- If the parameter `GotoTag` contains the `__` delimiter (two successive `'_'` characters), then the block is of the `INSTD` type. The part (substring) of the parameter before the delimiter (`DRV` in the block symbol above) is considered to be the name of an `IODRV` type block contained in the main file of the project. The REXYGEN Compiler compiler returns an error when such block does not exist. If the driver exists in the project, the other part of the `GotoTag` parameter (following the delimiter, `signal` in this case) is considered to be the name of a signal within the corresponding driver. This name is validated by the driver and in the case of success, an instance of the `INSTD` block is created. This instance collects real-time data from the driver and feeds the data into the control algorithm at each execution of the task it is included in.

- If there is no `__` delimiter in the `GotoTag` parameter, the block is of type `From`. A matching `Goto` block with the same `GotoTag` parameter and required visibility given by the `TagVisibility` parameter (see the `Goto` block description) is searched. In case it is not found, the REXYGEN Compiler compiler issues a warning and deletes the `From` block. Otherwise an "invisible" connection is created between the corresponding blocks. The `From` block is removed also in this case and thus it is not contained in the resulting control system configuration.

In the case of `INSTD` block, the `GotoTag` parameter includes the symbol of the driver `<DRV>` and the name of the signal `<signal>` of the given driver:

```
<DRV>__<signal>
```

E.g. the first digital input of a Modbus I/O device might be referenced by `MBM__DI1`. Detailed information about signal naming can be found in the user manual of the corresponding I/O driver.

Since version 2.50.5 it is possible to use placeholders in names of I/O driver signals. This is useful inside subsystems where this placeholder is replaced by the value of subsystem parameter. E.g. the flag `MBM__DI<id>` will refer to digital input 1, 2, 3 etc. depending on the parameter `id` of the subsystem the block is contained in. See the `SubSystem` function block for information on defining subsystem parameters.

Output

`value`	Signal coming from I/O driver or `Goto` block. The type of output is determined by the type of the signal which is being referred by the `GotoTag` parameter.	`Any`

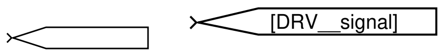
Parameter

`GotoTag`	Reference to a `Goto` block with the same `GotoTag` parameter, which should be connected with the `From` block or a reference to input signal of the REXYGEN I/O driver, which should provide data through the block's output.	`String`

Goto, OUTSTD – Signal source or output

Block Symbols

Licence: STANDARD



Function Description

The two blocks Goto (signal source) and OUTSTD (standard output) share the same symbol. They are used for providing signals, either internal or external.

In the function block library, you can only find the Goto block. It is converted to the OUTSTD block at compile time if necessary. The following rules define how the REXYGEN Compiler compiler distinguishes between the two block types:

- If the parameter GotoTag contains the __ delimiter (two successive '_' characters), then the block is of the OUTSTD type. The part (substring) of the parameter before the delimiter (DRV in the block symbol above) is considered to be the name of an IODRV type block contained in the main file of the project. The REXYGEN Compiler compiler returns an error when such block does not exist. If the driver exists in the project, the other part of the GotoTag parameter (following the delimiter, signal in this case) is considered to be the name of a signal within the appropriate driver. This name is validated by the driver and in the case of success, an instance of the OUTSTD block is created. This instance collects real-time data from the driver and feeds the data into the control algorithm at each execution of the task it is included in.

- If there is no __ delimiter in the GotoTag parameter, the block is of type Goto. A matching From block with the same GotoTag parameter for which the Goto block is visible is searched. In case it is not found, the REXYGEN Compiler compiler issues a warning and deletes the Goto block. Otherwise an "invisible" connection is created between the corresponding blocks. The Goto block is removed also in this case thus it is not contained in the resulting control system configuration.

The other parameter of the Goto block defines the visibility of the block within the given .mdl file. The TagVisibility parameter can be local, global or scoped, whose meaning is explained in the table below. This parameter is ignored if the block is compiled as the OUTSTD block.

In the case of OUTSTD block, the GotoTag parameter includes the symbol of the driver <DRV> and the name of the signal <signal> of the given driver:

```
<DRV>__<signal>
```

E.g. the first digital output of a Modbus I/O device might be referenced by MBM__DO1. Detailed information about signal naming can be found in the user manual of the corresponding I/O driver.

Since version 2.50.5 it is possible to use placeholders in names of I/O driver signals. This is useful inside subsystems where this placeholder is replaced by the value of subsystem parameter. E.g. the flag MBM__DO<id> will refer to digital output 1, 2, 3 etc. depending on the parameter id of the subsystem the block is contained in. See the SubSystem function block for information on defining subsystem parameters.

Input

value	Signal going to I/O driver or From block. In case of connection to an I/O driver, the type of input is determined by the I/O driver from the GotoTag parameter.	Any

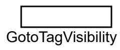
Parameters

GotoTag	Reference to a From block with the same GotoTag parameter, which should be connected with the Goto block or a reference to output signal of the REXYGEN control system driver, which should send the data from block input to the process.	String
TagVisibility	Visibility (availability) of the block within the .mdl file. Defines conditions under which the two corresponding Goto and From blocks are reciprocally available: ⊙local local — the two blocks must be in the same subsystem global — blocks can be anywhere in the given task (.mdl file) scoped — the From block must be placed in the same subsystem or in any lower hierarchical level below the GotoTagVisibility block with the same GotoTag parameter	String

GotoTagVisibility – Visibility of the signal source

Block Symbol

Licence: STANDARD



Function Description

The GotoTagVisibility blocks specify the visibility of the Goto blocks with scoped visibility. The symbol (tag) defined in the Goto block by the GotoTag parameter is available for all From blocks in the subsystem which contains the appropriate GotoTagVisibility block and also in all subsystems below in the hierarchy.

The GotoTagVisibility block is required only for Goto blocks whose TagVisibility parameter is set to scoped. There is no need for the GotoTagVisibility block for local or global visibility.

The GotoTagVisibility block is used only during project compilation by the REXYGEN Compiler compiler. It is not included in the binary configuration file for real-time execution.

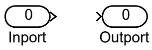
Parameter

GotoTag	Reference to a Goto block with the GotoTag parameter, whose visibility is defined by the position of this block (GotoTagVisibility)	String

Inport, Outport – Input and output port

Block Symbols

Licence: STANDARD



Function Description

The Inport and Outport blocks are used for connecting signals over individual hierarchical levels. There are two possible ways to use these blocks in the REXYGEN system:

1. To connect inputs and outputs of the subsystem. The blocks create an interface between the symbol of the subsystem and its inner algorithm (sequence of blocks contained in the subsystem). The Inport or Outport blocks are located inside the subsystem, the name of the given port is displayed in the subsystem symbol in the upper hierarchy level.

2. To provide connection between various tasks. The port blocks are located in the highest hierarchy level of the given task (.mdl file) in this case. The corresponding Inport and Outport blocks should have the same Block name. The connection between blocks in various tasks is checked and created by the REXYGEN Compiler compiler.

The ordering of the blocks to be connected is based on the Port parameter of the given block. The numberings of the input and output ports are independent on each other. The numbering is automatic in REXYGEN Studioand it starts at 1. The numbers of ports must be unique in the given hierarchy level, in case of manual modification of the port number the other ports are re-numbered automatically. Be aware that after re-numbering in an already connected subsystem the inputs (or outputs) in the upper hierarchy level are re-ordered, which results in probably unintended change in signal mapping!

In the blocks ‘Inport‘ and ‘Outport‘, it is also possible to explicitly specify the data type of the transmitted value using the ‘OutDataTypeStr‘ parameter. If no value is selected (the option ‘Inherit: auto‘ is chosen), the value type is determined automatically.

The ‘Description‘ parameter allows you to add a textual description of the block. This description is displayed in the properties of the subsystem and library block if the ‘Inport‘ or ‘Outport‘ is used to define the inputs and outputs of the subsystem.

Warning: The blocks Inport and Outport should not be use to connect arrays and other references between tasks (references often have **ref** in name and have a type **intptr** in the **Diagnostics** section of the REXYGEN Studio program). Consistence is not guaranteed in this case; incorrect value could be get and runtime code can crash in worst case scenario. Typical behaviour is that some array members are from one period of execution and other members of array from next period. The blocks SETPA and

GETPA ensure consistent read and write of the array between task. Some blocks guarantee consistence of references over task boundary (for example RM_AxisSpline). In this case, this is explicitly stated in the block manual.

Input

value	Value going to the output pin or Inport	Any

Output

value	Value coming from the input pin or Outport	Any

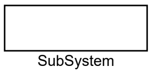
Parameters

Port	Ordering of the Inport or Outport pins	Long (I32)
OutDataTypeStr	Data type of item Inherit: auto double single uint8 int16 uint16 int32 uint32 boolean float int64 string array	String
Description	Description of the port	String

SubSystem – Subsystem block

Block Symbol

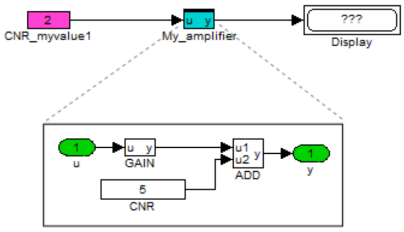
Licence: STANDARD



Function Description

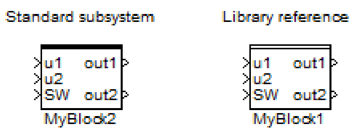
The SubSystem block is a cornerstone of hierarchical organization of block diagrams in REXYGEN. A subsystem is a container for a group of function blocks and their connections, which then appear as a single block. Nesting of subsystems is allowed, i.e. a subsystem can include additional subsystems.

The runtime core or REXYGEN executes the subsystem as an ordered sequence of blocks. Therefore the subsystem is sometimes referred to as sequence. All blocks from the surroundings of the subsystem are executed strictly before or strictly after the whole subsystem is executed.



Subsystems are also used for creating user-defined reusable components, which are then placed in user libraries.

A library reference can be distinguished from a standard subsystem by the style of the upper border.



Please refer to [2] for details on using subsystems and creating reusable components in REXYGEN.

Also see examples 0101-02 and 0101-03 demonstrating the use of subsystems. The examples are included in REXYGEN Studio.

Inputs

The ordering and names of the inputs are given by the numbers and names of the Inport blocks contained within the subsystem. See REXYGEN Studio manual [2] for details.

Outputs

The ordering and names of the outputs are given by the numbers and names of the `Outport` blocks contained within the subsystem. See REXYGEN Studio manual [2] for details.

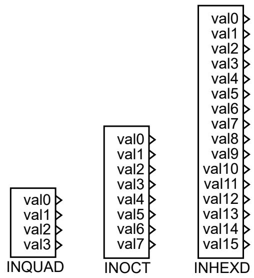
Parameters

The parameters of the subsystem are defined by the so-called subsystem mask. See REXYGEN Studio manual [2] for details.

INQUAD, INOCT, INHEXD – Multi-input blocks

Block Symbols

Licence: STANDARD



Function Description

The REXYGEN system allows not only reading of a single input signal but also simultaneous reading of multiple signals through just one block (for example all signals from one module or plug-in board). The blocks `INQUAD`, `INOCT` and `INHEXD` are designed for these purposes. They differ only in the maximum number of signals (4, 8 and 16, respectively).

The name of the block instance includes the symbol of the driver `<DRV>` and the name of the signal `<signal>` of the given driver:

```
<DRV>__<signal>
```

It is created the same way as the `GotoTag` parameter of the `INSTD` and `OUTSTD` blocks. E.g. the digital inputs of a Modbus I/O device might be referenced by `MBM__DI`. Detailed information about signal naming can be found in the user manual of the corresponding I/O driver.

The overhead necessary for data acquisition through input/output drivers is minimized when using these blocks, which is important mainly for very fast control algorithms with sampling period of 1 ms and lower. Moreover, all the inputs are read simultaneously or as successively as possible. Detailed information about using these blocks for particular driver can be found in the user manual for the given driver.

Since version 2.50.5 it is possible to use placeholders in names of I/O driver signals. This is useful inside subsystems where this placeholder is replaced by the value of subsystem parameter. E.g. the name `MBM__module<id>` will refer to module 1, 2, 3 etc. depending on the parameter `id` of the subsystem the block is contained in. See the `SubSystem` function block for information on defining subsystem parameters.

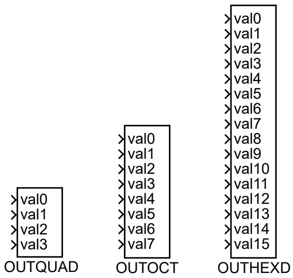
Outputs

`val`*i*	Input signals fed into the control algorithm through input/output drivers. The type and location of individual signals is described in the user manual for the given driver.	`Any`

OUTQUAD, OUTOCT, OUTHEXD – Multi-output blocks

Block Symbols

Licence: STANDARD



Function Description

The REXYGEN system allows not only writing of a single output signal but also simultaneous writing of multiple signals through just one block (for example all signals of one module or plug-in board). The blocks OUTQUAD, OUTOCT and OUTHEXD are designed for these purposes. They differ only in the maximum number of signals (4, 8 and 16, respectively). These blocks are not included in the RexLib function block library for Matlab-Simulink.

The name of the block instance includes the symbol of the driver <DRV> and the name of the signal <signal> of the given driver:

```
<DRV>__<signal>
```

It is created the same way as the GotoTag parameter of the INSTD and OUTSTD blocks. E.g. the digital outputs of a Modbus I/O device might be referenced by MBM__DO. Detailed information about signal naming can be found in the user manual of the corresponding I/O driver.

The overhead necessary for setting the outputs through input/output drivers is minimized when using these blocks, which is important mainly for very fast control algorithms with sampling period of 1 ms and lower. Moreover, all the inputs are written simultaneously or as successively as possible. Detailed information about using these blocks for particular driver can be found in the user manual for the given driver.

Since version 2.50.5 it is possible to use placeholders in names of I/O driver signals. This is useful inside subsystems where this placeholder is replaced by the value of subsystem parameter. E.g. the name MBM__module<id> will refer to signals of module 1, 2, 3 etc. depending on the parameter id of the subsystem the block is contained in. See the SubSystem function block for information on defining subsystem parameters.

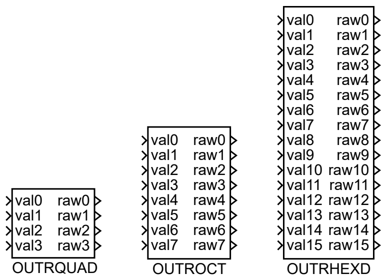
Inputs

`val`i	Signals to be sent to the process via the input/output driver. The type and location of individual signals is described in the user manual for the given driver.	`Any`

OUTRQUAD, OUTROCT, OUTRHEXD – Multi-output blocks with verification

Block Symbols

Licence: ADVANCED



Function Description

The **OUTRQUAD**, **OUTROCT** and **OUTRHEXD** blocks allow simultaneous writing of multiple signals, they are similar to the OUTQUAD, OUTOCT and OUTHEXD blocks. Additionally they provide feedback information about the result of write operation for the given output.

There are two ways to inform the control algorithm about the result of write operation through the **raw***i* output:

- Through the value of the output, which can e.g. contain the real bit value in case of exceeding the limits of D/A converter (thus the **raw** notation).
- Through reading the quality flags of the signal. This information can be separated from the signal by the VIN and QFD blocks.

The **raw***i* outputs are not always refreshed right at the moment of block execution, there is some delay given by the properties of the driver, communication line and/or target platform.

Inputs

val*i*	Output signals defined by the control algorithm through the input/output driver. The type and location of individual signals is described in the user manual for the given driver.	Any

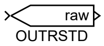
Outputs

raw*i*	Feedback information about the write operation result. The type and meaning of individual signals is described in the user manual for the given driver.	Any

OUTRSTD – Output block with verification

Block Symbol

Licence: ADVANCED



Function Description

The OUTRSTD block is similar to the OUTSTD block. Additionally it provides feedback information about the result of write operation for the output signal.

There are two ways to inform the control algorithm about the result of write operation through the raw output:

- Through the value of the output, which can e.g. contain the real bit value in case of exceeding the limits of D/A converter (thus the raw notation).
- Through reading the quality flags of the signal. This information can be separated from the signal by the VIN and QFD blocks.

The raw outputs is not refreshed right at the moment of block execution, there is some delay given by the properties of the driver, communication line and/or target platform.

Input

value	Output signal defined by the control algorithm through the input/output driver. The type and naming of the signal is described in the user manual for the given driver.	Any

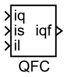
Output

raw	Feedback information about the write operation result. The type and meaning of the signal is described in the user manual for the given driver.	Any

QFC – Quality flags coding

Block Symbol

Licence: STANDARD



Function Description

The QFC block creates the resulting signal iqf representing the quality flags by combining three components iq, is and il. The quality flags are part of each input or output signal in the REXYGEN system. Further details about quality flags can be found in chapter 1.4 of this manual. The RexLib function block library for Matlab-Simulink does not use any quality flags.

It is possible to use the QFC block together with the VOUT block to force arbitrary quality flags for a given signal. Reversed function to the QFC block is performed by the QFD block.

Inputs

iq	Basic quality type flags, see table 1.2, page 19	Long (I32)
is	Substatus flags, see [1]	Long (I32)
il	Limits flags, see [1]	Long (I32)

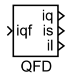
Output

iqf	Bit combination of the iq, is and il input signals	Long (I32)

QFD – Quality flags decoding

Block Symbol

Licence: STANDARD



Function Description

The QFD decomposes quality flags to individual components iq, is and il. The quality flags are part of each input or output signal in the REXYGEN system. Further details about quality flags can be found in chapter 1.4 of this manual. The RexLib function block library for Matlab-Simulink does not use any quality flags.

It is possible to use the QFD block together with the VIN block for detailed processing of quality flags of a given signal. Reversed function to the QFD block is performed by the QFC block.

Input

iqf	Quality flags to be decomposed to iq, is and il components	Long (I32)

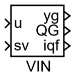
Outputs

iq	Basic quality type flags, see table 1.2, page 19	Long (I32)
is	Substatus flags, see [1]	Long (I32)
il	Limits flags, see [1]	Long (I32)

VIN – Validation of the input signal

Block Symbol

Licence: STANDARD



Function Description

The VIN block can be used for verification of the input signal quality in the REXYGEN system. Further details about quality flags can be found in chapter 1.4 of this manual.

The block continuously separates the quality flags from the input u and feeds them to the iqf output. Based on these quality flags and the GU parameter (Good if Uncertain), the input signals are processed in the following manner:

- For GU = off the output QG is set to on if the quality is GOOD. It is set to QG = off in case of BAD or UNCERTAIN quality.
- For GU = on the output QG is set to onif the quality is GOOD or UNCERTAIN. It is set to QG = off only in case of BAD quality.

The output yg is equal to the u input if QG = on. Otherwise it is set to yg = sv (substitution variable).

Inputs

u	Input signal whose quality is assessed. The type of the signal is determined upon the connected signal.	Any
sv	Substitute value for an error case	Any

Outputs

yg	Validated output signal (yg = u for QG = on or yg = sv for QG = off)	Any
QG	Indicator of input signal acceptability	Bool
iqf	Complete quality flag separated from the u input signal	Long (I32)

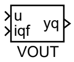
Parameter

GU	Acceptability of UNCERTAIN quality off … Uncertain quality unacceptable on …. Uncertain quality acceptable	Bool

VOUT – Validation of the output signal

Block Symbol

Licence: STANDARD



Function Description

It is possible to use the VOUT block to force arbitrary quality flags for a given signal. The desired quality flags are given by the input signal **iqf**. Further details about quality flags can be found in chapter 1.4 of this manual.

Inputs

u	Input signal whose quality flags are being replaced. The type of the signal is determined upon the connected signal.	Any
iqf	Desired quality flags	Long (I32)

Output

yq	Resulting signal composed from input **u** and quality flags given by the **iqf** input	Any

Chapter 4

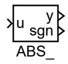
MATH – Math blocks

Contents

ABS – Absolute value

Block Symbol

Licence: STANDARD



Function Description

The ABS block computes the absolute value of the analog input signal u. The output y is equal to the absolute value of the input and the sgn output denotes the sign of the input signal.

$$\mathtt{sgn} = \begin{cases} -1, & \text{for } \mathtt{u} < 0, \\ 0, & \text{for } \mathtt{u} = 0, \\ 1, & \text{for } \mathtt{u} > 0. \end{cases}$$

Input

u	Analog input of the block	Double (F64)

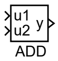
Outputs

y	Absolute value of the input signal	Double (F64)
sgn	Indication of the input signal sign	Long (I32)

ADD – Addition of two signals

Block Symbol

Licence: STANDARD



Function Description

The ADD blocks sums two analog input signals. The output is given by

$$y = u1 + u2.$$

Consider using the ADDOCT block for addition or subtraction of multiple signals.

Inputs

u1	First analog input of the block	Double (F64)
u2	Second analog input of the block	Double (F64)

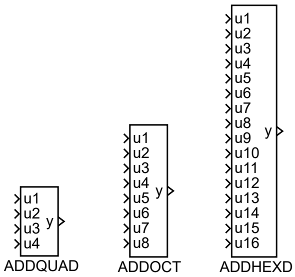
Output

y	Sum of the input signals	Double (F64)

ADDQUAD, ADDOCT, ADDHEXD – Multi-input addition

Block Symbols

Licence: STANDARD



Function Description

The ADDQUAD, ADDOCT and ADDHEXD blocks sum (or subtract) up to 16 input signals. The nl parameter defines the inputs which are subtracted instead of adding. For an empty nl parameter the block output is given by $y = u1 + u2 + u3 + u4 + u5 + u6 + u7 + \ldots + u16$. For e.g. nl=2,5,7, the block implements the function $y = u1 - u2 + u3 + u4 - u5 + u6 - u7 + \ldots + u16$.

Note that the ADD and SUB blocks are available for simple addition and subtraction operations.

Inputs

u1..u16	Analog input signals	Double (F64)

Output

y	Resulting value	Double (F64)

Parameter

nl	List of signals to subtract instead of adding. The format of the list is e.g. 1,3..5,8. Third-party programs (Simulink, OPC clients etc.) work with an integer number, which is a binary mask, i.e. 157 (binary 10011101) in the mentioned case.	Long (I32)

CNB – Boolean (logic) constant

Block Symbol

Licence: STANDARD

Function Description

The CNB block stands for a Boolean (logic) constant.

Output

Y	Logical output of the block		Bool

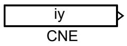
Parameter

YCN	Boolean constant off . . . Disabled on . . . . Enabled	⊙on	Bool

CNE – Enumeration constant

Block Symbol

Licence: STANDARD



Function Description

The CNE block allows selection of a constant from a predefined popup list. The popup list of constants is defined by the pupstr string, whose syntax is obvious from the default value shown below. The output value corresponds to the number at the beginning of the selected item. In case the pupstr string format is invalid, the output is set to 0.

There is a library called CNEs in Simulink, which contains CNE blocks with the most common lists of constants.

Parameters

yenum	Enumeration constant	⊙1: option A	String
pupstr	Popup list definition ⊙1: option A|2: option B|3: option C		String

Output

iy	Integer output of the block	Long (I32)

CNI – Integer constant

Block Symbol

Licence: STANDARD

Function Description

The CNI block stands for an integer constant.

Output

iy	Integer output of the block	Long (I32)

Parameter

icn	Integer constant	⊙1	Long (I32)
vtype	Output numeric type -- .... 2 ..... Byte (U8) 3 ..... Short (I16) 4 ..... Long (I32) 5 ..... Word (U16) 6 ..... DWord (U32) -- .... -- .... -- .... 10 .... Large (I64)	⊙4	Long (I32)

CNR – **Real constant**

Block Symbol

Licence: STANDARD

Function Description

The CNR block stands for a real constant.

Output

y	Analog output of the block		Double (F64)

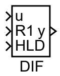
Parameter

ycn	Real constant	⊙1.0	Double (F64)

DIF – **Difference**

Block Symbol

Licence: STANDARD



Function Description

The DIF block differentiates the input signal u according to the following formula

$$y_k = u_k - u_{k-1},$$

where $u_k = u$, $y_k = y$ and u_{k-1} is the value of input u in the previous cycle (delay T_S, which is the execution period of the block).

Input

u	Analog input of the block	Double (F64)
R1	Block reset (same state as after init)	Bool
HLD	Hold block execution	Bool

Output

y	Difference of the input signal	Double (F64)

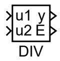
Parameters

ISSF	Zero output at start-up off ... In the first cycle the output will be $y = u$. on .... Zero output in the first cycle, $y = 0$.	Bool

DIV – Division of two signals

Block Symbol

Licence: STANDARD



Function Description

The DIV block divides two analog input signals $y = u1/u2$. In case $u2 = 0$, the output E is set to onand the output y is substituted by $y = yerr$.

Inputs

u1	First analog input of the block	Double (F64)
u2	Second analog input of the block	Double (F64)

Outputs

y	Quotient of the inputs	Double (F64)
E	Error flag – division by zero	Bool

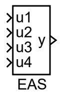
Parameter

yerr	Substitute value for an error case	⊙1.0	Double (F64)

EAS – Extended addition and subtraction

Block Symbol

Licence: STANDARD



Function Description

The EAS block sums input analog signals u1, u2, u3 and u4 with corresponding weights a, b, c and d. The output y is then given by

$$y = a * u1 + b * u2 + c * u3 + d * u4 + y0.$$

Inputs

u1	First analog input of the block	Double (F64)
u2	Second analog input of the block	Double (F64)
u3	Third analog input of the block	Double (F64)
u4	Fourth analog input of the block	Double (F64)

Output

y	Analog output of the block	Double (F64)

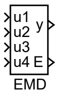
Parameters

a	Weighting coefficient of the u1 input	⊙1.0	Double (F64)
b	Weighting coefficient of the u2 input	⊙1.0	Double (F64)
c	Weighting coefficient of the u3 input	⊙1.0	Double (F64)
d	Weighting coefficient of the u4 input	⊙1.0	Double (F64)
y0	Additive constant (bias)		Double (F64)

EMD – Extended multiplication and division

Block Symbol

Licence: STANDARD



Function Description

The EMD block multiplies and divides analog input signals u1, u2, u3 and u4 with corresponding weights a, b, c and d. The output y is then given by

$$y = \frac{(a * u1 + a0)(b * u2 + b0)}{(c * u3 + c0)(d * u4 + d0)}. \tag{4.1}$$

The output E is set to on in the case that the denominator in the equation (4.1) is equal to 0 and the output y is substituted by y = yerr.

Inputs

u1	First analog input of the block	Double (F64)
u2	Second analog input of the block	Double (F64)
u3	Third analog input of the block	Double (F64)
u4	Fourth analog input of the block	Double (F64)

Outputs

y	Analog output of the block	Double (F64)
E	Error flag – division by zero	Bool

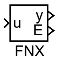
Parameters

a	Weighting coefficient of the u1 input	$\odot$1.0	Double (F64)
a0	Additive constant for u1 input		Double (F64)
b	Weighting coefficient of the u2 input	$\odot$1.0	Double (F64)
b0	Additive constant for u2 input		Double (F64)
c	Weighting coefficient of the u3 input	$\odot$1.0	Double (F64)
c0	Additive constant for u3 input		Double (F64)
d	Weighting coefficient of the u4 input	$\odot$1.0	Double (F64)
d0	Additive constant for u4 input		Double (F64)
yerr	Substitute value for an error case	$\odot$1.0	Double (F64)

FNX – Evaluation of single-variable function

Block Symbol

Licence: STANDARD



Function Description

The FNX block evaluates basic math functions of single variable. The table below shows the list of supported functions with corresponding constraints. The ifn parameter determines the active function.
List of functions:

ifn: shortcut	function	constraints on u
1: acos	arccosine	$u \in< -1.0, 1.0 >$
2: asin	arcsine	$u \in< -1.0, 1.0 >$
3: atan	arctangent	–
4: ceil	rounding towards the nearest higher integer	–
5: cos	cosine	–
6: cosh	hyperbolic cosine	–
7: exp	exponential function e^{u}	–
8: exp10	exponential function 10^{u}	–
9: fabs	absolute value	–
10: floor	rounding towards the nearest lower integer	–
11: log	logarithm	$u > 0$
12: log10	decimal logarithm	$u > 0$
13: random	arbitrary number $z \in< 0, 1 >$ (u independent)	–
14: sin	sine	–
15: sinh	hyperbolic sine	–
16: sqr	square function	–
17: sqrt	square root	$u > 0$
18: srand	changes the seed for the random function to u	$u \in \mathbb{N}$
19: tan	tangent	–
20: tanh	hyperbolic tangent	–

Note: All trigonometric functions process data in radians.

The error output is activated ($E = on$) in the case when the input value u falls out of its bounds or an error occurs during evaluation of the selected function (implementation dependent), e.g. square root of negative number. The output is set to substitute value in such case ($y = yerr$).

Input

u	Analog input of the block		Double (F64)

Outputs

y	Result of the selected function		Double (F64)
E	Error flag		Bool

Parameters

ifn	Function type (see table above)	⊙1	Long (I32)
yerr	Substitute value for an error case		Double (F64)

FNXY – Evaluation of two-variables function

Block Symbol

Licence: STANDARD

Function Description

The FNXY block evaluates basic math functions of two variables. The table below shows the list of supported functions with corresponding constraints. The ifn parameter determines the active function.
List of functions:

ifn: shortcut	function	constraints on u1, u2
1: atan2	arctangent u1/u2	–
2: fmod	remainder after division u1/u2	u2 $\neq$ 0.0
3: pow	exponentiation of the inputs $\mathtt{y} = \mathtt{u1}^{\mathtt{u2}}$	–

The atan2 function result belongs to the interval $\langle -\pi, \pi \rangle$. The signs of both inputs u1 a u2 are used to determine the appropriate quadrant.

The fmod function computes the remainder after division u1/u2 such that $\mathtt{u1} = i \cdot \mathtt{u2} + \mathtt{y}$, where i is an integer, the signs of the y output and the u1 input are the same and the following holds for the absolute value of the y output: $|\mathtt{y}| < |\mathtt{u2}|$.

The error output is activated (E = on) in the case when the input value u2 does not meet the constraints or an error occurs during evaluation of the selected function (implementation dependent), e.g. division by zero. The output is set to substitute value in such case (y = yerr).

Inputs

u1	First analog input of the block	Double (F64)
u2	Second analog input of the block	Double (F64)

Outputs

y	Result of the selected function	Double (F64)
E	Error flag off . . . No error on . . . . An error occurred	Bool

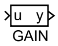
Parameters

`ifn`	Function type (see the table above) `1` ..... atan2 `2` ..... fmod `3` ..... pow	⊙1	`Long (I32)`
`yerr`	Substitute value for an error case		`Double (F64)`

`GAIN` – Multiplication by a constant

Block Symbol

Licence: STANDARD



Function Description

The `GAIN` block multiplies the analog input `u` by a real constant `k`. The output is then

$$\mathtt{y} = \mathtt{ku}.$$

Input

`u`	Analog input of the block		`Double (F64)`

Output

`y`	Analog output of the block		`Double (F64)`

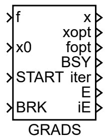
Parameter

`k`	Gain	⊙`1.0`	`Double (F64)`

GRADS – Gradient search optimization

Block Symbol

Licence: ADVANCED



Function Description

The GRADS block performs one-dimensional optimization of the $f(\mathtt{x}, v)$ function by gradient method, where $\mathtt{x} \in \langle \mathtt{xmin}, \mathtt{xmax} \rangle$ is the optimized variable and v is an arbitrary vector variable. It is assumed that the value of the function $f(\mathtt{x}, v)$ for given x at time k is enumerated and fed to the f input at time $k + \mathtt{n} * T_S$, where T_S is the execution period of the GRADS block. This means that the individual optimization iterations have a period of $\mathtt{n} * T_S$. The length of step of the gradient method is given by

$$
\begin{aligned}
grad &= (\mathtt{f}_i - \mathtt{f}_{i-1}) * (dx)_{i-1} \\
(dx)_i &= -\mathtt{gamma} * grad,
\end{aligned}
$$

where i stands for i-th iteration. The step size is restricted to lie within the interval $\langle \mathtt{dmin}, \mathtt{dmax} \rangle$. The value of the optimized variable for the next iteration is given by

$$x_{i+1} = x_i + (dx)_i$$

Inputs

f	Value of the optimized $f(.)$ for given variable x	Double (F64)
x0	Optimization starting point	Double (F64)
START	Starting signal (rising edge)	Bool
BRK	Termination signal	Bool

Outputs

x	Current value of the optimized variable	Double (F64)
xopt	Resulting optimal value of the x variable	Double (F64)
fopt	Resulting optimal value of the function $f(\mathtt{x}, v)$	Double (F64)
BSY	Indicator of running optimization	Bool
iter	Number of current iteration	Long (I32)
E	Error flag	Bool

iE	Error code 1 ..... $\mathtt{x} \notin <\mathtt{xmin}, \mathtt{xmax}>$ 2 ..... $\mathtt{x} = \mathtt{xmin}$ or $\mathtt{x} = \mathtt{xmax}$		Long (I32)

Parameters

xmin	Lower limit for the x variable		Double (F64)
xmax	Upper limit for the x variable	⊙10.0	Double (F64)
gamma	Coefficient for determining the step size of the gradient optimization method	⊙0.3	Double (F64)
d0	Initial step size	⊙0.05	Double (F64)
dmin	Minimum step size	⊙0.01	Double (F64)
dmax	Maximum step size	⊙1.0	Double (F64)
n	Iteration period (in sampling periods T_S)	⊙100	Long (I32)
itermax	Maximum number of iterations	⊙20	Long (I32)

IADD – Integer addition

Block Symbol

Licence: STANDARD

Function Description

The IADD block sums two integer input signals n = i1 + i2. The range of integer numbers in a computer is always restricted by the variable type. This block uses the vtype parameter to specify the type. If the sum fits in the range of the given type, the result is the ordinary sum. In the other cases the result depends on the SAT parameter.

The overflow is not checked for SAT = off, i.e. the output E = off and the output value n corresponds with the arithmetics of the processor. E.g. for the Short type, which has the range of -32768..+32767, we obtain 30000 + 2770 = -32766).

For SAT = on the overflow results in setting the error output to E = on and the n output to the nearest displayable value. For the above mentioned example we get 30000 + 2770 = 32767).

Inputs

i1	First integer input of the block	↓-9.22E+18 ↑9.22E+18	Long (I32)
i2	Second integer input of the block	↓-9.22E+18 ↑9.22E+18	Long (I32)

Outputs

n	Integer sum of the input signals	Long (I32)
E	Error flag off ... No error on .... An error occurred	Bool

Parameters

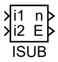
vtype	Numeric type -- .... 2 ..... Byte (U8) 3 ..... Short (I16) 4 ..... Long (I32) 5 ..... Word (U16) 6 ..... DWord (U32) -- .... -- .... -- .... 10 .... Large (I64)	⊙4	Long (I32)

SAT	Saturation (overflow) checking off ... Overflow is not checked on .... Overflow is checked	Bool

ISUB – Integer subtraction

Block Symbol

Licence: STANDARD



Function Description

The ISUB block subtracts two integer input signals n = i1 − i2. The range of integer numbers in a computer is always restricted by the variable type. This block uses the vtype parameter to specify the type. If the difference fits in the range of the given type, the result is the ordinary sum. In the other cases the result depends on the SAT parameter.

The overflow is not checked for SAT = off, i.e. the output E = off and the output value n corresponds with the arithmetics of the processor. E.g. for the Short type, which has the range of -32768..+32767, we obtain 30000 - -2770 = -32766).

For SAT = on the overflow results in setting the error output to E = on and the n output to the nearest displayable value. For the above mentioned example we get 30000 - -2770 = 32767).

Inputs

i1	First integer input of the block	↓-9.22E+18 ↑9.22E+18	Long (I32)
i2	Second integer input of the block	↓-9.22E+18 ↑9.22E+18	Long (I32)

Parameters

vtype	Numeric type	⊙4	Long (I32)
	2 ..... Byte (range 0 ... 255) 3 ..... Short (range -32768 ... 32767) 4 ..... Long (range -2147483648 ... 2147483647) 5 ..... Word (range 0 ... 65536) 6 ..... DWord (range 0 ... 4294967295) 10 .... Large (range -9223372036854775808...9223372036854775807)		
SAT	Saturation (overflow) checking		Bool
	off ... Overflow is not checked on .... Overflow is checked		

Outputs

n	Integer difference between the input signals	Long (I32)

`E`	Error flag `off` ... No error `on` .... An error occurred	`Bool`

IMUL – Integer multiplication

Block Symbol

Licence: STANDARD

Function Description

The IMUL block multiplies two integer input signals n = i1 ∗ i2. The range of integer numbers in a computer is always restricted by the variable type. This block uses the vtype parameter to specify the type. If the multiple fits in the range of the given type, the result is the ordinary multiple. In the other cases the result depends on the SAT parameter.

The overflow is not checked for SAT = off, i.e. the output E = off and the output value n corresponds with the arithmetics of the processor. E.g. for the Short type, which has the range of -32768..+32767, we obtain 2000 * 20 = -25536).

For SAT = on the overflow results in setting the error output to E = on and the n output to the nearest displayable value. For the above mentioned example we get 2000 * 20 = 32767).

Inputs

i1	First integer input of the block	↓-9.22E+18 ↑9.22E+18	Long (I32)
i2	Second integer input of the block	↓-9.22E+18 ↑9.22E+18	Long (I32)

Parameters

vtype	Numeric type -- .... 2 ..... Byte (U8) 3 ..... Short (I16) 4 ..... Long (I32) 5 ..... Word (U16) 6 ..... DWord (U32) -- .... -- .... -- .... 10 .... Large (I64)	⊙4	Long (I32)
SAT	Saturation (overflow) checking off ... Overflow is not checked on .... Overflow is checked		Bool

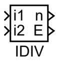
Outputs

`n`	Integer product of the input signals	`Long (I32)`
`E`	Error flag `off` ... No error `on` .... An error occurred	`Bool`

IDIV – Integer division

Block Symbol

Licence: STANDARD



Function Description

The IDIV block performs an integer division of two integer input signals, n = i1 ÷ i2, where ÷ stands for integer division operator. If the ordinary (non-integer, normal) quotient of the two operands is an integer number, the result of integer division is the same. In other cases the resulting value is obtained by trimming the non-integer quotient's decimals (i.e. rounding towards lower integer number). In case i2 = 0, the output E is set to on and the output n is substituted by n = nerr.

Inputs

i1	First integer input of the block	↓-9.22E+18 ↑9.22E+18	Long (I32)
i2	Second integer input of the block	↓-9.22E+18 ↑9.22E+18	Long (I32)

Outputs

n	Integer quotient of the inputs	Long (I32)
E	Error flag – division by zero	Bool

Parameters

vtype	Numeric type 2 ..... Byte 3 ..... Short 4 ..... Long 5 ..... Word 6 ..... DWord 10 .... Large	⊙4	Long (I32)
nerr	Substitute value for an error case	⊙1	Long (I32)

IMOD – Remainder after integer division

Block Symbol

Licence: STANDARD

Function Description

The IMOD block divides two integer input signals, n = i1%i2, where % stands for remainder after integer division operator (modulo). If both numbers are positive and the divisor is greater than one, the result is either zero (for commensurable numbers) or a positive integer lower than the divisor. In the case that one of the numbers is negative, the result has the sign of the dividend, e.g. $15\%10 = 5$, $15\%(-10) = 5$, but $(-15)\%10 = -5$. In case i2 = 0, the output E is set to on and the output n is substituted by n = nerr.

Inputs

i1	First integer input of the block	↓-9.22E+18 ↑9.22E+18	Long (I32)
i2	Second integer input of the block	↓-9.22E+18 ↑9.22E+18	Long (I32)

Outputs

n	Remainder after integer division	Long (I32)
E	Error flag – division by zero	Bool

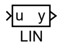
Parameters

vtype	Numeric type 2 ..... Byte 3 ..... Short 4 ..... Long 5 ..... Word 6 ..... DWord 10 .... Large	⊙4	Long (I32)
nerr	Substitute value for an error case	⊙1	Long (I32)

LIN – Linear interpolation

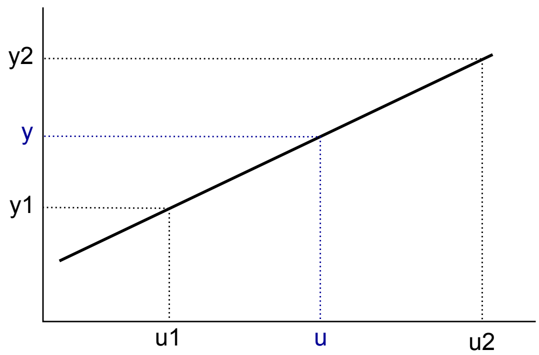
Block Symbol

Licence: STANDARD



Function Description

The LIN block performs linear interpolation. The following figure illustrates the influence of the input u and given interpolation points [u1, y1] and [u2, y2] on the output y.



Input

u	Analog input of the block		Double (F64)

Output

y	Analog output of the block		Double (F64)

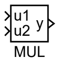
Parameters

u1	x-coordinate of the 1st interpolation node		Double (F64)
y1	y-coordinate of the 1st interpolation node		Double (F64)
u2	x-coordinate of the 2nd interpolation node	⊙1.0	Double (F64)
y2	y-coordinate of the 2nd interpolation node	⊙1.0	Double (F64)

MUL – Multiplication of two signals

Block Symbol

Licence: STANDARD



Function Description

The MUL block multiplies two analog input signals $y = u1 \cdot u2$.

Inputs

u1	First analog input of the block	Double (F64)
u2	Second analog input of the block	Double (F64)

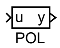
Output

y	Product of the input signals	Double (F64)

POL – Polynomial evaluation

Block Symbol

Licence: STANDARD



Function Description

The POL block evaluates the polynomial of the form:

$$y = a_0 + a_1 u + a_2 u^2 + a_3 u^3 + a_4 u^4 + a_5 u^5 + a_6 u^6 + a_7 u^7 + a_8 u^8.$$

The polynomial is internally evaluated by using the Horner scheme to improve the numerical robustness.

Input

u	Analog input of the block	Double (F64)

Output

y	Analog output of the block	Double (F64)

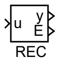
Parameters

ai	The i-th coefficient of the polynomial, $i = 0, 1, \ldots, 8$	Double (F64)

REC – Reciprocal value

Block Symbol

Licence: STANDARD



Function Description

The REC block computes the reciprocal value of the input signal u. The output is then

$$y = \frac{1}{u}.$$

In case $u = 0$, the error indicator is set to $E =$ on and the output is set to the substitutional value $y =$ yerr.

Input

u	Analog input of the block	Double (F64)

Outputs

y	Analog output of the block	Double (F64)
E	Error flag – division by zero	Bool

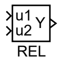
Parameter

yerr	Substitute value for an error case	⊙1.0	Double (F64)

REL – Relational operator

Block Symbol

Licence: STANDARD



Function Description

The REL block evaluates the binary relation u1 ∘ u2 between the values of the input signals and sets the output Y according to the result of the relation "∘". The output is set to Y = on when relation holds, otherwise it is zero (relation does not hold). The binary operation codes are listed below.

Inputs

u1	First analog input of the block	Double (F64)
u2	Second analog input of the block	Double (F64)

Output

Y	Logical output indicating whether the relation holds	Bool

Parameter

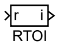
irel — Relation type — ⊙1 — Long (I32)

1 ..... equality (==)
2 ..... inequality (!=)
3 ..... less than (<)
4 ..... greater than (>)
5 ..... less than or equal to (<=)
6 ..... greater than or equal to (>=)

RTOI – Real to integer number conversion

Block Symbol

Licence: STANDARD



Function Description

The RTOI block converts the real number r to a signed integer number i. The resulting rounded value is defined by:

$$
\mathtt{i} := \begin{cases} -2147483648 & \text{for} \quad \mathtt{r} \leq -2147483648.0 \\ \mathrm{round}(\mathtt{r}) & \text{for} \quad -2147483648.0 < \mathtt{r} \leq 2147483647.0 \\ 2147483647 & \text{for} \quad \mathtt{r} > 2147483647.0 \end{cases},
$$

where round(r) stands for rounding to the nearest integer number. The number of the form n+0.5 (n is integer) is rounded to the integer number with the higher absolute value, i.e. round(1.5) = 2, round(−2.5) = −3.

Note that the numbers −2147483648 and 2147483647 correspond with the lowest and the highest signed number representable in 32-bit format respectively (0x7FFFFFFF and 0x80000000 in hexadecimal form in the C language). This limits are valid if the vtype parameter has default value. Limits of selected data type is applied in other cases.

Input

r	Analog input of the block	Double (F64)

Parameters

vtype	Output numeric type -- .... 2 ..... Byte (U8) 3 ..... Short (I16) 4 ..... Long (I32) 5 ..... Word (U16) 6 ..... DWord (U32) -- .... -- .... -- .... 10 .... Large (I64)	⊙4	Long (I32)
SAT	Saturation (overflow) checking	⊙on	Bool

Output

`i`	Rounded and converted input signal	`Long (I32)`

SQR – Square value

Block Symbol

Licence: STANDARD

Function Description

The SQR block raises the input u to the power of 2. The output is then

$$y = u^2.$$

Input

u	Analog input of the block	Double (F64)

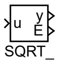
Output

y	Square of the input signal	Double (F64)

SQRT – Square root

Block Symbol

Licence: STANDARD



Function Description

The SQRT block computes the square root of the input u. The output is then

$$y = \sqrt{u}.$$

In case $u < 0$, the error indicator is activated (E = on) and the output y is set to the substitute value y = yerr.

Input

u	Analog input of the block	Double (F64)

Outputs

y	Square root of the input signal	Double (F64)
E	Error flag off . . . No error on . . . . Square root of negative number	Bool

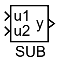
Parameter

yerr	Substitute value for an error case	⊙1.0	Double (F64)

SUB – Subtraction of two signals

Block Symbol

Licence: STANDARD



Function Description

The SUB block subtracts two input signals. The output is given by

$$y = u1 - u2.$$

Consider using the ADDOCT block for addition or subtraction of multiple signals.

Inputs

u1	Analog input of the block	Double (F64)
u2	Analog input of the block	Double (F64)

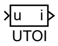
Output

y	Difference between the two input signals	Double (F64)

UTOI – Unsigned to signed integer number conversion

Block Symbol

Licence: STANDARD



Function Description

The block regards input (positive) number as a binary complement representation of signed number (e.g. comon representation in today processors). The range of the representation is defined by the parameter `bits`. Typical ussage is parsing signal from data structures of comunication driver. If one of comunicating processor is big-endian and second processor little-endian you can need swap byte order. It is realize by the `SWAP` parameter. It is recommended to set `SWAP=on` only in case the `bits` parameter is set to 16 or 32 because other cases ussualy not solve endianess problem.

Input

`u`	Unsigned input signal	↓-9.22337E+18 ↑9.22337E+18	Large (I64)

Parameters

`bits`	Valid (LSB) bits in input signal	↓2 ↑64 ⊙16	Long (I32)
`SWAP`	Swap input byte order		Bool

Output

`i`	Converted (signed) input signal		Large (I64)

Chapter 5

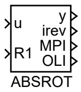
ANALOG – Analog signal processing

Contents

ABSROT – Processing data from absolute position sensor

Block Symbol

Licence: ADVANCED



Function Description

The ABSROT function block is intended for processing the data from absolute position sensor on rotary equipment, e.g. a shaft. The absolute sensor has a typical range of 5° to 355° (or -175° to +175°) but in some cases it is necessary to control the rotation over a range of more than one revolution. The function block assumes a continuous position signal, therefore the transition from 355° to 5° in the input signal means that one revolution has been completed and the angle is in fact 365°.

In the case of long-term unidirectional operation the precision of the estimated position y deteriorates due to the precision of the double data type. For that case the R1 input is available to reset the position y to the base range of the sensor. If the RESR flag is set to RESR = on, the irev revolutions counter is also reset by the R1 input. In all cases it is necessary to reset all accompanying signals (e.g. the sp input of the corresponding controller).

The MPI output indicates that the absolute sensor reading is near to the middle of the range, which may be the appropriate time to reset the block. On the other hand, the OLI output indicates that the sensor reached the so-called dead-angle where it cannot report valid data.

Inputs

u	Signal from the absolute position sensor	Double (F64)
R1	Block reset	Bool

Outputs

y	Position output	Double (F64)
irev	Number of finished revolutions	Long (I32)
MPI	Mid-point indicator	Bool
OLI	Off-limits indicator	Bool

Parameters

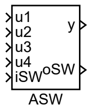
lolim	Lower limit of the sensor reading	⊙-3.14159265	Double (F64)
hilim	Upper limit of the sensor reading	⊙3.14159265	Double (F64)
tol	Tolerance for the mid-point indicator	⊙0.5	Double (F64)

hys	Hysteresis for the mid-point indicator	Double (F64)
RESR	Flag for resetting the revolutions counter off ... Reset only the estimated position y on .... Reset also the irev revolutions counter	Bool

ASW – Switch with automatic selection of input

Block Symbol

Licence: ADVANCED



Function Description

The ASW block copies one of the inputs u1, ..., u4 or one of the parameters p1, ..., p4 to the output y. The appropriate input signal is copied to the output as long as the input signal iSW belongs to the set $\{1, 2, 3, 4\}$ and the parameters are copied when iSW belongs to the set $\{-1, -2, -3, -4\}$ (i.e. y $=$ p1 for iSW $= -1$, y $=$ u3 for iSW $= 3$ etc.). If the iSW input signal differs from any of these values (i.e. iSW $= 0$ or iSW < -4 or iSW > 4), the output is set to the value of input or parameter which has changed the most recently. The signal or parameter is considered changed when it differs by more than delta from its value at the moment of its last change (i.e. the changes are measured integrally, not as a difference from the last sample). The following priority order is used when changes occur simultaneously in more than one signal: p4, p3, p2, p1, u4, u3, u2, u1. The identifier of input signal or parameter which is copied to the output y is always available at the oSW output.

The ASW block has one special feature. The updated value of y is copied to all the parameters p1, ..., p4. This results in all external tools reading the same value y. This is particularly useful in higher-level systems which use the set&follow method (e.g. a slider in Iconics Genesis). This feature is not implemented in Simulink as there are no ways to read the values of inputs by external programs.

ATTENTION! One of the inputs u1, ..., u4 can be delayed by one step when the block is contained in a loop. This might result in an illusion, that the priority is broken (the oSW output then shows that the most recently changed signal is the delayed one). In such a situation the LPBRK block(s) must be used in appropriate positions.

Inputs

u1..u4	Analog input signals to be selected from	Double (F64)
iSW	Active signal or parameter selector	Long (I32)

Outputs

y	The selected analog signal or parameter	Double (F64)
oSW	Identifier of the selected signal or parameter	Long (I32)

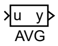
Parameters

`delta`	Threshold for detecting a change	⊙1e-06	`Double (F64)`
`p1..p4`	Parameters to be selected from		`Double (F64)`

AVG – Moving average filter

Block Symbol

Licence: STANDARD



Function Description

The AVG block computes a moving average from the last n samples according to the formula

$$y_k = \frac{1}{n}(u_k + u_{k-1} + \cdots + u_{k-n+1}).$$

There is a limitation $\mathtt{n} < N$, where N depends on the implementation.

If the last n samples are not yet known, the average is computed from the samples available.

Input

u	Input signal to be filtered	Double (F64)

Output

y	Filtered output signal	Double (F64)

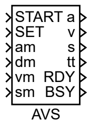
Parameter

n	Number of samples to compute the average from ↓1 ↑10000000 ⊙10	Long (I32)
nmax	Limit for parameter n (used for internal memory allocation) ↓10 ↑10000000 ⊙100	Long (I32)

AVS – Motion control unit

Block Symbol

Licence: ADVANCED



Function Description

The AVS block generates time-optimal trajectory from initial steady position 0 to a final steady position sm while respecting the constraints on the maximal acceleration am, maximal deceleration dm and maximal velocity vm. When rising edge (off→on) occurs at the SET input, the block is initialized for current values of the inputs am, dm, vm and sm. The RDY output is set to offbefore the first initialization and during the initialization phase, otherwise it is set to 1. When rising edge (off→on) occurs at the START input, the block generates the trajectory at the outputs a, v, s and tt, where the signals correspond to acceleration, velocity, position and time respectively. The BSY output is set to onwhile the trajectory is being generated, otherwise it is off.

Inputs

START	Starting signal (rising edge)	Bool
SET	Initialize/compute the trajectory for the current inputs	Bool
am	Maximal allowed acceleration [m/s^2]	Double (F64)
dm	Maximal allowed deceleration [m/s^2]	Double (F64)
vm	Maximum allowed velocity [m/s]	Double (F64)
sm	Desired final position [m] (initial position is 0)	Double (F64)

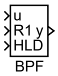
Outputs

a	Acceleration [m/s^2]	Double (F64)
v	Velocity [m/s]	Double (F64)
s	Position [m]	Double (F64)
tt	Time [s]	Double (F64)
RDY	Flag indicating that the block is ready to generate the trajectory	Bool
BSY	Flag indicating that the trajectory is being generated	Bool

BPF – Band-pass filter

Block Symbol

Licence: STANDARD



Function Description

The BPF implements a second order filter in the form

$$F_s = \frac{2\xi as}{a^2s^2 + 2\xi as + 1},$$

where a and ξ are are the block parameters fm and xi respectively. The fm parameter defines the middle of the frequency transmission band and xi is the relative damping coefficient.

If ISSF = on, then the state of the filter is set to the steady value at the block initialization according to the input signal u.

Input

u	Input signal to be filtered	Double (F64)
R1	Block reset (same state as after init)	Bool
HLD	Hold block execution	Bool

Output

y	Filtered output signal	Double (F64)

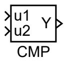
Parameters

fm	Peak frequency, middle of the frequency transmission band [Hz] ⊙1.0	Double (F64)
xi	Relative damping coefficient (recommended value 0.5 to 1) ⊙0.707	Double (F64)
ISSF	Steady state at start-up flag off . . . Zero initial state on . . . . Initial steady state	Bool

CMP – Comparator with hysteresis

Block Symbol

Licence: STANDARD



Function Description

The CMP block compares the inputs u1 and u2 with the hysteresis h as follows:

$$
\begin{aligned}
Y_{-1} &= 0, \\
Y_k &= hyst(e_k), \;\; k = 0, 1, 2, \ldots
\end{aligned}
$$

where

$$e_k = u1_k - u2_k$$

and

$$hyst(e_k) = \begin{cases} 0 & \text{for} \quad e_k \leq -h \\ Y_{k-1} & \text{for} \quad e_k \in (-h, h) \\ 1 & \text{for} \quad e_k \geq h \quad (e_k > h \text{ for } h = 0) \end{cases}$$

The indexed variables refer to the values of the corresponding signal in the cycle defined by the index, i.e. Y_{k-1} denotes the value of output in the previous cycle/step. The value Y_{-1} is used only once when the block is initialized ($k = 0$) and the difference of the input signals e_k is within the hysteresis limits.

Inputs

u1	First analog input of the block	Double (F64)
u2	Second analog input of the block	Double (F64)

Output

Y	Logical output of the block	Bool

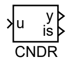
Parameter

hys	Hysteresis	↓0.0 ⊙0.5	Double (F64)

CNDR – Nonlinear conditioner

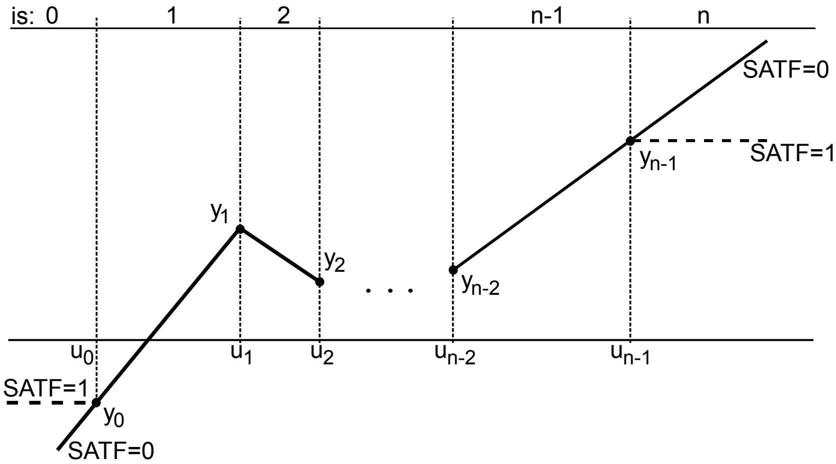
Block Symbol

Licence: STANDARD



Function Description

The CNDR block can be used for compensation of complex nonlinearities by a piecewise linear transformation which is depicted below.



It is important to note that in the case of u $< u_0$ or u $> u_{n-1}$ the output depends on the SATF parameter.

Input

u	Analog input of the block	Double (F64)

Outputs

y	Analog output of the block	Double (F64)
is	Active sector of nonlinearity (see the figure above)	Long (I32)

Parameters

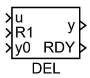
nmax	Reserved Size of *up*, *yp* arrays	↓4 ⊙10	Long (I32)
SATF	Saturation flag off . . . Signal not limited on . . . . Saturation limits active	⊙on	Bool

`up`	Vector of increasing u-coordinates	⊙`[0.0 3.9 3.9 9.0 14.5 20.0]`	`Double (F64)`
`yp`	Vector of y-coordinates	⊙`[0.0 0.0 15.8 38.4 72.0 115.0]`	`Double (F64)`

DEL – Delay with initialization

Block Symbol

Licence: STANDARD



Function Description

The DEL block implements a delay of the input signal u. The signal is shifted n samples backwards, i.e.

$$y_k = u_{k-n}.$$

If the last n samples are not yet known, the output is set to

$$y_k = y_0,$$

where $\mathtt{y}_0$ is the initialization input signal. This can happen after restarting the control system or after resetting the block (R1: off→on→off) and it is indicated by the output RDY = off.

Inputs

u	Analog input of the block	Double (F64)
R1	Block reset	Bool
y0	Initial output value	Double (F64)

Outputs

y	Delayed input signal	Double (F64)
RDY	Ready flag indicating that the buffer is filled with the input signal samples	Bool

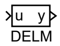
Parameter

n	Delay (number of samples). The resulting time delay is $\mathtt{n} \cdot T_S$, where T_S is the block execution period. ↓0 ↑10000000 ⊙10	Long (I32)
nmax	Limit for parameter n (used for internal memory allocation) ↓10 ↑10000000 ⊙100	Long (I32)

DELM – Time delay

Block Symbol

Licence: STANDARD



Function Description

The DELM block implements a time delay of the input signal. The length of the delay is given by rounding the del parameter to the nearest integer multiple of the block execution period. The output signal is y = 0 for the first del seconds after initialization.

Input

u	Analog input of the block	Double (F64)

Output

y	Delayed input signal	Double (F64)

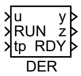
Parameter

del	Time delay [s] ⊙1.0	Double (F64)
nmax	Size of delay buffer del (number of samples). Used for internal memory allocation. ↓10 ↑10000000 ⊙100	Long (I32)

DER – Derivation, filtering and prediction from the last n+1 samples

Block Symbol

Licence: STANDARD



Function Description

The DER block interpolates the last $n + 1$ samples ($n \leq N - 1$, N is implementation dependent) of the input signal u by a line $y = at + b$ using the least squares method. The starting point of the time axis is set to the current sampling instant.

In case of RUN = on the outputs y and z are computed from the obtained parameters a and b of the linear interpolation as follows:

$$\begin{array}{llcl} \text{Derivation:} & y & = & a \\ \text{Filtering:} & z & = & b, \text{ for } t_p = 0 \\ \text{Prediction:} & z & = & at_p + b, \text{ for } t_p > 0 \\ \text{Retrodiction:} & z & = & at_p + b, \text{ for } t_p < 0 \end{array}$$

In case of RUN = off or $n + 1$ samples of the input signal are not yet available (RDY = off), the outputs are set to $y = 0$, $z = u$.

Inputs

u	Analog output of the block	Double (F64)
RUN	Enable execution off ... tracking (z = u) on .... filtering (y – estimate of the derivative, z – estimate of u at time t_p)	Bool
tp	Time instant for prediction/filtering (tp = 0 corresponds with the current sampling instant)	Double (F64)

Outputs

y	Estimate of input signal derivative	Double (F64)
z	Predicted/filtered input signal	Double (F64)
RDY	Ready flag (all $n + 1$ samples are available)	Bool

Parameters

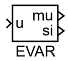
n	Number of samples for interpolation ($n + 1$ samples are used); $1 \leq n \leq nmax$ ↓1 ↑10000000 ⊙10	Long (I32)

`nmax`	Limit for parameter `n` (used for internal memory allocation) $\downarrow$1 $\uparrow$10000000 $\odot$10	`Long (I32)`

EVAR – Moving mean value and standard deviation

Block Symbol

Licence: STANDARD



Function Description

The EVAR block estimates the mean value mu (μ) and standard deviation si (σ) from the last n samples of the input signal u according to the formulas

$$\mu_k = \frac{1}{\mathtt{n}} \sum_{i=0}^{n-1} \mathtt{u}_{k-i}$$

$$\sigma_k = \sqrt{\frac{1}{\mathtt{n}} \sum_{i=0}^{n-1} \mathtt{u}_{k-i}^2 - \mu_k^2}$$

where k stands for the current sampling instant.

Input

u	Analog input of the block	Double (F64)

Outputs

mu	Mean value of the input signal	Double (F64)
si	Standard deviation of the input signal	Double (F64)

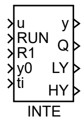
Parameter

n	Number of samples to estimate the statistical properties from ↓2 ↑10000000 ⊙100	Long (I32)
nmax	Limit value of parameter n (used for internal memory allocation) ↓10 ↑10000000 ⊙200	Long (I32)

INTE – Controlled integrator

Block Symbol

Licence: STANDARD



Function Description

The INTE block implements a controlled integrator with variable integral time constant ti and two indicators of the output signal level (ymin a ymax). If RUN = on and R1 = off then

$$y(t) = \frac{1}{T_i}\int_0^t u(\tau)d\tau + C,$$

where $C = y0$. If RUN = off and R1 = off then the output y is frozen to the last value before the falling edge at the RUN input signal. If R1 = on then the output y is set to the initial value y0. The integration uses the trapezoidal method as follows

$$y_k = y_{k-1} + \frac{T_S}{2T_i}(u_k + u_{k-1}),$$

where T_S is the block execution period. If $T_i = 0$, the block realize summation by following equation

$$y_k = y_{k-1} + u_k.$$

If $T_i < 0$, the block behaviour is undefined.

Consider using the SINT block, whose simpler structure and functionality might be sufficient for elementary tasks.

Inputs

u	Analog input of the block	Double (F64)
RUN	Enable execution off ... Integration stopped on .... Integration running	Bool
R1	Block reset, initialization of the integrator output to y0	Bool
y0	Initial output value	Double (F64)
ti	Integral time constant	Double (F64)

Outputs

y	Integrator output	Double (F64)
Q	Running integration indicator	Bool
LY	Lower level indicator (y < ymin)	Bool

HY	Upper level indicator (y > ymax)		Bool

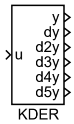
Parameters

ymin	Lower level definition	⊙-1.0	Double (F64)
ymax	Upper level definition	⊙1.0	Double (F64)
SAT	Limit output if level limit is reach		Bool

KDER – Derivation and filtering of the input signal

Block Symbol

Licence: ADVANCED



Function Description

The KDER block is a Kalman-type filter of the norder-th order aimed at estimation of derivatives of locally polynomial signals corrupted by noise. The order of derivatives ranges from 0 to norder $-$ 1. The block can be used for derivation of almost arbitrary input signal u $= u_0(t) + v(t)$, assuming that the frequency spectrums of the signal and noise differ.

The block is configured by only two parameters pbeta and norder. The pbeta parameter depends on the sampling period T_S, frequency properties of the input signal u and also the noise to signal ratio. An approximate formula pbeta $\approx T_S\omega_0$ can be used. The frequency spectrum of the input signal u should be located deep down below the cutoff frequency ω_0. But at the same time, the frequency spectrum of the noise should be as far away from the cutoff frequency ω_0 as possible. The cutoff frequency ω_0 and thus also the pbeta parameter must be lowered for strengthening the noise rejection.

The other parameter norder must be chosen with respect to the order of the estimated derivations. In most cases the 2nd or 3rd order filter is sufficient. Higher orders of the filter produce better derivation estimates for non-polynomial signals at the cost of slower tracking and higher computational cost.

Input

u	Input signal to be filtered	Double (F64)

Outputs

y	Filtered input signal	Double (F64)
dy	Estimated 1st order derivative	Double (F64)
d2y	Estimated 2nd order derivative	Double (F64)
d3y	Estimated 3rd order derivative	Double (F64)
d4y	Estimated 4th order derivative	Double (F64)
d5y	Estimated 5th order derivative	Double (F64)

Parameters

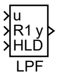
norder	Order of the derivative filter	↓2 ↑10 ⊙3	Long (I32)

`pbeta`	Bandwidth of the derivative filter	$\downarrow$`0.0` $\odot$`0.1`	`Double (F64)`

LPF – Low-pass filter

Block Symbol

Licence: STANDARD



Function Description

The LPF block implements a second order filter in the form

$$F_s = \frac{1}{a^2 s^2 + 2\xi a s + 1},$$

where

$$a = \frac{\sqrt{\sqrt{2}\sqrt{2\xi^4 - 2\xi^2 + 1} - 2\xi^2 + 1}}{2\pi f_b}$$

and fb and $\xi =$ xi are the block parameters. The fb parameter defines the filter bandwidth and xi is the relative damping coefficient. The recommended value is xi $= 0.71$ for the Butterworth filter and xi $= 0.87$ for the Bessel filter.

If ISSF = on, then the state of the filter is set to the steady value at the block initialization according to the input signal u.

Input

u	Input signal to be filtered	Double (F64)
R1	Block reset (same state as after init)	Bool
HLD	Hold block execution	Bool

Output

y	Filtered output signal	Double (F64)

Parameters

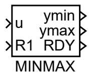
fb	Filter bandwidth [Hz]; the frequencies in the range $\langle 0, \mathtt{fb} \rangle$ pass through the filter, the attenuation at the frequency fb is 3 dB and approximately 40 dB at $10 \cdot \mathtt{fb}$; it must hold $f_b < \frac{1}{10 T_S}$ for proper function of the filter, where T_S is the block execution period ⊙1.0	Double (F64)
xi	Relative damping coefficient (recommended value 0.5 to 1) ⊙0.707	Double (F64)

`ISSF`	Steady state at start-up `off` ... Zero initial state `on` .... Initial steady state	`Bool`

MINMAX – Running minimum and maximum

Block Symbol

Licence: STANDARD



Function Description

The MINMAX function block evaluates minimum and maximum from the last n samples of the u input signal. The output RDY = off indicates that the buffer contains less than n samples. In such a case the minimum and maximum are found among the available samples.

Inputs

u	Analog input of the block	Double (F64)
R1	Block reset	Bool

Outputs

ymin	Minimal value found	Double (F64)
ymax	Maximal value found	Double (F64)
RDY	Ready flag (buffer filled)	Bool

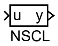
Parameters

n	Number of samples for analysis (buffer length) ↓1 ↑10000000 ⊙100	Long (I32)
nmax	Limit value of parameter n (used for internal memory allocation) ↓10 ↑10000000 ⊙200	Long (I32)

NSCL – Nonlinear scaling factor

Block Symbol

Licence: STANDARD

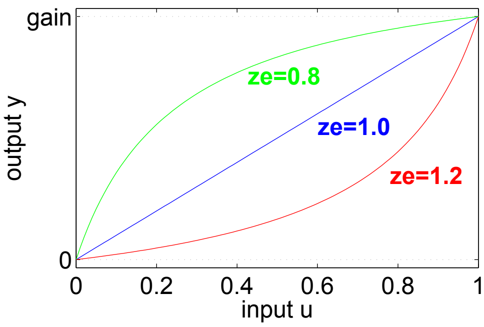


Function Description

The NSCL block compensates common nonlinearities of the real world (e.g. the servo valve nonlinearity) by using the formula

$$y = \text{gain} \frac{u}{ze + (1 - ze) \cdot u},$$

where gain and ze are the parameters of the block. The choice of ze within the interval $(0, 1)$ leads to concave transformation, while $ze > 1$ gives a convex transformation.



Input

u	Analog input of the block	Double (F64)

Output

y	Analog output of the block	Double (F64)

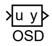
Parameters

gain	Signal gain	⊙1.0	Double (F64)
ze	Shaping parameter	⊙1.0	Double (F64)

OSD – One step delay

Block Symbol

Licence: STANDARD



Function Description

The OSD block implements a one step delay of the input signal u. The length of the step delay (in seconds) is given by the task period (see the EXEC function block description for details).

Inputs

u	Input of the block	Any

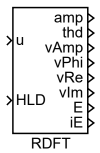
Outputs

y	Delayed input signal	Any

RDFT – Running discrete Fourier transform

Block Symbol

Licence: ADVANCED



Function Description

The RDFT function block analyzes the analog input signal using the discrete Fourier transform with the fundamental frequency freq and optional higher harmonic frequencies. The computations are performed over the last m samples of the input signal u, where m $=$ nper/freq/T_S, i.e. from the time-window of the length equivalent to nper periods of the fundamental frequency.

If nharm > 0 the number of monitored higher harmonic frequencies is given solely by this parameter. On the contrary, for nharm $= 0$ the monitored frequencies are given by the user-defined vector parameter freq2.

For each frequency the amplitude (vAmp output), phase-shift (vPhi output), real/cosine part (vRe output) and imaginary/sine part (vIm output). The output signals have the vector form, therefore the computed values for all the frequencies are contained within. Use the VTOR function block to disassemble the vector signals.

Inputs

u	Analog input of the block	Double (F64)
HLD	Hold	Bool

Outputs

amp	Amplitude of the fundamental frequency	Double (F64)
thd	Total harmonic distortion (only for nharm ≥ 1)	Double (F64)
vAmp	Vector of amplitudes at given frequencies	Reference
vPhi	Vector of phase-shifts at given frequencies	Reference
vRe	Vector of real parts at given frequencies	Reference
vIm	Vector of imaginary parts at given frequencies	Reference
E	Error flag	Bool
iE	Error code i ..... REXYGEN general error	Error

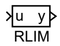
Parameters

`freq`	Fundamental frequency	↓1e-09 ↑1e+09 ⊙1.0	Double (F64)
`nper`	Number of periods to calculate upon	↓1 ↑10000 ⊙10	Long (I32)
`nharm`	Number of monitored harmonic frequencies	↓0 ↑16 ⊙3	Long (I32)
`ifrunit`	Frequency units 1 ..... Hz 2 ..... rad/s	↓1 ↑2 ⊙1	Long (I32)
`iphunit`	Phase shift units 1 ..... degrees 2 ..... radians	↓0 ↑2 ⊙1	Long (I32)
`nmax`	Allocated size of array	↓10 ↑10000000 ⊙8192	Long (I32)
`freq2`	Vector of user-defined monitored frequencies	⊙[2.0 3.0 4.0]	Double (F64)

RLIM – Rate limiter

Block Symbol

Licence: STANDARD



Function Description

The RLIM block copies the input signal u to the output y, but the maximum allowed rate of change is limited. The limits are given by the time constants tp and tn:

the steepest rise per second:	$1/\texttt{tp}$
the steepest descent per second:	$-1/\texttt{tn}$

Input

u	Input signal to be filtered	Double (F64)

Output

y	Filtered output signal	Double (F64)

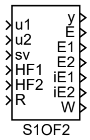
Parameters

tp	Time constant defining the maximum allowed rise	⊙2.0	Double (F64)
tn	Time constant defining the maximum allowed descent (note that $\texttt{tn} > 0$)	⊙2.0	Double (F64)

S1OF2 – One of two analog signals selector

Block Symbol

Licence: ADVANCED



Function Description

The S1OF2 block assesses the validity of two input signals u1 and u2 separately. The validation method is equal to the method used in the SAI block. If the signal u1 (or u2) is marked invalid, the output E1 (or E2) is set to on and the error code is sent to the iE1 (or iE2) output. The S1OF2 block also evaluates the difference between the two input signals. The internal flag D is set to on if the differences $|\mathtt{u1}-\mathtt{u2}|$ in the last nd samples exceed the given limit, which is given by the following inequation

$$|\mathtt{u1}-\mathtt{u2}| > \mathtt{pdev}\frac{\mathtt{vmax}-\mathtt{vmin}}{100},$$

where vmin and vmax are the minimal and maximal limits of the inputs u1 and u2 and pdev is the allowed percentage difference with respect to the overall range of the input signals. The value of the output y depends on the validity of the input signals (flags E1 and E2) and the internal difference flag D as follows:

(i) If $\mathtt{E1}=\mathtt{off}$ **and** $\mathtt{E2}=\mathtt{off}$ **and** $\mathtt{D}=\mathtt{off}$, then the output y depends on the mode parameter:

$$\mathtt{y} = \begin{cases} \frac{\mathtt{u1}+\mathtt{u2}}{2}, & \text{for } \mathtt{mode}=1, \\ \min(\mathtt{u1},\mathtt{u2}), & \text{for } \mathtt{mode}=2, \\ \max(\mathtt{u1},\mathtt{u2}), & \text{for } \mathtt{mode}=3, \end{cases}$$

and the output E is set to off unless set to on earlier.

(ii) If $\mathtt{E1}=\mathtt{off}$ **and** $\mathtt{E2}=\mathtt{off}$ **and** $\mathtt{D}=\mathtt{on}$, then $\mathtt{y}=\mathtt{sv}$ and $\mathtt{E}=\mathtt{on}$.

(iii) If $\mathtt{E1}=\mathtt{on}$ **and** $\mathtt{E2}=\mathtt{off}$ **(**$\mathtt{E1}=\mathtt{off}$ **and** $\mathtt{E2}=\mathtt{on}$**)** , then $\mathtt{y}=\mathtt{u2}$ ($\mathtt{y}=\mathtt{u1}$) and the output E is set to off unless set to on earlier.

(iv) If $\mathtt{E1}=\mathtt{on}$ **and** $\mathtt{E2}=\mathtt{on}$, then $\mathtt{y}=\mathtt{sv}$ and $\mathtt{E}=\mathtt{on}$.

The input R resets the inner error flags F1–F4 (see the SAI block) and the D flag. For the input R set permanently to on, the invalidity indicator E1 (E2) is set to on for only one cycle period whenever some invalidity condition is fulfilled. On the other hand, for $\mathtt{R}=0$, the output E1 (E2) is set to on and remains true until the reset (R: off→on). A similar rule holds for the E output. For the input R set permanently to on, the E output is set to `on` for only one cycle period whenever a rising edge occurs in the internal `D` flag (`D` = `off` → `on`). On the other hand, for `R` = 0, the output `E` is set to `on` and remains true until the reset (rising edge `R`: `off`→`on`). The output `W` is set to `on` only in the (iii) or (iv) cases, i.e. at least one input signal is invalid.

Inputs

`u1`	First analog input of the block	`Double (F64)`
`u2`	Second analog input of the block	`Double (F64)`
`sv`	Substitute value for an error case, i.e. `E` = `on`	`Double (F64)`
`HF1`	Hardware error flag for signal `u1` `off` ... The input module of the signal works normally `on` .... Hardware error of the input module occurred	`Bool`
`HF2`	Hardware error flag for signal `u2` `off` ... The input module of the signal works normally `on` .... Hardware error of the input module occurred	`Bool`
`R`	Reset inner error flags of the input signals `u1` and `u2`	`Bool`

Outputs

`y`	Analog output of the block	`Double (F64)`
`E`	Output signal invalidity indicator `off` ... Signal is valid `on` .... Signal is invalid	`Bool`
`E1`	Invalidity indicator for input u1 `off` ... Signal is valid `on` .... Signal is invalid, `y` = `u2`	`Bool`
`E2`	Invalidity indicator for input u2 `off` ... Signal is valid `on` .... Signal is invalid, `y` = `u1`	`Bool`
`iE1`	Reason of input `u1` invalidity `0` ..... Signal valid `1` ..... Signal out of range `2` ..... Signal varies too little `3` ..... Signal varies too little and signal out of range `4` ..... Signal varies too much `5` ..... Signal varies too much and signal out of range `6` ..... Signal varies too much and too little `7` ..... Signal varies too much and too little and signal out of range `8` ..... Hardware error	`Long (I32)`
`iE2`	Reason of input `u2` invalidity, see the `iE1` output	`Long (I32)`
`W`	Warning flag (invalid input signal) `off` ... Both input signals `u1` and `u2` are valid `on` .... At least one of the input signals is invalid	`Bool`

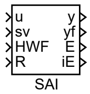
Parameters

nb	Number of samples which are not included in the validity assessment of the signals u1 and u2 after initialization of the block ⊙10	Long (I32)
nc	Number of samples for invariability testing (see the SAI block, condition F2) ⊙10	Long (I32)
nbits	Number of A/D converter bits (source of the signals u1 and u2) ⊙12	Long (I32)
nr	Number of samples for variability testing (see the SAI block, condition F3) ⊙10	Long (I32)
prate	Maximum allowed percentage change of the input u1 (u2) within the last nr samples (with respect to the overall range of the input signals vmax − vmin, see the SAI block) ⊙10.0	Double (F64)
nv	Number of samples for out-of-range testing (see the SAI block, condition F4) ⊙1	Long (I32)
vmin	Lower limit for the input signals u1 and u2 ⊙-1.0	Double (F64)
vmax	Upper limit for the input signals u1 and u2 ⊙1.0	Double (F64)
nd	Number of samples for deviation testing (inner flag D; D is always off for nd = 0) ⊙5	Long (I32)
pdev	Maximum allowed percentage deviation of the inputs u1 and u2 with respect to the overall range of the input signals vmax − vmin ⊙10.0	Double (F64)
mode	Defines how to compute the output signal y when both input signals are valid (E1 = off, E2 = off and D = off) ⊙1 1 ..... Average, $\mathtt{y} = \frac{\mathtt{u1}+\mathtt{u2}}{2}$ 2 ..... Minimum, $\mathtt{y} = \min(\mathtt{u1}, \mathtt{u2})$ 3 ..... Maximum, $\mathtt{y} = \max(\mathtt{u1}, \mathtt{u2})$	Long (I32)

SAI – Safety analog input

Block Symbol

Licence: ADVANCED



Function Description

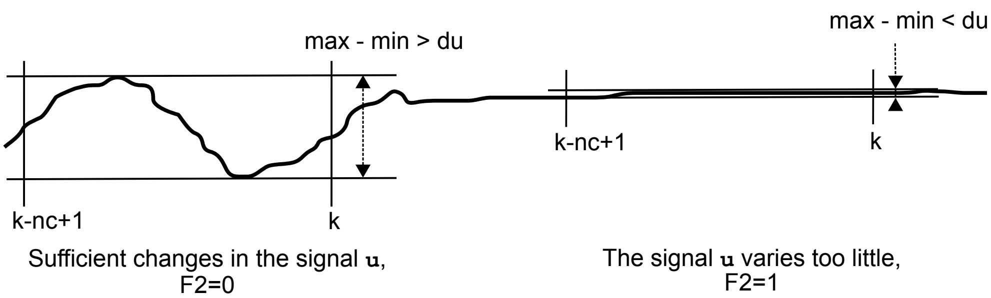
The SAI block tests the input signal u and assesses its validity. The input signal u is considered invalid (the output $\texttt{E} = \texttt{on}$) in the following cases:

F1: Hardware error. The input signal $\texttt{HWF} = \texttt{on}$.

F2: The input signal u varies too little. The last nc samples of the input u lies within the interval of width du,

$$\texttt{du} = \begin{cases} \frac{\texttt{vmax}-\texttt{vmin}}{2^{\texttt{nbits}}}, & \text{for } \texttt{nbits} \in \{8, 9, ..., 16\} \\ 0, & \text{for } \texttt{nbits} \notin \{8, 9, ..., 16\}, \end{cases}$$

where vmin and vmax are the lower and upper limits of the input u, respectively, and nbits is the number of A/D converter bits. The situation when the input signal u varies too little is shown in the following picture:



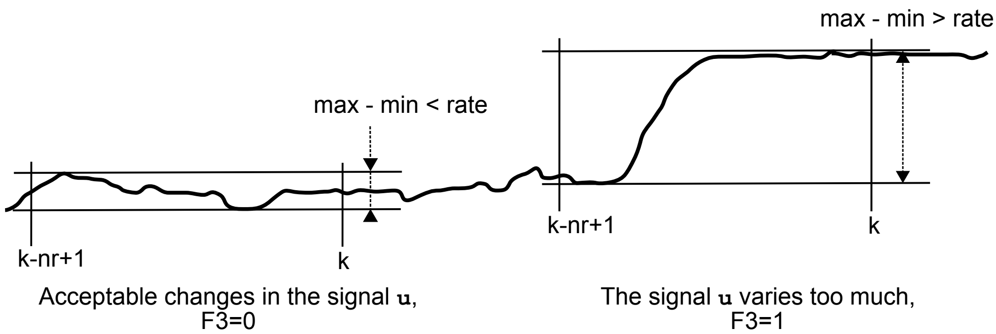
If the parameter nc is set to $\texttt{nc} = 0$, the condition F2 is never fulfilled.

F3: The input signal u varies too much. The last nr samples of the input u filtered by the SPIKE filter have a span which is greater than rate,

$$\texttt{rate} = \texttt{prate}\frac{\texttt{vmax} - \texttt{vmin}}{100},$$

where prate defines the allowed percentage change in the input signal u within the last nr samples (with respect to the overall range of the input signal $\texttt{u} \in \langle\texttt{vmin}, \texttt{vmax}\rangle$).

The block includes a SPIKE filter with fixed parameters mingap $= \frac{\texttt{vmax}-\texttt{vmin}}{100}$ and q $= 2$ suppressing peaks in the input signal to avoid undesirable fulfilling of this condition. See the SPIKE block description for more details. The situation when the input signal u varies too much is shown in the following picture:



If the parameter nr is set to nr $= 0$, the condition F3 is never fulfilled.

F4: The input signal u is out of range. The last nv samples of the input signal u lie out of the allowed range $\langle$vmin, vmax$\rangle$.

If the parameter nv is set to nv $= 0$, the condition F4 is never fulfilled.

The signal u is copied to the output y without any modification when it is considered valid. In the other case, the output y is determined by a substitute value from the sv input. In such a case the output E is set to on and the output iE provides the error code. The input R resets the inner error flags F1–F4. For the input R set permanently to on, the invalidity indicator E is set to on for only one cycle period whenever some invalidity condition is fulfilled. On the other hand, for R = off, the output E is set to on and remains true until the reset (rising edge R: off→on).

The table of error codes iE resulting from the inner error flags F1–F4:

F1	F2	F3	F4	iE
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	*	*	*	8

The nb parameter defines the number of samples which are not included in the validity assessment after initialization of the block (restart). Recommended setting is nb ≥ 5 to allow the SPIKE filter initial conditions to fade away.

Inputs

`u`	Analog input of the block	`Double (F64)`
`sv`	Substitute value to be used when the signal `u` is marked as invalid	`Double (F64)`
`HWF`	Hardware error indicator `off` ... The input module of the signal works normally `on` .... Hardware error of the input module occurred	`Bool`
`R`	Reset inner error flags `F1`–`F4`	`Bool`

Outputs

`y`	Analog output of the block	`Double (F64)`
`yf`	Filtered analog output signal `y`, output of the `SPIKE` filter	`Double (F64)`
`E`	Output signal invalidity indicator `off` ... Signal is valid $\quad$ `yf` = `sv` `on` .... Signal is invalid, `y` =	`Bool`
`iE`	Reason of invalidity `0` ..... Signal valid `1` ..... Signal out of range `2` ..... Signal varies too little `3` ..... Signal varies too little and signal out of range `4` ..... Signal varies too much `5` ..... Signal varies too much and signal out of range `6` ..... Signal varies too much and too little `7` ..... Signal varies too much and too little and signal out of range `8` ..... Hardware error	`Long (I32)`

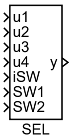
Parameters

`nb`	Number of samples which are not included in the validity assessment of the signal `u` after initialization of the block ⊙10	`Long (I32)`
`nc`	Number of samples for invariability testing (the `F2` condition) ⊙10	`Long (I32)`
`nbits`	Number of A/D converter bits ⊙12	`Long (I32)`
`nr`	Number of samples for variability testing (the `F3` condition) ⊙10	`Long (I32)`
`prate`	Maximum allowed percentage change of the input `u` within the last `nr` samples (with respect to the overall range of the input signal `vmax` − `vmin`) ⊙10.0	`Double (F64)`
`nv`	Number of samples for out-of-range testing (the `F4` condition) ⊙1	`Long (I32)`
`vmin`	Lower limit for the input signal `u` ⊙-1.0	`Double (F64)`
`vmax`	Upper limit for the input signal `u` ⊙1.0	`Double (F64)`

SEL – Selector switch for analog signals

Block Symbol

Licence: STANDARD



Function Description

The SEL block is obsolete, replace it by the SELQUAD block. Note the difference in binary selector signals SWn.

The SEL block selects one of the four input signals u1, u2, u3 and u4 and copies it to the output signal y. The selection is based on the iSW input or the binary inputs SW1 and SW2. These two modes are distinguished by the BINF binary flag. The signal is selected according to the following table:

iSW	SW1	SW2	y
0	off	off	u1
1	off	on	u2
2	on	off	u3
3	on	on	u4

Inputs

u1	First analog input of the block	Double (F64)
u2	Second analog input of the block	Double (F64)
u3	Third analog input of the block	Double (F64)
u4	Fourth analog input of the block	Double (F64)
iSW	Active signal selector, active when BINF = off	Long (I32)
SW1	Binary signal selector, active when BINF = on	Bool
SW2	Binary signal selector, active when BINF = on	Bool

Output

y	The selected signal	Double (F64)

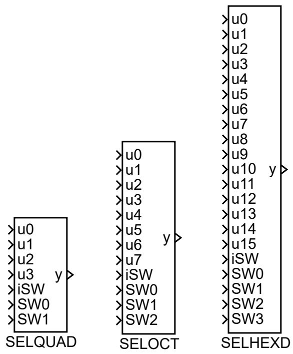
Parameter

BINF	Enable the binary selectors off ... Disabled (analog selector) on .... Enabled (binary selectors)	Bool

SELQUAD, SELOCT, SELHEXD – Selector switch for analog signals

Block Symbols

Licence: STANDARD



Function Description

The SELQUAD, SELOCT and SELHEX blocks select one of the input signals and copy it to the output signal y. The selection of the active signal u0. . . u15 is based on the iSW input or the binary inputs SW0. . . SW3. These two modes are distinguished by the BINF binary flag. The signal is selected according to the following table:

iSW	SW0	SW1	SW2	SW3	y
0	off	off	off	off	u0
1	on	off	off	off	u1
2	off	on	off	off	u2
3	on	on	off	off	u3
4	off	off	on	off	u4
5	on	off	on	off	u5
6	off	on	on	off	u6
7	on	on	on	off	u7
8	off	off	off	on	u8
9	on	off	off	on	u9
10	off	on	off	on	u10
11	on	on	off	on	u11
12	off	off	on	on	u12
13	on	off	on	on	u13
14	off	on	on	on	u14
15	on	on	on	on	u15

Please note that the only difference among the blocks is the number of inputs.

Inputs

`u0..15`	Analog inputs of the block	`Double (F64)`
`iSW`	Active signal selector	`Long (I32)`
`SW0..3`	Binary signal selectors	`Bool`

Output

`y`	The selected input signal	`Double (F64)`

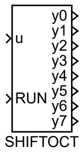
Parameter

`BINF`	Enable the binary selectors `off` ... Disabled (analog selector) `on` .... Enabled (binary selectors)	`Bool`

SHIFTOCT – Data shift register

Block Symbol

Licence: STANDARD



Function Description

The SHIFTOCT block works as a shift register with eight outputs of arbitrary data type.

If the RUN input is active, the following assignment is performed with each algorithm tick:

$$\begin{aligned} \mathtt{y}_i &= \mathtt{y}_{i-1},\ i = 1..7 \\ \mathtt{y0} &= \mathtt{u} \end{aligned}$$

Thus the value on each output y0 to y6 is shifted to the following output and the value on input u is assigned to output y0.

The block works with any data type of signal connected to the input u. Data type has to be specified by the vtype parameter. Outputs y0 to y8 then have the same data type.

If you need a triggered shift register, place the EDGE_ block in front of the RUN input.

Inputs

u	Data input of the register	Any
RUN	Enables outputs shift	Bool

Outputs

y0	First output of the block	Any
y1	Second output of the block	Any
y2	Third output of the block	Any
y3	Fourth output of the block	Any
y4	Fifth output of the block	Any
y5	Sixth output of the block	Any
y6	Seventh output of the block	Any
y7	Eighth output of the block	Any

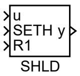
Parameters

`vtype`	Output data type `1` ..... Bool `2` ..... Byte (U8) `3` ..... Short (I16) `4` ..... Long (I32) `5` ..... Word (U16) `6` ..... DWord (U32) `7` ..... Float (F32) `8` ..... Double (F64) `--` .... `10` .... Large (I64)	⊙8 `Long (I32)`

SHLD – Sample and hold

Block Symbol

Licence: STANDARD



Function Description

The SHLD block is intended for holding the value of the input signal. It processes the input signal according to the mode parameter.

In *Triggered sampling* mode the block sets the output signal y to the value of the input signal u when rising edge (off→on) occurs at the SETH input. The output is held constant unless a new rising edge occurs at the SETH input.

If *Hold last value* mode is selected, the output signal y is set to the last value of the input signal u before the rising edge at the SETH input occured. It is kept constant as long as SETH = on. For SETH = off the input signal u is simply copied to the output y.

In *Hold current value* mode the u input is sampled right when the rising edge (off→on) occurs at the SETH input. It is kept constant as long as SETH = on. For SETH = off the input signal u is simply copied to the output y.

The binary input R1 sets the output y to the value y0, it overpowers the SETH input signal.

See also the PARR block, which can be used for storing a numeric value as well.

Inputs

u	Analog input of the block	Double (F64)
SETH	Trigger for the set and hold operation	Bool
R1	Block reset, R1 = on → y = y0	Bool

Output

y	Analog output of the block	Double (F64)

Parameter

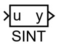
y0	Initial output value		Double (F64)
mode	Sampling mode	⊙3	Long (I32)

1 ..... Triggered sampling
2 ..... Hold last value
3 ..... Hold current value

SINT – Simple integrator

Block Symbol

Licence: STANDARD



Function Description

The SINT block implements a discrete integrator described by the following difference equation

$$y_k = y_{k-1} + \frac{T_S}{2T_i}(u_k + u_{k-1}),$$

where T_S is the block execution period and T_i is the integral time constant. If $T_i = 0$, the block realize summation by following equation

$$y_k = y_{k-1} + u_k.$$

If $T_i < 0$, the block behaviour is undefined. If y_k falls out of the saturation limits **ymin** and **ymax**, the output and state of the block are appropriately modified.

For more complex tasks, consider using the INTE block, which provides extended functionality.

Input

u	Analog input of the block	Double (F64)

Output

y	Analog output of the block	Double (F64)

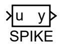
Parameters

ti	Integral time constant T_i	⊙1.0	Double (F64)
y0	Initial output value		Double (F64)
ymax	Upper limit of the output signal	⊙1.0	Double (F64)
ymin	Lower limit of the output signal	⊙-1.0	Double (F64)

SPIKE – Spike filter

Block Symbol

Licence: ADVANCED



Function Description

The SPIKE block implements a nonlinear filter for suppressing isolated peaks (pulses) in the input signal u. One cycle of the SPIKE filter performs the following transformation $(u, y) \rightarrow y$:

```
delta := y - u;
if abs(delta) < gap
  then
     begin
        y := u;
        gap := gap/q;
        ifgap < mingap then gap:= mingap;
     end
  else
     begin
       if delta < 0
          then y := y + gap
          else y := y - gap;
       gap := gap * q;
     end
```

where mingap and q are the block parameters.

The signal passes through the filter unaffected for sufficiently large mingap parameter, which defines the minimal size of the tolerance window. By lowering this parameter it is possible to find an appropriate value, which leads to suppression of the undesirable peaks but leaves the input signal intact otherwise. The recommended value is 1 % of the overall input signal range. The q parameter determines the adaptation speed of the tolerance window.

Input

u	Input signal to be filtered	Double (F64)

Output

y	Filtered output signal	Double (F64)

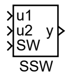
Parameters

`mingap`	Minimum size of the tolerance window	⊙0.01	`Double (F64)`
`q`	Tolerance window adaptation speed	↓1.0 ⊙2.0	`Double (F64)`

SSW – Simple switch

Block Symbol

Licence: STANDARD



Function Description

The SSW block selects one of two input signals u1 and u2 with respect to the binary input SW. The selected input is copied to the output y. If SW = off (SW = on), then the selected signal is u1 (u2).

Inputs

u1	First input of the block	Any
u2	Second input of the block	Any
SW	Signal selector off ... The u1 signal is selected, y = u1 on .... The u2 signal is selected, y = u2	Bool

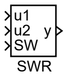
Output

y	Output of the block	Any

SWR – Selector with ramp

Block Symbol

Licence: STANDARD



Function Description

The SWR block selects one of two input signals u1 and u2 with respect to the binary input SW. The selected input is copied to the output y. If SW = off (SW = on), then the selected signal is u1 (u2). The output signal is not set immediately to the value of the selected input signal but tracks the selected input with given rate constraint (i.e. it follows a ramp). This rate constraint is configured independently for each input u1, u2 and is defined by time constants t1 and t2. As soon as the output reaches the level of the selected input signal, the rate limiter is disabled and remains inactive until the next signal switching.

Inputs

u1	First analog input of the block	Double (F64)
u2	Second analog input of the block	Double (F64)
SW	Signal selector off . . . The u1 signal is selected on . . . . The u2 signal is selected	Bool

Parameters

t1	Rate limiter time constant for switching from u2 to u1	⊙1.0	Double (F64)
t2	Rate limiter time constant for switching from u1 to u2	⊙1.0	Double (F64)
y0	Initial output value to start the tracking from (before the first switching of signals occurs)		Double (F64)

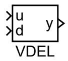
Output

y	Analog output of the block	Double (F64)

VDEL – Variable time delay

Block Symbol

Licence: STANDARD



Function Description

The VDEL block delays the input signal u by the time defined by the input signal d. More precisely, the delay is given by rounding the input signal d to the nearest integer multiple of the block execution period ($n \cdot T_S$). A substitute value y0 is used until n previous samples are available after the block initialization.

Inputs

u	Analog input of the block	Double (F64)
d	Time delay [s]	Double (F64)

Output

y	Delayed input signal	Double (F64)

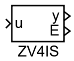
Parameter

y0	Initial/substitute output value	Double (F64)
nmax	Size of delay buffer (number of samples) for the time delay d. Used for internal memory allocation. ↓10 ↑10000000 ⊙1000	Long (I32)

`ZV4IS` – Zero vibration input shaper

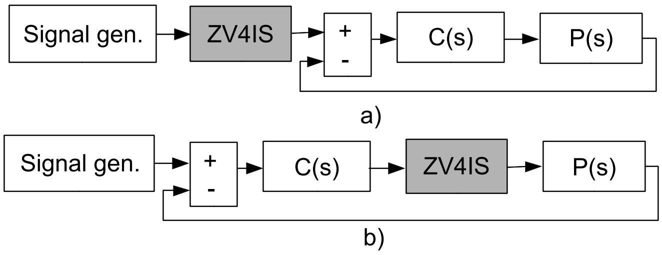
Block Symbol

Licence: ADVANCED



Function Description

The function block `ZV4IS` implements a band-stop frequency filter. The main field of application is in motion control of flexible systems where the low stiffness of mechanical construction causes an excitation of residual vibrations which can be observed in form of mechanical oscillations. Such vibration can cause significant deterioration of quality of control or even instability of control loops. They often lead to increased wear of mechanical components. Generally, the filter can be used in arbitrary application for a purpose of control of an oscillatory system or in signal processing for selective suppression of particular frequency.



The input shaping filter can be used in two different ways. By using an *open loop connection*, the input reference signal for an feedback loop coming from human operator or higher level of control structure is properly shaped in order to attenuate any unwanted oscillations. The internal dynamics of the filter does not influence a behaviour of the inferior loop. The only condition is correct tuning of feedback compensator $C(s)$, which has to work in linear mode. Otherwise, the frequency spectrum of the manipulating variable gets corrupted and unwanted oscillations can still be excited in a plant $P(s)$. The main disadvantage is passive vibration damping which works only in reference signal path. In case of any external disturbances acting on the plant, the vibrations may still arise. The second possible way of use is *feedback connection*. The input shaper is placed on the output side of feedback compensator $C(s)$ and modifies the manipulating variable acting on the plant. An additional dynamics of the filter is introduced and the compensator $C(s)$ needs to be properly tuned.

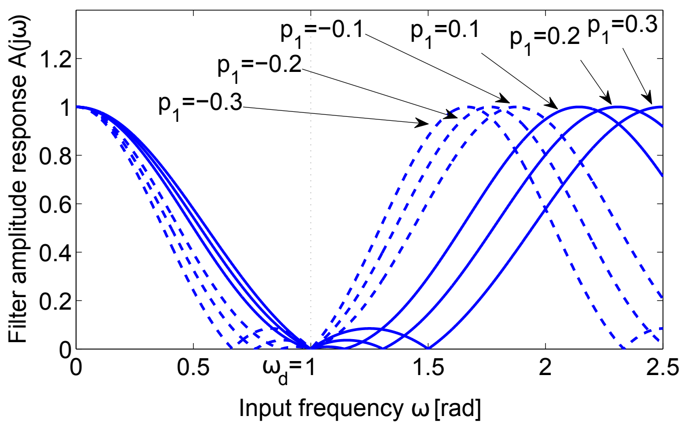
The algorithm of input shaper can be described in time domain

$$\mathtt{y}(t) = A_1\mathtt{u}(t-t_1) + A_2\mathtt{u}(t-t_2) + A_3\mathtt{u}(t-t_3) + A_4\mathtt{u}(t-t_4)$$

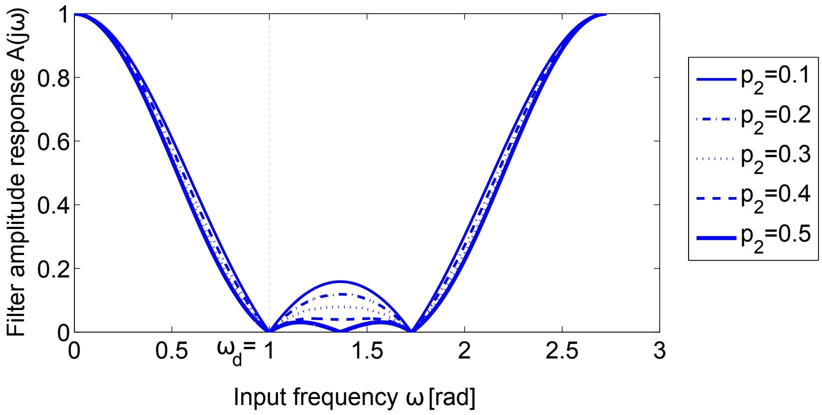
Thus, the filter has a structure of sum of weighted time delays of an input signal. The gains $A_1..A_4$ and time delay values $t_1..t_4$ depend on a choice of filter type, natural frequency and damping of controlled oscillatory mode of the system. The main advantage of this structure compared to commonly used notch filters is finite impulse response (which is especially important in motion control applications), warranted stability and monotone step response of the filter and generally lower dynamic delay introduced into a signal path.

For correct function of the filter, natural frequency `omega` and damping `xi` of the oscillatory mode need to be set. The parameter `ipar` sets a filter type. For `ipar` = `1`, one of ten basic filter types chosen by `istype` is used. Particular basic filters differ in shape and width of stop band in frequency domain. In case of precise knowledge of natural frequency and damping, the ZV (Zero Vibration) or ZVD filters can be used, because their response to input signal is faster compared to the other filters. In case of large uncertainty in system/signal model, robust UEI (Extra Insensitive) or UTHEI filters are good choice. Their advantage is wider stopband at the cost of slower response. The number on the end of the name has the meaning of maximum allowed level of excited vibrations for the given `omega` and `xi` (one, two or five percent).

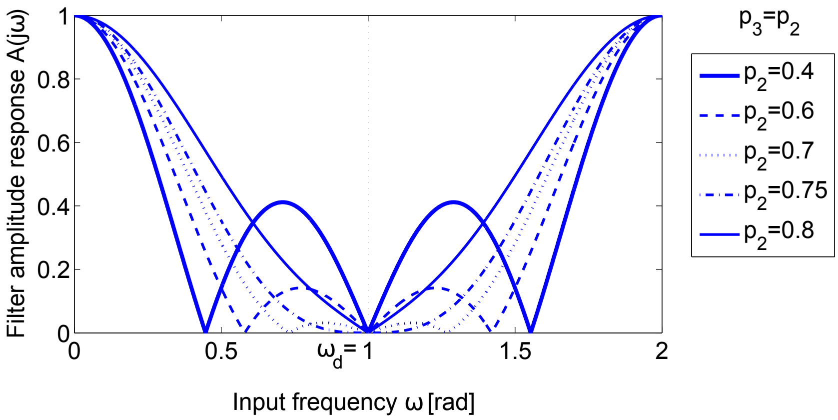
For precise tuning of the filter, complete parameterization `ipar` = `2` can be selected. For this choice, three parameters `p_alpha`,`p_a2` and `p_a3` which affect the shape of the filter frequency response can freely be assigned. These parameters can be used for finding of optimal compromise between robustness of the filter and introduced dynamical delay.



The asymmetry parameter `p_alpha` determines relative location of the stopband of filter frequency response with respect to chosen natural frequency. Positive values mean a shift to higher frequency range, negative values to lower frequency range, zero value leads to symmetrical shape of the characteristic (see the figure above). The parameter `p_alpha` also affects the overall filter length, thus the overall delay introduced into a signal path. Lower values result in slower filters and higher delay. Asymmetric filters can be used in cases where a lower or higher bound of the uncertainty in natural frequency parameter is known.



Insensitivity parameter p_a2 determines the width and attenuation level of the filter stopband. Higher values result in wider stopband and higher attenuation. For most applications, the value p_a2 $= 0.5$ is recommended for highest achievable robustness with respect to modeling errors.



The additional parameter p_a3 needs to be chosen for symmetrical filters (p_alpha $=$ 0). A rule for the most of the practical applications is to chose *equal values* p_a2 $=$ p_a3 from interval $< 0, 0.75 >$. Overall filter length is constant for this choice and only the shape of filter stopband is affected. Lower values lead to robust shapers with wide stopband and frequency response shape similar to standard THEI (Two-hump extra insensitive) filters. Higher values lead to narrow stopband and synchronous drop of two stopband peaks. The choice p_a2 $=$ p_a3 $= 0.75$ results in standard ZVDD filter with maximally flat and symmetric stopband shape. The proposed scheme can be used for systematic tuning of the filter.

Input

u	Input signal to be filtered	Double (F64)

Outputs

`y`	Filtered output signal		Double (F64)
`E`	Error flag `off` ... No error `on` .... An error occurred		Bool

Parameters

`omega`	Natural frequency	⊙1.0	Double (F64)
`xi`	Relative damping coefficient		Double (F64)
`ipar`	Specification `1` ..... Basic types of IS `2` ..... Complete parametrization	⊙1	Long (I32)
`istype`	Type `1` ..... ZV `2` ..... ZVD `3` ..... ZVDD `4` ..... MISZV `5` ..... UEI1 `6` ..... UEI2 `7` ..... UEI5 `8` ..... UTHEI1 `9` ..... UTHEI2 `10` UTHEI5	⊙2	Long (I32)
`p_alpha`	Shaper duration/assymetry parameter	⊙0.2	Double (F64)
`p_a2`	Insensitivity parameter	⊙0.5	Double (F64)
`p_a3`	Additional parameter (only for `p_alpha` = 0)	⊙0.5	Double (F64)
`nmax`	Size of data buffer (number of samples). Used for internal memory allocation.	↓10 ↑10000000 ⊙1000	Long (I32)

Chapter 6

GEN – Signal generators

Contents

ANLS – Controlled generator of piecewise linear function

Block Symbol

Licence: STANDARD

Function Description

The ANLS block generates a piecewise linear function of time given by nodes t1,y1; t2,y2; t3,y3; t4,y4. The initial value of output y is defined by the y0 parameter. The generation of the function starts when a rising edge occurs at the RUN input (and the internal timer is set to 0). The output y is then given by

$$y = y_i + \frac{y_{i+1} - y_i}{t_{i+1} - t_i}(t - t_i)$$

within the time intervals $\langle t_i, t_{i+1} \rangle, i = 0, \ldots, 3, t_0 = 0$.

To generate a step change in the output signal, it is necessary to to define two nodes in the same time instant (i.e. $t_i = t_{i+1}$). The generation ends when time t4 is reached or when time t_i is reached and the following node precedes the active one (i.e. $t_{i+1} < t_i$). The output holds its final value afterwards. But for the RPT parameter set to on, instead of holding the final value, the block returns to its initial state y0, the internal block timer is set to 0 and the sequence is generated repeatedly. This can be used to generate square or sawtooth functions. The generation can also be prematurely terminated by the RUN input signal set to off. In that case the block returns to its initial state y0, the internal block timer is set to 0 and is $= 0$ becomes the active time interval.

Input

RUN	Enable execution, run the analog sequence generation	Bool

Outputs

y	Analog output of the block	Double (F64)
is	Index of the active time interval	Long (I32)

Parameters

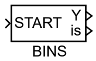
y0	Initial output value		Double (F64)
t1	Node 1 time	⊙1.0	Double (F64)
y1	Node 1 value		Double (F64)
t2	Node 2 time	⊙1.0	Double (F64)
y2	Node 2 value	⊙1.0	Double (F64)

t3	Node 3 time	⊙2.0	Double (F64)
y3	Node 3 value	⊙1.0	Double (F64)
t4	Node 4 time	⊙2.0	Double (F64)
y4	Node 4 value		Double (F64)
RPT	Repeating sequence off . . . Disabled on . . . . Enabled		Bool

BINS – Controlled binary sequence generator

Block Symbol

Licence: STANDARD



Function Description

The BINS block generates a binary sequence at the Y output, similarly to the BIS block. The binary sequence is given by the block parameters.

- The initial value of the output is given by the Y0 parameter.
- Whenever a rising edge (off→on) occurs at the START input (even when a binary sequence is being generated), the internal timer of the block is set to 0 and started.
- Whenever a rising edge occurs at the START input, the output Y is set to Y0.
- The output value is inverted at time instants t1, t2, ..., t8 (off→on, on→off).
- For RPT = off, the last switching of the output occurs at time t_i, where $t_{i+1} = 0$ and the output then holds its value until another rising edge (off→on) occurs at the START input.
- For RPT = on, instead of switching the output for the last time, the block returns to its initial state, the Y output is set to Y0, the internal block timer is set to 0 and started. As a result, the binary sequence is generated repeatedly.

On the contrary to the BIS block the changes in parameters t1...t8 are accepted only when a rising edge occurs at the START input.

The switching times are internally rounded to the nearest integer multiple of the execution period, which may result in e.g. disappearing of very thin pulses ($< T_S/2$) or melting successive thin pulses into one thick pulse. Therefore it is strongly recommended to use integer multiples of the execution period as the switching times.

Input

START	Starting signal (rising edge)	Bool

Outputs

Y	Logical output of the block	Bool
is	Index of the active time interval	Long (I32)

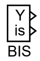
Parameters

`Y0`	Initial output value `off` ... Disabled/false `on` .... Enabled/true		`Bool`
`t1`	Switching time 1 [s]	↓0.0 ⊙1.0	`Double (F64)`
`t2`	Switching time 2 [s]	↓0.0 ⊙2.0	`Double (F64)`
`t3`	Switching time 3 [s]	↓0.0 ⊙3.0	`Double (F64)`
`t4`	Switching time 4 [s]	↓0.0 ⊙4.0	`Double (F64)`
`t5`	Switching time 5 [s]	↓0.0 ⊙5.0	`Double (F64)`
`t6`	Switching time 6 [s]	↓0.0 ⊙6.0	`Double (F64)`
`t7`	Switching time 7 [s]	↓0.0 ⊙7.0	`Double (F64)`
`t8`	Switching time 8 [s]	↓0.0 ⊙8.0	`Double (F64)`
`RPT`	Repeating sequence `off` ... Disabled `on` .... Enabled		`Bool`

BIS – Binary sequence generator

Block Symbol

Licence: STANDARD



Function Description

The BIS block generates a binary sequence at the Y output. The sequence is given by the block parameters.

- The initial value of the output is given by the Y0 parameter.
- The internal timer of the block is set to 0 when the block initializes.
- The internal timer of the block is immediately started when the block initializes.
- The output value is inverted at time instants t1, t2, ..., t8 (off→on, on→off).
- For RPT = off, the last switching of the output occurs at time t_i, where $t_{i+1} = 0$ and the output then holds its value indefinitely.
- For RPT = on, instead of switching the output for the last time, the block returns to its initial state, the Y output is set to Y0, the internal block timer is set to 0 and started. As a result, the binary sequence is generated repeatedly.

All the parameters t1...t8 can be changed in runtime and all changes are immediately accepted.

The switching times are internally rounded to the nearest integer multiple of the execution period, which may result in e.g. disappearing of very thin pulses ($< T_S/2$) or melting successive thin pulses into one thick pulse. Therefore it is strongly recommended to use integer multiples of the execution period as the switching times.

See also the BINS block, which allows for triggering the sequence by external signal.

Outputs

Y	Logical output of the block	Bool
is	Index of the active time interval	Long (I32)

Parameters

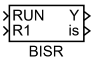
Y0	Initial output value off ... Disabled/false on .... Enabled/true		Bool
t1	Switching time 1 [s]	↓0.0 ⊙1.0	Double (F64)

t2	Switching time 2 [s]	↓0.0 ⊙2.0	Double (F64)
t3	Switching time 3 [s]	↓0.0 ⊙3.0	Double (F64)
t4	Switching time 4 [s]	↓0.0 ⊙4.0	Double (F64)
t5	Switching time 5 [s]	↓0.0 ⊙5.0	Double (F64)
t6	Switching time 6 [s]	↓0.0 ⊙6.0	Double (F64)
t7	Switching time 7 [s]	↓0.0 ⊙7.0	Double (F64)
t8	Switching time 8 [s]	↓0.0 ⊙8.0	Double (F64)
RPT	Repeating sequence off ... Disabled on .... Enabled		Bool

BISR – Binary sequence generator with reset

Block Symbol

Licence: STANDARD



Function Description

The BISR block generates a binary sequence at the Y output. The RUN input must be set to on for the whole duration of the sequence. When RUN is off, the sequence is paused and so is the internal timer.

The binary sequence is given by the block parameters. The initial value of the output is given by the Y0 parameter. The output value Y is inverted (off→on, on→off) at time instants t1, t2, ..., t8. The ADDT parameter defines whether the t_i instants are relative to the first rising edge at the RUN input or relative to the last switching of the Y output.

If there is less than 8 edges in the desired binary sequence, set any of the t_i parameters to zero and the remaining ones will be ignored.

Whenever a rising edge occurs at the R1 input, the output Y is set to Y0 and the internal timer is reset. The R1 input overpowers the RUN input.

For RPT = off, the last switching of the output occurs at time t_i, where $t_{i+1} = 0$ and the output then holds its value until another rising edge (off→on) occurs at the START input. For RPT = on, instead of switching the output for the last time, the block returns to its initial state, the Y output is set to Y0, the internal block timer is set to 0 and started. As a result, the binary sequence is generated repeatedly.

The BISR block is an extended version of the BINS block.

Input

RUN	Enable execution	Bool

Outputs

Y	Logical output of the block	Bool
is	Index of the active time interval	Long (I32)

Parameters

Y0	Initial output value off ... Disabled/false on .... Enabled/true	Bool
ADDT	Additive timing off ... Absolute timing (sequence as a whole) on .... Additive timing (segment by segment)	Bool

RPT	Repeating sequence off ... Disabled on .... Enabled		Bool
t1	Switching time 1 [s]	↓0.0 ⊙1.0	Double (F64)
t2	Switching time 2 [s]	↓0.0 ⊙2.0	Double (F64)
t3	Switching time 3 [s]	↓0.0 ⊙3.0	Double (F64)
t4	Switching time 4 [s]	↓0.0 ⊙4.0	Double (F64)
t5	Switching time 5 [s]	↓0.0 ⊙5.0	Double (F64)
t6	Switching time 6 [s]	↓0.0 ⊙6.0	Double (F64)
t7	Switching time 7 [s]	↓0.0 ⊙7.0	Double (F64)
t8	Switching time 8 [s]	↓0.0 ⊙8.0	Double (F64)

MP – Manual pulse generator

Block Symbol

Licence: STANDARD

Function Description

The MP block generates a pulse of width **pwidth** when a rising edge occurs at the BSTATE parameter (off→on). The algorithm immediately reverts the BSTATE parameter back to off (BSTATE stands for a shortly pressed button). If repetition is enabled (RPTF = on), it is possible to extend the pulse by repeated setting the BSTATE parameter to on. When repetition is disabled, the parameter BSTATE is not taken into account during generation of a pulse, i.e. the output pulses have always the specified width of **pwidth**.

The MP block reacts only to rising edge of the BSTATE parameter, therefore it cannot be used for generating a pulse immediately at the start of the REXYGEN system executive. Use the BIS block for such a purpose.

Output

Y	Logical output of the block		Bool

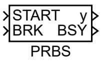
Parameters

pwidth	Pulse width [s] (0 means one pulse)	⊙1.0	Double (F64)
BSTATE	Output pulse activation off ... No action on .... Generate output pulse		Bool
RPTF	Allow pulse extension off ... Disabled on .... Enabled		Bool

PRBS – Pseudo-random binary sequence generator

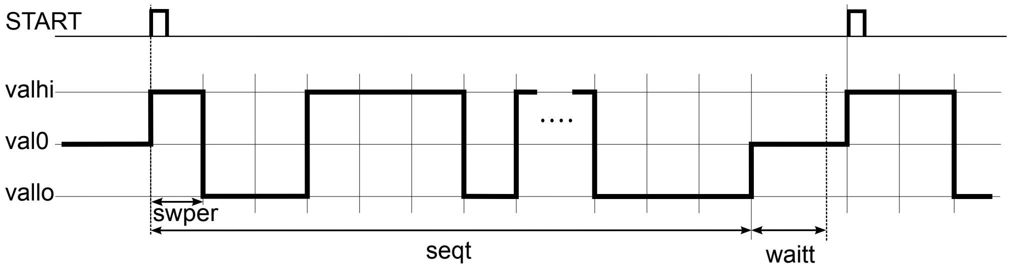
Block Symbol

Licence: STANDARD



Function Description

The PRBS block generates a pseudo-random binary sequence. The figure below displays how the sequence is generated.



The initial and final values of the sequence are val0. The sequence starts from this value when rising edge occurs at the START input (off→on), the output y is immediately switched to the valhi value. The generator then switches the output to the other limit value with the period of swper seconds and the probability of switching swprob. After seqt seconds the output is set back to val0. A waitt-second period follows to allow the settling of the controlled system response. Only then it is possible to start a new sequence. It is possible to terminate the sequence prematurely by the BRK = on input when necessary.

Inputs

START	Starting signal (rising edge)		Bool
BRK	Termination signal		Bool

Outputs

y	Generated pseudo-random binary sequence		Double (F64)
BSY	Busy flag		Bool

Parameters

val0	Initial and final value		Double (F64)
valhi	Upper level of the y output	⊙1.0	Double (F64)
vallo	Lower level of the y output	⊙-1.0	Double (F64)

`swper`	Period of random output switching [s]	⊙1.0	`Double (F64)`
`swprob`	Probability of switching	↓0.0 ↑1.0 ⊙0.2	`Double (F64)`
`seqt`	Length of the sequence [s]	⊙10.0	`Double (F64)`
`waitt`	Settling period [s]	⊙2.0	`Double (F64)`

SG, SGI – Signal generators

Block Symbols

Licence: STANDARD

Function Description

The SG and SGI blocks generate periodic signals of chosen type (isig parameter): sine wave, square, sawtooth and white noise with uniform distribution. The amplitude and frequency of the output signal y are given by the amp and freq parameter respectively. The output y can have a phase shift of phase $\in (0, 2\pi)$ in the deterministic signals (isig $\in \{1, 2, 3\}$).

The SGI block allows synchronization of multiple generators using the RUN and SYN inputs. The RUN parameter must be set to on to enable the generator, the SYN input synchronizes the generators during the output signal generation.

Inputs

RUN	Enable execution, run the binary sequence generation	Bool
SYN	Synchronization signal	Bool

Output

y	Analog output of the block	Double (F64)

Parameters

isig	Generated signal type 1 ..... Sine wave 2 ..... Symmetrical rectangular signal 3 ..... Sawtooth signal 4 ..... White noise with uniform distribution 5 ..... Triangular signal	⊙1	Long (I32)
amp	Amplitude of the generated signal	⊙1.0	Double (F64)
freq	Frequency of the generated signal	⊙1.0	Double (F64)
phase	Phase shift of the generated signal		Double (F64)
offset	Value added to the generated signal	⊙1.0	Double (F64)
ifrunit	Frequency units 1 ..... Hz 2 ..... rad/s	⊙1	Long (I32)

`iphunit`	Phase shift units `1` ..... degrees `2` ..... radians	⊙1	`Long (I32)`

Chapter 7

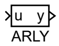
REG – Function blocks for control

Contents

ARLY – Advance relay

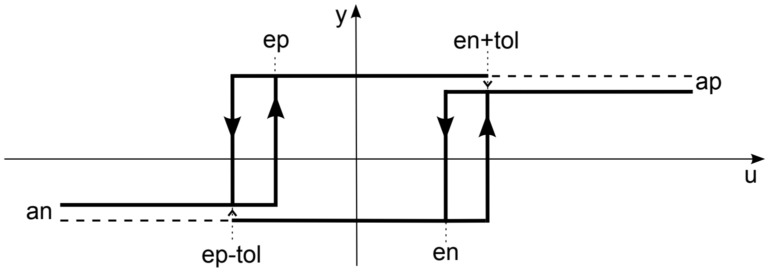
Block Symbol

Licence: STANDARD



Function Description

The ARLY block is a modification of the RLY block, which allows lowering the amplitude of steady state oscillations in relay feedback control loops. The block transforms the input signal u to the output signal y according to the diagram below.



Input

u	Analog input of the block	Double (F64)

Output

y	Analog output of the block	Double (F64)

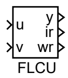
Parameters

ep	Value for switching the output to the "On" state	⊙-1.0	Double (F64)
en	Value for switching the output to the "Off" state	⊙1.0	Double (F64)
tol	Tolerance limit for the superposed noise of the input signal u	↓0.0 ⊙0.5	Double (F64)
ap	Value of the y output in the "On" state	⊙1.0	Double (F64)
an	Value of the y output in the "Off" state	⊙-1.0	Double (F64)
y0	Initial output value		Double (F64)

FLCU – Fuzzy logic controller unit

Block Symbol

Licence: ADVANCED



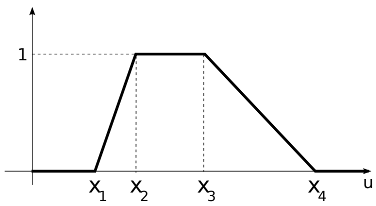
Function Description

The FLCU block implements a simple fuzzy logic controller with two inputs and one output. Introduction to fuzzy logic problems can be found in [3].

The output is defined by trapezoidal membership functions of linguistic terms of the u and v inputs, impulse membership functions of linguistic terms of the y output and inference rules in the form

$$\text{If } (\texttt{u} \text{ is } U_i) \text{ AND } (\texttt{v} \text{ is } V_j), \text{ then } (\texttt{y} \text{ is } Y_k),$$

where $U_i, i = 1, \ldots, \texttt{nu}$ are the linguistic terms of the u input; $V_j, j = 1, \ldots, \texttt{nv}$ are the linguistic terms of the v input and $Y_k, k = 1, \ldots, \texttt{ny}$ are the linguistic terms of the y output. Trapezoidal (triangular) membership functions of the u and v inputs are defined by four numbers as depicted below.



Not all numbers $x_1, \ldots, x_4$ are mutually different in triangular functions. The matrices of membership functions of the u and v input are composed of rows $[x_1, x_2, x_3, x_4]$. The dimensions of matrices mfu and mfv are $(\texttt{nu} \times 4)$ and $(\texttt{nv} \times 4)$ respectively.

The impulse 1st order membership functions of the y output are defined by the triplet

$$\texttt{y}_k,\ a_k,\ b_k,$$

where $\texttt{y}_k$ is the value assigned to the linguistic term $Y_k, k = 1, \ldots, \texttt{ny}$ in the case of $a_k = b_k = 0$. If $a_k \neq 0$ and $b_k \neq 0$, then the term Y_k is assigned the value of $\texttt{y}_k + a_k\texttt{u} + b_k\texttt{v}$. The output membership function matrix sty has a dimension of $(\texttt{ny} \times 3)$ and contains the rows $[\texttt{y}_k, a_k, b_k], k = 1, \ldots, \texttt{ny}$.

The set of inference rules is also a matrix whose rows are $[i_l, j_l, k_l, w_l], l = 1, \ldots, \texttt{nr}$, where i_l, j_l and k_l defines a particular linguistic term of the u and v inputs and y output respectively. The number w_l defines the weight of the rule in percents $w_l \in \{0, 1, \ldots, 100\}$. It is possible to suppress or emphasize a particular inference rule if necessary.

Inputs

u	First analog input of the block	Double (F64)
v	Second analog input of the block	Double (F64)

Outputs

y	Analog output of the block	Double (F64)
ir	Dominant rule	Long (I32)
wr	Degree of truth of the dominant rule	Double (F64)

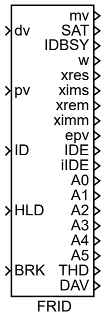
Parameters

umax	Upper limit of the u input	⊙1.0	Double (F64)
umin	Lower limit of the u input	⊙-1.0	Double (F64)
vmax	Upper limit of the v input	⊙1.0	Double (F64)
vmin	Lower limit of the v input	⊙-1.0	Double (F64)
nmax	Number of reserved (allocated) membership functions (for each inputs and output)	↓4 ↑10000 ⊙10	Long (I32)
mfu	Matrix of membership functions of the input u	⊙[-1 -1 -1 0; -1 0 0 1; 0 1 1 1]	Double (F64)
mfv	Matrix of membership functions of the input v	⊙[-1 -1 -1 0; -1 0 0 1; 0 1 1 1]	Double (F64)
sty	Matrix of membership functions of the output y	⊙[-1 0 0; 0 0 0; 1 0 0]	Double (F64)
rls	Matrix of inference rules	⊙[1 2 3 100; 1 1 1 100; 1 0 3 100]	Byte (U8)

FRID – * **Frequency response identification**

Block Symbol

Licence: ADVANCED



Function Description

The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

Inputs

dv	Feedforward control variable	Double (F64)
pv	Process variable	Double (F64)
ID	Start the tuning experiment	Bool
HLD	Hold	Bool
BRK	Stop the tuning experiment	Bool

Parameters

ubias	Static component of the exciting signal		Double (F64)
uamp	Amplitude of the exciting signal	⊙1.0	Double (F64)
wb	Frequency interval lower limit [rad/s]	⊙1.0	Double (F64)
wf	Frequency interval higher limit [rad/s]	⊙10.0	Double (F64)
isweep	Frequency sweeping mode 1 ..... Logarithmic 2 ..... Linear	⊙1	Long (I32)
cp	Sweeping Rate	⊙0.995	Double (F64)
iavg	Number of values for averaging	⊙10	Long (I32)
obw	Observer bandwith 1 ..... LOW 2 ..... NORMAL 3 ..... HIGH	⊙2	Long (I32)
stime	Settling period [s]	⊙10.0	Double (F64)

umax	Maximum generator amplitude	⊙1.0	Double (F64)
thdmin	Minimum demanded THD treshold	⊙0.1	Double (F64)
adapt_rc	Maximum rate of amplitude variation	⊙0.001	Double (F64)
pv_max	Maximum desired process value	⊙1.0	Double (F64)
pv_sat	Maximum allowed process value	⊙2.0	Double (F64)
ADAPT_EN	Enable automatic amplitude adaptation	⊙on	Bool
immode	Mesurement mode 1 ..... Manual specification of frequency points 2 ..... Linear series of nmw points in the interval <wb;wf> 3 ..... Logarithmic series of nmw points in the interval <wb;wf> 4 ..... Automatic detection of important frequencies (N/A)	⊙1	Long (I32)
nwm	Number of frequency response point for automatic mode		Long (I32)
wm	Frequency measurement points for manual meas. mode [array of rad/s]	⊙[2.0 4.0 6.0 8.0]	Double (F64)

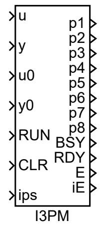
Outputs

mv	Manipulated variable (controller output)	Double (F64)
SAT	Saturation flag	Bool
IDBSY	Tuner busy flag	Bool
w	Actual frequency [rad/s]	Double (F64)
xres	real part of frequency response (sweeping)	Double (F64)
xims	imaginary part of frequency response (sweeping)	Double (F64)
xrem	real part of frequency response (measurement)	Double (F64)
ximm	imaginary part of frequency response (measurement)	Double (F64)
epv	Estimated process value	Double (F64)
IDE	Error indicator	Bool
iIDE	Error code	Long (I32)
A0	Estimated DC value	Double (F64)
A1	Estimated 1st harmonics amlitude	Double (F64)
A2	Estimated 2nd harmonics amlitude	Double (F64)
A3	Estimated 3rd harmonics amlitude	Double (F64)
A4	Estimated 4th harmonics amlitude	Double (F64)
A5	Estimated 5th harmonics amlitude	Double (F64)
THD	Total harmonic distorsion	Double (F64)
DAV	Data Valid	Bool

I3PM – Identification of a three parameter model

Block Symbol

Licence: ADVANCED



Function Description

The I3PM block is based on the generalized moment identification method. It provides a three parameter model of the system.

Inputs

u	Input of the identified system	Double (F64)
y	Output of the identified system	Double (F64)
u0	Input steady state	Double (F64)
y0	Output steady state	Double (F64)
RUN	Execute identification	Bool
CLR	Block reset	Bool
ips	Meaning of the output signals 0 ..... FOPDT model p1 ... gain p2 ... time delay p3 ... time constant 1 ..... moments of input and output p1 ... parameter $mu0$ p2 ... parameter $mu1$ p3 ... parameter $mu2$ p4 ... parameter $my0$ p5 ... parameter $my1$ p6 ... parameter $my2$ 2 ..... process moments p1 ... parameter $mp0$ p2 ... parameter $mp1$ p3 ... parameter $mp2$ 3 ..... characteristic numbers p1 ... parameter κ p2 ... parameter μ p3 ... parameter σ^2 p4 ... parameter σ	Long (I32)

Outputs

`p`i	Identified parameters with respect to `ips`, $i = 1, \ldots, 8$		`Double (F64)`
`BSY`	Busy flag		`Bool`
`RDY`	Ready flag		`Bool`
`E`	Error flag		`Bool`
`iE`	Error code `1` ..... Premature termination (`RUN` = `off`) `2` ..... `mu0` = 0 `3` ..... `mp0` = 0 `4` ..... $\sigma^2 < 0$		`Long (I32)`

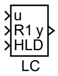
Parameters

`tident`	Duration of identification [s]	⊙`100.0`	`Double (F64)`
`irtype`	Controller type (control law) `1` ..... D `3` ..... ID `5` ..... PD `7` ..... PID `2` ..... I `4` ..... P `6` ..... PI	⊙`6`	`Long (I32)`
`ispeed`	Desired closed loop speed `1` ..... Slow closed loop `2` ..... Normal (middle fast) closed loop `3` ..... Fast closed loop	⊙`2`	`Long (I32)`

LC – Lead compensator

Block Symbol

Licence: STANDARD



Function Description

The LC block is a discrete simulator of derivative element

$$C(s) = \frac{\texttt{td} * s}{\frac{\texttt{td}}{\texttt{nd}} * s + 1},$$

where td is the derivative constant and nd determines the influence of parasite 1st order filter. It is recommended to use $2 \leq \texttt{nd} \leq 10$. If ISSF = on, then the state of the parasite filter is set to the steady value at the block initialization according to the input signal u.

The exact discretization at the sampling instants is used for discretization of the $C(s)$ transfer function.

Input

u	Analog input of the block	Double (F64)
R1	Block reset (same state as after init)	Bool
HLD	Hold block execution	Bool

Output

y	Analog output of the block	Double (F64)

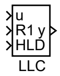
Parameters

td	Derivative time constant	⊙1.0	Double (F64)
nd	Derivative filtering parameter	⊙10.0	Double (F64)
ISSF	Steady state at start-up off ... Zero initial state on .... Initial steady state		Bool

LLC – Lead-lag compensator

Block Symbol

Licence: STANDARD



Function Description

The LLC block is a discrete simulator of integral-derivative element

$$C(s) = \frac{\mathtt{a} * \mathtt{tau} * s + 1}{\mathtt{tau} * s + 1},$$

where tau is the denominator time constant and the time constant of numerator is an a-multiple of tau ($\mathtt{a} * \mathtt{tau}$). If $\mathtt{ISSF} = \mathtt{on}$, then the state of the filter is set to the steady value at the block initialization according to the input signal u.

The exact discretization at the sampling instants is used for discretization of the $C(s)$ transfer function. The sampling period used for discretization is equivalent to the execution period of the LLC block.

Input

u	Analog input of the block	Double (F64)
R1	Block reset (same state as after init)	Bool
HLD	Hold block execution	Bool

Output

y	Analog output of the block	Double (F64)

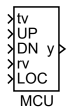
Parameters

tau	Time constant	⊙1.0	Double (F64)
a	Numerator time constant coefficient		Double (F64)
ISSF	Steady state at start-up off . . . Zero initial state on . . . . Initial steady state		Bool

MCU – Manual control unit

Block Symbol

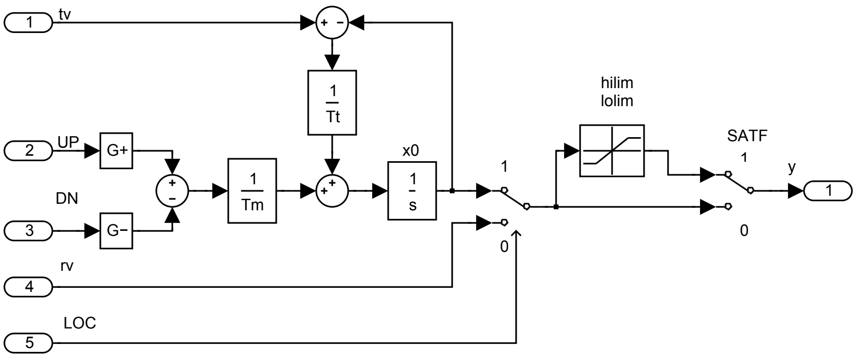
Licence: STANDARD



Function Description

The MCU block is suitable for manual setting of the numerical output value y, e.g. a setpoint. In the local mode (LOC = on) the value is set using the buttons UP and DN. The rate of increasing/decreasing of the output y from the initial value y0 is determined by the integration time constant tm and pushing time of the buttons. After elapsing ta seconds while a button is pushed, the rate is always multiplied by the factor q until the time tf is elapsed. Optionally, the output y range can be constrained (SATF = on) by saturation limits lolim and hilim. If none of the buttons is pushed (UP = off and DN = off), the output y tracks the input value tv. The tracking speed is controlled by the integration time constant tt.

In the remote mode (LOC = off), the input rv is optionally saturated (SATF = on) and copied to the output y. The detailed function of the block is depicted in the following diagram.



Inputs

tv	Tracking variable	Double (F64)
UP	The "up" signal	Bool
DN	The "down" signal	Bool
rv	Remote output value in the mode LOC = off	Double (F64)
LOC	Local or remote mode	Bool

Output

`y`	Analog output of the block		`Double (F64)`

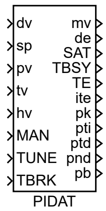
Parameters

`tt`	Tracking time constant of the input `tv`	⊙1.0	`Double (F64)`
`tm`	Initial value of integration time constant	⊙100.0	`Double (F64)`
`y0`	Initial output value		`Double (F64)`
`q`	Multiplication quotient	⊙5.0	`Double (F64)`
`ta`	Interval after which the rate is changed [s]	⊙4.0	`Double (F64)`
`tf`	Interval after which the rate changes no more [s]	⊙8.0	`Double (F64)`
`SATF`	Saturation flag `off` ... Signal not limited `on` .... Saturation limits active		`Bool`
`hilim`	Upper limit of the output signal	⊙1.0	`Double (F64)`
`lolim`	Lower limit of the output signal	⊙-1.0	`Double (F64)`

PIDAT – PID controller with relay autotuner

Block Symbol

Licence: AUTOTUNING



Function Description

The PIDAT block has the same control function as the PIDU block. Additionally it is equipped with the relay autotuning function.

In order to perform the autotuning experiment, it is necessary to drive the system to approximately steady state (at a suitable working point), choose the type of controller to be autotuned (PI or PID) and activate the TUNE input by setting it to on. The controlled process is regulated by special adaptive relay controller in the experiment which follows. One point of frequency response is estimated from the data measured during the experiment. Based on this information the controller parameters are computed. The amplitude of the relay controller (the level of system excitation) and its hysteresis is defined by the amp and hys parameters. In case of hys=0 the hysteresis is determined automatically according to the measurement noise properties on the controlled variable signal. The signal TBSY is set to onduring the tuning experiment. A successful experiment is indicated by and the controller parameters can be found on the outputs pk, pti, ptd, pnd and pb. The c weighting factor is assumed (and recommended) c=0. A failure during the experiment causes TE = on and the output ite provides further information about the problem. It is recommended to increase the amplitude amp in the case of error. The controller is equipped with a built-in function which decreases the amplitude when the deviation of output from the initial steady state exceeds the maxdev limit. The tuning experiment can be prematurely terminated by activating the TBRK input.

Inputs

dv	Feedforward control variable	Double (F64)
sp	Setpoint variable	Double (F64)
pv	Process variable	Double (F64)
tv	Tracking variable	Double (F64)
hv	Manual value	Double (F64)
MAN	Manual or automatic mode off . . . Automatic mode on . . . . Manual mode	Bool

`TUNE`	Start the tuning experiment	`Bool`
`TBRK`	Stop the tuning experiment	`Bool`

Outputs

`mv`	Manipulated variable (controller output)	`Double (F64)`
`de`	Deviation error	`Double (F64)`
`SAT`	Saturation flag `off` ... The controller implements a linear control law `on` .... The controller output is saturated	`Bool`
`TBSY`	Tuner busy flag	`Bool`
`TE`	Tuning error `off` ... Autotuning successful `on` .... An error occurred during the experiment	`Bool`
`ite`	Error code; expected time (in seconds) to finishing the tuning experiment while the tuning experiment is active `1000` .. Signal/noise ratio too low `1001` .. Hysteresis too high `1002` .. Too tight termination rule `1003` .. Phase out of interval	`Long (I32)`
`pk`	Proposed controller gain	`Double (F64)`
`pti`	Proposed integral time constant	`Double (F64)`
`ptd`	Proposed derivative time constant	`Double (F64)`
`pnd`	Proposed derivative component filtering	`Double (F64)`
`pb`	Proposed weighting factor – proportional component	`Double (F64)`

Parameters

`irtype`	Controller type (control law) ⊙`6` `1` ..... D `4` ..... P `7` ..... PID `2` ..... I `5` ..... PD `3` ..... ID `6` ..... PI	`Long (I32)`

`RACT`	Reverse action flag
	`off` ... Higher `mv` → higher `pv`
	`on` .... Higher `mv` → lower `pv`
		`Bool`

`k`	Controller gain K. By definition, the value 0 turns the controller off. Negative values are not allowed, use the `RACT` parameter for such a purpose. ↓`0.0` ⊙`1.0`	`Double (F64)`
`ti`	Integral time constant T_i. The value 0 disables the integrating part (the same effect as disabling it by the `irtype` parameter). ↓`0.0` ⊙`4.0`	`Double (F64)`
`td`	Derivative time constant T_d. The value 0 disables the derivative part (the same effect as disabling it by the `irtype` parameter). ↓`0.0` ⊙`1.0`	`Double (F64)`
`nd`	Derivative filtering parameter N. The value 0 disables the derivative part (the same effect as disabling it by the `irtype` parameter). ↓`0.0` ⊙`10.0`	`Double (F64)`

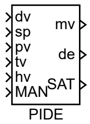
`b`	Setpoint weighting – proportional part	↓0.0 ⊙1.0	Double (F64)
`c`	Setpoint weighting – derivative part	↓0.0	Double (F64)
`tt`	Tracking time constant.	↓0.0 ⊙1.0	Double (F64)
`hilim`	Upper limit of the controller output	⊙1.0	Double (F64)
`lolim`	Lower limit of the controller output	⊙-1.0	Double (F64)
`iainf`	Type of apriori information 1 ..... No apriori information 2 ..... Astatic process (process with integration) 3 ..... Low order process 4 ..... Static process + slow closed loop step response 5 ..... Static process + middle fast (normal) closed loop step response 6 ..... Static process + fast closed loop step response	⊙1	Long (I32)
`k0`	Static gain of the process (must be provided in case of `iainf` = 3, 4, 5)	⊙1.0	Double (F64)
`n1`	Maximum number of half-periods for estimation of frequency response point	⊙20	Long (I32)
`mm`	Maximum number of half-periods for averaging	⊙4	Long (I32)
`amp`	Relay controller amplitude	⊙0.1	Double (F64)
`uhys`	Relay controller hysteresis		Double (F64)
`ntime`	Length of noise amplitude estimation period at the beginning of the tuning experiment [s]	⊙5.0	Double (F64)
`rerrap`	Termination value of the oscillation amplitude relative error	⊙0.1	Double (F64)
`aerrph`	Termination value of the absolute error in oscillation phase	⊙10.0	Double (F64)
`maxdev`	Maximal admissible deviation error from the initial steady state	⊙1.0	Double (F64)

It is recommended not to change the parameters `n1`, `mm`, `ntime`, `rerrap` and `aerrph`.

PIDE – PID controller with defined static error

Block Symbol

Licence: ADVANCED



Function Description

The PIDE block is a basis for creating a modified PI(D) controller which differs from the standard PI(D) controller (the PIDU block) by having a finite static gain (in fact, the value ε which causes the saturation of the output is entered). In the simplest case it can work autonomously and provide the standard functionality of the modified PID controller with two degrees of freedom in the automatic ($\texttt{MAN} = \texttt{off}$) or manual mode ($\texttt{MAN} = \texttt{on}$).

If in automatic mode and if the saturation limits are not active, the controller implements a linear control law given by

$$U(s) = \pm K \left[bW(s) - Y(s) + \frac{1}{T_i s + \beta} E(s) + \frac{T_d s}{\frac{T_d s}{N} + 1} (cW(s) - Y(s)) \right] + Z(s),$$

where

$$\beta = \frac{K\varepsilon}{1 - K\varepsilon}$$

$U(s)$ is the Laplace transform of the manipulated variable mv, $W(s)$ is the Laplace transform of the setpoint sp, $Y(s)$ is the Laplace transform of the process variable pv, $E(s)$ is the Laplace transform of the deviation error, $Z(s)$ is the Laplace transform of the feedforward control variable dv and K, T_i, T_d, N, ε $(= b_p/100)$, b and c are the controller parameters. The sign of the right hand side depends on the parameter RACT. The range of the manipulated variable mv (position controller output) is limited by parameters hilim, lolim.

By connecting the output mv of the controller to the controller input tv and properly setting the tracking time constant tt we obtain the bumpless operation of the controller in the case of the mode switching (manual, automatic) and also the correct operation of the controller when saturation of the output mv occurs (antiwindup).

In the manual mode ($\texttt{MAN} = \texttt{on}$), the input hv is copied to the output mv unless saturated. In this mode the inner controller state tracks the signal connected to the tv input so the successive switching to the automatic mode is bumpless. But the tracking is not precise for $\varepsilon > 0$.

Inputs

`dv`	Feedforward control variable		`Double (F64)`
`sp`	Setpoint variable		`Double (F64)`
`pv`	Process variable		`Double (F64)`
`tv`	Tracking variable		`Double (F64)`
`hv`	Manual value		`Double (F64)`
`MAN`	Manual or automatic mode `off` ... Automatic mode `on` .... Manual mode		`Bool`

Outputs

`mv`	Manipulated variable (controller output)		`Double (F64)`
`de`	Deviation error		`Double (F64)`
`SAT`	Saturation flag `off` ... The controller implements a linear control law `on` .... The controller output is saturated		`Bool`

Parameters

`irtype`	Controller type (control law) `1` ..... D `4` ..... P `7` ..... PID `2` ..... I `5` ..... PD `3` ..... ID `6` ..... PI	⊙6	`Long (I32)`

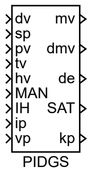
`RACT`	Reverse action flag
	`off` ... Higher `mv` → higher `pv`
	`on` .... Higher `mv` → lower `pv`
			`Bool`

`k`	Controller gain K	↓0.0 ⊙1.0	`Double (F64)`
`ti`	Integral time constant T_i	↓0.0 ⊙4.0	`Double (F64)`
`td`	Derivative time constant T_d	↓0.0 ⊙1.0	`Double (F64)`
`nd`	Derivative filtering parameter N	↓0.0 ⊙10.0	`Double (F64)`
`b`	Setpoint weighting – proportional part	↓0.0 ⊙1.0	`Double (F64)`
`c`	Setpoint weighting – derivative part	↓0.0	`Double (F64)`
`tt`	Tracking time constant. No meaning for controllers without integrator.	↓0.0 ⊙1.0	`Double (F64)`
`bp`	Static error coefficient		`Double (F64)`
`hilim`	Upper limit of the controller output	⊙1.0	`Double (F64)`
`lolim`	Lower limit of the controller output	⊙-1.0	`Double (F64)`

PIDGS – PID controller with gain scheduling

Block Symbol

Licence: ADVANCED



Function Description

The functionality of the PIDGS block is completely equivalent to the PIDU block. The only difference is that the PIDGS block has a at most six sets of basic PID controller parameters and allow bumpless switching of these sets by the ip (parameter set index) or vp inputs. In the latter case it is necessary to set GSCF = on and provide an array of threshold values thsha. The following rules define the active parameter set: the set 0 is active for vp < thrsha(0), the set 1 for thrsha(0) < vp < thrsha(1) etc. till the set 5 for thrsha(4) < vp. The index of the active parameter set is available at the kp output.

Inputs

dv	Feedforward control variable		Double (F64)
sp	Setpoint variable		Double (F64)
pv	Process variable		Double (F64)
tv	Tracking variable		Double (F64)
hv	Manual value		Double (F64)
MAN	Manual or automatic mode off . . . Automatic mode on . . . . Manual mode		Bool
IH	Integrator hold off . . . Integration enabled on . . . . Integration disabled		Bool
ip	Parameter set index	↓0 ↑5	Long (I32)
vp	Switching analog signal		Double (F64)

Outputs

mv	Manipulated variable (controller output)	Double (F64)
dmv	Controller velocity output (difference)	Double (F64)
de	Deviation error	Double (F64)
SAT	Saturation flag off . . . The controller implements a linear control law on . . . . The controller output is saturated	Bool

`kp`	Active parameter set index		`Long (I32)`

Parameters

`hilim`	Upper limit of the controller output	⊙`1.0`	`Double (F64)`
`lolim`	Lower limit of the controller output	⊙`-1.0`	`Double (F64)`
`dz`	Dead zone		`Double (F64)`
`icotype`	Controller output type `1` ..... Analog output `2` ..... Pulse width modulation (`PWM`) `3` ..... Step controller unit with position feedback (`SCU`) `4` ..... Step controller unit without position feedback (`SCUV`)	⊙`1`	`Long (I32)`
`nmax`	Reserved number of controller parameter sets	↓`4` ↑`10000` ⊙`10`	`Long (I32)`
`GSCF`	Switch parameters by analog signal `vp` `off` ... Index-based switching `on` .... Analog signal based switching		`Bool`
`hys`	Hysteresis for controller parameters switching		`Double (F64)`
`irtypea`	Vector of controller types (control laws) `1` ..... D `4` ..... P `7` ..... PID `2` ..... I `5` ..... PD `3` ..... ID `6` ..... PI	⊙`[6 6 6 6 6 6]`	`Byte (U8)`

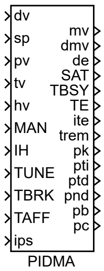
`RACTA`	Vector of reverse action flags
	`0` ..... Higher `mv` → higher `pv`
	`1` ..... Higher `mv` → lower `pv`
		⊙`[0 0 0 0 0 0]`	`Bool`

`ka`	Vector of controller gains K	⊙`[1.0 1.0 1.0 1.0 1.0 1.0]`	`Double (F64)`
`tia`	Vector of integral time constants T_i	⊙`[4.0 4.0 4.0 4.0 4.0 4.0]`	`Double (F64)`
`tda`	Vector of derivative time constants T_d	⊙`[1.0 1.0 1.0 1.0 1.0 1.0]`	`Double (F64)`
`nda`	Vector of derivative filtering parameters N	⊙`[10.0 10.0 10.0 10.0 10.0 10.0]`	`Double (F64)`
`ba`	Setpoint weighting factors – proportional part	⊙`[1.0 1.0 1.0 1.0 1.0 1.0]`	`Double (F64)`
`ca`	Setpoint weighting factors – derivative part	⊙`[0.0 0.0 0.0 0.0 0.0 0.0]`	`Double (F64)`
`tta`	Vector of tracking time constants	⊙`[1.0 1.0 1.0 1.0 1.0 1.0]`	`Double (F64)`
`thrsha`	Vector of thresholds for switching the parameters	⊙`[0.1 0.2 0.3 0.4 0.5 0]`	`Double (F64)`

PIDMA – PID controller with moment autotuner

Block Symbol

Licence: AUTOTUNING



Function Description

The PIDMA block has the same control function as the PIDU block. Additionally it is equipped with the moment autotuning function.

In the automatic mode (MAN = off), the block PIDMA implements the PID control law with two degrees of freedom in the form

$$U(s) = \pm K \left\{ bW(s) - Y(s) + \frac{1}{T_i s}\left[W(s) - Y(s)\right] + \frac{T_d s}{\frac{T_d}{N}s + 1}\left[cW(s) - Y(s)\right] \right\} + Z(s)$$

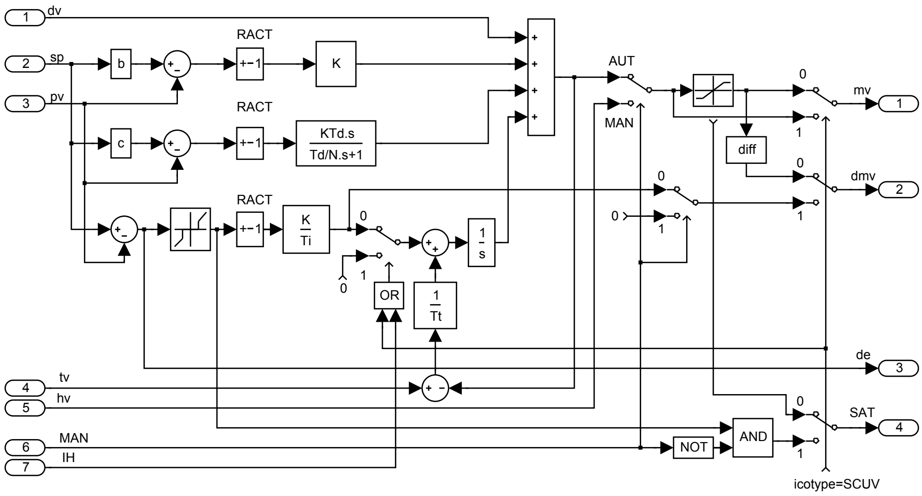
where $U(s)$ is Laplace transform of the manipulated variable mv, $W(s)$ is Laplace transform of the setpoint variable sp, $Y(s)$ is Laplace transform of the process variable pv, $Z(s)$ is Laplace transform of the feedforward control variable dv and K, T_i, T_d, N, b and c are the parameters of the controller. The sign of the right hand side depends on the parameter RACT. The range of the manipulated variable mv (position controller output) is limited by parameters hilim, lolim. The parameter dz determines the dead zone in the integral part of the controller. The integral part of the control law can be switched off and fixed on the current value by the integrator hold input IH = on. For the proper function of the controller it is necessary to connect the output mv of the controller to the controller input tv and properly set the tracking time constant tt.

The rule of thumb for a PID controller is $\mathtt{tt} \approx \sqrt{T_i T_d}$. For a PI controller the formula is $\mathtt{tt} \approx T_i/2$. In this way we obtain the bumpless operation of the controller in the case of the mode switching (manual, automatic) and also the correct operation of the controller when saturation of the output mv occurs (antiwindup).

The additional outputs dmv, de and SAT generate the velocity output (difference of mv), deviation error and saturation flag, respectively.

If the PIDMA block is connected with the block SCUV to configure the 3-point step controller without the positional feedback, then the parameter icotype must be set to 4 and the meaning of the outputs mv and dmv and SAT is modified in the following way: mv and `dmv` give the PD part and difference of I part of the control law, respectively, and `SAT` provides the information for the `SCUV` block whether the deviation error is less than the dead zone `dz` in the automatic mode. In this case, the setpoint weighting factor `c` should be zero.

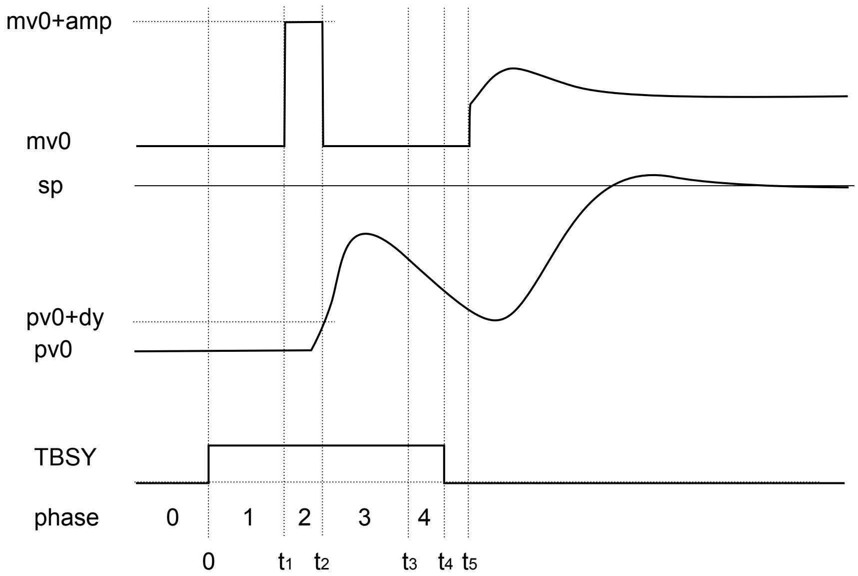
In the manual mode (`MAN` = `on`), the input `hv` is copied to the output `mv` unless saturated. The overall control function of the `PIDMA` block is quite clear from the following diagram:



The block `PIDMA` extends the control function of the standard PID controller by the built in autotuning feature. Before start of the autotuner the operator have to reach the steady state of the process at a suitable working point (in manual or automatic mode) and specify the required type of the controller `ittype` (`PI` or `PID`) and other tuning parameters (`iainf`, `DGC`, `tdg`, `tn`, `amp`, `dy` and `ispeed`). The identification experiment is started by the input `TUNE` (input `TBRK` finishes the experiment). In this mode (`TBSY` = `on`), first of all the noise and possible drift gradient (`DGC` = `on`) are estimated during the user specified time (`tdg`+`tn`) and then the rectangle pulse is applied to the input of the process and the first three process moments are identified from the pulse response. The amplitude of the pulse is set by the parameter `amp`. The pulse is finished when the process variable `pv` deviates from the steady value more than the `dy` threshold defines. The threshold is an absolute difference, therefore it is always a positive value. The duration of the tuning experiment depends on the dynamic behavior of the process. The remaining time to the end of the tuning is provided by the output `trem`.

If the identification experiment is properly finished (`TE` = `off`) and the input `ips` is equal to zero, then the optimal parameters immediately appear on the block outputs `pk`, `pti`, `ptd`, `pnd`, `pb`, `pc`. In the opposite case (`TE` = `on`) the output `ite` specifies the experiment error more closely. Other values of the `ips` input are reserved for custom specific purposes.

The function of the autotuner is illustrated in the following picture.



During the experiment, the output `ite` indicates the autotuner phases. In the phase of estimation of the response decay rate (`ite` = `-4`) the tuning experiment may be finished manually before its regular end. In this case the controller parameters are designed but the potential warning is indicated by setting the output `ite=100`.

At the end of the experiment (`TBSY` `on`→`off`), the function of the controller depends on the current controller mode. If the `TAFF` = `on` the designed controller parameters are immediately accepted.

Inputs

`dv`	Feedforward control variable	`Double (F64)`
`sp`	Setpoint variable	`Double (F64)`
`pv`	Process variable	`Double (F64)`
`tv`	Tracking variable	`Double (F64)`
`hv`	Manual value	`Double (F64)`
`MAN`	Manual or automatic mode `off` ... Automatic mode `on` .... Manual mode	`Bool`
`IH`	Integrator hold `off` ... Integration enabled `on` .... Integration disabled	`Bool`
`TUNE`	Start the tuning experiment (`off`→`on`) or force transition to the next tuning phase (see the description of the `ite` output)	`Bool`
`TBRK`	Stop the tuning experiment	`Bool`
`TAFF`	Tuning affirmation; determines the way the computed parameters are handled `off` ... Parameters are only computed `on` .... Parameters are set into the control law	`Bool`

`ips`	Meaning of the output signals `pk`, `pti`, `ptd`, `pnd`, `pb` and `pc` `0` ..... Designed parameters `k`, `ti`, `td`, `nd`, `b` and `c` of the PID control law `1` ..... Process moments: static gain (`pk`), resident time constant (`pti`), measure of the system response length (`ptd`) `2` ..... Three-parameter first-order plus dead-time model: static gain (`pk`), dead-time (`pti`), time constant (`ptd`) `3` ..... Three-parameter second-order plus dead-time model with double time constant: static gain (`pk`), dead-time (`pti`), time constant (`ptd`) `4` ..... Estimated boundaries for manual fine-tuning of the PID controller (`irtype` = 7) gain `k`: upper boundary k_{hi} (`pk`), lower boundary k_{lo} (`pti`) `>99` ... Reserved for diagnostic purposes	`Long (I32)`

Outputs

`mv`	Manipulated variable (controller output)	`Double (F64)`
`dmv`	Controller velocity output (difference)	`Double (F64)`
`de`	Deviation error	`Double (F64)`
`SAT`	Saturation flag `off` ... The controller implements a linear control law `on` .... The controller output is saturated	`Bool`
`TBSY`	Tuner busy flag	`Bool`
`TE`	Tuning error `off` ... Autotuning successful `on` .... An error occurred during the experiment	`Bool`
`ite`	Error code *Tuning error codes (after the experiment):* `0` ..... No error or waiting for steady state `1` ..... Too small pulse getdown threshold `2` ..... Too large pulse amplitude `3` ..... Steady state condition violation `4` ..... Too small pulse aplitude `5` ..... Peak search procedure failure `6` ..... Output saturation occurred during experiment `7` ..... Selected controller type not supported `8` ..... Process not monotonous `9` ..... Extrapolation failure `10` .... Unexpected values of moments (fatal) `11` .... Abnormal manual termination of tuning `12` .... Wrong direction of manipulated variable `100` ... Manual termination of tuning (warning)	`Long (I32)`

Tuning phases codes (during the experiment):

- `0` ..... Steady state reaching before the start of the experiment
- `-1` .... Drift gradient and noise estimation phase
- `-2` .... Pulse generation phase
- `-3` .... Searching the peak of system response
- `-4` .... Estimation of the system response decay rate

Remark about terminating the tuning phases

- `TUNE` .. The rising edge of the `TUNE` input during the phases `-2`, `-3` and `-4` causes the finishing of the current phase and transition to the next one (or finishing the experiment in the phase `-4`).

`trem`	Estimated time to finish the tuning experiment [s]	`Double (F64)`
`pk`	Proposed controller gain K (`ips` = 0)	`Double (F64)`
`pti`	Proposed integral time constant T_i (`ips` = 0)	`Double (F64)`
`ptd`	Proposed derivative time constant T_d (`ips` = 0)	`Double (F64)`
`pnd`	Proposed derivative component filtering N (`ips` = 0)	`Double (F64)`
`pb`	Proposed weighting factor – proportional component (`ips` = 0)	`Double (F64)`
`pc`	Proposed weighting factor – derivative component (`ips` = 0)	`Double (F64)`

Parameters

`irtype`	Controller type (control law) ⊙6 `1` ..... D `4` ..... P `7` ..... PID `2` ..... I `5` ..... PD `3` ..... ID `6` ..... PI	`Long (I32)`

`RACT`	Reverse action flag
	`off` ... Higher `mv` → higher `pv`
	`on` .... Higher `mv` → lower `pv`
		`Bool`

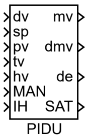
`k`	Controller gain K. By definition, the value 0 turns the controller off. Negative values are not allowed, use the `RACT` parameter for such a purpose. ↓0.0 ⊙1.0	`Double (F64)`
`ti`	Integral time constant T_i. The value 0 disables the integrating part (the same effect as disabling it by the `irtype` parameter). ↓0.0 ⊙4.0	`Double (F64)`
`td`	Derivative time constant T_d. The value 0 disables the derivative part (the same effect as disabling it by the `irtype` parameter). ↓0.0 ⊙1.0	`Double (F64)`
`nd`	Derivative filtering parameter N. The value 0 disables the derivative part (the same effect as disabling it by the `irtype` parameter). ↓0.0 ⊙10.0	`Double (F64)`
`b`	Setpoint weighting – proportional part ↓0.0 ↑2.0 ⊙1.0	`Double (F64)`
`c`	Setpoint weighting – derivative part ↓0.0 ↑2.0	`Double (F64)`

tt	Tracking time constant. The value 0 stands for an implicit value, which is $T_i/2$ or $\sqrt{T_i T_d}$ (see above) for controllers with integrating part. For controllers without integrating part, the value 0 disables tracking. If tracking is needed for a P or PD controller, it can be enabled by entering a positive value greater than the sampling time. It is not possible to turn off tracking for controllers with the integrating part (due to the windup effect). ↓0.0 ⊙1.0	Double (F64)
hilim	Upper limit of the controller output ⊙1.0	Double (F64)
lolim	Lower limit of the controller output ⊙-1.0	Double (F64)
dz	Dead zone	Double (F64)
icotype	Controller output type ⊙1 1 ..... Analog output 2 ..... Pulse width modulation (PWM) 3 ..... Step controller unit with position feedback (SCU) 4 ..... Step controller unit without position feedback (SCUV)	Long (I32)
ittype	Controller type to be designed ⊙6 6 ..... PI controller 7 ..... PID controller	Long (I32)
iainf	Type of apriori information ⊙1 1 ..... Static process 2 ..... Astatic process	Long (I32)
DGC	Drift gradient compensation ⊙on off ... Disabled on .... Enabled	Bool
tdg	Drift gradient estimation time [s] ⊙60.0	Double (F64)
tn	Length of noise estimation period [s] ⊙5.0	Double (F64)
amp	Tuning pulse amplitude ⊙0.5	Double (F64)
dy	Tuning pulse get down threshold (absolute difference from the steady pv value) ↓0.0 ⊙0.1	Double (F64)
ispeed	Desired closed loop speed ⊙2 1 ..... Slow closed loop 2 ..... Normal (middle fast) closed loop 3 ..... Fast closed loop	Long (I32)
ipid	PID controller form ⊙1 1 ..... Parallel form 2 ..... Series form	Long (I32)

PIDU – PID controller unit

Block Symbol

Licence: STANDARD



Function Description

The PIDU block is a basic block for creating a complete PID controller (or P, I, PI, PD, PID, PI+S). In the most simple case it works as a standalone unit with the standard PID controller functionality with two degrees of freedom. It can operate in automatic mode (MAN = off) or manual mode (MAN = on).

In the automatic mode (MAN = off), the block PIDU implements the PID control law with two degrees of freedom in the form

$$U(s) = \pm K \left\{ bW(s) - Y(s) + \frac{1}{T_i s} \left[W(s) - Y(s) \right] + \frac{T_d s}{\frac{T_d}{N} s + 1} \left[cW(s) - Y(s) \right] \right\} + Z(s)$$

where $U(s)$ is Laplace transform of the manipulated variable mv, $W(s)$ is Laplace transform of the setpoint variable sp, $Y(s)$ is Laplace transform of the process variable pv, $Z(s)$ is Laplace transform of the feedforward control variable dv and K, T_i, T_d, N, b and c are the parameters of the controller. The sign of the right hand side depends on the parameter RACT. The range of the manipulated variable mv (position controller output) is limited by parameters hilim, lolim. The parameter dz determines the dead zone in the integral part of the controller. The integral part of the control law can be switched off and fixed on the current value by the integrator hold input IH (IH = on). For the proper function of the controller it is necessary to connect the output mv of the controller to the controller input tv and properly set the tracking time constant tt.

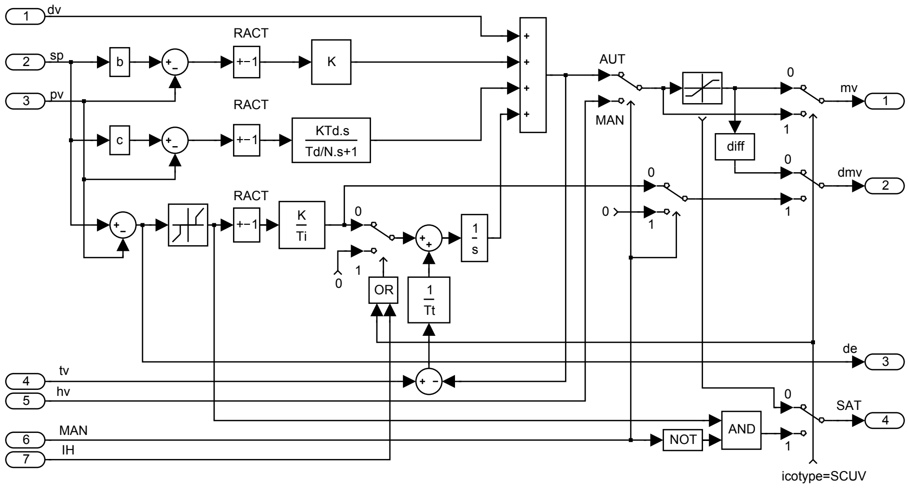
The rule of thumb for a PID controller is tt $\approx \sqrt{T_i T_d}$. For a PI controller the formula is tt $\approx T_i/2$. In this way we obtain the bumpless operation of the controller in the case of the mode switching (manual, automatic) and also the correct operation of the controller when saturation of the output mv occurs (antiwindup).

By adjusting the tt parameter, it is possible to tune the behaviour at saturation limits (so-called bouncing from limits due to noise) and when switching multiple controllers (bump in the controller output occurs when switching controllers while the control error is non-zero).

The additional outputs dmv, de and SAT generate the velocity output (difference of mv), deviation error and saturation flag, respectively.

If the PIDU block is connected with the SCUV block to configure the 3-point step controller without the positional feedback, then the parameter icotype must be set to 4 and the meaning of the outputs mv and dmv and SAT is modified in the following way: mv and dmv give the PD part and difference of I part of the control law, respectively, and SAT provides the information for the SCUV block whether the deviation error is less than the dead zone dz in the automatic mode. In this case, the setpoint weighting factor c should be zero.

In the manual mode (MAN = on), the input hv is copied to the output mv unless saturated. The overall control function of the PIDU block is quite clear from the following diagram:



Inputs

dv	Feedforward control variable	Double (F64)
sp	Setpoint variable	Double (F64)
pv	Process variable	Double (F64)
tv	Tracking variable	Double (F64)
hv	Manual value	Double (F64)
MAN	Manual or automatic mode off . . . Automatic mode on . . . . Manual mode	Bool
IH	Integrator hold off . . . Integration enabled on . . . . Integration disabled	Bool

Outputs

mv	Manipulated variable (controller output)	Double (F64)
dmv	Controller velocity output (difference)	Double (F64)
de	Deviation error	Double (F64)

`SAT`	Saturation flag `off` ... The controller implements a linear control law `on` .... The controller output is saturated		`Bool`

Parameters

`irtype`	Controller type (control law) `1` ..... D `4` ..... P `7` ..... PID `2` ..... I `5` ..... PD `3` ..... ID `6` ..... PI	⊙`6`	`Long (I32)`

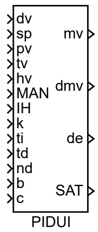
`RACT`	Reverse action flag
	`off` ... Higher `mv` → higher `pv`
	`on` .... Higher `mv` → lower `pv`
			`Bool`

`k`	Controller gain K. By definition, the value 0 turns the controller off. Negative values are not allowed, use the `RACT` parameter for such a purpose.	↓`0.0` ⊙`1.0`	`Double (F64)`
`ti`	Integral time constant T_i. The value 0 disables the integrating part (the same effect as disabling it by the `irtype` parameter).	↓`0.0` ⊙`4.0`	`Double (F64)`
`td`	Derivative time constant T_d. The value 0 disables the derivative part (the same effect as disabling it by the `irtype` parameter).	↓`0.0` ⊙`1.0`	`Double (F64)`
`nd`	Derivative filtering parameter N. The value 0 disables the derivative part (the same effect as disabling it by the `irtype` parameter).	↓`0.0` ⊙`10.0`	`Double (F64)`
`b`	Setpoint weighting – proportional part	↓`0.0` ↑`2.0` ⊙`1.0`	`Double (F64)`
`c`	Setpoint weighting – derivative part	↓`0.0` ↑`2.0`	`Double (F64)`
`tt`	Tracking time constant. The value 0 stands for an implicit value, which is $T_i/2$ or $\sqrt{T_i T_d}$ (see above) for controllers with integrating part. For controllers without integrating part, the value 0 disables tracking. If tracking is needed for a `P` or `PD` controller, it can be enabled by entering a positive value greater than the sampling time. It is not possible to turn off tracking for controllers with the integrating part (due to the windup effect).	↓`0.0` ⊙`1.0`	`Double (F64)`
`hilim`	Upper limit of the controller output	⊙`1.0`	`Double (F64)`
`lolim`	Lower limit of the controller output	⊙`-1.0`	`Double (F64)`
`dz`	Dead zone		`Double (F64)`
`icotype`	Controller output type `1` ..... Analog output `2` ..... Pulse width modulation (`PWM`) `3` ..... Step controller unit with position feedback (`SCU`) `4` ..... Step controller unit without position feedback (`SCUV`)	⊙`1`	`Long (I32)`

PIDUI – PID controller unit with variable parameters

Block Symbol

Licence: ADVANCED



Function Description

The functionality of the PIDUI block is completely equivalent to the PIDU block. The only difference is that the PID control algorithm parameters are defined by the input signals and therefore they can depend on the outputs of other blocks. This allows creation of special adaptive PID controllers.

Inputs

dv	Feedforward control variable	Double (F64)
sp	Setpoint variable	Double (F64)
pv	Process variable	Double (F64)
tv	Tracking variable	Double (F64)
hv	Manual value	Double (F64)
MAN	Manual or automatic mode off ... Automatic mode on .... Manual mode	Bool
IH	Integrator hold off ... Integration enabled on .... Integration disabled	Bool
k	Controller gain K	Double (F64)
ti	Integral time constant T_i	Double (F64)
td	Derivative time constant T_d	Double (F64)
nd	Derivative filtering parameter N	Double (F64)
b	Setpoint weighting – proportional part	Double (F64)
c	Setpoint weighting – derivative part	Double (F64)

Outputs

mv	Manipulated variable (controller output)	Double (F64)
dmv	Controller velocity output (difference)	Double (F64)
de	Deviation error	Double (F64)

`SAT`	Saturation flag `off` ... The controller implements a linear control law `on` .... The controller output is saturated		`Bool`

Parameters

`irtype`	Controller type (control law) `1` ..... D `4` ..... P `7` ..... PID `2` ..... I `5` ..... PD `3` ..... ID `6` ..... PI	⊙6	`Long (I32)`

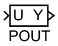
`RACT`	Reverse action flag
	`off` ... Higher `mv` → higher `pv`
	`on` .... Higher `mv` → lower `pv`
			`Bool`

`tt`	Tracking time constant	⊙1.0	`Double (F64)`
`hilim`	Upper limit of the controller output	⊙1.0	`Double (F64)`
`lolim`	Lower limit of the controller output	⊙-1.0	`Double (F64)`
`dz`	Dead zone		`Double (F64)`
`icotype`	Controller output type `1` ..... Analog output `2` ..... Pulse width modulation (`PWM`) `3` ..... Step controller unit with position feedback (`SCU`) `4` ..... Step controller unit without position feedback (`SCUV`)	⊙1	`Long (I32)`

POUT – Pulse output

Block Symbol

Licence: STANDARD



Function Description

The POUT block shapes the input pulses U in such a way, that the output pulse Y has a duration of at least dtime seconds and the idle period between two successive output pulses is at least btime seconds. The input pulse occuring sooner than the period of btime seconds since the last falling edge of the output signal elapses has no effect on the output signal Y.

Input

U	Logical input of the block		Bool

Output

Y	Logical output of the block		Bool

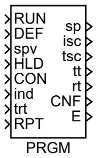
Parameters

dtime	Minimum width of the output pulse [s]	⊙1.0	Double (F64)
btime	Minimum delay between two successive output pulses [s]	⊙1.0	Double (F64)

PRGM – Setpoint programmer

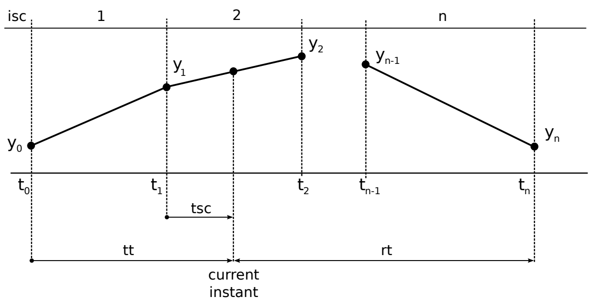
Block Symbol

Licence: STANDARD



Function Description

The PRGM block generates functions of time (programs) composed of n linear parts defined by (n + 1)-dimensional vectors of time (tm = $[t_0, \ldots, t_n]$) and output values (y = $[y_0, \ldots, y_n]$). The generated time-course is continuous piecewise linear, see figure below. This block is most commonly used as a setpoint generator for a controller. The program generation starts when RUN = on. In the case of RUN = off the programmer is set back to the initial state. The input DEF = on sets the output sp to the value spv. It follows a ramp to the nearest future node of the time function when DEF = off. The internal time of the generator is not affected by this input. The input HLD = on freezes the output sp and the internal time, thus also the outputs tsc, tt and rt. The program follows from freezing point as planned when HLD = off unless the input CON = on at the moment when the signal HLD on→off. In that case the program follows a ramp to reach the node with index ind in time trt. The node index ind must be equal to or higher than the index of current sector isc (at the moment when HLD on→off). If RPT = on, the program is generated repeatedly.



Inputs

RUN	Enable execution	Bool
DEF	Initialize sp to the value of spv	Bool
spv	Initializing constant	Double (F64)
HLD	Output and timer freezing	Bool

CON	Continue from defined node	Bool
ind	Index of the node to continue from	Long (I32)
trt	Time to reach the defined node with index ind	Double (F64)
RPT	Repetition flag	Bool

Outputs

sp	Setpoint variable (function value of the time function at given time)	Double (F64)
isc	Current function sector	Long (I32)
tsc	Time elapsed since the start of current sector	Double (F64)
tt	Time elapsed since the start of program generation	Double (F64)
rt	Remaining time till the end of program	Double (F64)
CNF	Flag indicating that the configured curve is being followed	Bool
E	Error flag – the node times are not ascending	Bool

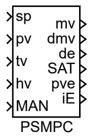
Parameters

nmax	Reserved (allocated) size of the tm, y vectors ↓4 ↑10000000 ⊙10	Long (I32)
tmunits	Time units ⊙1 1 ..... seconds 2 ..... minutes 3 ..... hours	Long (I32)
tm	$(\mathtt{n}+1)$-dimensional vector of ascending node times ⊙[0 1 2]	Double (F64)
y	$(\mathtt{n}+1)$-dimensional vector of node values (values of the time function) ⊙[0 1 0]	Double (F64)

PSMPC – Pulse-step model predictive controller

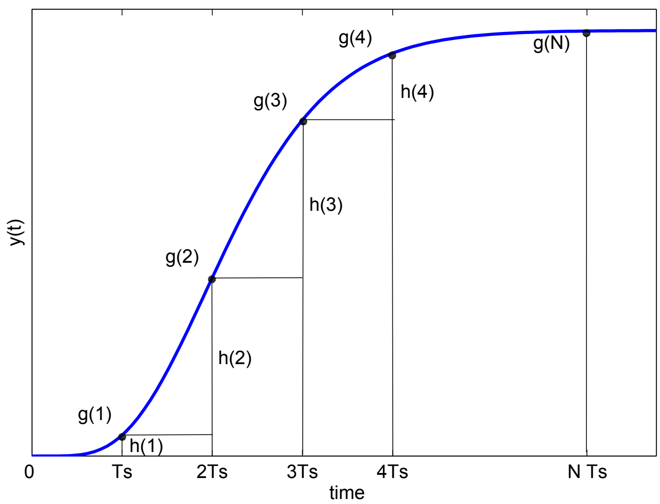
Block Symbol

Licence: ADVANCED



Function Description

The PSMPC block can be used for control of hardly controllable linear time-invariant systems with manipulated value constraints (e.g. time delay or non-minimum phase systems). It is especially well suited for the case when fast transition without overshoot from one level of controlled variable to another is required. In general, the PSMPC block can be used where the PID controllers are commonly used.



The PSMPC block is a predictive controller with explicitly defined constraints on the amplitude of manipulated variable.

The prediction is based on the discrete step response $g(j),\ j = 1, \ldots, N$ is used. The figure above shows how to obtain the discrete step response $g(j),\ j = 0, 1, \ldots, N$ and the discrete impulse response $h(j),\ j = 0, 1, \ldots, N$ with sampling period T_S from continuous step response. Note that N must be chosen such that $N \cdot T_S > t_{95}$, where t_{95} is the time to reach 95 % of the final steady state value.

For stable, linear and t-invariant systems with monotonous step response it is also possible to use the moment model set approach [4] and describe the system by only 3 characteristic numbers κ, μ, and σ^2, which can be obtained easily from a very short and simple experiment. The controlled system can be approximated by first order plus dead-time system

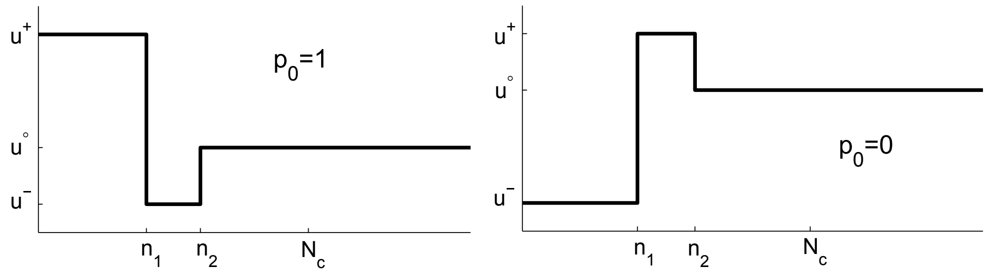
$$F_{FOPDT}(s) = \frac{K}{\tau s + 1} \cdot e^{-Ds}, \quad \kappa = K, \quad \mu = \tau + D, \quad \sigma^2 = \tau^2 \tag{7.1}$$

or second order plus dead-time system

$$F_{SOPDT}(s) = \frac{K}{(\tau s + 1)^2} \cdot e^{-Ds}, \quad \kappa = K, \quad \mu = 2\tau + D, \quad \sigma^2 = 2\tau^2 \tag{7.2}$$

with the same characteristic numbers. The type of approximation is selected by the `imtype` parameter.

To lower the computational burden of the open-loop optimization, the family of admissible control sequences contains only sequences in the so-called pulse-step shape depicted below:



Note that each of these sequences is uniquely defined by only four numbers $n_1, n_2 \in \{0, \ldots, N_C\}$, p_0 and $u^\infty \in \langle u^-, u^+\rangle$, where $N_C \in \{0, 1, \ldots\}$ is the control horizon and u^-, u^+ stand for the given lower and upper limit of the manipulated variable. The on-line optimization (with respect to p_0, n_1, n_2 and u^∞) minimizes the criterion

$$I = \sum_{i=N_1}^{N_2} \hat{e}(k+i|k)^2 + \lambda \sum_{i=0}^{N_C} \Delta\hat{u}(k+i\,|k)^2 \rightarrow \min, \tag{7.3}$$

where $\hat{e}(k+i|k)$ is the predicted control error at time k over the coincidence interval $i \in \{N_1, N_2\}$, $\Delta\hat{u}(k+i|k)$ are the differences of the control signal over the interval $i \in \{0, N_C\}$ and λ penalizes the changes in the control signal. The algorithm used for solving the optimization task (7.3) combines brute force and the least squares method. The value u^∞ is determined using the least squares method for all admissible combinations of p_0, n_1 and n_2 and the optimal control sequence is selected afterwards. The selected sequence in the pulse-step shape is optimal in the open-loop sense. To convert from open-loop to closed-loop control strategy, only the first element of the computed control sequence is applied and the whole optimization procedure is repeated in the next sampling instant.

The parameters N_1, N_2, H_C, and λ in the criterion (7.3) take the role of design parameters. Only the last parameter λ is meant for manual tuning of the controller. In the case the model in the form (7.1) or (7.2) is used, the parameters N_1 and N_2 are determined automatically with respect to the μ and σ^2 characteristic numbers. The controller can be then effectively tuned by adjusting the characteristic numbers κ, μ and σ^2.

Warning

It is necessary to set the `sr` array sufficiently large to avoid Matlab/Simulink crash when using the `PSMPC` block for simulation purposes. Especially when using FOPDT or SOPDT model, the `sr` array size must be greater than the length of the internally computed discrete step response.

Inputs

`sp`	Setpoint variable	`Double (F64)`
`pv`	Process variable	`Double (F64)`
`tv`	Tracking variable (applied control signal)	`Double (F64)`
`hv`	Manual value	`Double (F64)`
`MAN`	Manual or automatic mode `off` ... Automatic mode `on` .... Manual mode	`Bool`

Outputs

`mv`	Manipulated variable (controller output)	`Double (F64)`
`dmv`	Controller velocity output (difference)	`Double (F64)`
`de`	Deviation error	`Double (F64)`
`SAT`	Saturation flag `off` ... The controller implements a linear control law `on` .... The controller output is saturated	`Bool`
`pve`	Predicted process variable based on the controlled process model	`Double (F64)`
`iE`	Error code `0` ..... No error `1` ..... Incorrect FOPDT model `2` ..... Incorrect SOPDT model `3` ..... Invalid step response sequence	`Long (I32)`

Parameters

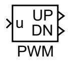
`nc`	Control horizon length (N_C)	⊙5	`Long (I32)`
`np1`	Start of coincidence interval (N_1)	⊙1	`Long (I32)`
`np2`	End of coincidence interval (N_2)	⊙10	`Long (I32)`
`lambda`	Control signal penalization coefficient (λ)	⊙0.05	`Double (F64)`
`umax`	Upper limit of the controller output (u^+)	⊙1.0	`Double (F64)`
`umin`	Lower limit of the controller output (u^-)	⊙-1.0	`Double (F64)`

`imtype`	Controlled process model type 1 ..... FOPDT model (7.1) 2 ..... SOPDT model (7.2) 3 ..... Discrete step response	⊙3	`Long (I32)`
`kappa`	Static gain (κ)	⊙1.0	`Double (F64)`
`mu`	Resident time constant (μ)	⊙20.0	`Double (F64)`
`sigma`	Measure of the system response length ($\sqrt{\sigma^2}$)	⊙10.0	`Double (F64)`
`nmax`	Reserved size of the *sr* array	↓10 ↑10000 ⊙32	`Long (I32)`
`sr`	Discrete step response sequence ($[g(1), \ldots, g(N)]$) ⊙[0 0.2642 0.5940 0.8009 0.9084 0.9596 0.9826 0.9927 0.9970 0.9988 0.9995]		`Double (F64)`

PWM – Pulse width modulation

Block Symbol

Licence: STANDARD



Function Description

The PWM block implements a pulse width modulation algorithm for proportional actuators. In the general, it is assumed the input signal u ranges in the interval from -1 to +1. The width L of the output pulse is computed by the expression:

$$L = \texttt{pertm} * |\texttt{u}|\,,$$

where pertm is the modulation time period. If $\texttt{u} > 0$ ($\texttt{u} < 0$), the pulse is generated in the output UP (DN). However, the width of the generated pulses are affected by other parameters of the block. The asymmetry factor asyfac determines the ratio of negative pulses duration to positive pulses duration. The modified pulse widths are given by:

$$\text{if} \quad \texttt{u} > 0 \quad \text{then} \quad L(\texttt{UP}) := \begin{cases} L & \text{for} \quad \texttt{asyfac} \leq 1.0 \\ L/\texttt{asyfac} & \text{for} \quad \texttt{asyfac} > 1.0 \end{cases}$$

$$\text{if} \quad \texttt{u} < 0 \quad \text{then} \quad L(\texttt{DN}) := \begin{cases} L * \texttt{asyfac} & \text{for} \quad \texttt{asyfac} \leq 1.0 \\ L & \text{for} \quad \texttt{asyfac} > 1.0 \end{cases}$$

Further, if the computed width is less than minimum pulse duration dtime the width is set to zero. If the pulse width differs from the modulation period pertm less than minimum pulse break time btime then width of the pulse is set to pertm. In the case the positive pulse is succeeded by the negative one (or vice versa) the latter pulse is possibly shifted in such a way that the distance between these pulses is at least equal to the minimum off time offtime. If SYNCH = on, then the change of the input value u causes the immediate recalculation of the current pulse widths if a synchronization condition is violated.

Input

u	Analog input of the block	Double (F64)

Outputs

UP	The "up" signal	Bool
DN	The "down" signal	Bool

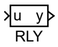
Parameters

pertm	Modulation period length [s]	⊙10.0	Double (F64)
dtime	Minimum width of the output pulse [s]	⊙0.1	Double (F64)
btime	Minimum delay between output pulses [s]	⊙0.1	Double (F64)
offtime	Minimum delay when altering direction [s]	⊙1.0	Double (F64)
asyfac	Asymmetry factor	⊙1.0	Double (F64)
SYNCH	Synchronization flag of the period start off ... Synchronization disabled on .... Synchronization enabled		Bool

RLY – Relay with hysteresis

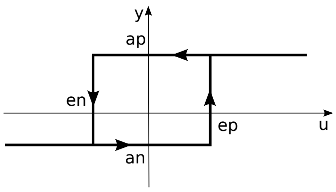
Block Symbol

Licence: STANDARD



Function Description

The RLY block transforms the input signal u to the output signal y according to the figure below.



Input

u	Analog input of the block	Double (F64)

Output

y	Analog output of the block	Double (F64)

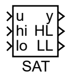
Parameters

ep	The value $u > ep$ causes $y = ap$ ("On")	⊙1.0	Double (F64)
en	The value $u < en$ causes $y = an$ ("Off")	⊙-1.0	Double (F64)
ap	Output value y in the "On" state	⊙1.0	Double (F64)
an	Output value y in the "Off" state	⊙-1.0	Double (F64)
y0	Initial output value at start-up		Double (F64)

SAT – Saturation with variable limits

Block Symbol

Licence: STANDARD



Function Description

The SAT block copies the input u to the output y if the input signal satisfies lolim $\leq$ u and u $\leq$ hilim, where lolim and hilim are state variables of the block. If u $<$ lolim (u $>$ hilim), then y $=$ lolim (y $=$ hilim). The upper and lower limits are either constants (HLD $=$ on) defined by parameters hilim0 and lolim0 respectively or input-driven variables (HLD $=$ off, hi and lo inputs). The maximal rate at which the active limits may vary is given by time constants tp (positive slope) and tn (negative slope). These rates are active even if the saturation limits are changed manually (HLD $=$ on) using the hilim0 and lolim0 parameters. To allow immediate changes of the saturation limits, set tp $= 0$ and tn $= 0$. The HL and LL outputs indicate the upper and lower saturation respectively.

If necessary, the hilim0 and lolim0 parameters are used as initial values for the input-driven saturation limits.

Inputs

u	Analog input of the block	Double (F64)
hi	Upper limit of the output signal (for the case HLD = off)	Double (F64)
lo	Lower limit of the output signal (for the case HLD = off)	Double (F64)

Outputs

y	Analog output of the block	Double (F64)
HL	Upper limit saturation indicator	Bool
LL	Lower limit saturation indicator	Bool

Parameters

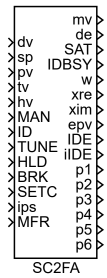
tp	Time constant defining the maximal positive slope of active limit changes	⊙1.0	Double (F64)
tn	Time constant defining the maximum negative slope of active limit changes	⊙1.0	Double (F64)
hilim0	Upper limit of the output (valid for HLD = on)	⊙1.0	Double (F64)
lolim0	Lower limit of the output (valid for HLD = on)	⊙-1.0	Double (F64)

HLD	Fixed saturation limits off … Variable limits on …. Fixed limits	⊙on	Bool

SC2FA – State controller for 2nd order system with frequency autotuner

Block Symbol

Licence: AUTOTUNING



Function Description

The SC2FA block implements a state controller for 2nd order system (7.4) with frequency autotuner. It is well suited especially for control (active damping) of lightly damped systems ($\xi < 0.1$). But it can be used as an autotuning controller for arbitrary system which can be described with sufficient precision by the transfer function

$$F(s) = \frac{b_1 s + b_0}{s^2 + 2\xi\Omega s + \Omega^2}, \tag{7.4}$$

where $\Omega > 0$ is the natural (undamped) frequency, ξ, $0 < \xi < 1$, is the damping coefficient and b_1, b_0 are arbitrary real numbers. The block has two operating modes: "Identification and design mode" and "Controller mode".

The "Identification and design mode" is activated by the binary input ID = on. Two points of frequency response with given phase delay are measured during the identification experiment. Based on these two points a model of the controlled system is built. The experiment itself is initiated by the rising edge of the RUN input. A harmonic signal with amplitude uamp, frequency ω and bias ubias then appears at the output mv. The frequency runs through the interval $\langle \mathtt{wb}, \mathtt{wf} \rangle$, it increases gradually. The current frequency is copied to the output w. The rate at which the frequency changes (sweeping) is determined by the cp parameter, which defines the relative shrinking of the initial period $T_b = \frac{2\pi}{\mathtt{wb}}$ of the exciting sine wave in time T_b, thus

$$c_p = \frac{\mathtt{wb}}{\omega(T_b)} = \frac{\mathtt{wb}}{\mathtt{wb} e^{\gamma T_b}} = e^{-\gamma T_b}.$$

The cp parameter usually lies within the interval $\mathtt{cp} \in \langle 0{,}95; 1)$. The lower the damping coefficient ξ of the controlled system is, the closer to one the cp parameter must be.

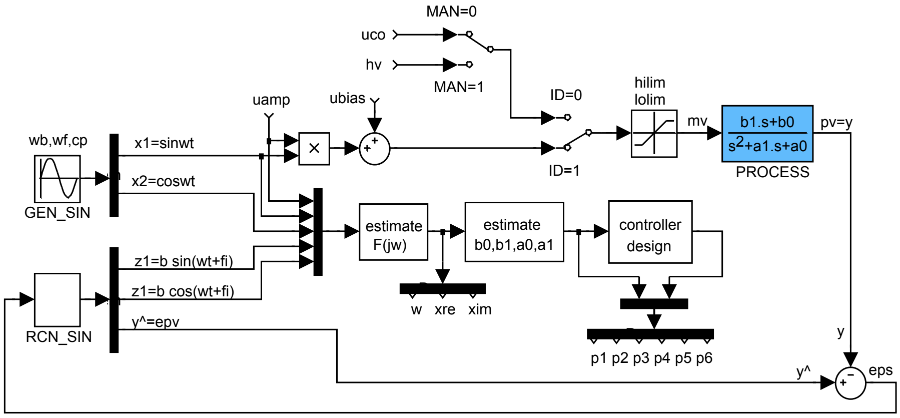
At the beginning of the identification period the exciting signal has a frequency of $\omega = \texttt{wb}$. After a period of `stime` seconds the estimation of current frequency response point starts. Its real and imaginary parts are available at the `xre` and `xim` outputs. If the `MANF` parameter is set to 0, then the frequency sweeping is stopped two times during the identification period. This happens when points with phase delay of `ph1` and `ph2` are reached for the first time. The breaks are `stime` seconds long. Default phase delay values are $-60°$ and $-120°$, respectively, but these can be changed to arbitrary values within the interval $(-360°, 0°)$, where $\texttt{ph1} > \texttt{ph2}$. At the end of each break an arithmetic average is computed from the last `iavg` frequency point estimates. Thus we get two points of frequency response which are successively used to compute the controlled process model in the form of (7.4). If the `MANF` parameter is set to 1, then the selection of two frequency response points is manual. To select the frequency, set the input `HLD` = `on`, which stops the frequency sweeping. The identification experiment continues after returning the input `HLD` to 0. The remaining functionality is unchanged.

It is possible to terminate the identification experiment prematurely in case of necessity by the input `BRK` = `on`. If the two points of frequency response are already identified at that moment, the controller parameters are designed in a standard way. Otherwise the controller design cannot be performed and the identification error is indicated by the output signal `IDE` = `on`.

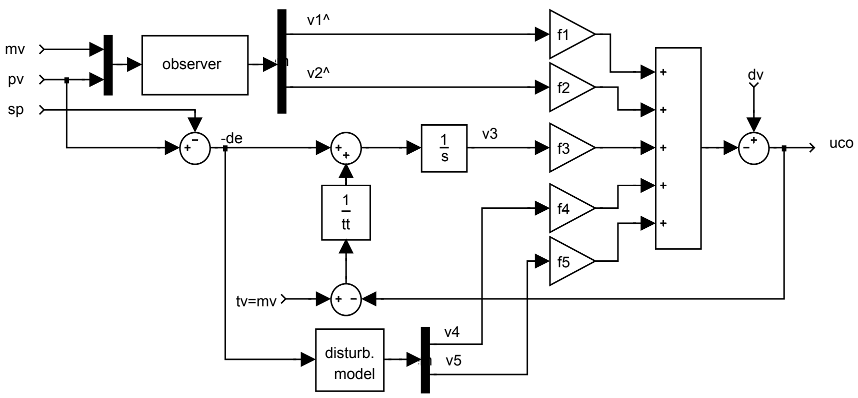
The `IDBSY` output is set to 1 during the "identification and design" phase. It is set back to 0 after the identification experiment finishes. A successful controller design is indicated by the output `IDE` = `off`. During the identification experiment the output `iIDE` displays the individual phases of the identification: `iIDE` = -1 means approaching the first point, `iIDE` = 1 means the break at the first point, `iIDE` = -2 means approaching the second point, `iIDE` = 2 means the break at the second point and `iIDE` = -3 means the last phase after leaving the second frequency response point. An error during the identification phase is indicated by the output `IDE` = `on` and the output `iIDE` provides more information about the error.

The computed state controller parameters are taken over by the control algorithm as soon as the `SETC` input is set to 1 (i.e. immediately if `SETC` is constantly set to `on`). The identified model and controller parameters can be obtained from the `p1`, `p2`, ..., `p6` outputs after setting the `ips` input to the appropriate value. After a successful identification it is possible to generate the frequency response of the controlled system model, which is initiated by a rising edge at the `MFR` input. The frequency response can be read from the `w`, `xre` and `xim` outputs, which allows easy confrontation of the model and the measured data.

The "Controller mode" (binary input `ID` = `off`) has manual (`MAN` = `on`) and automatic (`MAN` = `off`) submodes. After a cold start of the block with the input `ID` = `off` it is assumed that the block parameters `mb0`, `mb1`, `ma0` and `ma1` reflect formerly identified coefficients b_0, b_1, a_0 and a_1 of the controlled system transfer function and the state controller design is performed automatically. Moreover if the controller is in the automatic mode and `SETC` = `on`, then the control law uses the parameters from the very beginning. In this way the identification phase can be skipped when starting the block repeatedly.



The diagram above is a simplified inner structure of the frequency autotuning part of the controller. The diagram below shows the state feedback, observer and integrator anti-wind-up. The diagram does not show the fact, that the controller design block automatically adjusts the observer and state feedback parameters $f1, \ldots, f5$ after identification experiment (and `SETC` $=$ `on`).



The controlled system is assumed in the form of (7.4). Another forms of this transfer function are

$$F(s) = \frac{(b_1 s + b_0)}{s^2 + a_1 s + a_0} \tag{7.5}$$

and

$$F(s) = \frac{K_0 \Omega^2 (\tau s + 1)}{s^2 + 2\xi\Omega s + \Omega^2}. \tag{7.6}$$

The coefficients of these transfer functions can be found at the outputs `p1`,...,`p6` after the identification experiment (`IDBSY` = `off`). The output signals meaning is switched when a change occurs at the `ips` input.

Inputs

`dv`	Feedforward control variable	`Double (F64)`
`sp`	Setpoint variable	`Double (F64)`
`pv`	Process variable	`Double (F64)`
`tv`	Tracking variable	`Double (F64)`
`hv`	Manual value	`Double (F64)`

`MAN`	Manual or automatic mode `off` ... Automatic mode `on` .... Manual mode	`Bool`
`ID`	Identification or controller operating mode `off` ... Controller mode mode `on` .... Identification and design	`Bool`
`TUNE`	Start the tuning experiment (`off`→`on`), the exciting harmonic signal is generated	`Bool`
`HLD`	Stop frequency sweeping	`Bool`
`BRK`	Termination signal	`Bool`
`SETC`	Flag for accepting the new controller parameters and updating the control law `off` ... Parameters are only computed `on` .... Parameters are accepted as soon as computed `off`→`on` One-shot confirmation of the computed parameters	`Bool`
`ips`	Switch for changing the meaning of the output signals `0` ..... Two points of frequency response `p1` ... frequency of the 1st measured point in rad/s `p2` ... real part of the 1st point `p3` ... imaginary part of the 1st point `p4` ... frequency of the 2nd measured point in rad/s `p5` ... real part of the 2nd point `p6` ... imaginary part of the 2nd point `1` ..... Second order model in the form (7.5) `p1` ... b_1 parameter `p2` ... b_0 parameter `p3` ... a_1 parameter `p4` ... a_0 parameter `2` ..... Second order model in the form (7.6) `p1` ... K_0 parameter `p2` ... τ parameter `p3` ... Ω parameter in rad/s `p4` ... ξ parameter `p5` ... Ω parameter in Hz `p6` ... resonance frequency in Hz `3` ..... State feedback parameters `p1` ... f_1 parameter `p2` ... f_2 parameter `p3` ... f_3 parameter `p4` ... f_4 parameter `p5` ... f_5 parameter	`Long (I32)`
`MFR`	Generation of the parametric model frequency response at the `w`, `xre` and `xim` outputs (`off`→`on` triggers the generator)	`Bool`

Outputs

`mv`	Manipulated variable (controller output)	`Double (F64)`
`de`	Deviation error	`Double (F64)`

`SAT`	Saturation flag `off` ... The controller implements a linear control law `on` .... The controller output is saturated		`Bool`
`IDBSY`	Identification running `off` ... Identification not running `on` .... Identification in progress		`Bool`
`w`	Frequency response point estimate - frequency in rad/s		`Double (F64)`
`xre`	Frequency response point estimate - real part		`Double (F64)`
`xim`	Frequency response point estimate - imaginary part		`Double (F64)`
`epv`	Reconstructed `pv` signal		`Double (F64)`
`IDE`	Identification error indicator `off` ... Successful identification experiment `on` .... Identification error occurred		`Bool`
`iIDE`	Error code `101` ... Sampling period too low `102` ... Error identifying one or both frequency response point(s) `103` ... Manipulated variable saturation occurred during the identification experiment `104` ... Invalid process model		`Long (I32)`
`p1..p6`	Results of identification and design phase		`Double (F64)`

Parameters

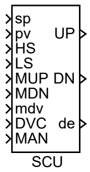
`ubias`	Static component of the exciting harmonic signal		`Double (F64)`
`uamp`	Amplitude of the exciting harmonic signal	⊙`1.0`	`Double (F64)`
`wb`	Frequency interval lower limit [rad/s]	⊙`1.0`	`Double (F64)`
`wf`	Frequency interval upper limit [rad/s]	⊙`10.0`	`Double (F64)`
`isweep`	Frequency sweeping mode `1` ..... Logarithmic `2` ..... Linear (not implemented yet)	⊙`1`	`Long (I32)`
`cp`	Sweeping rate	↓`0.5` ↑`1.0` ⊙`0.995`	`Double (F64)`
`iavg`	Number of values for averaging	⊙`10`	`Long (I32)`
`alpha`	Relative positioning of the observer poles (in identification phase)	⊙`2.0`	`Double (F64)`
`xi`	Observer damping coefficient (in identification phase)	⊙`0.707`	`Double (F64)`
`MANF`	Manual frequency response points selection `off` ... Disabled `on` .... Enabled		`Bool`
`ph1`	Phase delay of the 1st point in degrees	⊙`-60.0`	`Double (F64)`
`ph2`	Phase delay of the 2nd point in degrees	⊙`-120.0`	`Double (F64)`
`stime`	Settling period [s]	⊙`10.0`	`Double (F64)`
`ralpha`	Relative positioning of the observer poles	⊙`4.0`	`Double (F64)`
`rxi`	Observer damping coefficient	⊙`0.707`	`Double (F64)`
`acl1`	Relative positioning of the 1st closed-loop poles couple	⊙`1.0`	`Double (F64)`
`xicl1`	Damping of the 1st closed-loop poles couple	⊙`0.707`	`Double (F64)`

`INTGF`	Integrator flag `off` ... State-space model without integrator `on` .... Integrator included in the state-space model	⊙`on`	`Bool`
`apcl`	Relative position of the real pole	⊙`1.0`	`Double (F64)`
`DISF`	Disturbance flag `off` ... State space model without disturbance model `on` .... Disturbance model is included in the state space model		`Bool`
`dom`	Disturbance model natural frequency	⊙`1.0`	`Double (F64)`
`dxi`	Disturbance model damping coefficient		`Double (F64)`
`acl2`	Relative positioning of the 2nd closed-loop poles couple	⊙`2.0`	`Double (F64)`
`xicl2`	Damping of the 2nd closed-loop poles couple	⊙`0.707`	`Double (F64)`
`tt`	Tracking time constant	⊙`1.0`	`Double (F64)`
`hilim`	Upper limit of the controller output	⊙`1.0`	`Double (F64)`
`lolim`	Lower limit of the controller output	⊙`-1.0`	`Double (F64)`
`mb1p`	Controlled system transfer function coefficient b_1		`Double (F64)`
`mb0p`	Controlled system transfer function coefficient b_0	⊙`1.0`	`Double (F64)`
`ma1p`	Controlled system transfer function coefficient a_1	⊙`0.2`	`Double (F64)`
`ma0p`	Controlled system transfer function coefficient a_0	⊙`1.0`	`Double (F64)`

SCU – Step controller with position feedback

Block Symbol

Licence: STANDARD



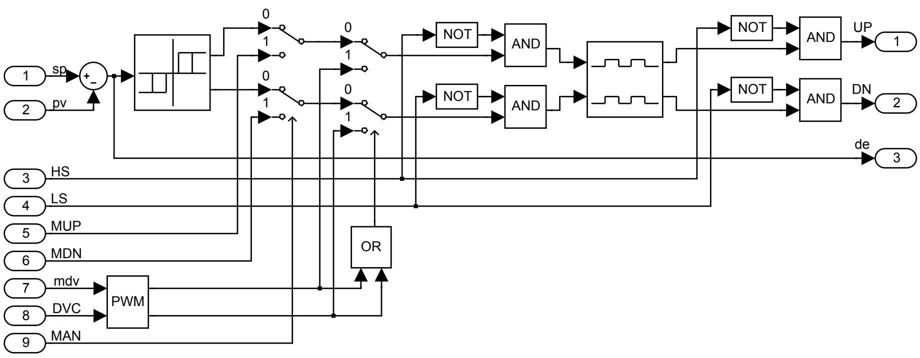
Function Description

The SCU block implements the secondary (inner) position controller of the step controller loop. PIDU function block or some of the derived function blocks (PIDMA, etc.) is assumed as the primary controller.

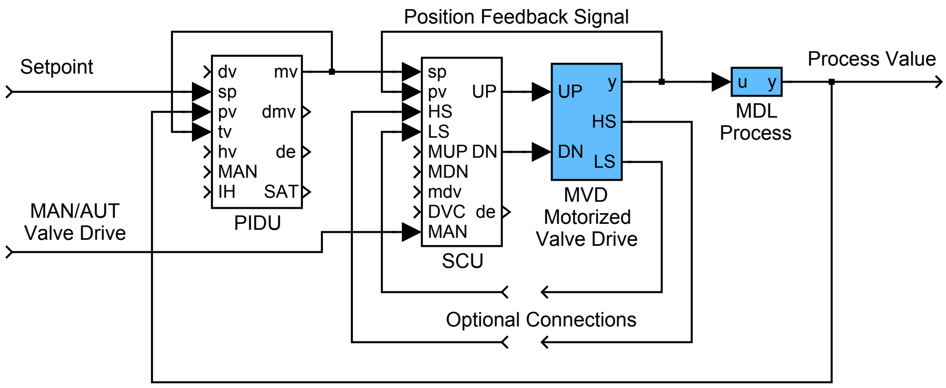
The SCU block processes the control deviation sp − pv by a three state element with parameters (thresholds) **thron** and **throff** (see the TSE block, use parameters **ep** = **thron**, **epoff** = **throff**, **en** = **-thron** and **enoff** = **-throff**). The parameter RACT determines whether the UP or DN pulse is generated for positive or negative value of the controller deviation. Two pulse outputs of the three state element are further shaped so that minimum pulse duration **dtime** and minimum pulse break time **btime** are guaranteed at the block UP and DN outputs. If signals from high and low limit switches of the valve are available, they should be connected to the HS and LS inputs.

There is also a group of input signals for manual control available. The manual mode is activated by the MAN = **on** input signal. Then it is possible to move the motor back and forth by the MUP and MDN input signals. It is also possible to specify a position increment/decrement request by the **mdv** input. In this case the request must be confirmed by a rising edge (**off**→**on**) in the DVC input signal.

The control function of the SCU block is quite clear from the following diagram.



The complete structure of the three-state step controller is depicted in the following figure.



Inputs

`sp`	Setpoint (output of the primary controller)	`Double (F64)`
`pv`	Controlled variable (position of the motorized valve drive)	`Double (F64)`
`HS`	Upper end switch (detects the upper limit position of the valve)	`Bool`
`LS`	Lower end switch (detects the lower limit position of the valve)	`Bool`
`MUP`	Manual `UP` signal	`Bool`
`MDN`	Manual `DN` signal	`Bool`
`mdv`	Manual differential value (requested position increment/decrement with higher priority than direct signals `MUP`/`MDN`)	`Double (F64)`
`DVC`	Differential value change command (`off`→`on`)	`Bool`
`MAN`	Manual or automatic mode `off` ... Automatic mode `on` .... Manual mode	`Bool`

Outputs

`UP`	The "up" signal	`Bool`
`DN`	The "down" signal	`Bool`
`de`	Deviation error	`Double (F64)`

Parameters

`thron`	Switch-on value	↓0.0 ⊙0.02	`Double (F64)`
`throff`	Switch-off value	↓0.0 ⊙0.01	`Double (F64)`
`dtime`	Minimum width of the output pulse [s]	↓0.0 ⊙0.1	`Double (F64)`
`btime`	Minimum delay between two subsequent output pulses [s] to do	↓0.0 ⊙0.1	`Double (F64)`

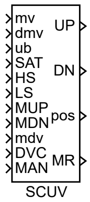
`RACT`	Reverse action flag
	`off` ... Higher `mv` $\rightarrow$ higher `pv`
	`on` .... Higher `mv` $\rightarrow$ lower `pv`
		`Bool`

`trun`	Motor time constant (determines the time during which the motor position changes by one unit) ↓`0.0` ⊙`10.0`	`Double (F64)`

SCUV – Step controller unit with velocity input

Block Symbol

Licence: STANDARD



Function Description

The block SCUV substitutes the secondary position controller SCU in the step controller loop when the position signal is not available. The primary controller PIDU (or some of the derived function blocks) is connected with the block SCUV using the block inputs mv, dmv and SAT.

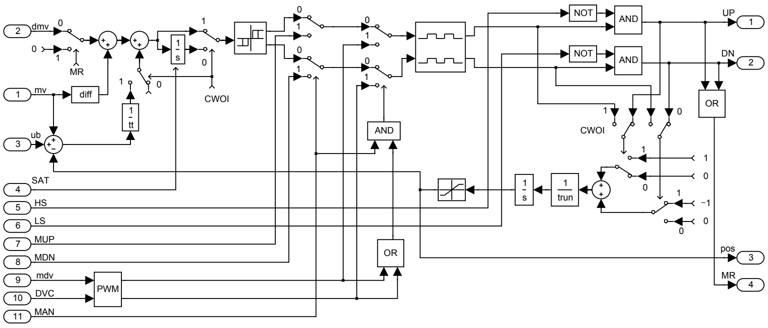
If the primary controller uses PI or PID control law (CWOI = off), then all three inputs mv, dmv and SAT of the block SCUV are sequentially processed by the special integration algorithm and by the three state element with parameters thron and throff (see the TSE block, use parameters ep = thron, epoff = throff, en = -thron and enoff = -throff). Pulse outputs of the three state element are further shaped in such a way that the minimum pulse duration time dtime and minimum pulse break time btime are guaranteed at the block outputs UP and DN. The parameter RACT determines the direction of motor moving. Note, the velocity output of the primary controller is reconstructed from input signals mv and dmv. Moreover, if the deviation error of the primary controller with icotype = 4 (working in automatic mode) is less than its dead zone (SAT = on), then the output of the corresponding internal integrator is set to zero.

The position pos of the valve is estimated by an integrator with the time constant trun. If signals from high and low limit switches of the valve are available, they should be connected to the inputs HS and LS.

If the primary controller uses P or PD control law (CWOI = on), then the deviation error of the primary controller can be eliminated by the bias ub manually. In this case, the control algorithm is slightly modified, the position of the motor pos is used and the proper settings of thron, throff and the tracking time constant tt are necessary for the suppressing of up/down pulses in the steady state.

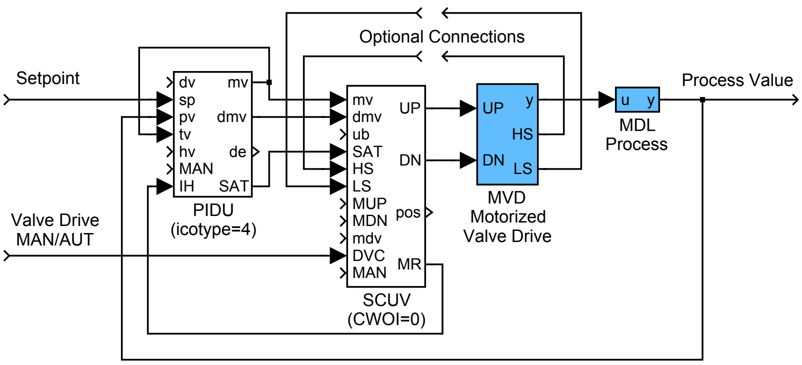
There is also a group of input signals for manual control available. The manual mode is activated by the MAN = on input signal. Then it is possible to move the motor back and forth by the MUP and MDN input signals. It is also possible to specify a position increment/decrement request by the mdv input. In this case the request must be confirmed by a rising edge (off→on) in the DVC input signal.

The overall control function of the SCUV block is obvious from the following diagram:

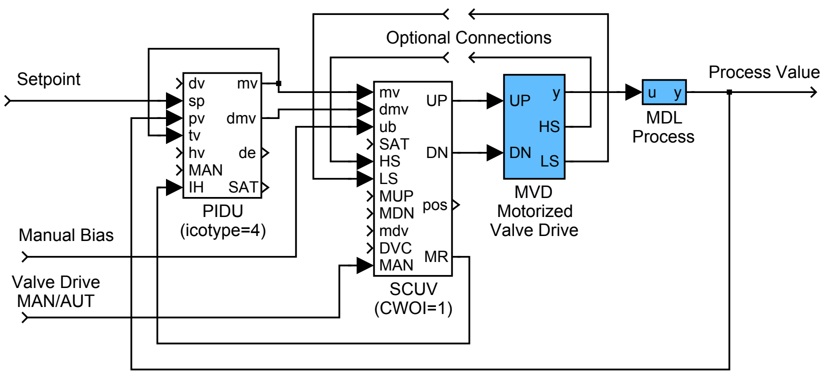


The complete structures of the three-state controllers are depicted in the following figures:

Primary controller with integration: I, PI, PID



Primary controller without integration: P, PD



Inputs

mv	Manipulated variable (controller output)	Double (F64)
dmv	Controller velocity output (difference)	Double (F64)

`ub`	Bias (only for P or PD primary controller)	`Double (F64)`
`SAT`	Internal integrator reset (connected to the `SAT` output of the primary controller)	`Bool`
`HS`	Upper end switch (detects the upper limit position of the valve)	`Bool`
`LS`	Lower end switch (detects the lower limit position of the valve)	`Bool`
`MUP`	Manual `UP` signal	`Bool`
`MDN`	Manual `DN` signal	`Bool`
`mdv`	Manual differential value (requested position increment/decrement with higher priority than direct signals `MUP`/`MDN`)	`Double (F64)`
`DVC`	Differential value change command (`off`→`on`)	`Bool`
`MAN`	Manual or automatic mode `off` ... Automatic mode `on` .... Manual mode	`Bool`

Outputs

`UP`	The "up" signal	`Bool`
`DN`	The "down" signal	`Bool`
`pos`	Position output of motor simulator	`Double (F64)`
`MR`	Request to move the motor `off` ... Motor idle (`UP` = `off` and `DN` = `off`) `on` .... Request to move (`UP` = `on` or `DN` = `on`)	`Bool`

Parameters

`thron`	Switch-on value ↓`0.0` ⊙`0.02`	`Double (F64)`
`throff`	Switch-off value ↓`0.0` ⊙`0.01`	`Double (F64)`
`dtime`	Minimum width of the output pulse [s] ↓`0.0` ⊙`0.1`	`Double (F64)`
`btime`	Minimum delay between two subsequent output pulses [s] ↓`0.0` ⊙`0.1`	`Double (F64)`

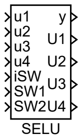
`RACT`	Reverse action flag
	`off` ... Higher `mv` → higher `pv`
	`on` .... Higher `mv` → lower `pv`
		`Bool`

`trun`	Motor time constant (determines the time during which the motor position changes by one unit) ↓`0.0` ⊙`10.0`	`Double (F64)`
`CWOI`	Controller without integration flag `off` ... The primary controller has an integrator (I, PI, PID) `on` .... The primary controller does not have an integrator (P, PD)	`Bool`
`tt`	Tracking time constant ↓`0.0` ⊙`1.0`	`Double (F64)`

SELU – Controller selector unit

Block Symbol

Licence: STANDARD



Function Description

The SELU block is tailored for selecting the active controller in selector control. It chooses one of the input signals u1, u2, u3, u4 and copies it to the output y. For BINF = off the active signal is selected by the iSW input. In the case of BINF = on the selection is based on the binary inputs SW1 and SW2 according to the following table:

iSW	SW1	SW2	y	U1	U2	U3	U4
0	off	off	u1	off	on	on	on
1	off	on	u2	on	off	on	on
2	on	off	u3	on	on	off	on
3	on	on	u4	on	on	on	off

This table also explains the meaning of the binary outputs U1, U2, U3 and U4, which are used by the inactive controllers in selector control for tracking purposes (via the SWU blocks).

Inputs

u1..u4	Signals to be selected from	Double (F64)
iSW	Active signal selector in case of BINF = off	Long (I32)
SW1	Binary signal selector, used when BINF = on	Bool
SW2	Binary signal selector, used when BINF = on	Bool

Outputs

y	Analog output of the block	Double (F64)
U1..U4	Binary output signal for selector control	Bool

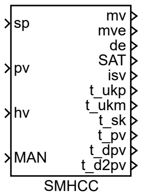
Parameter

BINF	Enable the binary selectors off ... Disabled (analog selector) on .... Enabled (binary selectors)	Bool

SMHCC – Sliding mode heating/cooling controller

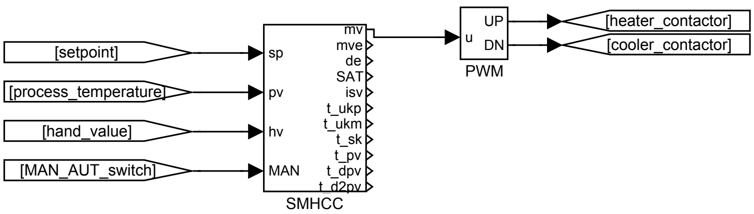
Block Symbol

Licence: ADVANCED



Function Description

The sliding mode heating/cooling controller SMHCC is a novel high quality control algorithm intended for temperature control of heating-cooling (possibly asymmetrical) processes with ON-OFF heaters and/or ON-OFF coolers. The plastic extruder is a typical example of such process. However, it can also be applied to many similar cases, for example in thermal systems where a conventional thermostat is employed. To provide the proper control function the block SMHCC must be combined with the block PWM (Pulse Width Modulation) as depicted in the following figure.



It is important to note that the block SMHCC works with two time periods. The first period T_S is the sampling time of the process temperature, and this period is equal to the period with which the block SMHCC itself is executed. The second period $T_C = i_{pwmc}T_S$ is the control period with which the block SMHCC generates manipulated variable. This period T_C is also equal to the cycle time of PWM block. At every instant when the manipulated variable mv is changed by SMHCC the PWM algorithm recalculates the width of the output pulse and starts a new PWM cycle. The time resolution T_R of the PWM block is third time period involved with. This period is equal to the period with which the block PWM is run and generally may be different from T_S. To achieve the high quality of control it is recommended to choose T_S as minimal as possible (i_{pwmc} as maximal as possible), the ratio T_C/T_S as maximal as possible but T_C should be sufficiently small with respect to the process dynamics. An example of reasonable values for an extruder temperature control is as follows:

$$T_S = 0.1,\ i_{pwmc} = 100,\ T_C = 10s,\ T_R = 0.01s.$$

The control law of the block SMHCC in automatic mode (MAN = off) is based on the discrete dynamic sliding mode control technique and special 3rd order filters for estimation of the first and second derivatives of the control error.

The first control stage, after a setpoint change or upset, is the *reaching phase* when the dynamic sliding variable

$$s_k \triangleq \ddot{e}_k + 2\xi\Omega\dot{e}_k + \Omega^2 e_k$$

is forced to zero. In the above definition of the sliding variable, $e_k, \dot{e}_k, \ddot{e}_k$ denote the filtered deviation error (pv − sp) and its first and second derivatives in the control period k, respectively, and ξ, Ω are the control parameters described below. In the second phase, s_k is hold at the zero value (*the sliding phase*) by the proper control "bangs". Here, the heating action is alternated by cooling action and *vice versa* rapidly. The amplitudes of control actions are adapted appropriately to guarantee $s_k = 0$ approximately. Thus, the hypothetical continuous dynamic sliding variable

$$s \triangleq \ddot{e} + 2\xi\Omega\dot{e} + \Omega^2 e$$

is approximately equal to zero at any time. Therefore the control deviation behaves according to the second order differential equation

$$s \triangleq \ddot{e} + 2\xi\Omega\dot{e} + \Omega^2 e = 0$$

describing so called *zero sliding dynamics*. From it follows that the evolution of e can be prescribed by the parameters ξ, Ω. For stable behavior, it must hold $\xi > 0, \Omega > 0$. A typical optimal value of ξ ranges in the interval $[4, 8]$ and ξ about 6 is often a satisfactory value. The optimal value of Ω strongly depends on the controlled process. The slower processes the lower optimal Ω. The recommended value of Ω for start of tuning is $\pi/(5T_C)$.
The manipulated variable mv usually ranges in the interval $[-1, 1]$. The positive (negative) value corresponds to heating (cooling). For example, mv = 1 means the full heating. The limits of mv can be reduced when needed by the controller parameters hilim_p and hilim_m. This reduction is probably necessary when the asymmetry between heating and cooling is significant. For example, if in the working zone the cooling is much more aggressive than heating, then these parameters should be set as hilim_p = 1 and hilim_m < 1. If we want to apply such limitation only in some time interval after a change of setpoint (during the transient response) then it is necessary to set initial value of the heating (cooling) action amplitude u0_p (u0_m) to the suitable value less than hilim_p (hilim_m). Otherwise set u0_p = hilim_p and u0_m = hilim_m.

The current amplitudes of heating and cooling uk_p, uk_m, respectively, are automatically adapted by the special algorithm to achieve so called *quasi sliding mode*, where the sign of s_k alternately changes its value. In such a case the controller output isv alternates the values 1 and −1. The rate of adaptation of the heating (cooling) amplitude is given by the time constant taup (taum). Both of these time constants have to be sufficiently high to provide the proper function of adaptation but the fine tuning is not necessary.

Note for completeness that the manipulated variable mv is determined from the action amplitudes uk_p, uk_m by the following expression

$$\text{if} \quad (s_k < 0.0) \quad \text{then} \quad \texttt{mv} = \texttt{uk\_p} \quad \text{else} \quad \texttt{mv} = -\texttt{uk\_m}.$$

Further, it is important to note that quasi sliding is seldom achievable because of a process dead time or disturbances. The suitable indicator of the quality of sliding is again the output isv. If the extraordinary fine tuning is required then it may be tried to find the better value for the bandwidth parameter beta of derivative filter, otherwise the default value 0.1 is preferred. In the manual mode (MAN = on) the controller input hv is (after limitation to the range [−hilim_m, hilim_p]) copied to the manipulated variable mv.

Inputs

sp	setpoint variable	Double (F64)
pv	process variable	Double (F64)
hv	manual value	Double (F64)
MAN	controller mode 0 ..... automatic mode 1 ..... manual mode	Bool

Outputs

mv	manipulated variable (position controller output)	Double (F64)
mve	equivalent manipulated variable	Double (F64)
de	deviation error	Double (F64)
SAT	saturation flag 0 ..... the controller implements a linear control law 1 ..... the controller output is saturated, mv ≥ hilim_p or mv ≤ -hilim_m	Bool
isv	number of the positive (+) or negative (−) sliding variable steps	Long (I32)
t_ukp	current amplitude of heating	Double (F64)
t_ukm	current amplitude of cooling	Double (F64)
t_sk	discrete dynamic sliding variable s_k	Double (F64)
t_pv	filtered control error -de	Double (F64)
t_dpv	filtered first derivative of the control error t_ek	Double (F64)
t_d2pv	filtered second derivative of the control error t_ek	Double (F64)

Parameters

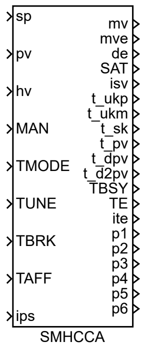
ipwmc	PWM cycle in the sampling periods of SMHCC (T_C/T_S)	Long (I32)
xi	relative damping ξ of sliding zero dynamics xi ≥ 0	Double (F64)
om	natural frequency Ω of sliding zero dynamics ↓(0.0)	Double (F64)
taup	time constant for adaptation of heating action amplitude in seconds	Double (F64)

taum	time constant for adaptation of cooling action amplitude in seconds		Double (F64)
beta	bandwidth parameter of the derivative filter	↓0	Double (F64)
hilim_p	high limit of the heating action amplitude	↓0.0 ↑1.0	Double (F64)
hilim_m	high limit of the cooling action amplitude	↓0.0 ↑1.0	Double (F64)
u0_p	initial value of the heating action amplitude after setpoint change and start of the block		Double (F64)
u0_m	initial value of the cooling action amplitude after setpoint change and start of the block		Double (F64)
sp_dif	Setpoint difference threshold	⊙10.0	Double (F64)
tauf	Equivalent manipulated variable filter time constant	⊙400.0	Double (F64)

SMHCCA – Sliding mode heating/cooling controller with autotuner

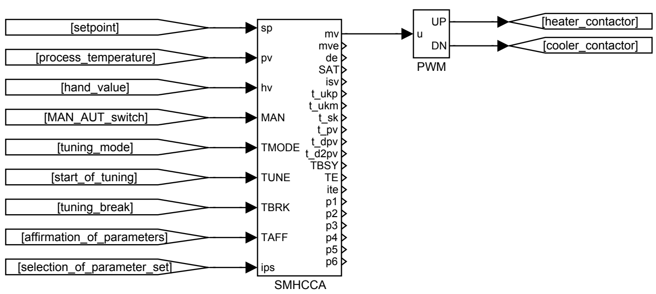
Block Symbol

Licence: AUTOTUNING



Function Description

The sliding mode heating/cooling controller (SMHCCA) is a novel high quality control algorithm with a built-in autotuner for automatic tuning of the controller parameters. The controller is mainly intended for temperature control of heating-cooling (possibly asymmetrical) processes with ON-OFF heaters and/or ON-OFF coolers. The plastic extruder heating/cooling system is a typical example of such process. However, it can also be applied to many similar cases, for example, to thermal systems where a conventional thermostat is normally employed. To provide the proper control function, the SMHCCA block must be combined with the PWM block (Pulse Width Modulation) as depicted in the following figure.



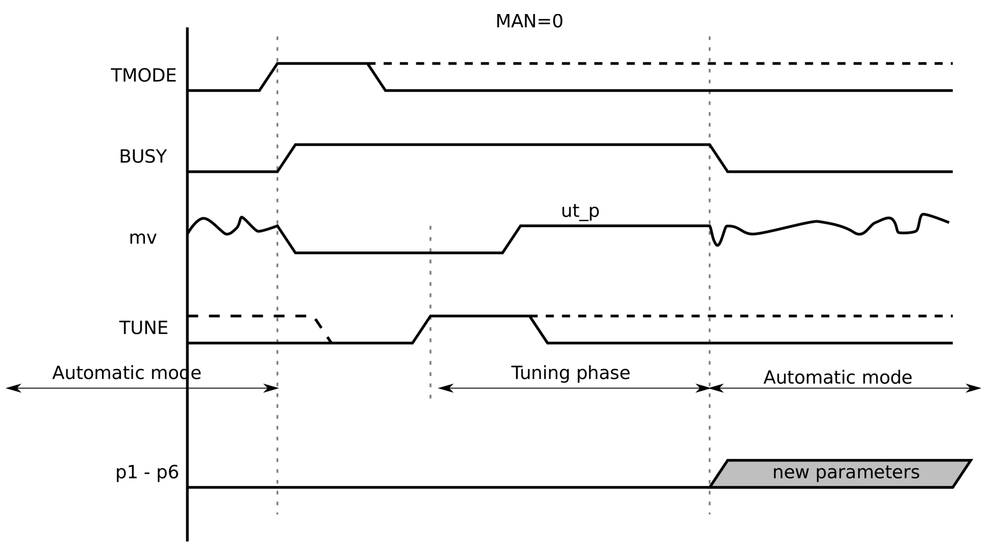
It is important to note that the block SMHCCA works with two time periods. The first period T_S is the sampling time of the process temperature, and this period is equal to the period with which the block SMHCCA itself is executed. The other period $T_C = i_{pwmc} T_S$ is the control period with which the block SMHCCA generates the manipulated variable. This period T_C is equal to the cycle time of PWM block. At every instant when the manipulated variable mv is changed by SMHCCA the PWM algorithm recalculates the width of the output pulse and starts a new PWM cycle. The time resolution T_R of the PWM block is third time period involved in. This period is equal to the period with which the block PWM is executed and generally may be different from T_S. To achieve the high quality of control it is recommended to choose T_S as minimal as possible (i_{pwmc} as maximal as possible), the ratio T_C/T_S as maximal as possible but T_C should be sufficiently small with respect to the process dynamics. An example of reasonable values for an extruder temperature control is as follows:

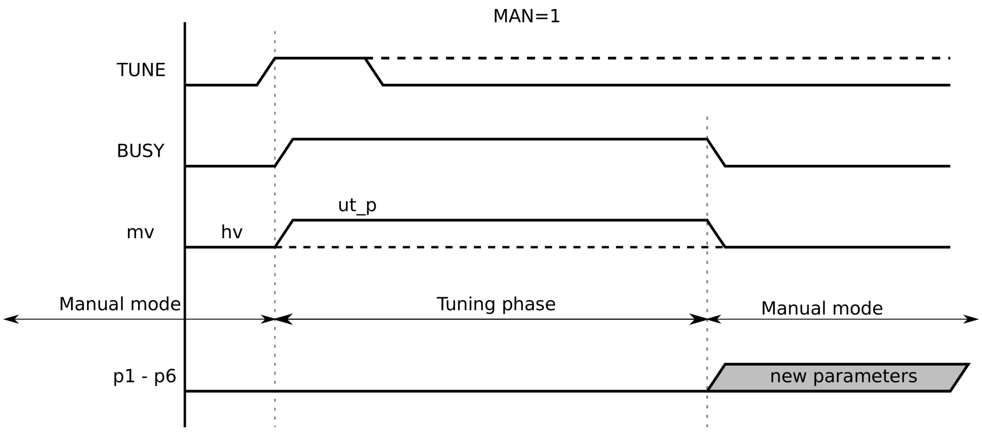
$$T_S = 0.1,\ i_{pwmc} = 50,\ T_C = 5s,\ T_R = 0.1s.$$

Notice however that for a faster controlled system the sampling periods T_S, T_C and T_R must be shortened! More precisely, the three minimal time constant of the process are important for selection of these time periods (all real thermal process has at least three time constants). For example, the sampling period $T_S = 0.1$ is sufficiently short for such processes that have at least three time constants, the minimal of them is greater than 10s and the maximal is greater than 100s. For the proper function of the controller it is necessary that these time parameters are suitably chosen by the user according to the actual dynamics of the process! If `SMHCCA` is implemented on a processor with floating point arithmetic then the accurate setting of the sampling periods T_S, T_C, T_R and the parameter `beta` is critical for correct function of the controller. Also, some other parameters with the clear meaning described below have to be chosen manually. All the remaining parameters (`xi`, `om`, `taup`, `taum`, `tauf`) can be set by the built-in autotuner automatically. The autotuner uses the two methods for this purpose.

- The first one is dedicated to situations where the asymmetry of the process is not enormous (approximately, it means that the gain ratio of heating/cooling or cooling/heating is less than 5).
- The second method provides the tuning support for the strong asymmetric processes and is not implemented yet (So far, this method has been developed and tested in Simulink only).

Despite the fact that the first method of the tuning is based only on the heating regime, the resulting parameters are usually satisfactory for both heating and cooling regimes because of the strong robustness of sliding mode control. The tuning procedure is very quick and can be accomplished during the normal rise time period of the process temperature from cold state to the setpoint usually without any temporization or degradation of control performance. Thus the tuning procedure can be included in every start up from cold state to the working point specified by the sufficiently high temperature setpoint. Now the implemented procedure will be described in detail. The tuning procedure starts in the tuning mode or in the manual mode. If the tuning mode (`TMODE` = `on`) is selected the manipulated variable `mv` is automatically set to zero and the output `TBSY` is set to 1 for indication of the tuning stage of the controller. The cold state of the process is preserved until the initialization pulse is applied to the input `TUNE` (`0` $\rightarrow$ `1`). After some time (depending on `beta`), when the noise amplitude is estimated, the heating is switched on with the amplitude given by the parameter ut_p. The process temperature pv and its two derivatives (outputs t_pv, t_dpv, t_d2pv) are observed to obtain the optimal parameters of the controller. If the tuning procedure ends without errors, then TBSY is set to 0 and the controller begins to work in manual or automatic mode according to the input MAN. If MAN = off and affirmation input TAFF is set to 1, then the controller starts to work in automatic mode with the new parameter set provided by the tuner (if TAFF = off, then the new parameters are only displayed on the outputs p1..p6). If some error occurs during the tuning, then the tuning procedure stops immediately or stops after the condition pv>sp is fulfilled, the output TE is set to 1 and ite indicate the type of error. Also in this case, the controller starts to work in the mode determined by the input MAN. If MAN = off then works in automatic mode with the initial parameters before tuning! The tuning errors are usually caused either by an inappropriate setting of the parameter beta or by the too low value of sp. The suitable value of beta ranges in the interval (0.001,0.1). If a drift and noise in pv are large the small beta must be chosen especially for the tuning phase. The default value (beta=0.01) should work well for extruder applications. The correct value gives properly filtered signal of the second derivative of the process temperature t_d2pv. This well-filtered signal (corresponding to the low value of beta) is mainly necessary for proper tuning. For control, the parameter beta may be sometimes slightly increased. The tuning procedure may be also started from manual mode (MAN = off) with any constant value of the input hv. However, the steady state must be provided in this case. Again, the tuning is started by the initialization pulse at the input TUNE (0 → 1) and after the stop of tuning the controller continues in the manual mode. In both cases the resulting parameters appear on the outputs p1,...,p6.





The control law of the block SMHCCA in automatic mode (MAN = off) is based on the discrete dynamic sliding mode control technique and special 3rd order filters for estimation of the first and second derivatives of the control error.

The first control stage, after a setpoint change or upset, is the *reaching phase* when the dynamic sliding variable

$$s_k \stackrel{\triangle}{=} \ddot{e}_k + 2\xi\Omega\dot{e}_k + \Omega^2 e_k$$

is forced to zero. In the above definition of the sliding variable, $e_k, \dot{e}_k, \ddot{e}_k$ denote the filtered deviation error (pv − sp) and its first and second derivatives in the control period k, respectively, and ξ, Ω are the control parameters described below. In the second phase, s_k is hold at the zero value (*the sliding phase*) by the proper control "bangs". Here, the heating action is alternated by cooling action and *vice versa* rapidly. The amplitudes of control actions are adapted appropriately to guarantee $s_k = 0$ approximately. Thus, the hypothetical continuous dynamic sliding variable

$$s \stackrel{\triangle}{=} \ddot{e} + 2\xi\Omega\dot{e} + \Omega^2 e$$

is approximately equal to zero at any time. Therefore the control deviation behaves according to the second order differential equation

$$s \stackrel{\triangle}{=} \ddot{e} + 2\xi\Omega\dot{e} + \Omega^2 e = 0$$

describing so called *zero sliding dynamics*. From it follows that the evolution of e can be prescribed by the parameters ξ, Ω. For stable behavior, it must hold $\xi > 0, \Omega > 0$. A typical optimal value of ξ ranges in the interval $[4, 8]$ and ξ about 6 is often a satisfactory value. The optimal value of Ω strongly depends on the controlled process. The slower processes the lower optimal Ω. The recommended value of Ω for start of tuning is $\pi/(5T_C)$.

The manipulated variable mv usually ranges in the interval $[-1, 1]$. The positive (negative) value corresponds to heating (cooling). For example, mv $= 1$ means the full heating. The limits of mv can be reduced when needed by the controller parameters hilim_p and hilim_m. This reduction is probably necessary when the asymmetry between heating and cooling is significant. For example, if in the working zone the cooling is much more aggressive than heating, then these parameters should be set as hilim_p = 1 and hilim_m < 1. If we want to apply such limitation only in some time interval after a change of setpoint (during the transient response) then it is necessary to set initial value of the heating (cooling) action amplitude u0_p (u0_m) to the suitable value less than hilim_p (hilim_m). Otherwise set u0_p = hilim_p and u0_m = hilim_m.

The current amplitudes of heating and cooling uk_p, uk_m, respectively, are automatically adapted by the special algorithm to achieve so called *quasi sliding mode*, where the sign of s_k alternately changes its value. In such a case the controller output isv alternates the values 1 and −1. The rate of adaptation of the heating (cooling) amplitude is given by time constant taup (taum). Both of these time constants have to be sufficiently high to provide the proper function of adaptation but the fine tuning is not necessary. Note for completeness that the manipulated variable mv is determined from the action amplitudes uk_p, uk_m by the following expression

$$\text{if} \quad (s_k < 0.0) \quad \text{then} \quad \texttt{mv} = \texttt{uk\_p} \quad \text{else} \quad \texttt{mv} = -\texttt{uk\_m}.$$

Further, it is important to note that quasi sliding is seldom achievable because of a process dead time or disturbances. The suitable indicator of the quality of sliding is again the output isv. If the extraordinary fine tuning is required then it may be tried to find the better value for the bandwidth parameter beta of derivative filter, otherwise the default value 0.1 is preferred.

In the manual mode (MAN = on) the controller input hv is (after limitation to the range [−hilim_m, hilim_p]) copied to the manipulated variable mv. The controller output mve provides the equivalent amplitude-modulated value of the manipulated variable mv for informative purposes. The output mve is obtained by the first order filter with the time constant tauf applied to mv.

Inputs

sp	Setpoint variable	Double (F64)
pv	Process variable	Double (F64)
hv	Manual value	Double (F64)
MAN	Manual or automatic mode 0 ..... Automatic mode 1 ..... Manual mode	Bool
TMODE	Tuning mode	Bool
TUNE	Start the tuning experiment: TUNE off→on	Bool
TBRK	Stop the tuning experiment: TBRK off→on	Bool
TAFF	Affirmation of the parameter set provided by the tuning procedure: TAFF = on	Bool

ips	Meaning of the output signals p1,...,p6 0 ..... Controller parameters p1 ... recommended control period T_C p2 ... xi p3 ... om p4 ... taup p5 ... taum p6 ... tauf 1 ..... Auxiliary parameters p1 ... htp2 – time of the peak in the second derivative of pv p2 ... hpeak2 – peak value in the second derivative of pv p3 ... d2 – peak to peak amplitude of t_d2pv p4 ... tgain	Long (I32)

Outputs

mv	Manipulated variable (controller output)	Double (F64)
mve	Equivalent manipulated variable	Double (F64)
de	Deviation error	Double (F64)
SAT	Saturation flag 0 ..... Signal not limited 1 ..... Saturation limits active, mv $\geq$ hilim_p or mv $\leq$ -hilim_m	Bool
isv	Number of the positive (+) or negative (−) sliding variable steps	Long (I32)
t_ukp	Current amplitude of heating	Double (F64)
t_ukm	Current amplitude of cooling	Double (F64)
t_sk	Discrete dynamic sliding variable	Double (F64)
t_pv	Filtered process variable pv by 3rd order filter	Double (F64)
t_dpv	Filtered first derivative of pv by 3rd order filter	Double (F64)
t_d2pv	Filtered second derivative of pv by 3rd order filter	Double (F64)
TBSY	Tuner busy flag (TBSY = on)	Bool
TE	Tuning error off ... Autotuning successful on .... An error occured during the experiment	Bool

`ite`	Error code 0 ..... No error 1 ..... Noise level in `pv` too high, check the temperature input 2 ..... Incorrect parameter `ut_p` 3 ..... Setpoint `sp` too low 4 ..... The two minimal process time constants are probably too small with respect to the sampling period T_S OR too high level of noise in the second derivative of `pv` (try to decrease the `beta` parameter) 5 ..... Premature termination of the tuning procedure (`TBRK`)	`Long (I32)`
`p`i	Identified parameters with respect to `ips`, $i = 1, ..., 6$	`Double (F64)`

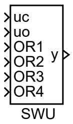
Parameters

`ipwmc`	PWM cycle (in sampling periods of the block, T_C/T_S) ⊙`100`	`Long (I32)`
`xi`	Relative damping of sliding zero dynamics ↓`0.5`↑`8.0`⊙`1.0`	`Double (F64)`
`om`	Natural frequency ω of sliding zero dynamics ↓`0.0`⊙`0.01`	`Double (F64)`
`taup`	Time constant for adaptation of heating action amplitude [s] ⊙`700.0`	`Double (F64)`
`taum`	Time constant for adaptation of cooling action amplitude [s] ⊙`400.0`	`Double (F64)`
`beta`	Bandwidth parameter of the derivative filter ⊙`0.01`	`Double (F64)`
`hilim_p`	Upper limit of the heating action amplitude ↓`0.0` ↑`1.0` ⊙`1.0`	`Double (F64)`
`hilim_m`	Upper limit of the cooling action amplitude ↓`0.0` ↑`1.0` ⊙`1.0`	`Double (F64)`
`u0_p`	Initial amplitude of the heating action ⊙`1.0`	`Double (F64)`
`u0_m`	Initial amplitude of the cooling action ⊙`1.0`	`Double (F64)`
`sp_dif`	Setpoint difference threshold for blocking of heating/cooling amplitudes reset ⊙`10.0`	`Double (F64)`
`tauf`	Time constant of the filter for obtaining the equivalent manipulated variable ⊙`400.0`	`Double (F64)`
`itm`	Tuning method ⊙`1` 1 ..... Restricted to symmetrical processes 2 ..... Asymmetrical processes (not implemented yet)	`Long (I32)`
`ut_p`	Amplitude of heating for tuning experiment ↓`0.0` ↑`1.0` ⊙`1.0`	`Double (F64)`
`ut_m`	Amplitude of cooling for tuning experiment ↓`0.0` ↑`1.0` ⊙`1.0`	`Double (F64)`

SWU – Switch unit

Block Symbol

Licence: STANDARD



Function Description

The SWU block is used to select the appropriate signal which should be tracked by the inactive PIDU and MCU units in complex control structures. The input signal uc is copied to the output y when all the binary inputs OR1, ..., OR4 are off, otherwise the output y takes over the uo input signal.

Inputs

uc	This input is copied to output y when all the binary inputs OR1, OR2, OR3 and OR4 are off	Double (F64)
uo	This input is copied to output y when any of the binary inputs OR1, OR2, OR3, OR4 is on	Double (F64)
OR1	First logical output of the block	Bool
OR2	Second logical output of the block	Bool
OR3	Third logical output of the block	Bool
OR4	Fourth logical output of the block	Bool

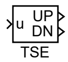
Output

y	Analog output of the block	Double (F64)

TSE – Three-state element

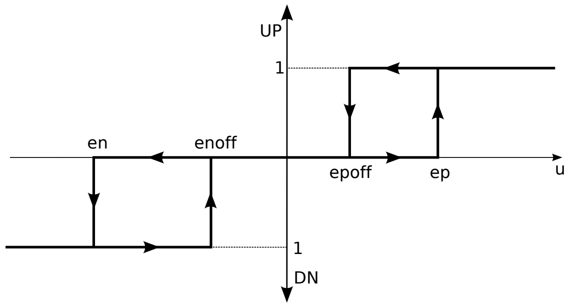
Block Symbol

Licence: STANDARD



Function Description

The TSE block transforms the analog input u to a three-state signal ("up", "idle" and "down") according to the diagram below.



Input

u	Analog input of the block	Double (F64)

Outputs

UP	The "up" signal	Bool
DN	The "down" signal	Bool

Parameters

ep	The input value u > ep results in UP = on and DN = off	⊙1.0	Double (F64)
en	The input value u < en results in UP = off and DN = off	⊙-1.0	Double (F64)
epoff	UP switch off value; if UP = on and u < epoff then UP = off	⊙0.5	Double (F64)
enoff	DN switch off value; if DN = on and u > enoff then DN = off	⊙-0.5	Double (F64)

Chapter 8

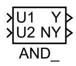
LOGIC – Logic control

Contents

AND – Logical product of two signals

Block Symbol

Licence: STANDARD



Function Description

The AND block computes the logical product of two input signals U1 and U2.

If you need to work with more input signals, use the ANDOCT block.

Inputs

U1	First logical input of the block	Bool
U2	Second logical input of the block	Bool

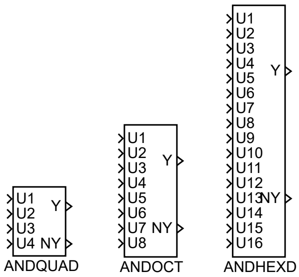
Outputs

Y	Output signal, logical product ($U1 \land U2$)	Bool
NY	Boolean complementation of Y ($NY = \neg Y$)	Bool

ANDQUAD, ANDOCT, ANDHEXD – Logical product of multiple signals

Block Symbols

Licence: STANDARD



Function Description

The ANDQUAD, ANDOCT and ANDHEXD blocks compute the logical product of up to sixteen input signals U1, U2, . . . , U16. The signals listed in the nl parameter are negated prior to computing the logical product.

For an empty nl parameter a simple logical product $Y = U1 \wedge U2 \wedge U3 \wedge U4 \wedge U5 \wedge U6 \wedge U7 \wedge U8$ is computed. For e.g. nl=1,3..5, the logical function is $Y = \neg U1 \wedge U2 \wedge \neg U3 \wedge \neg U4 \wedge \neg U5 \wedge U6 \wedge \ldots U16$.

If you have less than 4/8/16 signals, use the nl parameter to handle the unconnected inputs. If you have only two input signals, consider using the AND_ block.

Inputs

U1..U16	Logical inputs of the block	Bool

Outputs

Y	Result of the logical operation	Bool
NY	Boolean complementation of Y	Bool

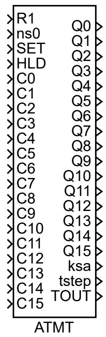
Parameter

nl	List of signals to negate. The format of the list is e.g. 1,3..5,8. Third-party programs (Simulink, OPC clients etc.) work with an integer number, which is a binary mask, i.e. 157 (binary 10011101) in the mentioned case.	Long (I32)

ATMT – Finite-state automaton

Block Symbol

Licence: STANDARD



Function Description

The `ATMT` block implements a finite state machine with at most 16 states and 16 transition rules.

The current state of the machine i, $i = 0, 1, \ldots, 15$ is indicated by the binary outputs `Q0`, `Q1`, ..., `Q15`. If the state i is active, the corresponding output is set to `Q`i=`on`. The current state is also indicated by the `ksa` output (`ksa` $\in \{0, 1, \ldots, 15\}$).

The transition conditions C_k, $k = 0, 1, \ldots, 15$ are activated by the binary inputs `C0`, `C1`, ..., `C15`. If `C`k = `on` the `k`-th transition condition is fulfilled. The transition cannot happen when `C`k = `off`.

The automat function is defined by the following table of transitions:

$$\begin{array}{ccc} S1 & C1 & FS1 \\ S2 & C2 & FS2 \\ & & \ldots \\ Sn & Cn & FSn \end{array}$$

Each row of this table represents one transition rule. For example the first row

$$S1 \quad C1 \quad FS1$$

has the meaning

`If` ($S1$ is the current state `AND` transition condition $C1$ is fulfilled)
`then` proceed to the following state $FS1$.

The above mentioned table can be easily constructed from the automat state diagram or SFC description (Sequential Function Charts, formerly Grafcet).

The `R1` = `on` input resets the automat to the initial state $S0$. The `SET` input allows manual transition from the current state to the `ns0` state when rising edge occurs. The

R1 input overpowers the SET input. The HLD = on input freezes the automat activity, the automat stays in the current state regardless of the Ci input signals and the tstep timer is not incremented. The TOUT output indicates that the machine remains in the given state longer than expected. The time limits TOi for individual states are defined by the touts array. There is no time limit for the given state when TOi is set to zero. The TOUT output is set to off whenever the automat changes its state.

It is possible to allow more state transitions in one cycle by the morestps parameter. However, this option must be thoroughly considered and tested, namely when the TOUT output is used in transition conditions. In such a case it is strongly recommended to incorporate the ksa output in the transition conditions as well.

The development tools of the REXYGEN system include also the SFCEditor program. You can create SFC schemes graphically using this tool. Run this editor from REXYGEN Studio by clicking the *Configure* button in the parameter dialog of the ATMT block.

Inputs

R1	Reset signal, R1 = on brings the automat to the initial state S0; the R1 input overpowers the SET input	Bool
ns0	This state is reached when rising edge occurs at the the SET input	Long (I32)
SET	The rising edge of this signal forces the transition to the ns0 state	Bool
HLD	The HLD = on freezes the automat, no transitions occur regardless of the input signals, tstep is not increasing	Bool
C0...C15	The transition conditions; Ci = on means that the i-th condition was fulfilled and the corresponding transition rule can be executed	Bool

Outputs

Q0...Q15	Output signals indicating the current state of the automat; the current state i is indicated by Qi = on	Bool
ksa	Integer code of the active state	Long (I32)
tstep	Time elapsed since the current state was reached; the timer is set to 0 whenever a state transition occurs	Double (F64)
TOUT	Flag indicating that the time limit for the current state was exceeded	Bool

Parameters

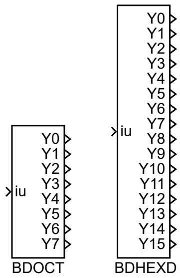
morestps	Allow multiple transitions in one cycle of the automat off ... Disabled on .... Enabled	Bool
sfcname	Filename of block configurator data file (filename is generated by system if parameter is empty)	String
STT	State transition table (matrix) ⊙[0 0 1; 1 1 2; 2 2 3; 3 3 0]	Byte (U8)

`touts`	Vector of timeouts $TO0$, $TO1$, …, $TO15$ for the states $S0$, $S1$, …, $S15$ ⊙`[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16]`	`Double (F64)`

BDOCT, BDHEXD – Bitwise demultiplexers

Block Symbols

Licence: STANDARD



Function Description

Both BDOCT and BDHEXD are bitwise demultiplexers for easy decomposition of the input signal to individual bits. The only difference is the number of outputs, the BDOCT block has 8 Boolean outputs while the BDHEXD block offers 16-bit decomposition. The output signals Y*i* correspond with the individual bits of the input signal iu, the Y0 output is the least significant bit.

Input

iu	Input signal to be decomposed		Long (I32)

Outputs

Y0...Y15	Individual bits of the input signal		Bool

Parameter

shift	Bit shift of the input signal	↓0 ↑31	Long (I32)
vtype	Input numeric type	⊙4	Long (I32)

-- ....
2 ..... Byte (U8)
3 ..... Short (I16)
4 ..... Long (I32)
5 ..... Word (U16)
6 ..... DWord (U32)
-- ....
-- ....
-- ....
10 .... Large (I64)

BITOP – Bitwise operation

Block Symbol

Licence: STANDARD

Function Description

The BITOP block performs bitwise operation i1 ∘ i2 on the signals i1 and i2, resulting in an integer output n. The type of operation is selected by the iop parameter described below. In case of logical negation or 2's complements the input i2 is ignored (i.e. the operation is unary).

Inputs

i1	First integer input of the block	Long (I32)
i2	Second integer input of the block	Long (I32)

Output

n	Result of the bitwise operation iop	Long (I32)

Parameter

iop   Bitwise operation   ⊙1   Long (I32)

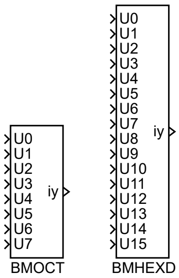
1 ..... Bitwise negation (Bit NOT)
2 ..... Bitwise logical sum (Bit OR)
3 ..... Bitwise logical product (Bit AND)
4 ..... Bitwise logical exclusive sum (Bit XOR)
5 ..... Shift of the i1 signal by i2 bits to the left (Shift Left)
6 ..... Shift of the i1 signal by i2 bits to the right (Shift Right)
7 ..... 2's complement of the i1 signal on 8 bits (2's Complement - Byte)
8 ..... 2's complement of the i1 signal on 16 bits (2's Complement - Word)
9 ..... 2's complement of the i1 signal on 32 bits (2's Complement - Long)

`vtype`	Input and output numeric type `--` .... `2` ..... Byte (U8) `3` ..... Short (I16) `4` ..... Long (I32) `5` ..... Word (U16) `6` ..... DWord (U32) `--` .... `--` .... `--` .... `10` .... Large (I64)	⊙4	`Long (I32)`

BMOCT, BMHEXD – Bitwise multiplexers

Block Symbols

Licence: STANDARD



Function Description

Both BMOCT and BMHEXD are bitwise multiplexers for easy composition of the output signal from individual bits. The only difference is the number of inputs, the BMOCT block has 8 Boolean inputs while the BMHEXD block offers 16-bit composition. The input signals Ui correspond with the individual bits of the output signal iy, the U0 input is the least significant bit.

Inputs

U0...U15	Individual bits of the output signal	Bool

Output

iy	Composed output signal	Long (I32)

Parameter

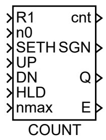
shift	Bit shift of the output signal	↓0 ↑31	Long (I32)
vtype	Output numeric type	⊙4	Long (I32)

-- ....
2 ..... Byte (U8)
3 ..... Short (I16)
4 ..... Long (I32)
5 ..... Word (U16)
6 ..... DWord (U32)
-- ....
-- ....
-- ....
10 .... Large (I64)

COUNT – Controlled counter

Block Symbol

Licence: STANDARD



Function Description

The COUNT block is designed for bidirectional pulse counting – more precisely, counting rising edges of the UP and DN input signals. When a rising edge occurs at the UP (DN) input, the cnt output is incremented (decremented) by 1. Simultaneous occurrence of rising edges at both inputs is indicated by the error output E set to on. The R1 input resets the counter to 0 and no addition or subtraction is performed unless the R1 input returns to off again. It is also possible to set the output cnt to the value n0 by the SETH input. Again, no addition or subtraction is performed unless the SETH input returns to off again. The R1 input has higher priority than the SETH input. The input HLD = on prevents both incrementing and decrementing. When the counter reaches the value cnt $\geq$ nmax, the Q output is set to on.

Inputs

R1	Block reset (R1 = on)	Bool
n0	Value to set the counter to (using the SETH input)	Long (I32)
SETH	Set the counter value to n0 (SETH = on)	Bool
UP	Incrementing input signal	Bool
DN	Decrementing input signal	Bool
HLD	Counter freeze off . . . Counter is running on . . . . Counter is locked	Bool
nmax	Counter target value	Long (I32)

Outputs

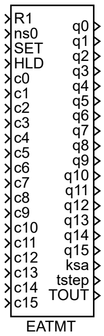
cnt	Total number of pulses	Long (I32)
SGN	Sign of the cnt output off . . . for cnt < 0 on . . . . for cnt ≥ 0	Bool
Q	Target value indicator off . . . for cnt < nmax on . . . . for cnt $\geq$ nmax	Bool

`E`	Indicator of simultaneous occurrence of rising edges at both inputs `UP` and `DN`	`Bool`

EATMT – Extended finite-state automaton

Block Symbol

Licence: ADVANCED



Function Description

The EATMT block implements a finite automat with at most 256 states and 256 transition rules, thus it extends the possibilities of the ATMT block.

The current state of the automat i, $i = 0, 1, \ldots, 255$ is indicated by individual bits of the integer outputs q0, q1, ..., q15. Only a single bit with index $i \,\mathrm{MOD}\, 16$ of the q($i \,\mathrm{DIV}\, 16$) output is set to 1. The remaining bits of that output and the other outputs are zero. The bits are numbered from zero, least significant bit first. Note that the DIV and MOD operators denote integer division and remainder after integer division respectively. The current state is also indicated by the ksa $\in \{0, 1, \ldots, 255\}$ output.

The transition conditions C_k, $k = 0, 1, \ldots, 255)$ are activated by individual bits of the inputs c0, c1, ..., c15. The k-th transition condition is fulfilled when the ($k \,\mathrm{MOD}\, 16$)-th bit of the input c($k \,\mathrm{DIV}\, 16$) is equal to 1. The transition cannot happen otherwise.

The BMHEXD or BMOCT bitwise multiplexers can be used for composition of the input signals c0, c1, ..., c15 from individual Boolean signals. Similarly the output signals q0, q1, ..., q15 can be decomposed using the BDHEXD or BDOCT bitwise demultiplexers.

The automat function is defined by the following table of transitions:

$$\begin{array}{ccc} S1 & C1 & FS1 \\ S2 & C2 & FS2 \\ & & \ldots \\ Sn & Cn & FSn \end{array}$$

Each row of this table represents one transition rule. For example the first row

$$S1 \quad C1 \quad FS1$$

has the meaning

If ($S1$ is the current state AND transition condition $C1$ is fulfilled)
then proceed to the following state $FS1$.

The above described meaning of the table row holds for $C1 < 1000$. Negation of the $(C1 - 1000)$-th transition condition is assumed for $C1 \geq 1000$.

The above mentioned table can be easily constructed from the automat state diagram or SFC description (Sequential Function Charts, formerly Grafcet).

The R1 = on input resets the automat to the initial state $S0$. The SET input allows manual transition from the current state to the ns0 state when rising edge occurs. The R1 input overpowers the SET input. The HLD = on input freezes the automat activity, the automat stays in the current state regardless of the ci input signals and the tstep timer is not incremented. The TOUT output indicates that the machine remains in the given state longer than expected. The time limits TOi for individual states are defined by the touts array. There is no time limit for the given state when TOi is set to zero. The TOUT output is set to off whenever the automat changes its state.

It is possible to allow more state transitions in one cycle by the morestps parameter. However, this option must be thoroughly considered and tested, namely when the TOUT output is used in transition conditions. In such a case it is strongly recommended to incorporate the ksa output in the transition conditions as well.

The development tools of the REXYGEN system include also the SFCEditor program. You can create SFC schemes graphically using this tool. Run this editor from REXYGEN Studio by clicking the *Configure* button in the parameter dialog of the EATMT block.

Inputs

R1	Reset signal, R1 = on brings the automat to the initial state S0; the R1 input overpowers the SET input	Bool
ns0	This state is reached when rising edge occurs at the the SET input	Long (I32)
SET	The rising edge of this signal forces the transition to the ns0 state	Bool
HLD	The HLD = on freezes the automat, no transitions occur regardless of the input signals, tstep is not increasing	Bool
c0...c15	Transition conditions, each input signal contains 16 transition conditions, see details above	

Outputs

q0...q15	Output signals indicating the current state of the automat, see details above	Long (I32)
ksa	Integer code of the active state	Long (I32)
tstep	Time elapsed since the current state was reached; the timer is set to 0 whenever a state transition occurs	Double (F64)
TOUT	Flag indicating that the time limit for the current state was exceeded	Bool

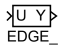
Parameters

`morestps`	Allow multiple transitions in one cycle of the automat `off` ... Disabled `on` .... Enabled	`Bool`
`sfcname`	Filename of block configurator data file (filename is generated by system if parameter is empty)	`String`
`STT`	State transition table (matrix) ⊙`[0 0 1; 1 1 2; 2 2 3; 3 3 0]`	`Short (I16)`
`touts`	Vector of timeouts `T00`, `T01`, ..., `T0255` for the states $S0$, $S1$, ..., $S255$ ⊙`[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16]`	`Double (F64)`

EDGE – Falling/rising edge detection in a binary signal

Block Symbol

Licence: STANDARD



Function Description

The EDGE block detects rising (off→on) and/or falling (on→off) edges in the binary input signal U. The type of edges to detect is determined by the iedge parameter. As long as the input signal remains constant, the output Y is off. In the case when an edge corresponding with the iedge parameter is detected, the output Y is set to on for one sampling period.

Input

U	Logical input of the block	Bool

Output

Y	Logical output of the block	Bool

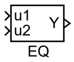
Parameter

iedge	Type of edges to detect 1 ..... Rising edge 2 ..... Falling edge 3 ..... Both edges	⊙1	Long (I32)

EQ – Equivalence of two signals

Block Symbol

Licence: STANDARD



Function Description

The block compares two input signals and Y=on is set if both signals have the same value. Both signals must be either of a numeric type or strings. A conversion between numeric types is performed: for example 2.0 (double) and 2 (long) are evaluated as equivalent. Comparison of matrices or other complex types is not supported.

Inputs

u1	Block input signal	Any
u2	Block input signal	Any

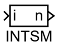
Output

Y	Output signal	Bool
NY	Boolean complementation of Y (NY $= \neg$Y)	Bool

INTSM – Integer number bit shift and mask

Block Symbol

Licence: STANDARD



Function Description

The `INTSM` block performs bit shift of input value `i` by `shift` bits right (if `shift` is positive) or left (if `shift` is negative). Free space resulting from shifting is filled with zeros.

Output value `n` is calculated as bitwise AND of shifted input `i` and bit mask `mask`.

Typical application of this block is extraction of one or more adjacent bits from a given position in integer register which was read from some external system.

Input

`i`	Integer value to shift and mask	↓-9.22337E+18 ↑9.22337E+18	Large (I64)

Parameters

`shift`	Bit shift (negative=left, positive=right)	↓-63 ↑63	Long (I32)
`mask`	Bit mask (applied after bit shift)	↓0 ↑4294970000 ⊙4294967295	Large (I64)
`vtype`	Output numeric type	⊙4	Long (I32)

-- ....
2 ..... Byte (U8)
3 ..... Short (I16)
4 ..... Long (I32)
5 ..... Word (U16)
6 ..... DWord (U32)
-- ....
-- ....
-- ....
10 .... Large (I64)

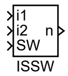
Output

`n`	Resulting integer value	Large (I64)

ISSW – Simple switch for integer signals

Block Symbol

Licence: STANDARD



Function Description

The ISSW block is a simple switch for integer input signals i1 and i2 whose decision variable is the binary input SW. If SW is off, then the output n is equal to the i1 signal. If SW is on, then the output n is equal to the i2 signal.

Inputs

i1	First integer input of the block	Long (I32)
i2	Second integer input of the block	Long (I32)
SW	Signal selector off . . . The i1 signal is selected on . . . . The i2 signal is selected	Bool

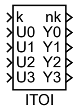
Output

n	Integer output of the block	Long (I32)

ITOI – Transformation of integer and binary numbers

Block Symbol

Licence: STANDARD



Function Description

The ITOI block transforms the input number k, or the binary number $(\texttt{U3 U2 U1 U0})_2$, from the set $\{0, 1, 2, \ldots, 15\}$ to the output number nk and its binary representation $(\texttt{Y3 Y2 Y1 Y0})_2$ from the same set. The transformation is described by the following table

k	0	1	2	...	15
nk	n0	n1	n2	...	n15

where n0, ..., n15 are given by the mapping table target vector fktab.

If BINF = off, then the integer input k is active, while for BINF = on the input is defined by the binary inputs $(\texttt{U3 U2 U1 U0})_2$.

Inputs

k	Integer input of the block	Long (I32)
U0	Binary input digit, weight of 1	Bool
U1	Binary input digit, weight of 2	Bool
U2	Binary input digit, weight of 4	Bool
U3	Binary input digit, weight of 8	Bool

Outputs

nk	Integer output of the block	Long (I32)
Y0	Binary output digit, weight of 1	Bool
Y1	Binary output digit, weight of 2	Bool
Y2	Binary output digit, weight of 4	Bool
Y3	Binary output digit, weight of 8	Bool

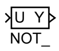
Parameters

BINF	Enable the binary selectors off ... Disabled (integer input k) on .... Enabled (binary input signals U3...U0)	⊙on	Bool
fktab	Vector of mapping table target values ⊙[0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15]		Byte (U8)

NOT – Boolean complementation

Block Symbol

Licence: STANDARD



Function Description

The NOT block negates the input signal.

Input

U	Logical input of the block	Bool

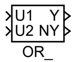
Output

Y	Logical output of the block ($Y = \neg U$)	Bool

OR – Logical sum of two signals

Block Symbol

Licence: STANDARD



Function Description

The OR block computes the logical sum of two input signals U1 and U2.

If you need to work with more input signals, use the OROCT block.

Inputs

U1	First logical input of the block	Bool
U2	Second logical input of the block	Bool

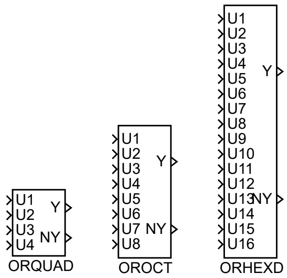
Outputs

Y	Logical output of the block ($U1 \vee U2$)	Bool
NY	Boolean complementation of Y ($NY = \neg Y$)	Bool

ORQUAD, OROCT, ORHEXD – Logical sum of multiple signals

Block Symbols

Licence: STANDARD



Function Description

The **ORQUAD**, **OROCT** and **ORHEXD** blocks compute the logical sum of up to sixteen input signals U1, U2, ..., U16. The signals listed in the **nl** parameter are negated prior to computing the logical sum.

For an empty **nl** parameter a simple logical sum $\mathtt{Y} = \mathtt{U1} \lor \mathtt{U2} \lor \mathtt{U3} \lor \mathtt{U4} \lor \mathtt{U5} \lor \mathtt{U6} \lor \mathtt{U7} \lor \ldots \lor \mathtt{U16}$ is computed. For e.g. nl=1,3..5, the logical function is $\mathtt{Y} = \neg\mathtt{U1} \lor \mathtt{U2} \lor \neg\mathtt{U3} \lor \neg\mathtt{U4} \lor \neg\mathtt{U5} \lor \mathtt{U6} \lor \ldots \lor \mathtt{U16}$.

If you have only two input signals, consider using the OR_ block.

Inputs

U1..U16	Logical inputs of the block	Bool

Outputs

Y	Result of the logical operation	Bool
NY	Boolean complementation of Y	Bool

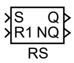
Parameter

nl	List of signals to negate. The format of the list is e.g. 1,3..5,8. Third-party programs (Simulink, OPC clients etc.) work with an integer number, which is a binary mask, i.e. 157 (binary 10011101) in the mentioned case.	Long (I32)

RS – Reset-set flip-flop circuit

Block Symbol

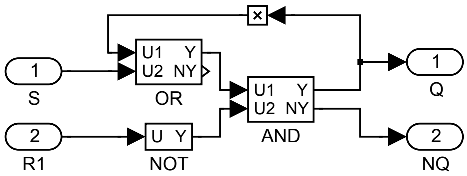
Licence: STANDARD



Function Description

The RS block is a flip-flop circuit, which sets its output permanently to on as soon as the input signal S is equal to on. The other input signal R1 resets the Q output to off even if the S input is on. The NQ output is simply the negation of the signal Q.

The block function is evident from the inner block structure depicted below.



Inputs

S	Flip-flop set, sets the Q output to on	Bool
R1	Priority flip-flop reset, sets the Q output to off, overpowers the S input	Bool

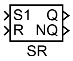
Outputs

Q	Flip-flop circuit state	Bool
NQ	Boolean complementation of Q	Bool

SR – Set-reset flip-flop circuit

Block Symbol

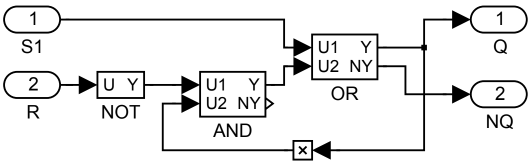
Licence: STANDARD



Function Description

The SR block is a flip-flop circuit, which sets its output permanently to on as soon as the input signal S1 is on. The other input signal R resets the Q output to off, but only if the S1 input is off. The NQ output is simply the negation of the signal Q.

The block function is evident from the inner block structure depicted below.



Inputs

S1	Priority flip-flop set, sets the Q output to on, overpowers the R input	Bool
R	Flip-flop reset, sets the Q output to off	Bool

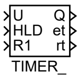
Outputs

Q	Flip-flop circuit state	Bool
NQ	Boolean complementation of Q	Bool

TIMER – Multipurpose timer

Block Symbol

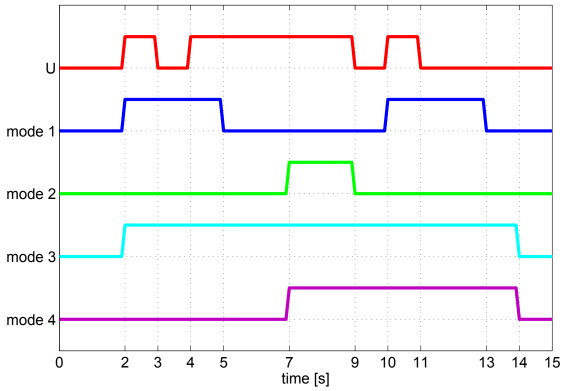
Licence: STANDARD



Function Description

The TIMER block either generates an output pulse of the given width pt (in seconds) or filters narrow pulses in the U input signal whose width is less than pt seconds. The operation mode is determined by the mode parameter.

The graph illustrates the behaviour of the block in individual modes for pt = 3:



The timer can be paused by the HLD input. The R1 input resets the timer. The reset signal overpowers the U input.

Inputs

U	Trigger of the timer	Bool
HLD	Timer hold	Bool
R1	Block reset (R1 = on)	Bool

Outputs

Q	Timer output	Bool
et	Elapsed time [s]	Double (F64)
rt	Remaining time [s]	Double (F64)

Parameters

`mode`	Timer mode	⊙1	`Long (I32)`
	`1` ..... Pulse – an output pulse of the length `pt` is generated upon rising edge at the `U` input. All input pulses during the generation of the output pulse are ignored. `2` ..... Delayed ON – the input signal `U` is copied to the `Q` output, but the start of the pulse is delayed by `pt` seconds. Any pulse shorter than `pt` is does not pass through the block. `3` ..... Delayed OFF – the input signal `U` is copied to the `Q` output, but the end of the pulse is delayed by `pt` seconds. If the break between two pulses is shorter than `pt`, the output remains `on` for the whole time. `4` ..... Delayed change – the `Q` output is set to the value of the `U` input no sooner than the input remains unchanged for `pt` seconds		
`pt`	Timer interval [s]	⊙1.0	`Double (F64)`

Chapter 9

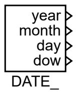
TIME – Blocks for handling time

Contents

DATE – Current date

Block Symbol

Licence: STANDARD



Function Description

The outputs of the DATE function block correspond with the actual date of the operating system. Use the DATETIME block for advanced operations with date and time.

Outputs

year	Year	Long (I32)
month	Month	Long (I32)
day	Day	Long (I32)
dow	Day of week, first day of week is Sunday (1)	Long (I32)

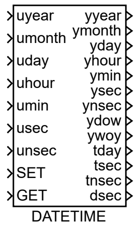
Parameter

tz	Timezone 1 ..... Local time 2 ..... UTC	⊙1	Long (I32)

DATETIME – Get, set and convert time

Block Symbol

Licence: STANDARD



Function Description

The DATETIME block is intended for advanced date/time operations in the REXYGEN system.

It allows synchronization of the operating system clock and the clock of the REXYGEN system. When the executive of the REXYGEN system is initialized, both clocks are the same. But during long-term operation the clocks may loose synchronization (e.g. due to daylight saving time). If re-synchronization is required, the rising edge (off→on) at the SET input adjusts the clock of the REXYGEN system according to the block inputs and parameters.

It is highly recommended not to adjust the REXYGEN system time when the controlled machine/process is in operation. Unexpected behavior might occur.

If date/time reading or conversion is required, the rising edge (off→on) at the GET input triggers the action and the block outputs are updated. The outputs starting with 't' denote the total number of respective units since January 1st, 2000 UTC.

Both reading and adjusting clock can be repeated periodically if set by getper and setper parameters.

If the difference of the two clocks is below the tolerance limit settol, the clock of the REXYGEN system is not adjusted at once, a gradual synchronization is used instead. In such a case, the timing of the REXYGEN system executive is negligibly altered and the clocks synchronization is achieved after some time. Afterwards the timing of the REXYGEN executive is reverted back to normal.

For simple date/time reading use the DATE_ and TIME function blocks.

Inputs

uyear	Input for setting year	Long (I32)
umonth	Input for setting month	Long (I32)
uday	Input for setting day	Long (I32)
uhour	Input for setting hours	Long (I32)
umin	Input for setting minutes	Long (I32)

usec	Input for setting seconds		Long (I32)
unsec	Input for setting nanoseconds	↓-9.22E+18 ↑9.22E+18	Large (I64)
SET	Trigger for setting time		Bool
GET	Trigger for getting time		Bool

Outputs

yyear	Year	Long (I32)
ymonth	Month	Long (I32)
yday	Day	Long (I32)
yhour	Hours	Long (I32)
ymin	Minutes	Long (I32)
ysec	Seconds	Long (I32)
ynsec	Nanoseconds	Long (I32)
ydow	Day of week	Long (I32)
ywoy	Week of year	Long (I32)
tday	Total number of days	Long (I32)
tsec	Total number of seconds	Long (I32)
tnsec	Total number of nanoseconds	Large (I64)
dsec	Number of seconds since midnight	Long (I32)

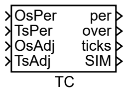
Parameters

isetmode	Source for setting time 1 ..... OS clock 2 ..... Block inputs 3 ..... The unsec input 4 ..... The usec input 5 ..... The unsec input, relative	⊙1	Long (I32)
igetmode	Source for getting or converting time 1 ..... OS clock 2 ..... Block inputs 3 ..... The unsec input 4 ..... The usec input 5 ..... The uday input 6 ..... REXYGEN clock	⊙6	Long (I32)
settol	Tolerance for setting the REXYGEN clock [s]	⊙1.0	Double (F64)
setper	Period for setting time [s] (0=not periodic)		Double (F64)
getper	Period for getting time [s] (0=not periodic)	⊙0.001	Double (F64)
FDOW	First day of week is Sunday off ... Week starts on Monday on .... Week starts on Sunday		Bool
tz	Timezone 1 ..... Local time 2 ..... UTC	⊙1	Long (I32)

TC – Timer control and status

Block Symbol

Licence: STANDARD



Function Description

The TC function block controls the internal timer of REXYGEN. It is possible to modify the actual basic tick period (e.g. the value set in the tick parameter of the EXEC block) or logical tick period (e.g. the time added to the timestamp of each tick if timer = CORETIMER is selected). By default, the logical and physical period is the same and is the EXEC:tick parameter. The discretization period of the blocks in the control algorithm is not affected by the TC block.

The actual period can be changed in two ways: set the desired value to the OsPer input or set OsAdj for one tick. OsAdj will temporarily increase or decrease the actual period until the total shift set on the OsAdj input is realized. How much the period increases is controlled by the OsMax parameter.

Example: Let's expect the tick period to be 0.1s and OsMax=0.2, so let's set OsAdj=1.0 to temporarily increase the real period to 0.12s (e.g. 20% defined in the OsMax parameter) until a total shift of 1s is realized, e.g. for 50 ticks.

Logical period control is the same using inputs/parameter TsPer, TsAdj, TsMax.

Note 1: The unconnected input or the input with a value of 0 is ignored.

Note 2: The actual period adjustment is not supported on Windows targets.

Note 3: The primary reason for this block is to synchronize with another controller in time-critical application, so the period should only be changed by a few percent. It is also possible to dramatically change the actual period to slow down or speed up the execution (for debugging and simulation reasons), but in this case some warning about missed tick or incorrect period could appear.

Inputs

OsPer	Physical tick period [s]	Double (F64)
TsPer	Logical (timestamp) tick period [s]	Double (F64)
OsAdj	Physical tick shift [s]	Double (F64)
TsAdj	Logical (timestamp) tick shift [s]	Double (F64)

Parameters

OsMax	Maximal relative quantum for physical adjustment ↓0.0 ↑1.0 ⊙0.1	Double (F64)

TsMax	Maximal relative quantum for logical adjustment ↓0.0 ↑1.0 ⊙0.1	Double (F64)

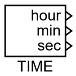
Outputs

per	Last physical tick period [s]	Double (F64)
over	Number of lost periods in the last tick	Long (I32)
ticks	Number of ticks since start	Large (I64)
SIM	Timer in simulation mode	Double (F64)

TIME – Current time

Block Symbol

Licence: STANDARD



Function Description

The outputs of the TIME function block correspond with the actual time of the operating system. Use the DATETIME block for advanced operations with date and time.

Outputs

hour	Hours	Long (I32)
min	Minutes	Long (I32)
sec	Seconds	Long (I32)

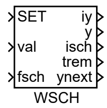
Parameter

tz	Timezone 1 ..... Local time 2 ..... UTC	⊙1	Long (I32)

WSCH – Weekly schedule

Block Symbol

Licence: STANDARD



Function Description

The WSCH function block is a weekly scheduler for e.g. heating (day, night, eco), ventilation (high, low, off), lighting, irrigation etc. Its outputs can be used for switching individual appliances on/off or adjusting the intensity or power of the connected devices.

During regular weekly schedule the outputs iy and y reflect the values from the wst table. This table contains triplets *day-hour-value*. E.g. the notation [2 6.5 21.5] states that on Tuesday, at 6:30 in the morning (24-hour format), the output y will be set to 21.5. The output iy will be set to 22 (rounding to nearest integer). The individual triplets are separated by semicolons.

The days in a week are numbered from 1 (Monday) to 7 (Sunday). Higher values can be used for special daily schedules, which can be forced using the fsch input or the specdays table. The active daily program is indicated by the isch output.

Alternatively it is possible to temporarily force a specific output value using the val input and a rising edge at the SET input (off→on). When a rising edge occurs at the SET input, the val input is copied to the y output and the isch output is set to 0. The forced value remains set until:

- the next interval as defined by the wst table, or
- another rising edge occurs at the SET input, or
- a different daily schedule is forced using the fsch input.

The list of special days (specdays) can be used for forcing a special daily schedule at given dates. E.g. you can force a Sunday daily schedule on holidays, Christmas, New Year, etc. The date is entered in the YYYYMMDD format. The notation [20160328 7] thus means that on March 28th, 2016, the Sunday daily schedule should be used. Individual pairs are separated by semicolons.

The trem and ynext outputs can be used for triggering specific actions in advance, before the y and iy are changed.

The iy output is meant for direct connection to function blocks with Boolean inputs (the conversion from type long to type bool is done automatically).

The nmax parameter defines memory allocation for the wst and specdays arrays. For nmax = 100 the wst list can contain up to 100 triplets *day-hour-value*. In typical applications there is no need to modify the nmax parameter.

Inputs

SET	Trigger for setting the y and iy outputs	Bool
val	Temporary value to set the y and iy outputs to	Double (F64)
fsch	Forced schedule 0 ..... standard weekly schedule 1 ..... Monday 2 ..... Tuesday .... 7 ..... Sunday 8 and above additional daily programs as defined by the wst table	Long (I32)

Outputs

iy	Integer output value	Long (I32)
y	Output value	Double (F64)
isch	Daily schedule identifier	Long (I32)
trem	Time remaining in the current section (in seconds)	Double (F64)
ynext	Output in the next section	Double (F64)

Parameters

tz	Timezone 1 ..... Local time 2 ..... UTC	⊙1	Long (I32)
nmax	Allocated size of arrays	↓10 ↑1000000 ⊙100	Long (I32)
wst	Weekly schedule table (list of triplets *day-hour-value*) ⊙[1 0.01 18.0; 2 6.0 22.0; 2 18.0 18.0; 3 6.0 22.0; 3 18.0 18.0; 4 6.0 22.0; 4 18.0 18.0		Double (F64)
specdays	List of special days (list of pairs *date-daily program*) ⊙[20150406 1; 20151224 1; 20151225 1; 20151226 1; 20160101 1; 20160328 1; 20170417 1; 20		Long (I32)

Chapter 10

ARC – Data archiving

Contents

The RexCore executive of the REXYGEN system consists of various interconnected subsystems (real-time subsystem, diagnostic subsystem, drivers subsystem, etc.). One of these subsystems is the *archiving subsystem*.

The archiving subsystem takes care of recording the history of the control algorithm. The first chapter describes the functionality of the archiving subsystem while the subsequent chapters describe the function blocks of the REXYGEN system.

The function blocks can be divided into groups by their use:

- Blocks for generating alarms and events
- Blocks for recording trends
- Blocks for handling archives

10.1 Functionality of the archiving subsystem

The archive in the REXYGEN system stores the history of events, alarms and trends of selected signals. There can be up to 15 archives in each target device. The types or archives are listed below:

RAM memory archive – Suitable for short-term data storage. The data access rate is very high but the data is lost on reboot.

Archive in a backed-up memory – Similar to the RAM archive but the data is not lost on restart. Data can be accessed fast but the capacity is usually quite limited (depends on the target platform).

Disk archive The disk archives are files in a proprietary binary format. The files are easily transferrable among individual platforms and the main advantage is the size, which is limited only by the capacity of the storage medium. On the other hand, the drawback is the relatively slow data access.

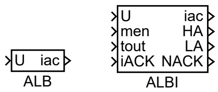
Not all hardware platforms support all types of archives. The individual types which are supported by the platform can be displayed in REXYGEN Studio in the Diagnostics tree view panel after clicking on the name of the target device (IP address). The supported types are listed in the lower left part of the `Target` tab.

10.2 Generating alarms and events

ALB, ALBI – Alarms for Boolean value

Block Symbols

Licence: STANDARD



Function Description

The ALB and ALBI blocks generate alarms or events when a Boolean input signal U changes. The men parameter selects whether the rising or falling or both edges in the input signal should be indicated. The iac output shows the current alarm (event) code.

The ALBI block is an extension of the ALB block as the alarms (events) are indicated also by Boolean output signals HA, LA and NACK. The type of edges to watch is selected by the men input signal and the alarms are acknowledged by the iACK input signal instead of parameters with the same name and meaning.

The events and alarms are differentiated by the lvl parameter in the REXYGEN system. The range $1 \leq \texttt{lvl} \leq 127$ is reserved for alarms. All starts, ends and acknowledgements of the alarms are stored in the archive. On the contrary, the range $128 \leq \texttt{lvl} \leq 255$ indicates events. Only the start (the time instant) of the event is stored in the archive.

Note: The input (parameter) iACK is set back to 0 immediately by the block algorithm. The funcionality is similar to the parameter BSTATE of the block MP.

Inputs

U	Logical input of the block whose changes are watched		Bool
men	Enable alarms 0 ..... All alarms disabled 1 ..... Low-alarm enabled (LA) (falling edge in the input signal U) 2 ..... High-alarm enabled (HA)(rising edge in the input signal U) 3 ..... All alarms enabled		Long (I32)
tout	Alarm activation delay time [s]	↓0.0	Double (F64)
iACK	Acknowledge alarm 1 ..... Low-alarm acknowledge 2 ..... High-alarm acknowledge 3 ..... Both alarms acknowledge Alarm is acknowledged on rising edge		Byte (U8)

Outputs

`iac`	Current alarm code `0` ..... All alarms inactive `1` ..... Low-alarm active (`LA`) `2` ..... High-alarm active (`HA`) `256` ... Low-alarm not acknowledged (`NACK`) `512` ... High-alarm not acknowledged (`NACK`)	`Long (I32)`
`HA`	High-alarm indicator	`Bool`
`LA`	Low-alarm indicator	`Bool`
`NACK`	Alarm-not-acknowledged indicator	`Bool`

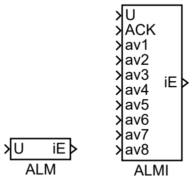
Parameters

`arc`	List of archives to store the events. The format of the list is e.g. `1,3..5,8`. The event will be stored in all listed archives (see the `ARC` block for details on archives numbering). Third-party programs (Simulink, OPC clients etc.) work with an integer number, which is a binary mask, i.e. `157` (binary `10011101`) in the mentioned case.	`Word (U16)`
`id`	Identification code of the alarm in the archive. This identifier must be unique in the whole target device with the REXYGEN control system (i.e. in all archiving blocks). Disabled for `id` = 0. ⊙1	`Word (U16)`
`lvl`	The level of the alarms (`HA` and `LA`) which differentiates alarms from events and defines the severity of the alarm/event ↓1 ⊙1	`Byte (U8)`
`Desc`	Extended description of the alarm which is displayed by the diagnostic tools of the REXYGEN system ⊙`Alarm Description`	`String`

ALM, ALMI – Alarm activation

Block Symbols

Licence: STANDARD



Function Description

The ALM and ALMI block is used to generate an alarm. The alarm is active when the input U=on. The alarm must be defined by the ALARMS block and is uniquely identified using the id parameter. Active alarms can be displayed by special component in HMI. Status change of the alarm and its acknowledgment is also stored in the archive (if defined in configuration in the ALARMS block). The alarm can be acknowledged by setting the parameter ACK=on.

Remark: The system displays the acknowledgment status, allows the user to acknowledge the ongoing alarm and can display the acknowledgment status in the HMI and in the archive. Acknowledging alarms has no further meaning for REXYGEN itself and nothing depends on it. Whether alarms will be acknowledged depends on the filter settings in the visualization and the design of the entire system.

Inputs

U	Alarm is active when U = on	Bool

Outputs

iE	Error code	Error

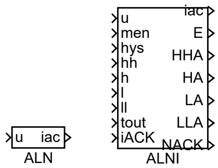
Parameters

id	Unique identification number of the alarm. Alarm is disabled for id = 0. Value id = −1 means that alarm is identified by name (e.g. name of the block without ALM(I) prefix) not by id. ↓-1 ↑65535 ⊙-1	Long (I32)
ACK	Set ACK = on acknowledge alarm. The algorithm immediately reverts the parameter to ACK = off.	Bool
av1..av8	Value associated with alarm. See description of the ALARMS block to clarify how this value is propagate into alarm description. ↓1.79769e+308 ⊙-1.79769e+308	Double (F64)

ALN, ALNI – Alarms for numerical value

Block Symbols

Licence: STANDARD



Function Description

The ALN and ALNI blocks generate two-level alarms or events when a limit value is exceeded (or not reached). There are four limit values the input signal u is compared to, namely high-limits h and hh and low-limits l and ll. The iac output shows the current alarm (event) code.

The ALNI block is an extension of the ALN block as the alarms (events) are indicated also by Boolean output signals HHA, HA, LA and LLA and the variables of the alarm algorithm are given by the input signals hys, hh, h, l and ll instead of parameters with the same name and meaning.

The events and alarms are differentiated by the lvl parameter in the REXYGEN system. The range $1 \leq \texttt{lvl} \leq 127$ is reserved for alarms. All starts, ends and acknowledgements of the alarms are stored in the archive. On the contrary, the range $128 \leq \texttt{lvl} \leq 255$ indicates events. Only the start (the time instant) of the event is stored in the archive.

Note 1: The input (parameter) iACK is set back to 0 immediately by the block algorithm. The functionality is similar to the parameter BSTATE of the block MP.

Note2: The parameter Desc can include formatting characters (multilingual texts, associated variables). Formatting rules are described in the ALARMS block.

Inputs

u	Analog input of the block which is checked to remain within the given limits		Double (F64)
hys	Alarm hysteresis for switching the alarm off	↓1e-10 ↑1e+10	Double (F64)
hh	The second high-alarm limit, must be greater than h		Double (F64)
h	High-alarm limit, must be greater than l		Double (F64)
l	Low-alarm limit, must be greater than ll		Double (F64)
ll	The second low-alarm limit		Double (F64)
tout	Alarm activation delay time [s]	↓0.0	Double (F64)
iACK	Alarm is acknowledged on rising edge of the individual bits of this input/parameter. E.g. value 15 acknowledges all alarms.		

Byte (U8)
- 1 ..... Second low-alarm acknowledge
- 2 ..... Low-alarm acknowledge
- 4 ..... High-alarm acknowledge
- 8 ..... Second high-alarm acknowledge

In case a one-level alarm is required, it is sufficient to set `lvl2=0` or set the `hh` and `ll` limits to extreme values which can never be reached by the input signal.

Outputs

`iac`	Current alarm code. Additional bitwise combinations of the codes may appear. E.g. `12` means both high alarms. `0` ..... Signal within limits `1` ..... Low-alarm active `2` ..... High-alarm active `4` ..... Second low-alarm active `8` ..... Second high-alarm active `256` ... Low-alarm not acknowledged `512` ... High-alarm not acknowledged `1024` .. Second low-alarm not acknowledged `2048` .. Second high-alarm not acknowledged `-1` .... Invalid alarm limits	Long (I32)
`E`	Error flag `off` ... No error `on` .... An error occurred, alarm limits disordered	Bool
`HHA`	The second high-alarm indicator	Bool
`HA`	High-alarm indicator	Bool
`LA`	Low-alarm indicator	Bool
`LLA`	The second low-alarm indicator	Bool
`NACK`	Alarm-not-acknowledged indicator	Bool

Parameters

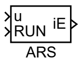
`acls`	Alarm class (data type to store) `1` ..... Bool `2` ..... Byte (U8) `3` ..... Short (I16) `4` ..... Long (I32) `5` ..... Word (U16) `6` ..... DWord (U32) `7` ..... Float (F32) `8` ..... Double (F64) `9` ..... Large (I64)	⊙8	Byte (U8)
`arc`	List of archives to store the events. The format of the list is e.g. `1,3..5,8`. The event will be stored in all listed archives (see the `ARC` block for details on archives numbering). Third-party programs (Simulink, OPC clients etc.) work with an integer number, which is a binary mask, i.e. `157` (binary `10011101`) in the mentioned case.		Word (U16)

`id`	Identification code of the alarm in the archive. This identifier must be unique in the whole target device with the REXYGEN control system (i.e. in all archiving blocks). Disabled for `id` = 0. ⊙1	`Word (U16)`
`lvl1`	The level of first high- and low-alarms (`HA` and `LA`) which differentiates alarms from events and defines the severity of the alarm/event ↓1 ⊙1	`Byte (U8)`
`lvl2`	The level of second high- and low-alarms (`HHA` and `LLA`) which differentiates alarms from events and defines the severity of the alarm/event ↓1 ⊙10	`Byte (U8)`
`Desc`	Extended description of the alarm which is displayed by the diagnostic tools of the REXYGEN system ⊙`Alarm Description`	`String`

ARS – Archive store value

Block Symbol

Licence: STANDARD



Function Description

The block allow to store value into archive subsystem. Written value must be connected to the `u` input. Value could be simple like bool, int or float, string or matrix/vector. Type of value must be set by the `type` parameter. The the parameter codetype=13:Reference must be set for vector or matrix. There is one archive item for each column of the matrix. Data are stored only if the input `RUN=on` is set. The parameter `subtype` allow write alarm type that write other alarm blocks (for example `L->H` for bool alarm, `HiHi` for numeric alarm). the value of this parameter is in range 0 to 7 and is not used in vector/matrix items. This parameter is usualy not needed.

Note 1: The archive subsystem is limited for 255 values, but no more then 512 bytes in one archive item (e.g. 128 values of type Short, 64 values of type Long, 32 values of type Double). Vector (matrix's column) is truncated to this size and stored into archive and no error nor warning is indicated, if the input array is bigger.

Note 2: The string value is limited to 65535 byte (i.e. characters if only characters from english keyboard is used; UTF-8 encoding is used). String is truncated to this size and stored into archive and no error nor warning is indicated, if the input string is bigger. It is recomended to not overcome 4000 bytes, because some reading functions has limited buffer and could failed for long strings.

Note 3: The parameter `id` is intended as bind source block (and also source signal) with item in archive (and with alarm subsystem in same cases). So REXYGENcheck unique this binding. The `ARS` block is intend to be low-level-function writing into archive, therefore parameters are not checked (mainly unique of `id` is not checked).

Inputs

`u`	Value to store into archive	`Any`
`RUN`	Enable execution	`Bool`

Parameters

type	Type of all trend buffers 1 ..... Bool 2 ..... Byte (U8) 3 ..... Short (I16) 4 ..... Long (I32) 5 ..... Word (U16) 6 ..... DWord (U32) 7 ..... Float (F32) 8 ..... Double (F64) 9 ..... Time 10 .... Large (I64) 11 .... Error 12 .... String 13 .... Reference	⊙12	Byte (U8)
arc	List of archives to write the events to		Word (U16)
id	Archive item ID. The block not check if id is unique in whole configuration.	⊙1	Word (U16)
lvl	Alarm level	⊙1	Word (U16)
Desc	Event description string	⊙Value Description	String
subtype	alarm subtype (for special ussage only)		

Output

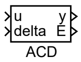
iE	Error code	Error

10.3 Trends recording

ACD – Archive compression using Delta criterion

Block Symbol

Licence: STANDARD



Function Description

The ACD block is meant for storing compressed analog signals to archives using archive events.

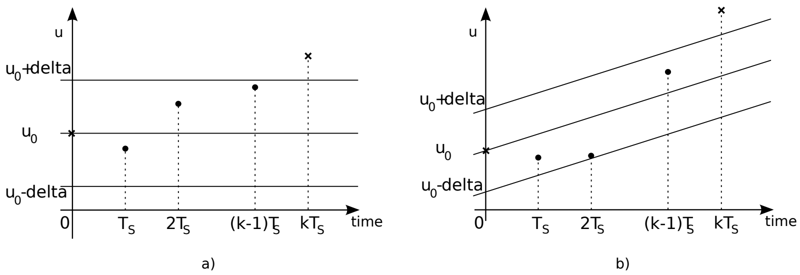
The main idea is to store the input signal **u** only when it changes significantly. The interval between two samples is in the range ⟨**tmin**,**tmax**⟩ seconds (rounded to the nearest multiple of the sampling period). A constant input signal is stored every **tmax** seconds while rapidly changing signal is stored every **tmin** seconds.

When the execution of the block is started, the first input value is stored. This value will be referred to as u0 in the latter. The rules for storing the following samples are given by the **delta** and TR input signals.

For TR = off the condition |u − u0| > **delta** is checked. If it holds and the last stored sample occurred more than **tmin** seconds ago, the value of input **u** is stored and u0=**u** is set. If the condition is fulfilled sooner than **tmin** seconds after the last stored value, the error output E is set to 1 and the first value following the **tmin** interval is stored. At that time the output E is set back to 0 and the whole procedure is repeated.

For TR = on the input signal values are compared to a signal with compensated trend. The condition for storing the signal is the same as in the previous case.

The following figure shows the archiving process for both cases: a) TR = off, b) TR = on. The stored samples are marked by the symbol ×.



Inputs

u	Signal to compress and store		Double (F64)
delta	Threshold for storing the signal	↓0.0 ↑1e+10	Double (F64)

Outputs

`y`	The last value stored in the archive	`Double (F64)`
`E`	Error flag – indicates that a significant change in the input signal occurred sooner than the `tmin` interval passes `off` ... No error `on` .... An error occurred	`Bool`

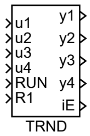
Parameters

`acls`	Archive class determining the variable type to store ⊙8 `1` ..... Bool `2` ..... Byte (U8) `3` ..... Short (I16) `4` ..... Long (I32) `5` ..... Word (U16) `6` ..... DWord (U32) `7` ..... Float (F32) `8` ..... Double (F64) `10` .... Large (I64)	`Byte (U8)`
`arc`	List of archives to store the events. The format of the list is e.g. `1,3..5,8`. The event will be stored in all listed archives (see the `ARC` block for details on archives numbering). Third-party programs (Simulink, OPC clients etc.) work with an integer number, which is a binary mask, i.e. `157` (binary `10011101`) in the mentioned case.	`Word (U16)`
`id`	Identification code of the event in the archive. This identifier must be unique in the whole target device with the REXYGEN control system (i.e. in all archiving blocks). Disabled for `id` = 0. ⊙1	`Word (U16)`
`tmin`	The shortest interval between two samples of the `u` input signal stored in the archive [s] ↓0.001 ↑1000000.0 ⊙1.0	`Double (F64)`
`tmax`	The longest interval between two samples of the `u` input signal stored in the archive [s] ↓1.0 ↑1000000.0 ⊙1000.0	`Double (F64)`
`TR`	Trend evaluation flag ⊙on `off` ... The deviation of the input signal from the last stored value is evaluated `on` .... The deviation of the input signal from the last value's trend is evaluated	`Bool`
`Desc`	Extended description of the event which is displayed by the diagnostic tools of the REXYGEN system ⊙`Value Description`	`String`

TRND – Real-time trend recording

Block Symbol

Licence: STANDARD



Function Description

The TRND block is designed for storing of up to 4 input signals (u1 to u4) in cyclic buffers in the memory of the target device. The main advantage of the TRND block is the synchronization with the real-time executive, which allows trending of even very fast signals (i.e. with very high sampling frequency). In contrary to asynchronous data storing in the higher level operator machine (host), there are no lost or multiply stored samples.

The number of stored signals is determined by the parameter n. In case the trend buffer of length l samples gets full, the oldest samples are overwritten. Data can be stored once in pfac executions of the block (decimation) and the data can be further processed according to the ptype1 to ptype4 parameters. The other decimation factor afac can be used for storing data in archives.

The type of trend buffers can be specified in order to conserve memory of the target device. The memory requirements of the trend buffers are defined by the formula $s \cdot \mathtt{n} \cdot \mathtt{l}$, where s is the size of the corresponding variable in bytes. The default type Double consumes 8 bytes per sample, thus for storing $\mathtt{n} = 4$ trends of this type and length $\mathtt{l} = 1000$, $8 \cdot 4 \cdot 1000 = 32000$ bytes are required. In case the input signals come from 16-bit A/D converter the Word type can be used and memory requirements drop to one quarter. Memory requirements and allowed ranges of individual types are summarized in table 1.1 on page 18 of this reference guide.

It can happen that the processed input value exceeds the representable limits when using different type of buffer than Double. In such a case the highest (lowest) representable number of the corresponding type is stored in the buffer and an error is binary encoded to the iE output according to the following table (the unused bits are omitted):

Error	Range underflow				Range overflow			
Input	u4	u3	u2	u1	u4	u3	u2	u1
Bit number	11	10	9	8	3	2	1	0
Bit weight	2048	1024	512	256	8	4	2	1

In case of simultaneous errors the resulting error code is given by the sum of the weights of individual errors. Note that underflow and overflow cannot happen simultaneously on a single input.

It is possible to read, display and export the stored data by the REXYGEN Studio in the `Watch` mode. After double-clicking on the corresponding `TRND` block, a new card with the prefix Trend will open.

WARNING: set any of the parameters `arc`, `afac`, `id` to 0/empty disable writing data into archive. The data are available in diagnostic tools only in this case.

Inputs

`u1..u4`	Analog inputs to be processed and stored in the trend	`Double (F64)`
`RUN`	Enable execution. The data are processed and stored if and only if `RUN` = `on`.	`Bool`
`R1`	Input for clearing the trend contents. The buffers are cleared when `R1` = `on`. This flag overpowers the `RUN` input.	`Bool`

Outputs

`y1..y4`	Analog outputs of the block set once in `pfac` executions of the block to the last values stored in the trend buffers	`Double (F64)`
`iE`	Error code, see the table above	`Long (I32)`

Parameters

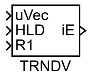
`n`	Number of signals to process and store in the trend buffers ↓1 ↑4 ⊙4	`Long (I32)`
`l`	Number of samples reserved in memory for each trend buffer ↓0 ↑268435000 ⊙1000	`Long (I32)`
`btype`	Type of all `n` trend buffers ⊙8 1 ..... Bool 4 ..... Long 7 ..... Float 2 ..... Byte 5 ..... Word 8 ..... Double 3 ..... Short 6 ..... DWord 10 .... Large	`Long (I32)`
`ptype`i	The way the signal `u`i, $i = 1 \ldots 4$, is processed. The last `pfac` samples are processed as selected and the result is stored in the i-th trend buffer. ⊙1 1 ..... No processing, just storing data 2 ..... Minimum from the last `pfac` samples 3 ..... Maximum from the last `pfac` samples 4 ..... Sum of the last `pfac` samples 5 ..... Simple average of the last `pfac` samples 6 ..... Root mean square of the last `pfac` samples 7 ..... Variance of the last `pfac` samples	`Long (I32)`
`pfac`	Multiple of the block execution period defining the period for storing the data in the trend buffers. Data are stored with the period of `pfac` $\cdot T_S$ unless `RUN` = `off`, where T_S is the block execution period in seconds. ↓1 ↑1000000 ⊙1	`Long (I32)`

`afac`	Every `afac`-th sample stored in the trend buffer is also stored in the archives specified by the `arc` parameter. There are no data stored in the archives for `afac` = 0. Data are stored with the period of `afac`·`pfac`·T_S, where T_S is the block execution period in seconds. ↓0 ↑1000000	`Long (I32)`
`arc`	List of archives to store the trend data. The format of the list is e.g. `1,3..5,8`. The data will be stored in all listed archives (see the `ARC` block for details on archives numbering). Third-party programs (Simulink, OPC clients etc.) work with an integer number, which is a binary mask, i.e. `157` (binary `10011101`) in the mentioned case.	`Word (U16)`
`id`	Identification code of the trend block. This identifier must be unique in the whole target device with the REXYGEN system (i.e. in all archiving blocks). Disabled for `id` = 0. ⊙1	`Word (U16)`
`Title`	Title of the trend to be displayed in the diagnostic tools of the REXYGEN system, e.g. in the `Watch` mode in the REXYGEN Studio program. ⊙`Trend Title`	`String`
`timesrc`	Source of timestamps. Each data sample in trend buffer is stored with a timestamp. For fast or short term trends where you are interested in sample-by-sample timing more than in absolute time, choose `CORETIMER` – REXYGEN internal technological time which is incremented by nominal period each base tick. For long running trends where you are interested mostly in absolute time shared with operating system (and possibly synchronized by NTP), choose `RTC`. Other values are intended for debug or special purposes. ⊙1 1 ..... `CORETIMER` – technological time – at current tick 2 ..... `CORETIMER-PRECISE` – technological time – at block execution 3 ..... `RTC` – real time clock (wallclock) from operating system – at current tick 4 ..... `RTC-PRECISE` – real time clock (wallclock) from operating system – at block execution 4 ..... `PFC` – raw high precision time (PerFormanceCounter)	`Long (I32)`
`SigNames`	Names of the signals to be displayed in the diagnostic tools of the REXYGEN system, e.g. in the `Watch` mode in the REXYGEN Studio program. Each line is name of one signal respectively.	`String`

TRNDV – Real-time trend recording with vector input

Block Symbol

Licence: STANDARD



Function Description

The TRNDV block is designed for storing input signals which arrive at the uVec input in vector form. On the contrary to the TRND block it allows storing more than 4 signals. The signals are stored in cyclic buffers in the memory of the target device. The main advantage of the TRNDV block is the synchronization with the real-time executive, which allows trending of even very fast signals (i.e. with very high sampling frequency). In contrary to asynchronous data storing in the higher level operator machine (host), there are no samples lost or multiply stored.

The number of stored signals is determined by the parameter n. In case the trend buffer of length l samples gets full, the oldest samples are overwritten. Data can be stored once in pfac executions of the block (decimation). The other decimation factor afac can be used for storing data in archives.

The type of trend buffers can be specified in order to conserve memory of the target device. The memory requirements of the trend buffers are defined by the formula $s \cdot \mathtt{n} \cdot \mathtt{l}$, where s is the size of the corresponding variable in bytes. The default type Double consumes 8 bytes per sample, thus for storing e.g. $\mathtt{n} = 4$ trends of this type and length $\mathtt{l} = 1000$, $8 \cdot 4 \cdot 1000 = 32000$ bytes are required. In case the input signals come from 16-bit A/D converter the Word type can be used and memory requirements drop to one quarter. Memory requirements and allowed ranges of individual types are summarized in table 1.1 on page 18 of this reference guide.

It is possible to read, display and export the stored data by the REXYGEN Studio in the Watch mode. After double-clicking on the corresponding TRNDLF block, a new card with the prefix Trend will open.

WARNING: set any of the parameters arc, afac, id to 0/empty disable writing data into archive. The data are available in diagnostic tools only in this case.

Inputs

uVec	Vector signal to record	Reference
HLD	Input for freezing the cyclic buffers, no data is appended when HLD = on	Bool
R1	Input for clearing the trend contents. The buffers are cleared when R1 = on. This flag overpowers the HLD input.	Bool

Output

iE	Error code i ..... REXYGEN general error	Error

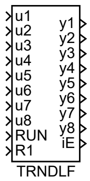
Parameters

n	Number of signals (trend buffers) ↓1 ↑64 ⊙8	Long (I32)
l	Number of samples per trend buffer ↓2 ↑268435000 ⊙1000	Long (I32)
btype	Type of all trend buffers ⊙8 1 ..... Bool 4 ..... Long 7 ..... Float 2 ..... Byte 5 ..... Word 8 ..... Double 3 ..... Short 6 ..... DWord 10 .... Large	Long (I32)
pfac	Multiple of the block execution period defining the period for storing the data in the trend buffers. Data are stored with the period of pfac $\cdot T_S$ unless RUN = off, where T_S is the block execution period in seconds. ↓1 ↑1000000 ⊙1	Long (I32)
afac	Every afac-th sample stored in the trend buffer is also stored in the archives specified by the arc parameter. There are no data stored in the archives for afac = 0. Data are stored with the period of afac$\cdot$pfac$\cdot T_S$, where T_S is the block execution period in seconds. ↓0 ↑1000000	Long (I32)
arc	List of archives to store the trend data. The format of the list is e.g. 1,3..5,8. The data will be stored in all listed archives (see the ARC block for details on archives numbering). Third-party programs (Simulink, OPC clients etc.) work with an integer number, which is a binary mask, i.e. 157 (binary 10011101) in the mentioned case.	Word (U16)
id	Identification code of the trend block. This identifier must be unique in the whole target device with the REXYGEN system (i.e. in all archiving blocks). Disabled for id = 0. ⊙1	Word (U16)
Title	Title of the trend to be displayed in the diagnostic tools of the REXYGEN system, e.g. in the Watch mode in the REXYGEN Studio program. ⊙Trend Title	String
timesrc	Source of timestamps. Each data sample in trend buffer is stored with a timestamp. For fast or short term trends where you are interested in sample-by-sample timing more than in absolute time, choose CORETIMER – REXYGEN internal technological time which is incremented by nominal period each base tick. For long running trends where you are interested mostly in absolute time shared with operating system (and possibly synchronized by NTP), choose RTC. Other values are intended for debug or special purposes. ⊙1	Long (I32)
SigNames	Names of the signals to be displayed in the diagnostic tools of the REXYGEN system, e.g. in the Watch mode in the REXYGEN Studio program. Each line is name of one signal respectively.	String

TRNDLF – * Real-time trend recording (lock-free)

Block Symbol

Licence: ADVANCED



Function Description

The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

Inputs

u1 ..u8	Analog inputs of the block	Double (F64)
RUN	Enable execution	Bool
R1	Input for clearing the trend contents. The buffers are cleared when R1 = on. This flag overpowers the RUN input.	Bool

Parameters

n	Number of signals (trend buffers)	↓1 ↑8 ⊙8	Long (I32)
l	Number of samples per trend buffer	↓0 ↑268435000 ⊙1024	Long (I32)
btype	Type of all trend buffers 1 ..... Bool 2 ..... Byte (U8) 3 ..... Short (I16) 4 ..... Long (I32) 5 ..... Word (U16) 6 ..... DWord (U32) 7 ..... Float (F32) 8 ..... Double (F64) -- .... 10 .... Large (I64)	⊙8	Long (I32)
Title	Trend title string	⊙Trend Title	String
timesrc	Source of timestamps	⊙1	Long (I32)
SigNames	Names of the signals to be displayed in the diagnostic tools of the REXYGEN system, e.g. in the Watch mode in the REXYGEN Studio program. Each line is name of one signal respectively.		String

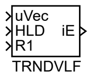
Outputs

y1	First analog output of the block	Double (F64)
y2	Second analog output of the block	Double (F64)
y3	Third analog output of the block	Double (F64)
y4	Fourth analog output of the block	Double (F64)
y5	Fifth analog output of the block	Double (F64)
y6	Sixth analog output of the block	Double (F64)
y7	Seventh analog output of the block	Double (F64)
y8	Eighth analog output of the block	Double (F64)
iE	Error code (bitwise multiplexed)	Long (I32)

TRNDVLF – * Real-time trend recording (for vector signals, lock-free)

Block Symbol

Licence: ADVANCED



Function Description

The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

Inputs

uVec	Vector signal to record	Reference
HLD	Hold	Bool
R1	Input for clearing the trend contents. The buffers are cleared when R1 = on. This flag overpowers the HLD input.	Bool

Parameters

n	Number of signals (trend buffers)	↓1 ↑64 ⊙8	Long (I32)
l	Number of samples per trend buffer	↓2 ↑268435000 ⊙1024	Long (I32)
btype	Type of all trend buffers 1 ..... Bool 2 ..... Byte (U8) 3 ..... Short (I16) 4 ..... Long (I32) 5 ..... Word (U16) 6 ..... DWord (U32) 7 ..... Float (F32) 8 ..... Double (F64) -- .... 10 .... Large (I64)	⊙8	Long (I32)
Title	Trend title string	⊙Trend Title	String
timesrc	Source of timestamps	⊙1	Long (I32)
SigNames	Names of the signals to be displayed in the diagnostic tools of the REXYGEN system, e.g. in the Watch mode in the REXYGEN Studio program. Each line is name of one signal respectively.		String

Output

`iE`	Error code `i` ..... REXYGEN general error	`Error`

10.4 Archive management

`AFLUSH` – Forced archive flushing

Block Symbol

Licence: STANDARD

Function Description

The `AFLUSH` block is intended for immediate storing of archive data to permanent memory (hard drive, flash disk, etc.). It is useful when power loss can be anticipated, e.g. emergency shutdown of the system following some failure. It forces the archive subsystem to write all archive data to avoid data loss. The write operation is initiated by a rising edge (`off`→`on`) at the `FLUSH` input regardless of the `period` parameter of the `ARC` block.

Input

`FLUSH`	Force archive flushing	`Bool`

Parameter

`arc`	List of archives to store the events. The format of the list is e.g. `1,3..5,8`. The event will be stored in all listed archives (see the `ARC` block for details on archives numbering). Third-party programs (Simulink, OPC clients etc.) work with an integer number, which is a binary mask, i.e. `157` (binary `10011101`) in the mentioned case.	`Word (U16)`

Chapter 11

STRING – Blocks for string operations

Contents

CNS – String constant

Block Symbol

Licence: STANDARD

Function Description

The CNS block is a simple string constant with maximal available size. A value of scv is always truncated to nmax.

Parameters

scv	String (constant) value		String
nmax	Allocated size of string [bytes]	↓0 ↑65520	Long (I32)

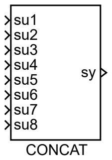
Output

sy	String output value	String

CONCAT – Concat string by pattern

Block Symbol

Licence: STANDARD



Function Description

Concatenates up to 8 input strings **su1** to **su8** by pattern specified in **ptrn** parameter.

Inputs

su1..8	String input value		String

Parameters

ptrn	Concatenation pattern	⊙%1%2%3%4	String
nmax	Allocated size of string [bytes]	↓0 ↑65520	Long (I32)

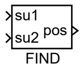
Output

sy	String output value		String

FIND – Find a Substring

Block Symbol

Licence: STANDARD



Function Description

The FIND block searches for the string su2 in the string su1 and returns a one-based index into su1 if a su2 is found or zero otherwise. Both su1 and su2 are truncated to nmax.

Inputs

su1	String input value	String
su2	String input value	String

Parameter

nmax	Allocated size of string [bytes]	↓0 ↑65520	Long (I32)

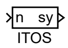
Output

pos	Position of substring	Long (I32)
iE	Error code	Error

ITOS – Integer number to string conversion

Block Symbol

Licence: STANDARD



Function Description

The ITOS block is used for converting an integer into text. The `len` parameter specifies the minimum length of the output string. If the number has a smaller number of digits, zeroes or spaces will be added according to the `mode` parameter. The `radix` parameter specifies the numerical system in which the conversion is to be performed. The output string does not contain any identification of the numerical system used (e.g. the 0x prefix for the hexadecimal system).

Input

`n`	Integer input of the block		`Long (I32)`

Output

`sy`	String output value		`String`

Parameters

`len`	Minimum length of output string	↓0 ↑30	`Long (I32)`
`mode`	Output string format 1 ..... Align right, fill with spaces 2 ..... Align right, fill with zeroes 3 ..... Align left, fill with spaces	⊙1	`Long (I32)`
`radix`	Radix 2 ..... Binary 8 ..... Octal 10 .... Decimal 16 .... Hexadecimal	⊙10	`Long (I32)`

LEN – String length

Block Symbol

Licence: STANDARD

Function Description

The LEN block returns the actual length of the string in su in UTF-8 characters.

Input

su	String input value		String

Parameter

nmax	Allocated size of string [bytes]	↓0 ↑65520	Long (I32)

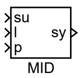
Output

len	Length of input string		Long (I32)

MID – Substring Extraction

Block Symbol

Licence: STANDARD



Function Description

The MID block extracts a substring sy from su. The parameters l and p specify position and length of the string being extracted in UTF-8 characters. The parameter p is one-based.

Inputs

su	String input value	String
l	Length of output string	Long (I32)
p	Position of output string (one-based)	Long (I32)

Parameter

nmax	Allocated size of string [bytes]	↓0 ↑65520	Long (I32)

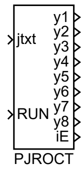
Output

sy	String output value	String
iE	Error code	Error

PJROCT – Parse JSON string (real output)

Block Symbol

Licence: STANDARD



Function Description

Parses input JSON string `jtxt` according to specified `name*` parameters when the input `RUN` is `on`. Output signals are `real` type. Value of the `yerr` parameter is put on the `y*` output when error occured (e.g. specified object is not exist or value is not a number).

This block expects text in JSON format on the `jtxt` input. The outputs of `y1` to `y7` then have the values (string) of the objects identified by the parameters `name1` to `name7`. If one of the parameters `name1` to `name7` is empty, the corresponding output will be empty and this is not considered as an error. The input string evaluates only if `RUN = on`. An error is indicated on the output `iE`. The following cases may occur:

- 0 - no error
- -1 - one of the parameters `name1` to `name7` refers to an object that does not appear in the input text (at the input `jtxt`)
- -103 - the text on the input `jtxt` does not correspond to the JSON format
- -106 - all of the parameters `name1` to `name7` refer to an object that does not appear in the input text (on the input `jtxt`)

Example: Let
`jtxt = "{"id": 12345, "params": {"temperature": 23, "pressure": 2.34 }, "description": "reactor1", "values" :[12, 34.5 , 45.0, 30.2]}"`
`name1 = "params.temperature"`,
`name2 = "values[0]"`,
`name3 = "pressure"`,
`name4 = "description"`,
then the output `y1` will be the `"23"` string, the output `y2` will be the `"12"` string, output `y3` will remain empty and an error will be signaled, the output `y4` will remain empty and an error will be signaled.

Inputs

`jtxt`	JSON formated string	`String`

RUN	Enable execution		Bool

Parameters

name1..8	Property name of JSON element		String
nmax	Allocated size of string [bytes]	↓0 ↑65520	Long (I32)
yerr	Substitute value for an error case		Double (F64)

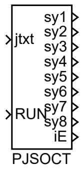
Outputs

y1..8	Block output signal		Double (F64)
iE	Error code		Error

PJSOCT – Parse JSON string (string output)

Block Symbol

Licence: STANDARD



Function Description

Parses input JSON string `jtxt` according to specified `name*` parameters when the input `RUN` is `on`. Output signals are `string` type.

This block expects text in JSON format on the `jtxt` input. The outputs of `sy1` to `sy7` then have the values of the objects identified by the parameters `name1` to `name7`. If one of the parameters `name1` to `name7` is empty, the corresponding output will be empty and this is not considered as an error. The input string evaluates only if `RUN = on`. An error is indicated on the output `iE`. The following cases may occur:

- 0 - no error
- -1 - one of the parameters `name1` to `name7` refers to an object that does not appear in the input text (at the input `jtxt`)
- -103 - the text on the input `jtxt` does not correspond to the JSON format
- -106 - all of the parameters `name1` to `name7` refer to an object that does not appear in the input text (on the input `jtxt`)

Example: Let
`jtxt = "{"id": 12345, "params": {"temperature": 23, "pressure": 2.34 }, "description": "reactor1", "values" :[12, 34.5 , 45.0, 30.2]}"`
`name1 = "params.temperature"`,
`name2 = "values[0]"`,
`name3 = "pressure"`,
`name4 = "description"`,
then the output `sy1` will be the `"23"` string, the output `sy2` will be the `"12"` string, output `sy3` will remain empty and an error will be signaled, the output `sy4` will be the `"reactor1"` string.

Inputs

`jtxt`	JSON formated string	`String`
`RUN`	Enable execution	`Bool`

Parameters

`name1..8`	Name of JSON object		`String`
`nmax`	Allocated size of string [bytes]	↓0 ↑65520	`Long (I32)`

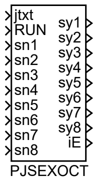
Outputs

`sy1..8`	String output value	`String`
`iE`	Error code	`Error`

PJSEXOCT – Parse JSON string (string output)

Block Symbol

Licence: STANDARD



Function Description

The block is almost the same as PJSOCT block, except `name*` parameters can contain control sequence `% + number` that is substituted by `sn + number` input.

Example: Let
`sn1 = "2"`,
`sn2 = "rpm"`,
`name1 = "motor[%1].temp"`,
`name2 = "motor[%1].%2"`,
then `name1` is expand to `motor[2].temp`, `name2` is expand to `motor[2].rpm`.

Inputs

jtxt	JSON formated string	String
RUN	Enable execution	Bool
sn1..8	Part of name of JSON object	String

Parameters

name1..8	Name of JSON object		String
nmax	Allocated size of string [bytes]	↓0 ↑65520	Long (I32)

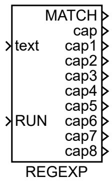
Outputs

sy1..8	String output value	Error
iE	Error code	Error

REGEXP – Regular expresion parser

Block Symbol

Licence: ADVANCED



Function Description

This block implements a subset of Perl or `C#` or Unix command `grep` regular expression syntax.

Supported syntax is :

- `(?i)` ... Must be at the beginning of the regex. Makes match case-insensitive
- `^` ... Match beginning of a buffer
- `$` ... Match end of a buffer
- `()` ... Grouping and substring capturing
- `\s` ... Match whitespace
- `\S` ... Match non-whitespace
- `\d` ... Match decimal digit
- `\n` ... Match new line character
- `\r` ... Match line feed character
- `\f` ... Match form feed character
- `\v` ... Match vertical tab character
- `\t` ... Match horizontal tab character
- `\b` ... Match backspace character
- `+` ... Match one or more times (greedy)
- `+?` ... Match one or more times (non-greedy)
- `*` ... Match zero or more times (greedy)

- `*?` ... Match zero or more times (non-greedy)
- `?` ... Match zero or once (non-greedy)
- `x|y` ... Match x or y (alternation operator)
- `\meta` ... Match one of the meta characters: ^$().[ ]*+?|\
- `\xHH` ... Match byte with hex value 0xHH, e.g. \x4a
- `[...]` ... Match any character from set. Ranges like [a-z] are supported.
- `[^...]` ... Match any character except the ones in set. Ranges like [a-z] are supported.

Examples

- `[0-9]+` ... Find first integer in input string (and put it into `cap` output)
- `[-+]?[0-9]*\.[0-9]+([eE][-+]?[0-9]+)?` ... Find first real number in input string (and put it into `cap` output)
- `^\s*(.*?)\s*$` ... Put trimmed input string into `cap1` output
- `num\s*:\s*([0-9]*\.[0-9]*)` ... Expect input string in JSON format; find integer parameter `num`, and put its value into `cap1`

Inputs

`text`	String to parse		`String`
`RUN`	Enable execution		`Bool`

Parameters

`expr`	Regular expresion pattern		`String`
`nmax`	Allocated size of string	↓0 ↑65534	`Long (I32)`
`bufmax`	Parser internal buffer size (0 = autodetect)	↓0 ↑10000000	`Long (I32)`

Outputs

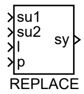
`MATCH`	Pattern match flag	`Bool`
`cap`	Whole matching string	`String`
`cap1`	Captured string (string matched to 1st bracket)	`String`
`cap2`	Captured string (string matched to 2nd bracket)	`String`
`cap3`	Captured string (string matched to 3rd bracket)	`String`
`cap4`	Captured string (string matched to 4th bracket)	`String`

`cap5`	Captured string (string matched to 5th bracket)	`String`
`cap6`	Captured string (string matched to 6th bracket)	`String`
`cap7`	Captured string (string matched to 7th bracket)	`String`
`cap8`	Captured string (string matched to 8th bracket)	`String`

REPLACE – Replace substring

Block Symbol

Licence: STANDARD



Function Description

The REPLACE block replaces a substring from su1 by the string su2 and puts the result in sy. The parameters l and p specify position and length of the string being replaced in UTF-8 characters. The parameter p is one-based.

Inputs

su1	String input value	String
su2	String input value	String
l	Length of origin text	Long (I32)
p	Position of origin text (one-based)	Long (I32)

Parameter

nmax	Allocated size of string [bytes]	↓0 ↑65520	Long (I32)

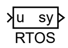
Output

sy	String output value	String
iE	Error code	Error

RTOS – Real Number to String Conversion

Block Symbol

Licence: STANDARD



Function Description

The RTOS converts a real number in u into a string value in su. Precision and format are specified by the prec and mode parameters.

Input

u	Analog input of the block	Double (F64)

Output

sy	String output value	String

Parameters

prec	Precision (number of digits)	↓0 ↑20	Long (I32)
mode	Output string format	⊙1	Long (I32)

1 ..... Best fit – fixed point, but for extremly small or big numbers exponential format; parameter prec is total maximum number of characters in output (mantisa for exponential format)

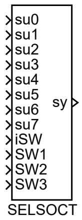
2 ..... Normal – fixed point format; parameter prec is number of places after the decimal point

3 ..... Exponential – scientific format; parameter prec is number of places after the decimal pointe

SELSOCT – Selector switch for string signals

Block Symbol

Licence: STANDARD



Function Description

The SELSOCT block selects one of the input strings and copy it to the output string sy. The selection of the active signal u0. . . u15 is based on the iSW input or the binary inputs SW1. . . SW3. These two modes are distinguished by the BINF binary flag. The signal is selected according to the following table:

iSW	SW1	SW2	SW3	y
0	off	off	off	u0
1	on	off	off	u1
2	off	on	off	u2
3	on	on	off	u3
4	off	off	on	u4
5	on	off	on	u5
6	off	on	on	u6
7	on	on	on	u7

Inputs

su0..7	String input value		String
iSW	Active signal selector		Long (I32)
SW1..3	Binary signal selector		Bool

Parameters

BINF	Enable the binary selectors		Bool
nmax	Allocated size of string [bytes]	↓0 ↑65520	Long (I32)

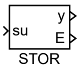
Output

sy	The selected input signal		String

STOR – String to real number conversion

Block Symbol

Licence: STANDARD



Function Description

The STOR converts a string in su into a real number in y. An error is signaled in E if unsuccessful.

Input

su	String input value	String

Parameter

yerr	Substitute value for an error case	Double (F64)

Outputs

y	Analog output of the block	Double (F64)
E	Error indicator	Bool

Chapter 12

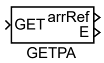
PARAM – Blocks for parameter handling

Contents

GETPA – Block for remote array parameter acquirement

Block Symbol

Licence: STANDARD



Function Description

The GETPA block is used for acquiring the array parameters of other blocks in the model remotely . The block operates in two modes, which are switched by the GETF parameter. For GETF = off the output arrRef is set to the value of the remote parameter at the start and every time when the remote parameter changes. If the GETF parameter is set to on, then the block works in single-shot read mode. In that case the remote parameter is read only when rising edge (off→on) occurs at the GET input.

The name of the remote parameter is determined by the string parameter sc (string connection), which has the form <block_path:parameter_name>. The path to the block whose parameter should be read can contain hierarchic levels separated by dots followed by the block name. The path can be either relative or absolute:

- Relative – starts at the level where the GETPA block is located. The string has to be prefixed with '.' in this case. Examples of relative paths: ".CNDR:yp", ".Lights.ATMT:touts".
- Relative to task – starts at the root level of the task where the SETPA block is located. The string has to be prefixed with '%' in this case. Examples of paths: "%CNDR:yp", "%Lights.ATMT:touts".
- Absolute – complete sequence of hierarchic levels down to the block. For referring to blocks located in the driver task (see the IOTASK block for details on configuration) the '&' followed by the driver's name is used at the beginning of the absolute path. Examples of absolute paths: "task1.inputs.ATMT:touts", "&EfaDrv.measurements.CNDR:yp".

The order and names of individual hierarchic levels are presented in a tree-like structure within the Diagnostics section of the REXYGEN Studio program.

Warning 1: If the remote parameter is in a task other than the GETPA block, block execution is delayed until the remote task is completed. It is necessary to avoid the so-called race conditions and guarantee the correct value reading. Therefore, it is recommended to include the GETPA block in a slower task (longer period/execution time) and read parameter in a faster task (shorter period/execution time). In the opposite situation (e.g. the GETPA block in a faster task), the SETPA block should be used in a slower task.

Note 1: If parameter GETF = off and source array is in same task as the GETPA block, data are not copy into intermediate array, but output is direct reference to original array.

It save resources (cpu time and memory). The `nmax, etype` parameters are ignored in this case.

Note 2: When using multiple `GETPA` blocks, it is not guaranteed to read all data from a remote task in the same tick. It is only guaranteed that the previous block will receive a value in the same or previous period as the next block (the order of blocks execution can be checked in REXYGEN Diagnostics).

Note 3: The remote parameter must be a primary array (for example `CNA:acn`, `RTOV:xVec`, `MX_MAT:ay`). The array reference (like `CNA:vec`, `RTOV:yVec`, `SUBSYSTEM:Outport`) is not supported.

Input

`GET`	Input for initiating one-shot parameter read. Array is read on rising edge of this input.	`Bool`

Outputs

`arrRef`	Array reference	`Reference`
`E`	Error flag	`Bool`

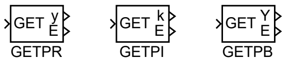
Parameters

`sc`	String connection to the parameter	`String`
`GETF`	Get parameter only when forced to. `off` ... Remote parameter is continuously read `on` .... One-shot mode, the remote parameter is read only when forced to by the `GET` input (rising edge)	`Bool`
`nmax`	Maximum size of array ↓10 ⊙256	`Long (I32)`
`etype`	Type of members of the acquired array. This is type of the intermediate (state) array where is copy of acquired data. The conversion is performed if original and intermediate array has different type. ⊙8 `2` ..... Byte `5` ..... Word `8` ..... Double `3` ..... Short `6` ..... DWord `10` .... Large `4` ..... Long `7` ..... Float	`Long (I32)`

GETPR, GETPI, GETPB – Blocks for remote parameter acquirement

Block Symbols

Licence: STANDARD



Function Description

The GETPR, GETPI and GETPB blocks are used for acquiring the parameters of other blocks in the model remotely . The only difference among the three blocks is the type of parameter which they are acquiring. The GETPR block is used for obtaining real parameters, the GETPI block for integer parameters and the GETPB block for Boolean parameters.

The blocks operate in two modes, which are switched by the GETF parameter. For GETF = off the output y (or k, Y) is set to the value of the remote parameter at the start and every time when the remote parameter changes. If the GETF parameter is set to on, then the blocks work in single-shot read mode. In that case the remote parameter is read only when rising edge (off→on) occurs at the GET input.

The name of the remote parameter is determined by the string parameter sc (string connection), which has the form <block_path:parameter_name>. It is also possible to access individual items of array-type parameters (e.g. the tout parameter of the ATMT block). This can be achieved using the square brackets and item number, e.g. .ATMT:touts[2]. The items are numbered from zero, thus the string connection stated above refers to the third element of the array.

The path to the block whose parameter should be read can contain hierarchic levels separated by dots followed by the block name. The path can be either relative or absolute:

- Relative – starts at the level where the GETPR block (or GETPI, GETPB) is located. The string has to be prefixed with '.' in this case. Examples of relative paths: ".GAIN:k", ".Motor1.Position:ycn".

- Relative to task – starts at the root level of the task where the GETPR block (or GETPI, GETPB, GETPS) is located. The string has to be prefixed with '%' in this case. Examples of paths: "%GAIN:k", "%Motor1.Position:ycn".

- Absolute – complete sequence of hierarchic levels down to the block. For referring to blocks located in the driver task (see the IOTASK block for details on configuration) the '&' followed by the driver's name is used at the beginning of the absolute path. Examples of absolute paths: "task1.inputs.lin1:u2", "&EfaDrv.measurements.DER1:n".

The order and names of individual hierarchic levels are presented in a tree-like structure within the Diagnostics section of the REXYGEN Studio program.

Warning: If the remote parameter is in a task other than the GETPx block, block execution is delayed until the remote task is completed. It is necessary to avoid the so-called race conditions and guarantee the correct value reading. Therefore, it is recommended to include the GETPx block in a slower task (longer period/execution time) and read parameter in a faster task (shorter period/execution time). In the opposite situation (e.g. the GETPx block in a faster task), the SETPx block should be used in a slower task.

Note: When using multiple GETPx blocks, it is not guaranteed to read all data from a remote task in the same tick. It is only guaranteed that the previous block will receive a value in the same or previous period as the next block (the order of blocks execution can be checked in REXYGEN Diagnostics). To obtain multiple values in the same period, it is needed to use the Inport and Outport blocks or the GETPA block.

Input

GET	Input for initiating one-shot parameter read (off→on)	Bool

Outputs

y	Parameter value, output of the GETPR block	Double (F64)
k	Parameter value, output of the GETPI block	Long (I32)
Y	Parameter value, output of the GETPB block	Bool
E	Error flag off ... No error on .... An error occurred	Bool

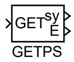
Parameters

sc	String connection to the remote parameter respecting the above mentioned notation	String
GETF	Continuous or one-shot mode off ... Remote parameter is continuously read on .... One-shot mode, the remote parameter is read only when forced to by the GET input (rising edge)	Bool

GETPS – * Block for remote string parameter acquirement

Block Symbol

Licence: STANDARD



Function Description

The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

Input

GET	Input for initiating one-shot parameter read	Bool

Parameters

sc	String connection to the parameter	String
GETF	Get parameter only when forced to off . . . Remote parameter is continuously read on . . . . One-shot mode	Bool
nmax	Allocated size of string	Long (I32)

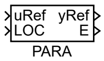
Outputs

sy	Parameter value	String
E	Error indicator off . . . No error on . . . . An error occurred	Bool

PARA – Block with input-defined array parameter

Block Symbol

Licence: STANDARD



Function Description

The PARA block allows, additionally to the standard way of parameter setting, changing one of its parameters by the input signal. The input-parameter pair is uRef and apar.

The Boolean input LOC (LOCal) determines whether the value of the apar parameter is read from the input uRef or is input-independent (LOC = on). In the local mode LOC = on the parameter apar contains the last value of input uRef entering the block right before LOC was set to on.

The output value is equivalent to the value of the parameter (yRef = apar).

Inputs

uRef	Array reference	Reference
LOC	Activation of local mode off ... The parameter follows the input on .... Local mode active	Bool

Output

yRef	Array reference	Reference

Parameters

SETS	Set array size flag. Use this flag to adjust the size of array when setting the parameter.	Bool
nmax	Allocated size of the apar array ↓10 ⊙100	Long (I32)
etype	Type of members of the apar array ⊙8 2 ..... Byte 5 ..... Word 8 ..... Double 3 ..... Short 6 ..... DWord 10 .... Large 4 ..... Long 7 ..... Float	Long (I32)
apar	Internal value of the parameter ⊙[0.0 1.0 2.0 3.0 4.0 5.0]	Double (F64)

PARE – Block with input-defined enumeration parameter

Block Symbol

Licence: STANDARD

Function Description

The block is similar to the the PARI block with the additional option to assign texts to numeric values. The corresponding text is set on the output sy. The block has two modes and the active mode is selected by the LIST parameter. If LIST=off a corresponding text for the input value is set on the output sy. If LIST=on the input number is considered as a bitfield, texts are defined for each bit and the output sy is composed of the texts that correspond to bits which are set. The behavior for undefined values is determined by the SATF parameter. If SATF=off, undefined values are set to output iy and the output sy is set to empty text. Undefined values are ignored if SAT=on. The pupstr parameter has the same format as in the CNE block: <number1>: <description1>|<number2>: <description2>|<number3>: <description3> ...

Inputs

ip	Parameter value	Long (I32)
LOC	Activation of local mode off ... The parameter follows the input on .... Local mode active	Bool

Parameters

ipar	Internal value of parameter	⊙1	Long (I32)
pupstr	Popup list definition ⊙1: option A|2: option B|3: option C		String
NUM	Number in string outut		Bool
LIST	Bitfield mode		Bool
SATF	Saturation flag (if undefined values are passed to output)		Bool

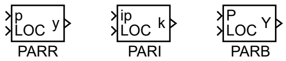
Outputs

iy	Integer output of the block	Long (I32)
sy	String output value	String

PARR, PARI, PARB – Blocks with input-defined parameter

Block Symbols

Licence: STANDARD



Function Description

The PARR, PARI and PARB blocks allow, additionally to the standard way of parameters setting, changing one of their parameters by the input signal. The input-parameter pairs are p and par for the PARR block, ip and ipar for the PARI block and finally P and PAR for the PARB block.

The Boolean input LOC (LOCal) determines whether the value of the par (or ipar, PAR) parameter is read from the input p (or ip, P) or is input-independent (LOC = on). In the local mode LOC = on the parameter par (or ipar, PAR) contains the last value of input p (or ip, P) entering the block right before LOC was set to on. Afterwards it is possible to modify the value manually.

The output value is equivalent to the value of the parameter y = par, (or k = ipar, Y = PAR). The output of the PARR and PARI blocks can be additionally constrained by the saturation limits ⟨lolim, hilim⟩. The saturation is active only when SATF = on.

See also the SHLD block, which can be used for storing a numeric value, similarly as in the PARR block.

Inputs

p	Parameter value (the PARR block)	Double (F64)
ip	Parameter value (the PARI block)	Long (I32)
P	Parameter value (the PARB block)	Bool
LOC	Activation of local mode off . . . The parameter follows the input on . . . . Local mode active	Bool

Output

y	Logical output of the PARR block	Double (F64)
k	Logical output of the PARI block	Long (I32)
Y	Logical output of the PARB block	Bool

Parameter

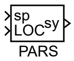
par	Initial value of the parameter (the PARR block)	⊙1.0	Double (F64)
ipar	Initial value of the parameter (the PARI block)	⊙1	Long (I32)
PAR	Initial value of the parameter (the PARB block)	⊙on	Bool

`SATF`	Activation of the saturation limits for the `PARR` and `PARI` blocks `off` … Signal not limited `on` …. Saturation limits active	`Bool`
`hilim`	Upper limit of the output signal (the `PARR` and `PARI` blocks) ⊙`1.0`	`Double (F64)`
`lolim`	Lower limit of the output signal (the `PARR` and `PARI` blocks) ⊙`-1.0`	`Double (F64)`

PARS – * Block with input-defined string parameter

Block Symbol

Licence: STANDARD



Function Description

The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

Inputs

sp	Parameter value	String
LOC	Activation of local mode	Bool

Parameters

spar	Internal value of the parameter	String
nmax	Allocated size of string	Long (I32)

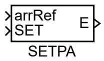
Output

sy	String output of the block	String

SETPA – Block for remote array parameter setting

Block Symbol

Licence: STANDARD



Function Description

The SETPA block is used for setting the array parameters of other blocks in the model remotely . The block operates in two modes, which are switched by the SETF parameter. For SETF = off the remote parameter cs is set to the value of the input vector signal arrRef at the start and every time when the input signal changes. If the SETF parameter is set to on, then the block works in one-shot write mode. In that case the remote parameter is set only when rising edge (off→on) occurs at the SET input.

The name of the remote parameter is determined by the string parameter sc (string connection), which has the form <block_path:parameter_name>. The path to the block whose parameter should be read can contain hierarchic levels separated by dots followed by the block name. The path can be either relative or absolute:

- Relative – starts at the level where the GETPA block is located. The string has to be prefixed with '.' in this case. Examples of relative paths: ".CNDR:yp", ".Lights.ATMT:touts".
- Relative to task – starts at the root level of the task where the SETPA block is located. The string has to be prefixed with '%' in this case. Examples of paths: "%GAIN:k", "%Motor1.Position:ycn".
- Absolute – complete sequence of hierarchic levels down to the block. For referring to blocks located in the driver task (see the IOTASK block for details on configuration) the '&' followed by the driver's name is used at the beginning of the absolute path. Examples of absolute paths: "task1.inputs.ATMT:touts", "&EfaDrv.measurements.CNDR:yp".

The order and names of individual hierarchic levels are presented in a tree-like structure within the Diagnostics section of the REXYGEN Studio program.

Warning: If the remote parameter is in a task other than the SETPA block, block execution is delayed until the remote task is completed. It is necessary to avoid the so-called race conditions and guarantee the correct value setting. Therefore, it is recommended to include the SETPA block in a slower task (longer period/execution time) and set parameter in a faster task (shorter period/execution time). In the opposite situation (e.g. the SETPA block in a faster task), the GETPA block should be used in a slower task.

Note 1: When using multiple SETPA blocks, it is not guaranteed that all data will be written to the remote task in the same tick. It is only guaranteed that the previous block will set a value in the same or previous period as the next block (the order of blocks execution can be checked in REXYGEN Diagnostics).

Note 2: The remote parameter must be a primary array (for example `CNA:acn`, `RTOV:xVec`, `MX_MAT:ay`). The array reference (like `CNA:vec`, `RTOV:yVec`, `SUBSYSTEM:Outport`) is not supported.

Inputs

`arrRef`	Array reference	`Reference`
`SET`	Input for initiating one-shot parameter write	`Bool`

Output

`E`	Error flag	`Bool`

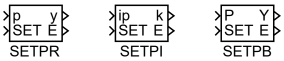
Parameters

`sc`	String connection to the parameter	`String`
`SETF`	Continuous or one-shot mode `off` ... Remote parameter is continuously updated `on` .... One-shot mode, the remote parameter is updated only when forced to by the `SET` input (rising edge)	`Bool`
`SETS`	Set array size flag. Use this flag to adjust the size of array when setting the parameter.	`Bool`

SETPR, SETPI, SETPB – Blocks for remote parameter setting

Block Symbols

Licence: STANDARD



Function Description

The SETPR, SETPI, SETPB and SETPS blocks are used for setting the parameters of other blocks in the model remotely. The only difference among the three blocks is the type of parameter which they are setting. The SETPR block is used for setting real parameters, the SETPI block for integer parameters, the SETPB block for Boolean parameters and the SETPS block for string parameters.

The blocks operate in two modes, which are switched by the SETF parameter. For SETF = off the remote parameter sc is set to the value of the input signal p (or ip, P) at the start and every time when the input changes. If the SETF parameter is set to on, then the blocks work in one-shot write mode. In that case the remote parameter is set only when rising edge (off→on) occurs at the SET input. Successful modification of the remote parameter is indicated by zero error output E = off and the output y (or k, Y) is set to the value of the modified parameter. The error output is set to E = on in case of write error.

The name of the remote parameter is determined by the string parameter sc (string connection), which has the form <block_path:parameter_name>. It is also possible to access individual items of array-type parameters (e.g. the tout parameter of the ATMT block). This can be achieved using the square brackets and item number, e.g. .ATMT:touts[2]. The items are numbered from zero, thus the string connection stated above refers to the third element of the array.

The path to the block whose parameter should be set can contain hierarchic levels separated by dots followed by the block name. The path can be either relative or absolute:

- Relative – starts at the level where the SETPR block (or SETPI, SETPB, SETPS) is located. The string has to be prefixed with '.' in this case. Examples of relative paths: ".GAIN:k", ".Motor1.Position:ycn".
- Relative to task – starts at the root level of the task where the SETPR block (or SETPI, SETPB, SETPS) is located. The string has to be prefixed with '%' in this case. Examples of paths: "%GAIN:k", "%Motor1.Position:ycn".
- Absolute – complete sequence of hierarchic levels down to the block. For referring to blocks located in the driver task (see the IOTASK block for details on configuration) the '&' followed by the driver's name is used at the beginning of the absolute path. Examples of absolute paths: "task1.inputs.lin1:u2", "&EfaDrv.measurements.DER1:n".

The order and names of individual hierarchic levels are displayed in a tree structure in the REXYGEN Diagnostics program.

Warning: If the remote parameter is in a task other than the SETPx block, block execution is delayed until the remote task is completed. It is necessary to avoid the so-called race conditions and guarantee the correct value setting. Therefore, it is recommended to include the SETPx block in a slower task (longer period/execution time) and set parameter in a faster task (shorter period/execution time). In the opposite situation (e.g. the SETPx block in a faster task), the GETPx block should be used in a slower task.

Note: When using multiple SETPx blocks, it is not guaranteed that all data will be written to the remote task in the same tick. It is only guaranteed that the previous block will set a value in the same or previous period as the next block (the order of blocks execution can be checked in REXYGEN Diagnostics). To send multiple values in the same period, it is needed to use the Inport and Outport blocks or the SETPA block.

Inputs

p	Desired parameter value at the SETPR block input	Double (F64)
ip	Desired parameter value at the SETPI block input	Long (I32)
P	Desired parameter value at the SETPB block input	Bool
SET	Input for initiating one-shot parameter write (off→on)	Bool

Outputs

y	Parameter value (the SETPR block)	Double (F64)
k	Parameter value (the SETPI block)	Long (I32)
Y	Parameter value (the SETPB block)	Bool
E	Error flag off ... No error on .... An error occurred	Bool

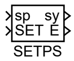
Parameters

sc	String connection to the remote parameter respecting the above mentioned notation	String
SETF	Continuous or one-shot mode off ... Remote parameter is continuously updated on .... One-shot mode, the remote parameter is updated only when forced to by the SET input (rising edge)	Bool

SETPS – * Block for remote string parameter setting

Block Symbol

Licence: STANDARD



Function Description

The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

Inputs

sp	Desired parameter value	String
SET	Input for initiating one-shot parameter write	Bool

Parameters

sc	String connection to the parameter	String
SETF	Set parameter only when forced to	Bool
nmax	Allocated size of string	Long (I32)

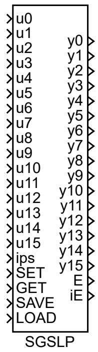
Outputs

sy	Parameter value	String
E	Error indicator	Bool

SGSLP – Set, get, save and load parameters

Block Symbol

Licence: ADVANCED



Function Description

The SGSLP block is a special function block for manipulation with parameters of other function blocks in the REXYGEN system configuration. It works also in the Matlab-Simulink system but its scope is limited to the .mdl file it is included in.

The block can manage up to 16 parameter sets, which are numbered from 0 to 15. The number of parameter sets is given by the nps parameter and the active set is defined by the ips input. If the ips input remains unconnected, the active parameter set is ips $= 0$. Each set contains up to 16 different parameters defined by the string parameters sc0 to sc15. Thus the SGSLP block can work with a maximum of 256 parameters of the REXYGEN system. An empty sc$i$ string means that no parameter is specified, otherwise one of the following syntaxes is used:

1. <block>:<param> – Specifies one function block named block and its parameter param. The same block and parameter are used for all nps parameter sets in this case.

2. <block>:<param><sep>...<block>:<param> – This syntax allows the parameters to differ among the parameter sets. In general, each sc$i$ string can contain up to 16 items in the form <blok>:<param> separated by comma or semi-colon. E.g. the third item of these is active for ips $= 2$. There should be exactly nps items in each non-empty sc$i$ string. If there is less items than nps none of the below described operations can be executed on the incomplete parameter set.

It is recommended not to use both syntaxes in one SGSLP block, all 16 sc$i$ strings should have the same form. The first syntax is for example used when producing nps types of goods, where many parameters must be changed for each type of production. The second syntax is usually used for saving user-defined parameters to disk (see the

SAVE operation below). In that case it is desirable to arrange automated switching of the ips input (e.g. using the ATMT block from the LOGIC library).

The broot parameter is suitable when all blocks whose parameters are to be controlled by the SGSLP block reside in the same subsystem or deeper in the hierarchy. It is inserted in front of each <block> substring in the sc$i$ parameters. The '.' character stands for the subsystem where the SGSLP block is located. No quotation marks are used to define the parameter, they are used here solely to highlight a single character. If the broot parameter is an empty string, all <block> items must contain full path. For example, to create a connection to the CNR block and its parameter ycn located in the same subsystem as the SGSLP block, broot = . and sc0 = CNR:ycn must be set. Or it is possible to leave the broot parameter empty and put the '.' character to the sc0 string. See the GETPR or SETPR blocks description for more details about full paths in the REXYGEN system.

The SGSLP block executes one of the below described operations when a rising edge (off→on) occurs at the input of the same name. The operations are:

SET – Sets the parameters of the corresponding parameter set ips to the values of the input signals ui. In case the parameter is successfully set, the same value is also sent to the yi output.

GET – Gets the parameters of the corresponding parameter set ips. In case the parameter is successfully read, its value is sent to the yi output.

SAVE – Saves the parameters of the corresponding parameter set ips to a file on the target platform. The parameters of the procedure and the format of the resulting file are described below.

LOAD – Loads the parameters of the corresponding parameter set ips from a file on the target platform. This operation is executed also during the initialization of the block but only when $0 \leq$ ips0 $\leq$ nps $- 1$. The parameters of the procedure and the format of the file are described below.

The LOAD and SAVE operations work with a file on the target platform. The name of the file is given by the fname parameter and the following rules:

- If no extension is specified in the fname parameter, the .rxs (ReX Status file) extension is added.
- A backup file is created when overwriting the file. The file name is preserved, only the extension is modified by adding the ' ' character right after the '.' (e.g. when no extension is specified, the backup file has a . rxs extension.
- The path is relative to the folder where the archives of the REXYGEN system are stored. The file should be located on a media which is not erased by system restart (flash drive or hard drive, not RAM).

The `SAVE` operation stores the data in a text file. Two lines are added for each parameter `sc`i, $i = 0, \ldots, m$, where $m < 16$ defines the nonempty `sc`m string with the highest number. The lines have the form:

"<block>:<param>", ..., "<block>:<param>"

<value>, ..., <value>

There are `nps` individual items `"<block>:<param>"` which are separated by commas. The second line contains the same number of `<value>` items which contain the value of the parameter at the same position in the line above. Note that the format of the file remains the same even for `sc`i containing only one `<block>:<param>` item (see the syntax no. 1 above). The `"<block>:<param>"` item is always listed `nps`-times in the file, which allows seamless switching of the `sc`i parameters syntax without modifying the file.

Consider using the `SILO` block if working with only a few values.

Inputs

`u`i	i-th analog input signal, $i = 0, \ldots, 15$	`Double (F64)`
`ips`	Parameter set index (numbered from zero)	`Long (I32)`
`SET`	Set the parameters of the `ips` parameter set according to the values of the `u`i inputs. The values can be found at the `y`i outputs after a successful operation.	`Bool`
`GET`	Get the parameters of the `ips` parameter set. The values can be found at the `y`i outputs after a successful operation.	`Bool`
`SAVE`	Save the `ips` parameter set to a file on the target device	`Bool`
`LOAD`	Load the `ips` parameter set from a file on the target device	`Bool`

Outputs

`y`i	i-th analog output signal, $i = 0, \ldots, 15$	`Double (F64)`
`E`	Error flag `off` ... No error `on` .... An error occurred (see `iE`)	`Bool`

iE	Error or warning code of the last operation 0 ..... Operation successful 1 ..... Fatal error of the Matlab system (only in Simulink), the block is no longer executed 2 ..... Error opening the file for reading (LOAD operation) 3 ..... Error opening the file for writing (SAVE operation) 4 ..... Incorrect file format 5 ..... The ips parameter set not found in the file 6 ..... Parameter not found in the configuration, name mismatch (LOAD operation) 7 ..... Unexpected end of file 8 ..... Error writing to file (disk full?) 9 ..... Parameter syntax error (the ':' character not found) 10 .... Only whitespace in the parameter name 11 .... Error creating the backup file 12 .... Error obtaining the parameter value by the GET operation (non-existing parameter?) 13 .... Error setting the parameter value by the SET operation (non-existing parameter?) 14 .... Timeout during obtaining/setting the parameter 15 .... The specified parameter is read-only 16 .... The ips parameter is out of range	Long (I32)

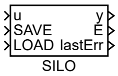
Parameters

nps	Number of parameter sets ↓1 ↑16 ⊙1	Long (I32)
ips0	Index of parameter set to load and set during the block initialization. No set is read for ips0 <0 or ips0 $\geq$ nps ↓-1 ↑15	Long (I32)
iprec	Precision (number of digits) for storing the values of double type in a file ↓2 ↑15 ⊙12	Long (I32)
icolw	Requested column width in the status file. Spaces are appended to the parameter value when necessary. ↓0 ↑22	Long (I32)
fname	Name of the file the SAVE and LOAD operations work with ⊙status	String
broot	Root block in hierarchy, inserted at the beginning of all sc$i$ parameters, see the description above ⊙.	String
sc$i$	Strings defining the connection of ui inputs and yi outputs to the parameters, $i = 0, \ldots, 15$, see details above	String

SILO – Save input value, load output value

Block Symbol

Licence: STANDARD



Function Description

The SILO block can be used to export or import a single value to/from a file. The value is saved when a rising edge (off→on) occurs at the SAVE input and the value is also set to the y output. The value is loaded at startup and when a rising edge (off→on) occurs at the LOAD input.

The outputs E and lastErr indicate an error during disk operation. The E indicator is reset on falling edge at the SAVE or LOAD input while the lastErr output holds the value until another disk operation is invoked. If the error occurs during the LOAD operation, a substitute value yerr is set to the y output.

Alternatively it is possible to write or read the value continuously if the corresponding flag (CSF, CLF) is set to on. The disk operation is then performed when the corresponding input is set to on. Beware, in that case the disk operation is executed in each cycle, which can cause excessive use of the storage medium. Thus it is necessary to use this feature with caution.

The fname parameter defines the location of the file on the target platform. The path is relative to the data folder of the RexCore runtime module.

Use the SGSLP function block for advanced and complex operations.

Inputs

u	Input signal	Double (F64)
SAVE	Save value to file	Bool
LOAD	Load value from file	Bool

Parameters

fname	Name of persistent storage file	String
CSF	Flag for continuous saving	Bool
CLF	Flag for continuous loading	Bool
yerr	Substitute value for an error case	Double (F64)

Outputs

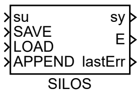
y	Output signal	Double (F64)
E	Error flag	Bool

`lastErr`	Result of last operation	`Long (I32)`

SILOS – Save input string, load output string

Block Symbol

Licence: STANDARD



Function Description

The SILOS block can be used to export or import a string to/from a file. The string is saved when a rising edge (off→on) occurs at the SAVE input and the string is also set to the sy output. The string is loaded at startup and when a rising edge (off→on) occurs at the LOAD input.

If a logical true (on) is brought to the APPEND input, the input string is added to the end of the file when it is saved. This mode is useful for logging events into text files. This input signal has no effect on loading from the file.

The LLO parameter is intended for choosing whether to load the entire file (off) or its last line only (on).

The outputs E and lastErr indicate an error during disk operation. The E indicator is reset on falling edge at the SAVE or LOAD input while the lastErr output holds the value until another disk operation is invoked.

Alternatively it is possible to write or read the string continuously if the corresponding flag (CSF, CLF) is set to on. The disk operation is then performed when the corresponding input is set to on. Beware, in that case the disk operation is executed in each cycle, which can cause excessive use of the storage medium. Thus it is necessary to use this feature with caution.

The fname parameter defines the location of the file on the target platform. The path is relative to the data folder of the RexCore runtime module.

Inputs

su	String input of the block	⊙0	String
SAVE	Save string to file		Bool
LOAD	Load string from file		Bool
APPEND	Append saved string to file		Bool

Outputs

sy	String output of the block	String
E	Error indicator off . . . No error on . . . . An error occurred	Bool
lastErr	Result of last operation	Long (I32)

Parameters

fname	Name of persistent storage file		String
CSF	Continuous saving		Bool
CLF	Continuous loading		Bool
LLO	Last line only loading		Bool
nmax	Allocated size of string	↓0 ↑65520	Long (I32)

Chapter 13

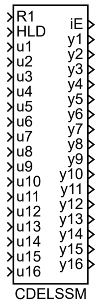
MODEL – Dynamic systems simulation

Contents

CDELSSM – Continuous state space model of a linear system with time delay

Block Symbol

Licence: ADVANCED



Function Description

The CDELSSM block (Continuous State Space Model with time DELay) simulates behavior of a linear system with time delay del

$$\begin{aligned} \frac{dx(t)}{dt} &= A_c x(t) + B_c u(t - del),\ x(0) = x0 \\ y(t) &= C_c x(t) + D_c u(t), \end{aligned}$$

where $x(t) \in \mathbb{R}^n$ is the state vector, $x0 \in \mathbb{R}^n$ is the initial value of the state vector, $u(t) \in \mathbb{R}^m$ is the input vector, $y(t) \in \mathbb{R}^p$ is the output vector. The matrix $A_c \in \mathbb{R}^{n\times n}$ is the system dynamics matrix, $B_c \in \mathbb{R}^{n\times m}$ is the input matrix, $C_c \in \mathbb{R}^{p\times n}$ is the output matrix and $D_c \in \mathbb{R}^{p\times m}$ is the direct transmission (feedthrough) matrix.

All matrices are specified in the same format as in Matlab, i.e. the whole matrix is placed in brackets, elements are entered by rows, elements of a row are separated by spaces (blanks), rows are separated by semicolons. The $x0$ vector is a column, therefore the elements are separated by semicolons (each element is in a separate row).

The simulated system is first converted to the discrete (discretized) state space model

$$\begin{aligned} x((k+1)T) &= A_d x(kT) + B_{d1} u((k-d)T) + B_{d2} u((k-d+1)T),\ x(0) = x0 \\ y(kT) &= C_c x(kT) + D_c u(kT), \end{aligned}$$

where $k \in \{1, 2, \ldots\}$ is the simulation step, T is the execution period of the block in seconds and d is a delay in simulation step such that $(d-1)T < del \le d.T$. The period T is not entered in the block, it is determined automatically as a period of the task (TASK, QTASK nebo IOTASK) containing the block.

If the input $u(t)$ is changed only in the moments of sampling and between two consecutive sampling instants is constant, i.e. $u(t) = u(kT)$ for $t \in [kT, (k+1)T)$, then the matrices A_d, B_{d1} and B_{d2} are determined by

$$\begin{aligned} A_d &= e^{A_c T} \\ B_{d1} &= e^{A_c(T-\Delta)} \int_0^{\Delta} e^{A_c \tau} B_c d\tau \\ B_{d2} &= \int_0^{T-\Delta} e^{A_c \tau} B_c d\tau, \end{aligned}$$

where $\Delta = del - (d-1)T$.

Computation of discrete matrices A_d, B_{d1} and B_{d2} is based on a method described in [5], which uses Padé approximations of matrix exponential and its integral and scaling technique.

During the real-time simulation, single simulation step of the above discrete state space model is computed in each execution time instant.

Inputs

`R1`	Reset signal. When `R1` = `on`, the state vector `x` is set to its initial value `x0`. The simulation continues on the falling edge of `R1` (`on`→`off`).	`Bool`
`HLD`	Simulation output holds its value if `HLD`=`on`.	`Bool`
`u1..u16`	Simulated system inputs. First m simulation inputs are used where m is the number of columns of the matrix `Bc`.	`Double (F64)`

Outputs

`iE`	Block error code `0` ..... O.K., the simulation runs correctly `-213` .. incompatibility of the state space model matrices dimensions `-510` .. the model is badly conditioned (some working matrix is singular or nearly singular) `xxx` ... error code `xxx` of REXYGEN, see appendix C for details	`Error`
`y1..y16`	Simulated system outputs. First p simulation outputs are used where p is the number of rows of the matrix `Cc`.	`Double (F64)`

Parameters

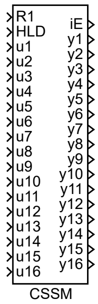
`UD`	Matrix `Dc` usage flag. If `UD`=`off`then the `Dc` matrix is not used for simulation (simulation behaves as if the `Dc` matrix is zero).	`Bool`
`del`	Model time delay [s]. ↓0.0	`Double (F64)`
`is`	Order of the Padé approximation of the matrix exponential for the computation of the discretized system matrices. ↓0 ↑4 ⊙2	`Long (I32)`
`eps`	Required accuracy of the Padé approximation. ↓0.0 ↑1.0 ⊙1e-15	`Double (F64)`
`Ac`	Matrix $(n \times n)$ of the continuous linear system dynamics.	`Double (F64)`

Bc	Input matrix $(n \times m)$ of the continuous linear system.	Double (F64)
Cc	Output matrix $(p \times n)$ of the continuous linear system.	Double (F64)
Dc	Direct transmission (feedthrough) matrix $(p \times m)$ of the continuous linear system. The matrix is used only if the parameter UD=on. If UD=off, the dimensions of the Dc matrix are not checked.	Double (F64)
x0	Initial value of the state vector (of dimension n) of the continuous linear system.	Double (F64)

CSSM – Continuous state space model of a linear system

Block Symbol

Licence: ADVANCED



Function Description

The CSSM block (Continuous State Space Model) simulates behavior of a linear system

$$\begin{aligned}\frac{dx(t)}{dt} &= A_c x(t) + B_c u(t),\ x(0) = x0\\ y(t) &= C_c x(t) + D_c u(t),\end{aligned}$$

where $x(t) \in \mathbb{R}^n$ is the state vector, $x0 \in \mathbb{R}^n$ is the initial value of the state vector, $u(t) \in \mathbb{R}^m$ is the input vector, $y(t) \in \mathbb{R}^p$ is the output vector. The matrix $A_c \in \mathbb{R}^{n\times n}$ is the system dynamics matrix, $B_c \in \mathbb{R}^{n\times m}$ is the input matrix, $C_c \in \mathbb{R}^{p\times n}$ is the output matrix and $D_c \in \mathbb{R}^{p\times m}$ is the direct transmission (feedthrough) matrix.

All matrices are specified in the same format as in Matlab, i.e. the whole matrix is placed in brackets, elements are entered by rows, elements of a row are separated by spaces (blanks), rows are separated by semicolons. The $x0$ vector is a column, therefore the elements are separated by semicolons (each element is in a separate row).

The simulated system is first converted to the discrete (discretized) state space model

$$\begin{aligned}x((k+1)T) &= A_d x(kT) + B_d u(kT),\ x(0) = x0\\ y(kT) &= C_c x(kT) + D_c u(kT),\end{aligned}$$

where $k \in \{1, 2, \ldots\}$ is the simulation step, T is the execution period of the block in seconds. The period T is not entered in the block, it is determined automatically as a period of the task (TASK, QTASK nebo IOTASK) containing the block.

If the input $u(t)$ is changed only in the moments of sampling and between two consecutive sampling instants is constant, i.e. $u(t) = u(kT)$ for $t \in [kT, (k+1)T)$, then the matrices A_d and B_d are determined by

$$\begin{aligned}A_d &= e^{A_c T}\\ B_d &= \int_0^T e^{A_c \tau} B_c d\tau\end{aligned}$$

Computation of discrete matrices A_d and B_d is based on a method described in [5], which uses Padé approximations of matrix exponential and its integral and scaling technique.

During the real-time simulation, single simulation step of the above discrete state space model is computed in each execution time instant.

Inputs

`R1`	Reset signal. When `R1` = `on`, the state vector `x` is set to its initial value `x0`. The simulation continues on the falling edge of `R1` (`on`→`off`).	`Bool`
`HLD`	Simulation output holds its value if `HLD`=`on`.	`Bool`
`u1..u16`	Simulated system inputs. First m simulation inputs are used where m is the number of columns of the matrix `Bc`.	`Double (F64)`

Outputs

`iE`	Block error code `0` ..... O.K., the simulation runs correctly `-213` .. incompatibility of the state space model matrices dimensions `-510` .. the model is badly conditioned (some working matrix is singular or nearly singular) `xxx` ... error code `xxx` of REXYGEN, see appendix C for details	`Error`
`y1..y16`	Simulated system outputs. First p simulation outputs are used where p is the number of rows of the matrix `Cc`.	`Double (F64)`

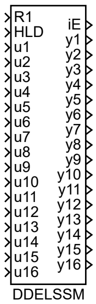
Parameters

`UD`	Matrix `Dc` usage flag. If `UD`=`off`then the `Dc` matrix is not used for simulation (simulation behaves as if the `Dc` matrix is zero).	`Bool`
`is`	Order of the Padé approximation of the matrix exponential for the computation of the discretized system matrices. ↓0 ↑4 ⊙2	`Long (I32)`
`eps`	Required accuracy of the Padé approximation. ↓0.0 ↑1.0 ⊙1e-15	`Double (F64)`
`Ac`	Matrix $(n \times n)$ of the continuous linear system dynamics.	`Double (F64)`
`Bc`	Input matrix $(n \times m)$ of the continuous linear system.	`Double (F64)`
`Cc`	Output matrix $(p \times n)$ of the continuous linear system.	`Double (F64)`
`Dc`	Direct transmission (feedthrough) matrix $(p \times m)$ of the continuous linear system. The matrix is used only if the parameter `UD`=`on`. If `UD`=`off`, the dimensions of the `Dc` matrix are not checked.	`Double (F64)`
`x0`	Initial value of the state vector (of dimension n) of the continuous linear system.	`Double (F64)`

DDELSSM – Discrete state space model of a linear system with time delay

Block Symbol

Licence: ADVANCED



Function Description

The DDELSSM block (Discrete State Space Model with time DELay) simulates behavior of a linear system with time delay del

$$
\begin{aligned}
x(k+1) &= A_d x(k) + B_d u(k-d),\ x(0) = x0 \\
y(k) &= C_d x(k) + D_d u(k),
\end{aligned}
$$

where k is the simulation step, $x(k) \in \mathbb{R}^n$ is the state vector, $x0 \in \mathbb{R}^n$ is the initial value of the state vector, $u(k) \in \mathbb{R}^m$ is the input vector, $y(k) \in \mathbb{R}^p$ is the output vector. The matrix $A_d \in \mathbb{R}^{n\times n}$ is the system dynamics matrix, $B_d \in \mathbb{R}^{n\times m}$ is the input matrix, $C_d \in \mathbb{R}^{p\times n}$ is the output matrix and $D_d \in \mathbb{R}^{p\times m}$ is the direct transmission (feedthrough) matrix. Number of steps of the delay d is the largest integer such that $d.T \leq del$, where T is the block execution period.

All matrices are specified in the same format as in Matlab, i.e. the whole matrix is placed in brackets, elements are entered by rows, elements of a row are separated by spaces (blanks), rows are separated by semicolons. The $x0$ vector is a column, therefore the elements are separated by semicolons (each element is in a separate row).

During the real-time simulation, single simulation step of the above discrete state space model is computed in each execution time instant.

Inputs

R1	Reset signal. When R1 = on, the state vector x is set to its initial value x0. The simulation continues on the falling edge of R1 (on→off).	Bool
HLD	Simulation output holds its value if HLD=on.	Bool
u1..u16	Simulated system inputs. First m simulation inputs are used where m is the number of columns of the matrix Bd.	Double (F64)

Outputs

iE	Block error code 0 ..... O.K., the simulation runs correctly -213 .. incompatibility of the state space model matrices dimensions xxx ... error code xxx of REXYGEN, see appendix C for details	Error
y1..y16	Simulated system outputs. First p simulation outputs are used where p is the number of rows of the matrix Cd.	Double (F64)

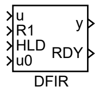
Parameters

UD	Matrix Dd usage flag. If UD=offthen the Dd matrix is not used for simulation (simulation behaves as if the Dd matrix is zero).	Bool
del	Model time delay [s]. ↓0.0	Double (F64)
Ad	Matrix $(n \times n)$ of the discrete linear system dynamics.	Double (F64)
Bd	Input matrix $(n \times m)$ of the discrete linear system.	Double (F64)
Cd	Output matrix $(p \times n)$ of the discrete linear system.	Double (F64)
Dd	Direct transmission (feedthrough) matrix $(p \times m)$ of the discrete linear system. The matrix is used only if the parameter UD=on. If UD=off, the dimensions of the Dd matrix are not checked.	Double (F64)
x0	Initial value of the state vector (of dimension n) of the discrete linear system.	Double (F64)

DFIR – Discrete finite input response filter

Block Symbol

Licence: ADVANCED



Function Description

The DFIR block is a filter whose impulse response (or response to any finite length input) is of finite duration, because it settles to zero in finite time. The calculation takes place in the form of a convolutional integral (sum) - the impulse characteristic is entered in the hk field already in discretized form for the correct period.

Input

u	Analog input of the block	Double (F64)
R1	Block reset	Bool
HLD	Hold – the block code is not executed if the input is set to on	Bool
u0	Initial input value (fill buffer)	Double (F64)

Output

y	Analog output of the block	Double (F64)
RDY	Ready flag	Bool

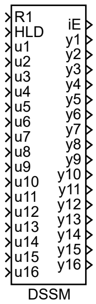
Parameters

nmax	Allocated size of array	↓10 ↑10000000 ⊙100	Long (I32)
hk	hk	⊙[0.6 0.3 0.1]	Double (F64)

DSSM – Discrete state space model of a linear system

Block Symbol

Licence: ADVANCED



Function Description

The DSSM block (Discrete State Space Model) simulates behavior of a linear system

$$\begin{aligned} x(k+1) &= A_d x(k) + B_d u(k), \; x(0) = x0 \\ y(k) &= C_d x(k) + D_d u(k), \end{aligned}$$

where k is the simulation step, $x(k) \in \mathbb{R}^n$ is the state vector, $x0 \in \mathbb{R}^n$ is the initial value of the state vector, $u(k) \in \mathbb{R}^m$ is the input vector, $y(k) \in \mathbb{R}^p$ is the output vector. The matrix $A_d \in \mathbb{R}^{n \times n}$ is the system dynamics matrix, $B_d \in \mathbb{R}^{n \times m}$ is the input matrix, $C_d \in \mathbb{R}^{p \times n}$ is the output matrix and $D_d \in \mathbb{R}^{p \times m}$ is the direct transmission (feedthrough) matrix.

All matrices are specified in the same format as in Matlab, i.e. the whole matrix is placed in brackets, elements are entered by rows, elements of a row are separated by spaces (blanks), rows are separated by semicolons. The $x0$ vector is a column, therefore the elements are separated by semicolons (each element is in a separate row).

During the real-time simulation, single simulation step of the above discrete state space model is computed in each execution time instant.

Inputs

R1	Reset signal. When R1 = on, the state vector x is set to its initial value x0. The simulation continues on the falling edge of R1 (on→off).	Bool
HLD	Simulation output holds its value if HLD=on.	Bool
u1..u16	Simulated system inputs. First m simulation inputs are used where m is the number of columns of the matrix Bd.	Double (F64)

Outputs

iE	Block error code 0 ..... O.K., the simulation runs correctly -213 .. incompatibility of the state space model matrices dimensions xxx ... error code xxx of REXYGEN, see appendix C for details	Error
y1..y16	Simulated system outputs. First p simulation outputs are used where p is the number of rows of the matrix Cd.	Double (F64)

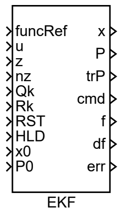
Parameters

UD	Matrix Dd usage flag. If UD=offthen the Dd matrix is not used for simulation (simulation behaves as if the Dd matrix is zero).	Bool
Ad	Matrix $(n \times n)$ of the discrete linear system dynamics.	Double (F64)
Bd	Input matrix $(n \times m)$ of the discrete linear system.	Double (F64)
Cd	Output matrix $(p \times n)$ of the discrete linear system.	Double (F64)
Dd	Direct transmission (feedthrough) matrix $(p \times m)$ of the discrete linear system. The matrix is used only if the parameter UD=on. If UD=off, the dimensions of the Dd matrix are not checked.	Double (F64)
x0	Initial value of the state vector (of dimension n) of the discrete linear system.	Double (F64)

EKF – Extended (nonlinear) Kalman filter

Block Symbol

Licence: MODEL



Function Description

The block implements a nonlinear state estimator known as Extended Kalman filter. The goal is to provide estimates of unmeasurable state quantities of a nonlinear dynamic system described by a state space model $\mathrm{d}x/\mathrm{d}t = f(x, u) + w(t), y = h(x, u) + v(t)$ for a continuous-time case and $x(k+1) = f(x(k), u(k)) + w(k), y(k) = h(x(k), u(k)) + v(k)$ for the case of a discrete-time system. The variables w, v are the process and observation noises which are both assumed to be zero mean multivariate Gaussian processes with covariance Q and R specified in the block parameters. The Extended Kalman filter is the nonlinear version of the Kalman filter which linearizes the state and output equations about the current working point. It is a predictor-corrector type algorithm which switches between open-loop prediction using the state equation and correction of the estimates by directly measured output quantities. The measurements can be supplied to the filter non-equidistantly in an arbitrary execution period of the block.

The prediction step is run in each execution period and solves the state equation by numerical integration, starting from an initial value $x0$ and initial covariance $P0$. Various numerical methods, chosen by the user specified parameter *solver*, are available to perform the integration of the vector state differential equation. A special choice of $solver = 1$ signalizes the discrete-time system case for which the numerical integration reduces to simple evaluation of the recursive formula given by the first-order difference equation in $x(k + 1) = f(x(k), u(k))$. Apart from the state vector, also its covariance matrix P is propagated in time, capturing the uncertainty of the estimates in the form of their (co)variances. Please refer to the documentation of the NSSM block for more details about the available numerical integration algorithms.

The filtering correction step takes place whenever the input of the block is set to $nz > 0$. This signalizes that new vector of measurements is available at the z input and it is used to correct the state and its covariance estimates from the prediction step. Multiple right sides of the output equation can be implemented in the cooperating REXLANG block. This may be useful e.g. for systems equipped with various sensors providing their data asynchronously to each other (and with respect to the block execution times) with different sampling periods. For the setting $nz = 0$, the user algorithm signalizes no output data available in the current execution period, forcing the filter to extrapolate the state estimates by performing the prediction step only.

The Extended Kalman filter is generally not an optimal filter in the sense of minimization of the mean-squared error of the obtained state estimates. However, it provides modest performance for sufficiently smooth nonlinear systems and is considered to be a de facto standard solution for nonlinear estimation. A special case is obtained by setting linear state and output equations in the cooperating `REXLANG` block. This case leads to standard linear Kalman filter which is stochastically optimal for the formulated state estimation problem.

Inputs

`funcRef`	Cooperating REXLANG block reference		`Reference`
`u`	Input vector of the model		`Reference`
`z`	Output (mesurement) vector of the model		`Reference`
`nz`	Index of the actual output vector set	↓1	`Long (I32)`
`Qk`	State noise covariance matrix		`Reference`
`Rk`	Output noise covariance matrix		`Reference`
`RST`	Block reset		`Bool`
`HLD`	Hold		`Bool`
`x0`	Initial state vector		`Reference`
`P0`	Initial covariance matrix		`Reference`

Parameters

`nmax`	Allocated size of output matrix (total number of items)	↓5 ↑10000 ⊙20	`Long (I32)`
`solver`	Numeric integration method	⊙2	`Long (I32)`

1 ..... Discrete equation
2 ..... Euler (1st order)
3 ..... 2nd order Adams-Bashforth
4 ..... 3rd order Adams-Bashforth
5 ..... 4th order Adams-Bashforth
6 ..... 5th order Adams-Bashforth
7 ..... 4th order Runge-Kutha
8 ..... Implicit Euler
9 ..... Implicit Euler(more iteration)
10 .... 2nd order Adams-Moulton implicit
11 .... 2nd order Adams-Moulton implicit (more iterations)
12 .... 3rd order Adams-Moulton implicit
13 .... 3nd order Adams-Moulton implicit (more iterations)
14 .... 2nd order RadauIIA implicit
15 .... 2nd order RadauIIA implicit (more iterations)
16 .... 3rd order RadauIIA implicit
17 .... 3rd order RadauIIA implicit (more iterations)

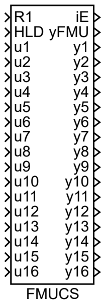
Outputs

`x`	Model state vector	`Reference`
`P`	Model state covariance matrix	`Reference`
`trP`	Trace of model state covariance matrix	`Reference`
`cmd`	Cooperating REXLANG block requested function	`Long (I32)`
`f`	Vector reference set by cooperating REXLANG block	`Reference`
`df`	Matrix reference set by cooperating REXLANG block	`Reference`
`err`	Error code (0 is OK, see SystemLog for details)	`Long (I32)`

FMUCS – * Import modelu FMU CS (pro Co-Simulation)

Block Symbol

Licence: ADVANCED



Function Description

The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

Inputs

R1	Reset bloku	Bool
HLD	Podržení aktuálního stavu modelu	Bool
u1	Analogový vstupní signál	Double (F64)
u2	Analogový vstupní signál	Double (F64)
u3	Analogový vstupní signál	Double (F64)
u4	Analogový vstupní signál	Double (F64)
u5	Analogový vstupní signál	Double (F64)
u6	Analogový vstupní signál	Double (F64)
u7	Analogový vstupní signál	Double (F64)
u8	Analogový vstupní signál	Double (F64)
u9	Analogový vstupní signál	Double (F64)
u10	Analogový vstupní signál	Double (F64)
u11	Analogový vstupní signál	Double (F64)
u12	Analogový vstupní signál	Double (F64)
u13	Analogový vstupní signál	Double (F64)
u14	Analogový vstupní signál	Double (F64)
u15	Analogový vstupní signál	Double (F64)
u16	Analogový vstupní signál	Double (F64)

Parameters

tstop	Koncový čas simulace	↓1e-06 ⊙1.0	Double (F64)

eps	Přesnost aproximace ↓0.0 ↑1.0 ⊙1e-06	Double (F64)
loglevel	Úroveň protokolování knihovny FMI do systémového logu ↓0 ↑7 ⊙2 0 ..... Nic 1 ..... Fatální 2 ..... Chyba 3 ..... Varování 4 ..... Info 5 ..... Podrobný 6 ..... Ladění 7 ..... Všechno	Long (I32)
SelPars	Seznam vybraných parametrů FMU	String
TUNEALLP	Považuj všechny vybrané parametry za laditelné parametry	Bool
p1	Analogový parametr bloku	Double (F64)
p2	Analogový parametr bloku	Double (F64)
p3	Analogový parametr bloku	Double (F64)
p4	Analogový parametr bloku	Double (F64)
p5	Analogový parametr bloku	Double (F64)
p6	Analogový parametr bloku	Double (F64)
p7	Analogový parametr bloku	Double (F64)
p8	Analogový parametr bloku	Double (F64)
p9	Analogový parametr bloku	Double (F64)
p10	Analogový parametr bloku	Double (F64)
p11	Analogový parametr bloku	Double (F64)
p12	Analogový parametr bloku	Double (F64)
p13	Analogový parametr bloku	Double (F64)
p14	Analogový parametr bloku	Double (F64)
p15	Analogový parametr bloku	Double (F64)
p16	Analogový parametr bloku	Double (F64)

Outputs

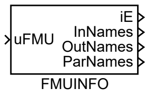
iE	Kód chyby	Error
yFMU	Výstupní odkaz na instanci FMU	Reference
y1	Analogový výstupní signál	Double (F64)
y2	Analogový výstupní signál	Double (F64)
y3	Analogový výstupní signál	Double (F64)
y4	Analogový výstupní signál	Double (F64)
y5	Analogový výstupní signál	Double (F64)
y6	Analogový výstupní signál	Double (F64)
y7	Analogový výstupní signál	Double (F64)
y8	Analogový výstupní signál	Double (F64)
y9	Analogový výstupní signál	Double (F64)
y10	Analogový výstupní signál	Double (F64)
y11	Analogový výstupní signál	Double (F64)

y12	Analogový výstupní signál	Double (F64)
y13	Analogový výstupní signál	Double (F64)
y14	Analogový výstupní signál	Double (F64)
y15	Analogový výstupní signál	Double (F64)
y16	Analogový výstupní signál	Double (F64)

FMUINFO – * Imformace o importovaném modelu FMU

Block Symbol

Licence: ADVANCED



Function Description

The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

Vstup

uFMU	Vstupní odkaz na instanci FMU		Reference

Parameters

SelPars	Seznam vybraných parametrů FMU		String
Separ	Oddělovač jmen v řetězcových výstupech	⊙,	String

Outputs

iE	Kód chyby	Error
InNames	Seznam jmen vstupů FMU	String
OutNames	Seznam jmen výstupů FMU	String
ParNames	Seznam jmen vybraných parametrů FMU	String

FOPDT – First order plus dead-time model

Block Symbol

Licence: STANDARD

Function Description

The FOPDT block is a discrete simulator of a first order continuous-time system with time delay, which can be described by the transfer function below:

$$P(s) = \frac{\mathtt{k0}}{(\mathtt{tau} \cdot s + 1)} \cdot e^{-\mathtt{del} \cdot s}$$

The exact discretization at the sampling instants is used for discretization of the $P(s)$ transfer function. The sampling period used for discretization is equivalent to the execution period of the FOPDT block.

Input

u	Analog input of the block	Double (F64)

Output

y	Analog output of the block	Double (F64)

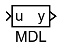
Parameters

k0	Static gain	⊙1.0	Double (F64)
del	Dead time [s]		Double (F64)
tau	Time constant	⊙1.0	Double (F64)
nmax	Size of delay buffer (number of samples) for the time delay del. Used for internal memory allocation.	↓10 ↑10000000 ⊙1000	Long (I32)

MDL – Process model

Block Symbol

Licence: STANDARD



Function Description

The MDL block is a discrete simulator of continuous-time system with transfer function

$$F(s) = \frac{K_0 e^{-Ds}}{(\tau_1 s + 1)(\tau_2 s + 1)},$$

where $K_0 > 0$ is the static gain k0, $D \geq 0$ is the time-delay del and $\tau_1, \tau_2 > 0$ are the system time-constants tau1 and tau2.

Input

u	Analog input of the block	Double (F64)

Output

y	Analog output of the block	Double (F64)

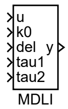
Parameters

k0	Static gain	⊙1.0	Double (F64)
del	Dead time [s]		Double (F64)
tau1	The first time constant	⊙1.0	Double (F64)
tau2	The second time constant	⊙2.0	Double (F64)
nmax	Size of delay buffer (number of samples) for the time delay del. Used for internal memory allocation.	↓10 ↑10000000 ⊙1000	Long (I32)

MDLI – Process model with input-defined parameters

Block Symbol

Licence: STANDARD



Function Description

The MDLI block is a discrete simulator of continuous-time system with transfer function

$$F(s) = \frac{K_0 e^{-Ds}}{(\tau_1 s + 1)(\tau_2 s + 1)},$$

where $K_0 > 0$ is the static gain k0, $D \geq 0$ is the time-delay del and $\tau_1, \tau_2 > 0$ are the system time-constants tau1 and tau2. In contrary to the MDL block the system is time variant. The system parameters are determined by the input signals.

Inputs

u	Analog input of the block	Double (F64)
k0	Static gain	Double (F64)
del	Dead time [s]	Double (F64)
tau1	The first time constant	Double (F64)
tau2	The second time constant	Double (F64)

Output

y	Analog output of the block	Double (F64)

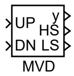
Parameters

nmax	Size of delay buffer (number of samples) for the time delay del. Used for internal memory allocation. ↓10 ↑10000000 ⊙1000	Long (I32)

MVD – Motorized valve drive

Block Symbol

Licence: STANDARD



Function Description

The MVD block simulates a servo valve. The UP (DN) input is a binary command for opening (closing) the valve at a constant speed 1/tv, where tv is a parameter of the block. The opening (closing) continues for UP = on (DN = on) until the full open y = hilim (full closed y = lolim) position is reached. The full open (full closed) position is signalized by the end switch HS (LS). The initial position at start-up is y = y0. If UP = DN = on or UP = DN = off, then the position of the valve remains unchanged (neither opening nor closing).

Inputs

UP	Open	Bool
DN	Close	Bool

Outputs

y	Valve position	Double (F64)
HS	Upper end switch	Bool
LS	Lower end switch	Bool

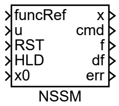
Parameters

y0	Initial valve position		Double (F64)
tv	Time required for transition between y = 0 and y = 1 [s]	⊙10.0	Double (F64)
hilim	Upper limit position (open)	⊙1.0	Double (F64)
lolim	Lower limit position (closed)		Double (F64)

NSSM – **Nonlinear State-Space Model**

Block Symbol

Licence: MODEL



Function Description

The block provides a solution to a nonlinear continuous-time state-space model in the form of $\mathrm{d}x/\mathrm{d}t = f(x,u), y = h(x,u)$ or its discrete-time counterpart defined as $x(k+1) = f(x(k), u(k)), y(k) = h(x(k), u(k))$. The equation is discretized into a form $x(t) = F(x(t-T), u(t))$, where T is sampling period of the NSSM block. The method used for discretization (i.e. a method to numerically solve the vector differential equation) depends on the `solver` parameter . Various methods for numerical integration are implemented including one step methods (like Runge-Kutta, Euler), multistep methods (Adams-Bashforth), and also implicit methods (Adams-Moulton). It is possible to choose different method order for each kind to find a suitable precision vs computational time trade-off. The block does not support variable step algorithms (the time-step for the solver is always the same as the execution period of the task where the block is inserted).

The non-linear-vector function $f(x,u)$ must be implemented in the REXLANG block that is connected to the NSSM block in a special way. The input `funcRef` of the NSSM block must be connected to the output `y0` of the REXLANG block and the output `y0` can not be used internally in the code/script of the REXLANG block. The outputs `x`, `f` and `df` of the NSSM block must be connected to the inputs of the REXLANG block. These inputs must be processed in the REXLANG code as an input array. The main function of the REXLANG block must set the value of $f(x,u)$ into the `f` vector (e.g. into the input array, where f is connected) and the matrix $\mathrm{d}f(x,u)/\mathrm{d}x$ into the `af` matrix.

The NSSM block calls the main-function of the REXLANG block when needed for numerical integration of the differential equation system (for example the Runge-Kutta method performs 4 calls in each execution period with different x-vector values). The REXLANG block should be disabled in the schematics of the algorithm to prevent its execution REXYGEN system itself. If the REXLANG must be executed by REXYGEN (e.g. for compute output function $y = h(x,u)$), it is recommended to connect the output `cmd` of the NSSM block into input of the REXLANG block to distinguish between calling by the NSSM block (`cmd` $= 0$) and calling by REXYGEN system (`cmd` $= -1$).

Notes:

- computation of the $\mathrm{d}f(x,u)/\mathrm{d}x$ is necessary for implicit methods only (explicit methods do not use it).

- size of the vector `x` (and also `f`, `df`) is defined by the size of the vector `x0`. The size should be changed by reset only (the `RST` input).
- `solver=1: discrete` signalizes a discrete-time state space model with the functions `f` and `h` designating the right side of the corresponding difference equation. This mode does not require numerical integration and the algorithm reduces to the execution of the code in the connecnted REXLANG block; the mode is used mainly for symmetry with the `EKF` block.
- for `NSSM` connecting the output `cmd` is necessary, because `cmd>0` indicate number of measurement and REXLANG must return $f = h(x, u)$, $\mathrm{d}f = \mathrm{d}h(x, u)/\mathrm{d}x$.

Inputs

`funcRef`	Cooperating REXLANG block reference	`Reference`
`u`	Input vector of the model	`Reference`
`RST`	Block reset	`Bool`
`HLD`	Hold	`Bool`
`x0`	Initial state vector	`Reference`

Parameters

`nmax`	Allocated size of output matrix (total number of items) ↓5 ↑10000 ⊙20	`Long (I32)`
`solver`	Numeric integration method ⊙2 1 ..... Discrete equation 2 ..... Euler (1st order) 3 ..... 2nd order Adams-Bashforth 4 ..... 3rd order Adams-Bashforth 5 ..... 4th order Adams-Bashforth 6 ..... 5th order Adams-Bashforth 7 ..... 4th order Runge-Kutta 8 ..... Implicit Euler 9 ..... Implicit Euler(more iteration) 10 .... 2nd order Adams-Moulton implicit 11 .... 2nd order Adams-Moulton implicit (more iterations) 12 .... 3rd order Adams-Moulton implicit 13 .... 3nd order Adams-Moulton implicit (more iterations) 14 .... 2nd order RadauIIA implicit 15 .... 2nd order RadauIIA implicit (more iterations) 16 .... 3rd order RadauIIA implicit 17 .... 3rd order RadauIIA implicit (more iterations)	`Long (I32)`

Outputs

`x`	Model state vector	`Reference`
`y`	Model output vector	`Reference`

`cmd`	Cooperating REXLANG block requested function	`Long (I32)`
`f`	Vector reference set by cooperating REXLANG block	`Reference`
`df`	Matrix reference set by cooperating REXLANG block	`Reference`
`err`	Error code (0 is OK, see SystemLog for details)	`Long (I32)`

SOPDT – Second order plus dead-time model

Block Symbol

Licence: STANDARD

Function Description

The SOPDT block is a discrete simulator of a second order continuous-time system with time delay, which can be described by one of the transfer functions below. The type of the model is selected by the itf parameter.

$$\begin{aligned}
\mathtt{itf} = 1: \quad P(s) &= \frac{\mathtt{pb1} \cdot s + \mathtt{pb0}}{s^2 + \mathtt{pa1} \cdot s + \mathtt{pa0}} \cdot e^{-\mathtt{del} \cdot s} \\
\mathtt{itf} = 2: \quad P(s) &= \frac{\mathtt{k0}\,(\mathtt{tau} \cdot s + 1)}{(\mathtt{tau1} \cdot s + 1)\,(\mathtt{tau2} \cdot s + 1)} \cdot e^{-\mathtt{del} \cdot s} \\
\mathtt{itf} = 3: \quad P(s) &= \frac{\mathtt{k0} \cdot \mathtt{om}^2 \cdot (\mathtt{tau}/\mathtt{om} \cdot s + 1)}{(s^2 + 2 \cdot \mathtt{xi} \cdot \mathtt{om} \cdot s + \mathtt{om}^2)} \cdot e^{-\mathtt{del} \cdot s} \\
\mathtt{itf} = 4: \quad P(s) &= \frac{\mathtt{k0}\,(\mathtt{tau} \cdot s + 1)}{(\mathtt{tau1} \cdot s + 1)\, s} \cdot e^{-\mathtt{del} \cdot s}
\end{aligned}$$

For simulation of first order plus dead time systems (FOPDT) use the LLC block with parameter a set to zero.

The exact discretization at the sampling instants is used for discretization of the $P(s)$ transfer function. The sampling period used for discretization is equivalent to the execution period of the SOPDT block.

Input

u	Analog input of the block	Double (F64)

Output

y	Analog output of the block	Double (F64)

Parameters

itf	Transfer function form 1 ..... A general form 2 ..... A form with real poles 3 ..... A form with complex poles 4 ..... A form with integrator	⊙1	Long (I32)

k0	Static gain	⊙1.0	Double (F64)
tau	Numerator time constant		Double (F64)
tau1	The first time constant	⊙1.0	Double (F64)
tau2	The second time constant	⊙1.0	Double (F64)
om	Natural frequency	⊙1.0	Double (F64)
xi	Relative damping coefficient	⊙1.0	Double (F64)
pb0	Numerator coefficient: s^0	⊙1.0	Double (F64)
pb1	Numerator coefficient: s^1	⊙1.0	Double (F64)
pa0	Denominator coefficient: s^0	⊙1.0	Double (F64)
pa1	Denominator coefficient: s^1	⊙1.0	Double (F64)
del	Dead time [s]		Double (F64)
nmax	Size of delay buffer (number of samples) for the time delay del. Used for internal memory allocation.	↓10 ↑10000000 ⊙1000	Long (I32)

Chapter 14

MATRIX – Blocks for matrix and vector operations

Contents

Implementation notice: First element of a matrix has index (0,0), first element of a vector has index (0).

The vector is one-column-matrix, not separate object. One-row-matrix is called a row vector, but that object should not be used as vector in REXYGEN.

The matrix inputs and outputs are references. It means one block (the MX_MAT block or the MX_VEC block most often) reserve memory for the matrix and other block (using same reference) write/read same space. The MB_DCOPY block (and second the MX_MAT block) must be used to create copy of the matrix.

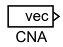
Some blocks using vector (MB_DCPY, RTOV, VTOR) not check exact dimensions (for example a 10x10 matrix is regard as 100-elements vector). Matrix is linearize into vector column by column, because a matrix is stored this way in memory (e.g. for a 10x10 matrix: element (1,0) has index 1 in vector, element (2,0) has index 2 in vector, element (0,1) has index 10 in vector, element (0,2) has index 11 in vector, etc.) These type of blocks could not be used with submatrix returned by the MX_DSAREF block. Behavior is undefined in this case.

The most matrix blocks has input and output matrix reference. Both are equal, but connecting input reference to output reference of previous block define execution order (the blocks are executed according signal flow in REXYGEN) and therfore computed matrix equation.

CNA – Array (vector/matrix) constant

Block Symbol

Licence: STANDARD



Function Description

The block CNA allocates memory for nmax elements of the type etype of the vector/matrix referenced by the output vec and initializes all elements to data stored in the parameter acn.

If the string parameter filename is not empty then it loads initalization data from the filename file on the host computer in CSV format. Column separator can be comma or semicolon or space (but the same in the whole file), decimal separator have to be dot, row separator is new line. Empty lines are skipped.

If the parameter TRN = on then the output reference vec contains transposed data.

Note: In case of etype = Large (I64), values loaded from parameter acn are converted to double-precision float due to implementation reasons, so you can loose precision for very large values. If this could be a problem, use external file for initialization which does not have this issue.

Parameters

filename	CSV data file		String
TRN	Transpose loaded matrix		Bool
nmax	Allocated size of output matrix (total number of items)	↓2 ↑10000000 ⊙100	Long (I32)
etype	Type of elements 1 ..... Bool 2 ..... Byte (U8) 3 ..... Short (I16) 4 ..... Long (I32) 5 ..... Word (U16) 6 ..... DWord (U32) 7 ..... Float (F32) 8 ..... Double (F64) -- .... 10 .... Large (I64)	⊙8	Long (I32)
acn	Initial array value	⊙[0 1 2 3]	Double (F64)

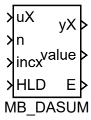
Output

vec	Reference to vector/matrix data	Reference

MB_DASUM – Sum of the absolute values

Block Symbol

Licence: STANDARD



Function Description

The output reference yX is always set to the input reference uX. If HLD = on then nothing is computed otherwise the BLAS function DASUM is called internally:

```
value = DASUM(N, uX, INCX);
```

where the values N and INCX are set in the following way:

- If the input n > 0 then N is set to n else N is set to the current number of the vector or matrix elements CNT referenced by uX.
- If the input incx > 0 then INCX is set to incx else INCX is set to 1.

The error flag E is set to on if:

- the reference uX is not defined (i.e. input uX is not connected),
- n < 0 or incx < 0,
- $(\mathtt{N} - 1) * \mathtt{INCX} + 1 > \mathtt{CNT}$.

See BLAS documentation [6] for more details.

Inputs

uX	Input reference to vector x	Reference
n	Number of processed vector elements	Long (I32)
incx	Index increment of vector x	Long (I32)
HLD	Hold	Bool

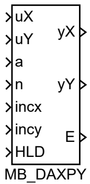
Outputs

yX	Output reference to vector x	Reference
value	Return value of the function	Double (F64)
E	Error flag	Bool

MB_DAXPY – Performs y := a*x + y for vectors x,y

Block Symbol

Licence: STANDARD



Function Description

The output references `yX` and `yY` are always set to the corresponding input references `uX` and `uY`. If `HLD = on` then nothing is computed otherwise the BLAS function `DAXPY` is called internally:

```
DAXPY(N, a, uX, INCX, uY, INCY);
```

where the values `N`, `INCX` and `INCY` are set in the following way:

- If the input `n` > 0 then `N` is set to `n` else `N` is set to the current number of the vector or matrix elements `CNTY` referenced by `uY`.
- If the input `incx` $\neq 0$ then `INCX` is set to `incx` else `INCX` is set to 1.
- If the input `incy` $\neq 0$ then `INCY` is set to `incy` else `INCY` is set to 1.

The error flag `E` is set to `on` if:

- the reference `uX` or `uY` is not defined (i.e. input `uX` or `uY` is not connected),
- `n` < 0,
- $(\texttt{N} - 1) * |\texttt{INCX}| + 1 > \texttt{CNTX}$, where `CNTX` is a number of the vector or matrix elements referenced by `uX`,
- $(\texttt{N} - 1) * |\texttt{INCY}| + 1 > \texttt{CNTY}$.

See BLAS documentation [6] for more details.

Inputs

`uX`	Input reference to vector x	`Reference`
`uY`	Input reference to vector y	`Reference`
`a`	Scalar coefficient a	`Double (F64)`
`n`	Number of processed vector elements	`Long (I32)`

`incx`	Index increment of vector x	`Long (I32)`
`incy`	Index increment of vector y	`Long (I32)`
`HLD`	Hold	`Bool`

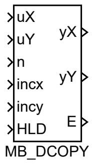
Outputs

`yX`	Output reference to vector x	`Reference`
`yY`	Output reference to vector y	`Reference`
`E`	Error indicator	`Bool`

MB_DCOPY – Copies vector x to vector y

Block Symbol

Licence: STANDARD



Function Description

The output references yX and yY are always set to the corresponding input references uX and uY. If HLD = on then nothing is computed otherwise the BLAS function DCOPY is called internally:

```
DCOPY(N, uX, INCX, uY, INCY);
```

where the values N, INCX and INCY are set in the following way:

- If the input n > 0 then N is set to n else N is set to the current number of the vector or matrix elements CNTX referenced by uX.
- If the input incx $\neq 0$ then INCX is set to incx else INCX is set to 1.
- If the input incy $\neq 0$ then INCY is set to incy else INCY is set to 1.

The error flag E is set to on if:

- the reference uX or uY is not defined (i.e. input uX or uY is not connected),
- n < 0,
- $(\mathtt{N} - 1) * |\mathtt{INCX}| + 1 > \mathtt{CNTX}$,
- $(\mathtt{N} - 1) * |\mathtt{INCY}| + 1 > \mathtt{CNTY}$, where CNTY is a number of the vector or matrix elements referenced by uY.

See BLAS documentation [6] for more details.

Inputs

uX	Input reference to vector x	Reference
uY	Input reference to vector y	Reference
n	Number of processed vector elements	Long (I32)
incx	Index increment of vector x	Long (I32)
incy	Index increment of vector y	Long (I32)
HLD	Hold	Bool

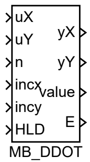
Outputs

`yX`	Output reference to vector x	`Reference`
`yY`	Output reference to vector y	`Reference`
`E`	Error indicator	`Bool`

MB_DDOT – Dot product of two vectors

Block Symbol

Licence: STANDARD



Function Description

The output references yX and yY are always set to the corresponding input references uX and uY. If HLD = on then nothing is computed otherwise the BLAS function DDOT is called internally:

```
DDOT(N, uX, INCX, uY, INCY);
```

where the values N, INCX and INCY are set in the following way:

- If the input n > 0 then N is set to n else N is set to the current number of the vector or matrix elements CNTX referenced by uX.
- If the input incx $\neq 0$ then INCX is set to incx else INCX is set to 1.
- If the input incy $\neq 0$ then INCY is set to incy else INCY is set to 1.

The error flag E is set to on if:

- the reference uX or uY is not defined (i.e. input uX or uY is not connected),
- n < 0,
- $(\mathtt{N} - 1) * |\mathtt{INCX}| + 1 > \mathtt{CNTX}$,
- $(\mathtt{N} - 1) * |\mathtt{INCY}| + 1 > \mathtt{CNTY}$, where CNTY is a number of the vector or matrix elements referenced by uY.

See BLAS documentation [6] for more details.

Inputs

uX	Input reference to vector x	Reference
uY	Input reference to vector y	Reference
n	Number of processed vector elements	Long (I32)
incx	Index increment of vector x	Long (I32)
incy	Index increment of vector y	Long (I32)
HLD	Hold	Bool

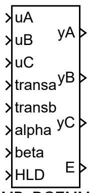
Outputs

`yX`	Output reference to vector x	`Reference`
`yY`	Output reference to vector y	`Reference`
`value`	Return value of the function	`Double (F64)`
`E`	Error indicator	`Bool`

MB_DGEMM – Performs C := alpha*op(A)*op(B) + beta*C, where op(X) = X or op(X) = X^T

Block Symbol

Licence: STANDARD



MB_DGEMM

Function Description

The output references yA, yB and yC are always set to the corresponding input references uA, uB and uC. If HLD = on then nothing is computed otherwise the BLAS function DGEMM is called internally:

```
DGEMM(sTRANSA, sTRANSB, M, N, KA, alpha, uA, LDA, uB, LDB, beta, uC, LDC);
```

where parameters of DGEMM are set in the following way:

- Integer inputs transa and transb are mapped to strings sTRANSA and sTRANSB: $\{0,1\} \rightarrow$ "N", $\{2\} \rightarrow$ "T" and $\{3\} \rightarrow$ "C".
- M is number of rows of the matrix referenced by uC.
- N is number of columns of the matrix referenced by uC.
- If the input transa is equal to 0 or 1 then KA is number of columns else KA is number rows of the matrix referenced by uA.
- LDA, LDB and LDC are leading dimensions of matrices referenced by uA, uB and uC.

The error flag E is set to on if:

- the reference uA or uB or uC is not defined (i.e. input uA or uB or uC is not connected),
- transa or transb is less than 0 or greater than 3
- KA $\neq$ KB; if the input transb is equal to 0 or 1 then KB is number of rows else KB is number of columns of the matrix referenced by uB (i.e. matrices $op(A)$ and $op(B)$ have to be multipliable).
- the call of the function DGEMM returns error using the function XERBLA, see the system log.

See BLAS documentation [6] for more details.

Inputs

uA	Input reference to matrix A		Reference
uB	Input reference to matrix B		Reference
uC	Input reference to matrix C		Reference
transa	Transposition of matrix A	↓0 ↑3	Long (I32)
transb	Transposition of matrix B	↓0 ↑3	Long (I32)
alpha	Scalar coefficient alpha		Double (F64)
beta	Scalar coefficient beta		Double (F64)
HLD	Hold		Bool

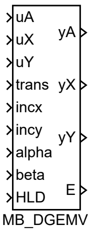
Outputs

yA	Output reference to matrix A	Reference
yB	Output reference to matrix B	Reference
yC	Output reference to matrix C	Reference
E	Error indicator	Bool

MB_DGEMV – Performs y := alpha*A*x + beta*y or y := alpha*A^T*x + beta*y

Block Symbol

Licence: STANDARD



Function Description

The output references yA, yX and yY are always set to the corresponding input references uA, uX and uY. If HLD = on then nothing is computed otherwise the BLAS function DGEMV is called internally:

```
DGEMV(sTRANS, M, N, alpha, uA, LDA, uX, INCX, beta, uY, INCY);
```

where parameters of DGEMV are set in the following way:

- Integer input trans is mapped to the string sTRANS: $\{0, 1\} \rightarrow$ "N", $\{2\} \rightarrow$ "T" and $\{3\} \rightarrow$ "C".
- M is number of rows of the matrix referenced by uA.
- N is number of columns of the matrix referenced by uA.
- LDA is the leading dimension of matrix referenced by uA.
- If the input incx $\neq 0$ then INCX is set to incx else INCX is set to 1.
- If the input incy $\neq 0$ then INCY is set to incy else INCY is set to 1.

The error flag E is set to on if:

- the reference uA or uX or uY is not defined (i.e. input uA or uX or uY is not connected),
- trans is less than 0 or greater than 3
- the call of the function DGEMV returns error using the function XERBLA, see the system log.

See BLAS documentation [6] for more details.

Inputs

`uA`	Input reference to matrix A		`Reference`
`uX`	Input reference to vector x		`Reference`
`uY`	Input reference to vector y		`Reference`
`trans`	Transposition of the input matrix	↓0 ↑3	`Long (I32)`
`incx`	Index increment of vector x		`Long (I32)`
`incy`	Index increment of vector y		`Long (I32)`
`alpha`	Scalar coefficient alpha		`Double (F64)`
`beta`	Scalar coefficient beta		`Double (F64)`
`HLD`	Hold		`Bool`

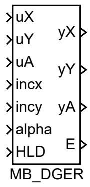
Outputs

`yA`	Output reference to matrix A	`Reference`
`yX`	Output reference to vector x	`Reference`
`yY`	Output reference to vector y	`Reference`
`E`	Error indicator	`Bool`

MB_DGER – Performs A := alpha*x*y^T + A

Block Symbol

Licence: STANDARD



Function Description

The output references yX, yY and yA are always set to the corresponding input references uX, uY and uA. If HLD = on then nothing is computed otherwise the BLAS function DGER is called internally:

```
DGER(M, N, alpha, uX, INCX, uY, INCY, uA, LDA);
```

where parameters of DGER are set in the following way:

- M is number of rows of the matrix referenced by uA.
- N is number of columns of the matrix referenced by uA.
- If the input incx $\neq 0$ then INCX is set to incx else INCX is set to 1.
- If the input incy $\neq 0$ then INCY is set to incy else INCY is set to 1.
- LDA is the leading dimension of matrix referenced by uA.

The error flag E is set to on if:

- the reference uX or uY or uA is not defined (i.e. input uX or uY or uA is not connected),
- the call of the function DGER returns error using the function XERBLA, see the system log.

See BLAS documentation [6] for more details.

Inputs

uX	Input reference to vector x	Reference
uY	Input reference to vector y	Reference
uA	Input reference to matrix A	Reference
incx	Index increment of vector x	Long (I32)
incy	Index increment of vector y	Long (I32)
alpha	Scalar coefficient alpha	Double (F64)
HLD	Hold	Bool

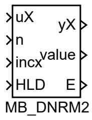
Outputs

`yX`	Output reference to vector x	`Reference`
`yY`	Output reference to vector y	`Reference`
`yA`	Output reference to matrix A	`Reference`
`E`	Error indicator	`Bool`

MB_DNRM2 – Euclidean norm of a vector

Block Symbol

Licence: STANDARD



Function Description

The output reference yX is always set to the input reference uX. If HLD = on then nothing is computed otherwise the BLAS function DNRM2 is called internally:

```
value = DNRM2(N, uX, INCX);
```

where the values N and INCX are set in the following way:

- If the input n > 0 then N is set to n else N is set to the current number of the vector or matrix elements CNT referenced by uX.
- If the input incx > 0 then INCX is set to incx else INCX is set to 1.

The error flag E is set to on if:

- the reference uX is not defined (i.e. input uX is not connected),
- n < 0 or incx < 0,
- $(\mathtt{N} - 1) * |\mathtt{INCX}| + 1 > \mathtt{CNT}$.

See BLAS documentation [6] for more details.

Use the block ML_DLANGE for computation of various norms of a matrix.

Inputs

uX	Input reference to vector x	Reference
n	Number of processed vector elements	Long (I32)
incx	Index increment of vector x	Long (I32)
HLD	Hold	Bool

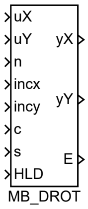
Outputs

yX	Output reference to vector x	Reference
value	Return value of the function	Double (F64)
E	Error indicator	Bool

MB_DROT – Plain rotation of a vector

Block Symbol

Licence: STANDARD



Function Description

The output references yX and yY are always set to the corresponding input references uX and uY. If HLD = on then nothing is computed otherwise the BLAS function DROT is called internally:

```
DROT(N, uX, INCX, uY, INCY, c, s);
```

where parameters of DROT are set in the following way:

- If the input n > 0 then N is set to n else N is set to the current number of the vector or matrix elements CNTX referenced by uX.
- If the input incx $\neq 0$ then INCX is set to incx else INCX is set to 1.
- If the input incy $\neq 0$ then INCY is set to incy else INCY is set to 1.

The error flag E is set to on if:

- the reference uX or uY is not defined (i.e. input uX or uY is not connected),
- n < 0,
- $(\mathtt{N} - 1) * |\mathtt{INCX}| + 1 > \mathtt{CNTX}$,
- $(\mathtt{N} - 1) * |\mathtt{INCY}| + 1 > \mathtt{CNTY}$, where CNTY is a number of the vector or matrix elements referenced by uY.

See BLAS documentation [6] for more details.

Inputs

uX	Input reference to vector x	Reference
uY	Input reference to vector y	Reference
n	Number of processed vector elements	Long (I32)
incx	Index increment of vector x	Long (I32)

`incy`	Index increment of vector y	`Long (I32)`
`c`	Scalar coefficient c	`Double (F64)`
`s`	Scalar coefficient s	`Double (F64)`
`HLD`	Hold	`Bool`

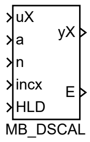
Outputs

`yX`	Output reference to vector x	`Reference`
`yY`	Output reference to vector y	`Reference`
`E`	Error indicator	`Bool`

MB_DSCAL – Scales a vector by a constant

Block Symbol

Licence: STANDARD



Function Description

The output references yX is always set to the corresponding input reference uX. If HLD = on then nothing is computed otherwise the BLAS function DSCAL is called internally:

```
DSCAL(N, a, uX, INCX);
```

where parameters of DSCAL are set in the following way:

- If the input n > 0 then N is set to n else N is set to the current number of the vector or matrix elements CNT referenced by uX.
- If the input incx $\neq 0$ then INCX is set to incx else INCX is set to 1.

The error flag E is set to on if:

- the reference uX is not defined (i.e. input uX is not connected),
- n < 0 or incx < 0,
- $(\mathtt{N} - 1) * \mathtt{INCX} + 1 > \mathtt{CNT}$.

See BLAS documentation [6] for more details.

Inputs

uX	Input reference to vector x	Reference
a	Scalar coefficient a	Double (F64)
n	Number of processed vector elements	Long (I32)
incx	Index increment of vector x	Long (I32)
HLD	Hold	Bool

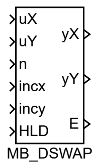
Outputs

yX	Output reference to vector x	Reference
E	Error indicator	Bool

MB_DSWAP – Interchanges two vectors

Block Symbol

Licence: STANDARD



Function Description

The output references yX and yY are always set to the corresponding input references uX and uY. If HLD = on then nothing is computed otherwise the BLAS function DSWAP is called internally:

```
DSWAP(N, uX, INCX, uY, INCY);
```

where the values N, INCX and INCY are set in the following way:

- If the input n > 0 then N is set to n else N is set to the current number of the vector or matrix elements CNTX referenced by uX.
- If the input incx $\neq 0$ then INCX is set to incx else INCX is set to 1.
- If the input incy $\neq 0$ then INCY is set to incy else INCY is set to 1.

The error flag E is set to on if:

- the reference uX or uY is not defined (i.e. input uX or uY is not connected),
- n < 0,
- $(\mathtt{N} - 1) * |\mathtt{INCX}| + 1 > \mathtt{CNTX}$,
- $(\mathtt{N} - 1) * |\mathtt{INCY}| + 1 > \mathtt{CNTY}$, where CNTY is a number of the vector or matrix elements referenced by uY.

See BLAS documentation [6] for more details.

Inputs

uX	Input reference to vector x	Reference
uY	Input reference to vector y	Reference
n	Number of processed vector elements	Long (I32)
incx	Index increment of vector x	Long (I32)
incy	Index increment of vector y	Long (I32)
HLD	Hold	Bool

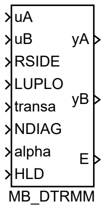
Outputs

yX	Output reference to vector x	Reference
yY	Output reference to vector y	Reference
E	Error indicator	Bool

MB_DTRMM – Performs B := alpha*op(A)*B or B := alpha*B*op(A), where op(X) = X or op(X) = X^T for triangular matrix A

Block Symbol

Licence: STANDARD



Function Description

The output references `yA` and `yB` are always set to the corresponding input references `uA` and `uB`. If `HLD = on` then nothing is computed otherwise the BLAS function `DTRMM` is called internally:

```
DTRMM(sRSIDE, sLUPLO, sTRANSA, sNDIAG, M, N, alpha, uA, LDA, uB, LDB);
```

where parameters of `DTRMM` are set in the following way:

- If `RSIDE` = `on` then the string `sRSIDE` is set to `"R"` else it is set to `"L"`.
- If `LUPLO` = `on` then the string `sLUPLO` is set to `"L"` else it is set to `"U"`.
- Integer input `transa` is mapped to the string `sTRANSA`: $\{0,1\} \rightarrow$ `"N"`, $\{2\} \rightarrow$ `"T"` and $\{3\} \rightarrow$ `"C"`.
- If `NDIAG` = `on` then the string `sNDIAG` is set to `"N"` else it is set to `"U"`.
- `M` is number of rows of the matrix referenced by `uB`.
- `N` is number of columns of the matrix referenced by `uB`.
- `LDA` and `LDB` are leading dimensions of matrices referenced by `uA` and `uB`.

The error flag `E` is set to `on` if:

- the reference `uA` or `uB` is not defined (i.e. input `uA` or `uB` is not connected),
- `transa` is less than 0 or greater than 3,
- matrix referenced by `uA` is not square or is not compatible with the matrix referenced by `uB`,
- the call of the function `DTRMM` returns error using the function `XERBLA`, see the system log.

See BLAS documentation [6] for more details.

Inputs

uA	Input reference to matrix A		Reference
uB	Input reference to matrix B		Reference
RSIDE	Operation is applied from right side		Bool
LUPLO	Matrix A is a lower triangular matrix		Bool
transa	Transposition of matrix A	↓0 ↑3	Long (I32)
NDIAG	Matrix A is not assumed to be unit triangular		Bool
alpha	Scalar coefficient alpha		Double (F64)
HLD	Hold		Bool

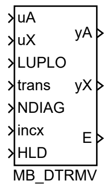
Outputs

yA	Output reference to matrix A	Reference
yB	Output reference to matrix B	Reference
E	Error indicator	Bool

MB_DTRMV – Performs x := A*x or x := A^T*x for triangular matrix A

Block Symbol

Licence: STANDARD



Function Description

The output references yA and yX are always set to the corresponding input references uA and uX. If HLD = on then nothing is computed otherwise the BLAS function DTRMV is called internally:

```
DTRMV(sLUPLO, sTRANS, sNDIAG, N, uA, LDA, uX, INCX);
```

where parameters of DTRMV are set in the following way:

- If LUPLO = on then the string sLUPLO is set to "L" else it is set to "U".
- Integer input trans is mapped to the string sTRANS: $\{0, 1\} \rightarrow$ "N", $\{2\} \rightarrow$ "T" and $\{3\} \rightarrow$ "C".
- If NDIAG = on then the string sNDIAG is set to "N" else it is set to "U".
- N is number of rows and columns of the square matrix referenced by uA.
- LDA is the leading dimension of matrix referenced by uA.
- If the input incx $\neq 0$ then INCX is set to incx else INCX is set to 1.

The error flag E is set to on if:

- the reference uA or uX is not defined (i.e. input uA or uX is not connected),
- trans is less than 0 or greater than 3,
- matrix referenced by uA is not square,
- $(\mathtt{N} - 1) * |\mathtt{INCX}| + 1 > \mathtt{CNTX}$, where CNTX is a number of the vector or matrix elements referenced by uX.
- the call of the function DTRMV returns error using the function XERBLA, see the system log.

See BLAS documentation [6] for more details.

Inputs

uA	Input reference to matrix A		Reference
uX	Input reference to vector x		Reference
LUPLO	Matrix A is a lower triangular matrix		Bool
trans	Transposition of the input matrix	↓0 ↑3	Long (I32)
NDIAG	Matrix A is not assumed to be unit triangular		Bool
incx	Index increment of vector x		Long (I32)
HLD	Hold		Bool

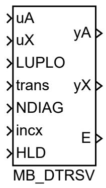
Outputs

yA	Output reference to matrix A	Reference
yX	Output reference to vector x	Reference
E	Error indicator	Bool

MB_DTRSV – Solves one of the system of equations A*x = b or A^T*x = b for triangular matrix A

Block Symbol

Licence: STANDARD



Function Description

The output references yA and yX are always set to the corresponding input references uA and uX. If HLD = on then nothing is computed otherwise the BLAS function DTRSV is called internally:

```
DTRSV(sLUPLO, sTRANS, sNDIAG, N, uA, LDA, uX, INCX);
```

where parameters of DTRSV are set in the following way:

- If LUPLO = on then the string sLUPLO is set to "L" else it is set to "U".
- Integer input trans is mapped to the string sTRANS: $\{0, 1\} \rightarrow$ "N", $\{2\} \rightarrow$ "T" and $\{3\} \rightarrow$ "C".
- If NDIAG = on then the string sNDIAG is set to "N" else it is set to "U".
- N is number of rows and columns of the square matrix referenced by uA.
- LDA is the leading dimension of matrix referenced by uA.
- If the input incx $\neq 0$ then INCX is set to incx else INCX is set to 1.

The error flag E is set to on if:

- the reference uA or uX is not defined (i.e. input uA or uX is not connected),
- trans is less than 0 or greater than 3,
- matrix referenced by uA is not square,
- $(\mathtt{N} - 1) * |\mathtt{INCX}| + 1 > \mathtt{CNTX}$, where CNTX is a number of the vector or matrix elements referenced by uX.
- the call of the function DTRMV returns error using the function XERBLA, see the system log.

See BLAS documentation [6] for more details.

Inputs

`uA`	Input reference to matrix A		`Reference`
`uX`	Input reference to vector x		`Reference`
`LUPLO`	Matrix A is a lower triangular matrix		`Bool`
`trans`	Transposition of the input matrix	↓0 ↑3	`Long (I32)`
`NDIAG`	Matrix A is not assumed to be unit triangular		`Bool`
`incx`	Index increment of vector x		`Long (I32)`
`HLD`	Hold		`Bool`

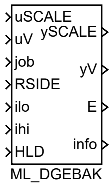
Outputs

`yA`	Output reference to matrix A	`Reference`
`yX`	Output reference to vector x	`Reference`
`E`	Error indicator	`Bool`

ML_DGEBAK – Backward transformation to ML_DGEBAL of left or right eigenvectors

Block Symbol

Licence: MATRIX



Function Description

The output references ySCALE and yV are always set to the corresponding input references uSCALE and uV. If HLD = on then nothing is computed otherwise the LAPACK function DGEBAK is called internally:

```
DGEBAK(sJOB, sRSIDE, N, ilo, IHI, uSCALE, M, uV, LDV, info);
```

where parameters of DGEBAK are set in the following way:

- Integer input job is mapped to the string sJOB: $\{0,1\} \to$ "N", $\{2\} \to$ "P", $\{3\} \to$ "S" and $\{4\} \to$ "B".
- If RSIDE = on then the string sRSIDE is set to "R" else it is set to "L".
- N is number of elements of the vector referenced by uSCALE.
- If the input ihi $\neq 0$ then IHI is set to ihi else IHI is set to N $- 1$.
- M is number of columns of the matrix referenced by uV.
- LDV is the leading dimension of the matrix referenced by uV.
- info is return code from the function DGEBAK.

The error flag E is set to on if:

- the reference uSCALE or uV is not defined (i.e. input uSCALE or uV is not connected),
- the call of the function DGEBAK returns error using the function XERBLA, see the return code info and system log.

Emphasize that the indices ilo and ihi start from zero unlike FORTRAN version where they start from one. See LAPACK documentation [7] for more details.

Inputs

uSCALE	Input reference to vector SCALE	Reference
uV	Reference to matrix of right or left eigenvectors to be transformed	Reference
job	Type of backward transformation required ↓0 ↑4	Long (I32)
RSIDE	Operation is applied from right side	Bool
ilo	Zero based low row and column index of working submatrix	Long (I32)
ihi	Zero based high row and column index of working submatrix	Long (I32)
HLD	Hold	Bool

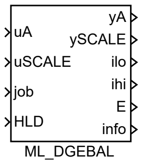
Outputs

ySCALE	Output reference to vector SCALE	Reference
yV	Reference to matrix of transformed right or left eigenvectors	Reference
E	Error indicator	Bool
info	LAPACK function result info. If info = -i, the i=th argument had an illegal value	Long (I32)

ML_DGEBAL – Balancing of a general real matrix

Block Symbol

Licence: MATRIX



Function Description

The output references `yA` and `ySCALE` are always set to the corresponding input references `uA` and `uSCALE`. If `HLD = on` then nothing is computed otherwise the LAPACK function `DGEBAL` is called internally:

```
DGEBAL(sJOB, N, uA, LDA, ilo, ihi, uSCALE, info);
```

where parameters of `DGEBAL` are set in the following way:

- Integer input `job` is mapped to the string `sJOB`: $\{0, 1\} \rightarrow$ "N", $\{2\} \rightarrow$ "P", $\{3\} \rightarrow$ "S" and $\{4\} \rightarrow$ "B".
- `N` is number of columns of the square matrix referenced by `uA`.
- `LDA` is the leading dimension of the matrix referenced by `uA`.
- `ilo` and `ihi` are returned low and high row and column indices of the balanced submatrix of the matrix referenced by `uA`.
- `info` is return code from the function `DGEBAL`.

The error flag `E` is set to `on` if:

- the reference `uA` or `uSCALE` is not defined (i.e. input `uA` or `uSCALE` is not connected),
- matrix referenced by `uA` is not square,
- number of elements of the vector referenced by `uSCALE` is less than `N`.
- the call of the function `DGEBAL` returns error using the function `XERBLA`, see the return code `info` and system log.

Emphasize that the indices `ilo` and `ihi` start from zero unlike FORTRAN version where they start from one. See LAPACK documentation [7] for more details.

Inputs

`uA`	Input reference to matrix A		`Reference`
`uSCALE`	Input reference to vector SCALE		`Reference`
`job`	Specifies the operations to be performed on matrix A	↓0 ↑4	`Long (I32)`
`HLD`	Hold		`Bool`

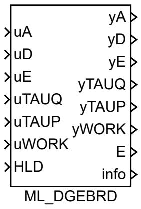
Outputs

`yA`	Output reference to matrix A	`Reference`
`ySCALE`	Output reference to vector SCALE	`Reference`
`ilo`	Zero based low row and column index of working submatrix	`Long (I32)`
`ihi`	Zero based high row and column index of working submatrix	`Long (I32)`
`E`	Error indicator	`Bool`
`info`	LAPACK function result info. If info = -i, the i=th argument had an illegal value	`Long (I32)`

ML_DGEBRD – Reduces a general real matrix to bidiagonal form by an orthogonal transformation

Block Symbol

Licence: MATRIX



Function Description

The output references yA, yD, yE, yTAUQ, yTAUP and yWORK are always set to the corresponding input references uA, uD, uE, uTAUQ, uTAUP and uWORK. If HLD = on then nothing is computed otherwise the LAPACK function DGEBRD is called internally:

```
DGEBRD(M, N, uA, LDA, uD, uE, uTAUQ, uTAUP, uWORK, info);
```

where parameters of DGEBRD are set in the following way:

- M is number of rows of the matrix referenced by uA.
- N is number of columns of the matrix referenced by uA.
- LDA is the leading dimension of the matrix referenced by uA.
- info is return code from the function DGEBRD.

The error flag E is set to on if:

- the reference uA or uD or uE or uTAUQ or uTAUP or uWORK is not defined (i.e. input uA or uD or uE or uTAUQ or uTAUP or uWORK is not connected),
- number of elements of any vector referenced by uD, uTAUQ and uTAUP is less than MINMN, where MINMN is minimum from M and N,
- number of elements of the vector referenced by uE is less than MINMN $-$ 1,
- the call of the function DGEBRD returns error using the function XERBLA, see the return code info and system log.

See LAPACK documentation [7] for more details.

Inputs

uA	Input reference to matrix A	Reference
uD	Diagonal elements of the bidiagonal matrix B	Reference
uE	Off-diagonal elements of the bidiagonal matrix B	Reference
uTAUQ	Reference to a vector of scalar factors of the elementary reflectors which represent the orthogonal matrix Q	Reference
uTAUP	Reference to a vector of scalar factors of the elementary reflectors which represent the orthogonal matrix P	Reference
uWORK	Input reference to working vector WORK	Reference
HLD	Hold	Bool

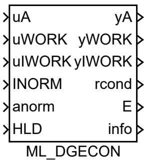
Outputs

yA	Output reference to matrix A	Reference
yD	Output reference to D	Reference
yE	Output reference to E	Reference
yTAUQ	Output reference to TAUQ	Reference
yTAUP	Output reference to TAUP	Reference
yWORK	Output reference to working vector WORK	Reference
E	Error indicator	Bool
info	LAPACK function result info. If info = -i, the i=th argument had an illegal value	Long (I32)

ML_DGECON – Estimates the reciprocal of the condition number of a general real matrix

Block Symbol

Licence: MATRIX



Function Description

The output references yA, yWORK and yIWORK are always set to the corresponding input references uA, uWORK and uIWORK. If HLD = on then nothing is computed otherwise the LAPACK function DGECON is called internally:

```
DGECON(sINORM, N, uA, LDA, anorm, rcond, uWORK, uIWORK, info);
```

where parameters of DGECON are set in the following way:

- If INORM = on then the string sINORM is set to "I" else it is set to "1".
- N is number of columns of the matrix referenced by uA.
- LDA is the leading dimension of the matrix referenced by uA.
- rcond is returned reciprocal value of the condition number of the matrix referenced by uA.
- info is return code from the function DGECON.

The error flag E is set to on if:

- the reference uA or uWORK or uIWORK is not defined (i.e. input uA or uWORK or uIWORK is not connected),
- the matrix referenced by uA is not square,
- number of elements of the vector referenced by uWORK is less than $4 * N$,
- number of elements of the integer vector referenced by uIWORK is less than N,
- the call of the function DGECON returns error using the function XERBLA, see the return code info and system log.

See LAPACK documentation [7] for more details.

Inputs

uA	Input reference to matrix A	Reference
uWORK	Input reference to working vector WORK	Reference
uIWORK	Input reference to integer working vector WORK	Reference
INORM	Use Infinity-norm	Bool
anorm	Norm of the original matrix A	Double (F64)
HLD	Hold	Bool

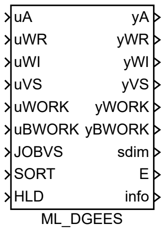
Outputs

yA	Output reference to matrix A	Reference
yWORK	Output reference to working vector WORK	Reference
yIWORK	Output reference to integer working vector WORK	Reference
rcond	The reciprocal of the condition number of the matrix A	Double (F64)
E	Error indicator	Bool
info	LAPACK function result info. If info = -i, the i=th argument had an illegal value	Long (I32)

ML_DGEES – Computes the eigenvalues, the Schur form, and, optionally, the matrix of Schur vectors

Block Symbol

Licence: MATRIX



Function Description

The output references yA, yWR, yWI, yVS, yWORK and yBWORK are always set to the corresponding input references uA, uWR, uWI, uVS, uWORK and uBWORK. If HLD = on then nothing is computed otherwise the LAPACK function DGEES is called internally:

```
DGEES(sJOBVS, sSORT, SELECT, N, uA, LDA, sdim, uWR, uWI, uVS, LDVS,uWORK,
    LWORK, uBWORK, info);
```

where parameters of DGEES are set in the following way:

- If JOBVS = on then the string sJOBVS is set to "V" else it is set to "N".
- If SORT = on then the string sSORT is set to "S" else it is set to "N".
- SELECT is the reference to Boolean eigenvalues sorting function which in this function block returns always true (i.e. on).
- N is number of columns of the matrix referenced by uA.
- LDA is the leading dimension of the matrix referenced by uA.
- sdim is returned number of eigenvalues for which the function SELECT is true.
- LDVS is the leading dimension of the matrix referenced by uVS.
- LWORK is number of elements in the vector referenced by uWORK.
- info is return code from the function DGEES.

The error flag E is set to on if:

- the reference uA or uWR or uWI or uVS or uWORK or uBWORK is not defined (i.e. input uA or uWR or uWI or uVS or uWORK or uBWORK is not connected),

- the matrix referenced by uA is not square,
- number of elements of any vector referenced by uWR, uWI and uBWORK is less than N,
- number of columns of the matrix referenced by uVS is not equal to N,
- the call of the function DGEES returns error using the function XERBLA, see the return code info and system log.

See LAPACK documentation [7] for more details.

Inputs

uA	Input reference to matrix A	Reference
uWR	Input reference to vector of real parts of eigenvalues	Reference
uWI	Input reference to vector of imaginary parts of eigenvalues	Reference
uVS	Input reference to orthogonal matrix of Schur vectors	Reference
uWORK	Input reference to working vector WORK	Reference
uBWORK	Input reference to Boolean working vector WORK	Reference
JOBVS	If true then Schur vectors are computed	Bool
SORT	If true then eigenvalues are sorted	Bool
HLD	Hold	Bool

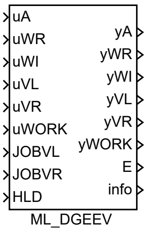
Outputs

yA	Output reference to matrix A	Reference
yWR	Output reference to vector of real parts of eigenvalues	Reference
yWI	Output reference to vector of imaginary parts of eigenvalues	Reference
yVS	Output reference to VS	Reference
yWORK	Output reference to working vector WORK	Reference
yBWORK	Output reference to Boolean working vector WORK	Reference
sdim	If SORT then number of eigenvalues for which SELECT is true else 0	Long (I32)
E	Error indicator	Bool
info	LAPACK function result info. If info = -i, the i=th argument had an illegal value	Long (I32)

ML_DGEEV – Computes the eigenvalues and, optionally, the left and/or right eigenvectors

Block Symbol

Licence: MATRIX



Function Description

The output references `yA`, `yWR`, `yWI`, `yVL`, `yVR` and `yWORK` are always set to the corresponding input references `uA`, `uWR`, `uWI`, `uVL`, `uVR` and `uWORK`. If `HLD = on` then nothing is computed otherwise the LAPACK function `DGEEV` is called internally:

```
DGEEV(sJOBVL, sJOBVR, N, uA, LDA, uWR, uWI, uVL, LDVL, uVR, LDVR,
    uWORK, LWORK, info);
```

where parameters of `DGEEV` are set in the following way:

- If `JOBVL` = `on` then the string `sJOBVL` is set to `"V"` else it is set to `"N"`.
- If `JOBVR` = `on` then the string `sJOBVR` is set to `"V"` else it is set to `"N"`.
- `N` is number of columns of the matrix referenced by `uA`.
- `LDA`, `LDVL` and `LDVR` are leading dimensions of the matrices referenced by `uA`, `uVL` and `uVR`.
- `LWORK` is number of elements of the vector referenced by `uWORK`.
- `info` is return code from the function `DGEEV`.

The error flag `E` is set to `on` if:

- the reference `uA` or `uWR` or `uWI` or `uVL` or `uVR` or `uWORK` is not defined (i.e. input `uA` or `uWR` or `uWI` or `uVL` or `uVR` or `uWORK` is not connected),
- the matrix referenced by `uA` is not square,
- number of elements of vectors referenced by `uWR` or `uWI` is less than `N`,
- number of columns of matrices referenced by `uVL` or `uVR` is not equal to `N`,

- the call of the function DGEEV returns error using the function XERBLA, see the return code info and system log.

See LAPACK documentation [7] for more details.

Inputs

uA	Input reference to matrix A	Reference
uWR	Input reference to vector of real parts of eigenvalues	Reference
uWI	Input reference to vector of imaginary parts of eigenvalues	Reference
uVL	Input reference to matrix of left eigenvectors	Reference
uVR	Input reference to matrix of right eigenvectors	Reference
uWORK	Input reference to working vector WORK	Reference
JOBVL	If true then left eigenvectors are computed	Bool
JOBVR	If true then right eigenvectors are computed	Bool
HLD	Hold	Bool

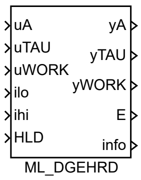
Outputs

yA	Output reference to matrix A	Reference
yWR	Output reference to vector of real parts of eigenvalues	Reference
yWI	Output reference to vector of imaginary parts of eigenvalues	Reference
yVL	Output reference to VL	Reference
yVR	Output reference to VR	Reference
yWORK	Output reference to working vector WORK	Reference
E	Error indicator	Bool
info	LAPACK function result info. If info = -i, the i=th argument had an illegal value	Long (I32)

ML_DGEHRD – Reduces a real general matrix A to upper Hessenberg form

Block Symbol

Licence: MATRIX



Function Description

The output references yA, yTAU and yWORK are always set to the corresponding input references uA, uTAU and uWORK. If HLD = on then nothing is computed otherwise the LAPACK function DGEHRD is called internally:

```
DGEHRD(N, ilo, IHI, uA, LDA, uTAU, uWORK, LWORK, info);
```

where parameters of DGEHRD are set in the following way:

- N is number of columns of the square matrix referenced by uA.
- If the input ihi $\neq 0$ then IHI is set to ihi else IHI is set to N $- 1$.
- LDA is the leading dimension of the matrix referenced by uA.
- LWORK is number of elements of the vector referenced by uWORK.
- info is return code from the function DGEHRD.

The error flag E is set to on if:

- the reference uA or uTAU or uWORK is not defined (i.e. input uA or uTAU or uWORK is not connected),
- matrix referenced by uA is not square,
- number of elements of the vector referenced by uTAU is less than N $- 1$.
- the call of the function DGEHRD returns error using the function XERBLA, see the return code info and system log.

Emphasize that the indices ilo and ihi start from zero unlike FORTRAN version where they start from one. See LAPACK documentation [7] for more details.

Inputs

uA	Input reference to matrix A	Reference
uTAU	Input reference to vector of scalar factors of the elementary reflectors	Reference
uWORK	Input reference to working vector WORK	Reference
ilo	Zero based low row and column index of working submatrix	Long (I32)
ihi	Zero based high row and column index of working submatrix	Long (I32)
HLD	Hold	Bool

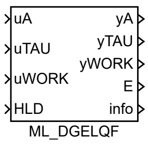
Outputs

yA	Output reference to matrix A	Reference
yTAU	Output reference to vector of scalar factors of the elementary reflectors	Reference
yWORK	Output reference to working vector WORK	Reference
E	Error indicator	Bool
info	LAPACK function result info. If info = -i, the i=th argument had an illegal value	Long (I32)

ML_DGELQF – Computes an LQ factorization of a real M-by-N matrix A

Block Symbol

Licence: MATRIX



Function Description

The output references yA, yTAU and yWORK are always set to the corresponding input references uA, uTAU and uWORK. If HLD = on then nothing is computed otherwise the LAPACK function DGELQF is called internally:

```
DGELQF(M, N, uA, LDA, uTAU, uWORK, LWORK, info);
```

where parameters of DGELQF are set in the following way:

- M is number of rows of the matrix referenced by uA.
- N is number of columns of the matrix referenced by uA.
- LDA is the leading dimension of the matrix referenced by uA.
- LWORK is number of elements of the vector referenced by uWORK.
- info is return code from the function DGELQF.

The error flag E is set to on if:

- the reference uA or uTAU or uWORK is not defined (i.e. input uA or uTAU or uWORK is not connected),
- number of elements of the vector referenced by uTAU is less than the minimum of number of rows and number of columns of the matrix referenced by uA.
- the call of the function DGELQF returns error using the function XERBLA, see the return code info and system log.

See LAPACK documentation [7] for more details.

Inputs

uA	Input reference to matrix A	Reference

uTAU	Input reference to vector of scalar factors of the elementary reflectors	Reference
uWORK	Input reference to working vector WORK	Reference
HLD	Hold	Bool

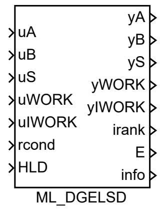
Outputs

yA	Output reference to matrix A	Reference
yTAU	Output reference to vector of scalar factors of the elementary reflectors	Reference
yWORK	Output reference to working vector WORK	Reference
E	Error indicator	Bool
info	LAPACK function result info. If info = -i, the i=th argument had an illegal value	Long (I32)

ML_DGELSD – Computes the minimum-norm solution to a real linear least squares problem

Block Symbol

Licence: MATRIX



Function Description

The output references yA, yB, yS, yWORK and yIWORK are always set to the corresponding input references uA, uB, uS, uWORK and uIWORK. If HLD = on then nothing is computed otherwise the LAPACK function DGELSD is called internally:

```
DGELSD(M, N, NRHS, uA, LDA, uB, LDB, uS, rcond, irank,uWORK,
     LWORK, uIWORK, info);
```

where parameters of DGELSD are set in the following way:

- M is number of rows of the matrix referenced by uA.
- N is number of columns of the matrix referenced by uA.
- NRHS is number of columns of the matrix referenced by uB.
- LDA and LDB are leading dimensions of the matrices referenced by uA and uB.
- irank is returned effective rank of the matrix referenced by uA.
- LWORK is number of elements in the vector referenced by uWORK.
- info is return code from the function DGELSD.

The error flag E is set to on if:

- the reference uA or uB or uS or uWORK or uIWORK is not defined (i.e. input uA or uB or uS or uWORK or uIWORK is not connected),
- the number of rows of the matrix referenced by uB is not equal to M,
- number of elements of any vector referenced by uS is less than the minimum of M and N,

- number of elements of the integer vector referenced by `uIWORK` is not sufficient (see details in the LAPACK documentation of the function `DGELSD`),
- the call of the function `DGELSD` returns error using the function `XERBLA`, see the return code `info` and system log.

See LAPACK documentation [7] for more details.

Inputs

`uA`	Input reference to matrix A	`Reference`
`uB`	Input reference to matrix B	`Reference`
`uS`	Input reference to vector of singular values	`Reference`
`uWORK`	Input reference to working vector WORK	`Reference`
`uIWORK`	Input reference to integer working vector WORK	`Reference`
`rcond`	Used to determine the effective rank of A	`Double (F64)`
`HLD`	Hold	`Bool`

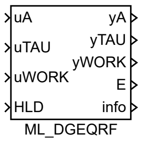
Outputs

`yA`	Output reference to matrix A	`Reference`
`yB`	Output reference to matrix B	`Reference`
`yS`	Output reference to vector of singular values	`Reference`
`yWORK`	Output reference to working vector WORK	`Reference`
`yIWORK`	Output reference to integer working vector WORK	`Reference`
`irank`	Effective rank of A	`Long (I32)`
`E`	Error indicator	`Bool`
`info`	LAPACK function result info. If info = -i, the i=th argument had an illegal value	`Long (I32)`

ML_DGEQRF – Computes an QR factorization of a real M-by-N matrix A

Block Symbol

Licence: MATRIX



Function Description

The output references yA, yTAU and yWORK are always set to the corresponding input references uA, uTAU and uWORK. If HLD = on then nothing is computed otherwise the LAPACK function DGEQRF is called internally:

```
DGEQRF(M, N, uA, LDA, uTAU, uWORK, LWORK, info);
```

where parameters of DGEQRF are set in the following way:

- M is number of rows of the matrix referenced by uA.
- N is number of columns of the matrix referenced by uA.
- LDA is the leading dimension of the matrix referenced by uA.
- LWORK is number of elements of the vector referenced by uWORK.
- info is return code from the function DGEQRF.

The error flag E is set to on if:

- the reference uA or uTAU or uWORK is not defined (i.e. input uA or uTAU or uWORK is not connected),
- number of elements of the vector referenced by uTAU is less than the minimum of number of rows and number of columns of the matrix referenced by uA.
- the call of the function DGEQRF returns error using the function XERBLA, see the return code info and system log.

See LAPACK documentation [7] for more details.

Inputs

uA	Input reference to matrix A	Reference

uTAU	Input reference to vector of scalar factors of the elementary reflectors	Reference
uWORK	Input reference to working vector WORK	Reference
HLD	Hold	Bool

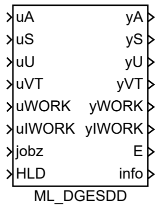
Outputs

yA	Output reference to matrix A	Reference
yTAU	Output reference to vector of scalar factors of the elementary reflectors	Reference
yWORK	Output reference to working vector WORK	Reference
E	Error indicator	Bool
info	LAPACK function result info. If info = -i, the i=th argument had an illegal value	Long (I32)

ML_DGESDD – Computes the singular value decomposition (SVD) of a real M-by-N matrix A

Block Symbol

Licence: MATRIX



Function Description

The output references yA, yS, yU, yVT, yWORK and yIWORK are always set to the corresponding input references uA, uS, uU, uVT, uWORK and uIWORK. If HLD = on then nothing is computed otherwise the LAPACK function DGESDD is called internally:

```
DGESDD(sJOBZ, M, N, uA, LDA, uS, uU, LDU, uVT, LDVT, uWORK, LWORK,
    uIWORK, info);
```

where parameters of DGESDD are set in the following way:

- Integer input jobz is mapped to the string sJOBZ: $\{0, 1\} \to$ "A", $\{2\} \to$ "S", $\{3\} \to$ "O" and $\{4\} \to$ "N".
- M is number of rows of the matrix referenced by uA.
- N is number of columns of the matrix referenced by uA.
- LDA, LDU and LDVT are leading dimensions of the matrices referenced by uA, uU and uVT.
- LWORK is number of elements of the vector referenced by uWORK.
- info is return code from the function DGESDD.

The error flag E is set to on if:

- the reference uA or uS or uU or uVT or uWORK or uIWORK is not defined (i.e. input uA or uS or uU or uVT or uWORK or uIWORK is not connected),
- number of elements of the vector referenced by uS is less than MINMN, the minimum of number of rows and number of columns of the matrix referenced by uA,
- number of elements of the integer vector referenced by uIWORK is less than 8∗MINMN,

- the call of the function `DGESDD` returns error using the function `XERBLA`, see the return code `info` and system log.

See LAPACK documentation [7] for more details.

Inputs

`uA`	Input reference to matrix A	`Reference`
`uS`	Input reference to vector of singular values	`Reference`
`uU`	Input reference to matrix containing left singular vectors of A	`Reference`
`uVT`	Input reference to matrix containing right singular vectors of A	`Reference`
`uWORK`	Input reference to working vector WORK	`Reference`
`uIWORK`	Input reference to integer working vector WORK	`Reference`
`jobz`	Specifies options for computing	`Long (I32)`
`HLD`	Hold	`Bool`

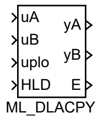
Outputs

`yA`	Output reference to matrix A	`Reference`
`yS`	Output reference to vector of singular values	`Reference`
`yU`	Output reference to matrix containing left singular vectors of A	`Reference`
`yVT`	Output reference to matrix containing right singular vectors of A	`Reference`
`yWORK`	Output reference to working vector WORK	`Reference`
`yIWORK`	Output reference to integer working vector WORK	`Reference`
`E`	Error indicator	`Bool`
`info`	LAPACK function result info. If info = -i, the i=th argument had an illegal value	`Long (I32)`

ML_DLACPY – Copies all or part of one matrix to another matrix

Block Symbol

Licence: STANDARD



Function Description

The output references yA and yB are always set to the corresponding input references uA and uB. If HLD = on then nothing is computed otherwise the LAPACK function DLACPY is called internally:

```
DLACPY(sUPLO, M, N, uA, LDA, uB, LDA);
```

where parameters of DLACPY are set in the following way:

- Integer input uplo is mapped to the string sUPLO: $\{0,1\} \rightarrow$ "A", $\{2\} \rightarrow$ "U" and $\{3\} \rightarrow$ "L".
- M is number of rows of the matrix referenced by uA.
- N is number of columns of the matrix referenced by uA.
- LDA is the leading dimension of the matrix referenced by uA.

The number of rows of the matrix referenced by uB is set to M and the leading dimension of the matrix referenced by uB is set to LDA
The error flag E is set to on if:

- the reference uA or uB is not defined (i.e. input uA or uB is not connected),
- the allocated number of elements of the matrix referenced by uA is different from the allocated number of elements of the matrix referenced by uB.

See LAPACK documentation [7] for more details.

Inputs

uA	Input reference to matrix A	Reference
uB	Input reference to matrix B	Reference
uplo	Part of the matrix to be copied 0 ..... All 1 ..... All 2 ..... Upper 3 ..... Lower	Long (I32)
HLD	Hold	Bool

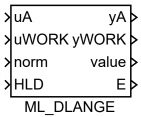
Outputs

yA	Output reference to matrix A	Reference
yB	Output reference to matrix B	Reference
E	Error indicator	Bool

ML_DLANGE – Computes one of the matrix norms of a general matrix

Block Symbol

Licence: STANDARD



Function Description

The output references yA and yWORK are always set to the corresponding input references uA and uWORK. If HLD = on then nothing is computed otherwise the LAPACK function DLANGE is called internally:

```
value = DLANGE(sNORM, M, N, uA, LDA, uWORK;
```

where parameters of DLACPY are set in the following way:

- Integer input norm is mapped to the string sNORM: $\{0, 1\} \rightarrow$ "F" (Frobenius norm), $\{2\} \rightarrow$ "M" (max(abs(A(i,j)))), $\{3\} \rightarrow$ "1" (one norm) and $\{4\} \rightarrow$ "I" (infinity norm).
- M is number of rows of the matrix referenced by uA.
- N is number of columns of the matrix referenced by uA.
- LDA is the leading dimension of the matrix referenced by uA.
- uWORK is the working vector of dimension LWORK $\geq$ M. uWORK is used only for infinity norm, otherwise it is not referenced.

The error flag E is set to on if:

- the reference uA is not defined (i.e. input uA is not connected),
- the reference uWORK is not defined for norm $= 4$ (i.e. input uWORK is not connected).

See LAPACK documentation [7] for more details.

Use the block MB_DNRM2 for computation of Frobenius norm of a vector.

Inputs

uA	Input reference to matrix A		Reference
uWORK	Input reference to working vector WORK		Reference
norm	The selected matrix norm	↓0 ↑4	Long (I32)
HLD	Hold		Bool

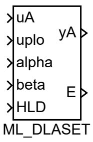
Outputs

yA	Output reference to matrix A	Reference
yWORK	Output reference to working vector WORK	Reference
value	Return value of the function	Double (F64)
E	Error indicator	Bool

ML_DLASET – Initilizes the off-diagonal elements and the diagonal elements of a matrix to given values

Block Symbol

Licence: STANDARD



Function Description

The output reference yA is always set to the corresponding input references uA. If HLD = on then nothing is computed otherwise the LAPACK function DLASET is called internally:

```
DLASET(sUPLO, M, N, alpha, beta,uA, LDA);
```

where parameters of DLACPY are set in the following way:

- Integer input uplo is mapped to the string sUPLO: $\{0, 1\} \rightarrow$ "A", $\{2\} \rightarrow$ "U" and $\{3\} \rightarrow$ "L".
- M is number of rows of the matrix referenced by uA.
- N is number of columns of the matrix referenced by uA.
- LDA is the leading dimension of the matrix referenced by uA.

The error flag E is set to on if:

- the reference uA is not defined (i.e. input uA is not connected),

See LAPACK documentation [7] for more details.

Inputs

uA	Input reference to matrix A	Reference
uplo	Part of the matrix to be set 0 ..... All 1 ..... All 2 ..... Upper 3 ..... Lower	Long (I32)
alpha	Scalar coefficient alpha	Double (F64)
beta	Scalar coefficient beta	Double (F64)
HLD	Hold	Bool

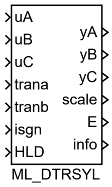
Outputs

yA	Output reference to matrix A	Reference
E	Error indicator	Bool

ML_DTRSYL – Solves the real Sylvester matrix equation for quasi-triangular matrices A and B

Block Symbol

Licence: MATRIX



Function Description

The output references yA, yB and yC are always set to the corresponding input references uA, uB and uC. If HLD = on then nothing is computed otherwise the LAPACK function DTRSYL is called internally:

```
DTRSYL(sTRANA, sTRANB, M, N, uA, LDA, uB, LDB, uC, LDC, scale, info);
```

where parameters of DTRSYL are set in the following way:

- Integer inputs trana and tranb are mapped to strings sTRANA and sTRANB: $\{0, 1\} \to$ "N", $\{2\} \to$ "T" and $\{3\} \to$ "C".
- M is number of rows of the matrix referenced by uA.
- N is number of columns of the matrix referenced by uB.
- LDA, LDB and LDC are leading dimensions of matrices referenced by uA, uB and uC.
- scale is returned scaling factor to avoid overflow.
- info is return code from the function DTRSYL.

The error flag E is set to on if:

- the reference uA or uB or uC is not defined (i.e. input uA or uB or uC is not connected),
- trana or tranb is less than 0 or greater than 3
- number of columns of the matrix referenced by uA is not equal to M
- number of rows of the matrix referenced by uB is not equal to N
- number of rows of the matrix referenced by uC is not equal to N or number of columns of this matrix is not equal to M,

- the call of the function DTRSYL returns error using the function XERBLA, see the system log.

See LAPACK documentation [7] for more details.

Inputs

uA	Input reference to matrix A		Reference
uB	Input reference to matrix B		Reference
uC	Input reference to matrix C		Reference
trana	Transposition of matrix A	↓0 ↑3	Long (I32)
tranb	Transposition of matrix B	↓0 ↑3	Long (I32)
isgn	Sign in the equation (1 or -1)	↓-1 ↑1	Long (I32)
HLD	Hold		Bool

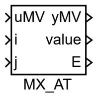
Outputs

yA	Output reference to matrix A	Reference
yB	Output reference to matrix B	Reference
yC	Output reference to matrix C	Reference
scale	Scale	Double (F64)
E	Error indicator	Bool
info	LAPACK function result info. If info = -i, the i=th argument had an illegal value	Long (I32)

MX_AT – Get Matrix/Vector element

Block Symbol

Licence: STANDARD



Function Description

The function block MX_AT returns the value (output value) of the matrix element at the i-th row and j-th column or the i-th vector element.

The output reference yMV is always set to the corresponding input reference uMV to the connected matrix/vector.
The error flag E is set to on if:

- the reference uMV is not defined (i.e. input uMV is not connected),
- the zero based row index $i < 0$ or $i \geq m$, where m is the number of rows,
- the zero based column index $j < 0$ or $j \geq n$, where n is the number of columns. Note that j must be 0 for a vector.

Inputs

uMV	Input reference to a matrix or vector		Reference
i	Row index of the element	↓0	Long (I32)
j	Column index of the element	↓0	Long (I32)

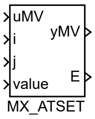
Outputs

yMV	Output reference to a matrix or vector	Reference
value	Value of element at position (i,j)	Long (I32)
E	Error indicator	Bool

MX_ATSET – Set Matrix/Vector element

Block Symbol

Licence: STANDARD



Function Description

The function block MX_ATSET sets the value (input value) to the matrix element at the i-th row and j-th column or to the i-th vector element.

The output reference yMV is always set to the corresponding input reference uMV to the connected matrix/vector.
The error flag E is set to on if:

- the reference uMV is not defined (i.e. input uMV is not connected),
- the zero based row index $i < 0$ or $i \geq m$, where m is the number of rows,
- the zero based column index $j < 0$ or $j \geq n$, where n is the number of columns. Note that j must be 0 for a vector.

Inputs

uMV	Input reference to a matrix or vector		Reference
i	Row index of the element	↓0	Long (I32)
j	Column index of the element	↓0	Long (I32)
value	Value which should be set to the element at position (i,j)		Long (I32)

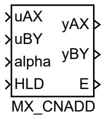
Outputs

yMV	Output reference to a matrix or vector	Reference
E	Error indicator	Bool

MX_CNADD – Add scalar to each Matrix/Vector element

Block Symbol

Licence: STANDARD



Function Description

The function block MX_CNADD adds the value of the input alpha to each matrix/vector element referenced by uAX and the result is stored to the matrix/vector referenced by uBY. If HLD = on then nothing is computed.

The output references yAX and yBY are always set to the corresponding input references uAX and uBY. The dimensions of the matrix/vector referenced by uBY are set to the dimensions of the matrix/vector referenced by uAX if they are different.
The error flag E is set to on if:

- the reference uAX of uBY is not defined (i.e. input uAX or uBY is not connected),
- the count of allocated elements of the matrix/vector referenced by uAX is different from the count of allocated elements of the matrix/vector referenced by uBY.

Inputs

uAX	Input reference to the matrix A or vector X	Reference
uBY	Input reference to the matrix B or vector Y	Reference
alpha	Scalar coefficient alpha	Double (F64)
HLD	Hold	Bool

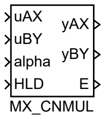
Outputs

yAX	Output reference to the matrix A or vector X	Reference
yBY	Output reference to the matrix B or vector Y	Reference
E	Error indicator	Bool

MX_CNMUL – Multiply a Matrix/Vector by a scalar

Block Symbol

Licence: STANDARD



Function Description

The function block `MX_CNADD` multiplies each matrix/vector element referenced by `uAX` by the value of the input `alpha` and the result is stored to the matrix/vector referenced by `uBY`. If `HLD = on` then nothing is computed.

The output references `yAX` and `yBY` are always set to the corresponding input references `uAX` and `uBY`. The dimensions of the matrix/vector referenced by `uBY` are set to the dimensions of the matrix/vector referenced by `uAX` if they are different.

The error flag `E` is set to `on` if:

- the reference `uAX` of `uBY` is not defined (i.e. input `uAX` or `uBY` is not connected),
- the count of allocated elements of the matrix/vector referenced by `uAX` is different from the count of allocated elements of the matrix/vector referenced by `uBY`.

Inputs

`uAX`	Input reference to the matrix A or vector X	`Reference`
`uBY`	Input reference to the matrix B or vector Y	`Reference`
`alpha`	Scalar coefficient alpha	`Double (F64)`
`HLD`	Hold	`Bool`

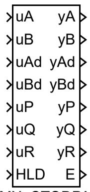
Outputs

`yAX`	Output reference to the matrix A or vector X	`Reference`
`yBY`	Output reference to the matrix B or vector Y	`Reference`
`E`	Error indicator	`Bool`

MX_CTODPA – Discretizes continuous model given by (A,B) to (Ad,Bd) using Pade approximations

Block Symbol

Licence: STANDARD



MX_CTODPA

Function Description

This function block discretizes a continuous state space model using Padé approximations of matrix exponential and its integral and scaling technique ([5]). The used technique is similar to method 3 Scaling and squaring described in [8].

The output references `yA`, `yB`, `yAd`, `yBd`, `yP` , `yQ` and `yR` are always set to the corresponding input references `uA`, `uB`, `uAd`, `uBd`, `uP`, `uQ` and `uR`. If `HLD = on` then nothing is computed otherwise the function `mCtoD` is called internally:

```
mCtoD(nRes, uAd, uBd, uA, uB, N, M, is, Ts, eps, uP, uQ, uR);
```

where parameters of `mCtoD` are set in the following way:

- `nRes` is return code from the function `mCtoD`.
- `N` is number of rows of the square system matrix referenced by `uA`.
- `M` is number of columns of the input matrix referenced by `uB`.
- `Ts` is sampling period for the discretization, which is equal to sampling period of the task containing this function block.

The error flag `E` is set to `on` if:

- the reference `uA` or `uB` or `uAd` or `uBd` or `uP` or `uQ` or `uR` is not defined (i.e. input `uA` or `uB` or `uAd` or `uBd` or `uP` or `uQ` or `uR` is not connected),
- number of columns of the matrix referenced by `uA` is not equal to `N`,
- number of rows of the matrix referenced by `uB` is not equal to `N`,
- number of elements of any matrix referenced by `uAd`, `uP`, `uQ` or `uR` is less than $N*N$,
- number of elements of the matrix referenced by `uBd` is less than $N*M$,

- the return code `nRes` of the function `mCtoD` is not equal to zero.

Inputs

uA	Input reference to matrix A	Reference
uB	Input reference to matrix B	Reference
uAd	Input reference to discretized matrix A	Reference
uBd	Input reference to discretized matrix B	Reference
uP	Input reference to a helper matrix	Reference
uQ	Input reference to a helper matrix	Reference
uR	Input reference to a helper matrix	Reference
HLD	Hold	Bool

Parameters

is	Pade approximation order	↓0 ↑4 ⊙2	Long (I32)
eps	Approximation accuracy	↓1e-20 ↑0.001 ⊙1e-15	Double (F64)

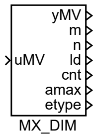
Outputs

yA	Output reference to matrix A	Reference
yB	Output reference to matrix B	Reference
yAd	Output reference to discretized matrix A	Reference
yBd	Output reference to discretized matrix B	Reference
yP	Output reference to a helper matrix	Reference
yQ	Output reference to a helper matrix	Reference
yR	Output reference to a helper matrix	Reference
E	Error indicator	Bool

MX_DIM – Matrix/Vector dimensions

Block Symbol

Licence: STANDARD



Function Description

The function block MX_DIM sets its outputs to the dimensions of the matrix or vector referenced by uMV.

The output reference yMV is always set to the corresponding input reference uMV. The error flag E is set to on if the reference uMV is not defined (i.e. input uMV is not connected).

Input

uMV	Input reference to a matrix or vector	Reference

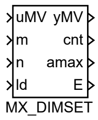
Outputs

yMV	Output reference to a matrix or vector	Reference
m	Number of matrix rows	Long (I32)
n	Number of matrix columns	Long (I32)
ld	Leading dimension (>= number of rows)	Long (I32)
cnt	Count of used matrix/vector elements	Long (I32)
amax	Count of reserved/allocated matrix/vector elements	Long (I32)
etype	Matrix/vector element type (double, long, byte etc.)	Long (I32)

MX_DIMSET – Set Matrix/Vector dimensions

Block Symbol

Licence: STANDARD



Function Description

The function block MX_DIMSET sets number rows m of the vector or number of rows m, number of columns n and the leading dimension ld of the matrix referenced by uMV. If any of the inputs m, n, ld is not connected, its original value is retained.

The output cnt contains the actual number of occupied elements of the matrix/vector and is determined by the formula

$$\texttt{cnt} = \texttt{ld} * (\texttt{n} - 1) + \texttt{m} \leq \texttt{amax} \ ,$$

where the output amax is the allocated count of matrix/vector elements. If this inequality is fulfilled the output cnt is set to the matrix/vector structure and can be retrieved by the MX_DIM block, otherwise the value of cnt shows the minimum necessary number of elements of the matrix/vector.

The output reference yMV is always set to the corresponding input reference uMV. The error flag E is set to on if:

- the reference uMV is not defined (i.e. input uMV is not connected),
- the number of rows m $<$ 1 or m $>$ ld,
- the number of columns n $<$ 1,
- the required number of elements cnt $>$ amax.

Inputs

uMV	Input reference to a matrix or vector	Reference
m	Number of matrix rows	Long (I32)
ld	Leading dimension (>= number of rows)	Long (I32)

Outputs

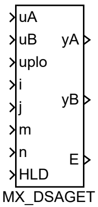
yMV	Output reference to a matrix or vector	Reference
cnt	Count of used matrix/vector elements	Long (I32)
amax	Number of allocated matrix/vector elements	Long (I32)

`E`	Error indicator	`Bool`

MX_DSAGET – Set subarray of A into B

Block Symbol

Licence: STANDARD



Function Description

Generally, the function block MX_DSAGET copies the subarray (submatrix) of matrix referenced by uA into the matrix referenced by uB.

The output references yA and yB are always set to the corresponding input references uA and uB. If HLD = on then nothing is copied otherwise the submatrix of matrix referenced by uA starting the row with zero based index I and the column with zero based index J containing M rows and N columns is copied (with respect to the value of the input uplo) to the matrix referenced by uB. The mentioned variables have the following meanings:

- If the input $i \leq 0$ then I is set to 0 else if $i \geq MA$ then I is set to $MA - 1$ else I is set to i, where MA is the number of rows of the matrix referenced by uA.
- If the input $j \leq 0$ then J is set to 0 else if $j \geq NA$ then J is set to $NA - 1$ else J is set to j, where NA is the number of columns of the matrix referenced by uA.
- Number of copied rows M is set in two stages. First, M is set to minimum of $MA - I$ and number of rows of the matrix referenced by uB. Second, if $m > 0$ then M is set to the minimum of m and M.
- Number of copied columns N is set in two stages. First, N is set to minimum of $NA - J$ and number of columns of the matrix referenced by uB. Second, if $n > 0$ then N is set to the minimum of n and N.

The error flag E is set to on if:

- the reference uA or uB is not defined (i.e. input uA or uB is not connected),
- uplo is less than 0 or greater than 3,
- the number of elements of the matrix referenced by uB is less than $M * N$.

Inputs

uA	Input reference to matrix A	Reference
uB	Input reference to matrix B	Reference
uplo	Part of the matrix to be copied 0 ..... All 2 ..... Upper 1 ..... All 3 ..... Lower	Long (I32)
i	Index of the subarray first row	Long (I32)
j	Index of the subarray first column	Long (I32)
m	Number of matrix rows	Long (I32)
n	Number of matrix columns	Long (I32)
HLD	Hold	Bool

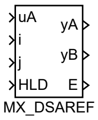
Outputs

yA	Output reference to matrix A	Reference
yB	Output reference to matrix B	Reference
E	Error indicator	Bool

MX_DSAREF – Set reference to subarray of A into B

Block Symbol

Licence: STANDARD



Function Description

The function block MX_DSAREF creates a reference yB to the subarray (submatrix) of matrix referenced by uA. This operation is very fast because no matrix element is copied.

The output reference yA is always set to the corresponding input reference uA, the output reference yB is created inside each instance of this function block. If HLD = on then no other operation is performed otherwise the reference to the matrix yB is created with the following properties:

- Number of rows of the submatrix is set to $\mathtt{M} - \mathtt{i}$, where M is number of rows of the matrix referenced by uA.
- Number of columns of the submatrix is set to $\mathtt{N} - \mathtt{j}$, where N is number of columns of the matrix referenced by uA.
- The first element in position $(0, 0)$ of the submatrix is the element of the matrix referenced by uA in position $(\mathtt{i}, \mathtt{j})$, all indices are zero based.
- The matrix referenced by yB has the same leading dimension as the matrix referenced by uA.

The error flag E is set to on if:

- the reference uA is not defined (i.e. input uA is not connected),
- $0 > \mathtt{i} \geq \mathtt{M}$.
- $0 > \mathtt{j} \geq \mathtt{N}$.

Inputs

uA	Input reference to matrix A	Reference
i	Index of the subarray first row	Long (I32)
j	Index of the subarray first column	Long (I32)
HLD	Hold	Bool

Parameter

`ay`	Output reference of the subarray	⊙[0 0]	`Double (F64)`

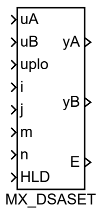
Outputs

`yA`	Output reference to matrix A	`Reference`
`yB`	Output reference to matrix B	`Reference`
`E`	Error indicator	`Bool`

MX_DSASET – Set A into subarray of B

Block Symbol

Licence: STANDARD



Function Description

Generally, the function block MX_DSASET copies the matrix referenced by uA into the subarray (submatrix) of the matrix referenced by uB.

The output references yA and yB are always set to the corresponding input references uA and uB. If HLD = on then nothing is copied otherwise the matrix referenced by uA is copied (with respect to the value of the input uplo) to the submatrix of the matrix referenced by uB to the row with zero based index I and the column with zero based index J containing M rows and N columns. The mentioned variables have the following meanings:

- If the input $i \leq 0$ then I is set to 0 else if $i \geq MB$ then I is set to $MB - 1$ else I is set to i, where MB is the number of rows of the matrix referenced by uB.
- If the input $j \leq 0$ then J is set to 0 else if $j \geq NB$ then J is set to $NB - 1$ else J is set to j, where NB is the number of columns of the matrix referenced by uB.
- Number of copied rows M is set in two stages. First, M is set to minimum of $MB - I$ and number of rows of the matrix referenced by uA. Second, if $m > 0$ then M is set to the minimum of m and M.
- Number of copied columns N is set in two stages. First, N is set to minimum of $NB - J$ and number of columns of the matrix referenced by uA. Second, if $n > 0$ then N is set to the minimum of n and N.

The error flag E is set to on if:

- the reference uA or uB is not defined (i.e. input uA or uB is not connected),
- uplo is less than 0 or greater than 3,
- the number of elements of the matrix referenced by uB is less than $M * N$.

Inputs

uA	Input reference to matrix A	Reference
uB	Input reference to matrix B	Reference
uplo	Part of the matrix to be copied 0 ..... All 2 ..... Upper 1 ..... All 3 ..... Lower	Long (I32)
i	Index of the subarray first row	Long (I32)
j	Index of the subarray first column	Long (I32)
m	Number of matrix rows	Long (I32)
n	Number of matrix columns	Long (I32)
HLD	Hold	Bool

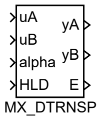
Outputs

yA	Output reference to matrix A	Reference
yB	Output reference to matrix B	Reference
E	Error indicator	Bool

MX_DTRNSP – General matrix transposition: B := alpha*A^T

Block Symbol

Licence: STANDARD



Function Description

The function block MX_DTRNSP stores the scalar multiple of the general (i.e. rectangular) matrix referenced by uA into the matrix referenced by uB.

The output references yA and yB are always set to the corresponding input references uA and uB. If HLD = on then nothing else is done otherwise the BLAS-like function X_DTRNSP is called internally:

```
X_DTRNSP(M, N, ALPHA, uA, LDA, uB, LDB);
```

where parameters of X_DTRNSP are set in the following way:

- M is number of rows of the matrix referenced by uA.
- N is number of columns of the matrix referenced by uA.
- If the input alpha is equal to 0 then ALPHA is set to 1 else ALPHA is set to alpha.
- LDA and LDB are leading dimensions of matrices referenced by uA and uB.

The error flag E is set to on if:

- the reference uA or uB is not defined (i.e. input uA or uB is not connected),
- the call of the function X_DTRNSP returns error using the function XERBLA, see the system log.

Inputs

uA	Input reference to matrix A	Reference
uB	Input reference to matrix B	Reference
alpha	Scalar coefficient alpha	Double (F64)
HLD	Hold	Bool

Outputs

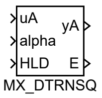
yA	Output reference to matrix A	Reference

yB	Output reference to matrix B	Reference
E	Error indicator	Bool

MX_DTRNSQ – Square matrix in-place transposition: A := alpha*A^T

Block Symbol

Licence: STANDARD



Function Description

The function block `MX_DTRNSQ` transpose the scalar multiple of the square matrix referenced by `uA` in-place.

The output reference `yA` is always set to the corresponding input references `uA`. If `HLD = on` then nothing else is done otherwise the BLAS-like function `X_DTRNSQ` is called internally:

```
X_DTRNSQ(N, ALPHA, uA, LDA);
```

where parameters of `X_DTRNSQ` are set in the following way:

- `N` is number of rows and columns of the matrix referenced by `uA`.
- If the input `alpha` is equal to 0 then `ALPHA` is set to 1 else `ALPHA` is set to `alpha`.
- `LDA` is the leading dimension of the matrix referenced by `uA`.

The error flag `E` is set to `on` if:

- the reference `uA` is not defined (i.e. input `uA` is not connected),
- the matrix referenced by `uA` is not square,
- the call of the function `X_DTRNSQ` returns error using the function `XERBLA`, see the system log.

Inputs

`uA`	Input reference to matrix A	`Reference`
`alpha`	Scalar coefficient alpha	`Double (F64)`
`HLD`	Hold	`Bool`

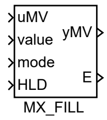
Outputs

`yA`	Output reference to matrix A	`Reference`
`E`	Error indicator	`Bool`

MX_FILL – Fill real matrix or vector

Block Symbol

Licence: STANDARD



Function Description

The function block MX_FILL fills elements of the matrix or vector referenced by uMV according to the input mode.

The output reference yMV is always set to the corresponding input references uMV. If HLD = on then nothing else is done.

The error flag E is set to on if:

- the reference uMV is not defined (i.e. input uMV is not connected),
- $0 >$ mode > 4.

Inputs

uMV	Input reference to a matrix or vector	Reference
value	Fill value of matrix/vector	Double (F64)
mode	Fill mode 0,1 . . . Value – All elements are set to value 2 . . . . . Ones – All elements are set to 1 3 . . . . . Diagonal value – Diagonal is set to value, the other elements to 0 4 . . . . . Diagonal ones – Initializes identity matrix (eye)	Long (I32)
HLD	Hold	Bool

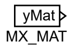
Outputs

yMV	Output reference to a matrix or vector	Reference
E	Error indicator	Bool

MX_MAT – Matrix data storage block

Block Symbol

Licence: STANDARD



Function Description

The function block MX_MAT allocates memory (during the block initialization) for $m * n$ elements of the type determined by the parameter etype of the matrix referenced by the output yMat. Also matrix leading dimension can be set by the parameter ld. If $ld < m$ then the leading dimension is set to m.

Note that the present version of the MATRIX function block set supports only matrices with the etype equal to 8: Double.

Parameters

m	Number of matrix rows	↓1 ↑1000000000 ⊙10	Long (I32)
n	Number of matrix columns	↓1 ↑1000000000 ⊙10	Long (I32)
ld	Leading dimension (>= number of rows)	↓0 ↑1000000000	Long (I32)
etype	Type of elements	⊙8	Long (I32)

1	Bool	6	DWord (U32)
2	Byte (U8)	7	Float (F32)
3	Short (I16)	8	Double (F64)
4	Long (I32)	--	
5	Word (U16)	10	Large (I64)

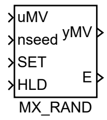
Output

yMat	Output reference to a matrix	Reference

MX_RAND – Randomly generated matrix or vector

Block Symbol

Licence: STANDARD



Function Description

The function block MX_RAND generates random elements of the matrix or vector referenced by uMV.

The output reference yMV is always set to the corresponding input references uMV. If HLD = on then nothing is generated otherwise pseudo-random values of the matrix or vector elements referenced by uMV are generated using these rules:

- If the parameter BIP is on then the generated elements are inside the interval [−scale; scale] else they are inside the interval [0; scale].
- Elements are internally generated using the standard C language function rand() which generates pseudo-random numbers in the range from 0 to RAND_MAX. Note, that the value of RAND_MAX can be platform dependent (and it should be at least 32767).
- The rising edge on the input SET causes that the standard C language function srand(nseed) (initailizes the pseudo-random generator with the value of nseed) is called before the generation of random elements. The same sequences of pseudo-random numbers are generated after calls of srand(nseed) for the same values of nseed.

The error flag E is set to on if the reference uMV is not defined (i.e. input uMV is not connected).

Inputs

uMV	Input reference to a matrix or vector	Reference
nseed	Random number seed	Long (I32)
SET	Set initial value of random number generator to nseed on rising edge	Bool
HLD	Hold	Bool

Parameters

BIP	Bipolar random values flag	Bool

`scale`	Random values multiplication factor	⊙1.0	`Double (F64)`

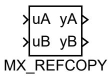
Outputs

`yMV`	Output reference to a matrix or vector	`Reference`
`E`	Error indicator	`Bool`

MX_REFCOPY – Copies input references of matrices A and B to their output references

Block Symbol

Licence: STANDARD



Function Description

The function block MX_REFCOPY is an administrative block of the MATRIX blockset. It does nothing else than copying the input references uA and uB to the corresponding output references yA and yB.

But suitable insertion of this block to the function block scheme can substantially influence (change) the execution order of blocks which can be very advantageous especially in combination with such blocks as e.g. MX_DSAREF.

Inputs

uA	Input reference to matrix A	Reference
uB	Input reference to matrix B	Reference

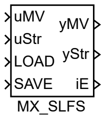
Outputs

yA	Output reference to matrix A	Reference
yB	Output reference to matrix B	Reference

MX_SLFS – Save or load a Matrix/Vector into file or string

Block Symbol

Licence: STANDARD



Function Description

The block allows to convert a matrix or vector into text form and vice versa. The matrix is supplied as a reference to the uMV input. The yMV output refers to the same matrix as the uMV input, and is intended to chain matrix blocks in the correct order, as is common with all MATRIX blocks. The text can be either in the input uStr (or output yStr for the opposite direction of conversion) or in the file. If the text is in a file, its name is the string connected to the uStr input. The usual REXYGENsystem file name rules applies , ie it is relative to datadir and ../ is not allowed to leave the directory. If the uStr input is unattached (or empty string), the path name of the file is used with the full path (that is, including the task name and all subsystems) with the .dat extension.

The format of a matrix in a text file or in text input and output is determined by the format parameter. Supported English and Czech CSV (i.e., columns separated by comma or semicolon), JSON format (created by Google and often used in web applications) and the format used by MATLAB (for entering a matrix in MATLAB scripts).

Conversion from text to matrix/vector or vice versa can be performed at each step of the algorithm or is triggered by the LOAD and SAVE inputs. The exact method is determined by the mode parameter and is explained in detail in the description of this parameter. If an error occurs, it is signaled to the iE output and in the log. After a fatal error, the conversion from/to the matrix stops. Error reset for mode = 1 .. 4 is done by setting LOAD = SAVE = off, resetting fatal error cannot be performed for mode = 5 .. 8 (must switch to mode = 1 .. 4 and then back).

The nmax parameter is used to alocate the output string. If nmax> 0, it is allocated specified number of chars during initialization. If this amount is insufficient, the block reports an error. If nmax = 0, the block increases the length of the output string as needed. If user don't specify the nmax parameter it can lead to full RAM memory in extreme situations and unpredictable behaviour of entire system.

Inputs

uMV	Input reference to a matrix or vector	Reference
uStr	Input string (to convert into matrix) or filename	String
LOAD	Trigger to move data to matrix/vector	Bool
SAVE	Trigger to move data from matrix/vector	Bool

Parameters

`mode`	Triggering mode 1 ..... level-triggered file 2 ..... edge-triggered file 3 ..... level-triggered string 4 ..... edge-triggered string 5 ..... continuous string to matrix 6 ..... continuous matrix to string 7 ..... continuous file to matrix 8 ..... continuous matrix to file	⊙2	`Long (I32)`
`format`	String/file format 1 ..... CSV 2 ..... CSV(semicolon) 3 ..... JSON 4 ..... MATLAB	⊙1	`Long (I32)`
`prec`	Number of digits for single value	↓0 ↑20 ⊙6	`Long (I32)`
`TRN`	Transposition flag		`Bool`
`nmax`	Allocated size of string	↓0	`Long (I32)`

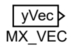
Outputs

`yMV`	Output reference to a matrix or vector	`Reference`
`yStr`	String representation of the matrix/vector	`String`
`iE`	Error code	`Error`

MX_VEC – Vector data storage block

Block Symbol

Licence: STANDARD



Function Description

The function block MX_VEC allocates memory (during the block initialization) for n elements of the type determined by the parameter etype of the vector referenced by the output yVec.

Note that the present version of the MATRIX function block set supports only vectors with the etype equal to 8: Double.

Parameters

n	Number of vector elements	↓1 ↑1000000000 ⊙10	Long (I32)
etype	Type of elements	⊙8	Long (I32)

1 ..... Bool
2 ..... Byte (U8)
3 ..... Short (I16)
4 ..... Long (I32)
5 ..... Word (U16)
6 ..... DWord (U32)
7 ..... Float (F32)
8 ..... Double (F64)
-- ....
10 .... Large (I64)

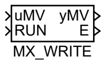
Output

yVec	Output reference to a vector	Reference

MX_WRITE – Write a Matrix/Vector to the console/system log

Block Symbol

Licence: STANDARD



Function Description

This function block can write a vector or matrix to the console or the system log. The severity of the console/system log output is set by the parameter `mode` in combination with settings of system log from REXYGEN Studio, menu `Target/Configure System Log`. Written data can be viewed in REXYGEN Studio, after opening the system log window by the command `Target/Show System Log`. The function block is very useful for debugging purposes of matrix/vector algorithms.

The output references `yMV` is always set to the input reference `uMV`. If `RUN = off` then nothing else is done otherwise matrix or vector is written to the system log if the configured target logging level for function blocks contains the configured `mode`. Format of each matrix/vector element is determined by parameters `mchars` and `mdec`.
The error flag `E` is set to `on` if:

- the reference `uMV` is not defined (i.e. input `uMV` is not connected),
- $3 >$ `mchars` > 25,
- $0 >$ `mdec` $>$ `mchars` $- 2$.

Inputs

uMV	Input reference to a matrix or vector	Reference
RUN	Enable execution	Bool

Parameters

Symbol	Matrix/vector symbolic name for console or log output	⊙A	String
mchars	Number of characters per single element	↓3 ↑25 ⊙8	Long (I32)
mdec	Number of decimal digits per single element	↓0 ↑23 ⊙4	Long (I32)
mode	Severity mode of writing 1 ..... None 2 ..... Verbose 3 ..... Information 4 ..... Warning 5 ..... Error	⊙3	Long (I32)

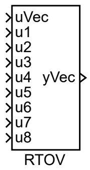
Outputs

`yMV`	Output reference to a matrix or vector	`Reference`
`E`	Error indicator	`Bool`

RTOV – Vector multiplexer

Block Symbol

Licence: STANDARD



Function Description

The RTOV block can be used to create vector signals in the REXYGEN system. It combines the scalar input signals into one vector output signal.

It is also possible to chain the RTOV blocks to create signals with more than 8 items.

The **nmax** parameter defines the maximal number of items in the vector (in other words, the size of memory allocated for the signal). The **offset** parameter defines the position of the first input signal **u1** in the resulting signal. Only the first **n** input signals are combined into the resulting **yVec** vector signal.

ATTENTION: Up to version 2.50.10.x output vector is one-row-matrix. Later version (2.51.0.9525 and later) use one-column-matrix. This change was necessary for consistance in matrix operation.

Inputs

uVec	Vector signal	Reference
u1	Analog input of the block	Double (F64)
u2	Analog input of the block	Double (F64)
u3	Analog input of the block	Double (F64)
u4	Analog input of the block	Double (F64)
u5	Analog input of the block	Double (F64)
u6	Analog input of the block	Double (F64)
u7	Analog input of the block	Double (F64)
u8	Analog input of the block	Double (F64)

Parameters

nmax	Allocated size of vector	$\downarrow$0 $\odot$8	Long (I32)
offset	Index of the first input in vector	$\downarrow$0	Long (I32)
n	Number of valid inputs	$\downarrow$0 $\uparrow$8 $\odot$8	Long (I32)

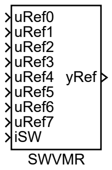
Output

yVec	Vector signal	Reference

SWVMR – Vector/matrix/reference signal switch

Block Symbol

Licence: STANDARD



Function Description

The SWVMR allows switching of vector or matrix signals. It also allow switching of motion axes in motion control algorithms (see the RM_Axis block).

Use the SSW block or its alternatives SWR and SELU for switching simple signals.

Inputs

uRef0	Vector signal	Reference
uRef1	Vector signal	Reference
uRef2	Vector signal	Reference
uRef3	Vector signal	Reference
uRef4	Vector signal	Reference
uRef5	Vector signal	Reference
uRef6	Vector signal	Reference
uRef7	Vector signal	Reference
iSW	Active signal selector	Long (I32)

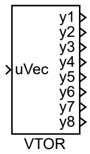
Output

yRef	Vector signal	Reference

VTOR – Vector demultiplexer

Block Symbol

Licence: STANDARD



Function Description

The VTOR block splits the input vector signal into individual signals. The user defines the starting item and the number of items to feed to the output signals using the offset and N parameter respectively.

Input

uVec	Vector signal	Reference

Parameters

n	Number of valid outputs	↓0 ↑8 ⊙8	Long (I32)
offset	Index of the first output	↓0	Long (I32)

Outputs

y1	Analog output of the block	Double (F64)
y2	Analog output of the block	Double (F64)
y3	Analog output of the block	Double (F64)
y4	Analog output of the block	Double (F64)
y5	Analog output of the block	Double (F64)
y6	Analog output of the block	Double (F64)
y7	Analog output of the block	Double (F64)
y8	Analog output of the block	Double (F64)

Chapter 15

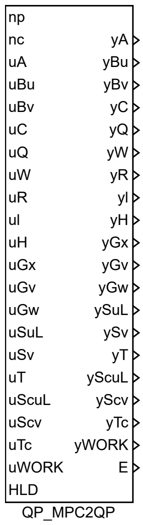
OPTIM – Optimization blocks

Contents

QP_MPC2QP – Conversion of MPC problem to quadratic programming

Block Symbol

Licence: ADVANCED



Function Description

Quadratic Programming (QP) is a standard technique which suites very well to solve model based predictive control (MPC) problems [9]. Quadratic Programming is an optimization technique that minimizes the sum of quadratic form and linear form.

The QP_MPC2QP block converts a linear MPC problem with quadratic optimization criterion to a quadratic programming problem. The block is compatible with the block QP_UPDATE and the QP solver QP_OASES.

MPC problem formulation

The MPC problem consists of a discrete linear time invariant state space model

$$\begin{aligned} x_{k+1} &= Ax_k + B_u u_k + B_v v_k \,, \\ y_k &= Cx_k \,, \end{aligned} \tag{15.1}$$

where $x \in \mathbb{R}^n$ is the state vector, $u \in \mathbb{R}^{m_u}$ is the input vector, $v \in \mathbb{R}^{m_v}$ is the disturbance vector and $y \in \mathbb{R}^p$ is the output vector. Matrices $A \in \mathbb{R}^{n\times n}$, $B_u \in \mathbb{R}^{n\times m_u}$, $B_v \in \mathbb{R}^{n\times m_v}$ and $C \in \mathbb{R}^{p\times n}$ are referenced by inputs uA, uBu, uBv and uC. The model based predictive control problem is formulated as an optimization problem – minimization of the quadratic optimality criterion (cost function) in the form

$$J = \sum_{k=1}^{n_p} \left\{ \hat{x}_k^T Q \hat{x}_k + \hat{x}_k^T W + \hat{u}_{k-1}^T R \hat{u}_{k-1} \right\} , \tag{15.2}$$

where symmetric and positive (semi-)definite matrices $Q \in \mathbb{R}^{n \times n}$ and $R \in \mathbb{R}^{m_u \times m_u}$ and the vector $W \in \mathbb{R}^n$ are referenced by inputs `uQ`, `uR` and `uW`, and n_p is the prediction horizon (input `np`).

Additional constraints on the state x and the output y may be required for the minimization process:

$$x_{\min} \leq x_k \leq x_{\max} \tag{15.3}$$

$$y_{\min} \leq y_k \leq y_{\max} \tag{15.4}$$

Predictor

From the state equation with the initial condition x_0 it holds

$$\begin{aligned}
x_1 &= Ax_0 + B_u u_0 + B_v v_0 \\
x_2 &= Ax_1 + B_u u_1 + B_v v_1 \\
&= A^2 x_0 + AB_u u_0 + B_u u_1 + AB_v v_0 + B_v v_1 \\
&\vdots \\
x_k &= A^k x_0 + \begin{bmatrix} A^{k-1}B_u & \dots & AB_u & B_u \end{bmatrix} \begin{bmatrix} u_0 \\ \vdots \\ u_{k-1} \end{bmatrix} \\
&\quad + \begin{bmatrix} A^{k-1}B_v & \dots & AB_v & B_v \end{bmatrix} \begin{bmatrix} v_0 \\ \vdots \\ v_{k-1} \end{bmatrix}
\end{aligned}$$

Thus, for the prediction horizon n_p we have

$$\underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n_p} \end{bmatrix}}_{X} = \underbrace{\begin{bmatrix} B_u & 0 & \dots & 0 \\ AB_u & B_u & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ A^{n_p-1}B_u & A^{n_p-2}B_u & \dots & B_u \end{bmatrix}}_{S_u} \underbrace{\begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{n_p-1} \end{bmatrix}}_{U}$$

$$+ \underbrace{\begin{bmatrix} B_v & 0 & \dots & 0 \\ AB_v & B_v & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ A^{n_p-1}B_v & A^{n_p-2}B_v & \dots & B_v \end{bmatrix}}_{S_v} \underbrace{\begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_{n_p-1} \end{bmatrix}}_{V} + \underbrace{\begin{bmatrix} A \\ A^2 \\ \vdots \\ A^{n_p} \end{bmatrix}}_{T} x_0$$

i.e.

$$X = S_u U + S_v V + T x_0 \tag{15.5}$$

Similarly, for the output equation we can get

$$\underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n_p} \end{bmatrix}}_{Y} = \begin{bmatrix} Cx_1 \\ Cx_2 \\ \vdots \\ Cx_{n_p} \end{bmatrix} = \underbrace{\begin{bmatrix} CB_u & 0 & \dots & 0 \\ CAB_u & CB_u & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ CA^{n_p-1}B_u & CA^{n_p-2}B_u & \dots & CB_u \end{bmatrix}}_{S_{cu}} \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{n_p-1} \end{bmatrix}$$

$$+ \underbrace{\begin{bmatrix} CB_v & 0 & \dots & 0 \\ CAB_v & CB_v & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ CA^{n_p-1}B_v & CA^{n_p-2}B_v & \dots & CB_v \end{bmatrix}}_{S_{cv}} \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_{n_p-1} \end{bmatrix} + \underbrace{\begin{bmatrix} CA \\ CA^2 \\ \vdots \\ CA^{n_p} \end{bmatrix}}_{T_c} x_0$$

i.e.

$$Y = S_{cu} U + S_{cv} V + T_c x_0 \tag{15.6}$$

and standard QP matrices A_{eq} and b_{eq}

$$A_{eq} = S_{cu} L, \quad b_{eq} = -S_{cv} V - T_c x_0 \tag{15.7}$$

Predictor for control horizon less than prediction horizon

Until now, it was assumed that optimal control would be sought over the entire prediction horizon n_p. For a long prediction horizon, this leads to time-consuming optimization, which can be accelerated by choosing a control horizon n_c (input `nc`) smaller than the prediction horizon n_p. Then U can be written as

$$U = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{n_c-1} \\ u_{n_c-1} \\ \vdots \\ u_{n_c-1} \end{bmatrix} \begin{matrix} \left.\vphantom{\begin{matrix} a \\ a \\ a \\ a \end{matrix}}\right\} n_c \\ \left.\vphantom{\begin{matrix} a \\ a \\ a \end{matrix}}\right\} n_p - n_c \end{matrix} \Bigg\} n_p$$

Note that the input u_k is the difference of the state and for control horizon n_c it holds $u_{k+n_c-1} = u_{k+n_c} = u_{k+n_c+1} = \dots = u_{k+n_p-1} = 0$ (for the step k). Then it can be written as

$$U = \begin{matrix} n_c \left\{ \begin{matrix} \\ \\ \\ \end{matrix} \right. \\ n_p - n_c \left\{ \begin{matrix} \\ \\ \end{matrix} \right. \end{matrix} \underbrace{\begin{bmatrix} I_{m_u} & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & I_{m_u} \\ & \mathbf{0} & \\ & & \end{bmatrix}}_{L} \begin{bmatrix} u_0 \\ \vdots \\ u_{n_c-1} \end{bmatrix} \triangleq LU_{n_c} \tag{15.8}$$

where $U \in \mathbb{R}^{n_p \cdot m_u}$, $U_{n_c} \in \mathbb{R}^{n_c \cdot m_u}$ and $I_{m_u} \in \mathbb{R}^{m_u \times m_u}$ is identity matrix.

The equations (15.5) and (15.6) are modified for U_{n_c} to

$$X = S_u LU_{n_c} + S_v V + Tx_0 \tag{15.9}$$
$$Y = S_{cu} LU_{n_c} + S_{cv} V + T_c x_0 \tag{15.10}$$

Matrices $S_u L \in \mathbb{R}^{n_p \cdot n \times n_c \cdot m_u}$, $S_v \in \mathbb{R}^{n_p \cdot n \times n_p \cdot m_v}$, $T \in \mathbb{R}^{n_p \cdot n \times n}$, $S_{cu} L \in \mathbb{R}^{n_p \cdot p \times n_c \cdot m_u}$, $S_{cv} \in \mathbb{R}^{n_p \cdot p \times n_p \cdot m_v}$ and $T_c \in \mathbb{R}^{n_p \cdot p \times n}$ are computed by this block, must be allocated e.g. by the `MX_MAT` blocks and references to the preallocated matrices must be connected to the block inputs `uSuL`, `uSv`, `uT`, `uScuL`, `uScv` and `uTc`.

The default value of the matrix $L \in \mathbb{R}^{n_p \cdot m_u \times n_c \cdot m_u}$ in equation 15.8 selects the first n_c subvectors $u_i, i = 0, \ldots, n_c - 1$ from U. The block also allows to select n_c subvectors u_i with arbitrary indices from $0, \ldots, n_p$, which are contained in the integer vector of dimension n_c referenced by the input `ul`. The elements of this vector must form an increasing sequence. If the input `ul` is not connected, the default value of L is used (the same value of L is obtained if the vector referenced by `ul` is equal to $[0,\ 1, \ldots, n_c{-}1]^T$).

Conversion of MPC with the same prediction and control horizons to QP

The standard form of cost function for QP is

$$J_{QP} = hU^T HU + U^T G \tag{15.11}$$

where U a vector of optimal control sequence, H is a symmetric and positive (semi-) definite Hessian matrix, G is a gradient vector and h is a scalar constant which is usually equal to 1 or 1/2.

The cost function (15.2) can be modified to the form

$$
\begin{aligned}
J &= \sum_{k=1}^{n_p} \{ x_k^T Q x_k + x_k^T W + u_{k-1}^T R u_{k-1} \} \\
&= \begin{bmatrix} x_1 & x_2 & \dots & x_{n_p} \end{bmatrix} \underbrace{\begin{bmatrix} Q & & & \\ & Q & & \\ & & \ddots & \\ & & & Q \end{bmatrix}}_{\bar{Q} = \bar{Q}^T} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n_p} \end{bmatrix} + \begin{bmatrix} x_1 & x_2 & \dots & x_{n_p} \end{bmatrix} \underbrace{\begin{bmatrix} W \\ W \\ \vdots \\ W \end{bmatrix}}_{\bar{W}} \\
&+ \begin{bmatrix} u_0 & u_1 & \dots & u_{n_p-1} \end{bmatrix} \underbrace{\begin{bmatrix} R & & & \\ & R & & \\ & & \ddots & \\ & & & R \end{bmatrix}}_{\bar{R} = \bar{R}^T} \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{n_p-1} \end{bmatrix} \\
&= X^T \bar{Q} X + X^T \bar{W} + U^T \bar{R} U \\
&= (U^T S_u^T + V^T S_v^T + x_0^T T^T) \bar{Q} (S_u U + S_v V + T x_0) \\
&+ (U^T S_u^T + V^T S_v^T + x_0^T T^T) \bar{W} + U^T \bar{R} U \\
&= U^T S_u^T \bar{Q} S_u U + U^T S_u^T \bar{Q} S_v V + U^T S_u^T \bar{Q} T x_0 \\
&+ V^T S_v^T \bar{Q} S_u U + V^T S_v^T \bar{Q} S_v V + V^T S_v^T \bar{Q} T x_0 \\
&+ x_0^T T^T \bar{Q} S_u U + x_0^T T^T \bar{Q} S_v V + x_0^T T^T \bar{Q} T x_0 \\
&+ U^T S_u^T \bar{W} + V^T S_v^T \bar{W} + x_0^T T^T \bar{W} + U^T \bar{R} U \\
&= U^T (S_u^T \bar{Q} S_u + \bar{R}) U + U^T S_u^T (2 \bar{Q} S_v V + 2 \bar{Q} T x_0 + \bar{W}) \\
&+ \underbrace{V^T S_v^T (\bar{Q} S_v V + 2 \bar{Q} T x_0 + \bar{W}) + x_0^T T^T (\bar{Q} T x_0 + \bar{W})}_{J_{\text{dif}}} \triangleq J_{QP} + J_{\text{dif}}
\end{aligned}
\tag{15.12}
$$

where J_{dif} is a constant independent of U. From here follows

$$
J_{QP} = U^T (S_u^T \bar{Q} S_u + \bar{R}) U + U^T S_u^T (2 \bar{Q} S_v V + 2 \bar{Q} T x_0 + \bar{W}) \tag{15.13}
$$

Comparing this equation with (15.11), it is obvious that

$$
\begin{aligned}
H &= \frac{1}{h} (S_u^T \bar{Q} S_u + \bar{R}) \\
G &= S_u^T (2 \bar{Q} S_v V + 2 \bar{Q} T x_0 + \bar{W})
\end{aligned}
\tag{15.14}
$$

Conversion of MPC with control horizon less than prediction horizon

Similarly as in previous subsection we can get for $n_c < n_p$

$$\begin{aligned} H &= \frac{1}{h} L^T (S_u^T \bar{Q} S_u + \bar{R}) L \\ G &= (S_u L)^T (2\bar{Q} S_v V + 2\bar{Q} T x_0 + \bar{W}) \triangleq G_v V + G_x x_0 + G_w \end{aligned} \tag{15.15}$$

where matrix L is defined by (15.8). The Hessian matrix H is a constant matrix for all steps k of the MPC. But gradient vector G is generally changing in each step k because vectors V and x_0 are changing. Therefore, G is composed of parts G_v, G_x and G_w, which are already constant vectors. The matrix H and vectors G_v, G_x and G_w are computed by this function block and are referenced by inputs `uH`, `uGv`, `uGx` and `uGw` which must be preallocated. The scalar constant h is the function block parameter.

Inputs

`np`	Prediction horizon	↓1 ↑1000000	`Long (I32)`
`nc`	Control horizon	↓1 ↑1000000	`Long (I32)`
`uA`	Input reference to system matrix A		`Reference`
`uBu`	Input reference to input matrix Bu of control vector u		`Reference`
`uBv`	Input reference to input matrix Bv of disturbance vector v		`Reference`
`uC`	Input reference to output matrix C		`Reference`
`uQ`	Input reference to symmetric matrix Q in cost function		`Reference`
`uW`	Input reference to vector W in cost function		`Reference`
`uR`	Input reference to symmetric matrix R in cost function		`Reference`
`ul`	Input reference to integer index vector l		`Reference`
`uH`	Input reference to Hessian matrix H		`Reference`
`uGx`	Input reference to part of gradient vector G corresponding to state vector x		`Reference`
`uGv`	Input reference to part of gradient vector G corresponding to disturbance vector v		`Reference`
`uGw`	Intput reference to part of gradient vector G corresponding to vector W		`Reference`
`uSuL`	Input reference to work matrix Su*L		`Reference`
`uSv`	Input reference to work matrix Sv		`Reference`
`uT`	Input reference to work matrix T		`Reference`
`uScuL`	Input reference to work matrix Scu*L		`Reference`
`uScv`	Input reference to work matrix Scv		`Reference`
`uTc`	Input reference to work matrix Tc		`Reference`
`uWORK`	Input reference to matrix WORK		`Reference`
`HLD`	Hold		`Bool`

Outputs

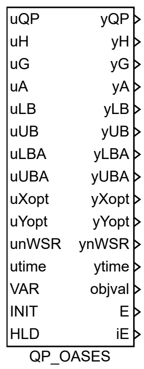
`yA`	Output reference to system matrix A	`Reference`

yBu	Output reference to input matrix Bu of control vector u	Reference
yBv	Output reference to input matrix Bv of disturbance vector v	Reference
yC	Output reference to output matrix C	Reference
yQ	Output reference to symmetric matrix Q in cost function	Reference
yW	Output reference to vector W in cost function	Reference
yR	Output reference to symmetric matrix R in cost function	Reference
yl	Output reference to integer index vector l	Reference
yH	Output reference to Hessian matrix H	Reference
yGx	Output reference to part of gradient vector G corresponding to state vector x	Reference
yGv	Output reference to part of gradient vector G corresponding to disturbance vector v	Reference
yGw	Output reference to part of gradient vector G corresponding to vector W	Reference
ySuL	Output reference to work matrix Su*L	Reference
ySv	Output reference to work matrix Sv	Reference
yT	Output reference to work matrix T	Reference
yScuL	Output reference to work matrix Scu*L	Reference
yScv	Output reference to work matrix Scv	Reference
yTc	Output reference to work matrix Tc	Reference
yWORK	Output reference to matrix WORK	Reference
E	Error indicator	Bool

QP_OASES – Quadratic programming using active set method

Block Symbol

Licence: ADVANCED



Function Description

The **QP_OASES** block solves a quadratic programming problem using active set method [10]

$$\begin{aligned} \min_{x \in \mathbb{R}^{\mathrm{nV}}} \quad & \tfrac{1}{2} x^T H x + x^T G \,, \\ \text{s. t.} \quad & \mathrm{lb}A \leq Ax \leq \mathrm{ub}A \,, \\ & \mathrm{lb} \leq x \leq \mathrm{ub} \,, \end{aligned}$$

where nV is number of variables, nC is number of constraints, the Hessian matrix $H \in \mathbb{R}^{\mathrm{nV} \times \mathrm{nV}}$ is symmetric and positive (semi-)definite, the gradient vector $G \in \mathbb{R}^{\mathrm{nV}}$, the constraint matrix $A \in \mathbb{R}^{\mathrm{nC} \times \mathrm{nV}}$, bound vectors $\mathrm{lb}, \mathrm{ub} \in \mathbb{R}^{\mathrm{nV}}$ and constraint vectors $\mathrm{lb}A, \mathrm{ub}A \in \mathrm{R}^{\mathrm{nC}}$.

The block wraps the **qpOASES** library[1], the use of which is described in the manual [11].

The output references **yH**, **yG**, **yA**, **yLB**, **yUB**, **yLBA**, **yUBA**, **yXopt** and **yYopt** are always set to the corresponding input **uH**, **uG**, **uA**, **uLB**, **uUB**, **uLBA**, **uUBA**, **uXopt** and **uYopt**. If the input **uQP** is not connected, the particular quadratic problem (QP) is allocated in the first execution of the function block (see below) and the output **yQP** is set to the reference of the allocated QP. If **uQP** is connected (to the **yQP** output of the previous **QP_OASES** block), the **yQP** output is set to **uQP** and the block works with an already allocated QP.

The block uses internal variables **nV** and **nC**. The value of **nV** is set to the number of rows of the vector G referenced by **uG**, the value of **nV** is set to the number of rows of

[1] **qpOASES** is distributed under the GNU Lesser General Public License, see Appendix A of [11].

the matrix A referenced by `uA`. If the reference `uA` is not defined (the matrix A is not connected), the value `nC = 0`.

To solve the QP problem, a `QProblem` object is created in the generic case (see Chapter 3 of [11]). However, the block can also solve the following special cases depending on the input references and the `hessianType` parameter:

`uH` not connected. In this case, it is assumed that Hessian matrix has a trivial value of the identity or zero matrix. The `hessianType` parameter must be equal to `HST_ZERO` or `HST_IDENTITY`, see Section 4.5 of the manual [11].

`uA` not connected. In this case, the constraint matrix A is not used (`nC = 0`, the `QProblemB` object is created, see Section 4.3 of the manual [11]. The `hessianType` parameter can be any allowed value.

`VAR = on`. If the input `VAR = on` during the first time the block is executed, an object of class `SQProblem` is created, see Section 4.2 of the manual [11]. In this case, all input matrices and vectors can change in each execution step in which `VAR = on`.

To obtain the solution of the QP problem, at least one of the input references `uXopt` and `uYopt` must be defined (connected to a vector). If connected to `uXopt`, the `yXopt` output will refer to the primal solution Xopt $\in \mathbb{R}^{\mathrm{nV}}$, if connected to `uYopt`, the `yYopt` output will refer to the dual solution Yopt $\in \mathbb{R}^{\mathrm{nV+nC}}$ of the QP problem. If both inputs are connected, both solutions will be obtained in each step. The optimal objective function value is indicated on the output `objval`.

The integer input `unWSR` specifies the maximum number of working set recalculations to be performed during the initial homotopy, see Section 3.2 of the manual [11]. Output `ynWSR` contains the number of working set recalculations actually performed. If the double input `utime` is connected and has a positive value, it contains the maximum allowed CPU time in seconds for the whole initialisation. The actually required CPU time for the initialization is indicated on the output `ytime`.

At least one vector must be connected from the `uXopt` and `uYopt` pair must be connected to obtain the solution of the QP problem. If uXopt is connected, the yXopt output will refer to the primary Xopt solution, if uYopt is connected, the yYopt output will refer to the dual Yopt solution of the QP task. If both inputs are connected, both solutions will be obtained in each step.

If the input `INIT = on` then the particular allocated QP problem is re-initialized. The `INIT` should be `on` for only a single period (edge) because no solution is computed during the QP initialisation. If `HLD = on` then nothing is computed.
The error flag `E` is set to `on` and the error code `iE` is set to zero if:

- the reference `uG` or `uLB` or `uUB` is not defined (i.e. input `uG` or `uLB` or `uUB` is not connected),
- the reference `uA` is defined and `uLBA` or `uUBA` is not defined (i.e. input `uA` is connected and `uLBA` or `uUBA` is not connected),

- both references uXopt and uYopt are not defined (i.e. neither of the inputs uXopt and uYopt is connected),
- the Hessian matrix H referenced by uH has a different number of rows and columns than nV,
- the number of rows of vectors referenced by uLB and uUB is not equal to nV (or the number of their columns is not equal to 1),
- the number of rows of vectors referenced by uLBA and uUBA is not equal to nC (or the number of their columns is not equal to 1) if the matrix A referenced by uA is connected,
- the number of rows of the vector referenced by uXopt is not equal to nV or the number of rows of the vector referenced by yOpt is not equal to nV+nC (or the number of their columns is not equal to 1),
- the internal space for transposed copies of matrices H or A is too small.

If the flag E is set to on and the error code iE is not zero then iE indicates the qpOASES error code, see the include file MessageHandling.hpp from qpOASES library.

Inputs

uQP	Input reference to quadratic programming problem	Reference
uH	Input reference to Hessian matrix H	Reference
uG	Input reference to gradient vector G	Reference
uA	Input reference to constraint matrix A	Reference
uLB	Input reference to lower bound vector LB	Reference
uUB	Input reference to upper bound vector LB	Reference
uLBA	Input reference to lower constraints' bound vector LB	Reference
uUBA	Input reference to upper constraints' bound vector LB	Reference
uXopt	Input reference to primal optimal solution	Reference
uYopt	Input reference to dual optimal solution	Reference
unWSR	Maximum number of initial working set recalculations	Long (I32)
utime	Maximum allowed CPU time in seconds for the whole initialisation	Double (F64)
VAR	Indiates that matrices H and A are time varying	Bool
INIT	Calls init() function instead of hotstart() in each block execution	Bool
HLD	If HLD = on then nothing is computed	Bool

Parameters

nVmax	Maximum number of optimization variables nV	Long (I32)
nCmax	Maximum number of optimization constraints nC	Long (I32)

hessianType	Hessian matrix type	Long (I32)
printLevel	Print level	Long (I32)
enableRamping	Enable ramping	Bool
enableFarBounds	Enable use of far bounds	Bool
enableFlippingBounds	Enable use of flipping bounds	Bool
enableRegularisation	Enable regularisation of semidefinite Hessian matrix	Bool
enableFullLITests	Enable use of condition-hardened linear independence tests	Bool
enableNZCTests	Enable nonzero curvature tests	Bool
enableDriftCorrection	Frequency of drift corrections (0 = off)	Long (I32)
enableCholeskyRefact	Frequency of full refactorisation of projected Hessian (0 = off)	Long (I32)
enableEqualities	Equalities shall be always treated as active constraints	Bool
terminationTolerance	Termination tolerance	Double (F64)
boundTolerance	If upper and lower limits differ less than this tolerance, they are regarded equal, i.e. as equality constraint	Double (F64)
boundRelaxation	Initial relaxation of bounds to start homotopy and initial value for far bounds.	Double (F64)
epsNum	Numerator tolerance for ratio tests	Double (F64)
epsDen	Denominator tolerance for ratio tests	Double (F64)
maxPrimalJump	Maximum allowed jump in primal variables in nonzero curvature tests	Double (F64)
maxDualJump	Maximum allowed jump in dual variables in linear independence tests	Double (F64)
initialRamping	Start value for ramping strategy	Double (F64)
finalRamping	Final value for ramping strategy	Double (F64)
initialFarBounds	Initial size of Far Bounds	Double (F64)
growFarBounds	Factor to grow Far Bounds	Double (F64)
initialStatusBounds	Initial status of bounds at first iteration	Long (I32)
epsFlipping	Tolerance of squared entry of Cholesky diagonal which triggers flipping bounds	Double (F64)
numRegularisationSteps	Maximum number of successive regularisation steps	Long (I32)
epsRegularisation	Scaling factor of identity matrix used for Hessian regularisation	Double (F64)
numRefinementSteps	Maximum number of iterative refinement steps	Long (I32)
epsIterRef	Early termination tolerance for iterative refinement	Double (F64)
epsLITests	Tolerance for linear independence tests	Double (F64)
epsNZCTests	Tolerance for nonzero curvature tests	Double (F64)

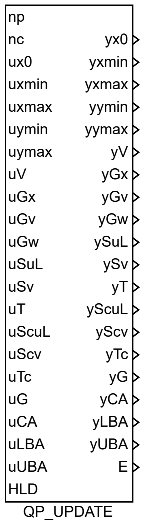
Outputs

yQP	Output reference to quadratic programming problem	Reference
yH	Output reference to Hessian matrix H	Reference
yG	Output reference to gradient vector G	Reference
yA	Output reference to constraint matrix A	Reference
yLB	Output reference to lower bound vector LB	Reference
yUB	Output reference to upper bound vector LB	Reference
yLBA	Output reference to lower constraints'bound vector LB	Reference
yUBA	Output reference to upper constraints' bound vector LB	Reference
yXopt	Output reference to primal optimal solution	Reference
yYopt	Output reference to dual optimal solution	Reference
ynWSR	Number of performed initial working set recalculations	Long (I32)
ytime	Elapsed CPU time in seconds for the whole initialisation	Double (F64)
objval	Optimal objective function value	Double (F64)
E	Error indicator	Bool
iE	Error code	Long (I32)

QP_UPDATE – Update matrices/vectors of quadratic programming

Block Symbol

Licence: ADVANCED



Function Description

The QP_UPDATE function block cooperates with the QP_MPC2QP block which converts the MPC problem described by equations (15.1)–(15.4) with prediction horizon n_p and control horizon n_c (inputs np and nc), to quadratic programming and pre-computes the Hessian matrix H, parts of the gradient vector G_x, G_v, G_w, matrices determining state constraints S_uL, S_v, T, and matrices determining output constraints $S_{cu}L$, S_{cv}, T_c. Besides the constant Hessian matrix H, the other vectors and matrices are connected to input references uGx, uGv, uGw, uSuL, uSv, uT, uScuL, uScv and uTc.

This block updates the QP problem for the given time instant with current values of state vector initial condition x_0, state vector bounds $x_{\min}$ and $x_{\max}$, output vector bounds $y_{\min}$ and $y_{\max}$, vector V (see eq. (15.5)) of disturbance prediction vectors $v_k, k = 0, \ldots, n_p - 1$. These vectors are referenced by inputs ux0, uxmin, uxmax, uymin, uymax and uV.

First, the gradient vector G referenced by the input uG is updated according to the equation (15.15):

$$G = G_x x_0 + G_v V + G_w.$$

The state constraints (15.3) can be rewritten using (15.9) for the prediction horizon n_p to

$$X_{\min} - S_v V - T x_0 \leq S_u L U_{n_c} \leq X_{\max} - S_v V - T x_0,$$

where $X_{\min}$ resp. $X_{\max}$ is a vector composed of n_p copies of the $x_{\min}$ resp. $x_{\max}$ vector. Similarly, the output constraints (15.4) can be rewritten using (15.10) to

$$Y_{\min} - S_{cv}V - T_c x_0 \leq S_{cu}LU_{n_c} \leq Y_{\max} - S_{cv}V - T_c x_0, .$$

where $Y_{\min}$ resp. $Y_{\max}$ is a vector composed of n_p copies of the $y_{\min}$ resp. $y_{\max}$ vector. The more compact form of these two equations is

$$\underbrace{\begin{bmatrix} X_{\min} - S_v V - T x_0 \\ Y_{\min} - S_{cv} V - T_c x_0 \end{bmatrix}}_{\mathrm{lb}A} \leq \underbrace{\begin{bmatrix} S_u L \\ S_{cu} L \end{bmatrix}}_{\mathrm{C}A} U_{n_c} \leq \underbrace{\begin{bmatrix} X_{\max} - S_v V - T x_0 \\ Y_{\max} - S_{cv} V - T_c x_0 \end{bmatrix}}_{\mathrm{ub}A}, \tag{15.16}$$

where the matrix CA and vectors lb$A$, ub$A$ are computed by this block, must be allocated e.g. by the `MX_MAT` blocks and references to the preallocated matrices must be connected to the block inputs `uCA`, `uLBA` and `uUBA`.

The last equation 15.16 is a general form of QP constraints. It covers both equality or inequality constraints for states and outputs.

If no state constraints are required, leave the `uxmin`, `uxmax`, `uSuL`, `uSv` and `uT` inputs disconnected. Then equation 15.16 gets the form

$$\underbrace{\begin{bmatrix} X_{\min} - S_v V - T x_0 \end{bmatrix}}_{\mathrm{lb}A} \leq \underbrace{\begin{bmatrix} S_u L \end{bmatrix}}_{\mathrm{C}A} U_{n_c} \leq \underbrace{\begin{bmatrix} X_{\max} - S_v V - T x_0 \end{bmatrix}}_{\mathrm{ub}A}. \tag{15.17}$$

Similarly, if no output constraints are required, leave the `uymin`, `uymax`, `uScuL`, `uScv` and `uTc` inputs disconnected. Then equation 15.16 gets the form

$$\underbrace{\begin{bmatrix} Y_{\min} - S_{cv} V - T_c x_0 \end{bmatrix}}_{\mathrm{lb}A} \leq \underbrace{\begin{bmatrix} S_{cu} L \end{bmatrix}}_{\mathrm{C}A} U_{n_c} \leq \underbrace{\begin{bmatrix} Y_{\max} - S_{cv} V - T_c x_0 \end{bmatrix}}_{\mathrm{ub}A}. \tag{15.18}$$

The output references `yx0`, `yxmin`, `yxmax`, `yymin`, `yymax`,`yV`, `yGx`, `yGv`, `yGw`, `ySuL`, `ySv`, `yT`, `yScuL`, `yScv`, `yTc`, `yG`, `yCA`, `yLBA` and `yUBA` are always set to the corresponding input `ux0`, `uxmin`, `uxmax`, `uymin`, `uymax`,`uV`, `uGx`, `uGv`, `uGw`, `uSuL`, `uSv`, `uT`, `uScuL`, `uScv`, `uTc`, `uG`, `uCA`, `uLBA` and `yUBA`.

If `HLD = on` then nothing is computed.

The error flag `E` is set to `on` if:

- the prediction horizon `np < 1` or control horizon `nc < 1`, or `nc > np`,
- the reference `ux0` is not defined or the element type of the array it references is not `Double (F64)`,
- the internal variable `bStateConstr = on` and at least one of the references `uxmin`, `uxmax` is not defined, or the element type of at least one of the arrays they reference is not `Double (F64)`,

- the internal variable bOutputConstr = on and the reference uymin is defined and the element type of the array it references is not Double (F64),
- the internal variable bOutputConstr = on and the reference uymax is defined and the element type of the array it references is not Double (F64),
- the reference uV is defined and the element type of the array it references is not Double (F64),
- the reference uG is defined and at least one of the references uGx, uGv, uSuL, uSv or uT is not defined,
- the reference uG is defined and the element type of the array it references is not Double (F64), or the reference uGx is defined and the element type of the array it references is not Double (F64), or the reference uGv is defined and the element type of the array it references is not Double (F64), or the reference uGw is defined and the element type of the array it references is not Double (F64),
- the reference uSuL is defined and the element type of the array it references is not Double (F64), or the reference uSv is defined and the element type of the array it references is not Double (F64), or the reference uT is defined and the element type of the array it references is not Double (F64),
- the reference uScuL is defined and the element type of the array it references is not Double (F64), or the reference uScv is defined and the element type of the array it references is not Double (F64), or the reference uTc is defined and the element type of the array it references is not Double (F64),
- the reference uCA or uLBA or uUBA or the element type of at least one of the arrays they reference is not Double (F64),
- the arrays referenced by defined references are too small or have incompatible dimensions.

If E = on, see the system log for details.

Inputs

np	Prediction horizon	↓1 ↑1000000	Long (I32)
nc	Control horizon	↓1 ↑1000000	Long (I32)
ux0	Input reference to initial condition vector x0 of the state vector x		Reference
uxmin	Input reference to vector of low limits of the state vector elements		Reference
uxmax	Input reference to vector of high limits of the state vector elements		Reference
uymin	Input reference to vector of low limits of the output inequalities		Reference

uymax	Input reference to vector of high limits of the output inequalities	Reference
uV	Input reference to vector predicted disturbancies	Reference
uGx	Input reference to part of gradient vector G corresponding to state vector x	Reference
uGv	Input reference to part of gradient vector G corresponding to disturbance vector v	Reference
uGw	Intput reference to part of gradient vector G corresponding to vector W	Reference
uSuL	Input reference to work matrix Su*L	Reference
uSv	Input reference to work matrix Sv	Reference
uT	Input reference to work matrix T	Reference
uScuL	Input reference to work matrix Scu*L	Reference
uScv	Input reference to work matrix Scv	Reference
uTc	Input reference to work matrix Tc	Reference
uG	Input reference to gradient vector G	Reference
uCA	Input reference to QP constraints matrix CA	Reference
uLBA	Input reference to lower constraints' bound vector LB	Reference
uUBA	Input reference to upper constraints' bound vector LB	Reference
HLD	Hold	Bool

Outputs

yx0	Output reference to initial condition vector x0 of the state vector x	Reference
yxmin	Output reference to vector of low limits of the state vector elements	Reference
yxmax	Output reference to vector of high limits of the state vector elements	Reference
yymin	Output reference to vector of low limits of the output inequalities	Reference
yymax	Output reference to vector of high limits of the output inequalities	Reference
yV	Output reference to vector predicted disturbancies	Reference
yGx	Output reference to part of gradient vector G corresponding to state vector x	Reference
yGv	Output reference to part of gradient vector G corresponding to disturbance vector v	Reference
yGw	Output reference to part of gradient vector G corresponding to vector W	Reference
ySuL	Output reference to work matrix Su*L	Reference
ySv	Output reference to work matrix Sv	Reference
yT	Output reference to work matrix T	Reference
yScuL	Output reference to work matrix Scu*L	Reference
yScv	Output reference to work matrix Scv	Reference

yTc	Output reference to work matrix Tc	Reference
yG	Output reference to gradient vector G	Reference
yCA	Output reference to QP constraints matrix CA	Reference
yLBA	Output reference to lower constraints' bound vector LB	Reference
yUBA	Output reference to upper constraints' bound vector LB	Reference
E	Error indicator	Bool

Chapter 16

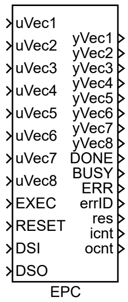
SPEC – Special blocks

Contents

EPC – External program call

Block Symbol

Licence: ADVANCED



Function Description

The EPC block executes an external program upon a rising edge (off→on) occurring at the EXEC input. The name and options of the program are defined by the cmd parameter. The format is the same as if the program was executed from the command line of the operating system.

It is possible to pass data from the REXYGEN system to the external program via files. The formatting of the files is defined by the format parameter. All the currently supported formats are textual and simple, which allows straightforward processing of the data in arbitrary program. Use e.g.
values=load('-ASCII', 'epc_inputVec1');
for loading the data in MATLAB or
values=read('epc_inputVec1',-1,32);
in SCILAB. The filename and number of columns must be adjusted for the given project. Data exchange in the opposite direction is naturally also supported, the REXYGEN system can read the files in the same format.

The block works in two modes. In *basic mode*, the rising edge on the EXEC input triggers reading the data on inputs and storing them in the ifns file. The values of the i-th input vector uVec<i> are stored in the i-th file from the ifns list. In *sampling mode*, the data from the input vectors are stored in each period of the control algorithm. In both cases the values from one time instant form one line in the file.

Analogically, the data from output files are copied to the outputs of the block (one line from the i-th file in the ofns list to the i-th output vector yVec<i>).

The inputs working in the *sampling mode* are defined by the sl list (comma-separated numbers). The outputs work always in the *sampling mode* – the last values are kept when the end of file is reached. The copying of data to input files can be blocked by the DSI input, the same holds for output data and the DSO input.

Use the RTOV block to combine individual signals into a vector one for the uVec input.

The RTOV blocks can be chained, therefore it is possible to create a vector of arbitrary dimension. Similarly, use the VTOR block to demultiplex a vector signal to individual signals.

Inputs

uVec1..uVec8	Input vector signal	Reference
EXEC	External program is called on rising edge	Bool
RESET	Block reset (deletes the input and output files and terminates the external program)	Bool
DSI	Disable inputs sampling	Bool
DSO	Disable outputs sampling	Bool

Outputs

yVec1..yVec8	Output vector signal	Reference
DONE	External program finished	Bool
BUSY	External program running	Bool
ERR	Error flag	Bool
errID	Error code i ..... REXYGEN general error	Error
res	External program return code	Long (I32)
icnt	Current input sample	Long (I32)
ocnt	Current output sample	Long (I32)

Parameters

cmd	Operating system command to execute	String
ifns	Input filenames (separated by semicolon) ⊙epc_uVec1;epc_uVec2	String
ofns	Output filenames (separated by semicolon) ⊙epc_yVec1;epc_yVec2	String
sl	List of inputs working in the *sampling mode*. The format of the list is e.g. 1,3..5,8. Third-party programs (Simulink, OPC clients etc.) work with an integer number, which is a binary mask, i.e. 157 (binary 10011101) in the mentioned case. ↓0 ↑255 ⊙85	Long (I32)
ifm	Maximum number of input samples ⊙10000	Long (I32)
format	Format of input and output files ⊙1 1 ..... Space-delimited values 2 ..... CSV (decimal point and commas) 3 ..... CSV (decimal comma and semicolons)	Long (I32)
nmax	Maximum output vectors length ↓2 ↑1000000 ⊙100	Long (I32)

Notes

- The called external program has the same priority as the calling task. This priority is high, in some cases higher than operating-system-kernel tasks. On Linux based systems, it is possible to lower the priority by using the `chrt` command:
  `chrt -o 0 extprg.sh`,
  where `extprg.sh` is the original external program.

- The size of signals is limited by parameter `nmax`. Bigger parameter means bigger memory consumption, so choose this parameter as small as possible.

- The filenames must respect the naming conventions of the target platform operating system. It is recommended to use only alphanumeric characters and an underscore to avoid problems. Also respect the capitalization, e.g. Linux is case-sensitive.

- The block also creates copies of the `ifns` and `ofns` files for implementation reasons. The names of these files are extended by the underscore character.

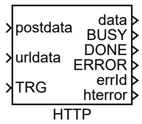
- The `ifns` and `ofns` paths are relative to the folder where the archives of the REXYGEN system are stored. It is recommended to define a symbolic link to a RAM-drive inside this folder for improved performance. On the other hand, for long series of data it is better to store the data on a permanent storage medium because the data can be appended e.g. after a power-failure recovery.

- The `OSCALL` block can be used for execution of some operating system functions.

HTTP – HTTP GET or POST request (obsolete)

Block Symbol

Licence: ADVANCED



Function Description

The HTTP block performs a single HTTP GET or POST request. Target address (URL) is defined by url parameter and urldata input. A final URL is formed in the way so that urldata input is simply added to url parameter.

HTTP request is started by the TRG parameter. Then the BUSY output is set until a request is finished, which is signaled by the DONE output. In case of an error, the ERROR output is set. The errId output carries last error identified by REXYGEN system error code. The hterror carries a HTTP status code. All data sent back by server to client is stored in the data output.

The block may be run in blocking or non-blocking mode which is specified by the BLOCKING parameter. In blocking mode, execution of a task is suspended until a request is finished. In non-blocking mode, the block performs only single operation depending on available data and execution of a task is not blocked. It is advised to always run HTTP block in non-blocking mode. It is however necessary to mention that on various operating systems some operations can not be performed in the non-blocking mode, so be careful and do not use this block in quick tasks or in tasks with short execution period. The non-blocking operation is best supported on GNU/Linux operating system. The maximal duration of a request performed by the HTTP block is specified by the timeout parameter.

The block supports user authentication using basic HTTP authentication method. User name and password may be specified by user and password parameters. The block also supports secure HTTP (HTTPS). It is also possible to let the block verify server's certificate by setting the VERIFY parameter. SSL certificate of a server or server's trusted certificate authority must be stored in the certificate parameter in a PEM format. The block does not support any certificate storage.

Parameters postmime and acceptmime specify MIME encoding of data being sent to server or expected encoding of a HTTP response.

Parameters nmax, postmax, and datamax specify maximum sizes of buffers allocated by the block. The nmax parameter is maximal size of any string parameter. The postmax parameter specifies a maximal size of postdata. The datamax parameter specifies a maximal size of data.

Inputs

`postdata`	Data to put in HTTP POST request		`String`
`urldata`	Data to append to URL address		`String`
`TRG`	Trigger of the selected action		`Bool`

Parameters

`url`	URL address to send the HTTP request to		`String`
`method`	HTTP request type 1 ..... GET 2 ..... POST	⊙1	`Long (I32)`
`user`	User name		`String`
`password`	Password		`String`
`certificate`	Authentication certificate		`String`
`VERIFY`	Enable server verification (valid certificate)		`Bool`
`postmime`	MIME encoding for POST request	⊙`application/json`	`String`
`acceptmime`	MIME encoding of HTTP response	⊙`application/json`	`String`
`timeout`	Timeout interval	⊙5.0	`Double (F64)`
`BLOCKING`	Wait for the operation to finish		`Bool`
`nmax`	Allocated size of string	↓0 ↑65520	`Long (I32)`
`postmax`	Allocated memory for POST request data	↓128 ↑65520 ⊙256	`Long (I32)`
`datamax`	Allocated memory for HTTP response	↓128 ↑10000000 ⊙1024	`Long (I32)`

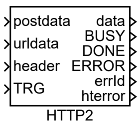
Outputs

`data`	Response data		`String`
`BUSY`	Sending HTTP request		`Bool`
`DONE`	HTTP request processed		`Bool`
`ERROR`	Error indicator		`Bool`
`errId`	Error code		`Error`
`hterror`	HTTP response		`Long (I32)`

HTTP2 – Block for generating HTTP GET or POST requests

Block Symbol

Licence: ADVANCED



Function Description

The HTTP block performs a single shot HTTP request. Target address (URL) is defined by url parameter and urldata input. A final URL is formed in the way so that urldata input is appended to the url parameter. The header input can be used for declaration of additional header fields.

A HTTP request is started by the TRG input. Then the BUSY output is set until the request is finished, which is signaled by a pulse at the DONE output. In case of an error, the pulse is generated at the ERROR output. The errId output carries information about the last error identified by REXYGEN system error code. The hterror carries a HTTP status code. All data received from server is published via the data output. All error outputs are reset when a new HTTP request is triggered by the TRG input.

The block may be run in blocking or non-blocking mode which is specified by the BLOCKING parameter. In blocking mode, execution of a task is suspended until the request is finished. In non-blocking mode, the block performs only single operation depending on available data and execution of a task is not blocked. It is advised to always run HTTP block in non-blocking mode. It is however necessary to mention that on various operating systems some operations cannot be performed in the non-blocking mode, so be careful and do not use this block in quick tasks (QTASK) or in tasks with short execution period. The non-blocking operation is best supported on GNU/Linux operating system. The maximal duration of a request performed by the HTTP block is specified by the timeout parameter.

The block supports user authentication using basic HTTP authentication method. User name and password may be specified by user and password parameters. The block also supports secure HTTP (HTTPS). It is also possible to let the block verify server's certificate by setting the VERIFY parameter. SSL certificate of a server or server's trusted certificate authority must be stored in the certificate parameter in a PEM format. The block does not support any certificate storage.

Parameters postmime and acceptmime specify MIME encoding of data being sent to server and expected encoding of the HTTP response.

Parameters nmax, postmax, and datamax specify maximum sizes of buffers allocated by the block. The nmax parameter is maximal size of any string parameter. The postmax parameter specifies a maximal size of postdata. The datamax parameter specifies a maximal size of `data`.

Inputs

`postdata`	Data to put in HTTP POST request		`String`
`urldata`	Data to append to URL address		`String`
`header`	Additional header fields		`String`
`TRG`	Trigger of the selected action		`Bool`

Parameters

`url`	URL address to send the HTTP request to		`String`
`method`	HTTP request type 1 ..... GET 2 ..... POST 3 ..... PUT 4 ..... DELETE 5 ..... HEAD 6 ..... TRACE 7 ..... PATCH 8 ..... OPTIONS 9 ..... CONNECT	⊙1	`Long (I32)`
`user`	User name		`String`
`password`	Password		`String`
`certificate`	Authentication certificate		`String`
`VERIFY`	Enable server verification (valid certificate)		`Bool`
`postmime`	MIME encoding for POST request	⊙`application/json`	`String`
`acceptmime`	MIME encoding for GET request	⊙`application/json`	`String`
`timeout`	Timeout interval	⊙5.0	`Double (F64)`
`BLOCKING`	Wait for the operation to finish		`Bool`
`nmax`	Allocated size of string	↓0 ↑65520	`Long (I32)`
`postmax`	Allocated memory for POST request data	↓128 ↑65520 ⊙4096	`Long (I32)`
`datamax`	Allocated memory for HTTP response	↓128 ↑10000000 ⊙64000	`Long (I32)`

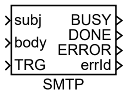
Outputs

`data`	Response data		`String`
`BUSY`	Sending HTTP request		`Bool`
`DONE`	HTTP request processed		`Bool`
`ERROR`	Error indicator		`Bool`
`errId`	Error code		`Error`
`hterror`	HTTP response		`Long (I32)`

SMTP – Send e-mail message via SMTP

Block Symbol

Licence: ADVANCED



Function Description

The SMTP block sends a single e-mail message via standard SMTP protocol. The block acts as a simple e-mail client. It does not implement a mail server.

The contents of a message is defined by the inputs subj and body. Parameters from and to specify sender and receiver of a message. A message is sent when the TRG parameter is set. Then the BUSY output is set until the request is finished, which is signaled by the DONE output. In case of an error, the ERROR output is set. The errId output carries the last error identified by REXYGEN system error code. The domain parameter must always be set to identify the target device. The default value should work in most cases. There can be multiple recipients of the message. In such a case, the individual e-mail addresses must be comma-separated and no space character may be present.

The block may be run in non-blocking or blocking mode, which is specified by the BLOCKING parameter.

- In the *blocking mode*, the execution of a task is suspended until the sending of e-mail is completed. This mode is typically used in tasks with long execution period, $T_S \geq 10s$. If the e-mail is not successfully sent until timeout expires, an error is indicated and the execution of the task is resumed.

- In the *non-blocking mode*, the SMTP block performs only a single operation in each execution of the block and the execution of a task is not suspended. This mode is typically used in tasks with short execution period, $T_S \leq 0.1s$. In this mode, the timeout parameter should be set to at least $50 \cdot T_S$, where T_S is the execution period in seconds.

It is recommended to run the SMTP block in the non-blocking mode. It is however necessary to mention that on various operating systems some operations may not be performed in the non-blocking mode, so be careful and do not use this block in quick tasks (see QTASK) or in tasks with extremely short execution period (few milliseconds). The non-blocking mode is best supported on GNU/Linux operating system.

The block supports user authentication using standard SMTP authentication method. User name and password may be specified by the user and password parameters. The block also supports secure connection. The encryption method is selected by the tls parameter. It is also possible to let the block verify server's certificate by setting the

`VERIFY` parameter. SSL certificate of a server or server's trusted certificate authority must be stored in the `certificate` parameter in a PEM format. The block does not support any certificate storage.

The length of the whole message (subject, body and headers) is limited to a maximum of 1024 characters.

Inputs

`subj`	Subject of the e-mail message		`String`
`body`	Body of the e-mail message		`String`
`TRG`	Trigger of the selected action		`Bool`

Parameters

`server`	SMTP server address		`String`
`to`	E-mail of the recipient		`String`
`from`	E-mail of the sender		`String`
`tls`	Encryption method `1` ..... None `2` ..... StartTLS `3` ..... TLS	⊙1	`Long (I32)`
`user`	User name		`String`
`password`	Password		`String`
`domain`	Domain name or identification of the target device		`String`
`auth`	Authentication method `1` ..... Login `2` ..... Plain	⊙1	`Long (I32)`
`certificate`	Authentication certificate		`String`
`VERIFY`	Enable server verification (valid certificate)		`Bool`
`timeout`	Timeout interval		`Double (F64)`
`BLOCKING`	Wait for the operation to finish		`Bool`

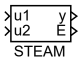
Outputs

`BUSY`	Sending e-mail		`Bool`
`DONE`	E-mail has been sent		`Bool`
`ERROR`	Error indicator		`Bool`
`errId`	Error code		`Error`

STEAM – Steam and water properties

Block Symbol

Licence: STANDARD



Function Description

The STEAM block calculates the thermodynamic properties of water and steam based on their other properties. The calculation is based on the IAPWS IF-97 standard (see http://www.iapws.org/relguide/IF97-Rev.pdf for more details). The units of temperature and pressure are defined by the parameter tunit and punit respectively; energy is in kilojoules (as it is used in IF-97 and by engineers), other properties use SI units. The function expressed by the block is defined by the func parameter. The parameter has the form <output property>_<1th input property><2nd input property> where properties is one of:

- T - Temperature
- p - Pressure
- h - Enthalpy [kJ/kg]
- v - Specific volume [m3/kg]
- rho - Density [kg/m3]
- s - Specific entropy
- u - Specific internal energy [kJ/kg]
- Cp - Specific isobaric heat capacity [kJ/kg/K]
- Cv - Specific isochoric heat capacity [kJ/kg/K]
- w - Speed of sound [m/s]
- my - Viscosity
- tc - Thermal Conductivity
- st - Surface Tension
- x - Vapour fraction
- vx - Vapour Volume Fraction

The output property can have attribute:

- sat - saturated value, i.e. for situation when water (liquid) is changed into steam (vapour)
- V - steam (vapour) for saturated conditions
- L - water (liquid) for saturated conditions

Examples:

- h_pT output is enthalpy for given pressure (1st input) and temperature (2nd input)
- Tsat_p saturated temperature (i.e. boiling temperature) for given pressure (1st input)
- hL_p enthalpy of (liquid) water for saturated conditions given by pressure (1st input)

Inputs

`u1`	1st input property	`Double (F64)`
`u2`	2nd input property (if required)	`Double (F64)`

Parameters

`func`	Calculated function	⊙1	`Long (I32)`
`punit`	Pressure unit 1 ..... MPa 2 ..... bar 3 ..... kPa	⊙1	`Long (I32)`
`tunit`	Temperature unit 1 ..... K 2 ..... °C	⊙1	`Long (I32)`

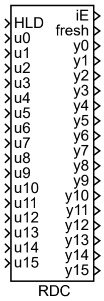
Outputs

`y`	Output property	`Double (F64)`
`E`	Error flag	`Bool`

RDC – Remote data connection

Block Symbol

Licence: ADVANCED



Function Description

The RDC block is a special input-output block. The values are transferred between two blocks on different computers, eventually two different Simulinks on the same computer or Simulink and the REXYGEN system on the same computer. In order to communicate, the two RDC blocks must have the same id number. The communication is based on UDP/IP protocol. This protocol is used as commonly as the more known TCP/IP, i.e. it works over all LAN networks and the Internet. The algorithm performs the following operations in each step:

- If HLD = on, the block execution is terminated.
- If the period parameter is a positive number, the difference between the system timer and the time of the last packet sending is evaluated. The block execution is stopped if the difference does not exceed the period parameter. If the period parameter is zero or negative, the time difference is not checked.
- A data packet is created. The packet includes block number, the so-called invoke number (serial number of the packet) and the values u0 to u15. All values are stored in the commonly used so-called network byte order, therefore the application is computer and/or processor independent.
- The packet is sent to the specified IP address and port.
- The invoke number is increased by 1.
- It is checked whether any incoming packets have been received.
- If so, the packet validity is checked (size, id number, invoke number).

- If the data is valid, all outputs `y0` to `y15` are set to the values contained in the packet received.
- The `fresh` output is updated. In case of error, the error code is displayed by the `err` output.

There are 16 values transmitted in each direction periodically between two blocks with the same `id` number. The `u(i)` input of the first block is transmitted the `y(i)` output of the other block. Unlike the TCP/IP protocol, the UDP/IP protocol does not have any mechanism for dealing with lost or duplicate packets, so it must be handled by the algorithm itself. The `invoke` number is used for this purpose. This state variable is increased by 1 each time a packet is sent. The block stores also the `invoke` number of the last received packet. It is possible to distinguish between various events by comparing these two invoke numbers. The packets with invoke numbers lower than the invoke number of the last received packet are denied unless the difference is grater than 10. This solves the situation when one of the `RDC` blocks is restarted and its `invoke` number is reset.

All `RDC` blocks in the same application must have the same `local port` number and the number of `RDC` blocks is limited to 64 for implementation reasons. If there are two applications using the `RDC` block running on the same machine, then each of them must use a different `local port` number.

Inputs

`HLD`	Input for disabling the execution of the block. No packets are received nor transmitted when `HLD` = `on`.	`Bool`
`u0..u15`	Values which are sent/written to the output values `y0` to `y15` of the paired block	`Double (F64)`

Outputs

`iE`	Displays the code of the last error. The error codes are listed below: `0` ..... No error	`Long (I32)`

	Persistent errors originating in the initialization phase (< 0). Cannot be fixed automatically. `-1` .... Maximum number of blocks exceeded (> 64) `-2` .... Local ports mismatch; the `lport` parameter must be the same for all `RDC` blocks within one application `-3` .... Error opening socket (the UDP/IP protocol is not available) `-4` .... Error assigning local port (port already occupied by another service or application) `-5` .... Error setting the so-called non-blocking socket mode (the `RDC` block requires this mode) `-10` ... Error initializing the socket library `-11` ... Error initializing the socket library `-12` ... Error initializing the socket library *Temporary errors originating in any cycle of the code (> 0). Can be fixed automatically.* `1` ..... Initialization successful, yet no data packet has been received `2` ..... Packet consistency error (incorrect length – transmission error or conflicting service/application is running) `4` ..... Error receiving packet (socket library error) `8` ..... Error sending packet (socket library error)	
`fresh`	Elapsed time (in seconds) since the last received packet. Can be used for detection of an error in the paired block.	`Double (F64)`
`y0..y15`	Values transmitted from the input ports `u0` to `u15` of the paired `RDC` block – data from the last packet received	`Double (F64)`

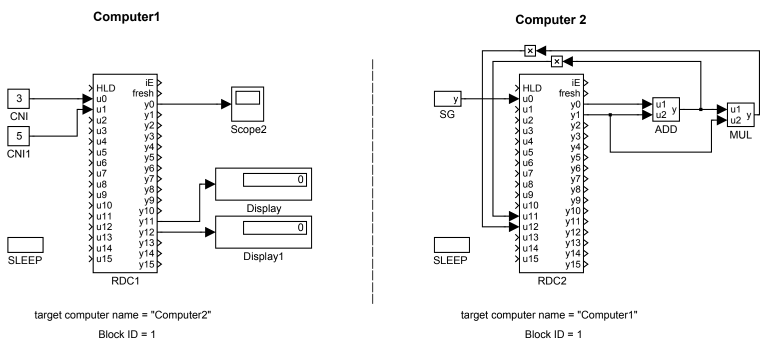
Parameters

`target`	Name or IP address running the paired `RDC` block. Broadcast address is allowed.	`String`
`rport`	Remote port – address of the UDP/IP protocol service, it is recommended to keep the default value unless necessary (service/application conflict) ⊙1288	`Word (U16)`
`lport`	Local port – similar meaning as the `rport` parameter; remote port applies to the receiving machine, local port applies to the machine sending the packet ⊙1288	`Word (U16)`
`id`	Block ID – this number is contained within the data packet in order to reach the proper target block (all blocks on the target receive the packet but only the one with the corresponding `id` decomposes it and uses the data contained to update its outputs) ↓1 ↑32767 ⊙1	`Long (I32)`

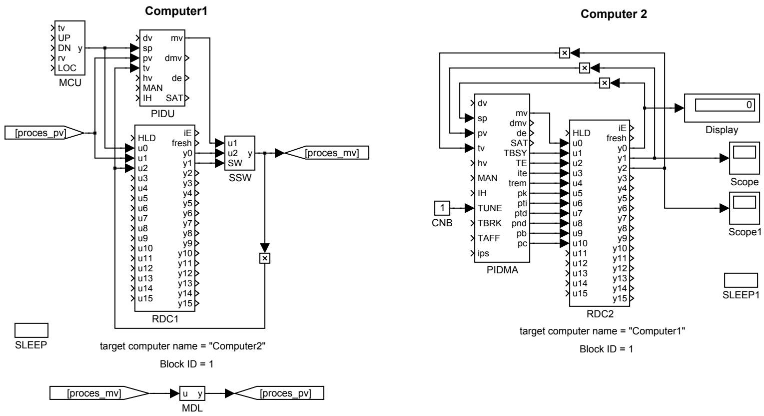
`period`	The shortest time interval between transmitting/receiving packets (in seconds). The packets are transmitted/received during each execution of the block for `period`≤0 while the positive values of this parameter are extremely useful when sending data out of the Simulink continuous models based on a `Variable step` solver.	`Double (F64)`

Example

The following example explains the function of the `RDC` block. The constants 3 and 5 are sent from `Computer1` to `Computer2`, where they appear at the `y0` and `y1` outputs of the `RDC2` block. The constants are then summed and multiplied and sent back to `Computer1` via the `u11` and `u12` outputs of the `RDC2` block. The displays connected to the `y11` and `y12` outputs of the `RDC1` block show the results of mathematical operations $3+5$ and $(3+5)*5$. The signal from the `SG` generator running on `Computer2` is transmitted to the `y0` output of the `RDC1` block, where it can be easily displayed. Note that `Display` and `Scope` are Matlab/Simulink blocks – to visualize data within the REXYGEN system, the `TRND` block or similar must be utilized.



The simplicity of the example is intentional. The goal is to demonstrate the functionality of the block, not the complexity of the system. In reality, the `RDC` block is used in more complex tasks, e.g. for remote tuning of the PID controller as shown below. The PID control algorithm is running on `Computer1` while the tuning algorithm is executed by `Computer2`. See the `PIDU`, `PIDMA` and `SSW` blocks for more details.



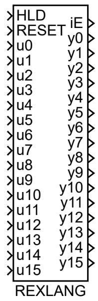
OPC server of the RDC block

There is also an OPC server embedded in the RDC block. Detailed description will be available soon.

REXLANG – User programmable block

Block Symbol

Licence: REXLANG



Function Description

The standard function blocks of the REXYGEN system cover the most typical needs in control applications. But there still exist situations where it is necessary (or more convenient) to implement an user-defined function. The `REXLANG` block covers this case. It implements an user-defined algorithm written in a scripting language very similar to the C language (or Java).

Scripting language

As mentioned, the scripting language is similar to the C language. Nevertheless, there are some differences and limitations:

- `Double`, `long` and `string` data types are supported (it is possible to use `int`, `short`, `bool` as well, but these are internally converted to `long`. The `float` type can be used but it is converted internally to `double`. The `typedef` type is not defined.
- Pointers and structures are not implemented. However, it is possible to define arrays and use the indexes (the `[ ]` operator). Block inputs, outputs and parameters can not be arrays.
- The '`,`' operator is not implemented.
- The preprocessor supports the following commands: `#include`, `#define`, `#ifdef` .. [`#else` ..] `#endif`, `#ifndef` .. [`#else` ..] `#endif` (i.e. `#pragma` and `#if` .. [`#else` ..] `#endif` are not supported).
- The standard ANSI C libraries are not implemented, however the majority of mathematic functions from `math.h` and some other functions are implemented (see the text below).

- The input, output and parameter keywords are defined for referencing the REXLANG block inputs, outputs and parameters. System functions for controlling the execution and diagnostics are implemented (see the text below).
- The main() function is executed periodically during runtime. Alongside the main() function the init() (executed once at startup), exit() (executed once when the control algorithm is stopped) and the parchange() (executed on parameters change in REXYGEN.
- The functions and procedures without parameters must be explicitly declared void.
- The identifiers cannot be overloaded, i.e. the keywords and built-in functions cannot share the name with an identifier. The local and global variables cannot share the same name.
- Array initializers are not supported. Neither in local arrays nor the global ones.
- User defined return values of main(), init() and exit() functions are written to iE output. Values < -99 will stop algorithm execution (reinicialization by RESET input needed for further algorithm run). Return values:

  iE >= 0 . . . No error occurred

  0 > iE >= -99 . . . Warning, no changes to function block algorithm execution

  iE < -99 . . . Error occured, function block algorithm execution stopped

Scripting language syntax

The scripting language syntax is based on the C language, but pointers are not supported and the data types are limited to long and double. Moreover the input, output and parameter keywords are defined for referencing the REXLANG block inputs, outputs and parameters. The syntax is as follows:

- <type> input(<input number>) <variable name>;
- <type> output(<outpt number>) <variable name>;
- <type> parameter(<parameter number>) <variable name>;

The input and parameter variables are read-only while the output variables are write-only. For example:

```
double input(1) input_signal; /* declaration of a variable of type
                                 double, which corresponds with the
                                 u1 input of the block */
long output(2) output_signal; /* declaration of a variable of type
                                 long, which corresponds with the y2
                                 output of the block */
```

```
input_signal = 3;               //not allowed, inputs are read-only
sum = output_signal + 1;        //not allowed, outputs are write-only
if (input_signal>1) output_signal = 3 + input_signal;  //correct
```

Available functions

The following functions are available in the scripting language:

- **Mathematic functions** (see ANSI C, `math.h`):
  `atan`, `sin`, `cos`, `exp`, `log`, `sqrt`, `tan`, `asin`, `acos`, `fabs`, `fmod`, `sinh`, `cosh`, `tanh`, `pow`, `atan2`, `ceil`, `floor` and `abs` Please note that the `abs` function works with integer numbers. All the other functions work with variables of type `double`.

- **Vector functions** (not part of ANSI C)

  `double max([n,]val1,...,valn)`
  : Returns the maximum value. The first parameter defining the number of items is optional.

  `double max(n,vec)`
  : Returns the value of maximal item in the `vec` vector.

  `double min([n,]val1,...,valn)`
  : Returns the minimum value. The first parameter defining the number of items is optional.

  `double min(n,vec)`
  : Returns the value of minimal item in the `vec` vector.

  `double poly([n,]x,an,...,a1,a0)`
  : Evaluates the polynomial $y = an*x^n + \ldots + a1*x + a0$. The first parameter defining the number of items is optional.

  `double poly(n,x,vec)`
  : Evaluates the polynomial $y = vec[n] * x^n + \ldots + vec[1] * x + vec[0]$.

  `double scal(n,vec1,vec2)`
  : Evaluates the scalar product $y = vec1[0] * vec2[0] + \ldots + vec1[n-1] * vec2[n-1]$.

  `double scal(n,vec1,vec2,skip1,skip2)`
  : Evaluates the scalar product $y = vec1[0] * vec2[0] + vec1[skip1] * vec2[skip2] + \ldots + vec1[(n-1)*skip1] * vec2[(n-1)*skip2]$. This is well suited for multiplication of matrices, which are stored as vectors (line by line or column by column).

  `double conv(n,vec1,vec2)`
  : Evaluates the convolutory product $y = vec1[0] * vec2[n-1] + vec1[1] * vec2[n-2] + \ldots + vec1[n-1] * vec2[0]$.

  `double sum(n,vec)`
  : Sums the items in a vector, i.e. $y = vec[0] + vec[1] + \ldots + vec[n-1]$.

`double sum([n,]val1,...,valn)`
: Sums the items, i.e. y = `val1` + `val2` + ... + `valn`. The first parameter defining the number of items is optional.

`[]array([n,]an-1,...,a1,a0)`
: Returns an array/vector with the given values. The first parameter defining the number of items is optional. The type of the returned value is chosen automatically to fit the type of parameters (all must be of the same type).

`[]subarray(idx,vec)`
: Returns a subarray/subvector of the `vec` array, starting at the `idx` index. The type of the returned value is chosen automatically according to the `vec` array.

`copyarray(count,vecSource,idxSource,vecTarget,idxTarget)`
: Copies `count` items of the `vecSource` array, starting at `idxSource` index, to the `vecTarget` array, starting at `idxTarget` index. Both arrays must be of the same type.

`void fillarray(vector, value, count)`
: Copies `value` to `count` items of the `vector` array (always starting from index 0).

- **String functions** (ANSI C contains analogous functions in the `string.h` file)

`string strsub(str,idx,len)`
: Returns a substring of length `len` starting at index `idx`.

`long strlen(str)`
: Returns string length (number of characters).

`long strfind(str,substr)`
: Returns the position of first occurrence of `substr` in `str`.

`long strrfind(str,substr)`
: Returns the position of last occurrence of `substr` in `str`.

`strreplace(str,pattern,substr)`
: Find all occurrences of `pattern` in `str` and replace it with `substr` (in-place replacement, so new string is stored into `str`).

`strupr(str)`
: Converts a string to uppercase.

`strlwr(str)`
: Converts a string to lowercase.

`strtrim(str)`
: Remove leading and trailing white characters (spaces) from a string.

`long str2long(str [, default])`
: Converts string to integer number. The first non-numerical character is considered the end of the input string and the remaining characters are ignored. If conversion failed, 2nd parameter is returned (or 0 if parameter is not set).

`double str2double(str [, default])`
: Converts string to a decimal number. The first non-numerical character is considered the end of the input string and the remaining characters are ignored. If conversion failed, 2nd parameter is returned (or 0 if parameter is not set).

`string long2str(num [, radix])`
: Converts an integer number `num` to text. The optional parameter `radix` specifies the numerical system in which the conversion is to be performed (typically 10 or 16). If `radix` is not specified, default value is `radix` = 10. The output string does not contain any identification of the numerical system used (e.g. the 0x prefix for the hexadecimal system).

`string double2str(num)`
: Converts a decimal number `num` to text.

`strcpy(dest,src)`
: Function copies the `src` string to the `dest` string. Implemented for compatibility with ANSI C. The construction `dest=src` yields the same result.

`strcat(dest,src)`
: Function appends a copy of the `src` string to the `dest` string. Implemented for compatibility with ANSI C. The construction `dest=dest+src` yields the same result.

`strcmp(str1,str2)`
: Function compares strings `str1` and `str2`. Implemented for compatibility with ANSI C. The construction `str1==str2` yields the same result.

`float2buf(buf,x[,endian])`
: Function for converting real number `x` into 4 elements of array `buf`. Each element represents an octet (byte) of number in single precision representation according to IEEE 754 (known as float). The function is useful for filling communication buffers. Optional 3rd parameter has the following meaning: 0 (default) = processor native endian, 1 = little endian, 2 = big endian.

`double2buf(buf,x[,endian])`
: Similar function to `float2buf`, but stores 8 elements in double precision format.

`double buf2float(buf[,endian])`
: Inverse function to `float2buf`

`double buf2double(buf[,endian])`
: Inverse function to `double2buf`

`long RegExp(str,regexp,capture[])`
: Compares the `str` string with regular expression `regexp`. When the string matches the pattern, the `capture` array contains individual sections of the regular expression. `capture[0]` is always the complete regular expression. The function return the number of captured strings or a negative value in case of an error. The regular expression may contain the following:

  `(?i)` ...Must be at the beginning of the regular expression. Makes the matching case-insensitive.

^ ... Match beginning of a string

$ ... Match end of a string

() ... Grouping and substring capturing

\s ... Match whitespace

\S ... Match non-whitespace

\d ... Match decimal digit

\n ... Match new line character

\r ... Match line feed character

\f ... Match vertical tab character

\v ... Match horizontal tab character

\t ... Match horizontal tab character

\b ... Match backspace character

+ ... Match one or more times (greedy)

+? ... Match one or more times (non-greedy)

* ... Match zero or more times (greedy)

*? ... Match zero or more times (non-greedy)

? ... Match zero or once (non-greedy)

x|y ... Match x or y (alternation operator)

\meta ... Match one of the meta characters: ^$().[]*+?|\

\xHH ... Match byte with hex value 0xHH, e.g. \x4a.

[...] ...] Match any character from the set. Ranges like [a-z are supported.

[^...] ... Match any character but the ones from the set.

Example:
`RegExp("48,string1,string2","^(\\d+),([^,]+),",capture);`
Result: `capture=["48,string1","48","string1"]`

`long ParseJson(json,cnt,names[],values[])`

The `json` string is supposed to contain text in JSON format. The `names` array contain property names of the requested objects (subitems are accessed via `.`, index of the array via `[]` - e.g. `"cars[1].model"`). The `values` array then contains values of the requested objects. The `cnt` parameter defines the number of requested objects (length of both the `names` and `values` arrays). The function returns the number of values, negative numbers indicate errors.

Note: String variable is declared just like in ANSI C, i.e. `char <variable name>[<maximum number of characters>];`. For passing the strings to functions use `char <variable name>[]` or `string <variable name>`.

- **System functions** (not part of ANSI C)

`Archive(arc, type, id, lvl_cnt, value)`

Stores a value into the archive subsystem. `arc` is a bit mask of archives to write the events to (e.g. for writing to archives 3,5 set `arc`=20 -> (BIN)10100 = (DEC)20). The archives are numbered from 1 and the maximum number of archives is limited to 15 (archive no. 0 is an internal system log). `type`

- `1` ... Bool
- `2` ... Byte (U8)
- `3` ... Short (I16)
- `4` ... Long (I32)
- `5` ... Word (U16)
- `6` ... DWord (U32)
- `7` ... Float (F32)
- `8` ... Double (F64)
- `9` ... Time
- `10` ... Large (I64)
- `11` ... Error
- `12` ... String
- `17` ... Bool Group
- `18` ... Byte Group (U8)
- `19` ... Short Group (I16)
- `20` ... Long Group (I32)
- `21` ... Word Group (U16)
- `22` ... DWord Group (U32)
- `23` ... Float Group (F32)
- `24` ... Double Group (F64)
- `25` ... Time Group
- `26` ... Large Group (I64)
- `27` ... Error Group

`id` is a unique archive item ID. `lvl_cnt` is Alarm level in case of alarms or number of elements in case of Group `type`. `value` is a value to be written or an array reference in case of Group `type`.

`Trace(id, val)`

Displays the `id` value and the `val` value. The function is intended for debugging. The `id` is a user-defined constant (from 0 to 9999) for easy identification of the displayed message. The `val` can be of any data type including text string. The output can be found in the system log of REXYGEN.

In order to view these debugging messages in System log it is necessary to enable them. Go to the menu

Target→Diagnostic messages and tick the *Information* checkbox in the *Function block messages* box. Logging has to be also enabled for the particular block by ticking the *Enable logging* checkbox in the *Runtime* tab of the block parameters dialog. By default, this is enabled after placing a new block from library. Only then are the messages displayed in the *System log*.

`TraceError(id, val) TraceWarning(id, val) TraceVerbose(id, val)`
: These commands are similar to the `Trace` command, only the output is routed to the corresponding logging group (Error, Warning, Verbose). Messages with the *Error* level are always written to the log. To view the *Warning* and *Verbose* messages, enable the corresponding message group. Go to the menu
Target→Diagnostic messages and tick the corresponding checkbox in the *Function block messages* box.

`Suspend(sec)`
: The script is suspended if its execution within the given sampling period takes more seconds than specified by the `sec` parameter. At the next start of the block the script continues from the point where it was suspended. Use `Suspend(0)` to suspend the code immediately.

`double GetPeriod()`
: Returns the sampling period of the block in seconds.

`double CurrentTime()`
: Returns the current time (in internal format). Intended for use with the `ElapsedTime()` function.

`double ElapsedTime(new_time, old_time)`
: Returns the elapsed time in seconds (decimal number), i.e. the difference between the two time values `new_time` and `old_time`. The `CurrentTime()` function is typically used in place of the `new_time` parameter.

`double Random()`
: Returns a pseudo-random number from the $\langle 0, 1)$ interval. The pseudo-random number generator is initialized prior to calling the `init()` function so the sequence is always the same.

`long QGet(var)`
: Returns the quality of the `var` variable (see the `QFC`, `QFD`, `VIN`, `VOUT` blocks). The function is intended for use with the inputs, outputs and parameters. It always returns 0 for internal variables.

`void QSet(var, value)`
: Sets the quality of the `var` variable (see the `QFC`, `QFD`, `VIN`, `VOUT` blocks). The function is intended for use with the inputs, outputs and parameters. It has no meaning for internal variables.

`long QPropag([n,]val1,...,valn)`
: Returns the quality resulting from merging of qualities of `val1,...,valn`. The basic rule for merging is that the resulting quality correspond with

the worst quality of `val1,...,valn`. To obtain the same behavior as in other blocks of the REXYGEN system, use this function to set the quality of output, use all the signals influencing the output as parameters.

`double LoadValue(fileid, idx)`
: Reads a value from a file. A binary file with `double` values or a text file with values on individual lines is supposed. The `idx` index (binary file) or line number (text file) starts at 0. The file is identified by `fileid`. At present the following values are supported:

  `0` . . . file on a disk identified by the `p0` parameter

  `1` . . . file on disk identified by name of the REXLANG block and extension `.dat`

  `2` . . . file on a disk identified by the `srcname` parameter, but the extension is changed to `.dat`

  `3` . . . `rexlang.dat` file in the current directory

  `4-7` . . . same like `0-3`, but format is text file. Each line contains one number. The index `idx` is the line number and starts at zero. Value `idx=-1` means next line (e.g. sequential writing).

`void SaveValue(fileid, idx, value)`
: Stores the `value` to a file. The meaning of parameters is the same as in the `LoadValue` function.

`void GetSystemTime(time)`
: Returns the system time. The time is usually returned as UTC but this can be altered by the operating system settings. The `time` parameter must be an array of at least 8 items of type `long`. The function fills the array with the following values in the given order: year, month, day (in the month), day of week, hours, minutes, seconds, milliseconds. On some platforms the milliseconds value has a limited precision or is not available at all (the function returns 0 ms).

`void Sleep(seconds)`
: Stop execution of the block's algorithm (and whole task) for defined time. Use this block with extreme caution and only if there is no other possibility to achieve the desired behaviour of your algorithm. The sleep interval should not exceed 900 milliseconds. The shortest interval is about 0.01s, the precise value depends on the target platform.

`long GetExtInt(ItemID)`
: Returns the value of input/output/parameter of arbitrary block in REXYGEN algorithm. Such an external data item is referenced by the `ItemID` parameter. The structure of the string parameter `ItemID` is the same as in e.g. the `sc` parameter of the GETPI function block. If the value cannot be obtained (e.g. invalid or non-existing `ItemID`, data type conflict, etc.), the `REXLANG` block issues an error and must be reset.

`long GetExtLong(ItemID)`
: See `GetExtInt(ItemID)`.

`double GetExtReal(ItemID)`
: Similar to `GetExtInt(ItemID)` but for decimal numbers.

`double GetExtDouble(ItemID)`
: See `GetExtReal(ItemID)`.

`string GetExtString(ItemID)`
: Similar to `GetExtInt(ItemID)` but for strings.

`void SetExt(ItemID, value)`
: Sets the input/output/parameter of arbitrary block in REXYGEN algorithm to `value`. Such an external data item is referenced by the `ItemID` parameter. The structure of the string parameter `ItemID` is the same as in e.g. the `sc` parameter of the `SETPI` function block. The type of the external data item (long/double/string) must correspond with the type of the `value` parameter. If the value cannot be set (e.g. invalid or non-existing `ItemID`, data type conflict, etc.), the `REXLANG` block issues an error and must be reset.

`int BrowseExt(ItemID, first_subitem_index, max_count, subitems, kinds)`
: Function browses task adress space. If `ItemID` is a block identifier (`block_path`), `subitems` string array will contain names of all inputs, outputs, parameters and internal states. Function returns number of subitems or negative error code. `kinds` values: executive = 0, module = 1, driver = 2, archive = 3, level = 4, task = 5, quicktask = 6, subsystem = 7, block = 8, input = 9, output = 10, internal state = 11, parameter or state array = 12, special = 13.

`long CallExt(ItemID)`
: Run (one step) arbitrary block in REXYGEN algorithm. Such an external block is referenced by the `ItemID` parameter. The structure of the string parameter `ItemID` is the same as in e.g. the `sc` parameter of the `GETPI` function block. The function returns result code of the calling block (see REXYGEN error codes). It is strongly recommended to call halted blocks only (set checkbox `Halt` on the property page `Runtime` in the parameters dialog of the block) and the block (or subsystem) should be in same task as the REXLANG block.

`long GetInArrRows(input)`
: Returns the number of rows of the array that is attached to the input with index `input` of the REXLANG block.

`long GetInArrCols(input)`
: Returns the number of columns of the array that is attached to the input with index `input` of the REXLANG block.

`long GetInArrMax(input)`
: Returns the maximum (allocated) size of the array that is attached to the input with index `input` of the REXLANG block.

`double GetInArrDouble(input, row, col)`
: Returns the member of the array that is attached to the input with index `input` of the REXLANG block.

`Void SetInArrValue(input, row, col, value)`
: Sets the member of the array that is attached to the input with index `input` of the REXLANG block.

`Void SetInArrDim(input, row, col)`
: Sets the dimension of the array that is attached to the input with index `input` of the REXLANG block.

`long memrd32(hMem, offset)`
: Reading physical memory. Get the handle by `Open(72,"/dev/mem",<physical address>,<area size>)`.

`long memwr32(hMem, offset, value)`
: Writing to physical memory. Get the handle by `OpenMemory("/dev/mem",<physical address>,<area size>)`.

- **Communication functions** (not part of ANSI C)

  This set of functions is intended for communication over TCP/IP, UDP/IP, serial line (RS-232 or RS-485), SPI bus and I2C bus. Only a brief list of available functions is given below, see the example projects of the REXYGEN system for more details.

`long OpenFile(string filename)`
: Function for opening a file. Identification number (the so-called handle) of the file is returned. If a negative value is returned, the opening was not successful.

`long OpenCom(string comname, long baudrate, long parity)`
: Function for opening a serial line. Starting from REXYGEN version 3.0, it is possible to enter virtual ports as `comname` as well. More information about virtual ports can be found in the description of the `UART` block. Identification number (the so-called handle) of serial port is returned. If a negative value is returned, the opening was not successful. Parity setting: 0=none, 1=odd, 2=even.

`long OpenUDP(string localname, long lclPort, string remotename, long remPort)`
: Function for opening a UDP socket. Identification number (the so-called handle) of the socket is returned. If a negative value is returned, the opening was not successful. Function open IPv4 or IPv6 socket according to `remotename` and `localname` or OS setting if DNS name is used. It is possible set empty locname (meaning any interface), empty remotename or 0 for remPort (meaning not used - e.g. write will not be called) or 0 for lclPort (meaning assign by UDP/IP stack).

`long OpenTCPsvr(string localname, long lclPort)`
: Function for opening a TCP socket (server, listening). Identification number (the so-called handle) of the socket is returned. If a negative value is returned, the opening was not successful. Function open IPv4 or IPv6 socket according to `remotename` and `localname` or OS setting if DNS name is used. It is possible set empty locname (meaning any interface).

`long OpenTCPcli(string remotename, long remPort)`
: Function for opening a TCP socket (client). Identification number (the so-called handle) of the socket is returned. If a negative value is returned, the opening was not successful. Function open IPv4 or IPv6 socket according to `remotename` and `localname` or OS setting if DNS name is used. WARNING: this function not wait on connection is established. It take few miliseconds on local network but can take few seconds for remote location. If the function `Write()` or `Read()` is used before connection is established, error code -307 (file open error) is returned.

`long OpenI2C(string devicename)`
: Function for opening the I2C bus. Identification number (the so-called handle) of the bus is returned. If a negative value is returned, the opening was not successful.

`long OpenSPI(string devicename)`
: Function for opening the SPI bus. Identification number (the so-called handle) of the bus is returned. If a negative value is returned, the opening was not successful.

`long OpenMemory(string devicename, long baseaddr, long size)`
: Function for mapping physical memory. Identification number (the so-called handle) of the memory address is returned. If a negative value is returned, the opening was not successful.

`long OpenDevice(string filename)`
: Same as `OpenFile()`, but `Write()` or `Read()` are not-blocking (e.g. if data are not readable/writeable, function return immediately with return code -1).

`long OpenSHM(string devicename, long deviceid, long size, long flags)`
: Function for mapping shared memory (linux only, call shmget()).

`void Close(long handle)`
: Closes the socket, serial line, file or any device opened by the `Open...` functions.

`void SetOptions(long handle, long params[])`
: Sets the parameters of a socket or serial line. The array size must be at least:

  – 22 for serial line (on Windows parameters for SetCommState() and SetCommTimeouts() in following order: BaudRate, fParity, Parity, StopBits, ByteSize, fDtrControl, fRtsControl, fAbortOnError, fBinary, fErrorChar, fNull, fDsrSensitivity, fInX, fOutX, fOutxCtsFlow, fOutxDsrFlow, fTXContinueOnXoff, ReadIntervalTimeout, ReadTotalTimeoutConstant, ReadTotalTimeoutMultiplier, WriteTotalTimeoutConstant, WriteTotalTimeoutMultiplier; linux use different function, but meaning of the parameters is as same as possible)

  – 2 for file (1st item is mode: 1=seek begin, 2=seek current, 3=seek end, 4=set file end, 2nd item is offset for seek),

- 3 for SPI (1st item is SPI mode, 2nd item is bits per word, 3rd item is max speed in Hz),
- 5 for I2C (1st item is slave address, 2nd item is 10-bit address flag, 3rd item is Packet Error Checking flag, 4th item is nuber of retries, 5th item is timeout)
- other handle types are not supported

`void GetOptions(long handle, long params[])`
: Reads parameters of a socket or serial line to the `params` array. The array size must be big enough, at least 2 for files, 2 for a socket and 22 for serial line (see `SetOptions`).

`long Accept(long hListen)`
: Accepts the connection to listening socket `hListen` invoked by the client. A communication socket handle or an error is returned.

`long Read(long handle, long buffer[], long count)`
: Receives data from a serial line or socket. The `count` parameter defines the maximum number of bytes to read. The count of bytes read or an error code is returned. Each byte of incoming data is put to the `buffer` array of type `long` in the corresponding order.

It is also possible to use the form
`long Read(long handle, string data[], long count)` (i.e. a string is used instead of a data array; one byte in the input file corresponds to one character; not applicable to binary files).
The error codes are:

`-1`	it is necessary to wait for the operation to finish (the function is "non-blocking")
`-309`	reading failed; the operating system error code appears in the log (when function block logging is enabled)
`-307`	file/socket is not open

`long Write(long handle, long buffer[], long count)`
: Sends the data to a serial line or socket. The `count` parameter defines the number of bytes to send. The count of bytes or en error code sent is returned. Each byte of outgoing data is read from the `buffer` array of type `long` in the corresponding order.

It is also possible to use the form
`long Write(long handle, string data)` (i.e. a string is used instead of a data array; one byte in the output file corresponds to one character; not applicable to binary files).
The error codes are:

`-1`	it is necessary to wait for the operation to finish (the function is "non-blocking")
`-310`	write failed; the operating system error code appears in the log (when function block logging is enabled)
`-307`	file/socket is not open

`long ReadLine(long handle, string data)`
: Read one line from (text) file, serial line or socket; read characters are in the variable `data` up to allocated size of the string; the function return real size (number of bytes) of line or error code.

`long DeleteFile(string filename)`
: Delete file. Return 0 if success; negative value is error code.

`long RenameFile(string filename, string newfilename)`
: Rename file. Return 0 if success; negative value is error code.

`bool ExistFile(string filename)`
: Return true if file or device exist (is it possible to open it for reading).

`long I2C(long handle, long addr, long bufW[], long cntW, long bufR[], long cntR)`
: Communication over the I2C bus. Works only in Linux operating system on devices with the I2C bus (e.g. Raspberry Pi). Sends and receives data to/from the slave device with address `addr`. The parameter `handle` is returned by the `OpenI2C` function, whose parameter defines the device name (according to the operating system). The parameter `bufW` is a buffer (an array) for the data which is sent out, `cntW` is the number of bytes to send out, `bufR` is a buffer (an array) for the data which comes in and `cntR` is the number of bytes to receive. The function returns 0 or an error code.

`long SPI(long handle, 0, long bufW[], long cntW, long bufR[], long cntR)`
: Execution of one transaction over the SPI bus. Works only in Linux operating system on devices with the SPI bus (e.g. Raspberry Pi). The parameter `handle` is returned by the `OpenSPI` function, whose parameter defines the device name (according to the operating system). The second parameter is always 0 (reserved for internal use). The parameter `bufW` is a buffer (an array) for the data which is sent out, `cntW` is the number of bytes to send out, `bufR` is a buffer (an array) for the data which comes in and `cntR` is the number of bytes to receive. Note that SPI communication is full-duplex, therefore the resulting length of the SPI transaction is given by maximum of the `cntW` and `cntR` parameters, not their sum. The function returns 0 or an error code.

`long Seek(long handle, long mode[], long offset)`
: Set position for Read/Write command. Parameter mode means: 1=offset from begin of the file, 2= offset from current position, 3=offset from end of the file.

`long Recv(long handle, long buffer[], long count)`
: *Obsolete function. Use `Read` instead.*

`long Send(long handle, long buffer[], long count)`
: *Obsolete function. Use `Write` instead.*

`long crc16(data,length,init,poly,flags,offset)`
: Compute 16-bit cyclic redundand code that is used as checksum/hash in many comunication protocols. `data` byte array (represented by long array) or string to compute hash `length` number of bytes in input array/text (could be -1 for whole string) `init` so called initial vector `poly` so called

control polynom `flags` 1...revert bit order (in input bytes as so as in result crc), 2...result crc is xored with 0xFFFF, 4...if `data` is long array, all 4 bytes in long are procesed (LSB first), 8... same like 4, but MSB first `offset` index of the first processed byte in data array (usually 0) Notice: there is same function for 32-bit CRC `long crc32(data,length,init,poly,flags,offset)`, and for 8-bit CRC `long crc8(data,length,init,poly,flags,offset)`. Initial vector, control polynom, flags for many protocols could be found on https://crccalc.com/ Examples: MODBUS: crc16("123456789",-1,0xFFFF,0x8005,1,0)); DECT-X: crc16("123456789",-1,0,0x0589,0,0));

Remarks

- The data type of inputs `u0..u15`, outputs `y0..y15` and parameters `p0..p15` is determined during compilation of the source code according to the `input`, `output` and `parameter` definitions.
- All error codes `< -99` require restarting of the `REXLANG` function block by input `RESET`. Of course it is necessary to remove the cause of the error first.
- WARNING! – The inputs and outputs of the block cannot be accessed within the `init()` function (the values of inputs are 0, outputs are not set).
- It is possible to include path in the `srcname` parameter. Otherwise the file is expected directly in the project directory or in the directories specified by the `-I` command line option of the REXYGEN Compiler compiler.
- All parameters of the vector functions are of type `double` (or array of type `double`). The only exception is the `n` parameter of type long. Note that the functions with one vector parameter exist in three variants:

  `double function(val1,...,valn)`
  : Vector is defined as a sequence of values of type `double`.

  `double function(n,val1,...,valn)`
  : Vector is defined as in the first case, only the first parameter defines the number of values – the size of the vector. This variant is compatible with the C compiler. The `n` parameter must be a number, not the so-called `const` variable and it must correspond with the number of the following elements defining the vector.

  `double function(n,vec)`
  : The `n` parameter is an arbitrary expression of type `long` and defines the number of elements the function takes into account.

- The optional parameter `n` of the vector functions must be specified if the compatibility with C/C++ compiler is required. In such a case all the nonstandard functions must be implemented as well and the functions with variable number of parameters need to know the parameter count.

- In all case it is important to keep in mind that the vectors start at index 0 and that the array limits are not checked (just like in the C language). E.g. if `double vec[10], x;` is defined, the elements have indexes 0 to 9. The expression `x=vec[10];` is neither a syntax nor runtime error, the value is not defined. More importantly, it is possible to write `vec[11]=x;`, which poses a threat, because some other variable might be overwritten and the program works unexpectedly or even crashes.
- Only the `parser error` and line number are reported during compilation. This means a syntax error. If everything seems fine, the problem can be caused by identifier/keyword/function name conflict.
- All jumps are translated as relative, i.e. the corresponding code is restricted to 32767 instructions (in portable format for various platforms).
- All valid variables and temporary results are stored in the stack, namely:
  - Global variables and local `static` variables (permanently at the beginning of the stack)
  - Return addresses of functions
  - Parameters of functions
  - Local function variables
  - Return value of function
  - Temporary results of operations (i.e. the expression `a=b+c;` is evaluated in the following manner: `b` is stored in the stack, `c` is stored in the stack (it follows after `b`), the sum is evaluated, both values are removed from the stack and the result is stored in the stack

  Each simple variable (`long` or `double`) thus counts as one item in the stack. For arrays, only the size is important, not the type.
- The arrays are passed to the functions as a reference. This means that the parameter counts as one item in the stack and that the function works directly with the referenced array, not its local copy.
- If the stack size is not sufficient (less than space required for global variables plus 10), the stack size is automatically set to twice the size of the space required for the global variables plus 100 (for computations, function parameters and local variables in the case that only a few global variables are present).
- If basic debug level is selected, several checks are performed during the execution of the script, namely initialization of the values which are read and array index

limits. Also a couple of uninitialized values are inserted in front of and at the back of each declared array. The NOP instructions with line number of the source file are added to the *.ill file.

- If full debug is selected, additional check is engaged – the attempts to access invalid data range are monitored (e.g. stack overflow).
- The term instruction in the context of this block refers to an instruction of a processor-independent mnemocode. The mnemocode is stored in the *.ill file.
- The Open() function set serial line always 19200Bd, no parity, 8 bit per character, 1 stopbit, binary mode, no timeout. Optional 2nd (bitrate) and 3th (parity) parametrs can be used in the Open() function.
- Accessing text file is significantly slower that binary file. A advantage of the text file is possibility view/edit data in file without special editor.
- This block does not call the parchange() function. It is necessary to call it in init() function (if it is required).
- The block's inputs are available in the init() function, but all are equal to zero. It is possible (but not common) to set block's outputs.
- The Open() function also allows opening of a regular file. Same codes like in the LoadValue() function are used.

Debugging the code

Use the Trace command mentioned above.

Inputs

HLD	Hold – the block code is not executed if the input is set to on	Bool
RESET	Rising edge resets the block. The block gets initialized again (all global variables are cleared and the Init() function is called).	Bool
u0..u15	Input signals which are accessible from the script	Any

Outputs

`iE`	Runtime error code. For error codes `iE` < -99 the algorithm is stopped until it is reinitialized by the `RESET` input or by restarting the executive) `0` ..... No error occurred, the whole `main()` function was executed (also the `init()` function). `-1` .... The execution was suspended using the `Suspend()` command, i.e. the execution will resume as soon as the `REXLANG` block is executed again `< -1` .. Error code of the REXYGEN system, see Appendix C `> 0` ... User-defined return values, algorithm execution without any change	`Error`
`y0..y15`	Output signals which can be set from within the script	`Any`

Parameters

`srcname`	Source file name ⊙`srcfile.c`	`String`
`srctype`	Coding of source file ⊙1 `1: C-like` Text file respecting the C-like syntax described above `2: STL` Text file respecting the IEC61131-3 standard. The standard is implemented with the same limitations as the C-like script (i.e. no structures, only INT, REAL and STRING data types, function blocks are global variables VAR_INPUT, outputs are global variables VAR_OUTPUT, parameters are global variables VAR_PARAMETER, standard functions according to specification, system and communication functions are the same as in C-like). `3: RLB` `REXLANG` binary file which results from compilation of C-like or STL scripts. Use this option if you do not wish to share the source code of your block. `4: ILL` Text file with mnemocodes, which can be compared to assembler. This choice is currently not supported.	`Long (I32)`
`stack`	Stack size defined as number of variables. Default and recommended value is 0, which enables automatic estimation of the necessary stack size.	`Long (I32)`
`debug`	Debug level – checking is safer but slows down the execution of the algorithm. Option `No check` can crash REXYGENapplication on target platform if code is incorect. ⊙3 `1` ..... No check `2` ..... Basic check `3` ..... Full check	`Long (I32)`
`strs`	Total size of buffer for strings. Enter the maximum number of characters to allocate memory for. The default value 0 means that the buffer size is determined automatically.	`Long (I32)`
`p0..p15`	Parameters which are accessible from the script	`Any`

Example C-like

The following example shows a simple code to sum two input signals and also sum two user-defined parameters.

```
double input(0) input_u0;
double input(2) input_u2;

double parameter(0) param_p0;
double parameter(1) param_p1;

double output(0) output_y0;
double output(1) output_y1;

double my_value;

long init(void)
{
  my_value = 3.14;
  return 0;
}

long main(void)
{
  output_y0 = input_u0 + input_u2;
  output_y1 = param_p0 + param_p1 + my_value;
  return 0;
}

long exit(void)
{
  return 0;
}
```

Example STL

And here is the same example in Structured Text.

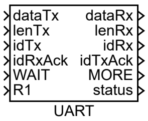
```
VAR_INPUT
  input_u0:REAL;
  input_u1:REAL;
  input_u2:REAL;
END_VAR

VAR_OUTPUT
  output_y0:REAL;
```

```
  output_y1:REAL;
END_VAR

VAR_PARAMETER
  param_p0:REAL;
  param_p1:REAL;
END_VAR

VAR
  my_value: REAL;
END_VAR

FUNCTION init : INT;
  my_value := 3.14;
  init := 0;
END_FUNCTION

FUNCTION main : INT;
  output_y0 := input_u0 + input_u2;
  output_y1 := param_p0 + param_p1 + my_value;
  main := 0;
END_FUNCTION

FUNCTION exit : INT;
  exit := 0;
END_FUNCTION
```

UART – UART communication block

Block Symbol

Licence: STANDARD



Function Description

The UART block allows you to read and write data via the Universal Asynchronous Receiver-Transmitter. The port parameter specifies device name. There it is possible to use two name types:

- the address of the physical device – Usually /dev/ttyS* for Linux target or COM* for Windows. Replace "*" symbol according to the chosen serial port!
- the virtual address – REXYGEN enables the creation of a virtual UART with which you can communicate inside REXYGEN with other blocks supporting UART such as REXLANG, PYTHON, another UART block or Modbus driver. On Linux devices, the virtual port is marked with the prefix pty: (pseudo terminal) and it is possible to connect to it from another application running on the device. On Windows devices, it is possible to use the prefix vcom, which enables communication within REXYGEN. Virtual port examples: pty:/tmp/vslave, vcom:vmaster.

UART communication has several general properties that are set using parameters such as baudrate, parity, databits and stopbits. Each packet that is received or transmitted is assigned a unique ID. The ID of the next packet is always one higher than the ID of the previous packet. Once the maximum ID is reached, the next ID assigned will be 0. The maximum ID value is determined by the maxId parameter. Data is sent with the rising edge of the idTx input.

Inputs

dataTx	Vector reference to transmitted data		Reference
lenTx	Transmitted data length (0 = whole vector)	↓0	Long (I32)
idTx	ID of the transmitted data packet	↓0	Long (I32)
idRxAck	ID of the last processed received data packet	↓0	Long (I32)
WAIT	Transmission suspended flag (data is buffered) on .... The transmitted data is still in the buffer off ... The transmitted data (the entire buffer) is sent		Bool
R1	Block reset		Bool

Outputs

`dataRx`	Vector reference to received data		`Reference`
`lenRx`	Received data length	↓0	`Long (I32)`
`idRx`	ID of the received data packet	↓0	`Long (I32)`
`idTxAck`	ID of the last processed transmitted data packet	↓0	`Long (I32)`
`MORE`	Additional data in the receive buffer flag		`Bool`
`status`	Internal status indicator `0` ..... No Error `1` ..... Transmit buffer overflow `2` ..... Transmit data error `256` ... Received data error `-1` .... Failed to open port		`Long (I32)`

Parameters

`port`	Communication device name		`String`
`baudrate`	Baudrate [bis/s] (0 = not set)	↓0 ↑4000000	`Long (I32)`
`parity`	Parity `0` ..... Not set `1` ..... No parity `2` ..... Odd parity `3` ..... Even parity		`Long (I32)`
`databits`	Number of data bits (0 = not set)	↓0 ↑3	`Long (I32)`
`stopbits`	Number of stop bits (0 = not set)	↓0 ↑2	`Long (I32)`
`maxId`	Max value used as ID of a packet	↓2 ↑10000000 ⊙4	`Long (I32)`
`maxLen`	Maximum length of the received data	↓1 ↑10000000 ⊙64	`Long (I32)`
`nmax`	Allocated size of array	↓8 ↑10000000 ⊙256	`Long (I32)`

Chapter 17

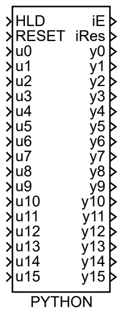
LANG – Special blocks

Contents

PYTHON – User programmable block in Python

Block Symbol

Licence: REXLANG



Function Description

The standard function blocks of the REXYGEN system cover the most typical needs in control applications. But there still exist situations where it is necessary (or more convenient) to implement an user-defined function. The REXLANG block covers this case for application where real-time behavior is strictly demanded. In the rest of the cases the PYTHON block can be used.

The PYTHON block implements an user-defined algorithm written in a Python scripting language and in comparison to the REXLANG block it provides a better user experience in the stage of development of the algorithm and can extend the feature set of REXYGEN system through various 3rd party libraries that are available in the Python environment.

Warning: the PYTHON block is intended for prototyping and experiments so please consider using the block in your application very carefully. It is an experimental block and always will be. There are many corner cases that may lead to unexpected behavior or even block the runtime. Packages may be poorly written or provide incorrect finalization and reinitialization which may even lead to a crash. Only a very limited support is provided for this block.

Scripting language

The scripting language is a standard Python v.3 language (see [12]). Every block references a script written in a *.py source file. The file can optionally contain functions with a reserved name that are then executed by REXYGEN. The `main()` function is executed periodically during runtime. Alongside the `main()` function the `init()` function is executed once at startup and after reset of the block, the `exit()` function is executed once when the control algorithm is stopped and before reset of the block and the `parchange()` function is executed on parameters change in REXYGEN.

Scripts on the target device

Standard python interpreter can load modules/scripts from various locations on the target device. The `PYTHON` block can reference any python script available for the standard interpreter and in addition the block can access scripts located in a directory `/rex/scripts/python`. User scripts can be directly uploaded to this directory or if the parameter `embedded` is set to `on` the script referenced by the block gets embedded in the REXYGEN configuration during compilation process and will be temporarily stored in the directory `/rex/scripts/python/embedded` during initialization of the block once the configuration is downloaded and executed on the target device.

Data exchange API

For the purpose of data exchange between a Python interpreter and REXYGEN system a module `PyRexExt` was developed as a native extension to the interpreter. The module contains an object `REX` that handles the data exchange operations. Use the following snippet at the start of the script to setup the data exchange API.

```
from PyRexExt import REX
```

I/O objects

`REX.u0 - REX.u15`
  – objects representing block *inputs* in Python environment

`REX.p0 - REX.p15`
  – objects representing block *parameters* in Python environment

`REX.y0 - REX.y15`
  – objects representing block *outputs* in Python environment

Access to values

All I/O objects contain a property `v`. Reading of the property `v` performs a conversion from REXYGEN data types to Python data types. The value then can be stored in variables and used in the block algorithm. A REXYGEN array type converts into a list of values in case of one-dimensional array or into a list of lists in case of multidimensional array.
Example of reading a value of the block input:

```
x = REX.u0.v
```

Writing to the property `v`, on the other hand, performs a conversion from Python data types to REXYGEN data types and exports the value to the corresponding block output/parameter.
Example of writing a value to the block output:

```
REX.y0.v = 5
```

Arrays

Input and output objects have a readonly property `size`. It is a tuple with number of rows and columns. Arrays can be manipulated through property `v` but direct conversions between REXYGEN arrays and Python lists are not very memory efficient. However, input and output objects support indexing operator `[]` that restricts the conversion overhead only on the specified item.
Example of reading a value of the block input for one-dimensional array:

```
x = REX.u0[0]
```

Example of writing a value to the block output for multidimensional array:

```
REX.u0[1, 3] = 5
```

External items

The object `REX` contains a method `Item` that returns a handle to an external REXYGEN item based on a connection string specified in a parameter of the method.
Example of creating a handle to an external item and setting its value:

```
cns = REX.Item("myproject_task.CNS:scv")
cns.v = "abc"
```

Tracing

The object `REX` contains methods `Trace`, `TraceError`, `TraceWarning`, `TraceVerbose` and `TraceInfo` can be used to write messages into REXYGEN system log. Every message has a stacktrace attached.
Example of logging a message:

```
REX.Trace("abc")
```

Additional features

`REX.RexDataPath` – `RexDataPath` is a string constant that contains a path to a data folder of the `REX` system on the given platform. That can come handy for writing a platform independent code that requires access to the file system using absolute paths.

Inputs

`HLD`	Hold – the block code is not executed if the input is set to `on`	`Bool`
`RESET`	Rising edge resets the block. The block gets initialized again (all global variables are cleared and the `init()` function is called).	`Bool`
`u0..u15`	Input signals which are accessible from the script.	`Any`

Outputs

iE	Runtime error code. 0 ..... No error occurred, the whole main() function was executed (also the init() function). xxx ... Error code of the REXYGEN system, see Appendix C	Error
iRes	Execution result code.	Long (I32)
y0..y15	Output signals which can be set from within the script.	Any

Parameters

srcname	Source file name	⊙program.py	String
embedded	Embedding of the script	⊙on	Bool
p0..p15	Parameters which are accessible from the script.		Any

Data types definition

For data exchange between REXYGEN system and Python environment the data types of block inputs signals u0..u15, outputs signals y0..y15 and parameters p0..p15 must be explicitly specified. For that purpose a configuration file must be created for every python script with the same name plus a suffix .cfg (e.g. program.py.cfg). If the file is missing during the compilation process it is created with all signal types set to double. It is not expected this file to be edited directly. User should use a build-in editor specific to the PYTHON block instead. Available types for inputs outputs and parameters are boolean, uint8, int16, uint16, int32, uint32, int64, float, double, string and in addition the inputs and outputs support array, numpy and image data types.

For types numpy and image the numpy python package must be installed on the target device. Inputs of the type numpy expect the connected signal to be of the type array that gets converted in the runtime to a native numpy representation. Inputs of the type image expects the connected signal to be of the type image data type from the RexVision module that also gets converted in the runtime to a native numpy representation and can therefore be directly used with the OpenCV Python package.

Outputs of the type numpy expect to be set in the script from a numpy array object that gets converted to a regular array. Outputs of the type image expect to be set in the script from a numpy array object that gets converted to image data type defined in the RexVision module.

Example data types definition

The following example shows a shortened JSON file describing the data types of the program inputs and outputs.

```
{
    "types": {
        "in": [
            {
                "idx": 0,
                "type": "double"
            },
             . . .
            {
                "idx": 15,
                "type": "double"
            }
        ],
        "param": [
            {
                "idx": 0,
                "type": "double"
            },
             . . .
            {
                "idx": 15,
                "type": "double"
            }
        ],
        "out": [
            {
                "idx": 0,
                "type": "double"
            },
             . . .
            {
                "idx": 15,
                "type": "double"
            }
        ]
    }
}
```

Example Python script

The following example shows a simple code to sum two input signals and also sum two user-defined parameters.

```
from PyRexExt import REX

def main():
    REX.y0.v = REX.u0.v + REX.u1.v
    REX.y1.v = REX.p0.v + REX.p1.v
    return
```

Installation - Debian

The Python environment should be correctly installed and configured just by installing the `PythonBlk_T` debian package. To install the package with optional `numpy` and `OpenCV` packages execute these commands from the terminal.

```
sudo apt install rex-pythonblkt
sudo apt install python3-numpy python3-opencv
```

Installation - Windows

To install the correct version of Python the recommended way is to download and install the 64-bit version from official repository (https://www.python.org/ftp/python/3.9.6/). During the installation make sure to enable installation of the `pip` program and adding of the python binaries to the system variable `PATH`.

To install `numpy` and `OpenCV` as optional dependencies execute following commands from the command line.

```
pip install numpy
pip install opencv-python
```

Limitations

Due to the limitations of the standard Python interpreter implementation it is not recommended to use multiple `PYTHON` block instances at the different levels of executive. Doing so can lead to an unpredictable behavior and instability of the `RexCore` program.

Chapter 18

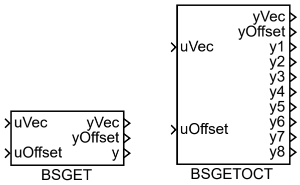
DSP – Digital Signal Processing blocks

Contents

BSGET, BSGETOCT – Binary Structure - Get a single value of given type

Block Symbols

Licence: ADVANCED



Function Description

This group of blocks is used for obtaining values from a binary structure (byte array). The BSSET and BSSETOCT blocks can be used to write to the binary structure.
If binary structures are received using communication, it is possible to process them directly in the block mediating communication. Typically this is a REXLANG or PYTHON programmable block. Using structures, however, it is possible to transfer data within the REXYGEN application as well. The binary structure is fed in the form of an array (vector) of bytes to the `uVec` input. The `uOffset` input specifies the offset (in bytes) of the desired value from the beginning of the structure. The value type is specified by the `type` parameter.
The `yOffset` output is the start of the next element in the structure. This is advantageous for chaining: if the structure contains several elements in a row, it is possible to connect the input `uOffset` to the output `yOffset` of the previous block and it is not necessary to calculate the offset.
The only difference between the blocks is that `BSGET` gets a single value. The `BSGETOCT` block is able to receive up to 8 values (the number is determined by the `m` parameter).

Inputs

`uVec`	Binary Structure (array of bytes) input	`Reference`
`uOffset`	Offset to start in the input Binary Structure (in bytes)	`Long (I32)`

Outputs

`yVec`	Copy of the `uVec` input, for easy chaining	`Reference`
`y`	Scalar value output (scalar type defined by parameter)	`Any`
`yOffset`	Offset after the last processed byte of the input Binary Structure (in bytes), for easy chaining	`Long (I32)`

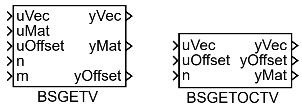
Parameters

`m`	Number of used values (for multi-blocks)	↓1 ↑8 ⊙8	`Long (I32)`
`BE`	Big-Endian byte order (default is Little-Endian, e.g. Intel)		`Bool`
`type`	Data type of item	↓2 ↑10 ⊙2	`Long (I32)`

BSGETV, BSGETOCTV – Binary Structure - Get matrix (all values of the same given type)

Block Symbols

Licence: ADVANCED



Function Description

This group of blocks is used for obtaining values from a binary structure (byte array). The BSSETV and BSSETOCTV blocks can be used to write to the binary structure.
The meaning of most of the parameters is the same as the BSGET block, but these blocks retrieve several values of the same type and store them in an array (matrix). A matrix always has `m` rows and `n` columns. For the `BSGETV` block, all elements are of the same type (determined by the `type` parameter) and the data is filled into the matrix fed to the `uMat` input. The `BSGETOCTV` block loads up to 8 vectors. Each row of the matrix can be of a different type. The block allocates the matrix itself. The matrix is available at the `yMat` output.

Inputs

`uVec`	Binary Structure (array of bytes) input	`Reference`
`uMat`	Reference of matrix for output values	`Reference`
`uOffset`	Offset to start in the input Binary Structure (in bytes)	`Long (I32)`
`n`	Number of matrix columns	`Long (I32)`
`m`	Number of active items	`Long (I32)`

Outputs

`yVec`	Copy of the `uVec`, for easy chaining	`Reference`
`yMat`	Copy of the `uMat`, for easy chaining	`Reference`
`yOffset`	Offset after the last processed byte of the input Binary Structure (in bytes), for easy chaining	`Long (I32)`

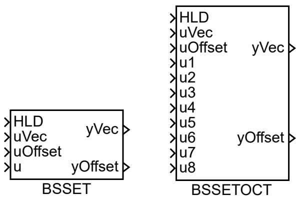
Parameters

`m`	Number of active items (for multi-blocks)	↓1 ↑8 ⊙8	`Long (I32)`
`BE`	Big-Endian byte order (default is Little-Endian, e.g. Intel)		`Bool`
`nmax`	Allocated size of output matrix (total number of items) `yMat`	↓1 ⊙32	`Long (I32)`
`type`	Data type of item	↓2 ↑10 ⊙2	`Long (I32)`

BSSET, BSSETOCT – Binary Structure - Set a single value of given type

Block Symbols

Licence: ADVANCED



Function Description

This group of blocks is used for setting values into a binary structure (byte array). The function is the inverse of the BSGET and BSGETOCT blocks, i.e. all signals have the same meaning, only the data is copied in the opposite direction - from the **u** input to the binary structure represented by the byte array connected to the **uVec** input. The block modifies the binary structure only if **HLD=off**.

Inputs

HLD	Hold	Bool
uVec	Binary Structure (array of bytes) input	Reference
uOffset	Offset to start in the input Binary Structure (in bytes)	Long (I32)
u	Scalar value input (scalar type defined by parameter)	Any

Outputs

yVec	Copy of the **uVec** input, for easy chaining	Reference
yOffset	Offset after the last processed byte of the input Binary Structure (in bytes), for easy chaining	Long (I32)

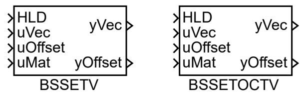
Parameters

m	Number of active items (for multi-blocks)	↓1 ↑8 ⊙8	Long (I32)
BE	Big-Endian byte order (default is Little-Endian, e.g. Intel)		Bool
type	Data type of item	↓2 ↑10 ⊙2	Long (I32)

BSSETV, BSSETOCTV – Binary Structure - Set matrix of given type

Block Symbols

Licence: ADVANCED



Function Description

This group of blocks is used to set the matrix of values into a binary structure (byte array). The function is the inverse of the BSGETV and BSGETOCTV blocks, i.e. all signals have the same meaning, only the data is copied in the opposite direction - from the `uMat` input to the binary structure represented by the byte array connected to the `uVec` input. The block modifies the binary structure only if `HLD=off`.
Unlike the BSGETV block, the numbers of rows and columns are not specified, but are determined from the actual size of the matrix connected to the `uMat` input.

Inputs

`HLD`	Hold	`Bool`
`uVec`	Binary Structure (array of bytes) input	`Reference`
`uOffset`	Offset to start in the input Binary Structure (in bytes)	`Long (I32)`
`uMat`	Reference of the matrix with input values	`Reference`

Outputs

`yVec`	Copy of the `uVec` input, for easy chaining	`Reference`
`yOffset`	Offset after the last processed byte of the input Binary Structure (in bytes), for easy chaining	`Long (I32)`

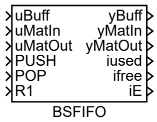
Parameters

`m`	Number of active items (for multi-blocks)	↓1 ↑8 ⊙8	`Long (I32)`
`BE`	Big-Endian byte order (default is Little-Endian, e.g. Intel)		`Bool`
`type`	Data type of item	↓2 ↑10 ⊙2	`Long (I32)`

BSFIFO – Binary Structure - Queueing serialize and deserialize

Block Symbol

Licence: ADVANCED



Function Description

This block sequentially adds or removes data to/from the buffer (passed to the `uBuff` input). The elementary unit in a buffer is a column. All matrices (ie matrices or vectors fed to the inputs `uBuff`, `uMatIn`, `uMatOut`) must have the same column size in bytes. Data is organized as either a queue (if `REV=off`) or a stack (if `REV=on`). The behavior of the block depends on the inputs in this way:

- If `PUSH=on`, the content of the `uMatIn` matrix (all defined columns) is inserted into the buffer.
- If `POP=on`, the number of columns determined by the `col` parameter is removed from the buffer and this data is inserted into the `uMatOut` matrix (it must be of sufficient size).
- If `R1=on`, the data is reloaded (mainly the number of valid columns) into the block buffer. Own data is transmitted by reference and is therefore shared. This signal has priority and blocks `PUSH`, `POP` signals.

Error states (e.g. mismatched matrix dimensions, insufficient space in some matrices, lack of data in the buffer) are indicated on the `iE` output and by a message in the SystemLog.

Inputs

`uBuff`	Binary Structure (array of bytes) input		`Reference`
`uMatIn`	Input reference to a matrix or vector (for PUSH)		`Reference`
`uMatOut`	Input reference to a matrix or vector (for POP)		`Reference`
`PUSH`	Enable push data		`Bool`
`POP`	Enable pop data		`Bool`
`R1`	Buffer reset (reload headers from `uBuff`)		`Bool`

Parameters

`OW`	Overwrite oldest items in buffer		`Bool`
`REV`	Pop last pushed item first		`Bool`
`col`	Number of output (pop) columns	⊙1	`Long (I32)`

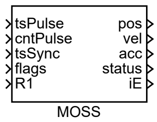
Outputs

`yBuff`	Copy of the `uBuff` input, for easy chaining	`Reference`
`yMatIn`	Output reference to a matrix or vector `uMatIn`	`Reference`
`yMatOut`	Output reference to a matrix or vector `uMatOut`	`Reference`
`iused`	Used bytes in queue	`Long (I32)`
`ifree`	Free bytes in queue	`Long (I32)`
`iE`	Error code	`Error`

MOSS – Motion smart sensor

Block Symbol

Licence: ADVANCED



Function Description

The MOSS block implements a precise position filter for a quadrature (incremental) encoder. The implementation requires special hardware that produce not only pulse count, but also the exact timestamp of last pulse, the direction of last pulse and the timestamp of the block start execution time (or similar reference time). Both timestamps must be from the same (or synchronized) source. The output of the MOSS block is not only the filtered position, but also the filtered velocity and acceleration. The filtering level is set by the alpha parameter. The setting is a compromise between noise reduction and signal delay (averaging).

Note 1: Some people think that a quadrature encoder will get a precise position value (accurate to 1 pulse) and the exact velocity could be obtained by dividing the pulse difference by the time difference. We believe that the MOSS block achieve better results. You can put both values into a graph (the TRND block) and check the differences.

Note 2: The filter is implemented as a Kalman filter for a second-order system (input is acceleration, output is position) discretized for a variable sampling period (current timestamp difference). Input and output noises are necessary parameters for Kalman filter design. If both noises are Gaussian, the only parameter is ratio of input and output noise that is the alpha parameter of the MOSS block.

Inputs

tsPulse	Last pulse timestamp	DWord (U32)
cntPulse	Last pulse count	DWord (U32)
tsSync	Timestamp of the Sync pulse. The Sync pulse is the time when the filtered outputs are valid.	DWord (U32)
flags	Input status flags (1: POS, 2: NEG, 4: RUN)	DWord (U32)
R1	Block reset	DWord (U32)

Parameters

freq	Source timestamp tick frequency [Hz] ↓0.0 ⊙100000000.0	Double (F64)
stall	Stalled time [s]. If no pulse is received in the stalled time interval, sensor is considered stopped and the outputs (pos, vel,acc) are set to 0. ↓0.0 ⊙0.08	Double (F64)

`alpha`	Kalman filter design parameter. A lower value means smoother (less noise) outputs, but more delayed in dynamic situation (when acceleration is changing). ↓0.0 ↑200.0 ⊙26.0	`Double (F64)`
`maxpos`	Kalman filter rounding optimization. If `pos` is greater then `maxpos`, the internal position processed by the Kalman filter is decremented by an integer multiple of `maxpos` and incremented back for output. This causes the filter algorithm to calculate small enough numbers and not reduce accuracy due to rounding errors. The default value should not normally be changed. ↓0.0 ⊙1e+10	`Double (F64)`
`mindivert`	Predictor minimal divert time [s]. If no pulse is received for a long time, the predictor output will drift. To overcome this drift, if no pulse is detected for longer then `mindivert` time, the output position is clamped to +-1 pulse from input (mesured) position. ↓0.0 ⊙0.003	`Double (F64)`

Outputs

`pos`	Filtered position	`Double (F64)`
`vel`	Filtered velocity	`Double (F64)`
`acc`	Filtered acceleration	`Double (F64)`
`status`	Output status flags (1: POS, 2: NEG, 4: RUN, 8: INIT, 16: PULSE, 32: STALLED, 64: DIVERT)	`Long (I32)`
`iE`	Error code `i` ..... REXYGEN general error	`Error`

Chapter 19

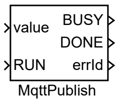
MQTT – Communication via MQTT protocol

Contents

MqttPublish – Publish MQTT message

Block Symbol

Licence: MQTT



Function Description

This function block depends on the MQTT driver. Please read the MQTTDrv manual [13] before use.

The MqttPublish block publishes messages to an MQTT broker through the connection established by the MQTTDrv driver.

The first parameter is the topic the block will publish the messages to. MQTT delivers Application Messages according to the Quality of Service (QoS) levels. Use the QoS parameter to set a different Quality of Service level. See the MQTT specification [14] for more details.

If the RETAIN parameter is set a RETAIN flag will be set on the outgoing PUBLISH Control Packet. See the MQTT specification [14] for more details.

The defBuffSize parameter can be used to optimize the memory usage of the block. It states the amount of the statically allocated memory for the inner buffer for the outgoing messages. If the value is unnecessarily large the memory is being wasted. On the other hand if the value of the parameter is too small it leads to frequent dynamic memory allocations which can be time consuming.

The message to be published is constructed from the value input signal. The value input signal is expected to be a string. If it is not a string it will be converted automatically. To request a message to be published in the current period set the RUN flag to on. The BUSY flag is on if the block has a pending request and waits for a response from a broker. When the response is received in the current cycle the DONE flag is set to on.

Inputs

value	Input signal	String
RUN	Enable execution	Bool

Parameters

topic	MQTT topic	String

QoS	Quality of Service 1 ..... QoS0 (At most once) 2 ..... QoS1 (At least once) 3 ..... QoS2 (Exactly once)	⊙1	Long (I32)
RETAIN	Retain last message	⊙on	Bool
defBuffSize	Default buffer size	↓1 ⊙2048	Long (I32)

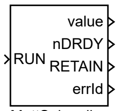
Outputs

BUSY	Busy flag	Bool
DONE	Indicator of finished transaction	Bool
errId	Error code	Error

MqttSubscribe – Subscribe to MQTT topic

Block Symbol

Licence: MQTT



Function Description

This function block depends on the MQTT driver. Please read the MQTTDrv manual [13] before use.

The MqttSubscribe block subscribes to a topic on an MQTT broker and receives Publish messages on that topic through the connection established by the MQTTDrv driver.

The first parameter is the topic the block will subscribe to. MQTT protocol delivers Application Messages according to the Quality of Service (QoS) levels. Use the QoS parameter to set a different Quality of Service level. See the MQTT specification [14] for more details.

By setting the type parameter of the block it can be specified the expected data type of the incoming message. The block converts the incoming message to the specified type and sets the value output signal in case of success or it sets the errId to the resulting error code.

The mode parameter has two available options: Last value and Buffered values. If Last value mode is used the block will always output only the last message received even if multiple messages were received in the last period. If the mode is set to Buffered values than the block buffers the incoming messages and outputs one by one in consecutive ticks of the task.

The defBuffSize parameter can be used to optimize the memory usage of the block. It states the amount of the statically allocated memory in the inner buffer for the incoming messages. If the value is unnecessarily large the memory is being wasted. On the other hand if the value of the parameter is too small it leads to frequent dynamic memory allocations which can be time consuming.

A Subscribe action is performed upon a rising edge (off→on) and an Unsubscribe action is performed upon a falling edge (on→off) at the RUN input.

The nDRDY output specifies how many messages were received and are available in the inner buffer. If the mode of the block is set to Last value the nDRDY output can only have value 0 or 1.

The RETAIN output flag is set if the received Publish packet had the RETAIN flag set. See the MQTT specification [14] for more details.

Note that subscribing to topics containing wildcards is not supported.

Input

RUN	Enable execution		Bool

Parameters

topic	MQTT topic		String
QoS	Quality of Service 1 ..... QoS0 (At most once) 2 ..... QoS1 (At least once) 3 ..... QoS2 (Exactly once)	⊙1	Long (I32)
type	Expected type of incoming data 1 ..... string 2 ..... double 3 ..... long 4 ..... bool 5 ..... byte vector/blob	⊙1	Long (I32)
mode	Incoming messages buffering mode 1 ..... Last value 2 ..... Buffered values	⊙1	Long (I32)
defBuffSize	Default buffer size	↓1 ⊙2048	Long (I32)

Outputs

value	Output signal		Any
nDRDY	Number of received messages	↓0 ↑10	Long (I32)
errId	Error code		Error

Chapter 20

MC_SINGLE – Motion control - single axis blocks

Contents

This library includes functional blocks for single axis motion control as it is defined in the PLCopen specification. It is recommended to study the PLCopen specification prior to using the blocks from this library. The knowledge of PLCopen is necessary for advanced use of the blocks included in this library.

PLCopen defines all blocks with the MC_ prefix. This notation is kept within this library. Nevertheless, there are also function blocks, which are not described by PLCopen or are described as *vendor specific*. These blocks can be recognized by the RM_ prefix. Note that PLCopen (and also IEC 61131-3 which is the base for PLCopen) does not use block parameters, all the parameters are specified by input signals. In the REXYGEN, block parameters are used to simplify usage of the blocks. To keep compatibility with PLCopen and improve usability of the blocks, almost all of them are implemented twice: with prefix MC_ without parameters (parameters are inputs) and with prefix MCP_ with parameters. Some blocks require additional *vendor specific* parameters. In such a case even the MC_-prefixed blocks contain parameters.

PLCopen specifies that all inputs/parameters are sampled at rising-edge of the Execute input. In REXYGEN block parameters are usually changed very rare. Therefore the parameters of the activated block have not be changed until block is finished (e.g. while output Busy is on).

The REXYGEN system does not allow input-output signals and all signals must have different name. For these reasons the Axis input-output signal, which is used in all blocks, is divided into input uAxis and output yAxis. The block algorithm copies the input uAxis to the output yAxis. The yAxis output is not necessary for the function of motion control blocks, but "chaining" the axis references makes it possible to order the blocks and define priorities. Other reference signals are either defined as input-only or use this mechanism as well.

PLCopen defines the outputs Busy, Active, CommandAborted as optional in almost all blocks. In REXYGEN, some of them are never set, but the outputs are defined to simplify future extensions and/or changes in the implementation.

Units used for position and distance of axis are implementation specific. It can be meters, millimeters, encoder ticks, angular degrees (for rotational axis) or any others, but all blocks connected to one axis must use the same position units. Time is always defined in seconds. Velocity unit is thus "position units per second" and acceleration unit is "position units per square second".

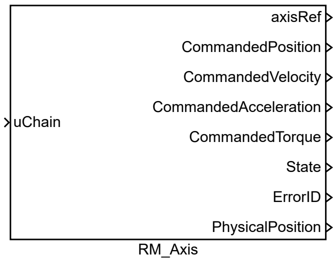
The REXYGEN system uses more threads for execution of the function blocks. In standard function blocks the synchronization is provided by the system and the user does not need to care about it. But using the `Reference` references could violate the synchronization mechanisms. However, there is no problem if all referenced blocks are located in the same task and therefore e.g. the `RM_Axis` block must be in the same task as all other blocks connected to this axis.

Some inputs/parameters are of enumeration type (for example `BufferMode` or `Direction`). It is possible to choose any of the defined values for this type in the `MCP_` version of the blocks, although not all of them are valid for all blocks (for example the block `MC_MoveVelocity` does not support `Direction` = `shortest_way`). Valid values for each block are listed in this manual.

RM_Axis – Motion control axis

Block Symbol

Licence: MOTION CONTROL



Function Description

The RM_AXIS block is a cornerstone of the motion control solution within the REXYGEN system. This base block keeps all status values and implements basic algorithm for one motion control axis (one motor), which includes limits checking, emergency stop, etc. The block is used for both real and virtual axes. The real axis must have a position feedback controller, which is out of this block's scope. The key status values are commanded position, velocity, acceleration and torque, as well as state of the axis, axis error code and a reference to the block, which controls the axis.

This block (like all blocks in the motion control library) does not implement a feedback controller which would keep the actual position as near to the commanded position as possible. Such a controller must be provided by using other blocks (e.g. PIDU) or external (hardware) controller. The feedback signals are used for lag checking, homing and could be used in special motion control blocks. The feedback signals are connected throw the RM_AxisSpline block.

The parameters of this block correspond with the requirements of the PLCopen standard for an axis. If improper parameters are set, the errorID output is set to -700 (invalid parameter) and all motion blocks fail with -703 error code (invalid state).

The parameters for limit velocity, acceleration and deceleration are twofold. One for application, e.g. limit value which could be set into movtion blocks. This value could be exceeded in some cases. Second limit is for system. The system limits must be higher then application limits and it is never exceeded. If some motion block generate path, that exceed system limit, error stop seqvence is activated.

Note that the default values for position, velocity and acceleration limits are intentionally set to 0, which makes them invalid. Limits must always be set by the user according to the real axis and the axis actuator.

Inputs

uChain	Input is not used by the block. User can connect any signal to define order of block's execution	Long (I32)

Outputs

`axisRef`	Axis reference (only `RM_Axis.axisRef-uAxis` or `yAxis-uAxis` connections are allowed)	`Reference`
`CommandedPosition`	Requested (commanded) position of the axis. The value is logical position that is put into the motion blocks. The position is different from `PhysicalPosition` if the axis is circular or homed.	`Double (F64)`
`CommandedVelocity`	Requested (commanded) velocity of the axis	`Double (F64)`
`CommandedAcceleration`	Requested (commanded) acceleration of the axis	`Double (F64)`
`CommandedTorque`	Requested (commanded) torque in the axis	`Double (F64)`
`State`	State of the axis `0` ..... Disabled `1` ..... Stand still `2` ..... Homing `3` ..... Discrete motion `4` ..... Continuous motion `5` ..... Synchronized motion `6` ..... Coordinated motion `7` ..... Stopping `8` ..... Error stop `9` ..... Drive error(simillar to Error stop, but fault is caused by external signnal)	`Long (I32)`
`ErrorID`	Result of the last operation `i` ..... REXYGEN general error	`Error`
`PhysicalPosition`	Requested (commanded) position of the axis. The value is physical position that is put into the feedback controller. The position is different from `CommandedPosition` if the axis is circular or homed.	`Double (F64)`

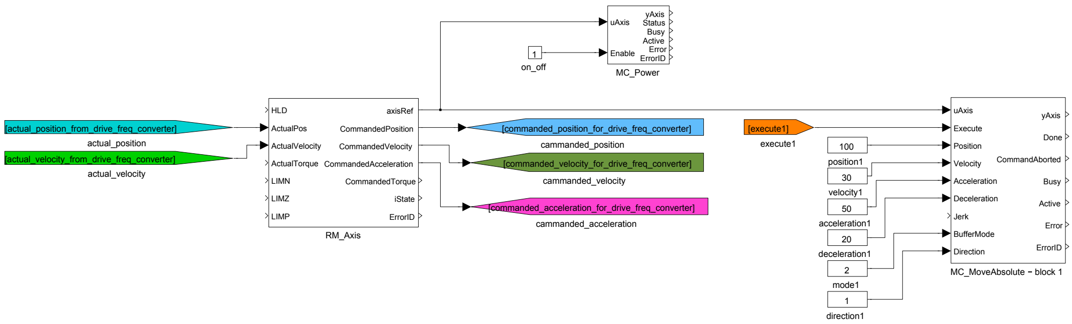
Parameters

`AxisType`	Type of the axis ⊙1 `1` ..... Linear axis `2` ..... Cyclic axis with cyclic position sensor `3` ..... Cyclic axis with linear position sensor	`Long (I32)`
`EnableLimitPos`	Enable positive position limit checking (e.g. if checked, `MaxPosAppl` is valid)	`Bool`
`MaxPosAppl`	Positive position limit for application (MC blocks). The value should be smaller then (before the) `MaxPosSystem` for linear axis. The value limit cyclic axis with linear senzor for few revolution (useful for robotic application) and must be bigger then (beyond the) `MaxPosSystem`.	`Double (F64)`
`MaxPosSystem`	Positive position limit for system. The value is never exceeded for linear axis. The value is end of revolution for cyclic axis.	`Double (F64)`
`EnableLimitNeg`	Enable negative position limit checking (e.g. if checked, `MinPosAppl` is valid)	`Bool`

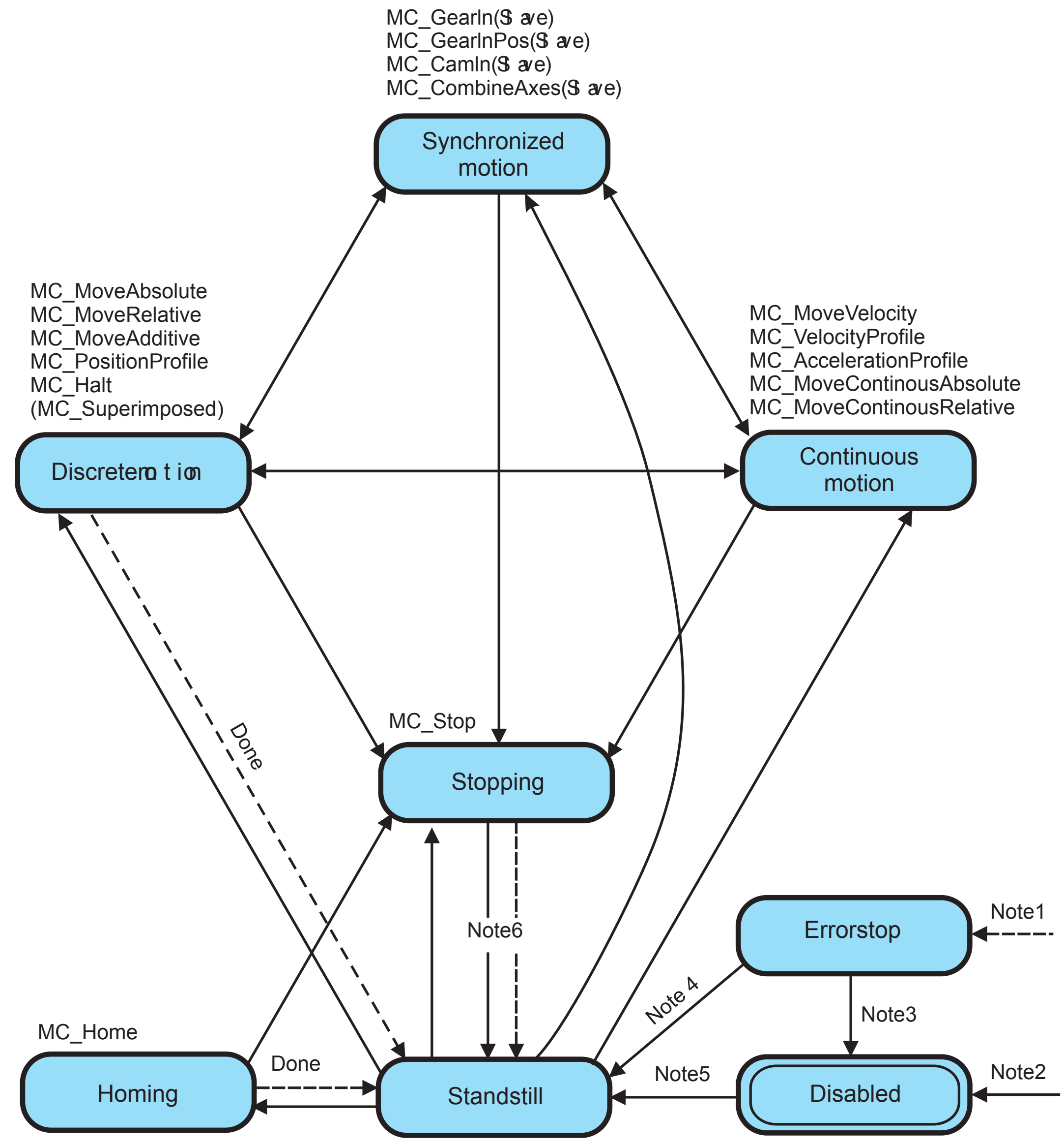
`MinPosAppl`	Negative position limit for application (MC blocks) The value should be bigger then (before the) `MinPosSystem` for linear axis. The value limit cyclic axis with linear senzor for few revolution (useful for robotic application) and must be smaller then (beyond the) `MinPosSystem`.	`Double (F64)`
`MinPosSystem`	Negative position limit for system. The value is never exceeded for linear axis. The value is begin of revolution for cyclic axis.	`Double (F64)`
`EnablePosLagMonitor`	Enable monitoring of position lag (e.g. if checked, `MaxPositionLag` is valid)	`Bool`
`MaxPositionLag`	Maximal position lag. Any moving is stopped and the axis is switched into error stop state if different between `PhysicalPosition` and `ActualPosition` exceed this value.	`Double (F64)`
`MaxVelocitySystem`	Maximal allowed velocity for system	`Double (F64)`
`MaxVelocityAppl`	Maximal allowed velocity for application (MC blocks)	`Double (F64)`
`MaxAccelerationSystem`	Maximal allowed acceleration for system	`Double (F64)`
`MaxAccelerationAppl`	Maximal allowed acceleration for application (MC blocks)	`Double (F64)`
`MaxDecelerationSystem`	Maximal allowed deceleration for system	`Double (F64)`
`MaxDecelerationAppl`	Maximal allowed deceleration for application (MC blocks)	`Double (F64)`
`DefaultJerk`	Maximal recomended jerk [unit/s^3]. Real jerk is not checked and could overcome this value.	`Double (F64)`
`MaxTorque`	Maximal motor torque/force (0=not used)	`Double (F64)`
`TorqueRatio`	Torque-Acceleration ratio. The requested torque value is useful for feedback controller. The most block don't generate it. The requested torque value is comuted as reqested acceleration multiplied by this parameter.	`Double (F64)`
`LoopDelay`	delay between commanded and actual values[s] The actual position value is deleyed from commanded value due communication with feedback controller, feedback loop, value interpolation and sampling period. The delay could be set into this parameter and then position lag is computed more precisely. (not yet implemented)	`Double (F64)`
`StartMode`	Some options when axis is enabled ⊙1 1 ..... start stopped 2 ..... start tracking	`Long (I32)`
`HomingRequired`	Homing is required before any move	`Bool`

Example

Following example illustrates basic principle of use of motion control blocks. It presents the minimal configuration which is needed for operation of a physical or virtual axis. The axis is represented by `RM_Axis` block. The limitations imposed on the motion trajectory in form of maximum velocity, acceleration, jerk and position have to be set in parameters of the `RM_Axis` block. The inputs can be connected to supply the values of actual position, speed and torque (feedback for slip monitoring) or logical limit switch signals for homing procedure. The `axisRef` output signal needs to be connected to any motion control block related to the corresponding axis. The axis has to be activated by enabling the `MC_Power` block. The state of the axis changes from Disabled to Standstill (see the following state transition diagram) and any discrete, continuous or synchronized motion can be started by executing a proper functional block (e.g. `MC_MoveAbsolute`). The trajectory of motion in form of desired position, velocity and acceleration is generated in output signals of the `RM_Axis` block. The reference values are provided to an actuator control loop which is implemented locally in REXYGEN system in the same or different task or they are transmitted via a serial communication interface to end device which controls the motor motion (servo amplifier, frequency inverter etc.). In case of any error, the axis performs an emergency stop and indicates the error ID. The error has to be confirmed by executing the `MC_Reset` block prior to any subsequent motion command. The following state diagram demonstrates the state transitions of an axis.



Axis state transition diagram



Note 1: From any state. An error in the axis occurred.

Note 2: From any state. MC_Power.Enable = FALSE and there is no error in the axis.

Note 3: MC_Reset AND MC_Power.Status = FALSE

Note 4: MC_Reset AND MC_Power.Status = TRUE AND MC_Power.Enable = TRUE

Note 5: MC_Power.Enable = TRUE AND MC_Power.Status = TRUE

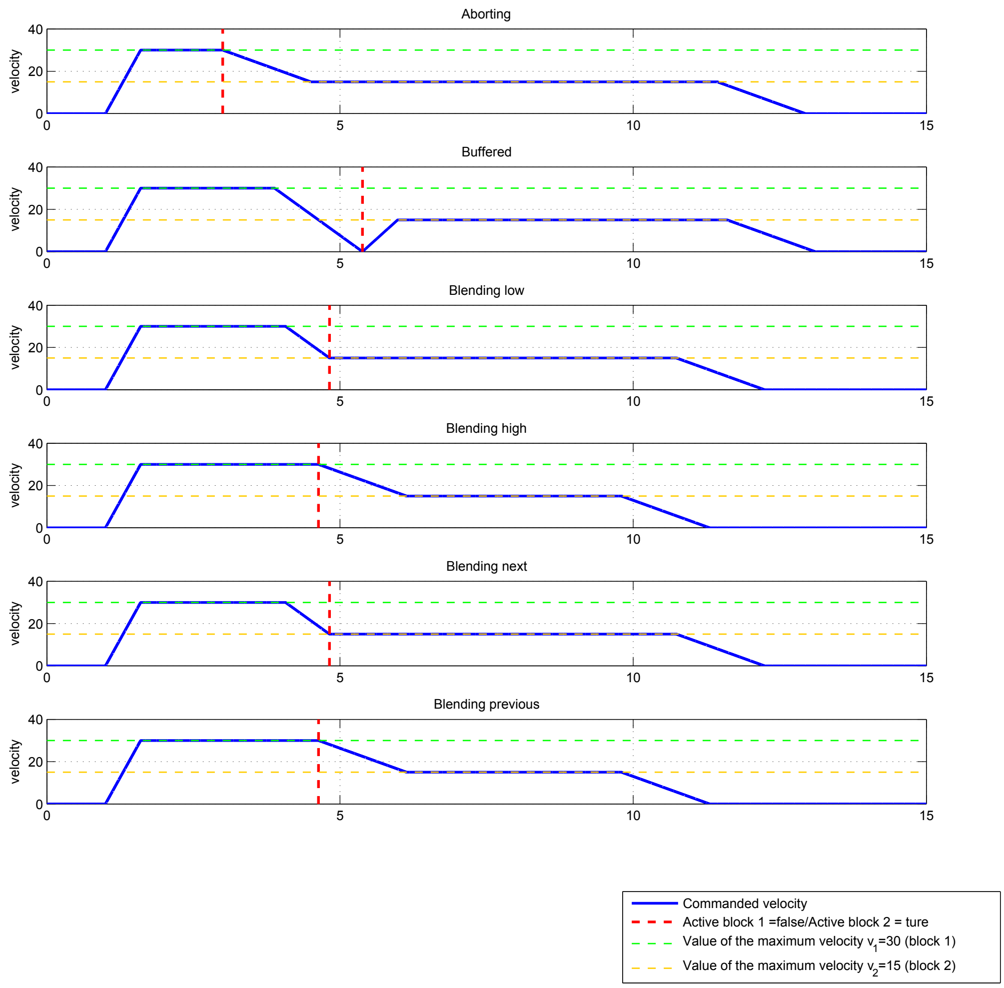
Note 6: MC_Stop.Done = TRUE AND MC_Stop.Execute = FALSE

Motion blending

According to `PLCOpen` specification, number of motion control blocks allow to specify `BufferMode` parameter, which determines a behaviour of the axis in case that a motion command is interrupted by another one before the first motion is finished. This transition from one motion to another (called "Blending") can be handled in various ways. The following table presents a brief explanation of functionality of each blending mode and the resulting shapes of generated trajectories are illustrated in the figure. For detailed description see full `PLCOpen` specification.

Aborting	The new motion is executed immediately
Buffered	the new motion is executed immediately after finishing the previous one, there is no blending
Blending low	the new motion is executed immediately after finishing the previous one, but the axis will not stop between the movements, the first motion ends with the lower limit for maximum velocity of both blocks at the first end-position
Blending high	the new motion is executed immediately after finishing the previous one, but the axis will not stop between the movements, the first motion ends with the higher limit for maximum velocity of both blocks at the first end-position
Blending previous	the new motion is executed immediately after finishing the previous one, but the axis will not stop between the movements, the first motion ends with the limit for maximum velocity of first block at the first end-position
Blending next	the new motion is executed immediately after finishing the previous one, but the axis will not stop between the movements, the first motion ends with the limit for maximum velocity of second block at the first end-position

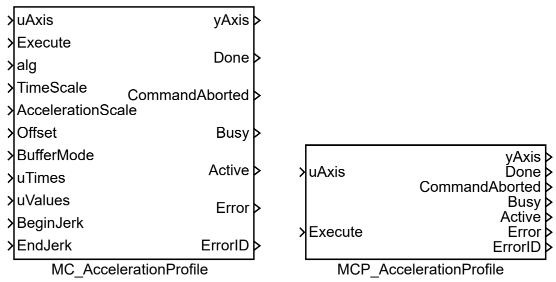
Illustration of blending modes



MC_AccelerationProfile, MCP_AccelerationProfile – Acceleration profile

Block Symbols

Licence: MOTION CONTROL



Function Description

The MC_AccelerationProfile *and* MCP_AccelerationProfile *blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the* MCP_ *version of the block.*

The MC_PositionProfile block commands a time-position locked motion profile. Block implements two possibilities for definition of time-acceleration function:

1. sequence of values: the user defines a sequence of time-acceleration pairs. In each time interval, the values of velocity are interpolated. Times sequence is in array times, position sequence is in array values. Time sequence must be increasing and must start with zero or zero must be between the first and last point. Execution always starts from zero time, so if the sequence start with negative time, part of the profile is not executed (could be used for debugging or time shift). For MC_VelocityProfile and MC_AccelerationProfile interpolation is linear, but for MC_PositionProfile, 3rd order polynomial is used in order to avoid steps in velocity.

2. spline: time sequence is the same as in previous case. Each interval is interpolated by 5th order polynomial $p(x) = a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$ where beginning of the time-interval is for $x = 0$, end of time-interval is for $x = 1$ and factors a_i are put in array values in ascending order (e.g. array values contains 6 values for each interval). This method allows smaller number of intervals and there is special editor for synthesis of the interpolating spline function.

For both case, the time sequence could be equally spaced and then array times includes only the first (usually zero) and last point.

Note 1: input TimePosition is missing, because all path data are in parameters of the block.

Note 2: parameter values must be set as vector in all cases, e.g. text string must not include semicolon.

Note 3: incorrect parameter `cSeg` (higher then real size of arrays `times` and/or `values`) leads to unpredictable result and in some cases crashes whole runtime execution (The problem is platform dependent and currently it is known only for SIMULINK - crash of whole MATLAB).

Note 4: in the spline mode, polynomial is always 5th order and always in position (also for sibling block `MC_PositionProfile` and `MC_VelocityProfile`) and it couldn't be changed. As the special editor exists, this is not important limitation.

Note 5: The block does not include ramp-in mode. If start position and/or velocity of profile is different from actual (commanded) position of axis, block fails with error `-707` (step). It is recommended to use `BufferMode=BlendingNext` to eliminate the problem with start velocity.

Inputs

`uAxis`	Axis reference (only `RM_Axis.axisRef`–`uAxis` or `yAxis`–`uAxis` connections are allowed)	`Reference`
`Execute`	The block is activated on rising edge	`Bool`
`TimeScale`	Overall scale factor in time	`Double (F64)`
`AccelerationScale`	Overall scale factor in value	`Double (F64)`
`Offset`	Overall profile offset in value	`Double (F64)`
`BufferMode`	Buffering mode `1` ..... Aborting (start immediately) `2` ..... Buffered (start after finish of previous motion) `3` ..... Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands) `4` ..... Blending high (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands) `5` ..... Blending previous (start after finishing the previous motion, previous motion finishes with its final velocity) `6` ..... Blending next (start after finishing the previous motion, previous motion finishes with the starting velocity of the next block)	`Long (I32)`

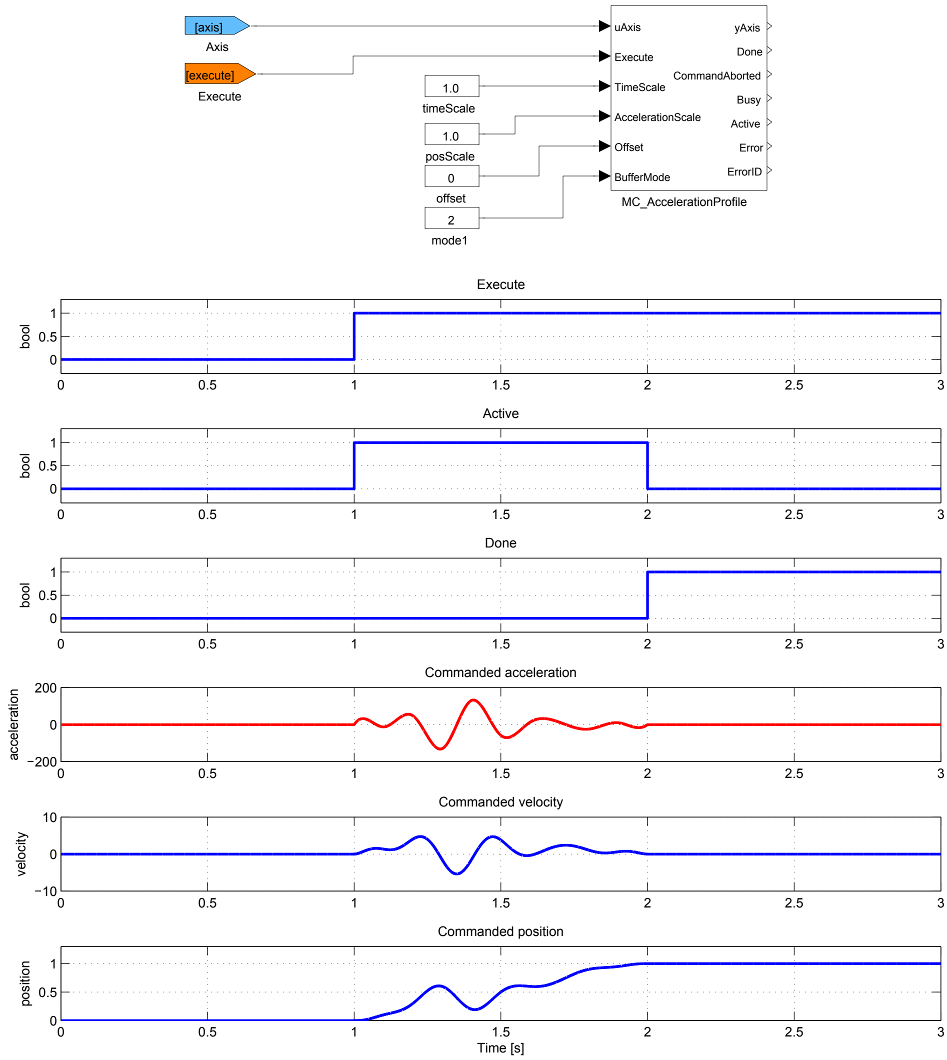
Outputs

`yAxis`	Axis reference (only `RM_Axis.axisRef`–`uAxis` or `yAxis`–`uAxis` connections are allowed)	`Reference`
`Done`	Algorithm finished	`Bool`
`CommandAborted`	Algorithm was aborted	`Bool`
`Busy`	Algorithm not finished yet	`Bool`
`Active`	The block is controlling the axis	`Bool`
`Error`	Error occurred	`Bool`
`ErrorID`	Result of the last operation `i` ..... REXYGEN general error	`Error`

Parameters

`alg`	Algorithm for interpolation `1` ..... Sequence of time/value pairs `2` ..... Sequence of equidistant values `3` ..... Spline `4` ..... Equidistant spline	⊙2	`Long (I32)`
`nmax`	Number of profile segments	⊙3	`Long (I32)`
`times`	Times when segments are switched		`Reference`
`values`	Values or interpolating polynomial coefficients (a0, a1, a2, ...)		`Reference`

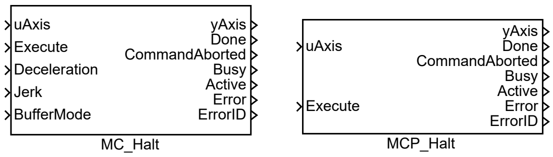
Example


[axis]
Axis
[execute]
Execute
1.0
timeScale
1.0
posScale
0
offset
2
mode1
uAxis
Execute
TimeScale
AccelerationScale
Offset
BufferMode
yAxis
Done
CommandAborted
Busy
Active
Error
ErrorID
MC_AccelerationProfile
Execute
Active
Done
bool
Commanded acceleration
acceleration
Commanded velocity
velocity
Commanded position
position
Time [s]

MC_Halt, MCP_Halt – Stopping a movement (interruptible)

Block Symbols

Licence: MOTION CONTROL



Function Description

The MC_Halt *and* MCP_Halt *blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the* MCP_ *version of the block.*

The MC_Halt block commands a controlled motion stop and transfers the axis to the state DiscreteMotion. After the axis has reached zero velocity, the Done output is set to true immediately and the axis state is changed to Standstill.

Note 1: Block MC_Halt is intended for temporary stop of an axis under normal working conditions. Any next motion command which cancels the MC_Halt can be executed in nonbuffered mode (opposite to MC_Stop, which cannot be interrupted). The new command can start even before the stopping sequence was finished.

Inputs

uAxis	Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool
Deceleration	Maximal allowed deceleration [unit/s^2]	Double (F64)
Jerk	Maximal allowed jerk [unit/s^3]	Double (F64)

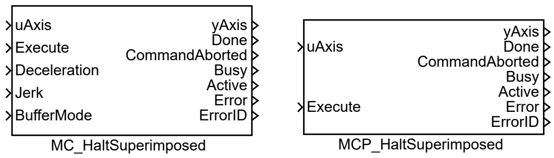
Outputs

yAxis	Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)	Reference
Done	Algorithm finished	Bool
CommandAborted	Algorithm was aborted	Bool
Busy	Algorithm not finished yet	Bool
Active	The block is controlling the axis	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation i ..... REXYGEN general error	Error

MC_HaltSuperimposed, MCP_HaltSuperimposed – Stopping a movement (superimposed and interruptible)

Block Symbols

Licence: MOTION CONTROL



Function Description

The MC_HaltSuperimposed *and* MCP_HaltSuperimposed *blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the* MCP_ *version of the block.*

Block MC_HaltSuperimposed commands a halt to all superimposed motions of the axis. The underlying motion is not interrupted.

Inputs

uAxis	Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool
Deceleration	Maximal allowed deceleration [unit/s^2]	Double (F64)
Jerk	Maximal allowed jerk [unit/s^3]	Double (F64)

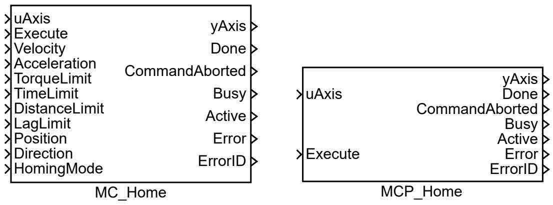
Outputs

yAxis	Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)	Reference
Done	Algorithm finished	Bool
CommandAborted	Algorithm was aborted	Bool
Busy	Algorithm not finished yet	Bool
Active	The block is controlling the axis	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation i ..... REXYGEN general error	Error

MC_Home, MCP_Home – Homing

Block Symbols

Licence: MOTION CONTROL



Function Description

The MC_Home *and* MCP_Home *blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the* MCP_ *version of the block.*

The MC_Home block commands the axis to perform the "search home" sequence. The details of this sequence are described in PLCopen and can be set by parameters of the block. The "Position" input is used to set the absolute position when reference signal is detected. This Function Bock completes at "StandStill".

Note 1: Parameter/input BufferMode is not supported. Mode is always Aborting. It is not limitation, because homing is typically done once in initialization sequence before some regular movement is proceeded.

Note 2: Homing procedure requires some of RM_Axis block input connected. Depending on homing mode, ActualPos, ActualTorque, LimP, LimZ, LimN can be required. It is expected that only one method is used. Therefore, there are no separate inputs for zero switch and encoder reference pulse (both must be connected to LimZ).

Note 3: HomingMode=4(Direct) only sets the actual position. Therefore, the MC_SetPosition block is not implemented. HomingMode=5(Absolute) only switches the axis from state Homing to state StandStill.

Note 4: Motion trajectory for homing procedure is implemented in simpler way than for regular motion commands - acceleration and deceleration is same (only one parameter) and jerk is not used. For extremely precise homing (position set), it is recommended to run homing procedure twice. First, homing procedure is run with "high" velocity to move near zero switch, then small movement (out of zero switch) follows and finally second homing procedure with "small" velocity is performed.

Note 5: HomingMode=6(Block) detect home-position when the actual torque reach value in parameter TorqueLimit or position lag reach value in parameter MaxPositionLag in attached RM_Axis block (only if the parameter has positive value).

Inputs

`uAxis`	Axis reference (only `RM_Axis.axisRef`–`uAxis` or `yAxis`–`uAxis` connections are allowed)	`Reference`
`Execute`	The block is activated on rising edge	`Bool`
`Velocity`	Maximal allowed velocity [unit/s]	`Double (F64)`
`Acceleration`	Maximal allowed acceleration [unit/s^2]	`Double (F64)`
`TorqueLimit`	Maximal allowed torque/force	`Double (F64)`
`TimeLimit`	Maximal allowed time for the whole algorithm [s]	`Double (F64)`
`DistanceLimit`	Maximal allowed distance for the whole algorithm [unit]	`Double (F64)`
`Position`	Requested target position (absolute) [unit]	`Double (F64)`
`Direction`	Direction of movement (cyclic axis or special case only) `1` ..... Positive `2` ..... Shortest `3` ..... Negative `4` ..... Current	`Long (I32)`
`HomingMode`	Homing mode algorithm `1` ..... Absolute switch `2` ..... Limit switch `3` ..... Reference pulse `4` ..... Direct (user reference) `5` ..... Absolute encoder `6` ..... Block	`Long (I32)`

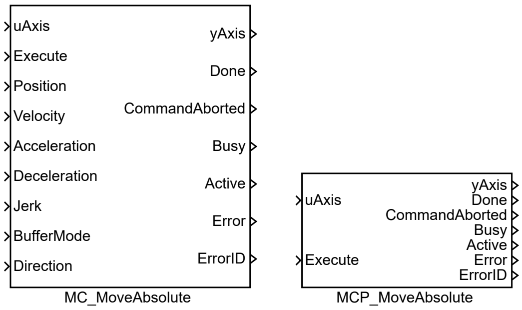
Outputs

`yAxis`	Axis reference (only `RM_Axis.axisRef`–`uAxis` or `yAxis`–`uAxis` connections are allowed)	`Reference`
`Done`	Algorithm finished	`Bool`
`CommandAborted`	Algorithm was aborted	`Bool`
`Busy`	Algorithm not finished yet	`Bool`
`Active`	The block is controlling the axis	`Bool`
`Error`	Error occurred	`Bool`
`ErrorID`	Result of the last operation `i` ..... REXYGEN general error	`Error`

MC_MoveAbsolute, MCP_MoveAbsolute – **Move to position (absolute coordinate)**

Block Symbols

Licence: MOTION CONTROL



Function Description

The MC_MoveAbsolute *and* MCP_MoveAbsolute *blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the* MCP_ *version of the block.*

The MC_MoveAbsolute block moves an axis to specified position as fast as possible. If no further action is pending, final velocity is zero (axis moves to position and stops) otherwise it depends on blending mode. For blending purposes, start and stop velocity of this block is maximum velocity with direction respecting current and final position. If start velocity of next pending block is in opposite direction, then blending velocity is always zero.

If next pending block is executed too late in order to reach requested velocity the generated output depends on jerk setting. If no limit for jerk is used (block input Jerk is zero or unconnected) block uses maximum acceleration or deceleration to reach the desired velocity as near as possible. If jerk is limited it is not possible to say what is the nearest velocity because also acceleration is important. For this reason, the axis is stopped and moved backward and blending velocity is always reached. Although this seems to be correct solution, it might look confusing in a real situation. Therefore, it is recommended to reorganize execution order of the motion blocks and avoid this situation.

The MC_MoveRelative block act almost same as MC_MoveAbsolute. The only difference is the final position is computed adding input Distance to current (when rising edge on input Execute occurred) position.

The MC_MoveAdditive block act almost same as MC_MoveRelative. The only difference is the final position is computed adding input Distance to final position of the previous block.

The MC_MoveSuperimposed block acts almost the same as the MC_MoveRelative block. The only difference is the current move is not aborted and superimposed move is executed immediately and added to current move. Original move act like superimposed move is not run.

The following table describes all inputs, parameters and outputs which are used in some of the blocks in the described block suite.

Inputs

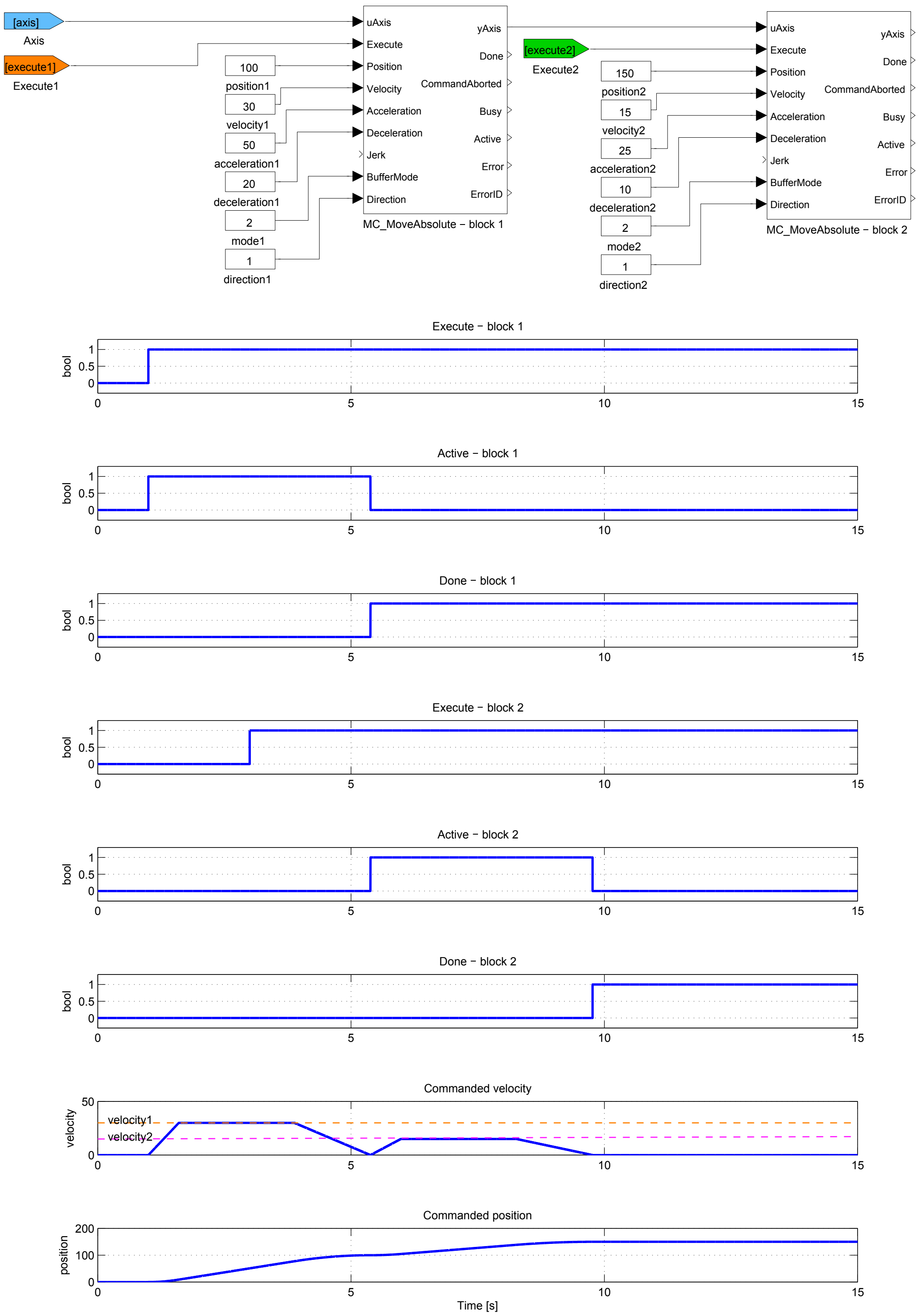
`uAxis`	AAxis reference (only `RM_Axis.axisRef`–`uAxis` or `yAxis`–`uAxis` connections are allowed)	`Reference`
`Execute`	The block is activated on rising edge	`Bool`
`Position`	Requested target position (absolute) [unit]	`Double (F64)`
`Velocity`	Maximal allowed velocity [unit/s]	`Double (F64)`
`Acceleration`	Maximal allowed acceleration [unit/s^2]	`Double (F64)`
`Deceleration`	Maximal allowed deceleration [unit/s^2]	`Double (F64)`
`Jerk`	Maximal allowed jerk [unit/s^3]	`Double (F64)`
`BufferMode`	Buffering mode `1` ..... Aborting (start immediately) `2` ..... Buffered (start after finish of previous motion) `3` ..... Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands) `4` ..... Blending high (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands) `5` ..... Blending previous (start after finishing the previous motion, previous motion finishes with its final velocity) `6` ..... Blending next (start after finishing the previous motion, previous motion finishes with the starting velocity of the next block)	`Long (I32)`
`Direction`	Direction of movement (cyclic axis or special case only) `1` ..... Positive `2` ..... Shortest `3` ..... Negative `4` ..... Current	`Long (I32)`

Outputs

`yAxis`	Axis reference (only `RM_Axis.axisRef`–`uAxis` or `yAxis`–`uAxis` connections are allowed)	`Reference`
`Done`	Algorithm finished	`Bool`
`CommandAborted`	Algorithm was aborted	`Bool`
`Busy`	Algorithm not finished yet	`Bool`
`Active`	The block is controlling the axis	`Bool`
`Error`	Error occurred	`Bool`

`ErrorID`	Result of the last operation `i` ..... REXYGEN general error	`Error`

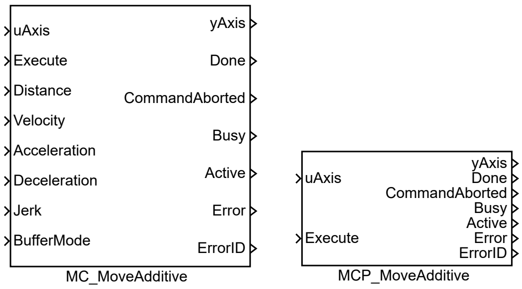
Example


[axis]
Axis
[execute1]
Execute1
100
position1
30
velocity1
50
acceleration1
20
deceleration1
2
mode1
1
direction1
uAxis
Execute
Position
Velocity
Acceleration
Deceleration
Jerk
BufferMode
Direction
yAxis
Done
CommandAborted
Busy
Active
Error
ErrorID
MC_MoveAbsolute – block 1
[execute2]
Execute2
150
position2
15
velocity2
25
acceleration2
10
deceleration2
2
mode2
1
direction2
MC_MoveAbsolute – block 2
Execute – block 1
Active – block 1
Done – block 1
Execute – block 2
Active – block 2
Done – block 2
bool
Commanded velocity
velocity
velocity1
velocity2
Commanded position
position
Time [s]

MC_MoveAdditive, MCP_MoveAdditive – Move to position (relative to previous motion)

Block Symbols

Licence: MOTION CONTROL



Function Description

The MC_MoveAdditive *and* MCP_MoveAdditive *blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the* MCP_ *version of the block.*

The MC_MoveAdditive block moves an axis to specified position as fast as possible. The final position is determined by adding the value of Distance parameter to final position of previous motion block which was controlling the axis. If no further action is pending, final velocity is zero (axis moves to position and stops) otherwise it depends on blending mode. For blending purposes, start and stop velocity of this block is maximum velocity with direction respecting current and final position. If start velocity of next pending block is in opposite direction, then blending velocity is always zero.

If next pending block is executed too late in order to reach requested velocity the generated output depends on jerk setting. If no limit for jerk is used (block input Jerk is zero or unconnected) block uses maximum acceleration or deceleration to reach the desired velocity as near as possible. If jerk is limited it is not possible to say what is the nearest velocity because also acceleration is important. For this reason, the axis is stopped and moved backward and blending velocity is always reached. Although this seems to be correct solution, it might look confusing in a real situation. Therefore, it is recommended to reorganize execution order of the motion blocks and avoid this situation.

Inputs

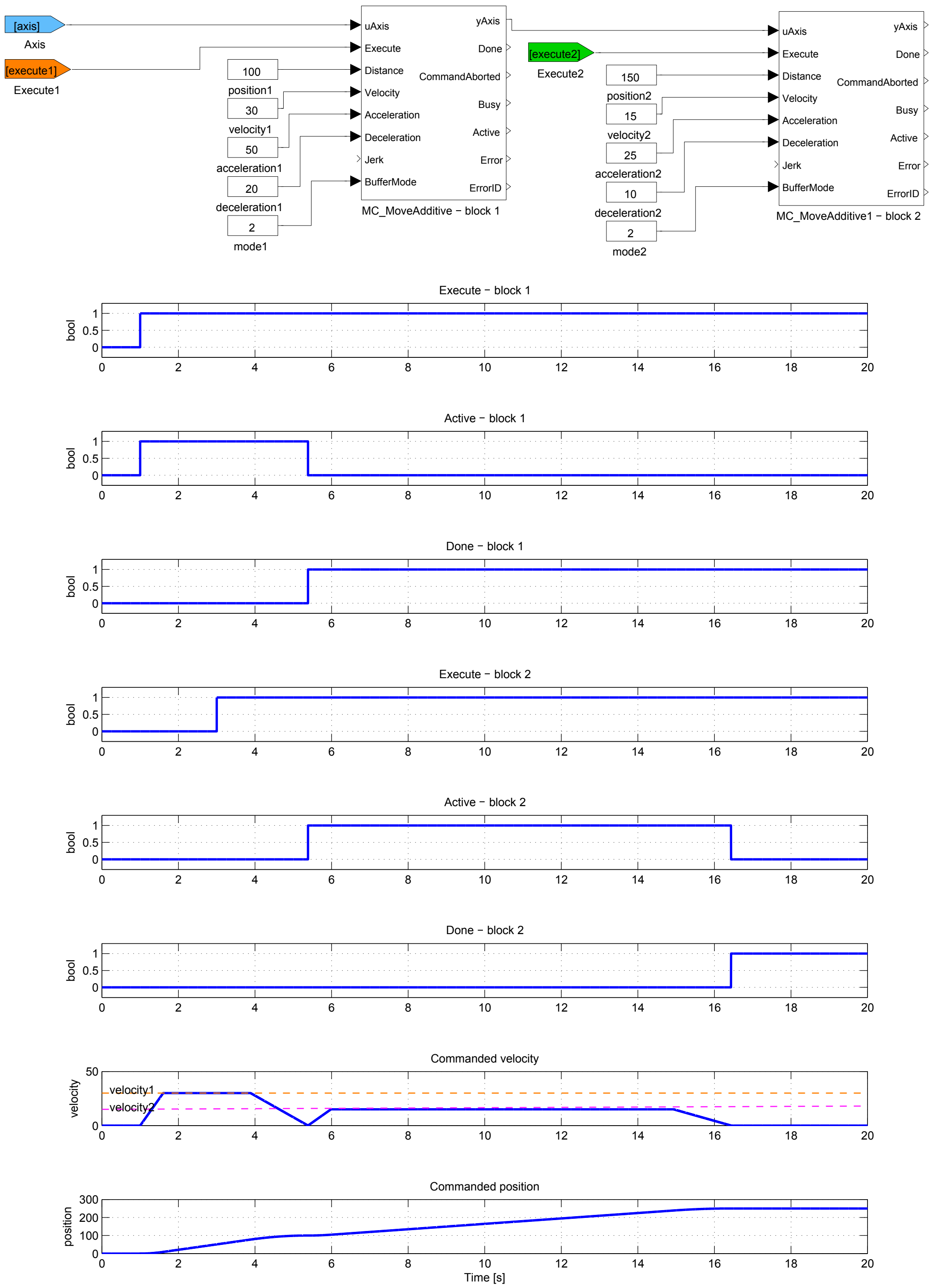
uAxis	Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool
Distance	Requested target distance (relative to start point) [unit]	Double (F64)

`Velocity`	Maximal allowed velocity [unit/s]	`Double (F64)`
`Acceleration`	Maximal allowed acceleration [unit/s^2]	`Double (F64)`
`Deceleration`	Maximal allowed deceleration [unit/s^2]	`Double (F64)`
`Jerk`	Maximal allowed jerk [unit/s^3]	`Double (F64)`
`BufferMode`	Buffering mode `1` ..... Aborting (start immediately) `2` ..... Buffered (start after finish of previous motion) `3` ..... Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands) `4` ..... Blending high (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands) `5` ..... Blending previous (start after finishing the previous motion, previous motion finishes with its final velocity) `6` ..... Blending next (start after finishing the previous motion, previous motion finishes with the starting velocity of the next block)	`Long (I32)`

Outputs

`yAxis`	Axis reference (only `RM_Axis.axisRef`–`uAxis` or `yAxis`–`uAxis` connections are allowed)	`Reference`
`Done`	Algorithm finished	`Bool`
`CommandAborted`	Algorithm was aborted	`Bool`
`Busy`	Algorithm not finished yet	`Bool`
`Active`	The block is controlling the axis	`Bool`
`Error`	Error occurred	`Bool`
`ErrorID`	Result of the last operation `i` ..... REXYGEN general error	`Error`

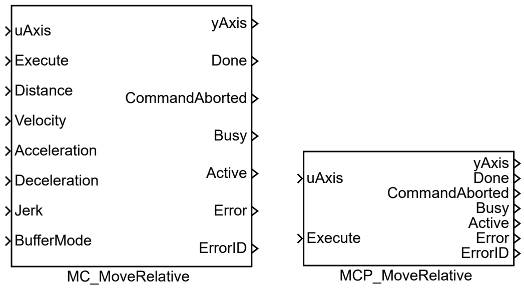
Example


[axis]
Axis
[execute1]
Execute1
100
position1
30
velocity1
50
acceleration1
20
deceleration1
2
mode1
uAxis
Execute
Distance
Velocity
Acceleration
Deceleration
Jerk
BufferMode
yAxis
Done
CommandAborted
Busy
Active
Error
ErrorID
MC_MoveAdditive − block 1
[execute2]
Execute2
150
position2
15
velocity2
25
acceleration2
10
deceleration2
2
mode2
MC_MoveAdditive1 − block 2
Execute − block 1
Active − block 1
Done − block 1
Execute − block 2
Active − block 2
Done − block 2
bool
Commanded velocity
velocity
velocity1
velocity2
Commanded position
position
Time [s]

MC_MoveRelative, MCP_MoveRelative – Move to position (relative to execution point)

Block Symbols

Licence: MOTION CONTROL



Function Description

The MC_MoveRelative *and* MCP_MoveRelative *blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the* MCP_ *version of the block.*

The MC_MoveRelative block moves an axis to specified position as fast as possible. The final position is determined by adding the value of Distance parameter to the actual position at the moment of triggering the Execute input. If no further action is pending, final velocity is zero (axis moves to position and stops) otherwise it depends on blending mode. For blending purposes, start and stop velocity of this block is maximum velocity with direction respecting current and final position. If start velocity of next pending block is in opposite direction, then blending velocity is always zero.

If next pending block is executed too late in order to reach requested velocity the generated output depends on jerk setting. If no limit for jerk is used (block input Jerk is zero or unconnected) block uses maximum acceleration or deceleration to reach the desired velocity as near as possible. If jerk is limited it is not possible to say what is the nearest velocity because also acceleration is important. For this reason, the axis is stopped and moved backward and blending velocity is always reached. Although this seems to be correct solution, it might look confusing in a real situation. Therefore, it is recommended to reorganize execution order of the motion blocks and avoid this situation.

Inputs

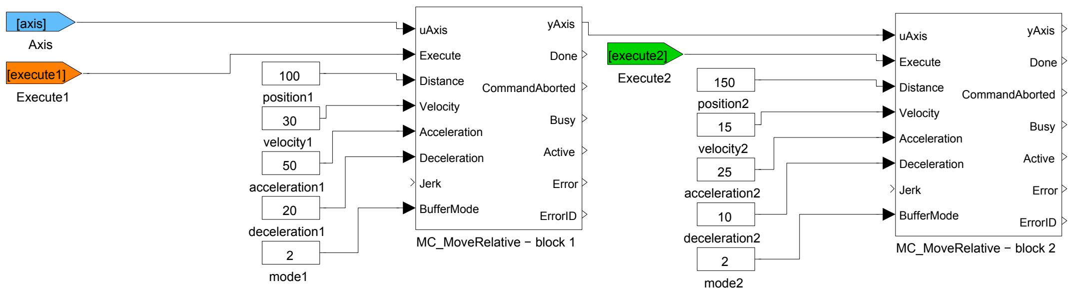
uAxis	Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool
Distance	Requested target distance (relative to execution point) [unit]	Double (F64)

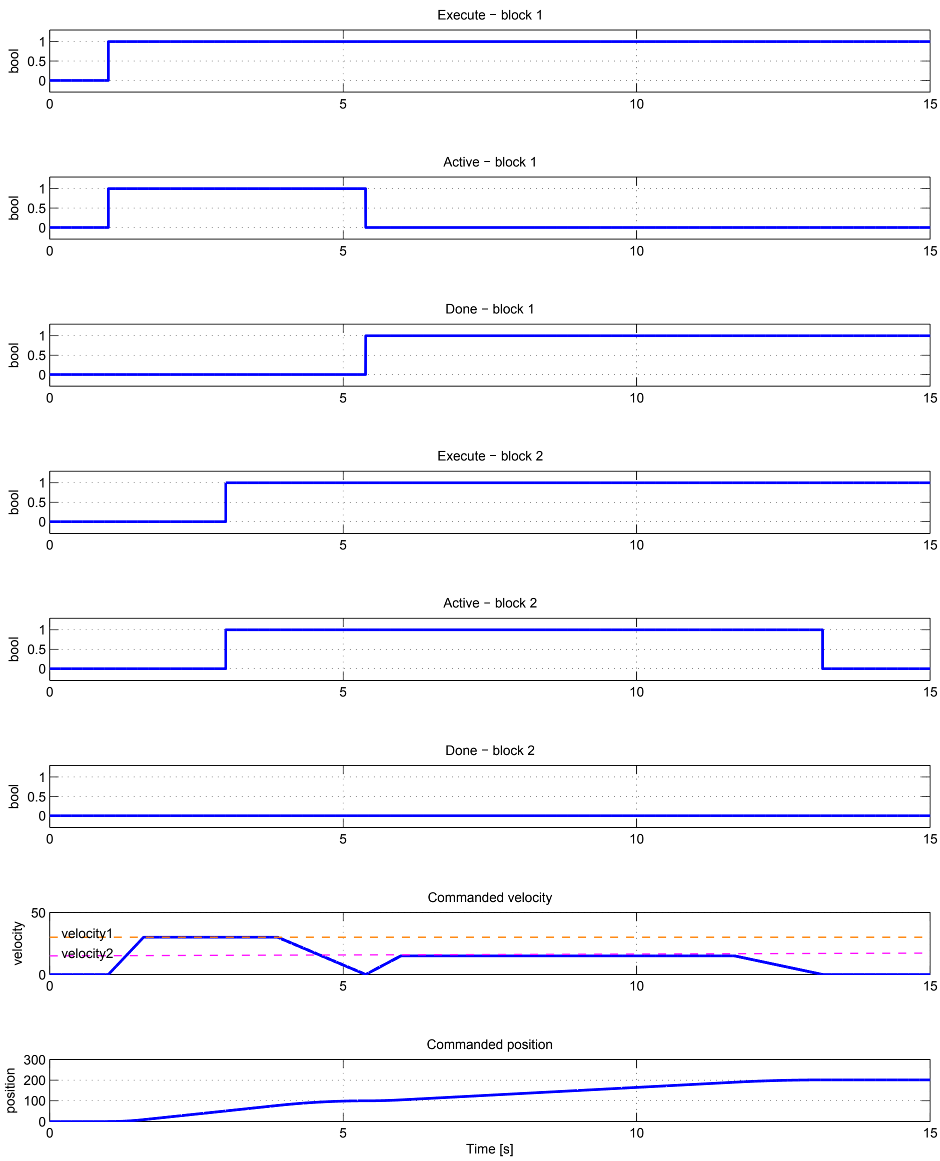
`Velocity`	Maximal allowed velocity [unit/s]	`Double (F64)`
`Acceleration`	Maximal allowed acceleration [[unit/s^2]	`Double (F64)`
`Deceleration`	Maximal allowed deceleration [[unit/s^2]	`Double (F64)`
`Jerk`	Maximal allowed jerk [[unit/s^3]	`Double (F64)`
`BufferMode`	Buffering mode `1` ..... Aborting (start immediately) `2` ..... Buffered (start after finish of previous motion) `3` ..... Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands) `4` ..... Blending high (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands) `5` ..... Blending previous (start after finishing the previous motion, previous motion finishes with its final velocity) `6` ..... Blending next (start after finishing the previous motion, previous motion finishes with the starting velocity of the next block)	`Long (I32)`

Outputs

`yAxis`	Axis reference (only `RM_Axis.axisRef`–`uAxis` or `yAxis`–`uAxis` connections are allowed)	`Reference`
`Done`	Algorithm finished	`Bool`
`CommandAborted`	Algorithm was aborted	`Bool`
`Busy`	Algorithm not finished yet	`Bool`
`Active`	The block is controlling the axis	`Bool`
`Error`	Error occurred	`Bool`
`ErrorID`	Result of the last operation `i` ..... REXYGEN general error	`Error`

Example

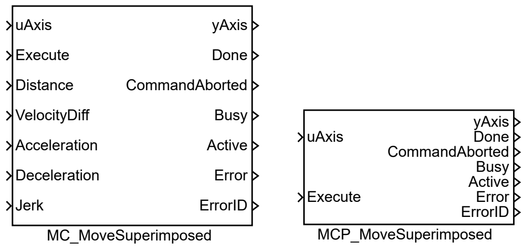

[axis]
Axis
[execute1]
Execute1
100
position1
30
velocity1
50
acceleration1
20
deceleration1
2
mode1
uAxis
Execute
Distance
Velocity
Acceleration
Deceleration
Jerk
BufferMode
yAxis
Done
CommandAborted
Busy
Active
Error
ErrorID
MC_MoveRelative − block 1
[execute2]
Execute2
150
position2
15
velocity2
25
acceleration2
10
deceleration2
2
mode2
MC_MoveRelative − block 2


Execute − block 1
Active − block 1
Done − block 1
Execute − block 2
Active − block 2
Done − block 2
bool
Commanded velocity
velocity
velocity1
velocity2
Commanded position
position
Time [s]

MC_MoveSuperimposed, MCP_MoveSuperimposed – Superimposed move

Block Symbols

Licence: MOTION CONTROL



Function Description

The MC_MoveSuperimposed *and* MCP_MoveSuperimposed *blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the* MCP_ *version of the block.*

The MC_MoveSuperimposed block moves an axis to specified position as fast as possible (with respect to set limitations). Final position is specified by input parameter Distance. In case that the axis is already in motion at the moment of execution of the MC_MoveSuperimposed block, the generated values of position, velocity and acceleration are added to the values provided by the previous motion block. If there is no previous motion, the block behaves in the same way as the MC_MoveRelative command.

Note:There is no BufferMode parameter which is irrelevant in the superimposed mode. If there is already an superimposed motion running at the moment of execution, the new block is started immediately (analogous to aborting mode).

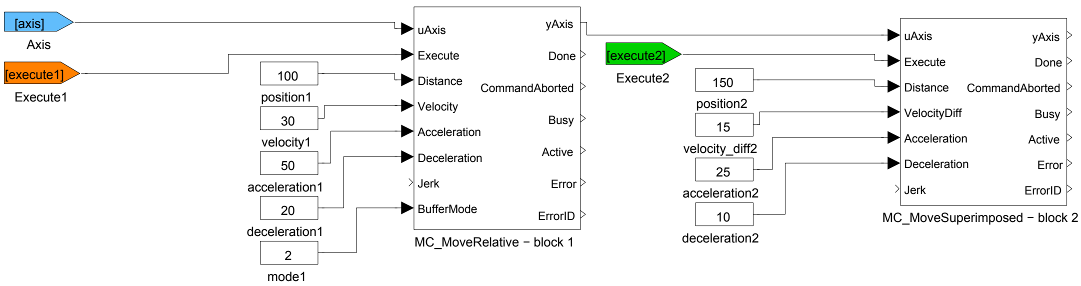
Inputs

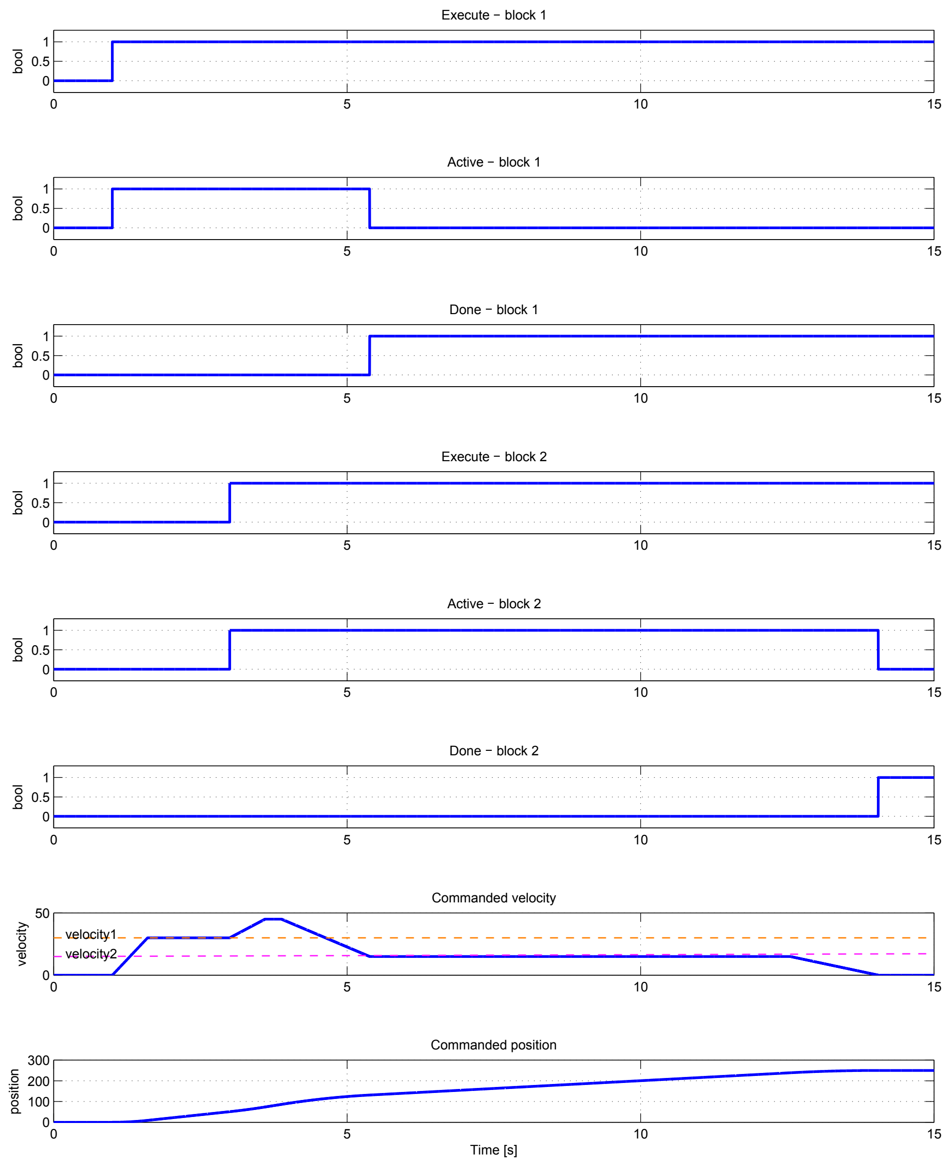
uAxis	Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool
Distance	Requested target distance (relative to execution point) [unit]	Double (F64)
VelocityDiff	Maximal allowed velocity [unit/s]	Double (F64)
Acceleration	Maximal allowed acceleration [unit/s^2]	Double (F64)
Deceleration	Maximal allowed deceleration [unit/s^2]	Double (F64)
Jerk	Maximal allowed jerk [unit/s^3]	Double (F64)

Outputs

yAxis	Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)	Reference
Done	Algorithm finished	Bool
CommandAborted	Algorithm was aborted	Bool
Busy	Algorithm not finished yet	Bool
Active	The block is controlling the axis	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation i ..... REXYGEN general error	Error

Example

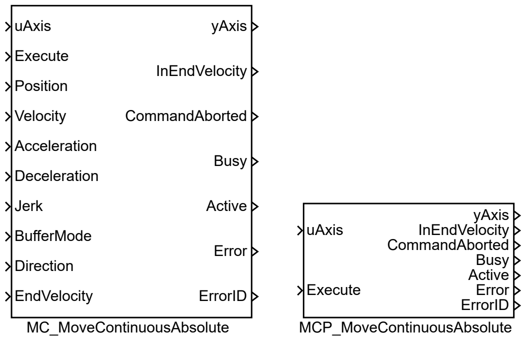

[axis]
Axis
[execute1]
Execute1
100
position1
30
velocity1
50
acceleration1
20
deceleration1
2
mode1
uAxis
Execute
Distance
Velocity
Acceleration
Deceleration
Jerk
BufferMode
yAxis
Done
CommandAborted
Busy
Active
Error
ErrorID
MC_MoveRelative − block 1
[execute2]
Execute2
150
position2
15
velocity_diff2
25
acceleration2
10
deceleration2
uAxis
Execute
Distance
VelocityDiff
Acceleration
Deceleration
Jerk
yAxis
Done
CommandAborted
Busy
Active
Error
ErrorID
MC_MoveSuperimposed − block 2


Execute − block 1
Active − block 1
Done − block 1
Execute − block 2
Active − block 2
Done − block 2
bool
Commanded velocity
velocity
velocity1
velocity2
Commanded position
position
Time [s]

MC_MoveContinuousAbsolute, MCP_MoveContinuousAbsolute – Move to position (absolute coordinate)

Block Symbols

Licence: MOTION CONTROL



Function Description

The MC_MoveContinuousAbsolute *and* MCP_MoveContinuousAbsolute *blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the* MCP_ *version of the block.*

The MC_MoveContinuousAbsolute block moves an axis to specified position as fast as possible. If no further action is pending, final velocity is specified by parameter EndVelocity. For blending purposes, start and stop velocity of this block is maximum velocity with direction respecting current and final position. If start velocity of next pending block is in opposite direction, then blending velocity is always zero.

If next pending block is executed too late in order to reach requested velocity the generated output depends on jerk setting. If no limit for jerk is used (block input Jerk is zero or unconnected) block uses maximum acceleration or deceleration to reach the desired velocity as near as possible. If jerk is limited it is not possible to say what is the nearest velocity because also acceleration is important. For this reason, the axis is stopped and moved backward and blending velocity is always reached. Although this seems to be correct solution, it might look confusing in a real situation. Therefore, it is recommended to reorganize execution order of the motion blocks and avoid this situation.

Note 1: If the EndVelocity is set to zero value, the block behaves in the same way as MC_MoveAbsolute.

Note 2: If next motion command is executed before the final position is reached, the block behaves in the same way as MC_MoveAbsolute.

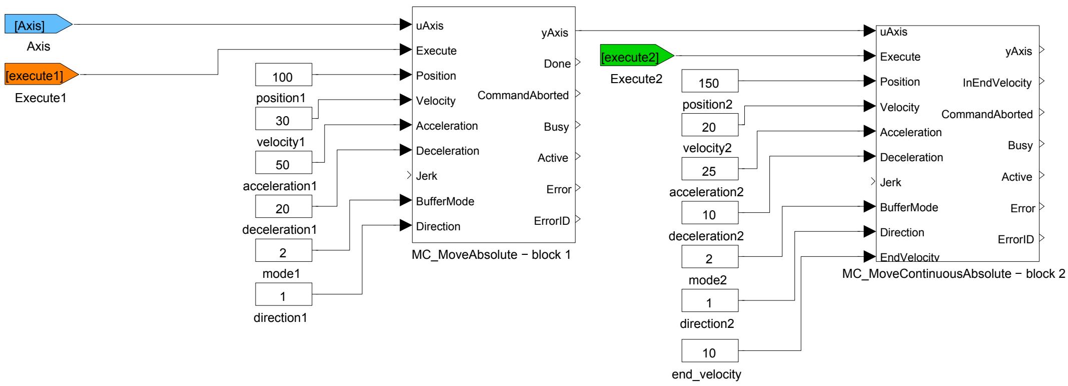
Inputs

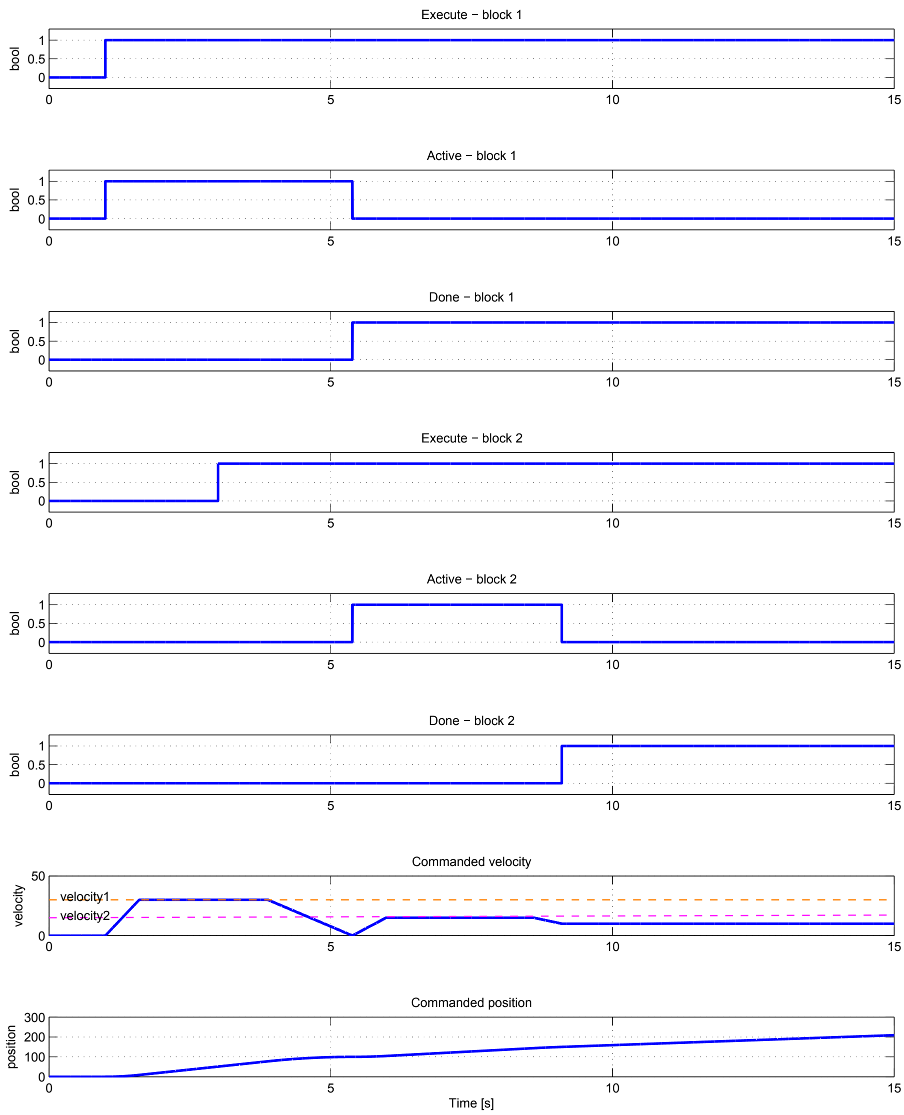
uAxis	Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool
Position	Requested target position (absolute) [unit]	Double (F64)
Velocity	Maximal allowed velocity [unit/s]	Double (F64)
Acceleration	Maximal allowed acceleration [unit/s^2]	Double (F64)
Deceleration	Maximal allowed deceleration [unit/s^2]	Double (F64)
Jerk	Maximal allowed jerk [unit/s^3]	Double (F64)
BufferMode	Buffering mode 1 ..... Aborting (start immediately) 2 ..... Buffered (start after finish of previous motion) 3 ..... Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands) 4 ..... Blending high (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands) 5 ..... Blending previous (start after finishing the previous motion, previous motion finishes with its final velocity) 6 ..... Blending next (start after finishing the previous motion, previous motion finishes with the starting velocity of the next block)	Long (I32)
Direction	Direction of movement (cyclic axis or special case only) 1 ..... Positive 2 ..... Shortest 3 ..... Negative 4 ..... Current	Long (I32)
EndVelocity	End velocity	Double (F64)

Outputs

yAxis	Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)	Reference
InEndVelocity	Algorithm finished	Bool
CommandAborted	Algorithm was aborted	Bool
Busy	Algorithm not finished yet	Bool
Active	The block is controlling the axis	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation i ..... REXYGEN general error	Error

Example

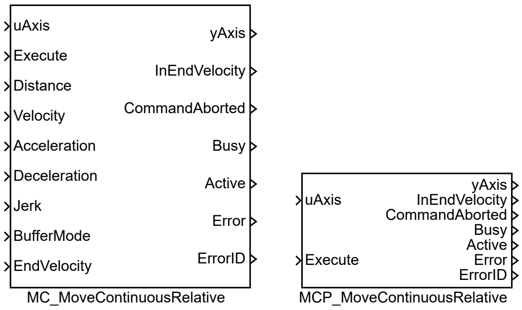

[Axis]
Axis
[execute1]
Execute1
100
position1
30
velocity1
50
acceleration1
20
deceleration1
2
mode1
1
direction1
uAxis
Execute
Position
Velocity
Acceleration
Deceleration
Jerk
BufferMode
Direction
yAxis
Done
CommandAborted
Busy
Active
Error
ErrorID
MC_MoveAbsolute – block 1
[execute2]
Execute2
150
position2
20
velocity2
25
acceleration2
10
deceleration2
2
mode2
1
direction2
10
end_velocity
uAxis
Execute
Position
Velocity
Acceleration
Deceleration
Jerk
BufferMode
Direction
EndVelocity
yAxis
InEndVelocity
CommandAborted
Busy
Active
Error
ErrorID
MC_MoveContinuousAbsolute – block 2


Execute – block 1
Active – block 1
Done – block 1
Execute – block 2
Active – block 2
Done – block 2
bool
Commanded velocity
velocity
velocity1
velocity2
Commanded position
position
Time [s]

MC_MoveContinuousRelative, MCP_MoveContinuousRelative – Move to position (relative to previous motion)

Block Symbols

Licence: MOTION CONTROL



Function Description

The MC_MoveContinuousRelative *and* MCP_MoveContinuousRelative *blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the* MCP_ *version of the block.*

The MC_MoveContinuousRelative block moves an axis to specified position as fast as possible. The final position is determined by adding the value of Distance parameter to the actual position at the moment of triggering the Execute input. If no further action is pending, final velocity is specified by parameter EndVelocity. For blending purposes, start and stop velocity of this block is maximum velocity with direction respecting current and final position. If start velocity of next pending block is in opposite direction, then blending velocity is always zero.

If next pending block is executed too late in order to reach requested velocity the generated output depends on jerk setting. If no limit for jerk is used (block input Jerk is zero or unconnected) block uses maximum acceleration or deceleration to reach the desired velocity as near as possible. If jerk is limited it is not possible to say what is the nearest velocity because also acceleration is important. For this reason, the axis is stopped and moved backward and blending velocity is always reached. Although this seems to be correct solution, it might look confusing in a real situation. Therefore, it is recommended to reorganize execution order of the motion blocks and avoid this situation.

Note 1: If the EndVelocity is set to zero value, the block behaves in the same way as MC_MoveRelative.

Note 2: If next motion command is executed before the final position is reached, the block behaves in the same way as MC_MoveRelative.

If next pending block is executed too late in order to reach requested velocity the generated output depends on jerk setting. If no limit for jerk is used (block input Jerk is zero or unconnected) block uses maximum acceleration or deceleration to reach the desired velocity as near as possible. If jerk is limited it is not possible to say what is the nearest velocity because also acceleration is important. For this reason, the axis is stopped and moved backward and blending velocity is always reached. Although this seems to be correct solution, it might look confusing in a real situation. Therefore, it is recommended to reorganize execution order of the motion blocks and avoid this situation.

If next pending block is executed too late in order to reach requested velocity the generated output depends on jerk setting. If no limit for jerk is used (block input `Jerk` is zero or unconnected) block uses maximum acceleration or deceleration to reach the desired velocity as near as possible. If jerk is limited it is not possible to say what is the nearest velocity because also acceleration is important. For this reason, the axis is stopped and moved backward and blending velocity is always reached. Although this seems to be correct solution, it might look confusing in a real situation. Therefore, it is recommended to reorganize execution order of the motion blocks and avoid this situation.

If next pending block is executed too late in order to reach requested velocity the generated output depends on jerk setting. If no limit for jerk is used (block input `Jerk` is zero or unconnected) block uses maximum acceleration or deceleration to reach the desired velocity as near as possible. If jerk is limited it is not possible to say what is the nearest velocity because also acceleration is important. For this reason, the axis is stopped and moved backward and blending velocity is always reached. Although this seems to be correct solution, it might look confusing in a real situation. Therefore, it is recommended to reorganize execution order of the motion blocks and avoid this situation.

Inputs

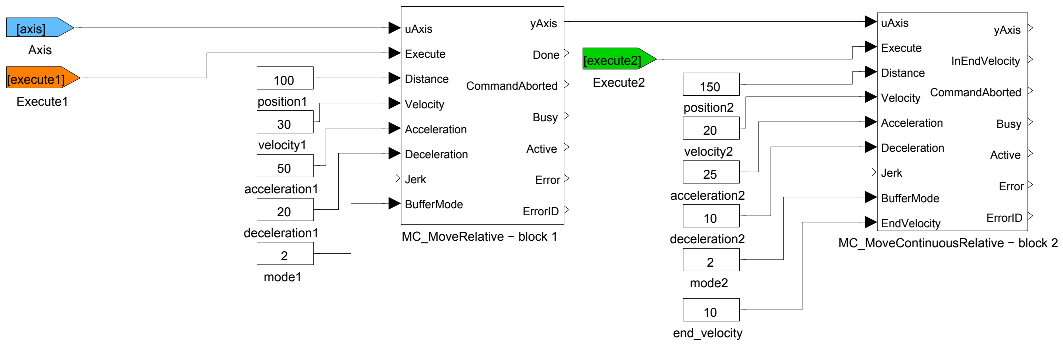
`uAxis`	Axis reference (only `RM_Axis.axisRef`–`uAxis` or `yAxis`–`uAxis` connections are allowed)	`Reference`
`Execute`	The block is activated on rising edge	`Bool`
`Distance`	Requested target distance (relative to execution point) [unit]	`Double (F64)`
`Velocity`	Maximal allowed velocity [unit/s]	`Double (F64)`
`Acceleration`	Maximal allowed acceleration [unit/s^2]	`Double (F64)`
`Deceleration`	Maximal allowed deceleration [unit/s^2]	`Double (F64)`
`Jerk`	Maximal allowed jerk [unit/s^3]	`Double (F64)`

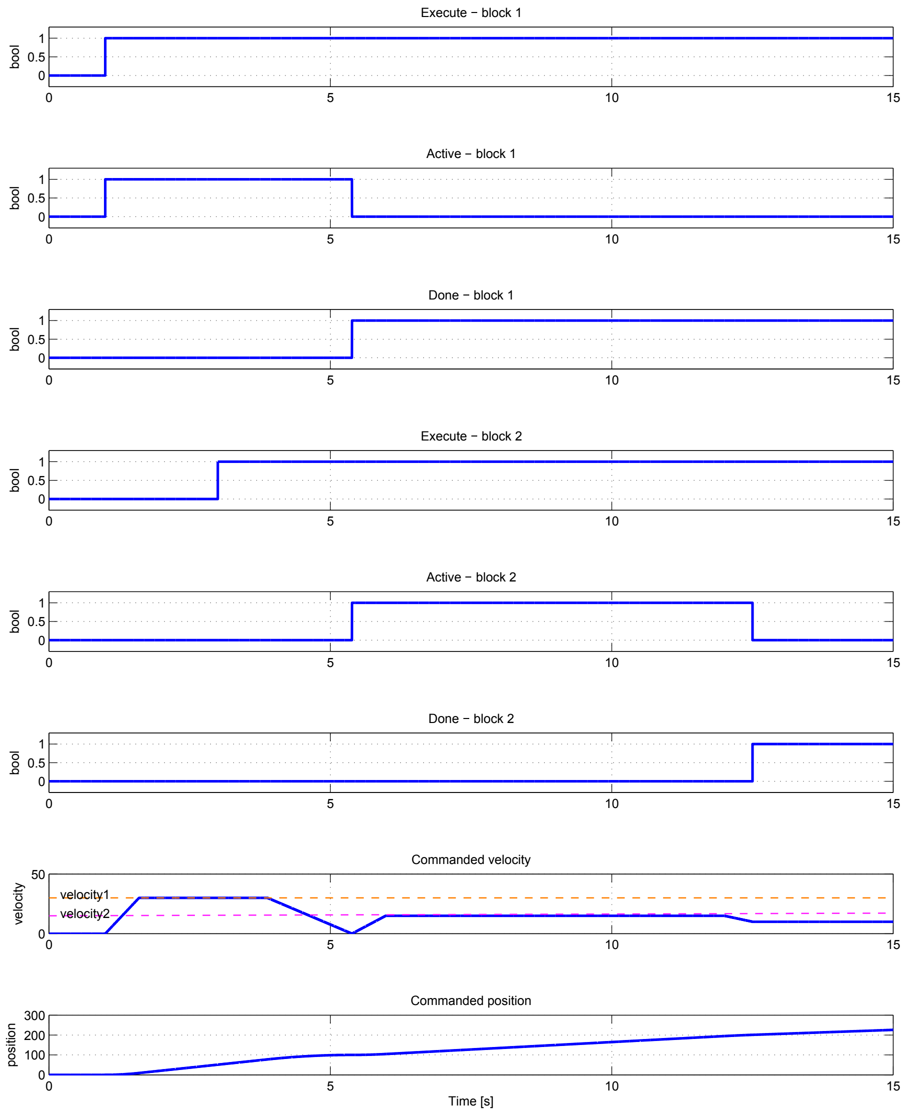
`BufferMode`	Buffering mode 1 ..... Aborting (start immediately) 2 ..... Buffered (start after finish of previous motion) 3 ..... Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands) 4 ..... Blending high (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands) 5 ..... Blending previous (start after finishing the previous motion, previous motion finishes with its final velocity) 6 ..... Blending next (start after finishing the previous motion, previous motion finishes with the starting velocity of the next block)	`Long (I32)`
`EndVelocity`	End velocity	`Long (I32)`

Outputs

`yAxis`	Axis reference (only `RM_Axis.axisRef`–`uAxis` or `yAxis`–`uAxis` connections are allowed)	`Reference`
`InEndVelocity`	PLCopen Done (algorithm finished)	`Bool`
`CommandAborted`	PLCopen CommandAborted (algorithm was aborted)	`Bool`
`Busy`	PLCopen Busy (algorithm not finished yet)	`Bool`
`Active`	PLCopen Active (the block is controlling the axis)	`Bool`
`Error`	PLCopen Error (error occurred)	`Bool`
`ErrorID`	Result of the last operation i ..... REXYGEN general error	`Error`

Example

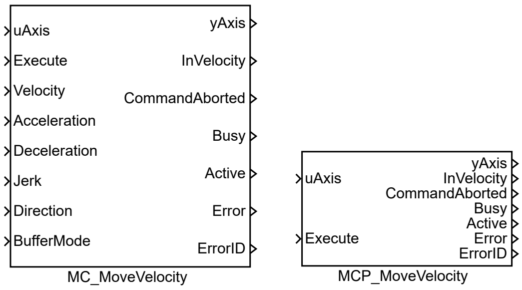

[axis]
Axis
[execute1]
Execute1
100
position1
30
velocity1
50
acceleration1
20
deceleration1
2
mode1
uAxis
Execute
Distance
Velocity
Acceleration
Deceleration
Jerk
BufferMode
yAxis
Done
CommandAborted
Busy
Active
Error
ErrorID
MC_MoveRelative – block 1
[execute2]
Execute2
150
position2
20
velocity2
25
acceleration2
10
deceleration2
2
mode2
10
end_velocity
uAxis
Execute
Distance
Velocity
Acceleration
Deceleration
Jerk
BufferMode
EndVelocity
yAxis
InEndVelocity
CommandAborted
Busy
Active
Error
ErrorID
MC_MoveContinuousRelative – block 2


Execute – block 1
Active – block 1
Done – block 1
Execute – block 2
Active – block 2
Done – block 2
bool
Commanded velocity
velocity
velocity1
velocity2
Commanded position
position
Time [s]

MC_MoveVelocity, MCP_MoveVelocity – Move with constant velocity

Block Symbols

Licence: MOTION CONTROL



Function Description

The MC_MoveVelocity *and* MCP_MoveVelocity *blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the* MCP_ *version of the block.*

The MC_MoveVelocity block changes axis velocity to specified value as fast as possible and keeps the specified velocity until the command is aborted by another block or event.
Note: parameter Direction enumerate also shortest_way although for this block it is not valid value.

Inputs

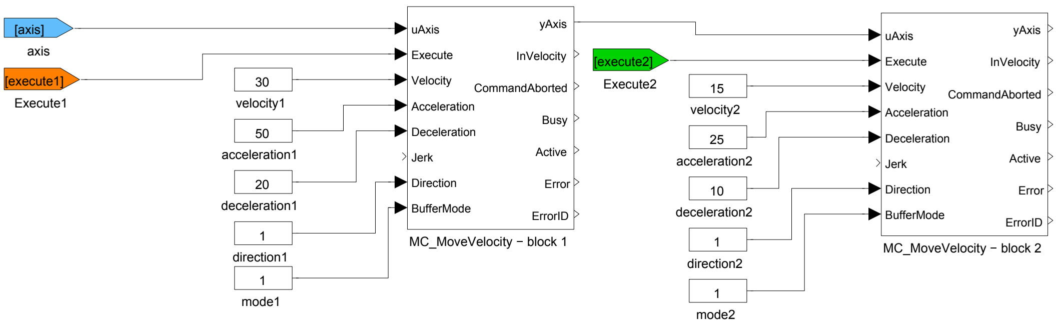
uAxis	Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool
Velocity	Maximal allowed velocity [unit/s]	Double (F64)
Acceleration	Maximal allowed acceleration unit/s^2]	Double (F64)
Deceleration	Maximal allowed deceleration [unit/s^2]	Double (F64)
Jerk	Maximal allowed jerk [unit/s^3]	Double (F64)
Direction	Direction of movement (cyclic axis or special case only) 1 ..... Positive 2 ..... Shortest 3 ..... Negative 4 ..... Current	Long (I32)

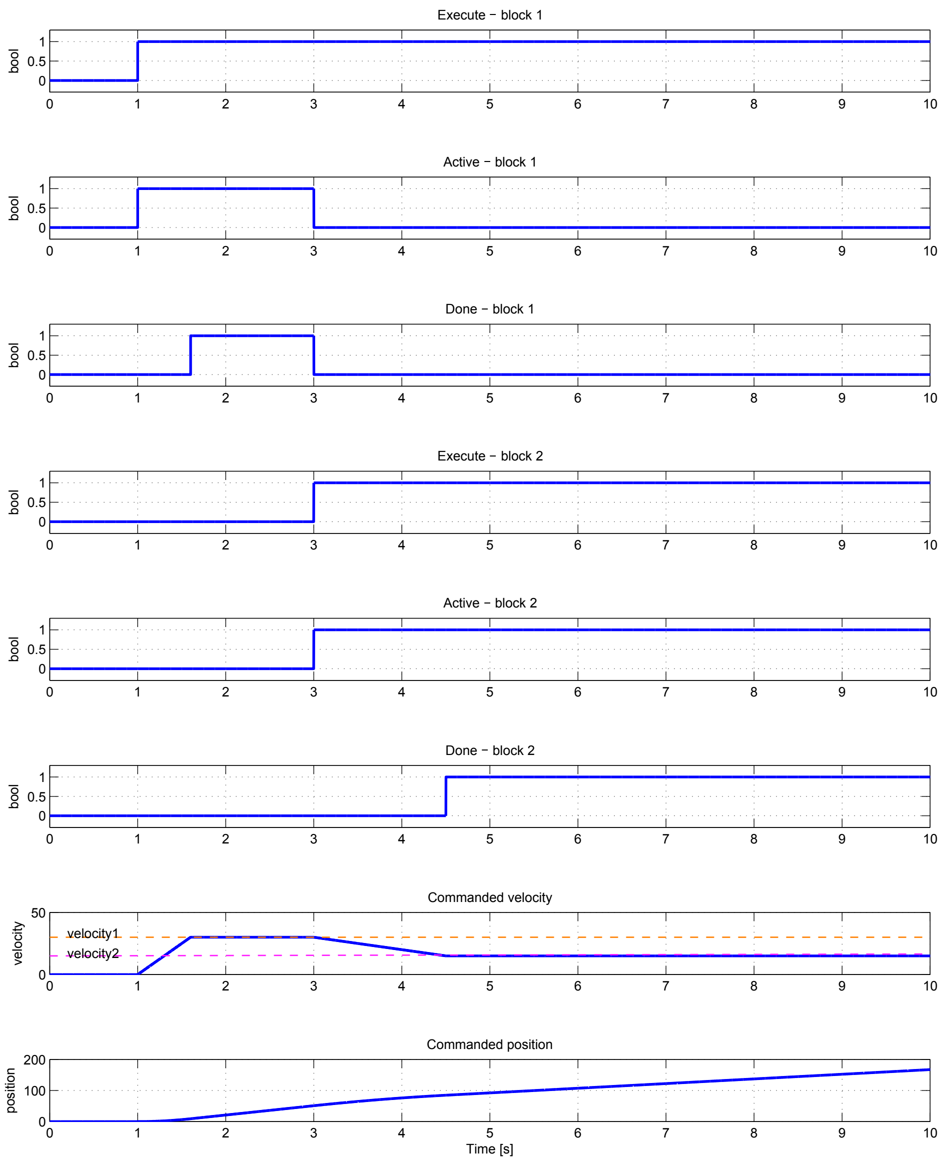
`BufferMode`	Buffering mode `1` ..... Aborting (start immediately) `2` ..... Buffered (start after finish of previous motion) `3` ..... Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands) `4` ..... Blending high (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands) `5` ..... Blending previous (start after finishing the previous motion, previous motion finishes with its final velocity) `6` ..... Blending next (start after finishing the previous motion, previous motion finishes with the starting velocity of the next block)	`Long (I32)`

Outputs

`yAxis`	Axis reference (only `RM_Axis.axisRef`–`uAxis` or `yAxis`–`uAxis` connections are allowed)	`Reference`
`InVelocity`	Requested velocity reached	`Bool`
`CommandAborted`	Algorithm was aborted	`Bool`
`Busy`	Algorithm not finished yet	`Bool`
`Active`	The block is controlling the axis	`Bool`
`Error`	Error occurred	`Bool`
`ErrorID`	Result of the last operation `i` ..... REXYGEN general error	`Error`

Example

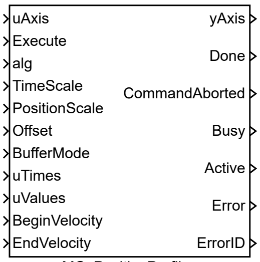

[axis]
axis
[execute1]
Execute1
30
velocity1
50
acceleration1
20
deceleration1
1
direction1
1
mode1
uAxis
Execute
Velocity
Acceleration
Deceleration
Jerk
Direction
BufferMode
yAxis
InVelocity
CommandAborted
Busy
Active
Error
ErrorID
MC_MoveVelocity − block 1
[execute2]
Execute2
15
velocity2
25
acceleration2
10
deceleration2
1
direction2
1
mode2
MC_MoveVelocity − block 2


Execute − block 1
Active − block 1
Done − block 1
Execute − block 2
Active − block 2
Done − block 2
Commanded velocity
velocity1
velocity2
Commanded position
bool
velocity
position
Time [s]

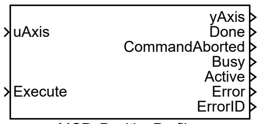
MC_PositionProfile, MCP_PositionProfile – Position profile

Block Symbols

Licence: MOTION CONTROL



MC_PositionProfile



MCP_PositionProfile

Function Description

The `MC_PositionProfile` *and* `MCP_PositionProfile` *blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the* `MCP_` *version of the block.*

The `MC_PositionProfile` block commands a time-position locked motion profile. Block implements two possibilities for definition of time-position function:

1. sequence of values: the user defines a sequence of time-position pairs. In each time interval, the values of position are interpolated. Times sequence is in array `times`, position sequence is in array values. Time sequence must be increasing and must start with zero or zero must be between the first and last point. Execution always starts from zero time, so if the sequence start with negative time, part of the profile is not executed (could be used for debugging or time shift). For `MC_VelocityProfile` and `MC_AccelerationProfile` interpolation is linear, but for `MC_PositionProfile`, 3rd order polynomial is used in order to avoid steps in velocity.

2. spline: time sequence is the same as in previous case. Each interval is interpolate byd 5th order polynomial $p(x) = a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$ where beginning of the time-interval is for $x = 0$, end of time-interval is for $x = 1$ and factors a_i are put in array `values` in ascending order (e.g. array `values` contains 6 values for each interval). This method allows smaller number of intervals and there is special editor for synthesis of the interpolating spline function.

For both case, the time sequence could be equally spaced and then array `times` includes only the first (usually zero) and last point.

Note 1: input `TimePosition` is missing, because all path data are in parameters of the block.

Note 2: parameter `values` must be set as vector in all cases, e.g. text string must not include semicolon.

Note 3: incorrect parameter `cSeg` (higher then real size of arrays `times` and/or `values`) leads to unpredictable result and in some cases crashes whole runtime execution (The problem is platform dependent and currently it is known only for SIMULINK - crash of whole MATLAB).

Note 4: in the spline mode, polynomial is always 5th order and always in position (also for sibling block `MC_VelocityProfile` and `MC_AccelerationProfile`) and it couldn't be changed. As the special editor exists, this is not important limitation.

Note 5: The block does not include ramp-in mode. If start position and/or velocity of profile is different from actual (commanded) position of axis, block fails with error `-707` (step). It is recommended to use `BufferMode=BlendingNext` to eliminate the problem with start velocity.

Inputs

`uAxis`	Axis reference (only `RM_Axis.axisRef`–`uAxis` or `yAxis`–`uAxis` connections are allowed)	`Reference`
`Execute`	The block is activated on rising edge	`Bool`
`TimeScale`	Overall scale factor in time	`Double (F64)`
`PositionScale`	Overall scale factor in value	`Double (F64)`
`Offset`	Overall profile offset in value	`Double (F64)`
`BufferMode`	Buffering mode `1` ..... Aborting (start immediately) `2` ..... Buffered (start after finish of previous motion) `3` ..... Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands) `4` ..... Blending high (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands) `5` ..... Blending previous (start after finishing the previous motion, previous motion finishes with its final velocity) `6` ..... Blending next (start after finishing the previous motion, previous motion finishes with the starting velocity of the next block)	`Long (I32)`

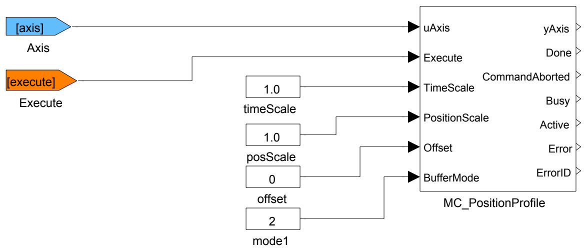
Outputs

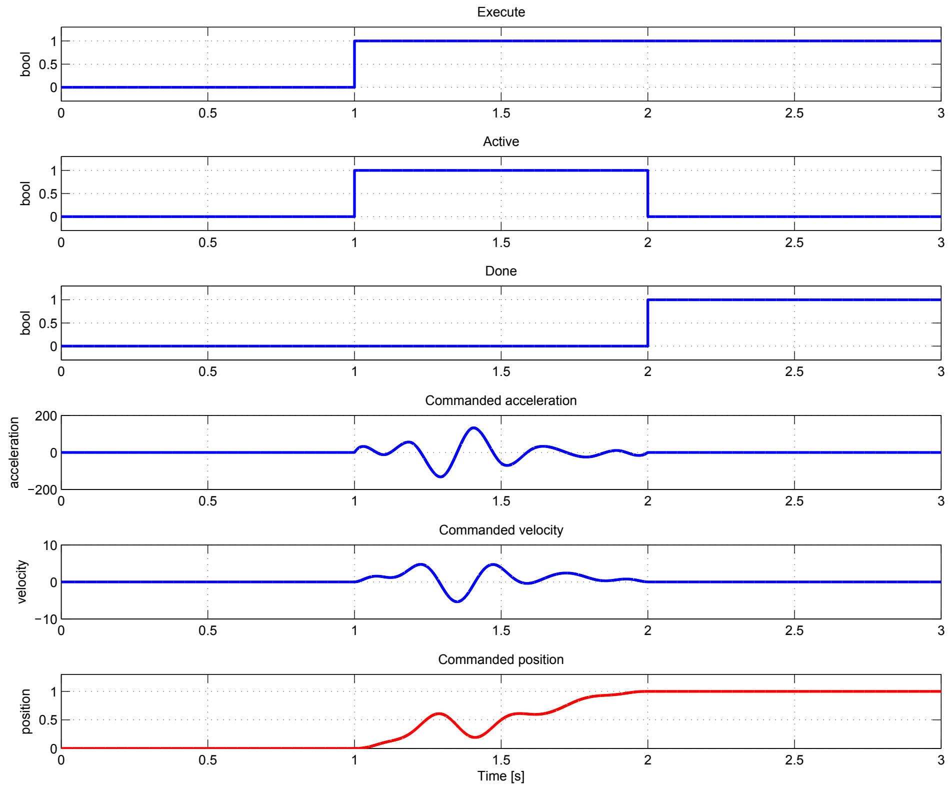
`yAxis`	Axis reference (only `RM_Axis.axisRef`–`uAxis` or `yAxis`–`uAxis` connections are allowed)	`Reference`
`Done`	Algorithm finished	`Bool`
`CommandAborted`	Algorithm was aborted	`Bool`
`Busy`	Algorithm not finished yet	`Bool`
`Active`	The block is controlling the axis	`Bool`
`Error`	Error occurred	`Bool`
`ErrorID`	Result of the last operation `i` ..... REXYGEN general error	`Error`

Parameters

`alg`	Algorithm for interpolation `1` ..... Sequence of time/value pairs `2` ..... Sequence of equidistant values `3` ..... Spline `4` ..... Equidistant spline	⊙2	`Long (I32)`
`nmax`	Number of profile segments	⊙3	`Long (I32)`
`times`	Times when segments are switched		`Reference`
`values`	Values or interpolating polynomial coefficients (a0, a1, a2, ...)		`Reference`

Example

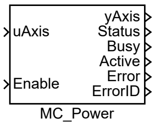

[axis]
Axis
[execute]
Execute
1.0
timeScale
1.0
posScale
0
offset
2
mode1
uAxis
Execute
TimeScale
PositionScale
Offset
BufferMode
yAxis
Done
CommandAborted
Busy
Active
Error
ErrorID
MC_PositionProfile


Execute
Active
Done
bool
Commanded acceleration
acceleration
Commanded velocity
velocity
Commanded position
position
Time [s]

MC_Power – Axis activation (power on/off)

Block Symbol

Licence: MOTION CONTROL



Function Description

The MC_Power block must be used with all axes. It is the only way to switch an axis from disable state to standstill (e.g. operation) state. The Enable input must be set (non zero value) for whole time the axis is active. The Status output can be used for switch on and switch off of the motor driver (logical signal for enabling the power stage of the drive).

The block does not implement optional parameters/inputs Enable_Positive, Enable_Negative. The same functionality can be implemented by throwing the limit switches (inputs limP and limN of block RM_Axis).

If the associated axis is turned off (by setting the Enable input to zero) while a motion is processed (commanded velocity is not zero), error stoping sequence is activated and the status is switched to off/diabled when the motion stops (commanded velocity reaches zero value).

Inputs

uAxis	Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)	Reference
Enable	Block function is enabled	Bool

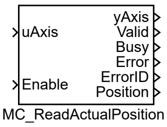
Outputs

yAxis	Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)	Reference
Status	Effective state of the power stage	Bool
Busy	Algorithm not finished yet	Bool
Active	The block is controlling the axis	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation i ..... REXYGEN general error	Error

MC_ReadActualPosition – Read actual position

Block Symbol

Licence: MOTION CONTROL



Function Description

The block `MC_ReadActualPosition` displays actual value of position of a connected axis on the output `Position`. The output is valid only while the block is enabled by the logical input signal `Enable`.

The block displays logical position value which is entered into all of the motion blocks as position input. In case that no absolute position encoder is used or the internal position is set in other way (e.g. via `MC_Home` block), the `CommandedPosition` output of the corresponding `RM_Axis` may display different value.

Inputs

`uAxis`	Axis reference (only `RM_Axis.axisRef`–`uAxis` or `yAxis`–`uAxis` connections are allowed)	`Reference`
`Enable`	Block function is enabled	`Bool`

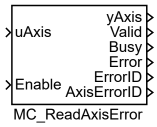
Outputs

`yAxis`	Axis reference (only `RM_Axis.axisRef`–`uAxis` or `yAxis`–`uAxis` connections are allowed)	`Reference`
`Valid`	Output value is valid	`Bool`
`Busy`	Algorithm not finished yet	`Bool`
`Error`	Error occurred	`Bool`
`ErrorID`	Result of the last operation `i` ..... REXYGEN general error	`Error`
`Position`	Actual absolute position	`Double (F64)`

MC_ReadAxisError – Read axis error

Block Symbol

Licence: MOTION CONTROL



Function Description

The block MC_ReadAxisError displays actual error code of a connected axis on the output AxisErrorID. In case of no error, the output is set to zero. The error value is valid only while the block is enabled by the logical input signal Enable. This block is implemented for sake of compatibility with PLCOpen specification as it displays duplicit information about an error which is also accessible on the ErrorID output of the RM_Axis block.

Inputs

uAxis	Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)	Reference
Enable	Block function is enabled	Bool

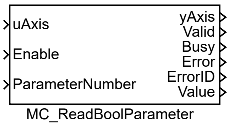
Outputs

yAxis	Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)	Reference
Valid	Output value is valid	Bool
Busy	Algorithm not finished yet	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation i ..... REXYGEN general error	Error
AxisErrorID	Result of the last operation read from axis i ..... REXYGEN general error	Error

MC_ReadBoolParameter – Read axis parameter (bool)

Block Symbol

Licence: MOTION CONTROL



Function Description

The block MC_ReadBoolParameter displays actual value of various signals related to the connected axis on its Value output. The user chooses from a set of accessible logical variables by setting the ParameterNumber input. The output value is valid only while the block is activated by the logical Enable input.

The block displays the parameters and outputs of RM_Axis block and is implemented for sake of compatibility with the PLCOpen specification.

Inputs

uAxis	Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)	Reference
Enable	Block function is enabled	Bool
ParameterNumber	Parameter ID 4 ..... Enable sw positive limit 5 ..... Enable sw negative limit 6 ..... Enable position lag monitoring	Long (I32)

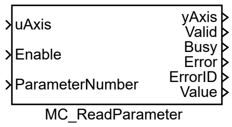
Outputs

yAxis	Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)	Reference
Valid	Output value is valid	Bool
Busy	Algorithm not finished yet	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation i ..... REXYGEN general error	Error
Value	Parameter value	Bool

MC_ReadParameter – Read axis parameter

Block Symbol

Licence: MOTION CONTROL



Function Description

The block `MC_ReadParameter` displays actual value of various system variables of the connected axis on its `Value` output. The user chooses from a set of accessible variables by setting the `ParameterNumber` input. The output value is valid only while the block is activated by the logical `Enable` input.

The block displays the parameters and outputs of `RM_Axis` block and is implemented for sake of compatibility with the `PLCOpen` specification.

Inputs

`uAxis`	Axis reference (only `RM_Axis.axisRef–uAxis` or `yAxis–uAxis` connections are allowed)	`Reference`
`Enable`	Block function is enabled	`Bool`
`ParameterNumber`	Parameter ID `1` ..... Commanded position `2` ..... Positive sw limit switch `3` ..... Negative sw limit switch `7` ..... Maximal position lag `8` ..... Maximal velocity (system) `9` ..... Maximal velocity (appl) `10` .... Actual velocity `11` .... Commanded velocity `12` .... Maximal acceleration (system) `13` .... Maximal acceleration (appl.) `14` .... Maximal deceleration (system) `15` .... Maximal deceleration (appl.) `16` .... Maximal jerk `1000` .. Actual position `1001` .. Maximal torque/force `1003` .. Actual torque/force `1004` .. Commanded torque/force	`Long (I32)`

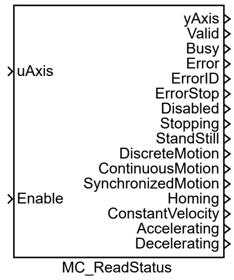
Outputs

yAxis	Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)	Reference
Valid	Output value is valid	Bool
Busy	Algorithm not finished yet	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation i ..... REXYGEN general error	Error
Value	Parameter value	Double (F64)

MC_ReadStatus – Read axis status

Block Symbol

Licence: MOTION CONTROL



Function Description

The block `MC_ReadStatus` indicates the state of the connected axis on its logical output signals. The values of the states are valid only while the `Enable` input is set to nonzero value. This state is indicated by `Valid` output.

Inputs

`uAxis`	Axis reference (only `RM_Axis.axisRef`–`uAxis` or `yAxis`–`uAxis` connections are allowed)	`Reference`
`Enable`	Block function is enabled	`Bool`

Outputs

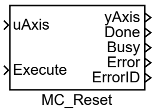
`yAxis`	Axis reference (only `RM_Axis.axisRef`–`uAxis` or `yAxis`–`uAxis` connections are allowed)	`Reference`
`Valid`	Output value is valid	`Bool`
`Busy`	Algorithm not finished yet	`Bool`
`Error`	Error occurred	`Bool`
`ErrorID`	Result of the last operation `i` ..... REXYGEN general error	`Error`
`ErrorStop`	Axis is in the ErrorStop state	`Bool`
`Disabled`	Axis is in the Disabled state	`Bool`
`Stopping`	Axis is in the Stoping state	`Bool`
`StandStill`	Axis is in the StandStill state	`Bool`
`DiscreteMotion`	Axis is in the DiscreteMotion state	`Bool`
`ContinuousMotion`	Axis is in the ContinuousMotion state	`Bool`
`SynchronizedMotion`	Axis is in the SynchronizedMotion state	`Bool`
`Homing`	Axis is in the Homing state	`Bool`

`ConstantVelocity`	Axis is moving with constant velocity	`Bool`
`Accelerating`	Axis is accelerating	`Bool`
`Decelerating`	Axis is decelerating	`Bool`

MC_Reset – **Reset axis errors**

Block Symbol

Licence: MOTION CONTROL



Function Description

The MC_Reset block makes the transition from the state ErrorStop to StandStill by resetting all internal axis-related errors.

Inputs

uAxis	Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool

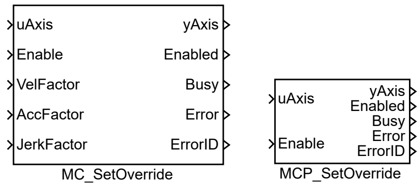
Outputs

yAxis	Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)	Reference
Done	Algorithm finished	Bool
Busy	Algorithm not finished yet	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation i ..... REXYGEN general error	Error

MC_SetOverride, MCP_SetOverride – Set override factors

Block Symbols

Licence: MOTION CONTROL



Function Description

The MC_SetOverride *and* MCP_SetOverride *blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the* MCP_ *version of the block.*

The MC_SetOverride block sets the values of override for the whole axis, and all functions that are working on that axis. The override parameters act as a factor that is multiplied to the commanded velocity, acceleration, deceleration and jerk of the move function block.

This block is level-sensitive (not edge-sensitive like other motion control blocks). So factors are update in each step while input Enable is not zero. It leads to reacalculation of movement's path if a block like MC_MoveAbsolute commands the axis. This recalculation needs lot of CPU time and also numerical problem could appear. For this reasons, a deadband (parameter diff) is established. The movement's path recalculation is proceeded only if one of the factors is changed more then the deadband.

Note: all factor must be positive. Factor greater then 1.0 are possible, but often lead to overshooting of axis limits and failure of movement (with errorID=-700 - invalid parameter; if factor is set before start of block) or error stop of axis (with errorID=-701 - out of range; if factor is changed within movement and actual value overshoot limit).

Inputs

uAxis	Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)	Reference
Enable	Block function is enabled	Bool
VelFactor	Velocity multiplication factor	Double (F64)
AccFactor	Acceleration/deceleration multiplication factor	Double (F64)
JerkFactor	Jerk multiplication factor	Double (F64)

Outputs

yAxis	Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)	Reference
Enabled	Block function is enabled	Bool
Busy	Algorithm not finished yet	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation i ..... REXYGEN general error	Error

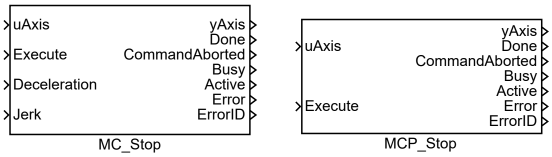
Parameter

diff	Deadband (difference for recalculation)	↓0.0 ↑1.0 ⊙0.1	Double (F64)

`MC_Stop`, `MCP_Stop` – **Stopping a movement**

Block Symbols

Licence: `MOTION CONTROL`



Function Description

The `MC_Stop` *and* `MCP_Stop` *blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the* `MCP_` *version of the block.*

The `MC_Stop` block commands a controlled motion stop and transfers the axis to the state `Stopping`. It aborts any ongoing Function Block execution. While the axis is in state Stopping, no other FB can perform any motion on the same axis. After the axis has reached velocity zero, the `Done` output is set to `true` immediately. The axis remains in the state `Stopping` as long as Execute is still `true` or velocity zero is not yet reached. As soon as `Done=true` and `Execute=false` the axis goes to state `StandStill`.

Note 1: parameter/input `BufferMode` is not supported. Mode is always `Aborting`.

Note 2: Failing stop-command could be dangerous. This block does not generate invalid-parameter-error but tries to stop the axis anyway (e.g. uses parameteres from `RM_Axis` or generates error-stop-sequence).

Inputs

`uAxis`	Axis reference (only `RM_Axis.axisRef`–`uAxis` or `yAxis`–`uAxis` connections are allowed	`Reference`
`Execute`	The block is activated on rising edge	`Bool`
`Deceleration`	Maximal allowed deceleration [unit/s^2]	`Double (F64)`
`Jerk`	Maximal allowed jerk [unit/s^3]	`Double (F64)`

Outputs

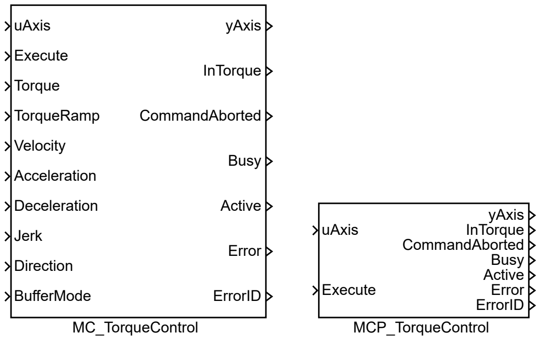
`yAxis`	Axis reference (only `RM_Axis.axisRef`–`uAxis` or `yAxis`–`uAxis` connections are allowed	`Reference`
`Done`	Algorithm finished	`Bool`
`CommandAborted`	Algorithm was aborted	`Bool`
`Busy`	Algorithm not finished yet	`Bool`
`Active`	The block is controlling the axis	`Bool`
`Error`	Error occurred	`Bool`

`ErrorID`	Result of the last operation `i` ..... REXYGEN general error	`Error`

MC_TorqueControl, MCP_TorqueControl – **Torque/force control**

Block Symbols

Licence: MOTION CONTROL



Function Description

The MC_TorqueControl *and* MCP_TorqueControl *blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the* MCP_ *version of the block.*

The MCP_TorqueControl block generates constant slope torque/force ramp until maximum requested value has been reached. Similar profile is generated for velocity. The motion trajectory is limited by maximum velocity, acceleration / deceleration, and jerk, or by the value of the torque, depending on the mechanical circumstances.

Inputs

uAxis	Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool
Torque	Maximal allowed torque/force	Double (F64)
TorqueRamp	Maximal allowed torque/force ramp	Double (F64)
Velocity	Maximal allowed velocity [unit/s]	Double (F64)
Acceleration	Maximal allowed acceleration [unit/s^2]	Double (F64)
Deceleration	Maximal allowed deceleration [uunit/s^2]	Double (F64)
Jerk	Maximal allowed jerk [unit/s^3]	Double (F64)
Direction	Direction of movement (cyclic axis or special case only) 1 ..... Positive 2 ..... Shortest 3 ..... Negative 4 ..... Current	Long (I32)

BufferMode	Buffering mode 1 ..... Aborting (start immediately) 2 ..... Buffered (start after finish of previous motion) 3 ..... Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands) 4 ..... Blending high (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands) 5 ..... Blending previous (start after finishing the previous motion, previous motion finishes with its final velocity) 6 ..... Blending next (start after finishing the previous motion, previous motion finishes with the starting velocity of the next block)	Long (I32)

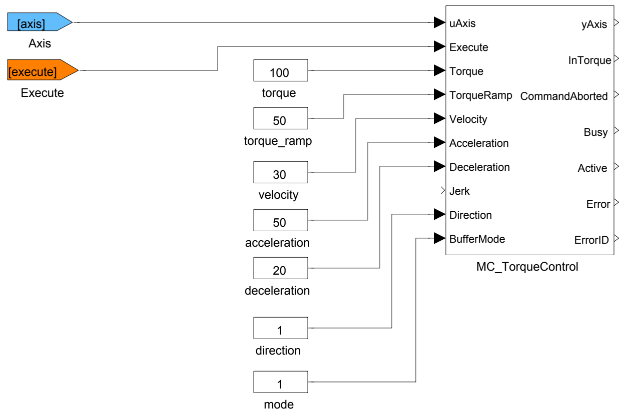
Outputs

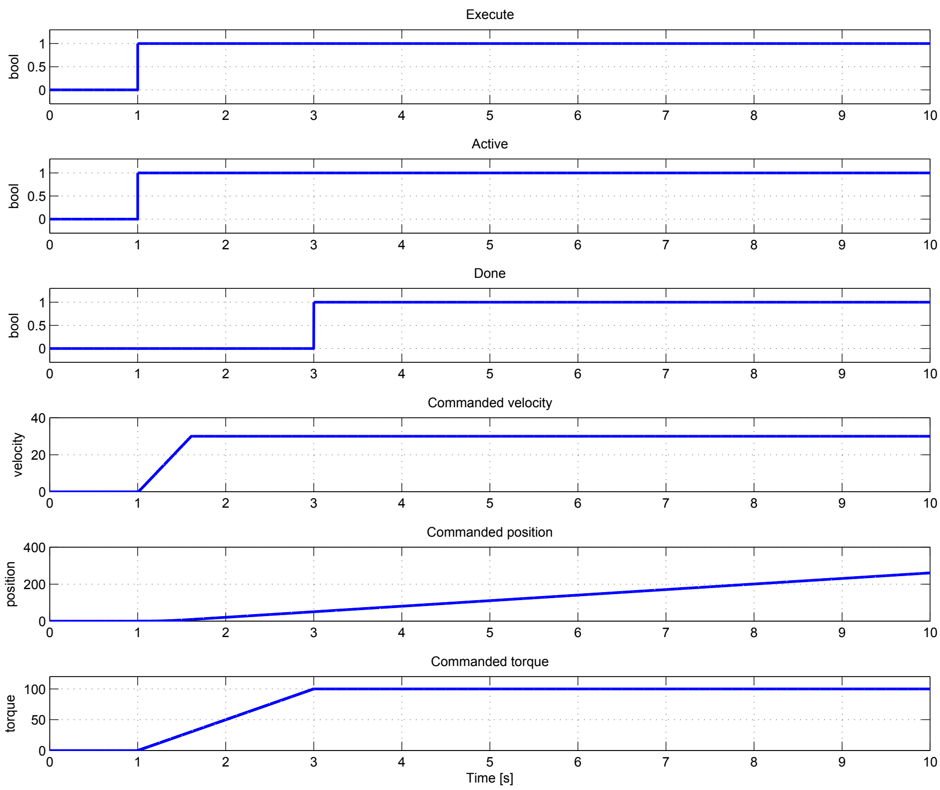
yAxis	Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)	Reference
InTorque	Requested torque/force is reached	Bool
CommandAborted	Algorithm was aborted	Bool
Busy	Algorithm not finished yet	Bool
Active	The block is controlling the axis	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation i ..... REXYGEN general error	Error

Parameter

kma	Torque/force to acceleration ratio	Double (F64)

Example

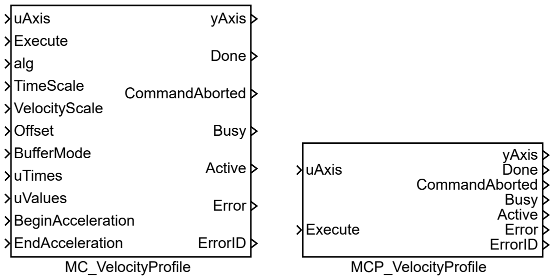

[axis]
Axis
[execute]
Execute
100
torque
50
torque_ramp
30
velocity
50
acceleration
20
deceleration
1
direction
1
mode
uAxis
Execute
Torque
TorqueRamp
Velocity
Acceleration
Deceleration
Jerk
Direction
BufferMode
yAxis
InTorque
CommandAborted
Busy
Active
Error
ErrorID
MC_TorqueControl


Execute
Active
Done
Commanded velocity
Commanded position
Commanded torque
bool
velocity
position
torque
Time [s]

MC_VelocityProfile, MCP_VelocityProfile – Velocity profile

Block Symbols

Licence: MOTION CONTROL



Function Description

The MC_PositionProfile block commands a time-position locked motion profile. Block implements two possibilities for definition of time-velocity function:

1. sequence of values: the user defines a sequence of time-velocity pairs. In each time interval, the values of velocity are interpolated. Times sequence is in array times, position sequence is in array values. Time sequence must be increasing and must start with zero or zero must be between the first and last point. Execution always starts from zero time, so if the sequence start with negative time, part of the profile is not executed (could be used for debugging or time shift). For MC_VelocityProfile and MC_AccelerationProfile interpolation is linear, but for MC_PositionProfile, 3rd order polynomial is used in order to avoid steps in velocity.

2. spline: time sequence is the same as in previous case. Each interval is interpolated by 5th order polynomial $p(x) = a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$ where beginning of the time-interval is for $x = 0$, end of time-interval is for $x = 1$ and factors a_i are put in array values in ascending order (e.g. array values contains 6 values for each interval). This method allows smaller number of intervals and there is special editor for synthesis of the interpolating spline function.

For both case, the time sequence could be equally spaced and then array times includes only the first (usually zero) and last point.

Note 1: input TimePosition is missing, because all path data are in parameters of the block.

Note 2: parameter values must be set as vector in all cases, e.g. text string must not include semicolon.

Note 3: incorrect parameter cSeg (higher then real size of arrays times and/or values) leads to unpredictable result and in some cases crashes whole runtime execution (The problem is platform dependent and currently it is known only for SIMULINK - crash of whole MATLAB).

Note 4: in the spline mode, polynomial is always 5th order and always in position (also for sibling block MC_PositionProfile and MC_AccelerationProfile) and it couldn't be changed. As the special editor exists, this is not important limitation.

Note 5: The block does not include ramp-in mode. If start position and/or velocity of profile is different from actual (commanded) position of axis, block fails with error -707 (step). It is recommended to use BufferMode=BlendingNext to eliminate the problem with start velocity.

Inputs

uAxis	Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool
TimeScale	Overall scale factor in time	Double (F64)
VelocityScale	Overall scale factor in value	Double (F64)
Offset	Overall profile offset in value	Double (F64)
BufferMode	Buffering mode 1 ..... Aborting (start immediately) 2 ..... Buffered (start after finish of previous motion) 3 ..... Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands) 4 ..... Blending high (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands) 5 ..... Blending previous (start after finishing the previous motion, previous motion finishes with its final velocity) 6 ..... Blending next (start after finishing the previous motion, previous motion finishes with the starting velocity of the next block)	Long (I32)

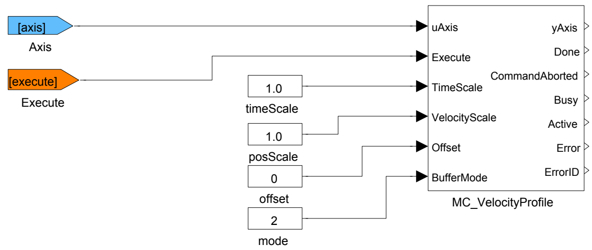
Outputs

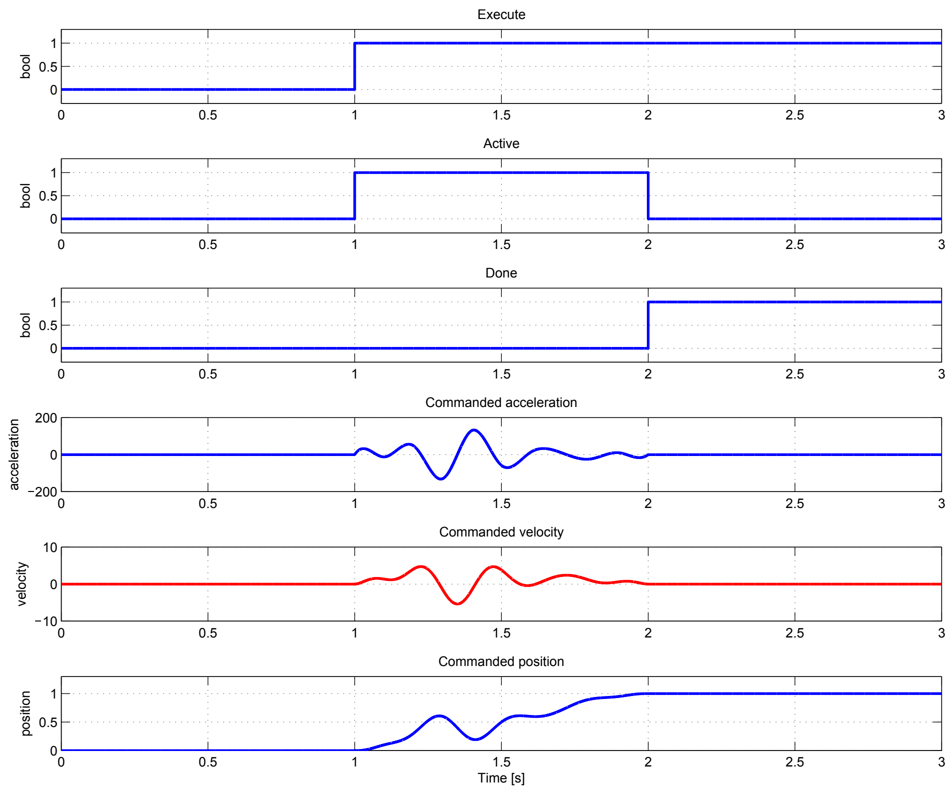
yAxis	Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)	Reference
Done	Algorithm finished	Bool
CommandAborted	Algorithm was aborted	Bool
Busy	Algorithm not finished yet	Bool
Active	The block is controlling the axis	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation i ..... REXYGEN general error	Error

Parameters

`alg`	Algorithm for interpolation `1` ..... Sequence of time/value pairs `2` ..... Sequence of equidistant values `3` ..... Spline `4` ..... Equidistant spline	⊙2	`Long (I32)`
`nmax`	Number of profile segments	⊙3	`Long (I32)`
`times`	Times when segments are switched		`Reference`
`values`	Values or interpolating polynomial coefficients (a0, a1, a2, ...)		`Reference`

Example

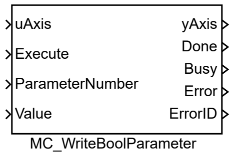

[axis]
Axis
[execute]
Execute
1.0
timeScale
1.0
posScale
0
offset
2
mode
uAxis
Execute
TimeScale
VelocityScale
Offset
BufferMode
yAxis
Done
CommandAborted
Busy
Active
Error
ErrorID
MC_VelocityProfile


Execute
Active
Done
bool
Commanded acceleration
acceleration
Commanded velocity
velocity
Commanded position
position
Time [s]

MC_WriteBoolParameter – Write axis parameter (bool)

Block Symbol

Licence: MOTION CONTROL



Function Description

The block `MC_WriteBoolParameter` writes desired value of various system parameters entered on its `Value` input to the connected axis. The user chooses from a set of accessible logical variables by setting the `ParameterNumber` input.

The block is implemented for sake of compatibility with the `PLCOpen` specification as the parameters can be written by the `SETPB` block.

Inputs

`uAxis`	Axis reference (only `RM_Axis.axisRef`–`uAxis` or `yAxis`–`uAxis` connections are allowed)	`Reference`
`Execute`	The block is activated on rising edge	`Bool`
`ParameterNumber`	Parameter ID `4` ..... Enable sw positive limit `5` ..... Enable sw negative limit `6` ..... Enable position lag monitoring	`Long (I32)`
`Value`	Parameter value	`Bool`

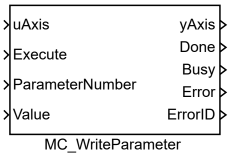
Outputs

`yAxis`	Axis reference (only `RM_Axis.axisRef`–`uAxis` or `yAxis`–`uAxis` connections are allowed)	`Reference`
`Done`	Algorithm finished	`Bool`
`Busy`	Algorithm not finished yet	`Bool`
`Error`	Error occurred	`Bool`
`ErrorID`	Result of the last operation `i` ..... REXYGEN general error	`Error`

MC_WriteParameter – Write axis parameter

Block Symbol

Licence: MOTION CONTROL



Function Description

The block MC_WriteParameter writes desired value of various system parameters entered on its Value input to the connected axis. The user chooses from a set of accessible variables by setting the ParameterNumber input.

The block is implemented for sake of compatibility with the PLCOpen specification as the parameters can be written by the SETPR block.

Inputs

uAxis	Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool
ParameterNumber	Parameter ID 2 ..... Positive sw limit switch 3 ..... Negative sw limit switch 7 ..... Maximal position lag 8 ..... Maximal velocity (system) 9 ..... Maximal velocity (appl) 13 .... Maximal acceleration (appl.) 15 .... Maximal deceleration (appl.) 16 .... Maximal jerk 1001 .. Maximal torque/force	Long (I32)
Value	Parameter value	Double (F64)

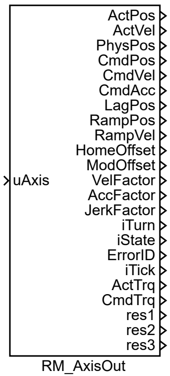
Outputs

yAxis	Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)	Reference
Done	Algorithm finished	Bool
Busy	Algorithm not finished yet	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation i ..... REXYGEN general error	Error

RM_AxisOut – Axis output

Block Symbol

Licence: MOTION CONTROL



Function Description

The RM_AxisOut block allows an access to important states of block RM_Axis. Same outputs are also available directly on RM_Axis (some of them), but this direct output is one step delayed. Blocks are ordered for execution by flow of a signal, so RM_Axis is first then all motion blocks (that actualize RM_Axis state), then RM_AxisOut (should be last) and finally waiting for next period.

Note: almost all blocks do not work with torque so commanded torque is 0. Commanded acceleration and torque should be used as feed-forward value for position/velocity controller so this value does not make any problem.

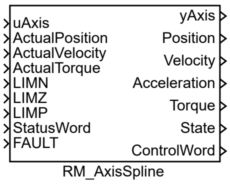
Inputs

uAxis	axis reference that must be connected to **axisRef** of the RM_Axis block (direct or indirect throw output **yAxis** of some other block)	**Reference**

RM_AxisSpline – **Commanded values interpolation**

Block Symbol

Licence: MOTION CONTROL



Function Description

The purpose of the block is to connect a virtual axis (represented by the RM_Axis block) to the motor or rather the servo drive and transform virtual axis into physical one. It includes some independent functions that are covered by this block.

The block has commanded and actual (feedback) signals to connect feedback controller. It includes inputs ActualPosition, ActualVelocity, ActualTorque and outputs Position, Velocity, Acceleration, Torque.

The feedback controller or servo drive usually works with different units (position unit is usually in encoder's tick that is transformed by gear ratio). The RM_AxisSpline block transforms drive unit into axis logical unit. The function is controlled by the DriveUnits and AxisUnits parameters.

The servo drive often uses integer types for compute or communicate position, velocity and torque. Position can overflow range of integer value when motor is running one direction long time. The RM_AxisSpline block expects this situation and set correct position if feedback signal overflow from maximum integer value to minimal integer value. This feature is controlled by the DriveBits and must be also supported by the servo drive to work correctly.

The servo drive has different working state and operation mode and require some sequence to switch into operation mode where motor follow requested position. The most common standard for the mode and sequencing is CiA402. The RM_AxisSpline block support the CiA402 standard by the StatusWord input, the ControlWord output and the DriveMode, DriveTimeout parameters. The servo drive must be set to Cyclic Synchronous Position Mode (or mode with similar functionality). There is also possible to use Velocity Mode, but position loop regulator must be realized in control system (typically by a PIDU block).

There are a lot of motion control blocks which implement complicated algorithms so they require bigger sampling period (typical update rate is from 10 to 200 ms). On the other side, the motor driver usually requires small sampling period for smooth/waveless movement. The RM_AxisSpline block solves this problem of multirate execution of motion planning and motion control levels. The block can run in another task than other motion control blocks with highest possible sampling period. It interpolates commanded position, velocity, acceleration and torque and generates smooth curve which is more suited for motor driver controllers.

There are many possibilities how to compute position (and velocity, acceleration, torque) between sampled points by slower task. This could be chosen by the `InterpolationMode` parameter, but torque is interpolated always by linear function. Most simple is linear interpolation, but it leads to steps in velocity. Better possibility is higher order polynom (e.g. 3th or 5th order). It generates a smooth curve, but leads to a huge acceleration if the original trajectory isn't the same polynomial. Drawback of polynomial interpolation could be solved by Bspline approximation, but it requires more samples and therefore bigger delay. Some original position values can be changed with this method.

Note 1: Because the execution time of motion blocks is varying in time, the block uses one or two step prediction for interpolation depending on actual conditions and timing of the motion blocks in slower tasks. The use of predicted values is signalized by states `Run1, Run2, Run3`.

Note 2: The interpolation functionality requires to put the block into different (faster) task than `RM_Axis`. For this reason, the block `RM_AxisSpline` has an internally safe solution for connecting axis references by the block `Inport` and `Outport` between different tasks.

Input

`uAxis`	Axis reference (only `RM_Axis.axisRef`–`uAxis` or `yAxis`–`uAxis` connections are allowed)	`Reference`
`ActualPosition`	Current position of the axis (feedback) [drive unit]	`Double (F64)`
`ActualVelocity`	Current velocity of the axis (feedback) [drive unit/s]	`Double (F64)`
`ActualTorque`	Current torque in the axis (feedback)	`Double (F64)`
`LIMN`	Limit switch in negative direction	`Bool`
`LIMZ`	Absolute switch or reference pulse for homing	`Bool`
`LIMP`	Limit switch in positive direction	`Bool`
`StatusWord`	Status register for drive control according CiA402 specification	`Long (I32)`
`FAULT`	External fault signal	`Bool`

Outputs

`yAxis`	Axis reference (only `RM_Axis.axisRef`–`uAxis` or `yAxis`–`uAxis` connections are allowed)	`Reference`
`Position`	Commanded interpolated position [drive unit]	`Double (F64)`
`Velocity`	Commanded interpolated velocity [drive unit/s]	`Double (F64)`
`Acceleration`	Commanded interpolated acceleration [drive unit/s/s]	`Double (F64)`
`Torque`	Commanded interpolated torque/force	`Double (F64)`

`State`	Interpolator state/error `0` ..... Off (interpolator is disabled, actual data put on output) `1` ..... Wait (not enough data in buffer, waiting) `2` ..... Run1 (interpolator running, data from first buffered interval) `3` ..... Run2 (interpolator running, data from second buffered interval) `4` ..... Run3 (interpolator running, data from third buffered interval) `-1` .... Overflow (interpolation buffer overflow, the interpolation restarts automatically, but a bump in output values may occur) `-2` .... Underflow (interpolation buffer underflow, the interpolation restarts automatically, but a bump in output values may occur) `-3` .... Busy (data from `RM_Axis` cannot be read consistently, it usually indicates, that some task is overloaded) `-4` .... Slow (the task with `RM_AxisSpline` has longer period then a task with `RM_Axis`)	`Long (I32)`
`ControlWord`	Control register for drive control according CiA402 specification	`Long (I32)`

Parameters

`InterpolationMode` Algorithm for interpolation ⊙2 `Long (I32)`

- `1` ..... linear (position is interpolated by linear function, velocity is derivation of position, e. g. step function, acceleration is zero)
- `2` ..... cubic spline (position is cubic polynom computed from begin and end position and velocity, velocity is derivation of position, acceleration is derivation of velocity)
- `3` ..... quintic spline (position is cubic polynom computed from begin and end position and velocity and acceleration, velocity is derivation of position, acceleration is derivation of velocity)
- `4` ..... cubic Bspline approximation (position is cubic polynom computed from 2 position before and 2 position after current time interval, position is approximation, e.g. values in defined points not necessary same, velocity is derivation of position, acceleration is derivation of velocity)
- `5` ..... quintic Bspline approximation (position is cubic polynom computed from 2 position before and 2 position after current time interval, position is approximation, e.g. values in defined points not necessary same, velocity is derivation of position, acceleration is derivation of velocity)
- `6` ..... all linear (each output is interpolated by linear function independently, e.g. velocity is not derivation of position, acceleration is not derivation of velocity)
- `7` ..... all cubic (each output is interpolated by cubic polynom independently, e.g. velocity is not derivation of position)
- `8` ..... reserved for future use
- `9` ..... reserved for future use

`ReverseLimit` Invert meaning of LIMN, LIMZ and LIMP inputs `Bool`

`InterpolationMode` Drive control mode ⊙2 `Long (I32)`

- `1` ..... Simplified CiA402 (only basic check of `StatusWord`, e.g. fault bit only, and direct switching of `ControlWord`, e.g. without sequencing)
- `2` ..... Strict CiA402 (full check of `StatusWord` in each state and full sequencing of `ControlWord`)

`DriveTimeout` Drive control response timeout [s] (for Strict CiA402 mode only) `Double (F64)`

`DriveBits` number of valid bits (negative value means signed number) in the `Position` output and the `ActualPosition` input ↓-64 ↑63 ⊙-32 `Long (I32)`

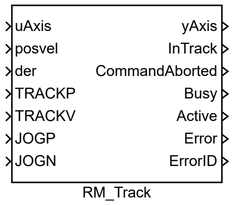
`DriveUnits` Distance in drive units for position transformation (value correspond to `AxisUnits`) `Double (F64)`

`AxisUnits`	Distance in axis units for position transformation (value correspond to `DriveUnits`)	`Double (F64)`
`VelocityCalculate`	if checked, the input `ActualVelocity` is ignored and velocity is calculated by actual position difference	`Bool`

RM_Track – Tracking and inching

Block Symbol

Licence: MOTION CONTROL



Function Description

The RM_Track block includes few useful functions.

If the input TRACK is active (not zero), the block tries to track requested position (input pos) with respect to the limits for velocity, acceleration/decelertation and jerk. The block expects that requested position is changed in each step and therefore recalculates the path in each step. This is difference to MC_MoveAbsolute block, which does not allow to change target position while the movement is not finished. This mode is useful if position is generated out of the motion control subsystem, even thought the MC_PositionProfile block is better if whole path is known.

If the input JOGP is active (not zero), the block works like the MC_MoveVelocity block (e.g. moves axis with velocity given by parameter pv in positive direction with respect to maximum acceleration and jerk). When input JOGP is released (switched to zero), the block activates stopping sequence and releases the axis when the sequence is finished. This mode is useful for jogging (e.g. setting of position of axis by an operator using up/down buttons).

Input JOGN works like JOGP, but direction is negative.

Note 1: This block hasn't parameter BufferMode. Mode is always aborting.

Note 2: If more functions are selected, only the first one is activated. Order is TRACK, JOGP, JOGN. Simultaneous activation of more than one function is not recommended.

Inputs

uAxis	Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)	Reference
posvel	Requested target position or velocity [unit]	Double (F64)
TRACKP	Position tracking mode	Bool
TRACKV	Velocity tracking mode	Bool
JOGP	Moving positive direction mode	Bool
JOGN	Moving negative direction mode	Bool

Parameters

pv	Maximal allowed velocity [unit/s]		Double (F64)
pa	Maximal allowed acceleration [unit/s^2]		Double (F64)
pd	Maximal allowed deceleration [unit/s^2]		Double (F64)
pj	Maximal allowed jerk [unit/s^3]		Double (F64)
iLen	Length of buffer for estimation	⊙10	Long (I32)

Outputs

yAxis	Axis reference (only RM_Axis.axisRef–uAxis or yAxis–uAxis connections are allowed)	Reference
InTrack	Requested position is reached	Bool
CommandAborted	Algorithm was aborted	Bool
Busy	Algorithm not finished yet	Bool
Active	The block is controlling the axis	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation i ..... REXYGEN general error	Error

Chapter 21

MC_MULTI – Motion control - multi axis blocks

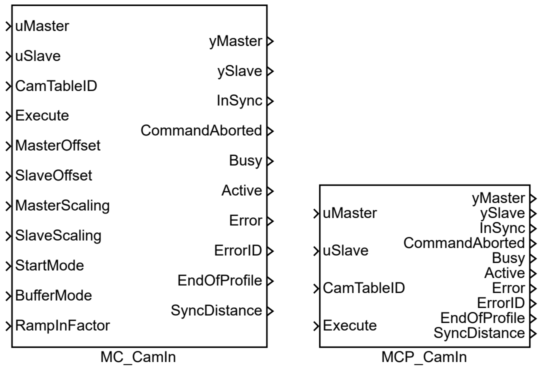
Contents

This block set is the second part of motion control blocks library according to the PLCopen standard for multi axis control. General vendor specific rules are the same as described in chapter 20 (the MC_SINGLE library, blocks for single axis motion control).

MC_CamIn, MCP_CamIn – Engage the cam

Block Symbols

Licence: MOTION CONTROL



Function Description

The MC_CamIn *and* MCP_CamIn *blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the* MCP_ *version of the block.*

The MC_CamIn block switches on a mode in which the slave axis is commanded to position which corresponds to the position of master axis transformed with with a function defined by the MCP_CamTableSelect block (connected to CamTableID input). Denoting the transformation as $Cam(x)$, master axis position $PosM$ and slave axis position $PosS$, we obtain (for absolute relationship, without phasing): $PosS = Cam((PosM - MasterOffset)/MasterScaling) * SlaveScaling + SlaveOffset$. This form of synchronized motion of the slave axis is called electronic cam.

The cam mode is switched off by executing other motion block on slave axis with mode aborting or by executing a MC_CamOut block. The cam mode is also finished when the master axis leaves a non-periodic cam profile. This situation is indicated by the EndOfProfile output.

In case of a difference between real position and/or velocity of slave axis and cam-profile slave axis position and velocity, some transient trajectory must be generated to cancel this offset. This mode is called ramp-in. The ramp-in function is added to the cam profile to eliminate the difference in start position. The RampIn parametr is an average velocity of the ramp-in function. Ramp-in path is not generated for RampIn=0 and error -707 (position or velocity step) is invoked if some difference is detected. Recommended value for the RampIn parametr is 0.1 to 0.5 of maximal slave axis velocity. The parameter has to be lowered if maximal velocity or acceleration error is detected.

Inputs

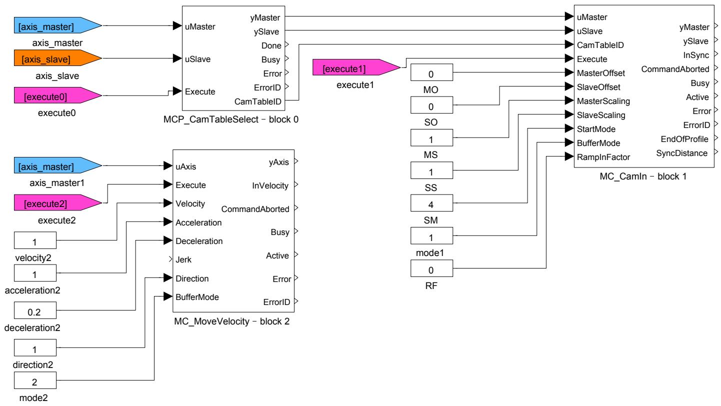
uMaster	Master axis reference	Reference
uSlave	Slave axis reference	Reference
CamTableID	Cam table reference (connect to MCP_CamTableSelect.CamTableID)	Reference
Execute	The block is activated on rising edge	Bool
MasterOffset	Offset in cam table on master side [unit]	Double (F64)
SlaveOffset	Offset in cam table on slave side [unit]	Double (F64)
MasterScaling	Overall scaling factor in cam table on master side	Double (F64)
SlaveScaling	Overall scaling factor in cam table on slave side	Double (F64)
StartMode	Select relative or absolute cam table 1 ..... Master relative 2 ..... Slave relative 3 ..... Both relative 4 ..... Both absolute	Long (I32)
BufferMode	Buffering mode 1 ..... Aborting (start immediately) 2 ..... Buffered (start after finish of previous motion) 3 ..... Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands) 4 ..... Blending high (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands) 5 ..... Blending previous (start after finishing the previous motion, previous motion finishes with its final velocity) 6 ..... Blending next (start after finishing the previous motion, previous motion finishes with the starting velocity of the next block)	Long (I32)
RampIn	RampIn factor (0 = RampIn mode not used); average additive velocity (absolute value) during ramp-in process	Double (F64)

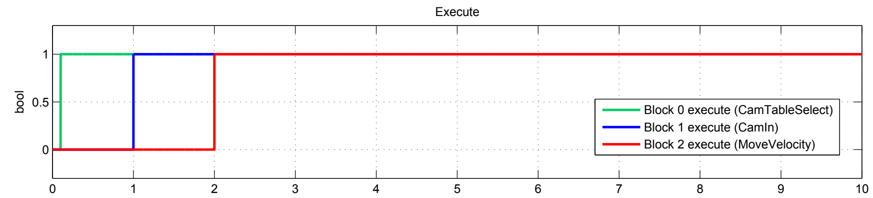
Outputs

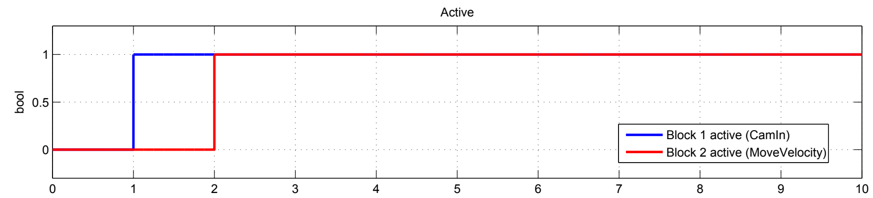
yMaster	Master axis reference	Reference
ySlave	Slave axis reference	Reference
InSync	Slave axis reached the cam profile	Bool
CommandAborted	Algorithm was aborted	Bool
Busy	Algorithm not finished yet	Bool
Active	The block is controlling the axis	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation i ..... REXYGEN general error	Error
EndOfProfile	Indicate end of cam profile (not periodic cam only)	Bool

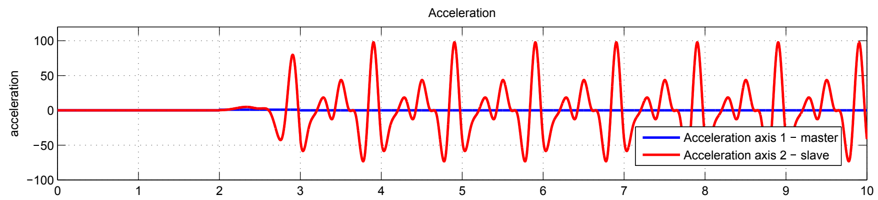
`SyncDistance`	Position deviation of the slave axis from synchronized position	`Double (F64)`

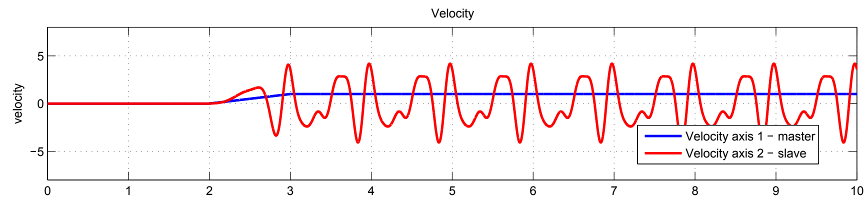
Example

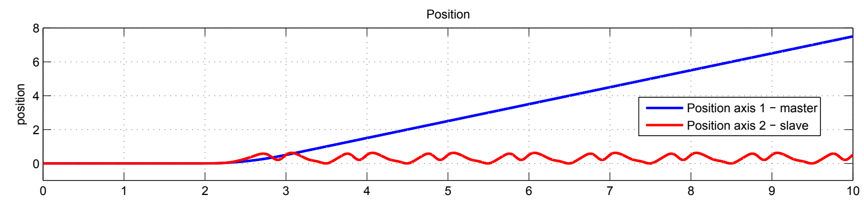

[axis_master]
axis_master
[axis_slave]
axis_slave
[execute0]
execute0
uMaster
uSlave
Execute
yMaster
ySlave
Done
Busy
Error
ErrorID
CamTableID
MCP_CamTableSelect - block 0
[execute1]
execute1
0
MO
0
SO
1
MS
1
SS
4
SM
1
mode1
0
RF
uMaster
uSlave
CamTableID
Execute
MasterOffset
SlaveOffset
MasterScaling
SlaveScaling
StartMode
BufferMode
RampInFactor
yMaster
ySlave
InSync
CommandAborted
Busy
Active
Error
ErrorID
EndOfProfile
SyncDistance
MC_CamIn - block 1
[axis_master]
axis_master1
[execute2]
execute2
1
velocity2
1
acceleration2
0.2
deceleration2
1
direction2
2
mode2
uAxis
Execute
Velocity
Acceleration
Deceleration
Jerk
Direction
BufferMode
yAxis
InVelocity
CommandAborted
Busy
Active
Error
ErrorID
MC_MoveVelocity - block 2


Execute
bool
Block 0 execute (CamTableSelect)
Block 1 execute (CamIn)
Block 2 execute (MoveVelocity)


Active
bool
Block 1 active (CamIn)
Block 2 active (MoveVelocity)


Acceleration
acceleration
Acceleration axis 1 - master
Acceleration axis 2 - slave

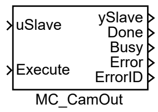

Velocity
velocity
Velocity axis 1 - master
Velocity axis 2 - slave


Position
position
Position axis 1 - master
Position axis 2 - slave

MC_CamOut – Disengage the cam

Block Symbol

Licence: MOTION CONTROL



Function Description

The **MC_CamOut** block switches off the cam mode on slave axis. If cam mode is not active, the block does nothing (no error is activated).

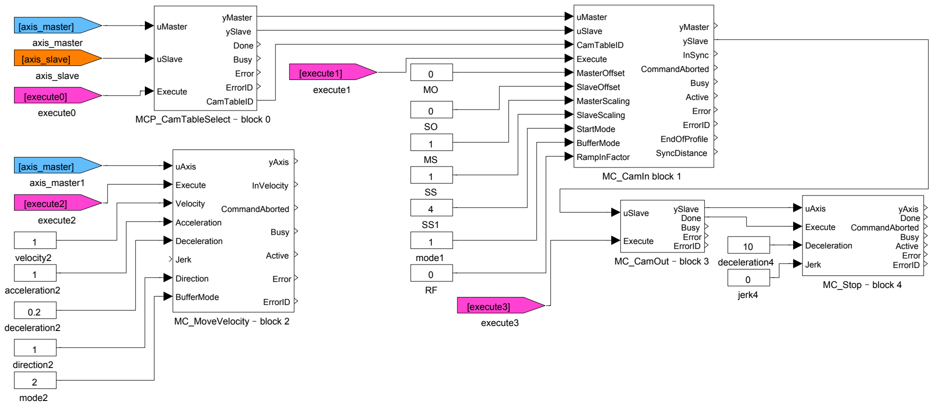
Inputs

uSlave	Slave axis reference	Reference
Execute	The block is activated on rising edge	Bool

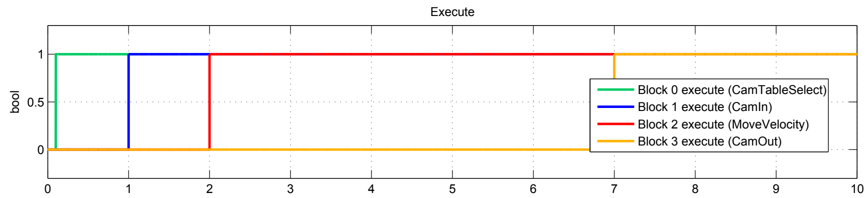
Outputs

ySlave	Slave axis reference	Reference
Done	Algorithm finished	Bool
Busy	Algorithm not finished yet	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation i ..... REXYGEN general error	Error

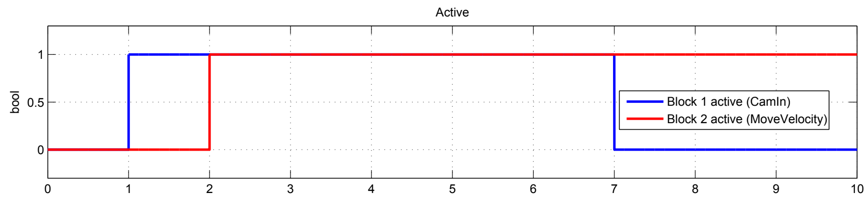
Example



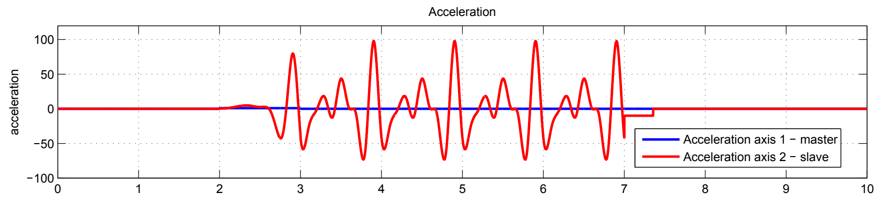
[axis_master]
axis_master
[axis_slave]
axis_slave
[execute0]
execute0
MCP_CamTableSelect - block 0
[execute1]
execute1
MC_CamIn block 1
[axis_master]
axis_master1
[execute2]
execute2
velocity2
acceleration2
deceleration2
direction2
mode2
MC_MoveVelocity - block 2
MO
SO
MS
SS
SS1
mode1
RF
[execute3]
execute3
MC_CamOut - block 3
deceleration4
jerk4
MC_Stop - block 4



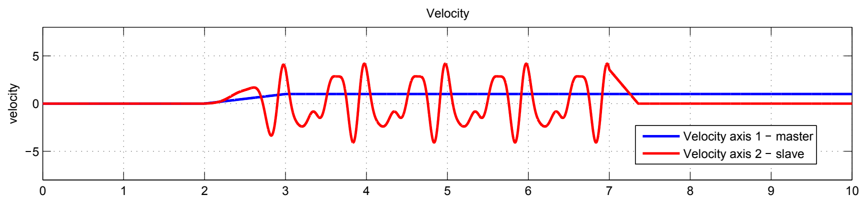
Execute
bool
Block 0 execute (CamTableSelect)
Block 1 execute (CamIn)
Block 2 execute (MoveVelocity)
Block 3 execute (CamOut)



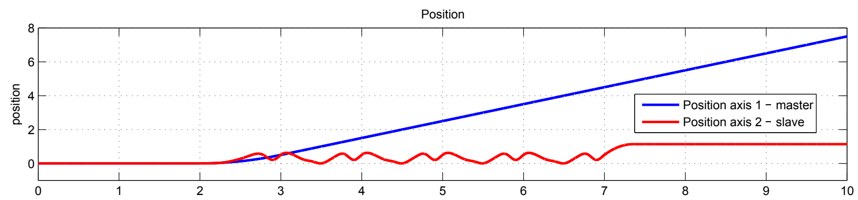
Active
bool
Block 1 active (CamIn)
Block 2 active (MoveVelocity)



Acceleration
acceleration
Acceleration axis 1 - master
Acceleration axis 2 - slave



Velocity
velocity
Velocity axis 1 - master
Velocity axis 2 - slave

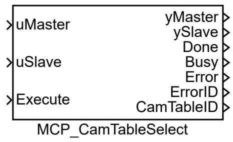


Position
position
Position axis 1 - master
Position axis 2 - slave

MCP_CamTableSelect – Cam definition

Block Symbol

Licence: MOTION CONTROL



Function Description

The MCP_CamTableSelect block defines a cam profile. The definition is similar to MC_PositionProfile block, but the time axis is replaced by master position axis. There are also two possible ways for cam profile definition:

1. sequence of values: given sequence of master-slave position pairs. In each master position interval, value of slave position is interpolated by 3rd-order polynomial (simple linear interpolation would lead to steps in velocity at interval border). Master position sequence is in array/parameter **mvalues**, slave position sequence is in array/parameter **svalues**. Master position sequence must be increasing.

2. spline: master position sequence is the same as in previous case. Each interval is interpolated by 5th-order polynomial $p(x) = a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$ where beginning of time-interval is defined for $x = 0$, end of time-interval holds for $x = 1$ and factors a_i are put in array/parameter **svalues** in ascending order (e.g. array/parameter **svalues** contain 6 values for each interval). This method allows to reduce the number of intervals and there is special graphical editor available for interpolating curve synthesis.

For both cases the master position sequence can be equidistantly spaced in time and then the time array includes only first and last point.

Note 1: input CamTable which is defined in PLCOpen specification is missing, because all path data are set in the parameters of the block.

Note 2: parameter **svalues** must be set as a vector in all cases, e.g. text string must not include a semicolon.

Note 3: incorrect parameter value cSeg (higher then real size of arrays times and/or values) can lead to unpredictable results and in some cases to crash of the whole runtime execution (The problem is platform dependent and currently it is observed only for SIMULINK version).

Inputs

uMaster	Master axis reference	Reference
uSlave	Slave axis reference	Reference
Execute	The block is activated on rising edge	Bool

Outputs

yMaster	Master axis reference	Reference
ySlave	Slave axis reference	Reference
Done	Algorithm finished	Bool
Busy	Algorithm not finished yet	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation i ..... REXYGEN general error	Error
CamTableID	Cam table reference (connect to MC_CamIn.CamTableID)	Reference

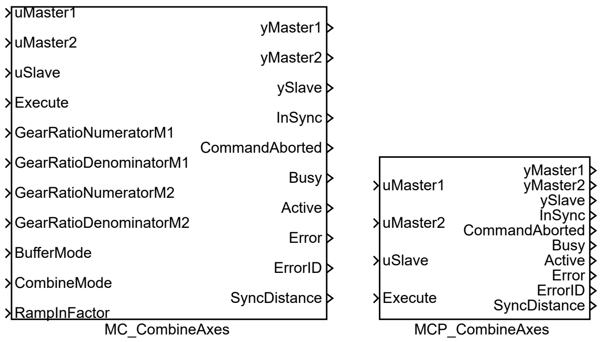
Parameters

alg	Algorithm for interpolation 1 ..... Sequence of time/value pairs 2 ..... Sequence of equidistant values 3 ..... Spline 4 ..... Equidistant spline	⊙2	Long (I32)
nmax	Number of profile segments	⊙3	Long (I32)
Periodic	Indicate periodic cam profile	⊙on	Bool
camname	Filename of special editor data file (filename is generated by system if parameter is empty)		String
mvalues	Master positions where segments are switched	⊙[0 30]	Double (F64)
sValues	Slave positions or interpolating polynomial coefficients (a0, a1, a2, ...)	⊙[0 100 100 0]	Double (F64)

MC_CombineAxes, MCP_CombineAxes – Combine the motion of 2 axes into a third axis

Block Symbols

Licence: MOTION CONTROL



Function Description

The `MC_CombineAxes` block combines a motion of two master axes into a slave axis command. The slave axis indicates synchronized motion state. Following relationship holds:

$$\begin{aligned}\texttt{SlavePosition} &= \texttt{Master1Position} \cdot \frac{\texttt{GearRatioNumeratorM1}}{\texttt{GearRatioDenominatorM1}} + \\ &+ \texttt{Master2Position} \cdot \frac{\texttt{GearRatioNumeratorM2}}{\texttt{GearRatioDenominatorM2}}\end{aligned}$$

Negative number can be set in `GearRatio...` parameter to obtain the resulting slave movement in form of difference of master axes positions.

Inputs

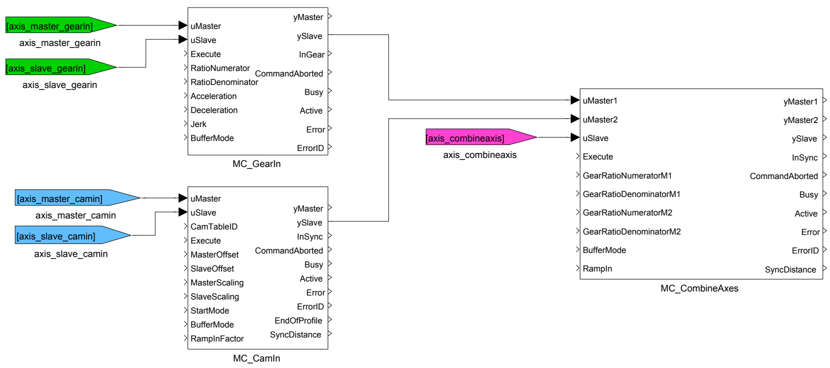
`uMaster1`	First master axis reference	`Reference`
`uMaster2`	Second master axis reference	`Reference`
`uSlave`	Slave axis reference	`Reference`
`Execute`	The block is activated on rising edge	`Bool`
`GearRatioNumeratorM1`	Numerator for the gear factor for master axis 1	`Long (I32)`
`GearRatioDenominatorM1`	Denominator for the gear factor for master axis 1	`Long (I32)`
`GearRatioNumeratorM2`	Numerator for the gear factor for master axis 2	`Long (I32)`
`GearRatioDenominatorM2`	Denominator for the gear factor for master axis 2	`Long (I32)`

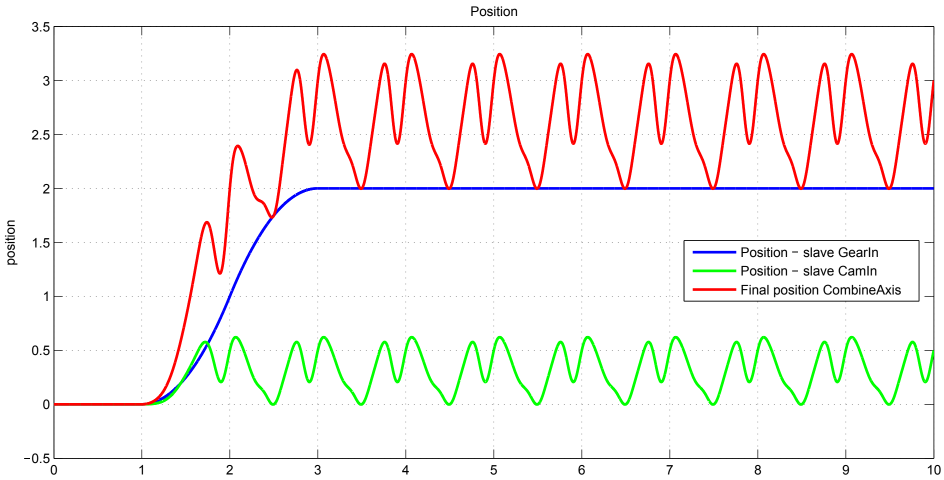
`BufferMode`	Buffering mode `1` ..... Aborting (start immediately) `2` ..... Buffered (start after finish of previous motion) `3` ..... Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands) `4` ..... Blending high (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands) `5` ..... Blending previous (start after finishing the previous motion, previous motion finishes with its final velocity) `6` ..... Blending next (start after finishing the previous motion, previous motion finishes with the starting velocity of the next block)	`Long (I32)`
`RampIn`	RampIn factor (0 = RampIn mode not used)	`Double (F64)`

Outputs

`yMaster1`	First master axis reference	`Reference`
`yMaster2`	Second master axis reference	`Reference`
`ySlave`	Slave axis reference	`Reference`
`InSync`	Slave axis reached the cam profile	`Bool`
`CommandAborted`	Algorithm was aborted	`Bool`
`Busy`	Algorithm not finished yet	`Bool`
`Active`	The block is controlling the axis	`Bool`
`Error`	Error occurred	`Bool`
`ErrorID`	Result of the last operation `i` ..... REXYGEN general error	`Error`
`SyncDistance`	Position deviation of the slave axis from synchronized position	`Double (F64)`

Example

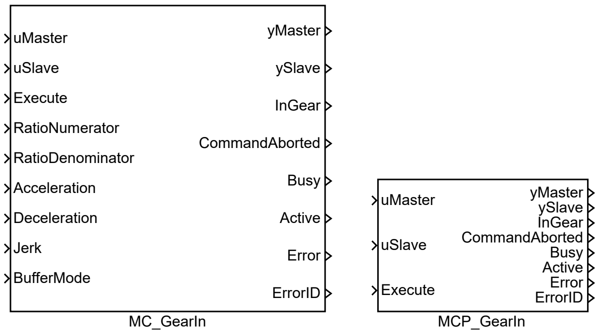

[axis_master_gearin]
axis_master_gearin
[axis_slave_gearin]
axis_slave_gearin
uMaster
uSlave
Execute
RatioNumerator
RatioDenominator
Acceleration
Deceleration
Jerk
BufferMode
yMaster
ySlave
InGear
CommandAborted
Busy
Active
Error
ErrorID
MC_GearIn
[axis_master_camin]
axis_master_camin
[axis_slave_camin]
axis_slave_camin
uMaster
uSlave
CamTableID
Execute
MasterOffset
SlaveOffset
MasterScaling
SlaveScaling
StartMode
BufferMode
RampInFactor
yMaster
ySlave
InSync
CommandAborted
Busy
Active
Error
ErrorID
EndOfProfile
SyncDistance
MC_CamIn
[axis_combineaxis]
axis_combineaxis
uMaster1
uMaster2
uSlave
Execute
GearRatioNumeratorM1
GearRatioDenominatorM1
GearRatioNumeratorM2
GearRatioDenominatorM2
BufferMode
RampIn
yMaster1
yMaster2
ySlave
InSync
CommandAborted
Busy
Active
Error
ErrorID
SyncDistance
MC_CombineAxes


Position
position
Position − slave GearIn
Position − slave CamIn
Final position CombineAxis

MC_GearIn, MCP_GearIn – Engange the master/slave velocity ratio

Block Symbols

Licence: MOTION CONTROL



Function Description

The `MC_GearIn` *and* `MCP_GearIn` *blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the* `MCP_` *version of the block.*

The `MC_GearIn` block commands the slave axis motion in such a way that a preset ratio between master and slave velocities is maintained. Considering the velocity of master axis $VelM$ and velocity of slave axis $VelS$, following relation holds (without phasing): $VelS = VelM * RatioNumerator/RatioDenominator$. Position and acceleration is commanded to be consistent with velocity; position/distance ratio is also locked. This mode of synchronized motion is called electronic gear.

The gear mode is switched off by executing other motion block on slave axis with mode `aborting` or by executing a `MC_GearIn` block.

Similarly to the `MC_CamIn` block, ramp-in mode is activated if initial velocity of slave axis is different from master axis and gearing ratio. Parameters `Acceleration`, `Deceleration`, `Jerk` are used during ramp-in mode.

Inputs

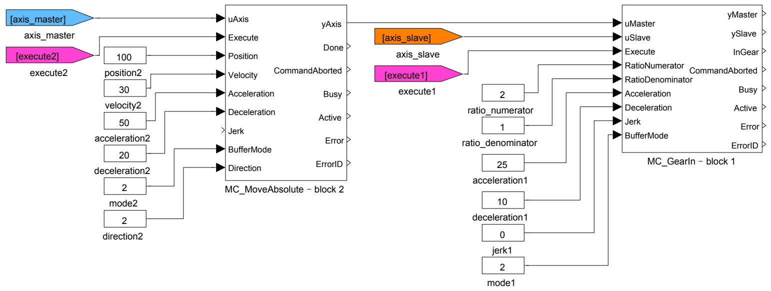
`uMaster`	Master axis reference	`Reference`
`uSlave`	Slave axis reference	`Reference`
`Execute`	The block is activated on rising edge	`Bool`
`RatioNumerator`	Gear ratio Numerator	`Long (I32)`
`RatioDenominator`	Gear ratio Denominator	`Long (I32)`
`Acceleration`	Maximal allowed acceleration [unit/s^2]	`Double (F64)`
`Deceleration`	Maximal allowed deceleration [unit/s^2]	`Double (F64)`
`Jerk`	Maximal allowed jerk [unit/s^3]	`Double (F64)`

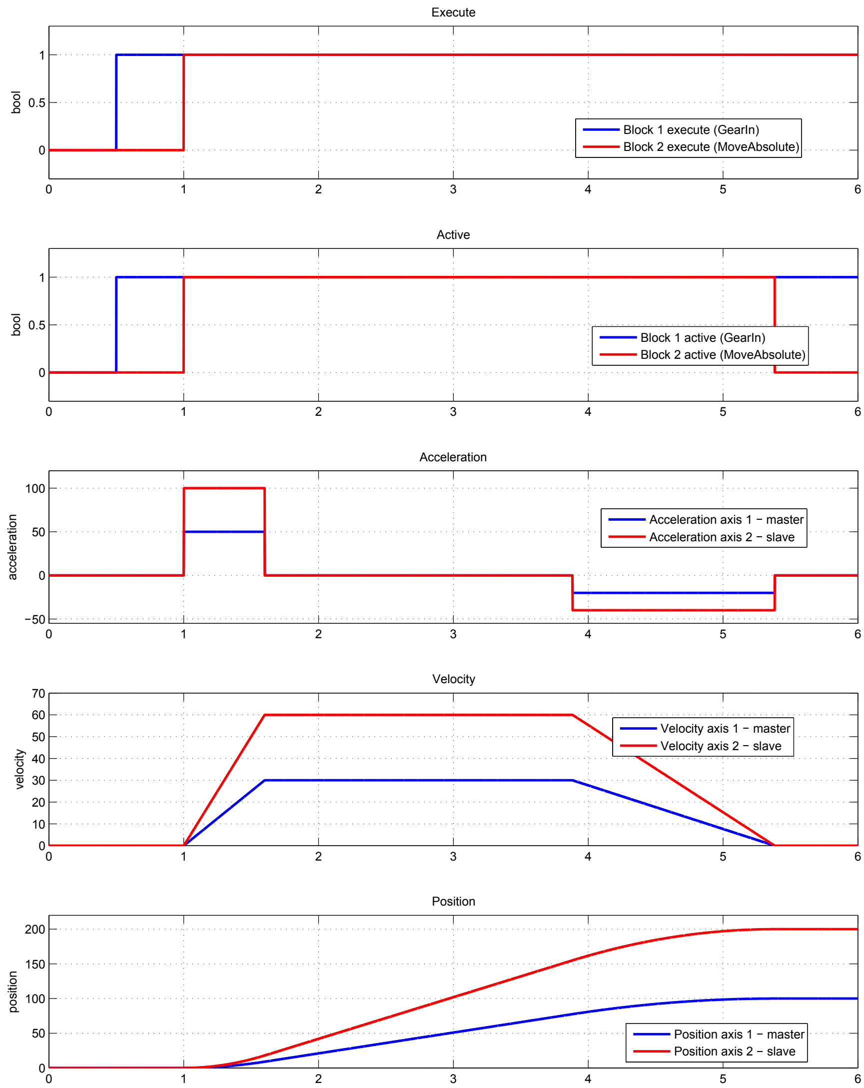
`BufferMode`	Buffering mode `1` ..... Aborting (start immediately) `2` ..... Buffered (start after finish of previous motion) `3` ..... Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands) `4` ..... Blending high (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands) `5` ..... Blending previous (start after finishing the previous motion, previous motion finishes with its final velocity) `6` ..... Blending next (start after finishing the previous motion, previous motion finishes with the starting velocity of the next block)	`Long (I32)`

Outputs

`yMaster`	Master axis reference	`Reference`
`ySlave`	Slave axis reference	`Reference`
`InGear`	Slave axis reached gearing ratio	`Bool`
`CommandAborted`	Algorithm was aborted	`Bool`
`Busy`	Algorithm not finished yet	`Bool`
`Active`	The block is controlling the axis	`Bool`
`Error`	Error occurred	`Bool`
`ErrorID`	Result of the last operation `i` ..... REXYGEN general error	`Error`

Example

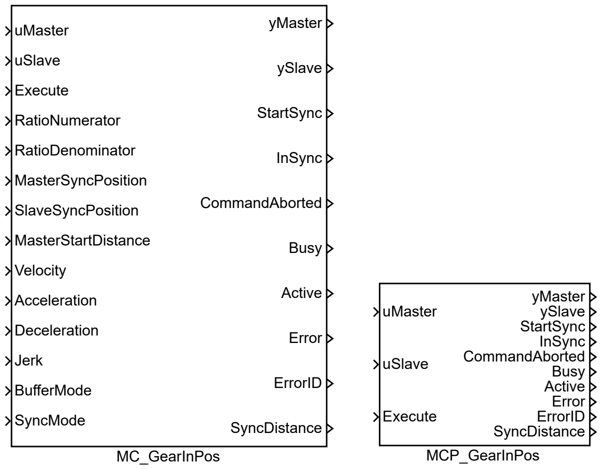

[axis_master]
axis_master
[execute2]
execute2
100
position2
30
velocity2
50
acceleration2
20
deceleration2
2
mode2
2
direction2
uAxis
Execute
Position
Velocity
Acceleration
Deceleration
Jerk
BufferMode
Direction
yAxis
Done
CommandAborted
Busy
Active
Error
ErrorID
MC_MoveAbsolute - block 2
[axis_slave]
axis_slave
[execute1]
execute1
2
ratio_numerator
1
ratio_denominator
25
acceleration1
10
deceleration1
0
jerk1
2
mode1
uMaster
uSlave
Execute
RatioNumerator
RatioDenominator
Acceleration
Deceleration
Jerk
BufferMode
yMaster
ySlave
InGear
CommandAborted
Busy
Active
Error
ErrorID
MC_GearIn - block 1


Execute
bool
Block 1 execute (GearIn)
Block 2 execute (MoveAbsolute)
Active
bool
Block 1 active (GearIn)
Block 2 active (MoveAbsolute)
Acceleration
acceleration
Acceleration axis 1 - master
Acceleration axis 2 - slave
Velocity
velocity
Velocity axis 1 - master
Velocity axis 2 - slave
Position
position
Position axis 1 - master
Position axis 2 - slave

MC_GearInPos, MCP_GearInPos – Engage the master/slave velocity ratio in defined position

Block Symbols

Licence: MOTION CONTROL



Function Description

The MC_GearInPos *and* MCP_GearInPos *blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the* MCP_ *version of the block.*

The functional block MC_GearInPos engages a synchronized motion of master and slave axes in such a way that the ratio of velocities of both axes is maintained at a constant value. Compared to MC_GearIn, also the master to slave *position ratio* is determined in a given reference point, i.e. following relation holds:

$$\frac{SlavePosition - SlaveSyncPosition}{MasterPosition - MasterSyncPosition} = \frac{\texttt{RatioNumerator}}{\texttt{RatioDenominator}}.$$

In case that the slave position does not fulfill this condition of synchronicity at the moment of block activation (i.e. in an instant of positive edge of Execute input and after execution of previous commands in buffered mode), synchronization procedure is started and indicated by output StartSync. During this procedure, proper slave trajectory which results in smooth synchronization of both axes is generated with respect to actual master motion and slave limits for Velocity, Acceleration, Deceleration and Jerk (these limits are not applied from the moment of successful synchronization). Parameter setting MasterStartDistance=0 leads to immediate start of synchronization procedure at the moment of block activation (by the Execute input). Otherwise, the synchronization starts as soon as the master position enters the interval MasterSyncPosition ± MasterStartDistance.

Notes:
1. The synchronization procedure uses two algorithms: I. The algorithm implemented in MC_MoveAbsolute is recomputed in every time instant in such a way, that the end velocity is set to actual velocity of master axis. II. The position, velocity and acceleration is generated in the same manner as in the synchronized motion and a proper 5th order interpolation polynomial is added to achieve smooth transition to the synchronized state. The length of interpolation trajectory is computed in such a way that maximum velocity, acceleration and jerk do not violate the specified limits (for the interpolation polynomial). The first algorithm cannot be used for nonzero acceleration of the master axis whereas the second does not guarantee the compliance of maximum limits for the overall slave trajectory. Both algorithms are combined in a proper way to achieve the synchronized motion of both axes.

2. The block parameters (execution of synchronization and velocity/acceleration limits) have to be chosen so that the slave position is close to SlaveSyncPosition approximately at the moment when the master position enters the range for synchronization given by MasterSyncPosition and MasterStartDistance. Violation of this rule can lead to unpredictable behaviour of the slave axis during the synchronization or to an overrun of the specified limits for slave axis. However, the motion of both axes is usually well defined and predictable in standard applications and correct synchronization can be performed easily by proper configuration of motion commands and functional block parameters.

Inputs

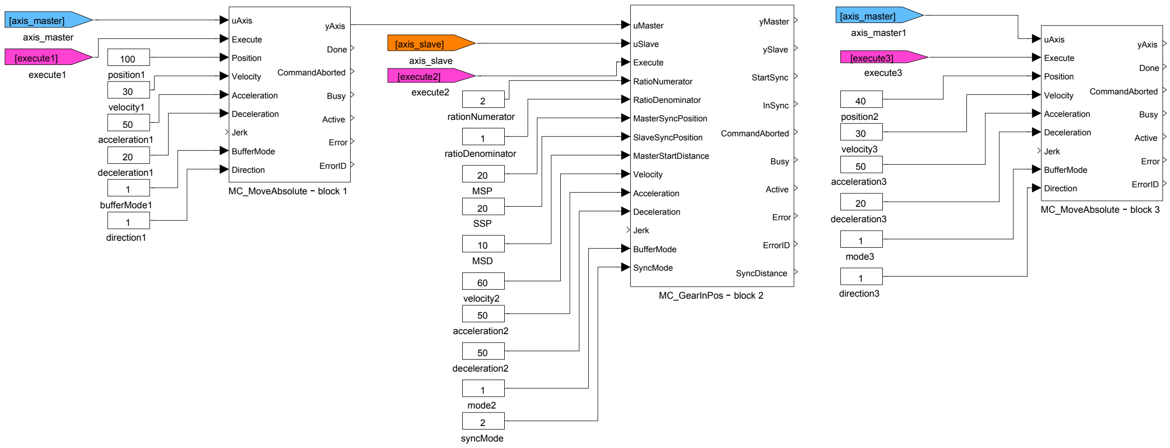
uMaster	Master axis reference	Reference
uSlave	Slave axis reference	Reference
Execute	The block is activated on rising edge	Bool
RatioNumerator	Gear ratio Numerator	Long (I32)
RatioDenominator	Gear ratio Denominator	Long (I32)
MasterSyncPosition	Master position for synchronization	Double (F64)
SlaveSyncPosition	Slave position for synchronization	Double (F64)
MasterStartDistance	Master distance for starting gear in procedure	Double (F64)
Velocity	Maximal allowed velocity [unit/s]	Double (F64)
Acceleration	Maximal allowed acceleration [unit/s^2]	Double (F64)
Deceleration	Maximal allowed deceleration [unit/s^2]	Double (F64)
Jerk	Maximal allowed jerk [unit/s^3]	Double (F64)

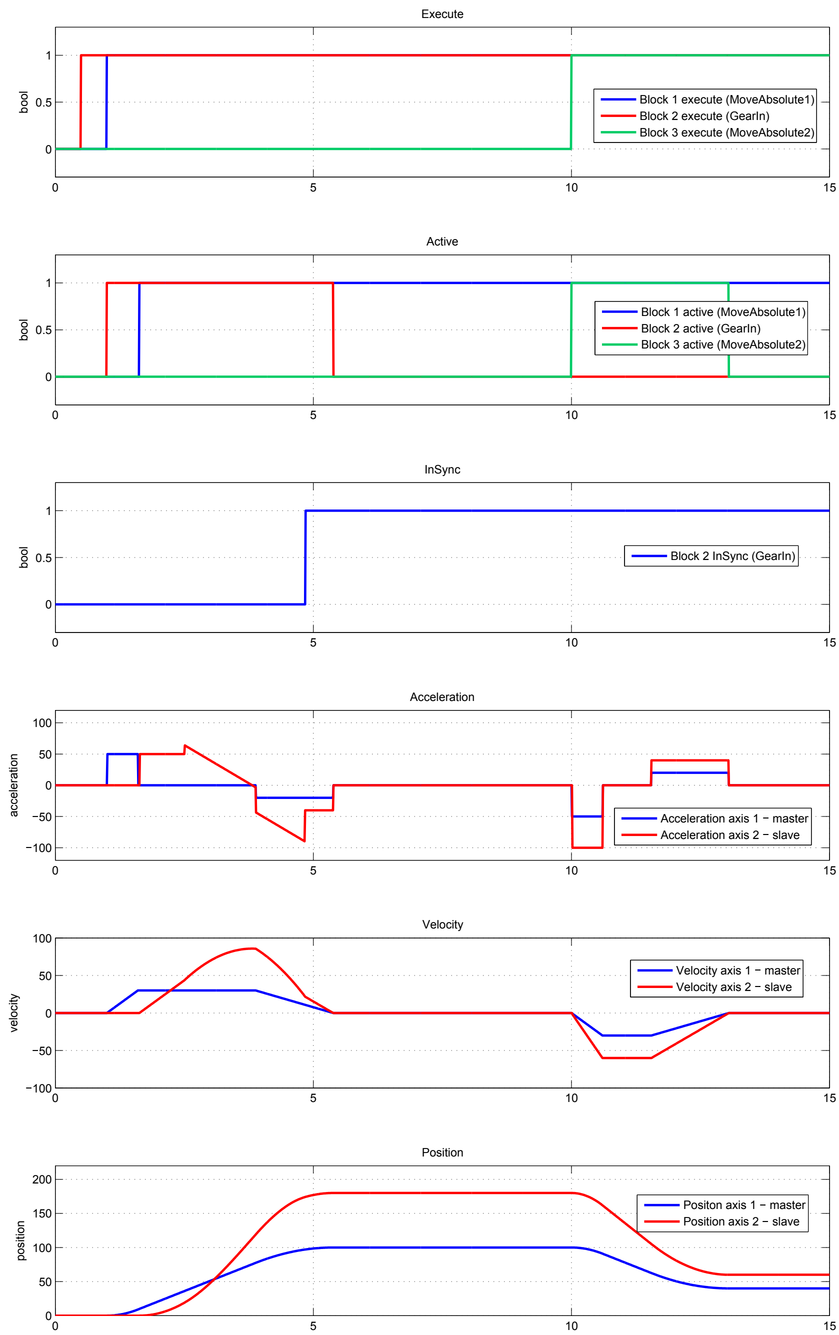
`BufferMode`	Buffering mode 1 ..... Aborting (start immediately) 2 ..... Buffered (start after finish of previous motion) 3 ..... Blending low (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands) 4 ..... Blending high (start after finishing the previous motion, previous motion finishes with the lowest velocity of both commands) 5 ..... Blending previous (start after finishing the previous motion, previous motion finishes with its final velocity) 6 ..... Blending next (start after finishing the previous motion, previous motion finishes with the starting velocity of the next block)	`Long (I32)`
`SyncMode`	Synchronization mode (cyclic axes only) 1 ..... CatchUp 2 ..... Shortest 3 ..... SlowDown	`Long (I32)`

Outputs

`yMaster`	Master axis reference	`Reference`
`ySlave`	Slave axis reference	`Reference`
`StartSync`	Commanded gearing starts	`Bool`
`InSync`	Slave axis reached the cam profile	`Bool`
`CommandAborted`	Algorithm was aborted	`Bool`
`Busy`	Algorithm not finished yet	`Bool`
`Active`	The block is controlling the axis	`Bool`
`Error`	Error occurred	`Bool`
`ErrorID`	Result of the last operation i ..... REXYGEN general error	`Error`
`SyncDistance`	Position deviation of the slave axis from synchronized position	`Double (F64)`

Example

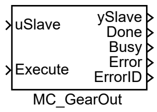

[axis_master]
axis_master
[execute1]
execute1
100
position1
30
velocity1
50
acceleration1
20
deceleration1
1
bufferMode1
1
direction1
uAxis
Execute
Position
Velocity
Acceleration
Deceleration
Jerk
BufferMode
Direction
yAxis
Done
CommandAborted
Busy
Active
Error
ErrorID
MC_MoveAbsolute - block 1
[axis_slave]
axis_slave
[execute2]
execute2
2
rationNumerator
1
ratioDenominator
20
MSP
20
SSP
10
MSD
60
velocity2
50
acceleration2
50
deceleration2
1
mode2
2
syncMode
uMaster
uSlave
Execute
RatioNumerator
RatioDenominator
MasterSyncPosition
SlaveSyncPosition
MasterStartDistance
Velocity
Acceleration
Deceleration
Jerk
BufferMode
SyncMode
yMaster
ySlave
StartSync
InSync
CommandAborted
Busy
Active
Error
ErrorID
SyncDistance
MC_GearInPos - block 2
[axis_master]
axis_master1
[execute3]
execute3
40
position2
30
velocity3
50
acceleration3
20
deceleration3
1
mode3
1
direction3
uAxis
Execute
Position
Velocity
Acceleration
Deceleration
Jerk
BufferMode
Direction
yAxis
Done
CommandAborted
Busy
Active
Error
ErrorID
MC_MoveAbsolute - block 3


Execute
bool
Block 1 execute (MoveAbsolute1)
Block 2 execute (GearIn)
Block 3 execute (MoveAbsolute2)
Active
bool
Block 1 active (MoveAbsolute1)
Block 2 active (GearIn)
Block 3 active (MoveAbsolute2)
InSync
bool
Block 2 InSync (GearIn)
Acceleration
acceleration
Acceleration axis 1 − master
Acceleration axis 2 − slave
Velocity
velocity
Velocity axis 1 − master
Velocity axis 2 − slave
Position
position
Positon axis 1 − master
Position axis 2 − slave

MC_GearOut – Disengange the master/slave velocity ratio

Block Symbol

Licence: MOTION CONTROL



Function Description

The MC_GearOut block switches off the gearing mode on the slave axis. If gearing mode is not active (no MC_GearIn block commands slave axis at this moment), block does nothing (no error is activated).

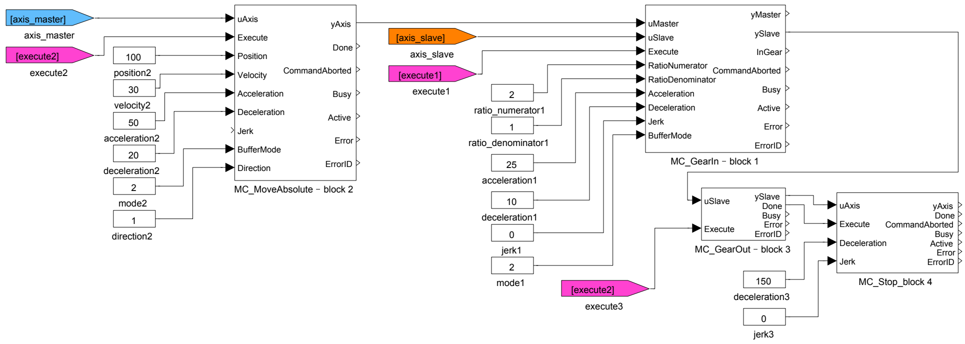
Inputs

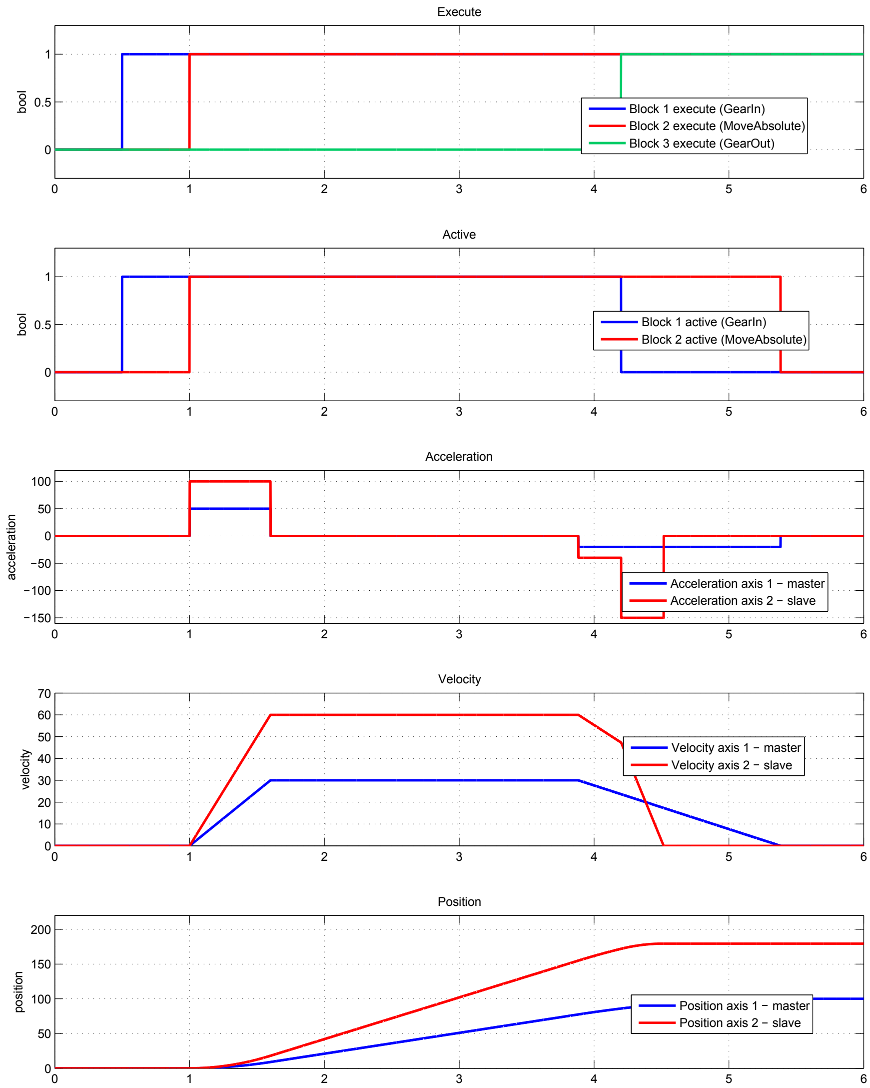
uSlave	Slave axis reference	Reference
Execute	The block is activated on rising edge	Bool

Outputs

ySlave	Slave axis reference	Reference
Done	Algorithm finished	Bool
Busy	Algorithm not finished yet	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation i ..... REXYGEN general error	Error

Example

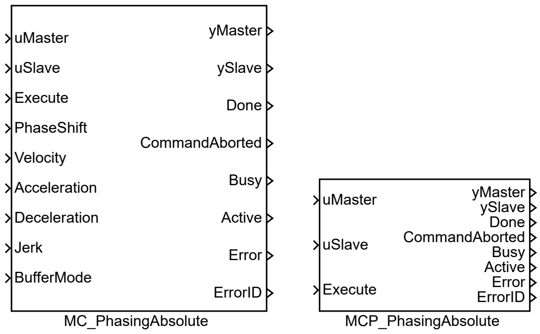

[axis_master]
axis_master
[execute2]
execute2
100
position2
30
velocity2
50
acceleration2
20
deceleration2
2
mode2
1
direction2
uAxis
Execute
Position
Velocity
Acceleration
Deceleration
Jerk
BufferMode
Direction
yAxis
Done
CommandAborted
Busy
Active
Error
ErrorID
MC_MoveAbsolute - block 2
[axis_slave]
axis_slave
[execute1]
execute1
2
ratio_numerator1
1
ratio_denominator1
25
acceleration1
10
deceleration1
0
jerk1
2
mode1
uMaster
uSlave
Execute
RatioNumerator
RatioDenominator
Acceleration
Deceleration
Jerk
BufferMode
yMaster
ySlave
InGear
CommandAborted
Busy
Active
Error
ErrorID
MC_GearIn - block 1
uSlave
Execute
ySlave
Done
Busy
Error
ErrorID
MC_GearOut - block 3
[execute2]
execute3
uAxis
Execute
Deceleration
Jerk
yAxis
Done
CommandAborted
Busy
Active
Error
ErrorID
MC_Stop_block 4
150
deceleration3
0
jerk3


Execute
bool
Block 1 execute (GearIn)
Block 2 execute (MoveAbsolute)
Block 3 execute (GearOut)
Active
bool
Block 1 active (GearIn)
Block 2 active (MoveAbsolute)
Acceleration
acceleration
Acceleration axis 1 − master
Acceleration axis 2 − slave
Velocity
velocity
Velocity axis 1 − master
Velocity axis 2 − slave
Position
position
Position axis 1 − master
Position axis 2 − slave

MC_PhasingAbsolute, MCP_PhasingAbsolute – Phase shift in synchronized motion (absolute coordinates)

Block Symbols

Licence: MOTION CONTROL



Function Description

The MC_PhasingAbsolute *and* MCP_PhasingAbsolute *blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the* MCP_ *version of the block.*

The MC_PhasingAbsolute block introduces an additional phase shift in master-slave relation defined by an electronic cam (MC_CamIn) or electronic gear (MC_GearIn). The functionality of this command is very similar to MC_MoveSuperimposed (additive motion from 0 to PhaseShift position with respect to maximum velocity acceleration and jerk). The only difference is that the additive position/velocity/acceleration is added to master axis reference position in the functional dependence defined by a cam or gear ratio for the computation of slave motion instead of its direct summation with master axis movement. The absolute value of final phase shift is specified by PhaseShift parameter.

Note: The motion command is analogous to rotation of a mechanical cam by angle PhaseShift

Inputs

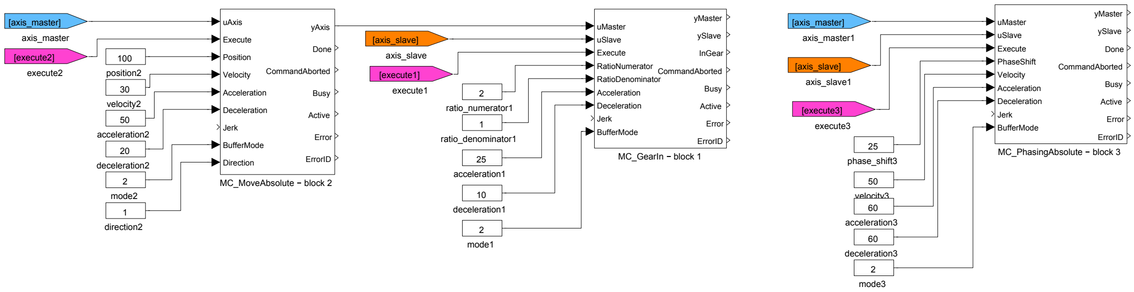
uMaster	Master axis reference	Reference
uSlave	Slave axis reference	Reference
Execute	The block is activated on rising edge	Bool
PhaseShift	Requested phase shift (distance on master axis) for cam	Double (F64)
Velocity	Maximal allowed velocity [unit/s]	Double (F64)
Acceleration	Maximal allowed acceleration [unit/s^2]	Double (F64)
Deceleration	Maximal allowed deceleration [unit/s^2]	Double (F64)

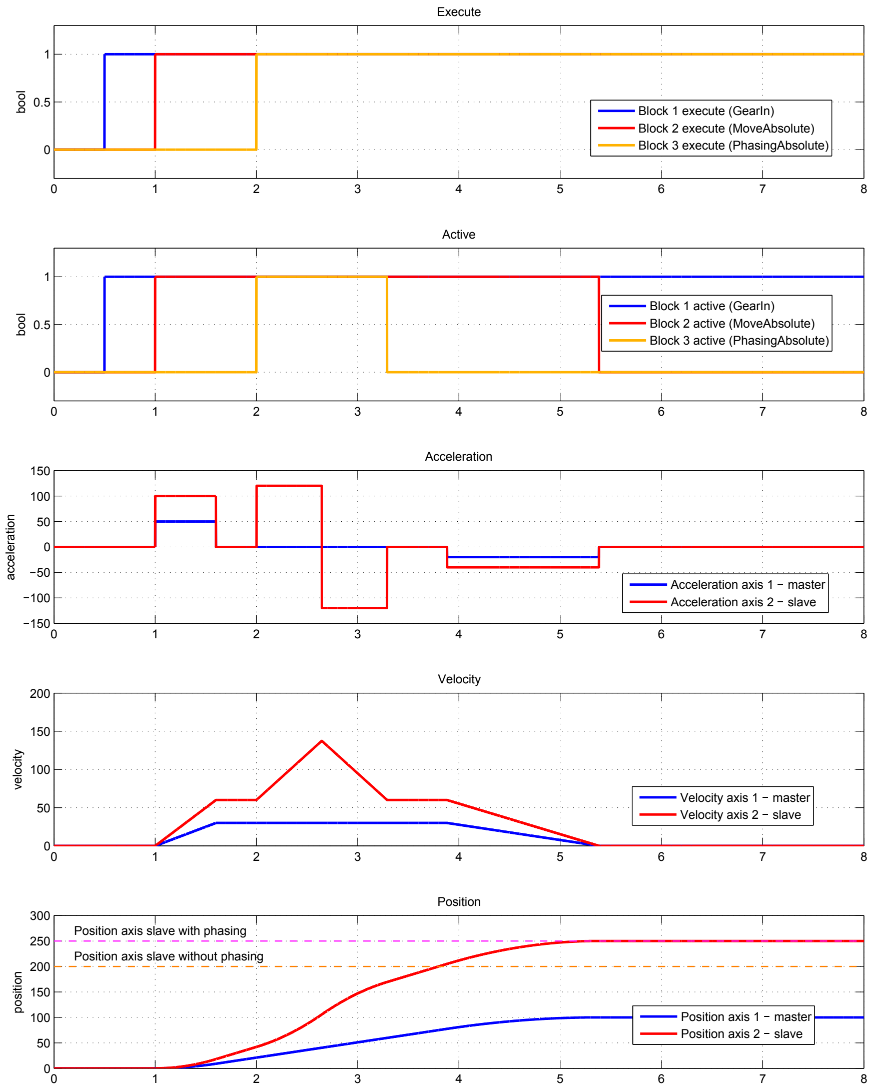
`Jerk`	Maximal allowed jerk [unit/s^3]	`Double (F64)`
`BufferMode`	Buffering mode `1` ..... Aborting `2` ..... Buffered	`Long (I32)`

Outputs

`yMaster`	Master axis reference	`Reference`
`ySlave`	Slave axis reference	`Reference`
`Done`	Algorithm finished	`Bool`
`CommandAborted`	Algorithm was aborted	`Bool`
`Busy`	Algorithm not finished yet	`Bool`
`Active`	The block is controlling the axis	`Bool`
`Error`	Error occurred	`Bool`
`ErrorID`	Result of the last operation `i` ..... REXYGEN general error	`Error`

Example

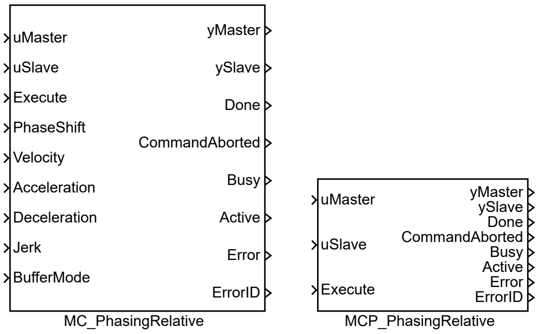

[axis_master]
axis_master
[execute2]
execute2
100
position2
30
velocity2
50
acceleration2
20
deceleration2
2
mode2
1
direction2
uAxis
Execute
Position
Velocity
Acceleration
Deceleration
Jerk
BufferMode
Direction
yAxis
Done
CommandAborted
Busy
Active
Error
ErrorID
MC_MoveAbsolute − block 2
[axis_slave]
axis_slave
[execute1]
execute1
2
ratio_numerator1
1
ratio_denominator1
25
acceleration1
10
deceleration1
2
mode1
uMaster
uSlave
Execute
RatioNumerator
RatioDenominator
Acceleration
Deceleration
Jerk
BufferMode
yMaster
ySlave
InGear
CommandAborted
Busy
Active
Error
ErrorID
MC_GearIn − block 1
[axis_master]
axis_master1
[axis_slave]
axis_slave1
[execute3]
execute3
25
phase_shift3
50
velocity3
60
acceleration3
60
deceleration3
2
mode3
uMaster
uSlave
Execute
PhaseShift
Velocity
Acceleration
Deceleration
Jerk
BufferMode
yMaster
ySlave
Done
CommandAborted
Busy
Active
Error
ErrorID
MC_PhasingAbsolute − block 3


Execute
bool
Block 1 execute (GearIn)
Block 2 execute (MoveAbsolute)
Block 3 execute (PhasingAbsolute)
Active
bool
Block 1 active (GearIn)
Block 2 active (MoveAbsolute)
Block 3 active (PhasingAbsolute)
Acceleration
acceleration
Acceleration axis 1 − master
Acceleration axis 2 − slave
Velocity
velocity
Velocity axis 1 − master
Velocity axis 2 − slave
Position
position
Position axis slave with phasing
Position axis slave without phasing
Position axis 1 − master
Position axis 2 − slave

MC_PhasingRelative, MCP_PhasingRelative – Phase shift in synchronized motion (relative coordinates)

Block Symbols

Licence: MOTION CONTROL



Function Description

The MC_PhasingRelative *and* MCP_PhasingRelative *blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the* MCP_ *version of the block.*

The MC_PhasingRelative introduces an additional phase shift in master-slave relation defined by an electronic cam (MC_CamIn) or electronic gear (MC_GearIn). The functionality of this command is very similar to MC_MoveSuperimposed (additive motion from 0 to PhaseShift position with respect to maximum velocity acceleration and jerk). The only difference is that the additive position/velocity/acceleration is added to master axis reference position in the functional dependence defined by a cam or gear ratio for the computation of slave motion instead of its direct summation with master axis movement. The relative value of final phase shift with respect to previous value is specified by PhaseShift parameter. Note: The motion command is analogous to rotation of a mechanical cam by angle PhaseShift

Inputs

uMaster	Master axis reference	Reference
uSlave	Slave axis reference	Reference
Execute	The block is activated on rising edge	Bool
PhaseShift	Requested phase shift (distance on master axis) for cam	Double (F64)
Velocity	Maximal allowed velocity [unit/s]	Double (F64)
Acceleration	Maximal allowed acceleration [unit/s^2]	Double (F64)
Deceleration	Maximal allowed deceleration [unit/s^2]	Double (F64)

Jerk	Maximal allowed jerk [unit/s^3]	Double (F64)
BufferMode	Buffering mode 1 ..... Aborting 2 ..... Buffered	Long (I32)

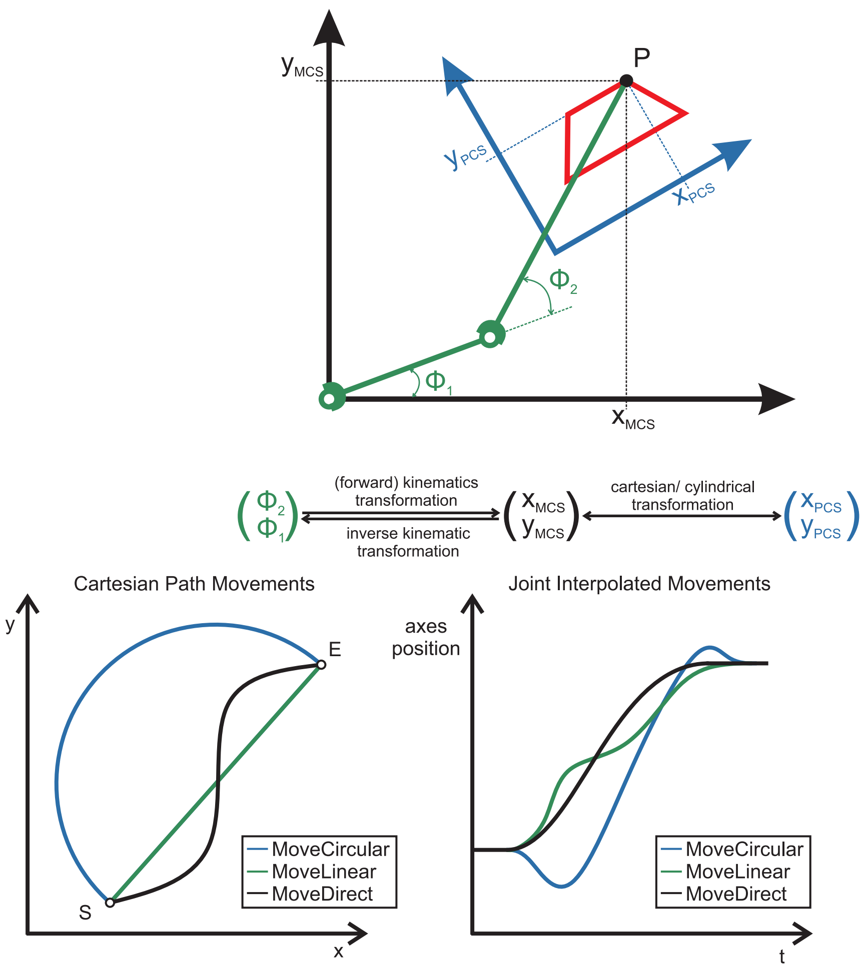
Outputs

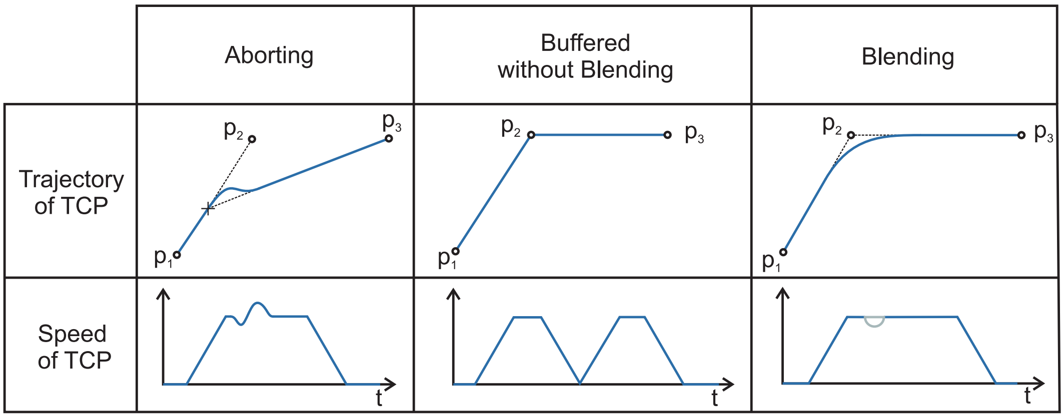
yMaster	Master axis reference	Reference
ySlave	Slave axis reference	Reference
Done	Algorithm finished	Bool
CommandAborted	Algorithm was aborted	Bool
Busy	Algorithm not finished yet	Bool
Active	The block is controlling the axis	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation i ..... REXYGEN general error	Error

Chapter 22

MC_COORD – Motion control - coordinated movement blocks

Contents

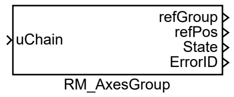

P
yMCS
yPCS
xPCS
Φ2
Φ1
xMCS
(forward) kinematics transformation
inverse kinematic transformation
cartesian/ cylindrical transformation
Cartesian Path Movements
Joint Interpolated Movements
y
E
S
x
axes position
t
MoveCircular
MoveLinear
MoveDirect


Aborting
Buffered
without Blending
Blending
Trajectory
of TCP
Speed
of TCP
p1
p2
p3
t

RM_AxesGroup – Axes group for coordinated motion control

Block Symbol

Licence: COORDINATED MOTION



Function Description

Note 1: Applicable for all non-administrative (moving) function blocks.
Note 2: In the states GroupErrorStop or GroupStopping, all Function Blocks canbe called, although they will not be executed, except MC_GroupReset for GroupErrorStop and any occurring Error– they will generate the transition to GroupStandby or GroupErrorStop respectively
Note 3: MC_GroupStop.DONE AND NOT MC_GroupStop.EXECUTE
Note 4: Transition is applicable if last axis is removed from the group
Note 5: Transition is applicable while group is not empty.
Note 6: MC_GroupDisable and MC_UngroupAllAxes can be issued in all states and will change the state to GroupDisabled.

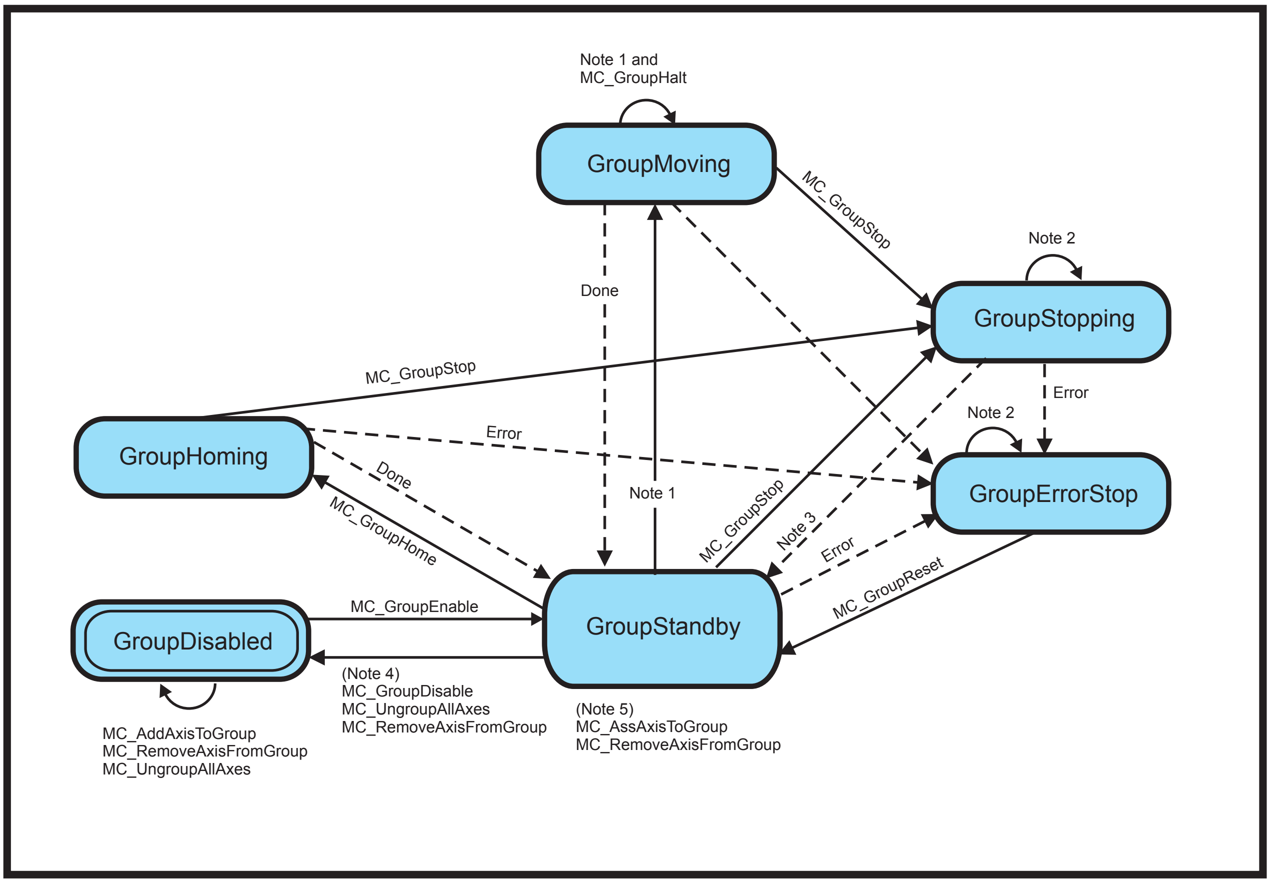
Parameters

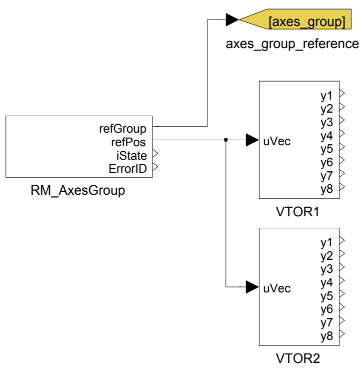
`McsCount`	Number of axis in MCS	↓1 ↑6 ⊙6	`Long (I32)`
`AcsCount`	Number of axis in ACS	↓1 ↑16 ⊙6	`Long (I32)`
`PosCount`	Number of position axis	↓1 ↑6 ⊙3	`Long (I32)`
`Velocity`	Maximal allowed velocity [unit/s]		`Double (F64)`
`Acceleration`	Maximal allowed acceleration [unit/s^2]		`Double (F64)`
`Jerk`	Maximal allowed jerk [unit/s^3]		`Double (F64)`

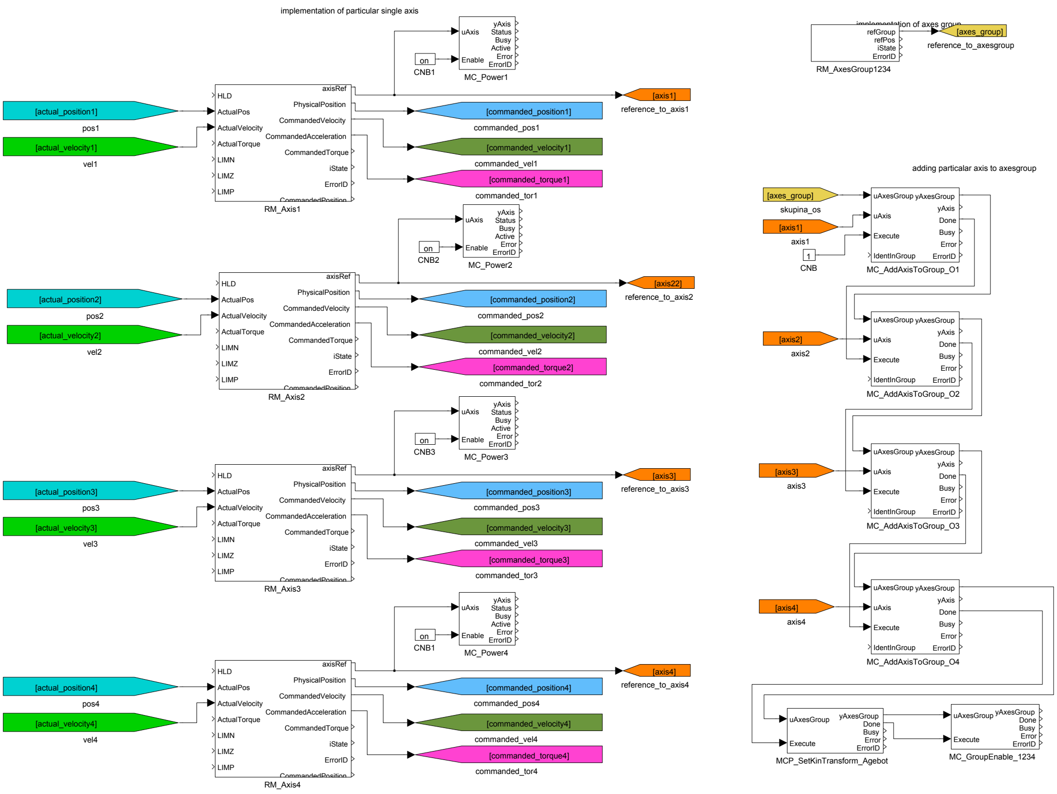
Outputs

`refGroup`	Axes group reference	`Reference`
`refPos`	Position, velocity and acceleration vector	`Reference`
`iState`	Group status 0 ..... Disabled 1 ..... Standby 2 ..... Homing 6 ..... Moving 7 ..... Stopping 8 ..... Error stop	`Long (I32)`
`ErrorID`	Result of the last operation i ..... REXYGEN general error	`Error`

The State Diagram of AxesGroup



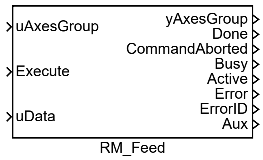



implementation of particular single axis
implementation of axes group
adding particalar axis to axesgroup
[actual_position1]
pos1
[actual_velocity1]
vel1
RM_Axis1
MC_Power1
CNB1
[axis1]
reference_to_axis1
[commanded_position1]
commanded_pos1
[commanded_velocity1]
commanded_vel1
[commanded_torque1]
commanded_tor1
[actual_position2]
pos2
[actual_velocity2]
vel2
RM_Axis2
MC_Power2
CNB2
[axis22]
reference_to_axis2
[commanded_position2]
commanded_pos2
[commanded_velocity2]
commanded_vel2
[commanded_torque2]
commanded_tor2
[actual_position3]
pos3
[actual_velocity3]
vel3
RM_Axis3
MC_Power3
CNB3
[axis3]
reference_to_axis3
[commanded_position3]
commanded_pos3
[commanded_velocity3]
commanded_vel3
[commanded_torque3]
commanded_tor3
[actual_position4]
pos4
[actual_velocity4]
vel4
RM_Axis4
MC_Power4
[axis4]
reference_to_axis4
[commanded_position4]
commanded_pos4
[commanded_velocity4]
commanded_vel4
[commanded_torque4]
commanded_tor4
RM_AxesGroup1234
[axes_group]
reference_to_axesgroup
skupina_os
axis1
CNB
MC_AddAxisToGroup_O1
axis2
MC_AddAxisToGroup_O2
axis3
MC_AddAxisToGroup_O3
axis4
MC_AddAxisToGroup_O4
MCP_SetKinTransform_Agebot
MC_GroupEnable_1234

RM_Feed – * MC Feeder ???

Block Symbol

Licence: COORDINATED MOTION



Function Description

The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

Inputs

uAxesGroup	Axes group reference	Reference
Execute	The block is activated on rising edge	Bool

Parameters

Filename	0		String
VelFactor	0	↓0.01 ↑100.0 ⊙1.0	Double (F64)
Relative	0		Bool
CoordSystem	0	↓1 ↑3 ⊙2	Long (I32)
BufferMode	0	↓1 ↑6 ⊙1	Long (I32)
TransitionMode	0	↓0 ↑15 ⊙1	Long (I32)
TransitionParameter	0		Double (F64)

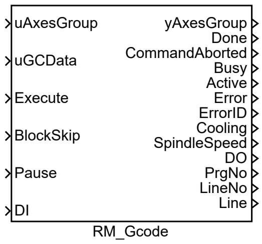
Outputs

yAxesGroup	Axes group reference	Reference
Done	Algorithm finished	Bool
CommandAborted	Algorithm was aborted	Bool
Busy	Algorithm not finished yet	Bool
Active	The block is controlling the axis	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation	Error
Aux	0	Double (F64)

RM_Gcode – * CNC motion control

Block Symbol

Licence: COORDINATED MOTION



Function Description

The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

Inputs

uAxesGroup	Axes group reference	Reference
Execute	The block is activated on rising edge	Bool
BlockSkip	MILAN	Bool

Parameters

BaseDir	Directory of the G-code files		String
MainFile	Source file number		Long (I32)
CoordSystem	0	↓1 ↑3 ⊙3	Long (I32)
BufferMode	Buffering mode 1 ..... Aborting 2 ..... Buffered 3 ..... Blending low 4 ..... Blending high 5 ..... Blending previous 6 ..... Blending next	⊙1	Long (I32)
TransitionMode	Transition mode in blending mode 1 ..... TMNone 2 ..... TMStartVelocity 3 ..... TMConstantVelocity 4 ..... TMCornerDistance 5 ..... TMMaxCornerDeviation 11 .... Smooth	⊙1	Long (I32)
TransitionParameter	Parametr for transition (depends on transition mode)		Double (F64)

workOffsets	Sets with initial coordinate	⊙[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]	Double (F64)
toolOffsets	Sets of tool offset	⊙[0 0 0]	Double (F64)
cutterOffsets	Tool radii	⊙[0 0 0]	Double (F64)

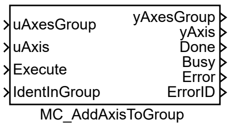
Outputs

yAxesGroup	Axes group reference	Reference
Done	Algorithm finished	Bool
CommandAborted	Algorithm was aborted	Bool
Busy	Algorithm not finished yet	Bool
Active	The block is controlling the axis	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation i ..... REXYGEN general error	Error
Cooling	Cooling	Bool
LineNo	Current executed line number	Long (I32)
Line	Current line of G-code	String
SpindleSpeed	Spindle speed	Double (F64)

MC_AddAxisToGroup – Adds one axis to a group

Block Symbol

Licence: COORDINATED MOTION



Function Description

The function block **MC_AddAxisToGroup** adds one **uAxis** to the group in a structure **uAxesGroup**. Axes Group is implemented by the function block RM_AxesGroup. The input **uAxis** must be defined by the function block RM_Axis from the MC_SINGLE library.

Note 1: Every **IdentInGroup** is unique and can be used only for one time otherwise the error is set.

Inputs

uAxesGroup	Axes group reference	Reference
uAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Execute	The block is activated on rising edge	Bool
IdentInGroup	The order of axes in the group (0 = first unassigned)	Long (I32)

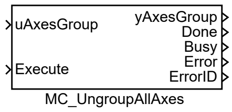
Outputs

yAxesGroup	Axes group reference	Reference
yAxis	Axis reference (only RM_Axis.axisRef-uAxis or yAxis-uAxis connections are allowed)	Reference
Done	Algorithm finished	Bool
Busy	Algorithm not finished yet	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation i ..... REXYGEN general error	Error

MC_UngroupAllAxes – Removes all axes from the group

Block Symbol

Licence: COORDINATED MOTION



Function Description

The function block **MC_UngroupAllAxes** removes all axes from the group **uAxesGroup**. After finalization the state is changed to "GroupDisabled".

Note 1: If the function block is execute in the group state "GroupDisabled", "GroupStandBy" or "GroupErrorStop" the error is set and the block is not execute.

Inputs

uAxesGroup	Axes group reference	Reference
Execute	The block is activated on rising edge	Bool

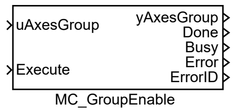
Outputs

yAxesGroup	Axes group reference	Reference
Done	Algorithm finished	Bool
Busy	Algorithm not finished yet	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation i ..... REXYGEN general error	Error

MC_GroupEnable – Changes the state of a group to GroupEnable

Block Symbol

Licence: COORDINATED MOTION



Function Description

The function block **MC_GroupEnable** changes the state for the group **uAxesGroup** from "GroupDisabled" to "GroupStandby".

Inputs

uAxesGroup	Axes group reference	Reference
Execute	The block is activated on rising edge	Bool

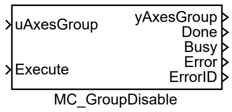
Outputs

yAxesGroup	Axes group reference	Reference
Done	Algorithm finished	Bool
Busy	Algorithm not finished yet	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation i ..... REXYGEN general error	Error

MC_GroupDisable – Changes the state of a group to GroupDisabled

Block Symbol

Licence: COORDINATED MOTION



Function Description

The function block `MC_GroupDisable` changes the state for the group `uAxesGroup` to "GroupDisabled". If the axes are not standing still while issuing this command the state of the group is changed to "Stopping". It is mean stopping with the maximal allowed deceleration. When stopping is done the state of the group is changed to "GroupDisabled".

Inputs

`uAxesGroup`	Axes group reference	`Reference`
`Execute`	The block is activated on rising edge	`Bool`

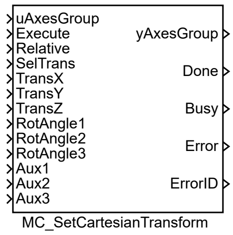
Outputs

`yAxesGroup`	Axes group reference	`Reference`
`Done`	Algorithm finished	`Bool`
`Busy`	Algorithm not finished yet	`Bool`
`Error`	Error occurred	`Bool`
`ErrorID`	Result of the last operation `i` ..... REXYGEN general error	`Error`

MC_SetCartesianTransform – Sets Cartesian transformation

Block Symbol

Licence: COORDINATED MOTION



Function Description

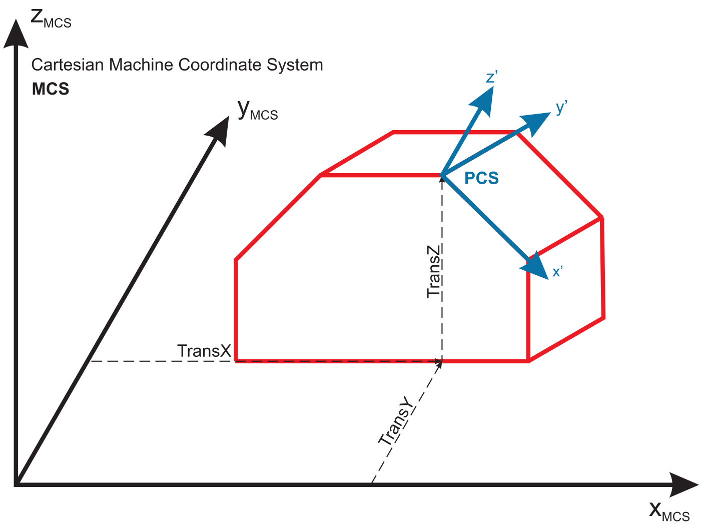
The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

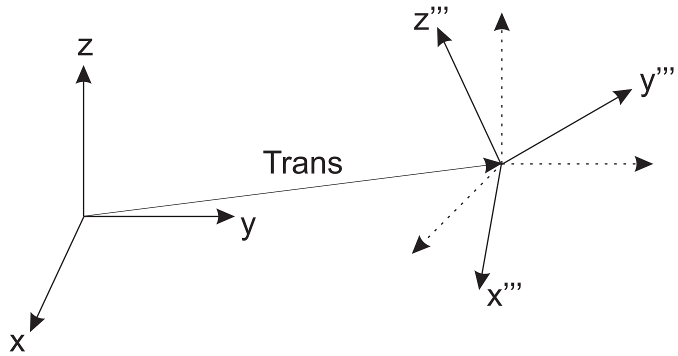
Inputs

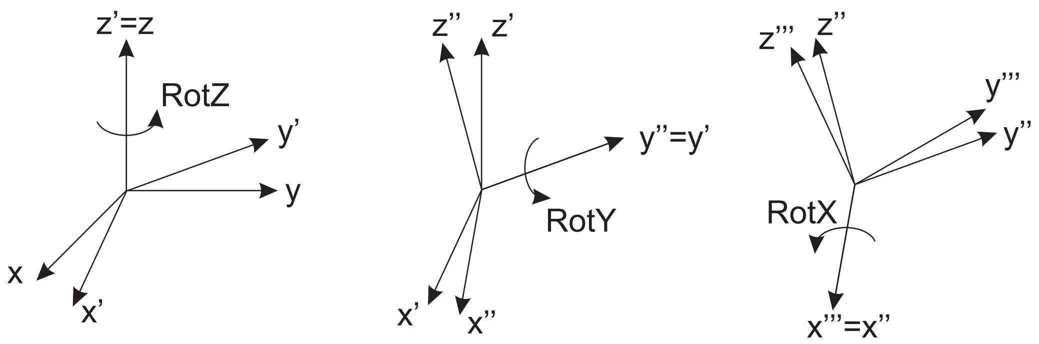
`uAxesGroup`	Axes group reference	`Reference`
`Execute`	The block is activated on rising edge	`Bool`
`TransX`	X-component of translation vector	`Double (F64)`
`TransY`	Y-component of translation vector	`Double (F64)`
`TransZ`	Z-component of translation vector	`Double (F64)`
`RotAngle1`	Rotation angle component	`Double (F64)`
`RotAngle2`	Rotation angle component	`Double (F64)`
`RotAngle3`	Rotation angle component	`Double (F64)`
`Relative`	Mode of position inputs	`Bool`

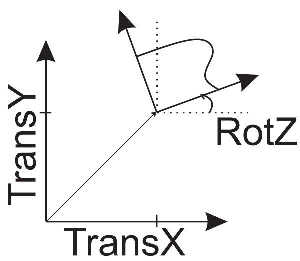
Outputs

`yAxesGroup`	Axes group reference	`Reference`
`Done`	Algorithm finished	`Bool`
`Busy`	Algorithm not finished yet	`Bool`
`Error`	Error occurred	`Bool`
`ErrorID`	Result of the last operation `i` ..... REXYGEN general error	`Error`


z$_{MCS}$
Cartesian Machine Coordinate System
MCS
y$_{MCS}$
z'
y'
PCS
x'
TransZ
TransX
TransY
x$_{MCS}$


z
z'''
y'''
Trans
y
x'''
x

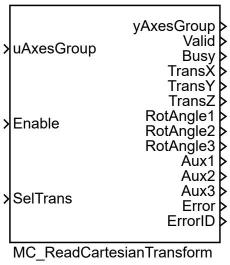

z'=z
RotZ
y'
y
x
x'
z''
z'
y''=y'
RotY
x'
x''
z'''
z''
y'''
y''
RotX
x'''=x''


TransY
RotZ
TransX

MC_ReadCartesianTransform – Reads the parameter of the cartesian transformation

Block Symbol

Licence: COORDINATED MOTION



Function Description

The function block `MC_ReadCartesianTransform` reads the parameter of the cartesian transformation that is active between the MCS and PCS. The parameters are valid only if the output `Valid` is true which is achieved by setting the input `Enable` on true. If more than one transformation is active, the resulting cartesian transformation is given.

Inputs

`uAxesGroup`	Axes group reference	`Reference`
`Enable`	Block function is enabled	`Bool`

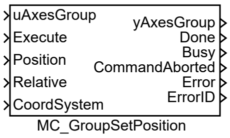
Outputs

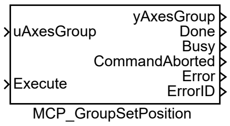
`yAxesGroup`	Axes group reference	`Reference`
`Valid`	Output value is valid	`Bool`
`Busy`	Algorithm not finished yet	`Bool`
`TransX`	X-component of translation vector	`Double (F64)`
`TransY`	Y-component of translation vector	`Double (F64)`
`TransZ`	Z-component of translation vector	`Double (F64)`
`RotAngle1`	Rotation angle component	`Double (F64)`
`RotAngle2`	Rotation angle component	`Double (F64)`
`RotAngle3`	Rotation angle component	`Double (F64)`
`Error`	Error occurred	`Bool`
`ErrorID`	Result of the last operation `i` ..... REXYGEN general error	`Error`

MC_GroupSetPosition, MCP_GroupSetPosition – Sets the position of all axes in a group

Block Symbols

Licence: COORDINATED MOTION





Function Description

The MC_GroupSetPosition *and* MCP_GroupSetPosition *blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the* MCP_ *version of the block.*

The function block MC_GroupSetPosition sets the position of all axes in the group uAxesGroup without moving the axes. The new coordinates are described by the input Position. With the coordinate system input CoordSystem the according coordinate system is selected. The function block MC_GroupSetPosition shifts position of the addressed coordinate system and affect the higher level coordinate systems (so if ACS selected, MCS and PCS are affected).

Inputs

uAxesGroup	Axes group reference	Reference
Execute	The block is activated on rising edge	Bool
Position	Array of coordinates (positions and orientations)	Reference
Relative	Mode of position inputs off ... absolute on .... relative	Bool
CoordSystem	Reference to the coordinate system used 1 ..... ACS 2 ..... MCS 3 ..... PCS	Long (I32)

Outputs

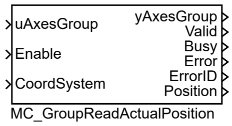
yAxesGroup	Axes group reference	Reference
Done	Algorithm finished	Bool
Busy	Algorithm not finished yet	Bool
CommandAborted	Algorithm was aborted	Bool
Error	Error occurred	Bool

ErrorID	Result of the last operation i ..... REXYGEN general error	Error

MC_GroupReadActualPosition – Read actual position in the selected coordinate system

Block Symbol

Licence: COORDINATED MOTION



Function Description

The function block MC_GroupReadActualPosition returns the actual position in the selected coordinate system of an axes group. The position is valid only if the output Valid is true which is achieved by setting the input Enable on true.

Inputs

uAxesGroup	Axes group reference	Reference
Enable	Block function is enabled	Bool
CoordSystem	Reference to the coordinate system used 1 ..... ACS 2 ..... MCS 3 ..... PCS	Long (I32)

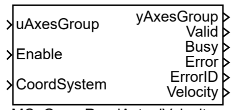
Outputs

yAxesGroup	Axes group reference	Reference
Valid	Output value is valid	Bool
Busy	Algorithm not finished yet	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation i ..... REXYGEN general error	Error
Position	xxx	Reference

MC_GroupReadActualVelocity – Read actual velocity in the selected coordinate system

Block Symbol

Licence: COORDINATED MOTION



MC_GroupReadActualVelocity

Function Description

The function block `MC_GroupReadActualVelocity` returns the actual velocity in the selected coordinate system of an axes group. The position is valid only if the output `Valid` is true which is achieved by setting the input `Enable` on true.

Inputs

`uAxesGroup`	Axes group reference	`Reference`
`Enable`	Block function is enabled	`Bool`
`CoordSystem`	Reference to the coordinate system used `1` ..... ACS `2` ..... MCS `3` ..... PCS	`Long (I32)`

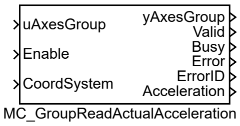
Outputs

`yAxesGroup`	Axes group reference	`Reference`
`Valid`	Output value is valid	`Bool`
`Busy`	Algorithm not finished yet	`Bool`
`Error`	Error occurred	`Bool`
`ErrorID`	Result of the last operation `i` ..... REXYGEN general error	`Error`
`Velocity`	xxx	`Reference`

`MC_GroupReadActualAcceleration` – Read actual acceleration in the selected coordinate system

Block Symbol

Licence: COORDINATED MOTION



Function Description

The function block `MC_GroupReadActualAcceleration` returns the actual velocity in the selected coordinate system of an axes group. The position is valid only if the output `Valid` is true which is achieved by setting the input `Enable` on true.

Inputs

`uAxesGroup`	Axes group reference	`Reference`
`Enable`	Block function is enabled	`Bool`
`CoordSystem`	Reference to the coordinate system used `1` ..... ACS `2` ..... MCS `3` ..... PCS	`Long (I32)`

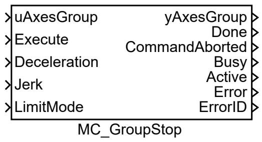
Outputs

`yAxesGroup`	Axes group reference	`Reference`
`Valid`	Output value is valid	`Bool`
`Busy`	Algorithm not finished yet	`Bool`
`Error`	Error occurred	`Bool`
`ErrorID`	Result of the last operation `i` ..... REXYGEN general error	`Error`
`Acceleration`	xxx	`Reference`

MC_GroupStop – Stopping a group movement

Block Symbol

Licence: COORDINATED MOTION



Function Description

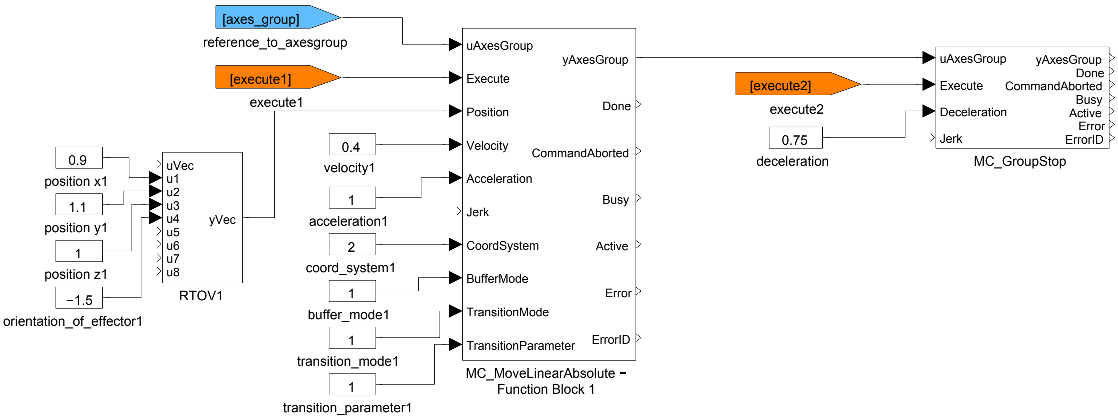
The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

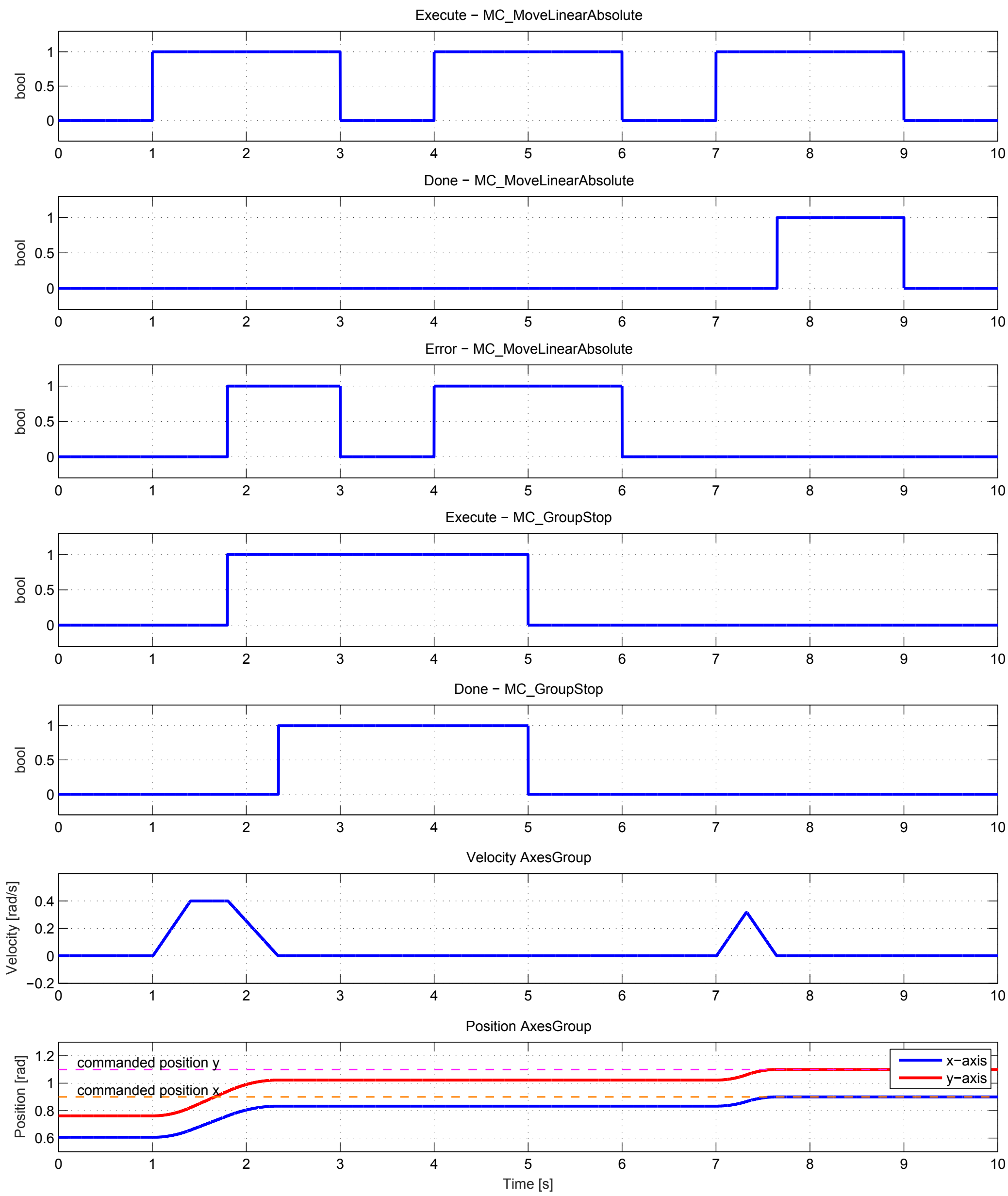
Inputs

uAxesGroup	Axes group reference	Reference
Execute	The block is activated on rising edge	Bool
Deceleration	Maximal allowed deceleration [unit/s^2]	Double (F64)
Jerk	Maximal allowed jerk [unit/s^3]	Double (F64)

Outputs

yAxesGroup	Axes group reference	Reference
Done	Algorithm finished	Bool
CommandAborted	Algorithm was aborted	Bool
Busy	Algorithm not finished yet	Bool
Active	The block is controlling the axis	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation i ..... REXYGEN general error	Error


[axes_group]
reference_to_axesgroup
[execute1]
execute1
0.9
position x1
1.1
position y1
1
position z1
−1.5
orientation_of_effector1
uVec
u1
u2
u3
u4
u5
u6
u7
u8
yVec
RTOV1
0.4
velocity1
1
acceleration1
2
coord_system1
1
buffer_mode1
1
transition_mode1
1
transition_parameter1
uAxesGroup
Execute
Position
Velocity
Acceleration
Jerk
CoordSystem
BufferMode
TransitionMode
TransitionParameter
yAxesGroup
Done
CommandAborted
Busy
Active
Error
ErrorID
MC_MoveLinearAbsolute − Function Block 1
[execute2]
execute2
0.75
deceleration
uAxesGroup
Execute
Deceleration
Jerk
yAxesGroup
Done
CommandAborted
Busy
Active
Error
ErrorID
MC_GroupStop

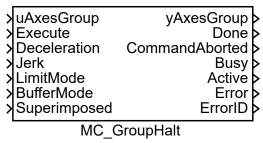


Execute − MC_MoveLinearAbsolute
Done − MC_MoveLinearAbsolute
Error − MC_MoveLinearAbsolute
Execute − MC_GroupStop
Done − MC_GroupStop
Velocity AxesGroup
Position AxesGroup
bool
Velocity [rad/s]
Position [rad]
commanded position y
commanded position x
x−axis
y−axis
Time [s]

MC_GroupHalt – Stopping a group movement (interruptible)

Block Symbol

Licence: COORDINATED MOTION



Function Description

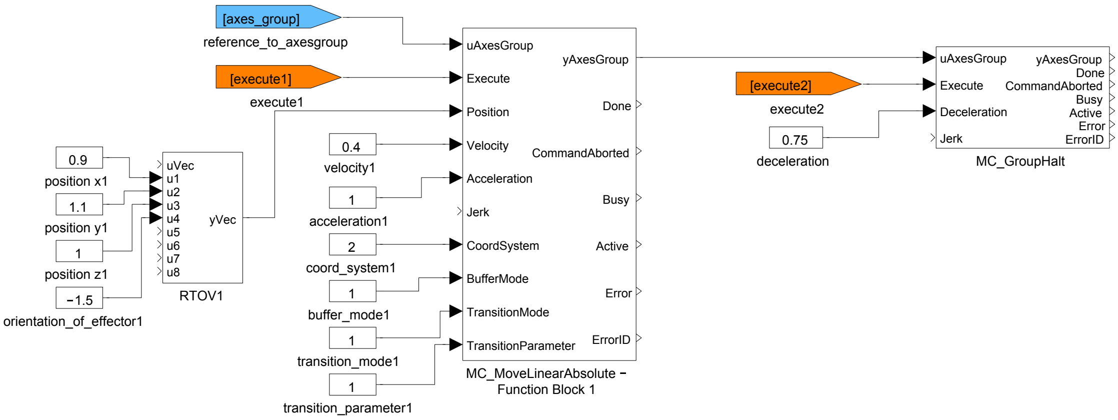
The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

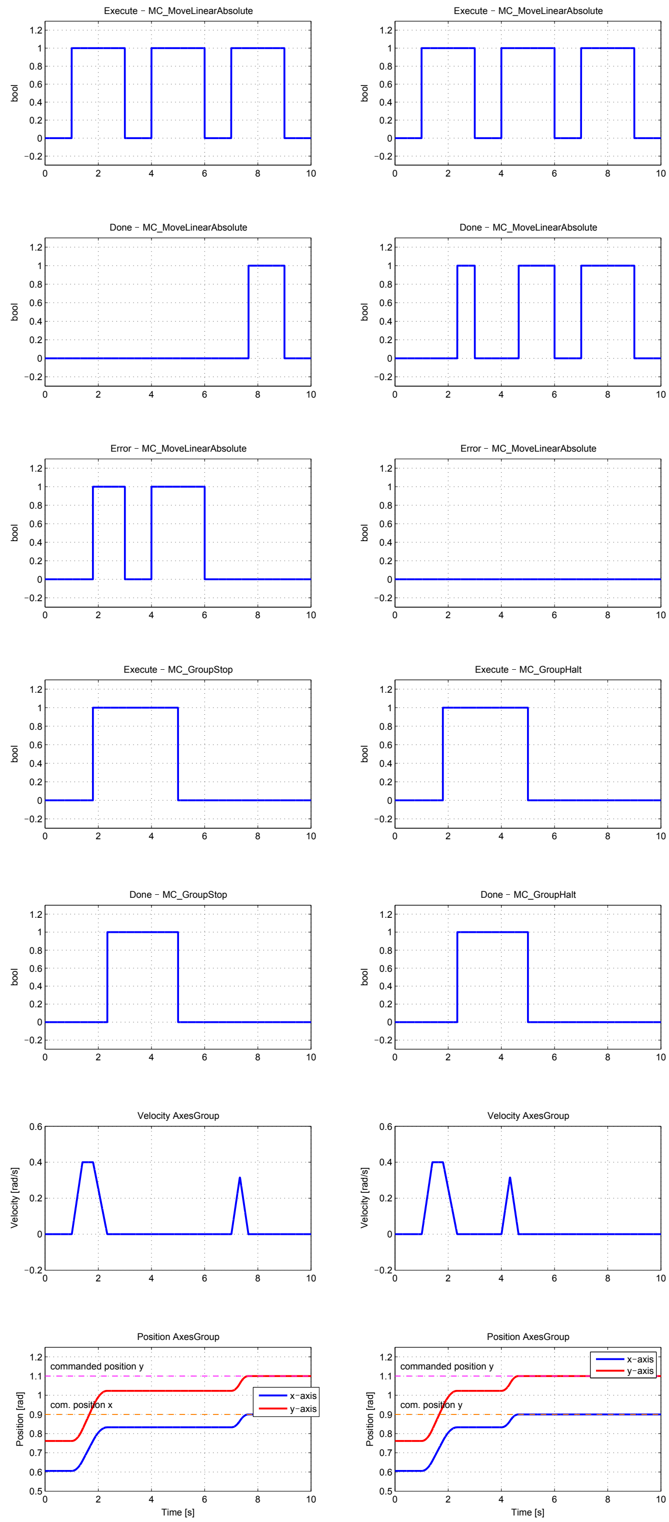
Inputs

uAxesGroup	Axes group reference	Reference
Execute	The block is activated on rising edge	Bool
Deceleration	Maximal allowed deceleration [unit/s^2]	Double (F64)
Jerk	Maximal allowed jerk [unit/s^3]	Double (F64)

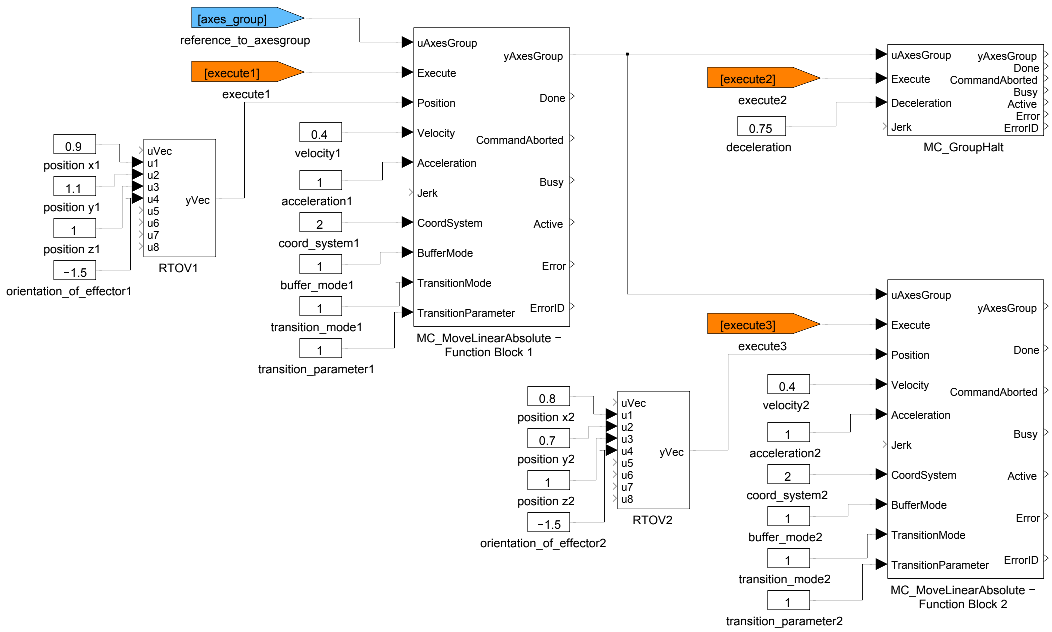
Outputs

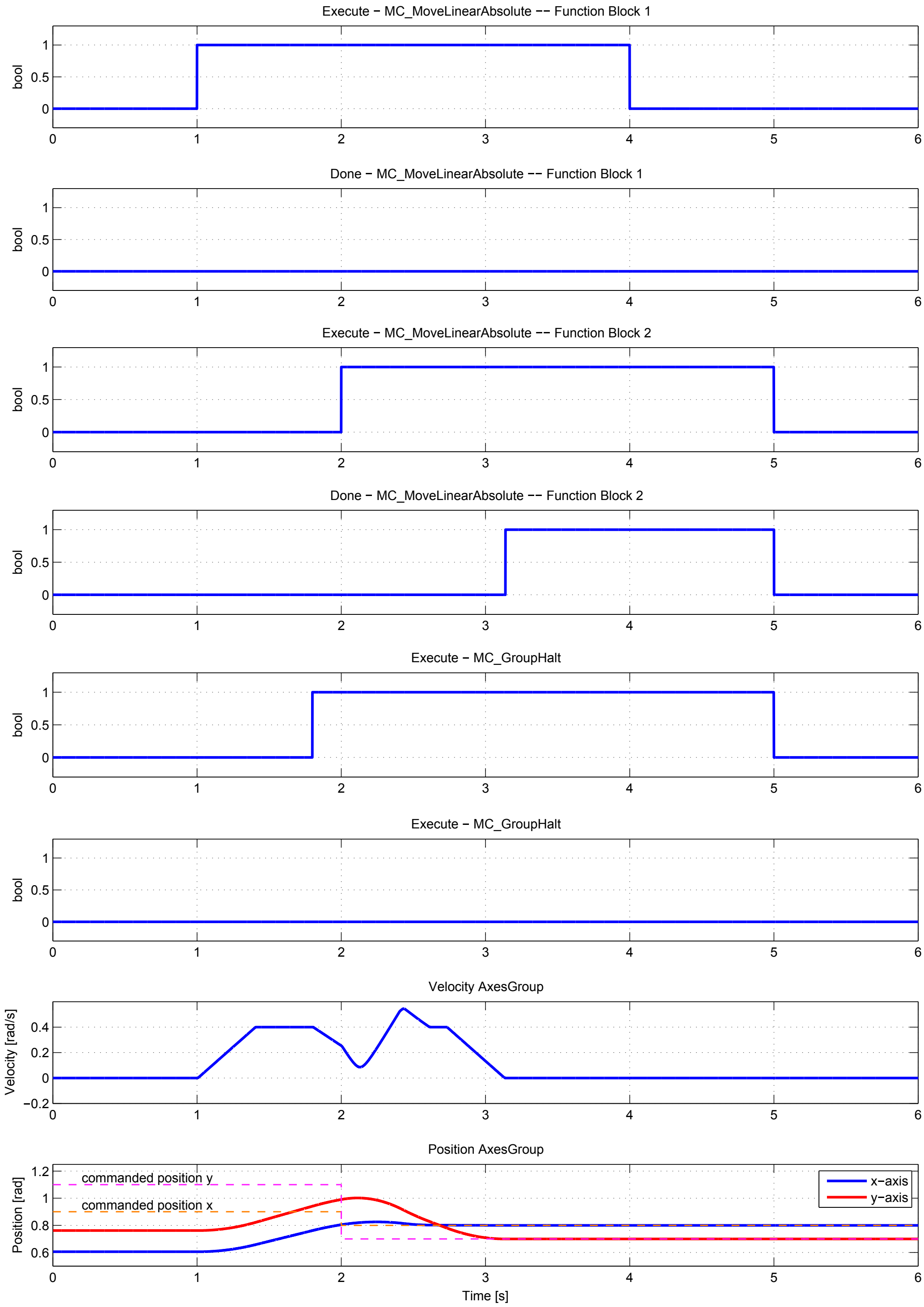
yAxesGroup	Axes group reference	Reference
Done	Algorithm finished	Bool
CommandAborted	Algorithm was aborted	Bool
Busy	Algorithm not finished yet	Bool
Active	The block is controlling the axis	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation i ..... REXYGEN general error	Error


[axes_group]
reference_to_axesgroup
[execute1]
execute1
0.9
position x1
1.1
position y1
1
position z1
−1.5
orientation_of_effector1
uVec
u1
u2
u3
u4
u5
u6
u7
u8
yVec
RTOV1
0.4
velocity1
1
acceleration1
2
coord_system1
1
buffer_mode1
1
transition_mode1
1
transition_parameter1
uAxesGroup
Execute
Position
Velocity
Acceleration
Jerk
CoordSystem
BufferMode
TransitionMode
TransitionParameter
yAxesGroup
Done
CommandAborted
Busy
Active
Error
ErrorID
MC_MoveLinearAbsolute −
Function Block 1
[execute2]
execute2
0.75
deceleration
uAxesGroup
Execute
Deceleration
Jerk
yAxesGroup
Done
CommandAborted
Busy
Active
Error
ErrorID
MC_GroupHalt



Execute - MC_MoveLinearAbsolute
Done - MC_MoveLinearAbsolute
Error - MC_MoveLinearAbsolute
Execute - MC_GroupStop
Execute - MC_GroupHalt
Done - MC_GroupStop
Done - MC_GroupHalt
Velocity AxesGroup
Position AxesGroup
bool
Velocity [rad/s]
Position [rad]
Time [s]
commanded position y
com. position x
com. position y
x-axis
y-axis

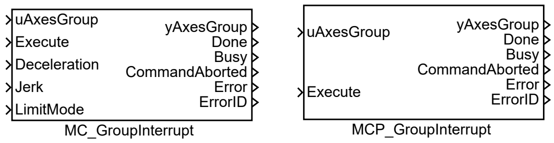

[axes_group]
reference_to_axesgroup
[execute1]
execute1
0.9
position x1
1.1
position y1
1
position z1
−1.5
orientation_of_effector1
uVec
u1
u2
u3
u4
u5
u6
u7
u8
yVec
RTOV1
0.4
velocity1
1
acceleration1
2
coord_system1
1
buffer_mode1
1
transition_mode1
1
transition_parameter1
uAxesGroup
Execute
Position
Velocity
Acceleration
Jerk
CoordSystem
BufferMode
TransitionMode
TransitionParameter
yAxesGroup
Done
CommandAborted
Busy
Active
Error
ErrorID
MC_MoveLinearAbsolute − Function Block 1
[execute2]
execute2
0.75
deceleration
uAxesGroup
Execute
Deceleration
Jerk
yAxesGroup
Done
CommandAborted
Busy
Active
Error
ErrorID
MC_GroupHalt
[execute3]
execute3
0.8
position x2
0.7
position y2
1
position z2
−1.5
orientation_of_effector2
RTOV2
0.4
velocity2
1
acceleration2
2
coord_system2
1
buffer_mode2
1
transition_mode2
1
transition_parameter2
MC_MoveLinearAbsolute − Function Block 2


Execute − MC_MoveLinearAbsolute −− Function Block 1
Done − MC_MoveLinearAbsolute −− Function Block 1
Execute − MC_MoveLinearAbsolute −− Function Block 2
Done − MC_MoveLinearAbsolute −− Function Block 2
Execute − MC_GroupHalt
Execute − MC_GroupHalt
Velocity AxesGroup
Position AxesGroup
bool
Velocity [rad/s]
Position [rad]
Time [s]
commanded position y
commanded position x
x−axis
y−axis

MC_GroupInterrupt, MCP_GroupInterrupt – **Read a group interrupt**

Block Symbols

Licence: COORDINATED MOTION



Function Description

The MC_GroupInterrupt *and* MCP_GroupInterrupt *blocks offer the same functionality, the only difference is that some of the inputs are available as parameters in the* MCP_ *version of the block.*

The function block MC_GroupInterrupt interrupts the on-going motion and stops the group from moving, however does not abort the interrupted motion (meaning that at the interrupted FB the output CommandAborted will not be Set, Busy is still high and Active is reset). It stores all relevant track or path information internally at the moment it becomes active. The uAxesGroup stays in the original state even if the velocity zero is reached and the Done output is set.

Note 1: This function block is complementary to the function block MC_GroupContinue which execution the uAxesGroup state is reset to the original state (before MC_GroupInterrupt execution)

Inputs

uAxesGroup	Axes group reference	Reference
Execute	The block is activated on rising edge	Bool
Deceleration	Maximal allowed deceleration [unit/s^2]	Double (F64)
Jerk	Maximal allowed jerk [unit/s^3]	Double (F64)

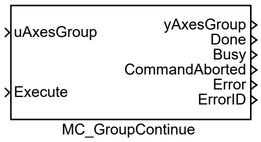
Outputs

yAxesGroup	Axes group reference	Reference
Done	Algorithm finished	Bool
Busy	Algorithm not finished yet	Bool
CommandAborted	Algorithm was aborted	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation i ..... REXYGEN general error	Error

MC_GroupContinue – Continuation of interrupted movement

Block Symbol

Licence: COORDINATED MOTION



Function Description

The function block MC_GroupContinue transfers the program back to the situation at issuing MC_GroupInterrupt. It uses internally the data set as stored at issuing MC_GroupInterrupt, and at the end (output Done set) transfer the control on the group back to the original FB doing the movements on the axes group, meaning also that at the originally interrupted FB the output Busy is still high and the output Active is set again.

Inputs

uAxesGroup	Axes group reference	Reference
Execute	The block is activated on rising edge	Bool

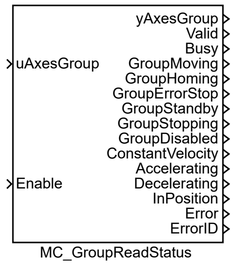
Outputs

yAxesGroup	Axes group reference	Reference
Done	Algorithm finished	Bool
Busy	Algorithm not finished yet	Bool
CommandAborted	Algorithm was aborted	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation i ..... REXYGEN general error	Error

MC_GroupReadStatus – Read a group status

Block Symbol

Licence: COORDINATED MOTION



Function Description

The function block MC_GroupReadStatus returns the status of the uAxesGroup. The status is valid only if the output Valid is true which is achieved by setting the input Enable on true.

Inputs

uAxesGroup	Axes group reference	Reference
Enable	Block function is enabled	Bool

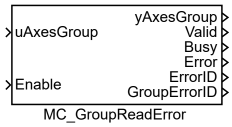
Outputs

yAxesGroup	Axes group reference	Reference
Valid	Output value is valid	Bool
Busy	Algorithm not finished yet	Bool
GroupMoving	State GroupMoving	Bool
GroupHoming	State GroupHoming	Bool
GroupErrorStop	State ErrorStop	Bool
GroupStandby	State Standby	Bool
GroupStopping	State Stopping	Bool
GroupDisabled	State Disabled	Bool
ConstantVelocity	Constant velocity motion	Bool
Accelerating	Accelerating	Bool
Decelerating	Decelerating	Bool
InPosition	Symptom achieve the desired position	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation i ..... REXYGEN general error	Error

MC_GroupReadError – Read a group error

Block Symbol

Licence: COORDINATED MOTION



Function Description

The function block MC_GroupReadError describes general error on the uAxesGroup which is not relating to the function blocks. If the output GroupErrorID is equal to 0 there is no error on the axes group. The actual error code GroupErrorID is valid only if the output Valid is true which is achieved by setting the input Enable on true.

Note 1: This function block is implemented because of compatibility with the PLCopen norm. The same error value is on the output ErrorID of the function block RM_AxesGroup.

Inputs

uAxesGroup	Axes group reference	Reference
Enable	Block function is enabled	Bool

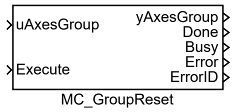
Outputs

yAxesGroup	Axes group reference	Reference
Valid	Output value is valid	Bool
Busy	Algorithm not finished yet	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation i ..... REXYGEN general error	Error
GroupErrorID	Result of the last operation i ..... REXYGEN general error	Error

MC_GroupReset – Reset axes errors

Block Symbol

Licence: COORDINATED MOTION



Function Description

The function block MC_GroupReset makes the transition from the state "GroupErrorStop" to "GroupStandBy" by resetting all internal group-related errors. This function block also resets all axes in this group like the function block MC_Reset from the MC_SINGLE library.

Inputs

uAxesGroup	Axes group reference	Reference
Execute	The block is activated on rising edge	Bool

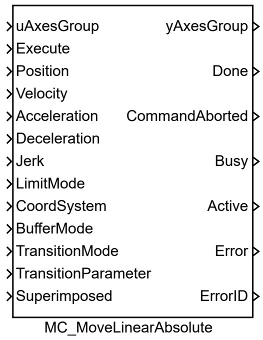
Outputs

yAxesGroup	Axes group reference	Reference
Done	Algorithm finished	Bool
Busy	Algorithm not finished yet	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation i ..... REXYGEN general error	Error

MC_MoveLinearAbsolute – Linear move to position (absolute coordinates)

Block Symbol

Licence: COORDINATED MOTION



Function Description

The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

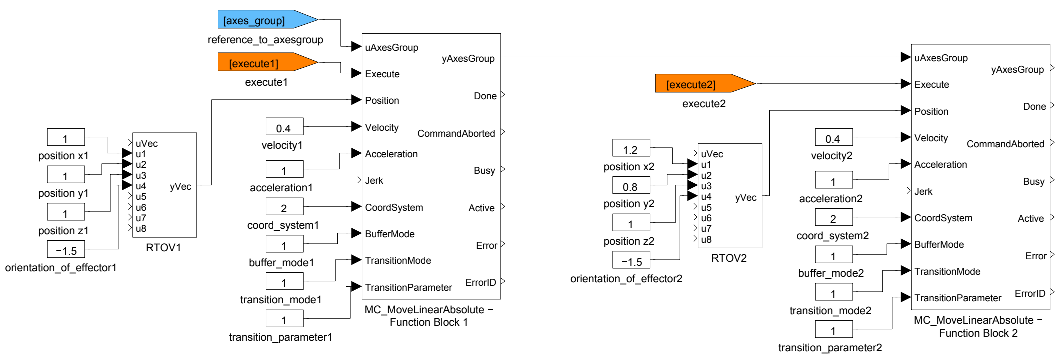
Inputs

`uAxesGroup`	Axes group reference	`Reference`
`Execute`	The block is activated on rising edge	`Bool`
`Position`	Array of coordinates (positions and orientations)	`Reference`
`Velocity`	Maximal allowed velocity [unit/s]	`Double (F64)`
`Acceleration`	Maximal allowed acceleration [unit/s^2]	`Double (F64)`
`Jerk`	Maximal allowed jerk [unit/s^3]	`Double (F64)`
`CoordSystem`	Reference to the coordinate system used `1` ..... ACS `2` ..... MCS `3` ..... PCS	`Long (I32)`
`BufferMode`	Buffering mode `1` ..... Aborting `2` ..... Buffered `3` ..... Blending low `4` ..... Blending high `5` ..... Blending previous `6` ..... Blending next	`Long (I32)`

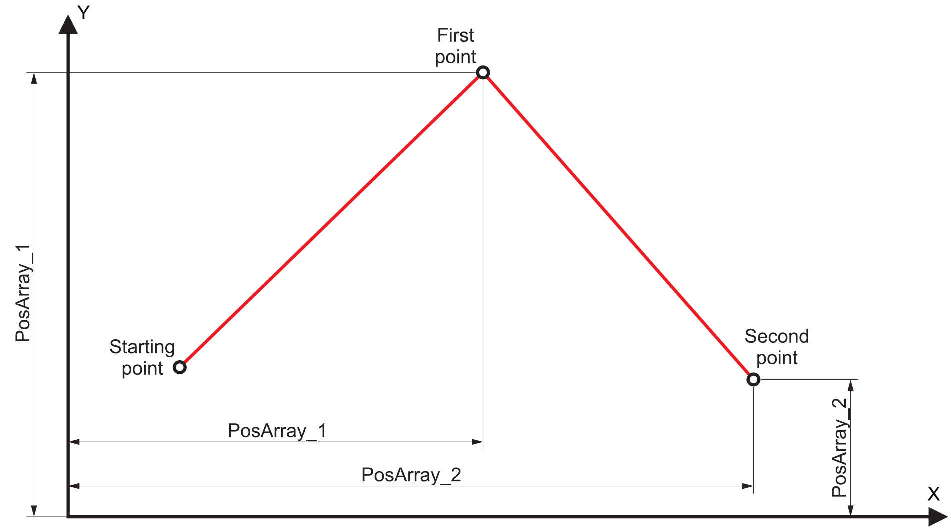
`TransitionMode`	Transition mode in blending mode	`Long (I32)`
	`1` ..... TMNone	
	`2` ..... TMStartVelocity	
	`3` ..... TMConstantVelocity	
	`4` ..... TMCornerDistance	
	`5` ..... TMMaxCornerDeviation	
	`11` .... Smooth	
`TransitionParameter`	Parametr for transition (depends on transition mode)	`Double (F64)`

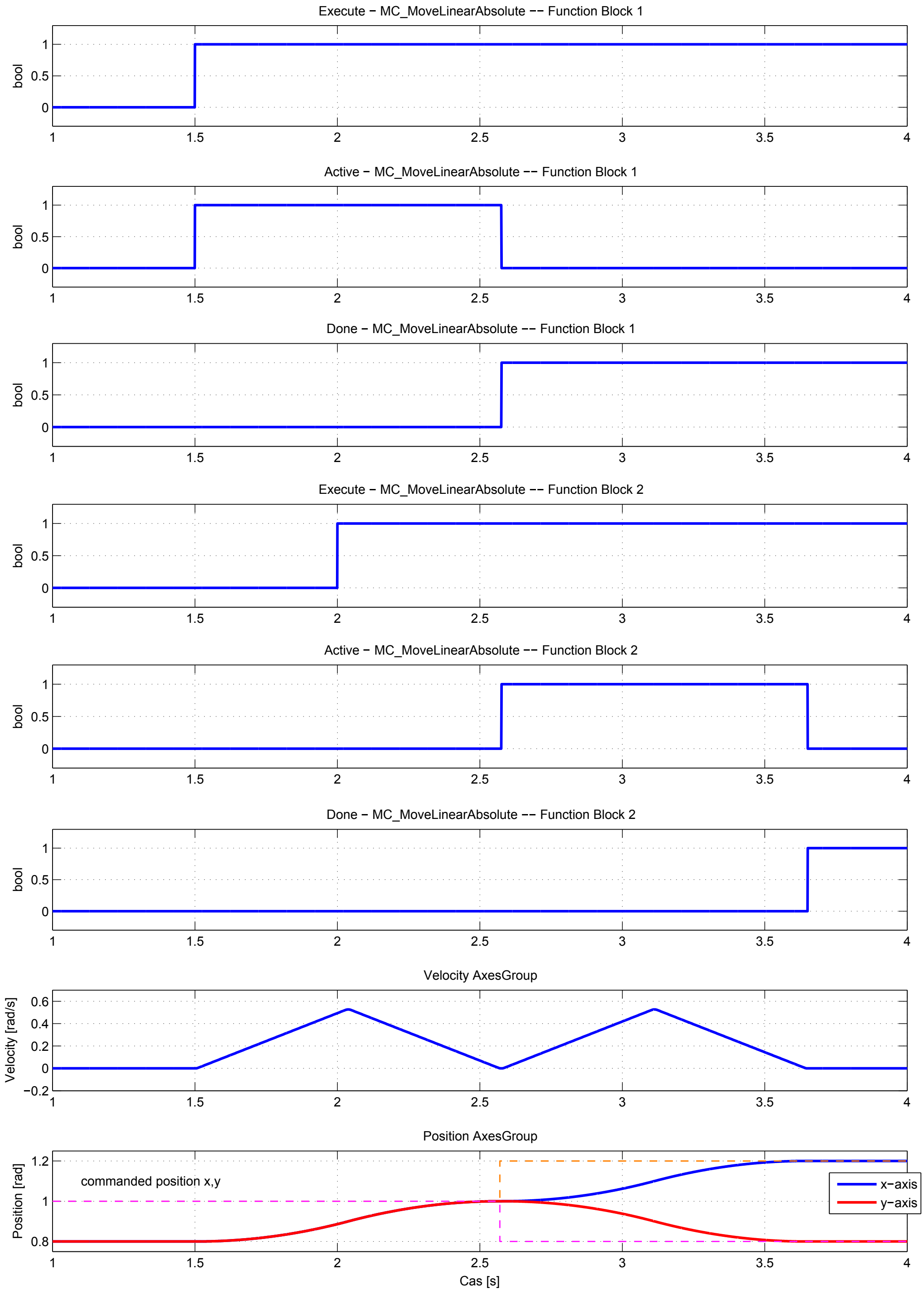
Outputs

`yAxesGroup`	Axes group reference	`Reference`
`Done`	Algorithm finished	`Bool`
`CommandAborted`	Algorithm was aborted	`Bool`
`Busy`	Algorithm not finished yet	`Bool`
`Active`	The block is controlling the axis	`Bool`
`Error`	Error occurred	`Bool`
`ErrorID`	Result of the last operation	`Error`
	`i` ..... REXYGEN general error	



Sequence of two complete motions (Done>Execute)

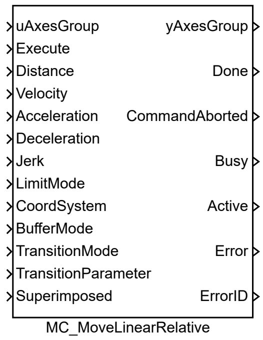



Execute − MC_MoveLinearAbsolute −− Function Block 1
Active − MC_MoveLinearAbsolute −− Function Block 1
Done − MC_MoveLinearAbsolute −− Function Block 1
Execute − MC_MoveLinearAbsolute −− Function Block 2
Active − MC_MoveLinearAbsolute −− Function Block 2
Done − MC_MoveLinearAbsolute −− Function Block 2
bool
Velocity AxesGroup
Velocity [rad/s]
Position AxesGroup
Position [rad]
commanded position x,y
x−axis
y−axis
Cas [s]

MC_MoveLinearRelative – Linear move to position (relative to execution point)

Block Symbol

Licence: COORDINATED MOTION



Function Description

The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

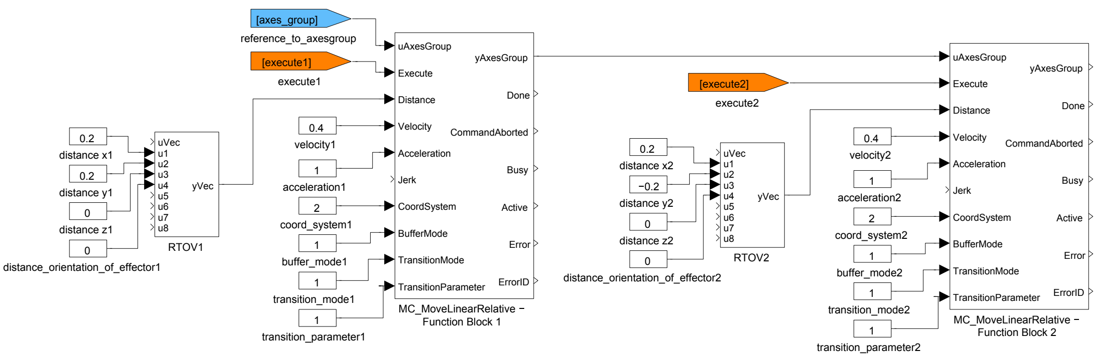
Inputs

`uAxesGroup`	Axes group reference	`Reference`
`Execute`	The block is activated on rising edge	`Bool`
`Distance`	Array of coordinates (relative distances and orientations)	`Reference`
`Velocity`	Maximal allowed velocity [unit/s]	`Double (F64)`
`Acceleration`	Maximal allowed acceleration [unit/s^2]	`Double (F64)`
`Jerk`	Maximal allowed jerk [unit/s^3]	`Double (F64)`
`CoordSystem`	Reference to the coordinate system used 1 ..... ACS 2 ..... MCS 3 ..... PCS	`Long (I32)`
`BufferMode`	Buffering mode 1 ..... Aborting 2 ..... Buffered 3 ..... Blending low 4 ..... Blending high 5 ..... Blending previous 6 ..... Blending next	`Long (I32)`

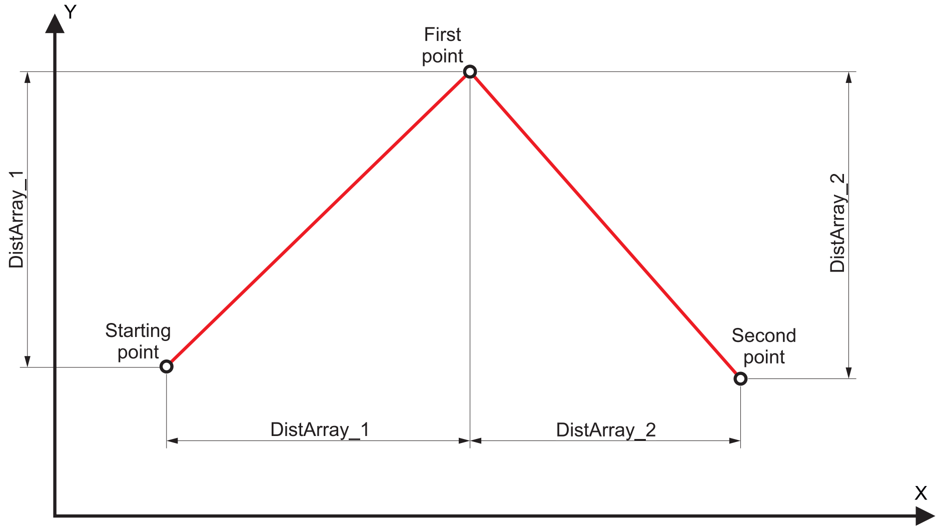
`TransitionMode`	Transition mode in blending mode 1 ..... TMNone 2 ..... TMStartVelocity 3 ..... TMConstantVelocity 4 ..... TMCornerDistance 5 ..... TMMaxCornerDeviation 11 .... Smooth	`Long (I32)`
`TransitionParameter`	Parametr for transition (depends on transition mode)	`Double (F64)`

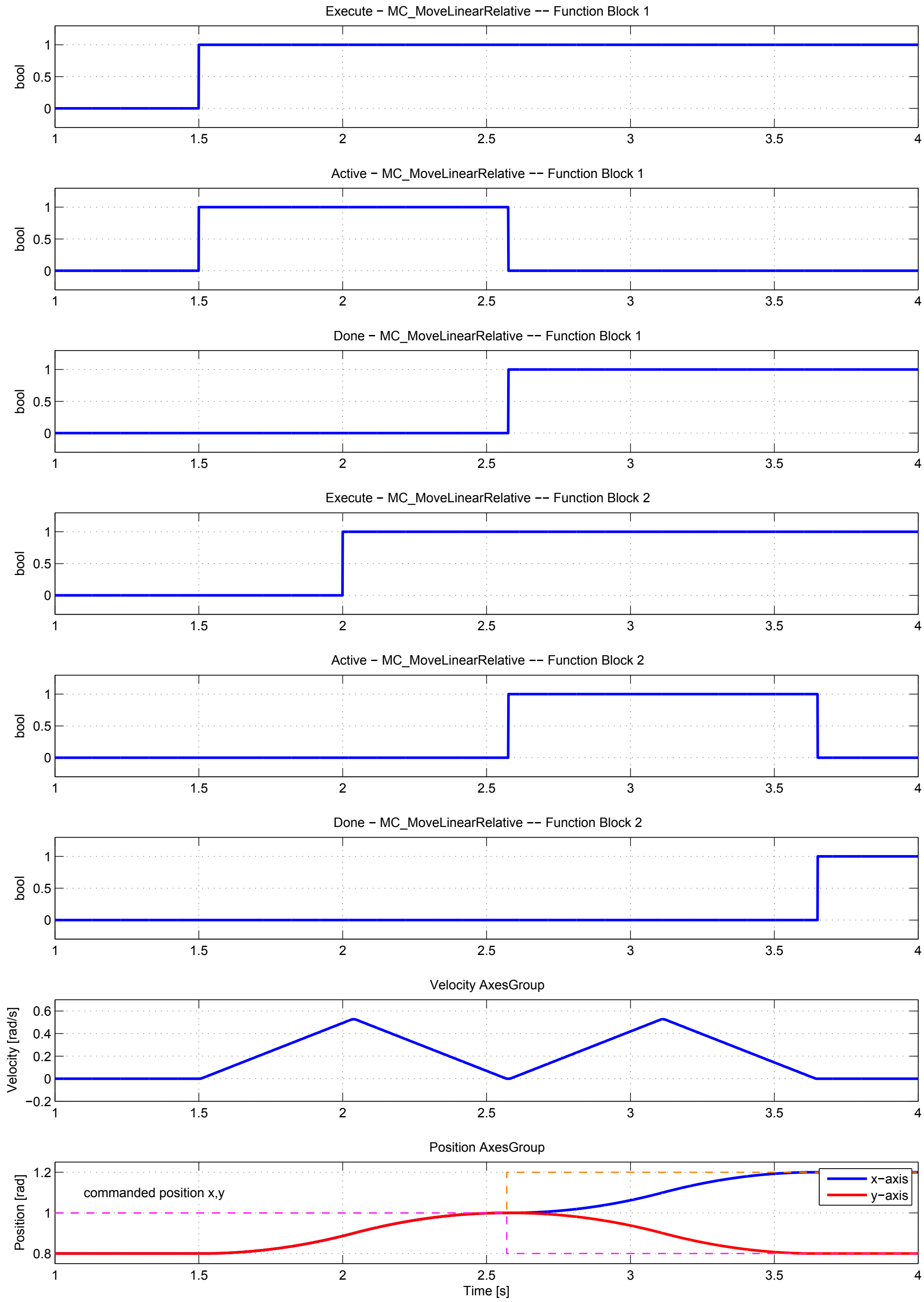
Outputs

`yAxesGroup`	Axes group reference	`Reference`
`Done`	Algorithm finished	`Bool`
`CommandAborted`	Algorithm was aborted	`Bool`
`Busy`	Algorithm not finished yet	`Bool`
`Active`	The block is controlling the axis	`Bool`
`Error`	Error occurred	`Bool`
`ErrorID`	Result of the last operation i ..... REXYGEN general error	`Error`



Sequence of two complete motions (Done>Execute)

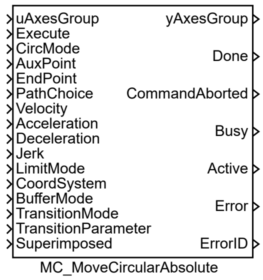



Execute – MC_MoveLinearRelative –– Function Block 1
Active – MC_MoveLinearRelative –– Function Block 1
Done – MC_MoveLinearRelative –– Function Block 1
Execute – MC_MoveLinearRelative –– Function Block 2
Active – MC_MoveLinearRelative –– Function Block 2
Done – MC_MoveLinearRelative –– Function Block 2
bool
Velocity AxesGroup
Velocity [rad/s]
Position AxesGroup
Position [rad]
commanded position x,y
x–axis
y–axis
Time [s]

MC_MoveCircularAbsolute – Circular move to position (absolute coordinates)

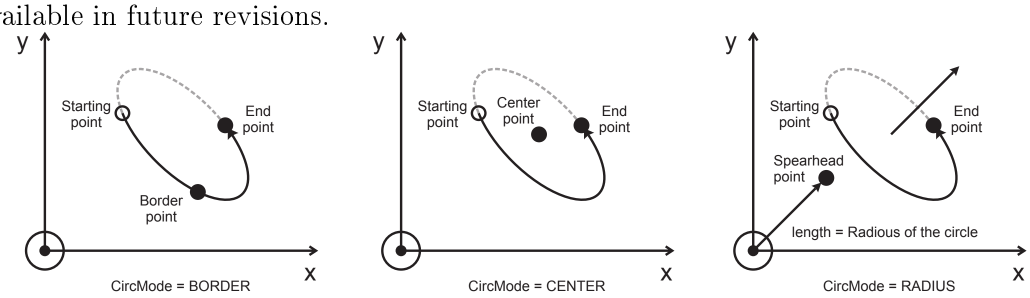
Block Symbol

Licence: COORDINATED MOTION



Function Description

The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.



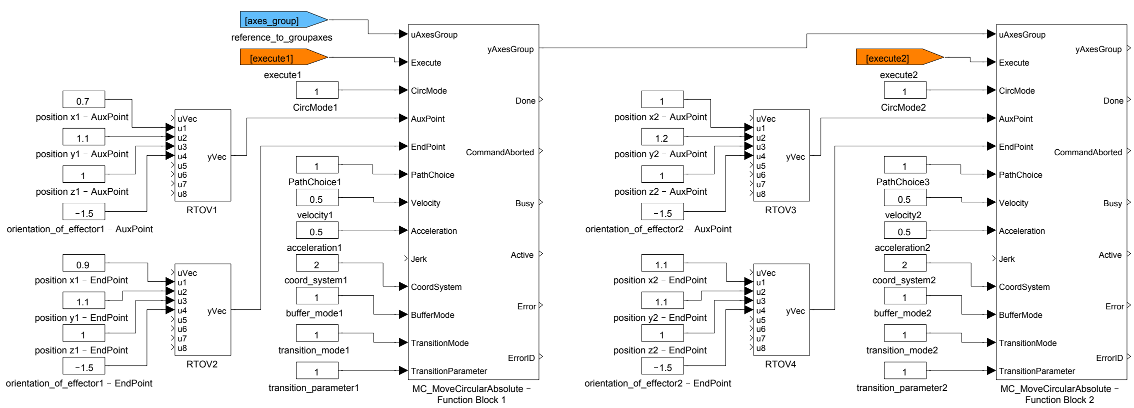
Inputs

`uAxesGroup`	Axes group reference	`Reference`
`Execute`	The block is activated on rising edge	`Bool`
`CircMode`	Specifies the meaning of the input signals AuxPoint and CircDirection 1 ..... BORDER 2 ..... CENTER 3 ..... RADIUS	`Long (I32)`
`AuxPoint`	Next coordinates to define circle (depend on CircMode)	`Reference`
`EndPoint`	Target axes coordinates position	`Reference`
`PathChoice`	Choice of path 1 ..... Clockwise 2 ..... CounterClockwise	`Long (I32)`
`Velocity`	Maximal allowed velocity [unit/s]	`Double (F64)`
`Acceleration`	Maximal allowed acceleration [unit/s^2]	`Double (F64)`

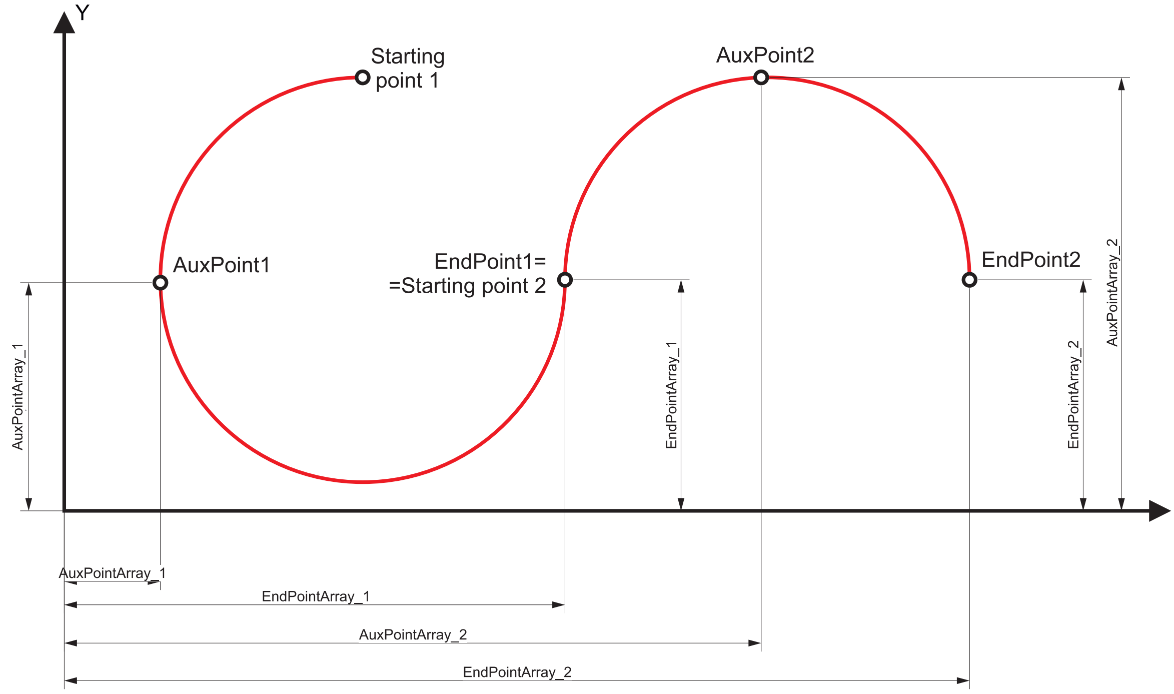
`Jerk`	Maximal allowed jerk [unit/s^3]	`Double (F64)`
`CoordSystem`	Reference to the coordinate system used 1 ..... ACS 2 ..... MCS 3 ..... PCS	`Long (I32)`
`BufferMode`	Buffering mode 1 ..... Aborting 2 ..... Buffered 3 ..... Blending low 4 ..... Blending high 5 ..... Blending previous 6 ..... Blending next	`Long (I32)`
`TransitionMode`	Transition mode in blending mode 1 ..... TMNone 2 ..... TMStartVelocity 3 ..... TMConstantVelocity 4 ..... TMCornerDistance 5 ..... TMMaxCornerDeviation 11 .... Smooth	`Long (I32)`
`TransitionParameter`	Parametr for transition (depends on transition mode)	`Double (F64)`

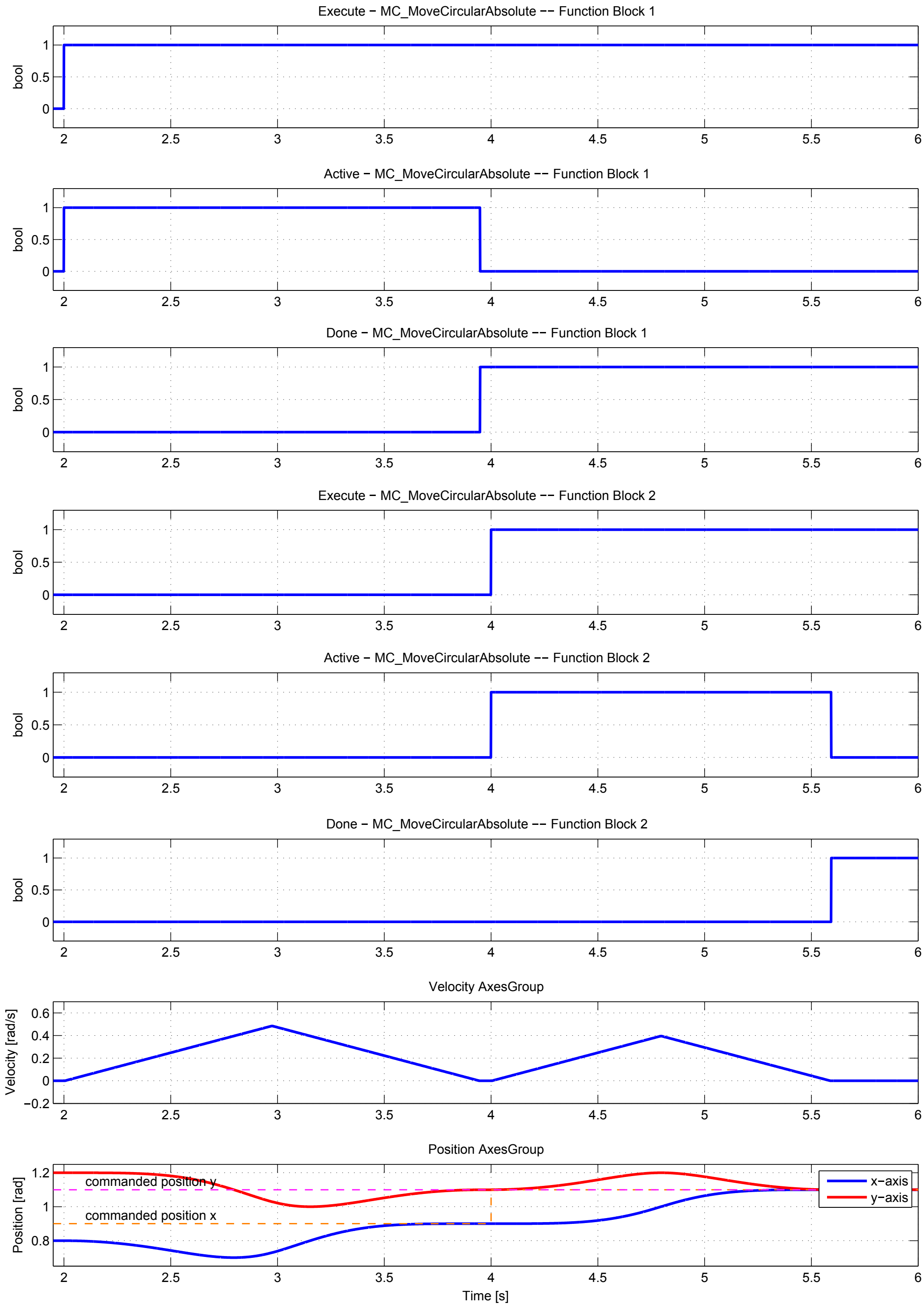
Outputs

`yAxesGroup`	Axes group reference	`Reference`
`Done`	Algorithm finished	`Bool`
`CommandAborted`	Algorithm was aborted	`Bool`
`Busy`	Algorithm not finished yet	`Bool`
`Active`	The block is controlling the axis	`Bool`
`Error`	Error occurred	`Bool`
`ErrorID`	Result of the last operation i ..... REXYGEN general error	`Error`



MC_MoveCircularAbsolute - Example

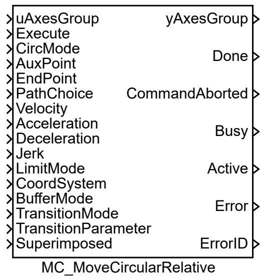



Execute – MC_MoveCircularAbsolute –– Function Block 1
Active – MC_MoveCircularAbsolute –– Function Block 1
Done – MC_MoveCircularAbsolute –– Function Block 1
Execute – MC_MoveCircularAbsolute –– Function Block 2
Active – MC_MoveCircularAbsolute –– Function Block 2
Done – MC_MoveCircularAbsolute –– Function Block 2
bool
Velocity AxesGroup
Velocity [rad/s]
Position AxesGroup
Position [rad]
commanded position y
commanded position x
x–axis
y–axis
Time [s]

MC_MoveCircularRelative – Circular move to position (relative to execution point)

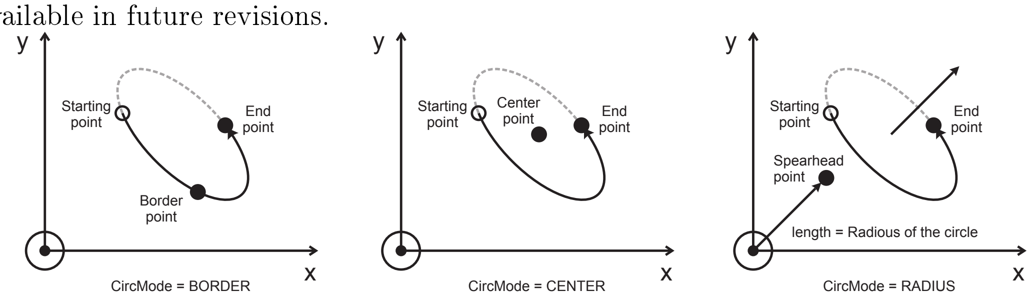
Block Symbol

Licence: COORDINATED MOTION



Function Description

The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.



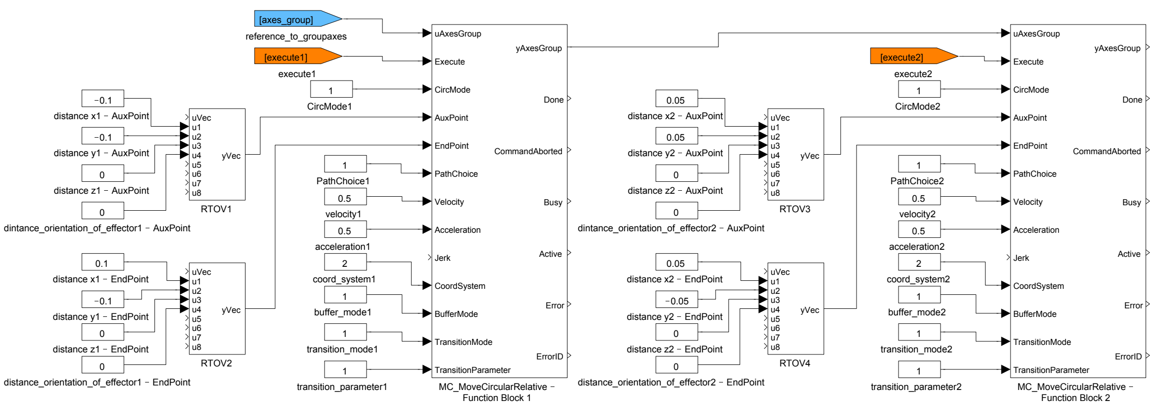
Inputs

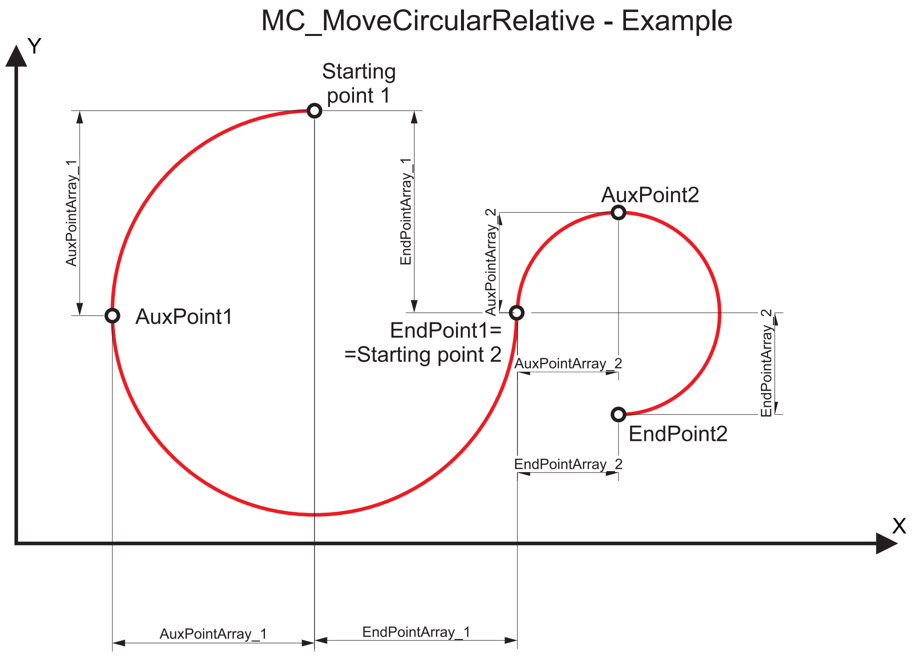
`uAxesGroup`	Axes group reference	`Reference`
`Execute`	The block is activated on rising edge	`Bool`
`CircMode`	Specifies the meaning of the input signals AuxPoint and CircDirection 1 ..... BORDER 2 ..... CENTER 3 ..... RADIUS	`Long (I32)`
`AuxPoint`	Next coordinates to define circle (depend on CircMode)	`Reference`
`EndPoint`	Target axes coordinates position	`Reference`
`PathChoice`	Choice of path 1 ..... Clockwise 2 ..... CounterClockwise	`Long (I32)`
`Velocity`	Maximal allowed velocity [unit/s]	`Double (F64)`
`Acceleration`	Maximal allowed acceleration [unit/s^2]	`Double (F64)`

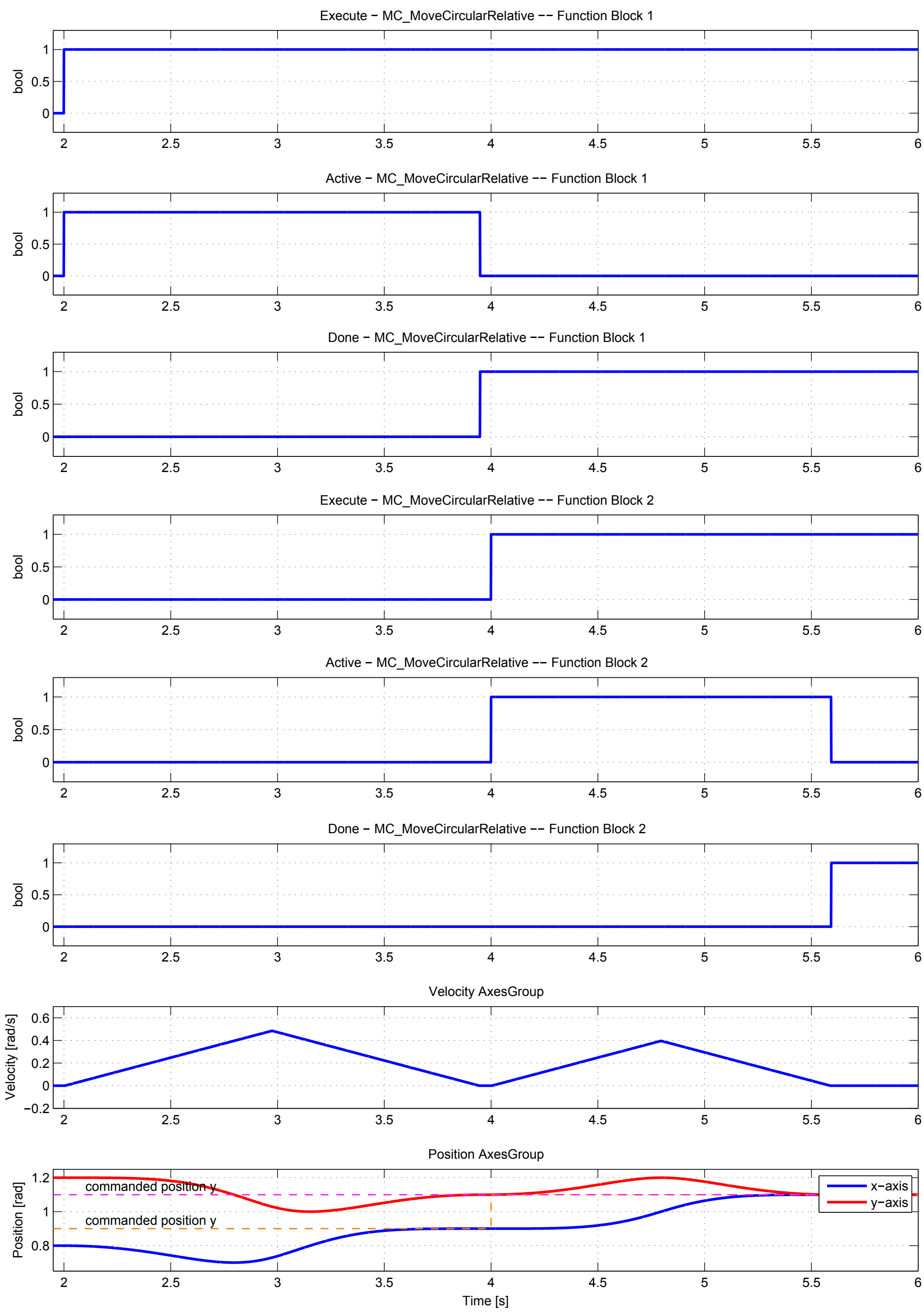
`Jerk`	Maximal allowed jerk [unit/s^3]	`Double (F64)`
`CoordSystem`	Reference to the coordinate system used 1 ..... ACS 2 ..... MCS 3 ..... PCS	`Long (I32)`
`BufferMode`	Buffering mode 1 ..... Aborting 2 ..... Buffered 3 ..... Blending low 4 ..... Blending high 5 ..... Blending previous 6 ..... Blending next	`Long (I32)`
`TransitionMode`	Transition mode in blending mode 1 ..... TMNone 2 ..... TMStartVelocity 3 ..... TMConstantVelocity 4 ..... TMCornerDistance 5 ..... TMMaxCornerDeviation 11 .... Smooth	`Long (I32)`
`TransitionParameter`	Parametr for transition (depends on transition mode)	`Double (F64)`

Outputs

`yAxesGroup`	Axes group reference	`Reference`
`Done`	Algorithm finished	`Bool`
`CommandAborted`	Algorithm was aborted	`Bool`
`Busy`	Algorithm not finished yet	`Bool`
`Active`	The block is controlling the axis	`Bool`
`Error`	Error occurred	`Bool`
`ErrorID`	Result of the last operation i ..... REXYGEN general error	`Error`



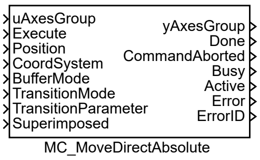

MC_MoveCircularRelative - Example
Y
X
Starting
point 1
AuxPoint1
EndPoint1=
=Starting point 2
AuxPoint2
EndPoint2
AuxPointArray_1
EndPointArray_1
AuxPointArray_2
EndPointArray_2
AuxPointArray_1
EndPointArray_1
AuxPointArray_2
EndPointArray_2


Execute – MC_MoveCircularRelative –– Function Block 1
Active – MC_MoveCircularRelative –– Function Block 1
Done – MC_MoveCircularRelative –– Function Block 1
Execute – MC_MoveCircularRelative –– Function Block 2
Active – MC_MoveCircularRelative –– Function Block 2
Done – MC_MoveCircularRelative –– Function Block 2
bool
Velocity AxesGroup
Velocity [rad/s]
Position AxesGroup
Position [rad]
commanded position x
commanded position y
x–axis
y–axis
Time [s]

MC_MoveDirectAbsolute – Direct move to position (absolute coordinates)

Block Symbol

Licence: COORDINATED MOTION



Function Description

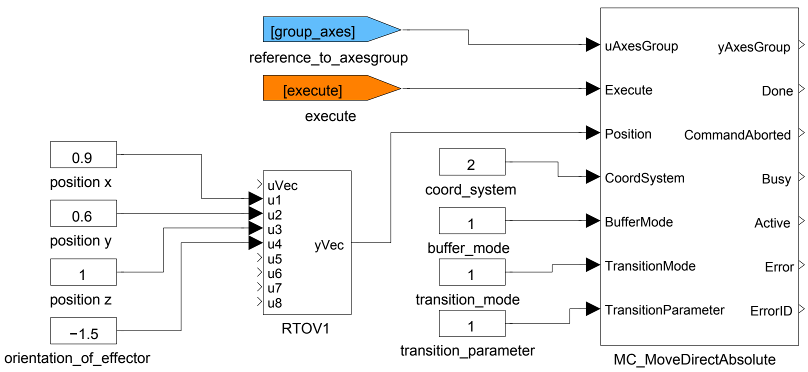
The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

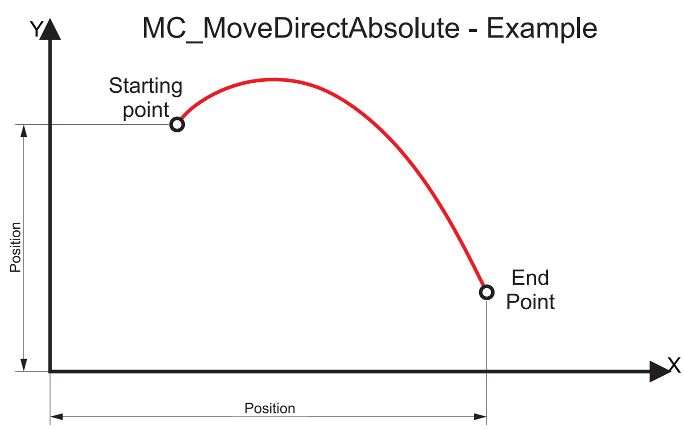
Inputs

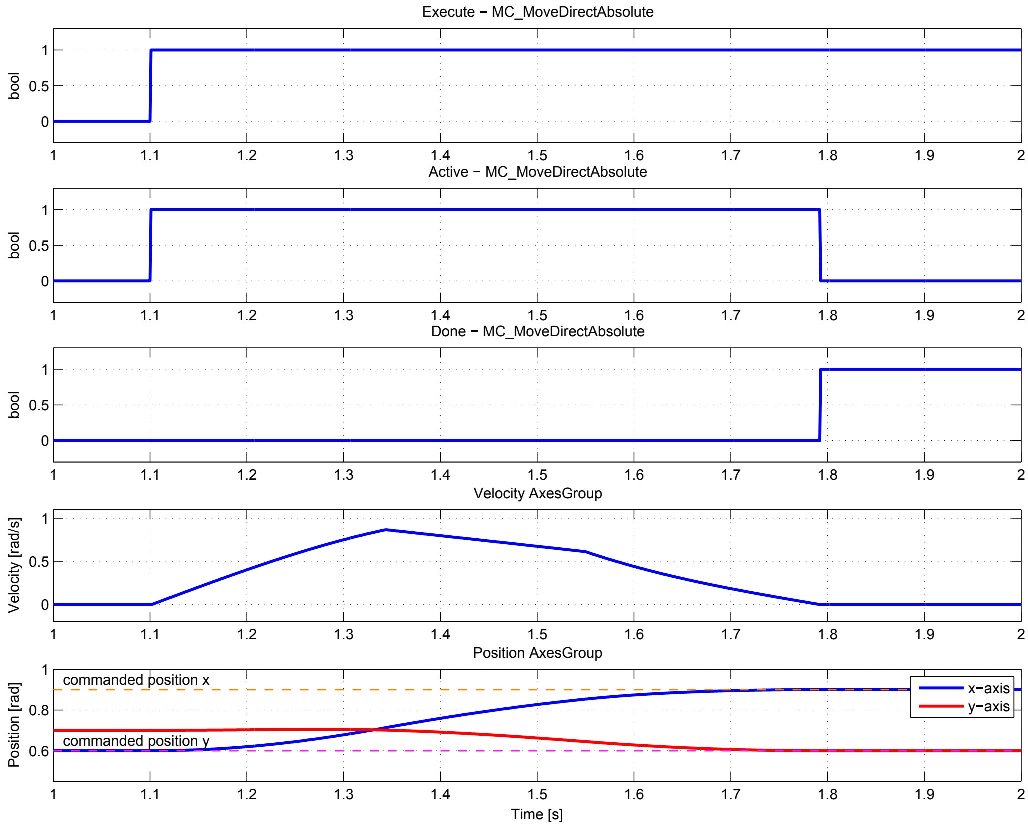
`uAxesGroup`	Axes group reference	`Reference`
`Execute`	The block is activated on rising edge	`Bool`
`Position`	Array of coordinates (positions and orientations)	`Reference`
`CoordSystem`	Reference to the coordinate system used `1` ..... ACS `2` ..... MCS `3` ..... PCS	`Long (I32)`
`BufferMode`	Buffering mode `1` ..... Aborting `2` ..... Buffered `3` ..... Blending low `4` ..... Blending high `5` ..... Blending previous `6` ..... Blending next	`Long (I32)`
`TransitionMode`	Transition mode in blending mode `1` ..... TMNone `2` ..... TMStartVelocity `3` ..... TMConstantVelocity `4` ..... TMCornerDistance `5` ..... TMMaxCornerDeviation `11` .... Smooth	`Long (I32)`
`TransitionParameter`	Parametr for transition (depends on transition mode)	`Double (F64)`

Outputs

`yAxesGroup`	Axes group reference	`Reference`
`Done`	Algorithm finished	`Bool`
`CommandAborted`	Algorithm was aborted	`Bool`
`Busy`	Algorithm not finished yet	`Bool`
`Active`	The block is controlling the axis	`Bool`
`Error`	Error occurred	`Bool`
`ErrorID`	Result of the last operation `i` ..... REXYGEN general error	`Error`





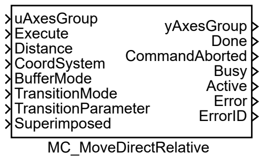


Execute − MC_MoveDirectAbsolute
bool
Active − MC_MoveDirectAbsolute
bool
Done − MC_MoveDirectAbsolute
bool
Velocity AxesGroup
Velocity [rad/s]
Position AxesGroup
Position [rad]
commanded position x
commanded position y
x−axis
y−axis
Time [s]

MC_MoveDirectRelative – Direct move to position (relative to execution point)

Block Symbol

Licence: COORDINATED MOTION



Function Description

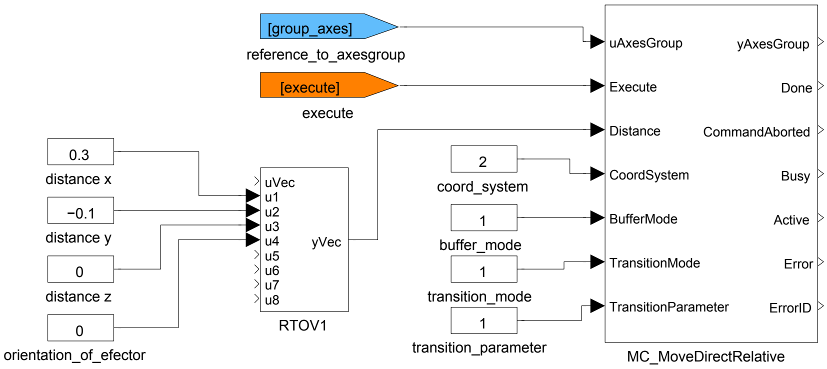
The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

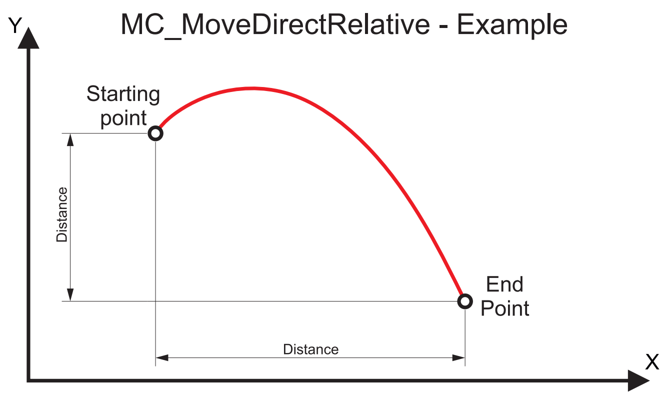
Inputs

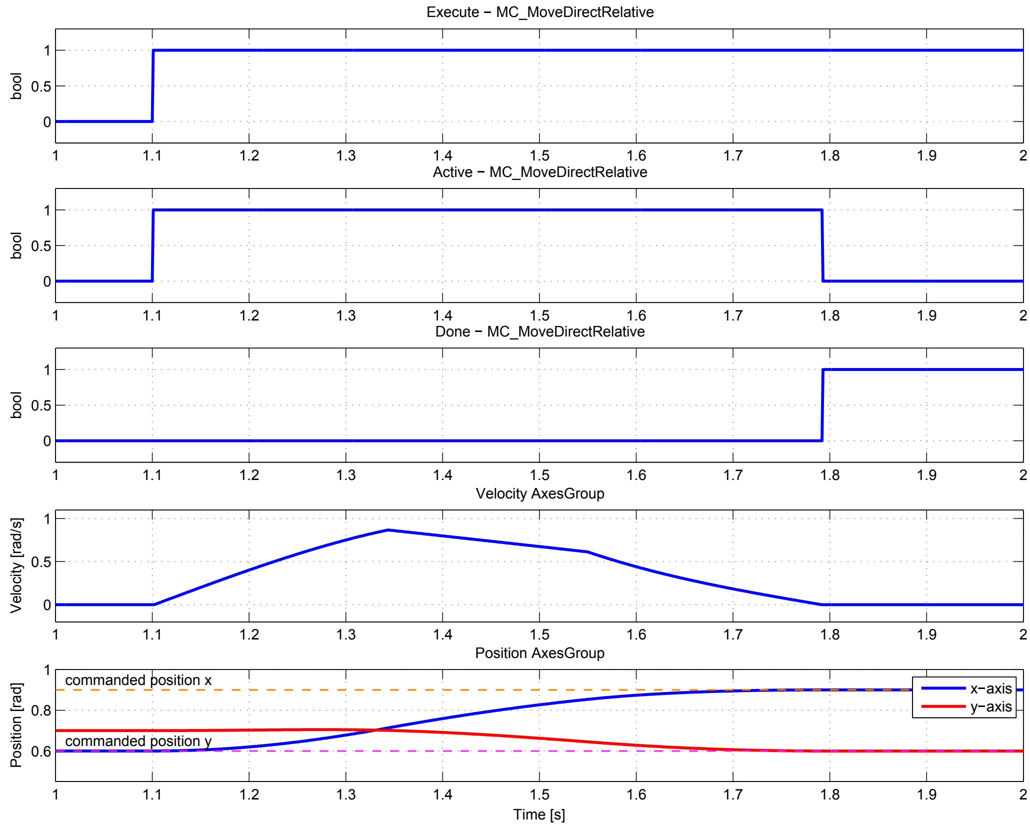
`uAxesGroup`	Axes group reference	`Reference`
`Execute`	The block is activated on rising edge	`Bool`
`Distance`	Array of coordinates (relative distances and orientations)	`Reference`
`CoordSystem`	Reference to the coordinate system used 1 ..... ACS 2 ..... MCS 3 ..... PCS	`Long (I32)`
`BufferMode`	Buffering mode 1 ..... Aborting 2 ..... Buffered 3 ..... Blending low 4 ..... Blending high 5 ..... Blending previous 6 ..... Blending next	`Long (I32)`
`TransitionMode`	Transition mode in blending mode 1 ..... TMNone 2 ..... TMStartVelocity 3 ..... TMConstantVelocity 4 ..... TMCornerDistance 5 ..... TMMaxCornerDeviation 11 .... Smooth	`Long (I32)`
`TransitionParameter`	Parametr for transition (depends on transition mode)	`Double (F64)`

Outputs

`yAxesGroup`	Axes group reference	`Reference`
`Done`	Algorithm finished	`Bool`
`CommandAborted`	Algorithm was aborted	`Bool`
`Busy`	Algorithm not finished yet	`Bool`
`Active`	The block is controlling the axis	`Bool`
`Error`	Error occurred	`Bool`
`ErrorID`	Result of the last operation `i` ..... REXYGEN general error	`Error`



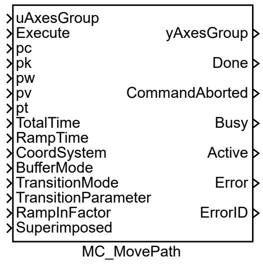



Execute − MC_MoveDirectRelative
bool
Active − MC_MoveDirectRelative
bool
Done − MC_MoveDirectRelative
bool
Velocity AxesGroup
Velocity [rad/s]
Position AxesGroup
Position [rad]
commanded position x
commanded position y
x−axis
y−axis
Time [s]

MC_MovePath – General spatial trajectory generation

Block Symbol

Licence: COORDINATED MOTION



Function Description

The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

Inputs

`uAxesGroup`	Axes group reference	`Reference`
`Execute`	The block is activated on rising edge	`Bool`
`TotalTime`	Time [s] for whole move	`Double (F64)`
`RampTime`	Time [s] for acceleration/deceleration	`Double (F64)`
`CoordSystem`	Reference to the coordinate system used 1 ..... ACS 2 ..... MCS 3 ..... PCS	`Long (I32)`
`BufferMode`	Buffering mode 1 ..... Aborting 2 ..... Buffered 3 ..... Blending low 4 ..... Blending high 5 ..... Blending previous 6 ..... Blending next	`Long (I32)`
`TransitionMode`	Transition mode in blending mode 1 ..... TMNone 2 ..... TMStartVelocity 3 ..... TMConstantVelocity 4 ..... TMCornerDistance 5 ..... TMMaxCornerDeviation 11 .... Smooth	`Long (I32)`
`TransitionParameter`	Parametr for transition (depends on transition mode)	`Double (F64)`
`RampIn`	RampIn factor (0 = RampIn mode not used)	`Double (F64)`

Parameters

`pc`	Control-points matrix ⊙`[0.0 1.0 2.0; 0.0 1.0 1.0; 0.0 1.0 0.0]`		`Double (F64)`
`pk`	Knot-points vector	⊙`[0.0 0.0 0.0 0.0 0.5 1.0 1.0]`	`Double (F64)`
`pw`	Weighting vector	⊙`[1.0 1.0 1.0]`	`Double (F64)`
`pv`	Polynoms for feedrate definition ⊙`[0.0 0.05 0.95; 0.0 0.1 0.1; 0.0 0.0 0.0; 0.1 0.0 -0.1; -0.05 0.0 0.05; 0.0 0.`		`Double (F64)`
`pt`	Knot-points (time [s]) for feedrate	⊙`[0.0 1.0 10.0 11.0]`	`Double (F64)`
`user`	Only for special edit	⊙`[0.0 1.0 2.0 3.0]`	`Double (F64)`

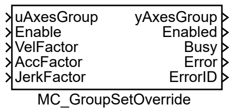
Outputs

`yAxesGroup`	Axes group reference	`Reference`
`Done`	Algorithm finished	`Bool`
`CommandAborted`	Algorithm was aborted	`Bool`
`Busy`	Algorithm not finished yet	`Bool`
`Active`	The block is controlling the axis	`Bool`
`Error`	Error occurred	`Bool`
`ErrorID`	Result of the last operation `i` ..... REXYGEN general error	`Error`

MC_GroupSetOverride – Set group override factors

Block Symbol

Licence: COORDINATED MOTION



Function Description

The function block description is not yet available. Below you can find partial description of the inputs, outputs and parameters of the block. Complete documentation will be available in future revisions.

Inputs

uAxesGroup	Axes group reference	Reference
Enable	Block function is enabled	Bool
VelFactor	Velocity multiplication factor	Double (F64)
AccFactor	Acceleration/deceleration multiplication factor	Double (F64)
JerkFactor	Jerk multiplication factor	Double (F64)

Parameter

diff	Deadband (difference for recalculation)	⊙0.05	Double (F64)

Outputs

yAxesGroup	Axes group reference	Reference
Enabled	Signal that the override faktor are set successfully	Bool
Busy	Algorithm not finished yet	Bool
Error	Error occurred	Bool
ErrorID	Result of the last operation i ..... REXYGEN general error	Error

Chapter 23

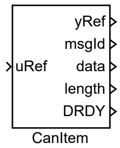
CanDrv – Communication via CAN bus

Contents

CanItem – Secondary received CAN message

Block Symbol

Licence: CANDRV



Function Description

The block is used with the CanRecv block. The **uRef** input of the **CanItem** block must be connected to the **itemRef** output of some CanRecv block or to the **yRef** output of another **CanItem** block.

This block shows the previous message that has passed the filter in the CanRecv block.

If more than one **CanItem** block is connected (directly or indirectly through the **yRef** output of the **CanItem** block already connected to the CanRecv block) then the first executed **CanItem** block shows the first message before the last received message (which is shown by the **CanRecv** block), the second executed **CanItem** block shows the second message before the last received message (which is shown by the **CanRecv** block) etc. It is strongly recommended to connect the **CanItem** blocks in a daisy chain. Unexpected ordering of messages may occur if the blocks are connected in a tree-like structure.

If no message has been received since start of the CAN driver, the data outputs have fallback values **msgId** = -1 and **length** = -1.

The **DRDY** output is set to DRDY= **on** if the message has been received during the last period, i.e. after previous execution of the **CanItem** block. At the same moment, the outputs **msgId**, **data** and **length** are updated. If there is no new data, **DRDY** output is set to DRDY= **off** and the data values are kept on the other outputs (**msgId**, **data** and **length**).

Input

uRef	Secondary received packet reference		Reference

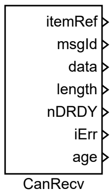
Outputs

yRef	Secondary received packet reference		Reference
msgId	CAN message ID (COB-ID)		Long (I32)
data	Message data (8 bytes maximum, LSB first)	↓-9.22337E+18 ↑9.22337E+18	Large (I64)
length	Message length (number of bytes of data)	↓0 ↑8	Long (I32)
DRDY	Received message in the last period flag		Bool

CanRecv – Receive CAN message

Block Symbol

Licence: CANDRV



Function Description

The CanRecv block receives message via CAN bus. The message is defined by the msgId, data and length inputs and the RTR and EXT parameters.

Number of messages received in the current task period (i.e. since the previous execution) is indicated by the nDRDY output.

The data from the last received message is available at the msgId, data and length outputs. Previous messages (with respect to the nmax parameter) are available using the CanItem block(s) linked to the itemRef output.

The block must be linked with the CanDrv driver. The driver must be configured to use the simple CAN mode (i.e. the parameter NodeMode = 256).

The block's name must be in the form <DRV>__<blkname> (see e.g. OUTQUAD or OUTOCT blocks for details about referencing data from I/O drivers). The <blkname> part of the name has no special meaning in this case and it is recommended to keep the original CanRecv.

The block supports short (11-bit) and long (29-bit) message IDs (see the EXT parameter) and RequestToReceive messages (see the RTR parameter). FD mode which allows up to 64 data bytes in a single message is not supported.

Outputs

itemRef	Secondary received packet reference		Reference
msgId	CAN message ID (COB-ID)		Long (I32)
data	Message data (8 bytes maximum, LSB first)	↓-9.22337E+18 ↑9.22337E+18	Large (I64)
length	Message length (number of bytes of data)	↓0 ↑8	Long (I32)
nDRDY	Number of received messages in the last period	↑255	Word (U16)
iErr	Error code		Error
age	Elapsed time since the last received message [s]	↓0.0	Double (F64)

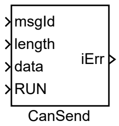
Parameters

`filterId`	MessageId of packets to receive by this block	↓0 ↑536870911	`Long (I32)`
`filterIdMask`	Mask for the `filterId` parameter (marks valid bits)	↓0 ↑536870911	`Long (I32)`
`filterLength`	Data length of packets to receive by this block (-1 allows all lengths)	↓-1 ↑8	`Long (I32)`
`RTR`	Request To Receive flag	⊙on	`Bool`
`EXT`	Extended message ID (29bits)	⊙on	`Bool`
`timeout`	Error is indicated if no packet is received within the timeout interval [s]	↓0.0	`Double (F64)`
`nmax`	Maximum number of received messages in one period	↓1 ↑255	`Long (I32)`

CanSend – Send CAN message

Block Symbol

Licence: CANDRV



Function Description

The CanSend block sends message via CAN bus. The message content is defined by the msgId, data and length inputs and the RTR and EXT parameters. Message is sent only if the input RUN is set to RUN = on.

The block must be linked with the CanDrv driver. The driver must be configured to use the simple CAN mode (i.e. the parameter NodeMode = 256).

The block's name must be in the form <DRV>__<blkname> (see e.g. OUTQUAD or OUTOCT blocks for details about referencing data from I/O drivers). The <blkname> part of the name has no special meaning in this case and it is recommended to keep the original CanSend.

The block supports short (11-bit) and long (29-bit) message IDs (see the EXT parameter) and RequestToReceive messages (see the RTR parameter). FD mode which allows up to 64 data bytes in a single message is not supported.

Inputs

msgId	CAN message ID (COB-ID)	↓0 ↑536870911	Long (I32)
length	Message length (number of bytes of data)	↓0 ↑8	Long (I32)
data	Message data (8 bytes maximum, LSB first)	↓-9.22337E+18 ↑9.22337E+18	Large (I64)
RUN	Sending message is enabled		Bool

Output

iErr	Error code		Error

Parameters

RTR	Request To Receive flag	⊙on	Bool
EXT	Extended message ID (29bits)	⊙on	Bool

Chapter 24

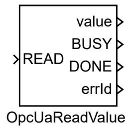
OpcUaDrv – Communication using OPC UA

Contents

OpcUaReadValue – Read value from OPC UA Server

Block Symbol

Licence: ADVANCED



Function Description

This function block depends on the OpcUa driver. Please read the OpcUaDrv manual [15] before use.

The OpcUaReadValue block reads value of an OPC UA Node through a connection established by the OPC UA client driver.

The first two parameters are NodeId and NodeId_type. The NodeId_type specifies what type of information it is expected to be entered as the NodeId parameter. If the value is **string**, **numeric**, **guid** than the NodeId parameter should contain the id of the actual OPC UA Node on the server prefixed with the index of the namespace declared in the configuration of the driver separated by a colon (e.g. 1:myNode).

If the value of the NodeId_type parameter is set to **path** than the NodeId parameter should contain the path to the desired Node in the server structure. Every segment of the path is composed from the attribute BrowserName of the node and the BrowserName is similarly with regular NodeId types prefixed with the index of the namespace declared in the configuration of the driver separated by a colon (e.g. /1:myDevice/1:myNode). The path is relative to the *Objects* folder in the OPC UA server structure.

The parameter type specifies the expected Node's value data type. The block converts the Node's value to the specified type and sets the value output signal in case of success or it sets the errId to the resulting error code.

Input

READ	Enable execution		Bool

Parameters

NodeId	OPC UA Node Id		String
NodeId_type	Type of Node ID 1 ..... string 2 ..... numeric 3 ..... guid 4 ..... path	⊙1	Long (I32)

`type`	Expected type of incoming data 1 ..... string 2 ..... double 3 ..... long 4 ..... bool	⊙1	`Long (I32)`

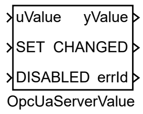
Outputs

`value`	Output signal	`Any`
`BUSY`	Busy flag	`Bool`
`DONE`	Indicator of finished transaction	`Bool`
`errId`	Error code	`Error`

OpcUaServerValue – Expose value as an OPC UA Node

Block Symbol

Licence: ADVANCED



Function Description

This function block depends on the OpcUa driver. Please read the `OpcUaDrv` manual [15] before use.

The `OpcUaServerValue` block exposes an OPC UA Node through OPC UA server driver.

The first two parameters are `NodeId` and `NodeId_type`. The `NodeId_type` specifies how the value entered as the `NodeId` parameter should be treated. The parameter `NodeId` specifies the *NodeId* that the OPC UA Node represented by the block should be exposed with.

The input `DISABLE` controls whether the OPC UA Node is exposed on the server or not. When the `SET` input is set to `on` the value on the input `uValue` port is set to the OPC UA Node's value. If the parameter `READONLY` is set to `off` the Node's value can also be changed from outside of the algorithm through the OPC UA communication protocol.

The output signal `yValue` is set to the Node's value on every tick. The parameter `type` specifies the Node's value data type, the data type of the `uValue` input and `yValue` output.

Inputs

`uValue`	Input signal	`Any`
`SET`	Set the input value to OPC UA Node value	`Bool`
`DISABLE`	Disable OPC UA Node	`Bool`

Parameters

`NodeId`	OPC UA Node Id		`String`
`NodeId_type`	OPC UA Node Id type `1` ..... string `2` ..... numeric `3` ..... guid	⊙1	`String`

`type`	Value data type 1 ..... string 2 ..... double 3 ..... long 4 ..... bool	⊙1	`Long (I32)`
`BrowseName`	OPC UA Node Browse name		`String`
`Description`	OPC UA Node description		`String`
`DisplayName`	OPC UA Node display name		`String`
`READONLY`	Set OPC Node value as read only	⊙on	`Bool`

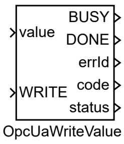
Outputs

`yValue`	Output signal	`Any`
`CHANGED`	Value of the node changed though the OPC UA protocol	`Bool`
`errId`	Error code	`Error`

`OpcUaWriteValue` – Write value to OPC UA Server

Block Symbol

Licence: ADVANCED



Function Description

This function block depends on the OpcUa driver. Please read the `OpcUaDrv` manual [15] before use.

The `OpcUaWriteValue` block writes value to the OPC UA Node through a connection established by the OPC UA client driver.

The first two parameters are `NodeId` and `NodeId_type`. The `NodeId_type` specifies what type of information it is expected to be entered as the `NodeId` parameter. If the value is `string`, `numeric`, `guid` than the `NodeId` parameter should contain the id of the actual OPC UA Node on the server prefixed with the index of the namespace declared in the configuration of the driver separated by a colon (e.g. `1:myNode`).

If the value of the `NodeId_type` parameter is set to `path` than the `NodeId` parameter should contain the path to the desired Node in the server structure. Every segment of the path is composed from the attribute `BrowserName` of the node and the `BrowserName` is similarly with regular `NodeId` types prefixed with the index of the namespace declared in the configuration of the driver separated by a colon (e.g. `/1:myDevice/1:myNode`). The path is relative to the *Objects* folder in the OPC UA server structure.

The parameter `type` specifies the expected Node's value data type. The input signal `value` is converted to the specified type and is than written to the Node's value attribute.

When the process of writing the value is finished the result code defined by OPC UA is set to the `code` output and it's textual representation is set to the `status` output.

Inputs

`value`	Input signal	`Any`
`WRITE`	Enable execution	`Bool`

Parameters

`NodeId`	OPC UA Node Id	`String`

NodeId_type	Type of Node ID 1 ..... string 2 ..... numeric 3 ..... guid 4 ..... path	⊙1	Long (I32)
type	Value data type 1 ..... string 2 ..... double 3 ..... long 4 ..... bool	⊙1	Long (I32)

Outputs

BUSY	Busy flag	Bool
DONE	Indicator of finished transaction	Bool
errId	Error code	Error
code	OPC UA result status code	DWord (U32)
status	OPC UA result status string	String

Appendix A

Licensing options

From the licensing point of view, there are several versions of the RexCore runtime module to provide maximum flexibility for individual projects. The table below compares the individual variants.

The function blocks are divided into several licensing groups. The STANDARD function blocks are always available, the other groups require activation by a corresponding licence.

	RexCore DEMO	RexCore Starter	RexCore Plus	RexCore Professional	RexCore Ultimate
Function blocks					
STANDARD	•	•	•	•	•
ADVANCED	•	–	•	•	•
REXLANG	•	–	•	•	•
MOTION CONTROL	•	–	○	○	•
COORDINATED MOTION	•	–	○	○	•
AUTOTUNING	–	–	○	○	•
MATRIX	•	–	○	○	•
I/O drivers					
Basic I/O drivers	•	•	•	•	•
Additional I/O drivers	•	○	○	•	•

(• . . . included, ○ . . . optional, – . . . not available)

See Appendix B for details about licensing of individual function blocks.

Appendix B

Licensing of individual function blocks

To maximize flexibility for individual projects, function blocks of the REXYGEN system are divided into several licensing groups. The table below shows the groups the function blocks belong to. See Appendix A for detailed information about the individual licensing options.

Function block name	Licensing group	
	STANDARD	Other
ABS_	•	
ABSROT		ADVANCED
ACD	•	
ADD	•	
ADDHEXD	•	
ADDOCT	•	
ADDQUAD	•	
AFLUSH	•	
ALB	•	
ALBI	•	
ALN	•	
ALNI	•	
ARS	•	
AND_	•	
ANDHEXD	•	
ANDOCT	•	
ANDQUAD	•	
ANLS	•	
ARC	•	
ARLY	•	

The list continues on the next page...

Function block name	Licensing group	
	STANDARD	Other
ASW		ADVANCED
ATMT	•	
AVG	•	
AVS		ADVANCED
BDHEXD	•	
BDOCT	•	
BINS	•	
BIS	•	
BISR	•	
BITOP	•	
BMHEXD	•	
BMOCT	•	
BPF	•	
CanItem		CANDRV
CanRecv		CANDRV
CanSend		CANDRV
CDELSSM		ADVANCED
CMP	•	
CNA	•	
CNB	•	
CNDR	•	
CNE	•	
CNI	•	
CNR	•	
CNS	•	
CONCAT	•	
COND		
COUNT	•	
CSSM		ADVANCED
DATE_	•	
DATETIME	•	
DDELSSM		ADVANCED
DEL	•	
DELM	•	
DER	•	
DFIR		ADVANCED
DIF_	•	
Display	•	
DIV	•	

The list continues on the next page...

Function block name	Licensing group	
	STANDARD	Other
DSSM		ADVANCED
EAS	•	
EATMT		ADVANCED
EDGE_	•	
EKF		MODEL
EMD	•	
EPC		ADVANCED
EQ	•	
EVAR	•	
EXEC	•	
FIND	•	
FLCU		ADVANCED
FNX	•	
FNXY	•	
FOPDT	•	
FRID		ADVANCED
From	•	
GAIN	•	
GETPA	•	
GETPB	•	
GETPI	•	
GETPR	•	
GETPS	•	
Goto	•	
GotoTagVisibility	•	
GRADS		ADVANCED
HMI	•	
HTTP		ADVANCED
HTTP2		ADVANCED
I3PM		ADVANCED
IADD	•	
IDIV	•	
IMOD	•	
IMUL	•	
INFO	•	
INHEXD	•	
INOCT	•	
Inport	•	
INQUAD	•	

The list continues on the next page...

Function block name	Licensing group STANDARD	Other
INSTD	•	
INTE	•	
INTSM	•	
IODRV	•	
IOTASK	•	
ISSW	•	
ISUB	•	
ITOI	•	
ITOS	•	
KDER		ADVANCED
LC	•	
LEN	•	
LIN	•	
LLC	•	
LPBRK	•	
LPF	•	
MC_AccelerationProfile		MOTION CONTROL
MC_AddAxisToGroup		COORDINATED MOTION
MC_CamIn		MOTION CONTROL
MC_CamOut		MOTION CONTROL
MC_CombineAxes		MOTION CONTROL
MC_GearIn		MOTION CONTROL
MC_GearInPos		MOTION CONTROL
MC_GearOut		MOTION CONTROL
MC_GroupContinue		COORDINATED MOTION
MC_GroupDisable		COORDINATED MOTION
MC_GroupEnable		COORDINATED MOTION
MC_GroupHalt		COORDINATED MOTION
MC_GroupInterrupt		COORDINATED MOTION
MC_GroupReadActualAcceleration		COORDINATED MOTION
MC_GroupReadActualPosition		COORDINATED MOTION
MC_GroupReadActualVelocity		COORDINATED MOTION
MC_GroupReadError		COORDINATED MOTION
MC_GroupReadStatus		COORDINATED MOTION
MC_GroupReset		COORDINATED MOTION
MC_GroupSetOverride		COORDINATED MOTION
MC_GroupSetPosition		COORDINATED MOTION
MC_GroupStop		COORDINATED MOTION
MC_Halt		MOTION CONTROL

The list continues on the next page...

Function block name	Licensing group STANDARD	Other
MC_HaltSuperimposed		MOTION CONTROL
MC_Home		MOTION CONTROL
MC_MoveAbsolute		MOTION CONTROL
MC_MoveAdditive		MOTION CONTROL
MC_MoveCircularAbsolute		COORDINATED MOTION
MC_MoveCircularRelative		COORDINATED MOTION
MC_MoveContinuousAbsolute		MOTION CONTROL
MC_MoveContinuousRelative		MOTION CONTROL
MC_MoveDirectAbsolute		COORDINATED MOTION
MC_MoveDirectRelative		COORDINATED MOTION
MC_MoveLinearAbsolute		COORDINATED MOTION
MC_MoveLinearRelative		COORDINATED MOTION
MC_MovePath		COORDINATED MOTION
MC_MovePath_PH		COORDINATED MOTION
MC_MoveRelative		MOTION CONTROL
MC_MoveSuperimposed		MOTION CONTROL
MC_MoveVelocity		MOTION CONTROL
MC_PhasingAbsolute		MOTION CONTROL
MC_PhasingRelative		MOTION CONTROL
MC_PositionProfile		MOTION CONTROL
MC_Power		MOTION CONTROL
MC_ReadActualPosition		MOTION CONTROL
MC_ReadAxisError		MOTION CONTROL
MC_ReadBoolParameter		MOTION CONTROL
MC_ReadCartesianTransform		COORDINATED MOTION
MC_ReadParameter		MOTION CONTROL
MC_ReadStatus		MOTION CONTROL
MC_Reset		MOTION CONTROL
MC_SetCartesianTransform		COORDINATED MOTION
MC_SetOverride		MOTION CONTROL
MC_Stop		MOTION CONTROL
MC_TorqueControl		MOTION CONTROL
MC_UngroupAllAxes		COORDINATED MOTION
MC_VelocityProfile		MOTION CONTROL
MC_WriteBoolParameter		MOTION CONTROL
MC_WriteParameter		MOTION CONTROL
MCP_AccelerationProfile		MOTION CONTROL
MCP_CamIn		MOTION CONTROL
MCP_CamTableSelect		MOTION CONTROL

The list continues on the next page...

Function block name	Licensing group	
	STANDARD	Other
MCP_CombineAxes		MOTION CONTROL
MCP_GearIn		MOTION CONTROL
MCP_GearInPos		MOTION CONTROL
MCP_GroupHalt		COORDINATED MOTION
MCP_GroupInterrupt		COORDINATED MOTION
MCP_GroupSetOverride		COORDINATED MOTION
MCP_GroupSetPosition		COORDINATED MOTION
MCP_GroupStop		COORDINATED MOTION
MCP_Halt		MOTION CONTROL
MCP_HaltSuperimposed		MOTION CONTROL
MCP_Home		MOTION CONTROL
MCP_MoveAbsolute		MOTION CONTROL
MCP_MoveAdditive		MOTION CONTROL
MCP_MoveCircularAbsolute		COORDINATED MOTION
MCP_MoveCircularRelative		COORDINATED MOTION
MCP_MoveContinuousAbsolute		MOTION CONTROL
MCP_MoveContinuousRelative		MOTION CONTROL
MCP_MoveDirectAbsolute		COORDINATED MOTION
MCP_MoveDirectRelative		COORDINATED MOTION
MCP_MoveLinearAbsolute		COORDINATED MOTION
MCP_MoveLinearRelative		COORDINATED MOTION
MCP_MovePath		COORDINATED MOTION
MCP_MovePath_PH		COORDINATED MOTION
MCP_MoveRelative		MOTION CONTROL
MCP_MoveSuperimposed		MOTION CONTROL
MCP_MoveVelocity		MOTION CONTROL
MCP_PhasingAbsolute		MOTION CONTROL
MCP_PhasingRelative		MOTION CONTROL
MCP_PositionProfile		MOTION CONTROL
MCP_SetCartesianTransform		COORDINATED MOTION
MCP_SetKinTransform_Arm		COORDINATED MOTION
MCP_SetKinTransform_Schunk		COORDINATED MOTION
MCP_SetKinTransform_UR		COORDINATED MOTION
MCP_SetOverride		MOTION CONTROL
MCP_Stop		MOTION CONTROL
MCP_TorqueControl		MOTION CONTROL
MCP_VelocityProfile		MOTION CONTROL
MCU	•	
MDL	•	

The list continues on the next page...

Function block name	Licensing group	
	STANDARD	Other
MDLI	•	
MID	•	
MINMAX	•	
MODULE	•	
MP	•	
MqttPublish		MQTTDRV
MqttSubscribe		MQTTDRV
MUL	•	
MVD	•	
NOT_	•	
NSCL	•	
NSSM		MODEL
OpcUaReadValue		ADVANCED
OpcUaServerValue		ADVANCED
OpcUaWriteValue		ADVANCED
OR_	•	
ORHEXD	•	
OROCT	•	
ORQUAD	•	
OSD	•	
OSCALL	•	
OUTHEXD	•	
OUTOCT	•	
Outport	•	
OUTQUAD	•	
OUTRHEXD		ADVANCED
OUTROCT		ADVANCED
OUTRQUAD		ADVANCED
OUTRSTD		ADVANCED
OUTSTD	•	
PARA	•	
PARB	•	
PARE	•	
PARI	•	
PARR	•	
PARS	•	
PGAVR		
PGBAT		
PGBUS		

The list continues on the next page...

Function block name	Licensing group	
	STANDARD	Other
PGCB		
PGENG		
PGGEN		
PGGS		
PGINV		
PGLOAD		
PGMAINS		
PGSENS		
PGSG		
PGSIM		
PGSOLAR		
PGWIND		
PIDAT		AUTOTUNING
PIDE		ADVANCED
PIDGS		ADVANCED
PIDMA		AUTOTUNING
PIDU	•	
PIDUI		ADVANCED
PJROCT	•	
PJSEXOCT	•	
PJSEXOCT	•	
PJSOCT	•	
POL	•	
POUT	•	
PRBS	•	
PRGM	•	
PROJECT	•	
PSMPC		ADVANCED
PWM	•	
PYTHON		REXLANG
QFC		ADVANCED
QFD		ADVANCED
QTASK	•	
QP_MPC2QP		ADVANCED
QP_UPDATE		ADVANCED
QP_OASES		ADVANCED
QCEDPOPT		ADVANCED
RDC		ADVANCED
REC	•	

The list continues on the next page...

Function block name	Licensing group STANDARD	Other
REGEXP		ADVANCED
REL	•	
REPLACE	•	
REXLANG		REXLANG
RLIM	•	
RLY	•	
RM_AxesGroup		COORDINATED MOTION
RM_Axis		MOTION CONTROL
RM_AxisOut		MOTION CONTROL
RM_AxisSpline		MOTION CONTROL
RM_DirectTorque		MOTION CONTROL
RM_DirectVelocity		MOTION CONTROL
RM_DriveMode		MOTION CONTROL
RM_Feed		COORDINATED MOTION
RM_Gcode		COORDINATED MOTION
RM_GroupTrack		COORDINATED MOTION
RM_HomeOffset		MOTION CONTROL
RM_Track		MOTION CONTROL
RS	•	
RTOI	•	
RTOS	•	
RTOV	•	
S_AND		
S_BC		
S_CMP		
S_CTS		
S_LB		
S_NOT		
S_OR		
S_PULS		
S_PV		
S_RS		
S_SEL		
S_SELVAL		
S_SR		
S_SUMC		
S_TDE		
S_TDR		
S_TLATCH		

The list continues on the next page...

Function block name	Licensing group	
	STANDARD	Other
S_VALB		
S_VALC		
S1OF2		ADVANCED
SAI		ADVANCED
SAT	•	
SC2FA		AUTOTUNING
SCU	•	
SCUV	•	
SEL	•	
SELHEXD	•	
SELOCT	•	
SELQUAD	•	
SELSOCT	•	
SELU	•	
SETPA	•	
SETPB	•	
SETPI	•	
SETPR	•	
SETPS	•	
SG	•	
SGI	•	
SGSLP		ADVANCED
SHIFTOCT	•	
SHLD	•	
SILO	•	
SILOS	•	
SINT	•	
SLEEP	•	
SMHCC		ADVANCED
SMHCCA		AUTOTUNING
SMTP		ADVANCED
SOPDT	•	
SPIKE		ADVANCED
SQR	•	
SQRT_	•	
SR	•	
SRTF		ADVANCED
SSW	•	
STEAM	•	

The list continues on the next page...

Function block name	Licensing group	
	STANDARD	Other
STOR	•	
SUB	•	
SubSystem	•	
SWR	•	
SWU	•	
SWVMR	•	
TASK	•	
TIME	•	
TIMER_	•	
TIODRV	•	
TRND	•	
TRNDV	•	
TSE	•	
UTOI	•	
VDEL	•	
VIN		ADVANCED
VOUT		ADVANCED
VTOR	•	
WASM		REXLANG
WSCH	•	
WWW	•	
ZV4IS		ADVANCED
DFIR		ADVANCED
PGSIM		
PGMAINS		
PGBUS		
PGLOAD		
PGGEN		
PGCB		
PGSENS		
PGENG		
PGAVR		
PGSG		
PGINV		
PGSOLAR		
PGWIND		
PGBAT		
PGGS		
CanSend		CANDRV

The list continues on the next page...

Function block name	Licensing group	
	STANDARD	Other
CanRecv		CANDRV
CanItem		CANDRV
MqttPublish		MQTTDRV
MqttSubscribe		MQTTDRV
EKF		MODEL
NSSM		MODEL
RM_HomeOffset		MOTION CONTROL
PARE	•	
EQ	•	
PYTHON		REXLANG
WASM		REXLANG
RM_DriveMode		MOTION CONTROL
RM_DirectTorque		MOTION CONTROL
RM_DirectVelocity		MOTION CONTROL
COND		
TESTS		
S_CMPT		
S_RCK		
S_POR		
OpcUaReadValue		
OpcUaWriteValue		
OpcUaServerValue		
STEAM	•	
PJSEXOCT	•	
BISR	•	
DP2M		
MBAL		
MOFN		
TB1		
TB2		
TB3		
TB6		
VAC		
OSD	•	
CNT	•	
CNDT	•	
CONCAT_DT	•	
SPLIT_DT	•	
STR2DT	•	

The list continues on the next page...

Function block name	Licensing group	
	STANDARD	Other
DT2STR	•	
WEEK	•	
T2STR	•	
TZ2UTC	•	
UTC2TZ	•	
SYSLOG	•	
SYSEVENT	•	
ALM	•	
ALARMS	•	
TRIM	•	

Appendix C

Error codes of the REXYGEN system

Success codes

`0` .... Success
`-1` ...... False
`-2` ...... First value is greater
`-3` ...... Second value is greater
`-4` ...... Parameter changed
`-5` ...... Success, no server transaction done
`-6` ...... Value too big
`-7` ...... Value too small
`-8` ...... Operation in progress
`-9` ...... REXYGEN I/O driver warning
`-10` ..... No more archive items
`-11` ..... Object is array
`-12` ..... Closed
`-13` ..... End of file
`-14` ..... Parameter may be incorrect

General failure codes

`-100` .... Not enough memory
`-101` .... Assertion failure
`-102` .... Timeout
`-103` .... General input variable error
`-104` .... Invalid configuration version
`-105` .... Not implemented
`-106` .... Invalid parameter
`-107` .... COM/OLE error
`-108` .... REXYGEN Module error - some driver or block is not installed or licensed

-109 .... REXYGEN I/O driver error
-110 .... Task creation error
-111 .... Operating system call error
-112 .... Invalid operating system version
-113 .... Access denied by operating system
-114 .... Block period has not been set
-115 .... Initialization failed
-116 .... REXYGEN configuration is being changed
-117 .... Invalid target device
-118 .... Access denied by REXYGEN security mechanism
-119 .... Block or object is not installed or licensed
-120 .... Checksum mismatch
-121 .... Object already exists
-122 .... Object doesn't exist
-123 .... System user doesn't belong to any REXYGEN group
-124 .... Password mismatch
-125 .... Bad user name or password
-126 .... Target device is not compatible
-127 .... Resource is locked by another module and can not be used
-128 .... String is not valid in UTF8 codepage
-129 .... Start of executive not allowed
-130 .... Some resource count reached limit
-131 .... Text value has been truncated
-132 .... Unsufficient buffer for requested operation
-133 .... Block execution halted due to runtime error

Class registration, symbol and validation error codes

-200 .... Class not registered
-201 .... Class already registered
-202 .... Not enough space for registry
-203 .... Registry index out of range
-204 .... Invalid context
-205 .... Invalid identifier
-206 .... Invalid input flag
-207 .... Invalid input mask
-208 .... Invalid object type
-209 .... Invalid variable type
-210 .... Invalid object workspace
-211 .... Symbol not found
-212 .... Symbol is ambiguous
-213 .... Range check error
-214 .... Not enough search space
-215 .... Write to read-only variable denied
-216 .... Data not ready

-217 .... Value out of range
-218 .... Input connection error
-219 .... Loop of type UNKNOWN detected
-220 .... REXLANG compilation error

Stream and file system codes

-300 .... Stream overflow
-301 .... Stream underflow
-302 .... Stream send error
-303 .... Stream receive error
-304 .... Stream download error
-305 .... Stream upload error
-306 .... File creation error
-307 .... File open error
-308 .... File close error
-309 .... File read error
-310 .... File write error
-311 .... Invalid format
-312 .... Unable to compress files
-313 .... Unable to extract files

Communication errors

-400 .... Network communication failure
-401 .... Communication already initialized
-402 .... Communication finished successfully
-403 .... Communicaton closed unexpectedly
-404 .... Unknown command
-405 .... Unexpected command
-406 .... Communicaton closed unexpectedly, probably 'Too many clients'
-407 .... Communication timeout
-408 .... Target device not found
-409 .... Link failed
-410 .... REXYGEN configuration has been changed
-411 .... REXYGEN executive is being terminated
-412 .... REXYGEN executive was terminated
-413 .... Connection refused
-414 .... Target device is unreachable
-415 .... Unable to resolve target in DNS
-416 .... Error reading from socket
-417 .... Error writing to socket
-418 .... Invalid operation on socket
-419 .... Reserved for socket 1
-420 .... Reserved for socket 2
-421 .... Reserved for socket 3

-422 .... Reserved for socket 4
-423 .... Reserved for socket 5
-424 .... Unable to create SSL context
-425 .... Unable to load certificate
-426 .... SSL handshake error
-427 .... Certificate verification error
-428 .... Reserved for SSL 2
-429 .... Reserved for SSL 3
-430 .... Reserved for SSL 4
-431 .... Reserved for SSL 5
-432 .... Relay rejected
-433 .... STARTTLS rejected
-434 .... Authentication method rejected
-435 .... Authentication failed
-436 .... Send operation failed
-437 .... Receive operation failed
-438 .... Communication command failed
-439 .... Receiving buffer too small
-440 .... Sending buffer too small
-441 .... Invalid header
-442 .... HTTP server responded with error
-443 .... HTTP server responded with redirect
-444 .... Operation would blok
-445 .... Invalid operation
-446 .... Communication closed
-447 .... Connection cancelled

Numerical error codes

-500 .... General numeric error
-501 .... Division by zero
-502 .... Numeric stack overflow
-503 .... Invalid numeric instruction
-504 .... Invalid numeric address
-505 .... Invalid numeric type
-506 .... Not initialized numeric value
-507 .... Numeric argument overflow/underflow
-508 .... Numeric range check error
-509 .... Invalid subvector/submatrix range
-510 .... Numeric value too close to zero

Archive system codes

-600 .... Archive seek underflow
-601 .... Archive semaphore fatal error
-602 .... Archive cleared

- `-603` .... Archive reconstructed from saved vars
- `-604` .... Archive reconstructed from normal vars
- `-605` .... Archive check summ error
- `-606` .... Archive integrity error
- `-607` .... Archive sizes changed
- `-608` .... Maximum size of disk archive file exceeded

Motion control codes

- `-700` .... MC - Invalid parameter
- `-701` .... MC - Out of range
- `-702` .... MC - Position not reachable
- `-703` .... MC - Invalid axis state
- `-704` .... MC - Torque limit exceeded
- `-705` .... MC - Time limit exceeded
- `-706` .... MC - Distance limit exceeded
- `-707` .... MC - Step change in position or velocity
- `-708` .... MC - Base axis error or invalid state
- `-709` .... MC - Stopped by drive FAULT
- `-710` .... MC - Stopped by POSITION limit
- `-711` .... MC - Stopped by VELOCITY limit
- `-712` .... MC - Stopped by ACCELERATION limit
- `-713` .... MC - Stopped by LIMITSWITCH
- `-714` .... MC - Stopped by position LAG
- `-715` .... MC - Axis disabled during motion
- `-716` .... MC - Transition failed
- `-717` .... MC - Servodrive failed or disabled
- `-718` .... MC - Not used
- `-719` .... MC - Not used
- `-720` .... MC - General failure
- `-721` .... MC - Not implemented
- `-722` .... MC - Command is aborted
- `-723` .... MC - Conflict in block and axis periods
- `-724` .... MC - Busy, waiting for activation

Licensing codes

- `-800` .... Unable to identify Ethernet interface
- `-801` .... Unable to identify CPU
- `-802` .... Unable to identify HDD
- `-803` .... Invalid device code
- `-804` .... Invalid licensing key
- `-805` .... Not licensed

Webserver-related errors

-900 .... Web request too large
-901 .... Web reply too large
-902 .... Invalid format
-903 .... Invalid parameter

RexVision-related errors

-1000 ... Result is not evaluated
-1001 ... The searched object/pattern can not be found
-1002 ... The search criterion returned more corresponding objects

FMI standard related errors

-1100 ... FMI Context allocation failure
-1101 ... Invalid FMU version
-1102 ... FMI XML parsing error
-1103 ... FMI Model Exchange kind required
-1104 ... FMI Co-Simulation kind required
-1105 ... Could not create FMU loading mechanism
-1106 ... Instantiation of FMU failed
-1107 ... Termination of FMU failed
-1108 ... FMU reset failed
-1109 ... FMU Experiment setup failed
-1110 ... Entering FMU initialization mode failed
-1111 ... Exiting FMU initialization mode failed
-1112 ... Error getting FMU variable list
-1113 ... Error getting FMU real variable
-1114 ... Error setting FMU real variable
-1115 ... Error getting FMU integer variable
-1116 ... Error setting FMU integer variable
-1117 ... Error getting FMU boolean variable
-1118 ... Error setting FMU boolean variable
-1119 ... Doing a FMU simulation step failed
-1120 ... FMU has too many inputs
-1121 ... FMU has too many ouputs
-1122 ... FMU has too many parameters

Bibliography

[1] OPC Foundation. *Data Access Custom Interface Specification Version 3.00*. OPC Foundation, P.O. Box 140524, Austin, Texas, USA, 2003.

[2] REX Controls s.r.o.. *REXYGEN Studio – User manual*, 2020. →.

[3] Schlegel Miloš. Fuzzy controller: a tutorial. *www.rexcontrols.com*.

[4] Miloš Schlegel, Pavel Balda, and Milan Štětina. Robust PID autotuner: method of moments. *Automatizace*, 46(4):242–246, 2003.

[5] M. Schlegel and P. Balda. Diskretizace spojitého lineárního systému (in Czech). *Automatizace*, 11, 1987.

[6] BLAS 3.8.0. – netlib.org. Basic Linear Algebra Subprograms Version 3.8.0, 2017.

[7] LAPACK 3.8.0 – netlib.org. Linear Algebra PACKage Version 3.8.0, 2018.

[8] Cleve Moler and Charles Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Review*, 45(1):3–49, 2003.

[9] J.M. Maciejowski. *Predictive Control with Constraints.* Prentice Hall, England., 2002.

[10] H.J. Ferreau, C. Kirches, A. Potschka, H.G. Bock, and M. Diehl. qpOASES: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6(4):327–363, 2014.

[11] H.J. Ferreau. *qpOASES User's Manual, Version 3.2 (April 2017)*, 2017.

[12] Python Software Foundation. Python documentation, 2019.

[13] REX Controls s.r.o.. *MQTTDrv driver of REXYGEN – user guide*, 2020. →.

[14] OASIS. MQTT Version 3.1.1, 2014.

[15] REX Controls s.r.o.. *OpcUaDrv driver of REXYGEN – user guide*, 2020. →.

Index

Documentation reference number: 15831