Simulink® PLC Coder™

Reference

MATLAB&SIMULINK®

MathWorks®

How to Contact MathWorks

Latest news: www.mathworks.com

Sales and services: www.mathworks.com/sales_and_services

User community: www.mathworks.com/matlabcentral

Technical support: www.mathworks.com/support/contact_us

Phone: 508-647-7000

The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

Revision History

March 2019	Online only	New for Version 3.0 (Release 2019a)
September 2019	Online only	Revised for Version 3.1 (Release 2019b)
March 2020	Online only	Revised for Version 3.2 (Release 2020a)
September 2020	Online only	Revised for Version 3.3 (Release 2020b)
March 2021	Online only	Revised for Version 3.4 (Release 2021a)

Contents

iii

Blocks

ADD

Add inputs
Library:
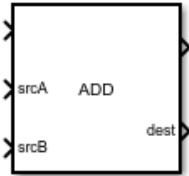
Description

The ADD block implements the ADD ladder logic instruction. When the rung conditions are true, the block adds source A (srcA) to source B (srcB) and outputs the result to the destination (dest).

Ports

Input

EnableIn — Enable Input
off (default) | on

Controls the execution of the block. **EnableIn** reflects the rung state preceding the block. If the rung state preceding the block is false, **EnableIn** is false, the block does not execute and the outputs are not updated.

srcA — First input signal
scalar

The first input signal to the addition operation.

Data Types: int8 | int16 | int32 | single

srcB — Second input signal
scalar

The second input signal to the addition operation.

Data Types: int8 | int16 | int32 | single

Output

EnableOut — Enable Output
off (default) | on

By default, **EnableOut** follows the state of **EnableIn**. If the **EnableIn** input to the block is false, the logic implemented by the block is not executed and **EnableOut** signal is set to false.

dest — Output signal
scalar

Output signal resulting from the addition operation.

Data Types: `int8` | `int16` | `int32` | `single`

See Also
CPT | DIV | MUL | SUB

Introduced in R2019a

AFI

Always False
Library:

Description

The AFI block implements the `AFI` ladder logic instruction. This block sets its **EnableOut** signal to false. Use this block to temporarily disable a rung while you are debugging.

Ports

Input

EnableIn — Enable Input
off (default) | on

Controls the execution of the block. **EnableIn** reflects the rung state preceding the block. If the rung state preceding the block is false, **EnableIn** is false, the block does not execute and the outputs are not updated.

Output

EnableOut — Enable Output
off (default) | on

When the block executes, it automatically sets the **EnableOut** to false. This disables the subsequent blocks on the rung.

See Also
AFI | JMP | LBL | MCR | NOP | TND

Introduced in R2019a

AND

Bitwise AND
Library:

Description

The AND block implements the AND ladder logic instruction. When the rung conditions are true, the block performs bitwise AND operation on the values at source A with the values at source B. The result of this operation is available at the destination port (`dest`).

Ports

Input

EnableIn — Enable Input
off (default) | on

Controls the execution of the block. **EnableIn** reflects the rung state preceding the block. If the rung state preceding the block is false, **EnableIn** is false, the block does not execute and the outputs are not updated.

srcA — First input signal
scalar

The first input signal to the bitwise AND operation. If the datatype is `single` (REAL – ladder logic equivalent), the input value is converted to `int32` (DINT – ladder logic equivalent). `int8, int16` (SINT,INT – ladder logic equivalent) datatypes are converted to `int32` (DINT – ladder logic equivalent) by filling the upper bits with 0s.

Data Types: `int8` | `int16` | `int32` | `single`

srcB — Second input signal
scalar

The second input signal to the bitwise AND operation. If the datatype is `single` (REAL – ladder logic equivalent), the input value is converted to `int32` (DINT – ladder logic equivalent). `int8, int16` (SINT,INT – ladder logic equivalent) datatypes are converted to `int32` (DINT – ladder logic equivalent) by filling the upper bits with 0s.

Data Types: `int8` | `int16` | `int32` | `single`

Output

EnableOut — Enable Output
off (default) | on

By default, **EnableOut** follows the state of **EnableIn**. If the **EnableIn** input to the block is false, the logic implemented by the block is not executed and **EnableOut** signal is set to false.

`dest` — Output signal
scalar

Output signal resulting from the bitwise AND operation. If the datatype is `single` (REAL – ladder logic equivalent), the resultant `int32` (DINT – ladder logic equivalent) is converted to REAL (`single` – ladder logic equivalent).

Data Types: `int8` | `int16` | `int32` | `single`

See Also
NOT | OR

Introduced in R2019a

AOI Runner

AOI Runner
Library:

Description

AOI Runner is a top organisational unit. It consists of AOI block. The AOI runner acts as an encapsulation around the ladder diagram function block.

See Also

Ladder Diagram Program | Ladder Diagram Subroutine | Ladder Diagram Function Block (AOI) | PLC Controller | Task

Introduced in R2019a

CLR

Clear
Library:
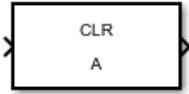
Description

The CLR block implements the `CLR` ladder logic instruction. When the rung conditions are true, the block clears all the bits specified in the `Data To Clear` tag.

Ports

Input

EnableIn — Enable Input
off (default) | on

Controls the execution of the block. **EnableIn** reflects the rung state preceding the block. If the rung state preceding the block is false, **EnableIn** is false, the block does not execute and the outputs are not updated.

Output

EnableOut — Enable Output
off (default) | on

By default, **EnableOut** follows the state of **EnableIn**. If the **EnableIn** input to the block is false, the logic implemented by the block is not executed and **EnableOut** signal is set to false.

Parameters

Data To Clear — Operand data
A (default) | boolean

Specify the data bits to be cleared. Data bits are specified in the format of `tags`. In Ladder Diagrams, tags (variables) are used for representing all inputs, outputs, and internal memory with attributes such as `Data Type`, `Initial Value`, and `size`. To change the attributes of the `Data To Clear`, open the **Program Variables** table within the Ladder Diagram Program block.

Programmatic Use
Block Parameter: PLCOperandTag
Type: character vector
Value: character vector
Default: `'A'`

See Also
MOV

Introduced in R2019a

ControllerTags

Syntax

Description

The **Controller Tags** table is used to specify the global variable and I/O symbol attributes.

For example, the controller tag table can be used to specify attributes for global variables (tags) as shown.

| Block Parameters: Controller Tags						✕
Global Variable and Symbol Spreadsheet						
Name	Mapping Type	Port	Data Type	Size	Initial Value	Delete
Start	Input Symbol ▾	1 ▾	boolean ⌄	>>	1	true
Stop	Input Symbol ▾	2 ▾	boolean ⌄	>>	1	false
Motor	Output Symbol ▾	1 ▾	boolean ⌄	>>	1	false
				OK	Cancel	Help	Apply

See Also

AOI Runner | Ladder Diagram Program | Ladder Diagram Subroutine | Ladder Diagram Function Block (AOI) | Task

Introduced in R2019a

COP

Copy File
Library:
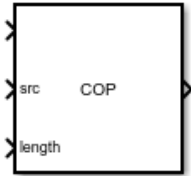
Description

The COP block implements the COP instruction. When the rung conditions are true, the block is used to copy the data of the source and store it at the destination, keeping the source unchanged.

Limitations

- Source and destination elements support only numerical data types.
- Source initial element index and destination element initial index start from zero.

Ports

Input

EnableIn — Enable Input
off (default) | on

Controls the execution of the block. **EnableIn** reflects the rung state preceding the block. If the rung state preceding the block is false, **EnableIn** is false, the block does not execute and the outputs are not updated.

src — Source
scalar | vector | matrix

It gives the value stored at source. This data is copied into destination.

Data Types: int8 | int16 | int32 | int64 | single | String | Structure

length — Length of elements
scalar

It gives the number or source elements to copy.

Data Types: int8 | int16 | int32

Output

EnableOut — Enable Output
off (default) | on

By default, **EnableOut** follows the state of **EnableIn**. If the **EnableIn** input to the block is false, the logic implemented by the block is not executed and **EnableOut** signal is set to false.

Parameters

`Source Initial Element Index` — Source Start Index
1 (default) | scalar

Source Start Index is used only when the source is an array. It specifies the index of the first element to be copied into destination. If the source is not an array, then the value of `Source Inital Element Index` is 1.

Programmatic Use
Block Parameter: PLCSRCArrayIndex
Type: scalar
Value: scalar
Default: 1

Data Types: `int8` | `int16` | `int32` | `single`

`Destination Array Name` — Destination
A (default) | boolean

The data copied from source is stored in destination. The destination is specified in the format of `tags`. In Ladder Diagrams, tags (variables) are used for representing all inputs, outputs, and internal memory with attributes such as `Data Type`, `Initial Value`, and `size`. To change the attributes of the `Destination Array Name`, open the **Program Variables** table within the Ladder Diagram Program block.

Programmatic Use
Block Parameter: PLCOperandTag
Type: character vector
Value: character vector
Default: `'A'`

`Destination Initial Element Index` — Destination Start Index
scalar

Destination Start Index is used only when the source is an array. It specifies the start index of destination where the data is copied. If the source and destination data is not stored in an array, then the value of `Destination Initial Element Index` is 1.

Programmatic Use
Block Parameter: PLCDestArrayIndex
Type: scalar
Value: scalar
Default: 0

Data Types: `int8` | `int16` | `int32` | `single`

See Also
FLL

Introduced in R2019a

CPT

Evaluate expression
Library:
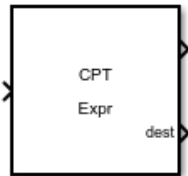
Description

The CPT block implements the CPT ladder logic instruction. When the rung conditions are true, the block evaluates the expression **Expr** and outputs the result to the destination (`dest`).

Limitations

* The CPT ladder diagram instruction does not support direct operand calls. Currently MOD, AND, XOR, FTD, and TOD instructions are unsupported as CPT operands.

Ports

Input

EnableIn — Enable Input
off (default) | on

Controls the execution of the block. **EnableIn** reflects the rung state preceding the block. If the rung state preceding the block is false, **EnableIn** is false, the block does not execute and the outputs are not updated.

Output

EnableOut — Enable Output
off (default) | on

By default, **EnableOut** follows the state of **EnableIn**. If the **EnableIn** input to the block is false, the logic implemented by the block is not executed and **EnableOut** signal is set to false.

dest — Output Signal
scalar

The result obtained after computing the expression is placed at the destination.

Data Types: `int8` | `int16` | `int32` | `single`

Parameters

Expression to Evaluate — Expression to evaluate
'Expr' (default) | character vector

Specify the expression to be evaluated. An expression consisting of `tags` and/or immediate values separated by operators. In Ladder Diagrams, tags (variables) are used for representing all inputs, outputs, and internal memory with attributes such as `Data Type`, `Initial Value`, and `size`. To change the attributes of the `Expression to Evaluate`, open the **Program Variables** table within the Ladder Diagram Program block.

Programmatic Use
Block Parameter: PLCOperandTag
Type: character vector
Value: character vector
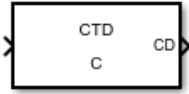**Default**: 'Expr'

See Also
ADD | DIV | MUL | SUB

Introduced in R2019a

CTD

Count Down
Library:

Description

The CTD block implements the CTD ladder logic instruction. When the rung conditions are true, the block counts downwards.

Ports

Input

EnableIn — Enable Input
off (default) | on

Controls the execution of the block. **EnableIn** reflects the rung state preceding the block. If the rung state preceding the block is false, **EnableIn** is false, the block does not execute and the outputs are not updated.

Output

CD — Count Down Enable Output
off (default) | on

The count down enable output contains the rung-in condition when the instruction was last executed. By default, **CD** follows the state of **EnableIn**. If the **EnableIn** input to the block is false, the logic implemented by the block is not executed and **CD** signal is set to false.

Parameters

Counter Tag — Counter Structure
C (default) | character vector

Specify the counter parameters in the format of `tags`. In Ladder Diagrams, tags (variables) are used for representing all inputs, outputs, and internal memory with attributes such as `Data Type`, `Initial Value`, and `size`. To change the attributes of the `Operand Tag`, open the **Program Variables** table within the Ladder Diagram Program block. The `Data Type` of the timer tag is of the `Bus:COUNTER` type with its initial value specified as a structure containing the following fields:

Field	Description	Default Value
PRE	The preset value specifies the value which the accumulated value must reach before the instruction indicates it is done	0
ACC	The accumulated value specifies the number of transitions the instruction has counted.	0
CU	The count up enable contains `rung-condition-in` when the instruction was last executed.	1
CD	Count down enabled.	1
DN	The done bit when set indicates the counting operation is complete	0
OV	The overflow bit when set indicates the counter incremented past the upper limit of 2,147,483,647	0
UN	The underflow when set indicates the counter decremented past the lower limit of -2,147,483,648.	0

Programmatic Use
Block Parameter: PLCOperandTag
Type: character vector
Value: character vector
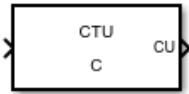**Default**: `'T'`

See Also

CTD | RES | RTO | TOF | TON

Introduced in R2019a

CTU

Count Up
Library:

Description

The CTU block implements the CTU ladder logic instruction. When the rung conditions are true, the block counts upwards.

Ports

Input

EnableIn — Enable Input
off (default) | on

Controls the execution of the block. **EnableIn** reflects the rung state preceding the block. If the rung state preceding the block is false, **EnableIn** is false, the block does not execute and the outputs are not updated.

Output

CU — Count Up Enable Output
off (default) | on

The count up enable output contains the rung-in condition when the instruction was last executed. By default, **CU** follows the state of **EnableIn**. If the **EnableIn** input to the block is false, the logic implemented by the block is not executed and **CU** signal is set to false.

Parameters

Counter Tag — Counter Structure
C (default) | character vector

Specify the counter parameters in the format of tags. In Ladder Diagrams, tags (variables) are used for representing all inputs, outputs, and internal memory with attributes such as Data Type, Initial Value, and size. To change the attributes of the Operand Tag, open the **Program Variables** table within the Ladder Diagram Program block. The Data Type of the timer tag is of the Bus:COUNTER type with its initial value specified as a structure containing the following fields:

Field	Description	Default Value
PRE	The preset value specifies the value which the accumulated value must reach before the instruction indicates it is done	0
ACC	The accumulated value specifies the number of transitions the instruction has counted.	0
CU	The count up enable contains `rung-condition-in` when the instruction was last executed.	1
CD	Count down enabled.	1
DN	The done bit when set indicates the counting operation is complete	0
OV	The overflow bit when set indicates the counter incremented past the upper limit of 2,147,483,647	0
UN	The underflow when set indicates the counter decremented past the lower limit of -2,147,483,648.	0

Programmatic Use
Block Parameter: PLCOperandTag
Type: character vector
Value: character vector
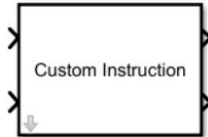**Default**: `'T'`

See Also
CTU | RES | RTO | TOF | TON

Introduced in R2019a

Custom Instruction

Create custom ladder logic instruction
Library:

Description

The Custom Instruction block implements user-defined instructions for a ladder diagram model. When the rung conditions are true, the block executes the specified custom logic. You can save these instructions in a user-defined library named `plcuserlib.slx`. You can also import, simulate, and export your ladder logic instructions by using your custom blocks saved in `plcuserlib.slx` library.

Ports

Input

`EnableIn` — Enable Input
off (default) | on

The **EnableIn** port controls the execution of the block and also reflects the rung state preceding the block. If the rung state preceding the block is false, the **EnableIn** signal is set to false, the block does not execute the custom logic, and the outputs are not updated.

`src` — Input signal
scalar

Input signal to the Custom Instruction block.

Data Types: `int8` | `int16` | `int32` | `single`

Output

`EnableOut` — Enable Output
off (default) | on

If the **EnableIn** input to the block is false, the custom logic implemented by the block is not executed and **EnableOut** signal is set to false. If **EnableIn** is true and the custom instruction executes, **EnableOut** signal is set to true.

Parameters

Inputs and Outputs

`Instruction Name` — Name of user-defined ladder logic instruction
sampleBlock (default) | character vector

Name of the ladder logic instruction that you want to create. The Rockwell Automation® Studio 5000 IDE must support the ladder logic instruction name.

`Number of Inputs` — Number of input signals to block
1 (default) | scalar

The number of input signals to the block that are required for your custom ladder logic instruction.

Programmatic Use
Block Parameter: `NumInputs`
Type: scalar
Value: scalar
Default: 1

Data Types: `int8` | `int16` | `int32` | `single`

`Input Types` — Data type of input signal
`{{'SINT', 'INT','DINT','REAL'}}` (default) | character vector

The data type of the input signal specified as a cell array. If there is more than one input signal, specify the data type as a comma-separated list of cell arrays for each signal. For example, if you have two input signals with the same data type, then specify the **Input Types** as `{{'SINT','INT','DINT','REAL'},{'SINT','INT','DINT','REAL'}}`.

Programmatic Use
Block Parameter: `InputTypeList`
Type: cell array
Value: cell array
Default: `{{'SINT','INT','DINT','REAL'}}`

Data Types: `character vector`

`Number of Outputs` — Number of output signals from block
1 (default) | scalar

The number of output signals from the user-defined Custom Instruction block.

Programmatic Use
Block Parameter: `NumOutputs`
Type: scalar
Value: scalar
Default: 1

Data Types: `int8` | `int16` | `int32` | `single`

`Output Types` — Data type of output signal
`{{'SINT','INT','DINT','REAL'}}` (default) | character vector

The data type of the output signal specified as a cell array. If there is more than one output signal, specify the data type as a comma-separated list of cell arrays for each signal. For example, if you have two output signals with the same data type, then specify the **Output Types** as `{{'SINT','INT','DINT','REAL'},{'SINT','INT','DINT','REAL'}}`.

Programmatic Use
Block Parameter: `OutputTypeList`
Type: cell array
Value: cell array

Default: {{'SINT','INT','DINT','REAL'}}

Data Types: character vector

See Also

plcimportladder | plcladderinstructions

Topics
"Create Custom Instruction in PLC Ladder Diagram Models"
"Import L5X Ladder Files into Simulink"
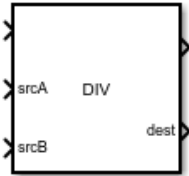"Modeling and Simulation of Ladder Diagrams in Simulink"
"Generating Ladder Diagram Code from Simulink"

Introduced in R2020a

DIV

Divide one input by another
Library:

Description

The DIV block implements the `DIV` ladder logic instruction. When the rung conditions are true, the block divides the dividend at source A (`srcA`) by the divisor at source B (`srcB`) and outputs the result to the destination (`dest`).

Note If you perform a divide by zero operation a `"Divide by zero detected"` diagnostic message is generated during code generation.

Ports

Input

EnableIn — Enable Input
off (default) | on

Controls the execution of the block. **EnableIn** reflects the rung state preceding the block. If the rung state preceding the block is false, **EnableIn** is false, the block does not execute and the outputs are not updated.

srcA — First input signal
scalar

The first input signal to the division operation.

Data Types: `int8` | `int16` | `int32` | `single`

srcB — Second input signal
scalar

The second input signal to the division operation.

Data Types: `int8` | `int16` | `int32` | `single`

Output

EnableOut — Enable Output
off (default) | on

By default, **EnableOut** follows the state of **EnableIn**. If the **EnableIn** input to the block is false, the logic implemented by the block is not executed and **EnableOut** signal is set to false.

`dest` — Output signal
scalar

Output signal resulting from the division operation.

Data Types: `int8` | `int16` | `int32` | `single`

See Also
ADD | CPT | MUL | SUB

Introduced in R2019a

EQU

Equal To
Library:

Description

The EQU block implements the EQU instruction. When the rung conditions are true,the block verifies whether the value at source A is equal to the value at source B.

Ports

Input

EnableIn — Enable Input
off (default) | on

Controls the execution of the block. **EnableIn** reflects the rung state preceding the block. If the rung state preceding the block is false, **EnableIn** is false, the block does not execute and the outputs are not updated.

srcA — First input signal
scalar

Value to test against srcB.

Data Types: int8 | int16 | int32 | single

srcB — Second input signal
scalar

Value to test against srcA.

Data Types: int8 | int16 | int32 | single

Output

EnableOut — Enable Output
off (default) | on

If the **EnableIn** input to the block is false, the logic implemented by the block is not executed and **EnableOut** signal is set to false. If **EnableIn** is true, **EnableOut** signal is set to true, if srcA is equal to srcb.

See Also

GEQ | GRT | LEQ | LES | NEQ

Introduced in R2019a

FBC

File Bit Comparison
Library:
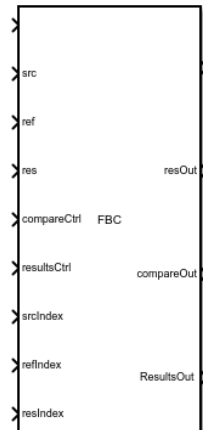
Description

The FBC block implements the FBC instruction. The block compares the source and reference data bit by bit and stores the outcome in result.

Ports

Input

EnableIn — Enable Input
off (default) | on

This acts as the enabler for the block. The FBC block executes only when EnableIn is true.

src — Source
integer

It gives the value stored at source. This value is compared with reference. Since the block performs a bit comparison, the source is an array variable.

Data Types: `int32`

ref — Reference
integer

It gives the value stored at reference. This value is compared with source. Since the block performs a bit comparison, the reference is an array variable.

Data Types: `int32`

res — Result
integer

The result of bit comparison between source and reference is stored at Result. This is an array variable.

Data Types: `int32`

compareCtrl — Compare Control
structure

It is a structure consisting of length and position variables. The length variable stores the number of bits to compare. The position variable stores the current position in the source. The initial value of position variable is 0.

resultsCtrl — Results Control
structure

It is a structure consisting of length and position variables. The length variable stores the number of storage locations in the result. The position variable stores the current position in the result. The initial value of position variable is 0.

srcIndex — Source Index
scalar

Source Start Index specifies the index of the starting element in source for comparison. Typically the value of srcIndex is 0.

Data Types: `int8` | `int16` | `int32` | `single`

refIndex — Reference Index
scalar

Reference Start Index specifies the index of the starting element in reference for comparison. Typically the value of refIndex is 0.

Data Types: `int8` | `int16` | `int32` | `single`

resIndex — Result Index
scalar

Result Start Index specifies the index of the starting element in result for storing the result of comparison. Typically the value of resIndex is 0.

Data Types: `int8` | `int16` | `int32` | `single`

Output

EnableOut — Enable Output
off (default) | on

When set, EnableOut provides the result of the block at the output. Once EnableIn is set, it automatically sets the EnableOut.

resOut — Result Output
integer

The result of comparison stored in res is available at the output port. The data can be read by connecting a variable write block at resOut.

Data Types: `int32`

`compareOut` — Compare Output

structure

The comparison operation modifies the values of position and length variable of compareCtrl. The compareCtrl structure is available at the compareOut port. The data can be read by connecting a variable write block to it.

`ResultsOut` — Results Output

structure

The comparison operation modifies the values of position and length variable of resultsCtrl. The resultsCtrl structure is available at the resultsOut port. The data can be read by connecting a variable write block to it.

Parameters

`Sample time (-1 for inherited)` — Discrete interval between sample time hits

-1 (default) | scalar

Enter the discrete interval between sample time hits or specify -1 to inherit the sample time

See also "Specify Sample Time".

Programmatic Use
Block Parameter: `SampleTime`
Type: character vector
Value: real scalar
Default: `'-1'`

See Also

Introduced in R2019a

FLL

File Fill
Library:

Description

The FLL block implements the FLL instruction. When the rung conditions are true, the block fills a block of memory with the provided source value. The Source remains unchanged.

Limitations

- Source elements should be scalar numeric data types.
- Destination elements should be scalar numeric data types.
- Source and destination elements should be of the same data type.
- Source initial element index and destination element initial index start from zero.

Ports

Input

EnableIn — Enable Input
off (default) | on

Controls the execution of the block. **EnableIn** reflects the rung state preceding the block. If the rung state preceding the block is false, **EnableIn** is false, the block does not execute and the outputs are not updated.

src — Source
scalar | vector | matrix

It gives the value stored at source. This data is copied into destination.

Data Types: int8 | int16 | int32 | int64 | single | String | Structure

length — Length of elements
scalar

Number of destination elements to fill.

Data Types: int8 | int16 | int32

Output

EnableOut — Enable Output
off (default) | on

By default, **EnableOut** follows the state of **EnableIn**. If the **EnableIn** input to the block is false, the logic implemented by the block is not executed and **EnableOut** signal is set to false.

Parameters

Destination Array Name — Destination
A (default) | boolean

The data copied from source is filled to the destination. The destination is specified in the format of `tags`. In Ladder Diagrams, tags (variables) are used for representing all inputs, outputs, and internal memory with attributes such as `Data Type`, `Initial Value`, and `size`. To change the attributes of the `Destination Array Name`, open the **Program Variables** table within the Ladder Diagram Program block.

Programmatic Use
Block Parameter: PLCOperandTag
Type: character vector
Value: character vector
Default: 'A'

Destination Initial Element Index — Destination Start Index
scalar

Destination Start Index is used only when the source is an array. It specifies the start index of destination where the data is copied. If the source and destination data is not stored in an array, then the value of `Destination Initial Element Index` is 1.

Programmatic Use
Block Parameter: PLCDestArrayIndex
Type: scalar
Value: scalar
Default: 0

Data Types: `int8` | `int16` | `int32` | `single`
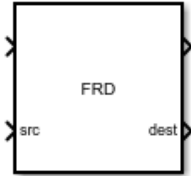
See Also
COP

Introduced in R2019a

FRD

Convert to integer
Library:

Description

The FRD block implements the FRD ladder logic instruction. When the rung conditions are true, the block converts the BCD value at source A (srcA) to a decimal value and outputs the result to the destination (dest).

Note The FRD block is not supported by Simulink® Design Verifier™.

Ports

Input

EnableIn — Enable Input
off (default) | on

Controls the execution of the block. **EnableIn** reflects the rung state preceding the block. If the rung state preceding the block is false, **EnableIn** is false, the block does not execute and the outputs are not updated.

srcA — Input signal
scalar

The input value to the conversion operation.

Data Types: int8 | int16 | int32

Output

EnableOut — Enable Output
off (default) | on

By default, **EnableOut** follows the state of **EnableIn**. If the **EnableIn** input to the block is false, the logic implemented by the block is not executed and **EnableOut** signal is set to false.

dest — Output signal
scalar

The decimal equivalent value of input is present at the destination.

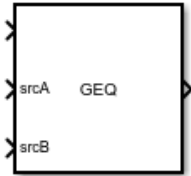Data Types: int8 | int16 | int32

See Also
ADD | CPT | DIV | MUL | SUB

Introduced in R2019a

GEQ

Greater Than or Equal To
Library:

Description

The GEQ block implements the GEQ instruction. When the rung conditions are true,the block verifies whether the value at source A is greater than or equal to the value at source B.

Ports

Input

EnableIn — Enable Input
off (default) | on

Controls the execution of the block. **EnableIn** reflects the rung state preceding the block. If the rung state preceding the block is false, **EnableIn** is false, the block does not execute and the outputs are not updated.

srcA — First input signal
scalar

Value to test against srcB.

Data Types: int8 | int16 | int32 | single

srcB — Second input signal
scalar

Value to test against srcA.

Data Types: int8 | int16 | int32 | single

Output

EnableOut — Enable Output
off (default) | on

If the **EnableIn** input to the block is false, the logic implemented by the block is not executed and **EnableOut** signal is set to false. If **EnableIn** is true, **EnableOut** signal is set to true, if srcA is greater than or equal to srcb.

See Also
EQU | GRT | LEQ | LES | NEQ

Introduced in R2019a

GRT

Greater than
Library:

Description

The GRT block implements the GRT instruction. When the rung conditions are true,the block verifies whether the value at source A is greater than the value at source B.

Ports

Input

EnableIn — Enable Input
off (default) | on

Controls the execution of the block. **EnableIn** reflects the rung state preceding the block. If the rung state preceding the block is false, **EnableIn** is false, the block does not execute and the outputs are not updated.

srcA — First input signal
scalar

Value to test against `srcB`.

Data Types: `int8` | `int16` | `int32` | `single`

srcB — Second input signal
scalar

Value to test against `srcA`.

Data Types: `int8` | `int16` | `int32` | `single`

Output

EnableOut — Enable Output
off (default) | on

If the **EnableIn** input to the block is false, the logic implemented by the block is not executed and **EnableOut** signal is set to false. If **EnableIn** is true, **EnableOut** signal is set to true, if `srcA` is greater than `srcb`.

See Also

EQU | GEQ | LEQ | LES | NEQ

Introduced in R2019a

JMP

Jump
Library:

Description

The JMP block implements the `JMP` ladder logic instruction. When the rung conditions are true, the block skips a part of ladder logic and the rung execution moves to the block referenced by the label block. Jump block can only skip the rungs after it. The block does not support a backward jump.

Ports

Input

EnableIn — Enable Input
off (default) | on

Controls the execution of the block. **EnableIn** reflects the rung state preceding the block. If the rung state preceding the block is false, **EnableIn** is false, the block does not execute and the outputs are not updated.

Output

EN — Enable Output
off (default) | on

By default, **EnableOut** follows the state of **EnableIn**. If the **EnableIn** input to the block is false, the logic implemented by the block is not executed and **EnableOut** signal is set to false.

Parameters

Jump to Label — Name of the label
L (default) | character vector

Specify the name for associated `LBL` instruction.

Programmatic Use
Block Parameter: `PLCLabelTag`
Type: character vector
Value: character vector
Default: `'L'`

See Also

AFI | LBL | MCR | NOP | TND

Introduced in R2019a

Junction

Junction
Library:

Description

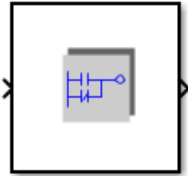The Junction block is used to connect one or more ladder logic branches.

See Also

Power Rail Terminal | Power Rail Start | Rung 1

Introduced in R2019a

Function Block (AOI)

Ladder Diagram Function Block (AOI)
Library:

Description

The Ladder Diagram function block is a standalone, organizational unit. Use this block in the absence of a general hierarchy of controller, task, program, and ladder diagram. It is also called an AOI block. The Ladder Diagram function block consists of a ladder diagram and function block variables. The function block variables represent the function variables used in the ladder diagram and provide information about its data types, size, and initial value.

Limitations

- Input and output variable data types must be `SINT`, `DINT`, `INT`, `REAL`, or `BOOL`.
- Input and output variable size must be one.
- You must not use global variables in ladder diagrams inside the `Logic Routine`.

Ports

Input

EnableIn — Enable Input
off (default) | on

The **EnableIn** port controls the execution of the block and reflects the rung state preceding the block. If the rung state preceding the block is false, the **EnableIn** signal is set to false, the block does not execute the custom logic, and the outputs are not updated.

Output

EnableOut — Enable Output
off (default) | on

If the **EnableIn** input to the block is false, the custom logic implemented by the block is not executed and the **EnableOut** signal is set to false. If **EnableIn** is true and the custom instruction executes, the **EnableOut** signal is set to true.

Parameters

Inputs and Outputs

`Function Block Name` — Name of AOI
FB (default) | character vector

Name of the AOI function block that you want to create. The Rockwell Automation Studio 5000 IDE must support the ladder logic instruction name.

Programmatic Use
Block Parameter: `PLCPOUName`
Type: character vector
Value: character vector
Default: `'FB'`

`Block Data Tag` — Name of the AOI instance
D (default) | character vector

Name of the AOI instance that you want to generate code for or the instance name for the imported AOI instruction. The Rockwell Automation Studio 5000 IDE must support the ladder logic instruction name.

Programmatic Use
Block Parameter: `PLCOperandTag`
Type: character vector
Value: character vector
Default: `'D'`

`Variable Table` — Open AOI variable table
Boolean

Button that opens the AOI variable table.

Programmatic Use
Block Parameter: `PLCEditVariableSS`
Type: button

`Logic Routine` — Open AOI ladder logic
Boolean

Button that opens the ladder logic routine to implement the AOI ladder logic.

Programmatic Use
Block Parameter: `PLCEditPOU`
Type: button

`Allow EnableInFalse Routine` — Enable `EnableInFalse` routine ladder logic
Boolean

Check box that you select to enable the ladder logic inside the `EnableInFalse` routine.

Programmatic Use
Block Parameter: `PLCAllowEnableInFalse`
Type: check-box

EnableInFalse Routine — Open EnableInFalse routine ladder logic
Boolean

Button that opens the ladder logic routine to implement the AOI `EnableInFalse` ladder logic.

Programmatic Use
Block Parameter: PLCEditEnableInFalse
Type: button

Allow Prescan Routine — Enable Prescan routine ladder logic
Boolean

Check box that you select to enable the ladder logic inside the `Prescan` routine.

Programmatic Use
Block Parameter: PLCAllowPrescan
Type: check-box

Prescan Routine — Open Prescan routine ladder logic
Boolean

Button that opens the ladder logic routine to implement the AOI `Prescan` ladder logic.

Programmatic Use
Block Parameter: PLCEditPrescan
Type: button

See Also

AOI Runner | Ladder Diagram Program | Ladder Diagram Subroutine | Ladder Diagram Function Block (AOI) | PLC Controller | Task

Introduced in R2019a

Program

Ladder Diagram Program
Library:
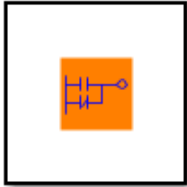
Description

The Ladder Diagram program consists of the Ladder Diagram representation of the logic and program variables. The program executes Ladder Diagram rungs from top to bottom and from left to right. The program variables consists of local and external data types along with its data types, initial values and size. To understand the link between program structure in the Rockwell IDE and Program structure using Simulink PLC Coder refer to

Tips

• When naming Programs choose different names for the Programs even if they are in different Tasks. Rockwell does not allow Programs to have the same names even if they are in different Tasks. For example, if you have an Input Task and Programs named Program and Program 1 these Program names will be unable for use in other Tasks in the same controller.
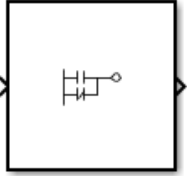
See Also

AOI Runner | Ladder Diagram Program | Ladder Diagram Subroutine | Ladder Diagram Function Block (AOI) | PLC Controller | Task

Introduced in R2019a

Subroutine

Ladder Diagram Subroutine
Library:

Description

Ladder Diagram Subroutine allows you to create and define a named ladder logic routine. You can edit the logic implemented by the subroutine by clicking on the `Routine Logic` button found under the block parameters menu of this block.

Ports

Input

EnableIn — Enable Input
off (default) | on

Controls the execution of the block. **EnableIn** reflects the rung state preceding the block. If the rung state preceding the block is false, **EnableIn** is false, the block does not execute and the outputs are not updated.

Output

EnableOut — Enable Output
off (default) | on

By default, **EnableOut** follows the state of **EnableIn**. If the **EnableIn** input to the block is false, the logic implemented by the block is not executed and **EnableOut** signal is set to false.

Parameters

Routine Name — Name of the routine
R (default) | character vector

Specify the name of the ladder logic subroutine.

Programmatic Use
Block Parameter: PLCPOUName
Type: character vector
Value: character vector
Default: 'R'

Routine Logic — Open ladder logic
boolean

Button that opens the ladder logic subroutine.

Programmatic Use
Block Parameter: PLCOpenRoutine
Type: button

See Also
AOI Runner | Ladder Diagram Program | Ladder Diagram Subroutine | Ladder Diagram Function Block (AOI) | PLC Controller | Task

Introduced in R2019a

LBL

Label
Library:

Description

The LBL block implements the LBL ladder logic instruction. This block is used along with JMP block. The rung execution jumps to the block referenced by LBL block.

Ports

Input

EnableIn — Enable Input
off (default) | on

Controls the execution of the block. **EnableIn** reflects the rung state preceding the block. If the rung state preceding the block is false, **EnableIn** is false, the block does not execute and the outputs are not updated.

Output

EN — Enable Output
off (default) | on

By default, **EnableOut** follows the state of **EnableIn**. If the **EnableIn** input to the block is false, the logic implemented by the block is not executed and **EnableOut** signal is set to false.

Parameters

Label — Name of the label
L (default) | character vector

Specify the name for the LBL.

Programmatic Use
Block Parameter: PLCLabelTag
Type: character vector
Value: character vector
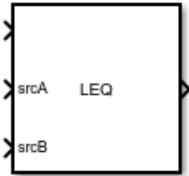**Default**: 'L'

See Also
AFI | JMP | MCR | NOP | TND

Introduced in R2019a

LEQ

Less Than or Equal To
Library:

Description

The LEQ block implements the LEQ instruction. When the rung conditions are true,the block verifies whether the value at source A is less than or equal to the value at source B.

Ports

Input

EnableIn — Enable Input
off (default) | on

Controls the execution of the block. **EnableIn** reflects the rung state preceding the block. If the rung state preceding the block is false, **EnableIn** is false, the block does not execute and the outputs are not updated.

srcA — First input signal
scalar

Value to test against srcB.

Data Types: int8 | int16 | int32 | single

srcB — Second input signal
scalar

Value to test against srcA.

Data Types: int8 | int16 | int32 | single

Output

EnableOut — Enable Output
off (default) | on

If the **EnableIn** input to the block is false, the logic implemented by the block is not executed and **EnableOut** signal is set to false. If **EnableIn** is true, **EnableOut** signal is set to true, if srcA is less than or equal to srcb.

See Also
EQU | GEQ | GRT | LES | NEQ

Introduced in R2019a

LES

Less Than
Library:
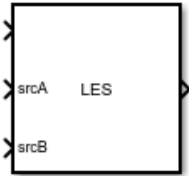
Description

The LES block implements the LES instruction. When the rung conditions are true,the block verifies whether the value at source A is less than the value at source B.

Ports

Input

EnableIn — Enable Input
off (default) | on

Controls the execution of the block. **EnableIn** reflects the rung state preceding the block. If the rung state preceding the block is false, **EnableIn** is false, the block does not execute and the outputs are not updated.

srcA — First input signal
scalar

Value to test against srcB.

Data Types: int8 | int16 | int32 | single

srcB — Second input signal
scalar

Value to test against srcA.

Data Types: int8 | int16 | int32 | single

Output

EnableOut — Enable Output
off (default) | on

If the **EnableIn** input to the block is false, the logic implemented by the block is not executed and **EnableOut** signal is set to false. If **EnableIn** is true, **EnableOut** signal is set to true, if srcA is less than srcb.

See Also
EQU | GEQ | GRT | LEQ | NEQ

Introduced in R2019a

MCR

Master Control Reset
Library:

Description

The MCR block implements the MCR instruction. The block simulates a master control relay that can selectively disable a section of rungs.

Ports

Input

EnableIn — Enable Input
off (default) | on

Controls the execution of the block. **EnableIn** reflects the rung state preceding the block. If the rung state preceding the block is false, **EnableIn** is false, the block does not execute and the outputs are not updated.

Output

EnableOut — Enable Output
off (default) | on

When the block executes, it automatically sets the **EnableOut** to false. This disables all the subsequent blocks on the rung.

See Also
AFI | JMP | LBL | NOP | TND

Introduced in R2019a

MOV

Move
Library:

Description

The MOV block implements the `MOV` instruction. When the rung conditions are true, the block moves a copy of the source to the destination, keeping the source unchanged.

Ports

Input

EnableIn — Enable Input
off (default) | on

Controls the execution of the block. **EnableIn** reflects the rung state preceding the block. If the rung state preceding the block is false, **EnableIn** is false, the block does not execute and the outputs are not updated.

src — Input signal
scalar

Value to move.

Data Types: `int8` | `int16` | `int32` | `single`

Output

EnableOut — Enable Output
off (default) | on

By default, **EnableOut** follows the state of **EnableIn**. If the **EnableIn** input to the block is false, the logic implemented by the block is not executed and **EnableOut** signal is set to false.

dest — Output signal
scalar

Tag to store the result.
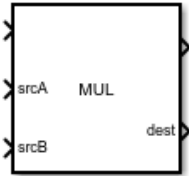
Data Types: `int8` | `int16` | `int32` | `single`

See Also
CLR

Introduced in R2019a

MUL

Multiply inputs
Library:

Description

The MUL block implements the `MUL` ladder logic instruction. When the rung conditions are true, the block multiplies the multiplicand at source A (`srcA`) with the multiplier at source B (`srcB`) and outputs the result to the destination (`dest`).

Ports

Input

EnableIn — Enable Input
off (default) | on

Controls the execution of the block. **EnableIn** reflects the rung state preceding the block. If the rung state preceding the block is false, **EnableIn** is false, the block does not execute and the outputs are not updated.

srcA — First input signal
scalar

The first input signal to the multiplication operation.

Data Types: `int8` | `int16` | `int32` | `single`

srcB — Second input signal
scalar

The second input signal to the multiplication operation.

Data Types: `int8` | `int16` | `int32` | `single`

Output

EnableOut — Enable Output
off (default) | on

By default, **EnableOut** follows the state of **EnableIn**. If the **EnableIn** input to the block is false, the logic implemented by the block is not executed and **EnableOut** signal is set to false.

dest — Output signal
scalar

Output signal resulting from the multiplication operation.

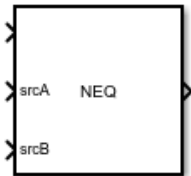Data Types: `int8` | `int16` | `int32` | `single`

See Also
ADD | CPT | DIV | SUB

Introduced in R2019a

NEQ

Not Equal To
Library:

Description

The NEQ block implements the `NEQ` instruction. When the rung conditions are true,the block verifies whether the value at source A is not equal to the value at source B.

Ports

Input

EnableIn — Enable Input
off (default) | on

Controls the execution of the block. **EnableIn** reflects the rung state preceding the block. If the rung state preceding the block is false, **EnableIn** is false, the block does not execute and the outputs are not updated.

srcA — First input signal
scalar

Value to test against `srcB`.

Data Types: `int8` | `int16` | `int32` | `single`

srcB — Second input signal
scalar

Value to test against `srcA`.

Data Types: `int8` | `int16` | `int32` | `single`

Output

EnableOut — Enable Output
off (default) | on

If the **EnableIn** input to the block is false, the logic implemented by the block is not executed and **EnableOut** signal is set to false. If **EnableIn** is true, **EnableOut** signal is set to true, if `srcA` is not equal to `srcb`.

See Also

EQU | GEQ | GRT | LEQ | LES

Introduced in R2019a

NOP

No Operation
Library:

Description

The NOP block implements the NOP function. The block acts as a placeholder. It performs no operation on enable or disable.

Ports

Input

EnableIn — Enable Input
off (default) | on

Controls the execution of the block. **EnableIn** reflects the rung state preceding the block. If the rung state preceding the block is false, **EnableIn** is false, the block does not execute and the outputs are not updated.

Output

EnableOut — Enable Output
off (default) | on

When the block executes, it automatically sets the **EnableOut** to false. This disables all the subsequent blocks on the rung.
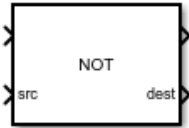
See Also
AFI | JMP | LBL | MCR | TND

Introduced in R2019a

NOT

Bitwise NOT
Library:

Description

The NOT block implements the `NOT` ladder logic instruction. When the rung conditions are true, the block performs bitwise NOT operation on the values at source. The result of this operation is available at the destination port (`dest`).

Ports

Input

EnableIn — Enable Input
off (default) | on

Controls the execution of the block. **EnableIn** reflects the rung state preceding the block. If the rung state preceding the block is false, **EnableIn** is false, the block does not execute and the outputs are not updated.

src — Input signal
scalar

The input signal on which to perform the bitwise NOT operation. If the datatype is `single` (REAL – ladder logic equivalent), the input value is converted to `int32` (DINT – ladder logic equivalent). `int8, int16` (SINT,INT – ladder logic equivalent) datatypes are converted to `int32` (DINT – ladder logic equivalent) by filling the upper bits with 0s.

Data Types: `int8` | `int16` | `int32` | `single`

Output

EnableOut — Enable Output
off (default) | on

By default, **EnableOut** follows the state of **EnableIn**. If the **EnableIn** input to the block is false, the logic implemented by the block is not executed and **EnableOut** signal is set to false.

dest — Output signal
scalar

Output signal resulting from the bitwise NOT operation. If the datatype is `single` (REAL – ladder logic equivalent), the resultant `int32` (DINT – ladder logic equivalent) is converted to REAL (`single` – ladder logic equivalent).

Data Types: `int8` | `int16` | `int32` | `single`

See Also
AND | OR

Introduced in R2019a

ONS

One Shot
Library:

Description

The ONS block implements the `ONS` instruction. The block makes the remainder of the rung true every time the `rung-condition-in` transitions from false to true.

Ports

Input

EnableIn — Enable Input
off (default) | on

Controls the execution of the block. **EnableIn** reflects the rung state preceding the block. If the rung state preceding the block is false, **EnableIn** is false, the block does not execute and the outputs are not updated.

Output

EN — Enable Output
off (default) | on

By default, **EnableOut** follows the state of **EnableIn**. If the **EnableIn** input to the block is false, the logic implemented by the block is not executed and **EnableOut** signal is set to false.

Parameters

Storage Bit — Internal storage bit
SB (default) | boolean

Internal storage bit that retains the `rung-condition-in` from the last time the instruction was executed. Specified in the format of `tags`. In Ladder Diagrams, tags (variables) are used for representing all inputs, outputs, and internal memory with attributes such as `Data Type`, `Initial Value`, and `size`. To change the attributes of the `Operand Tag`, open the **Program Variables** table within the Ladder Diagram Program block.

Programmatic Use
Block Parameter: PLCOperandTag
Type: character vector
Value: character vector
Default: `'SB'`

See Also

OSF | OSR

Introduced in R2019a

OR

Bitwise OR
Library:

Description

The OR block implements the `OR` ladder logic instruction. When the rung conditions are true, the block performs bitwise OR operation on the values at source A with the values at source B. The result of this operation is available at the destination port (`dest`).

Ports

Input

`EnableIn` — Enable Input
off (default) | on

Controls the execution of the block. **EnableIn** reflects the rung state preceding the block. If the rung state preceding the block is false, **EnableIn** is false, the block does not execute and the outputs are not updated.

`srcA` — First input signal
scalar

The first input signal to the bitwise OR operation. If the datatype is `single` (REAL – ladder logic equivalent), the input value is converted to `int32` (DINT – ladder logic equivalent). `int8, int16` (SINT,INT – ladder logic equivalent) datatypes are converted to `int32` (DINT – ladder logic equivalent) by filling the upper bits with 0s.

Data Types: `int8` | `int16` | `int32` | `single`

`srcB` — Second input signal
scalar

The second input signal to the bitwise OR operation. If the datatype is `single` (REAL – ladder logic equivalent), the input value is converted to `int32` (DINT – ladder logic equivalent). `int8, int16` (SINT,INT – ladder logic equivalent) datatypes are converted to `int32` (DINT – ladder logic equivalent) by filling the upper bits with 0s.

Data Types: `int8` | `int16` | `int32` | `single`

Output

`EnableOut` — Enable Output
off (default) | on

By default, **EnableOut** follows the state of **EnableIn**. If the **EnableIn** input to the block is false, the logic implemented by the block is not executed and **EnableOut** signal is set to false.

`dest` — Output signal
scalar

Output signal resulting from the bitwise OR operation. If the datatype is `single` (REAL – ladder logic equivalent), the resultant `int32` (DINT – ladder logic equivalent) is converted to REAL (`single` – ladder logic equivalent).

Data Types: `int8` | `int16` | `int32` | `single`

See Also
AND | NOT

Introduced in R2019a

OSF

One Shot Falling
Library:

Description

The OSF block implements the `OSF` instruction. The block sets the output bit for one scan when the `rung-condition-in` transitions from true to false.

Ports

Input

EnableIn — Enable Input
off (default) | on

Controls the execution of the block. **EnableIn** reflects the rung state preceding the block. If the rung state preceding the block is false, **EnableIn** is false, the block does not execute and the outputs are not updated.

Output

EN — Enable Output
off (default) | on

By default, **EnableOut** follows the state of **EnableIn**. If the **EnableIn** input to the block is false, the logic implemented by the block is not executed and **EnableOut** signal is set to false.

OB — Output Bit
boolean

Bit to be modified when the `rung-condition-in` transitions from false to true.

Data Types: `Boolean`

Parameters

Storage Bit — Internal storage bit
SB (default) | boolean

Internal storage bit that retains the `rung-condition-in` from the last time the instruction was executed. Specified in the format of `tags`. In Ladder Diagrams, tags (variables) are used for representing all inputs, outputs, and internal memory with attributes such as `Data Type`, `Initial Value`, and `size`. To change the attributes of the `Operand Tag`, open the **Program Variables** table within the Ladder Diagram Program block.

Programmatic Use
Block Parameter: PLCOperandTag
Type: character vector
Value: character vector
Default: 'SB'

See Also
ONS | OSR

Introduced in R2019a

OSR

One Shot Rising
Library:

Description

The OSR block implements the `OSR` instruction. The block sets the output bit for one scan when the `rung-condition-in` transitions from false to true.

Ports

Input

EnableIn — Enable Input
off (default) | on

Controls the execution of the block. **EnableIn** reflects the rung state preceding the block. If the rung state preceding the block is false, **EnableIn** is false, the block does not execute and the outputs are not updated.

Output

EN — Enable Output
off (default) | on

By default, **EnableOut** follows the state of **EnableIn**. If the **EnableIn** input to the block is false, the logic implemented by the block is not executed and **EnableOut** signal is set to false.

OB — Output Bit
boolean

Bit to be modified when the `rung-condition-in` transitions from false to true.

Data Types: `Boolean`

Parameters

Storage Bit — Internal storage bit
SB (default) | boolean

Internal storage bit that retains the `rung-condition-in` from the last time the instruction was executed. Specified in the format of `tags`. In Ladder Diagrams, tags (variables) are used for representing all inputs, outputs, and internal memory with attributes such as `Data Type`, `Initial Value`, and `size`. To change the attributes of the `Operand Tag`, open the **Program Variables** table within the Ladder Diagram Program block.

Programmatic Use
Block Parameter: PLCOperandTag
Type: character vector
Value: character vector
Default: 'SB'

See Also

ONS | OSF

Introduced in R2019a

OTE

Output Energize
Library:

Description

OTE is one of the building blocks of a ladder diagram. The OTE block implements the OTE ladder logic instruction. The block examines the state of the rung-condition-in and sets or clears the operand tag (data bit). If rung condition is true, the block sets the data bit to true.

Parameters

Operand Tag — Bit to be tested
A (default) | boolean

Specify the data bits to be modified. Data bits are specified in the format of tags. In Ladder Diagrams, tags (variables) are used for representing all inputs, outputs, and internal memory with attributes such as Data Type, Initial Value, and size. To change the attributes of the Operand Tag, open the **Program Variables** table within the Ladder Diagram Program block.

Programmatic Use
Block Parameter: PLCOperandTag
Type: character vector
Value: character vector
Default: 'A'

See Also
OTL | OTU | XIC | XIO

Introduced in R2019a

OTL

Output Latch
Library:

Description

OTL is one of the building blocks of a ladder diagram. The OTL block implements the OTL ladder logic instruction. When the rung condition is true, the block sets the operand tag (data bit) to true. The data bit remains true until it is cleared, typically by an OTU block. When the rung condition is changed to false, the block does not change the status of the data bit.

Parameters

Operand Tag — Bit to be tested
A (default) | boolean

Specify the data bits to be modified. Data bits are specified in the format of tags. In Ladder Diagrams, tags (variables) are used for representing all inputs, outputs, and internal memory with attributes such as Data Type, Initial Value, and size. To change the attributes of the Operand Tag, open the **Program Variables** table within the Ladder Diagram Program block.

Programmatic Use
Block Parameter: PLCOperandTag
Type: character vector
Value: character vector
Default: 'A'

See Also
OTE | OTU | XIC | XIO

Introduced in R2019a

OTU

Output Unlatch
Library:

Description

OTU is one of the building blocks of a ladder diagram. The OTU block implements the OTU ladder logic instruction. When the rung condition is true, the block clears the operand tag (data bit) to false. When the rung condition is changed to false, the block does not change the status of the data bit. It is generally used after the OTL block to unlatch the state and disable the rung.

Parameters

Operand Tag — Bit to be tested
A (default) | boolean

Specify the data bits to be modified. Data bits are specified in the format of tags. In Ladder Diagrams, tags (variables) are used for representing all inputs, outputs, and internal memory with attributes such as Data Type, Initial Value, and size. To change the attributes of the Operand Tag, open the **Program Variables** table within the Ladder Diagram Program block.

Programmatic Use
Block Parameter: PLCOperandTag
Type: character vector
Value: character vector
Default: 'A'

See Also
OTE | OTL | XIC | XIO

Introduced in R2019a

PLC Controller

PLC Controller
Library:

Description

In Ladder Diagram, the controller is a top organisational unit that typically consists of task and controller tags. There can be a single task or multiple tasks. All the tasks inside a controller are executed in parallel. There are two types of controller blocks available in Simulink PLC Coder™:

- PLC Controller Suite
- PLC Controller

PLC controller suite is a block hierarchy that models a simple complete Ladder Logic Controller structure whereas a PLC controller block consists of ladder logic semantics. The controller tags store the information of global variables such as DataType, Mapping type, Port, Address etc. The global variables defined for a ladder diagram form the input and output ports of the controller.
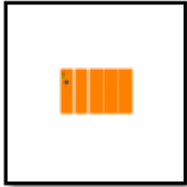
See Also

AOI Runner | Ladder Diagram Program | Ladder Diagram Subroutine | Ladder Diagram Function Block (AOI) | Task

Introduced in R2019a

PLC Controller Suite

PLC Controller
Library:

Description

In Ladder Diagram, the controller is a top organisational unit that typically consists of task and controller tags. There can be a single task or multiple tasks. All the tasks inside a controller are executed in parallel. There are two types of controller blocks available in Simulink PLC Coder:

- PLC Controller Suite
- PLC Controller

PLC controller suite is a block hierarchy that models a simple complete Ladder Logic Controller structure whereas a PLC controller block consists of ladder logic semantics. The controller tags store the information of global variables such as DataType, Mapping type, Port, Address etc. The global variables defined for a ladder diagram form the input and output ports of the controller.

See Also
AOI Runner | Ladder Diagram Program | Ladder Diagram Subroutine | Ladder Diagram Function Block (AOI) | Task

Power Rail Start

Power Rail Start
Library:

Description

The block is for reference purpose only. Do not use this block for ladder diagram modeling.

See Also

Junction | Power Rail Terminal | Rung 1

Introduced in R2019a

Power Rail Terminal

Power Rail Terminal
Library:

Description

The block is for reference purpose only. Do not use this block for ladder diagram modeling.

See Also
Junction | Power Rail Start | Rung 1
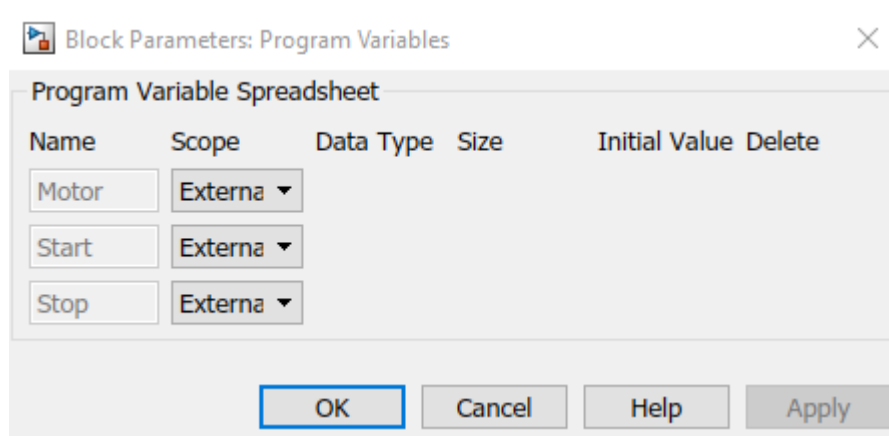
Introduced in R2019a

ProgramVariables

Syntax

Description

The **Program Variables** table within the Ladder Diagram Program block contains attributes associated with tags. The tags can have attributes such as `Data Type`, `Initial Value`, and `size`.

For example, the program variables table can be used to specify attributes for variables (tags) as shown.

See Also

AOI Runner | Ladder Diagram Program | Ladder Diagram Subroutine | Ladder Diagram Function Block (AOI) | Task

Introduced in R2019a

RES

Reset
Library:

Description

The RES block implements the RES ladder logic instruction. When the rung conditions are true, the block resets the value of TIMER, COUNTER, or a CONTROL structure.

Ports

Input

EnableIn — Enable Input
off (default) | on

Controls the execution of the block. **EnableIn** reflects the rung state preceding the block. If the rung state preceding the block is false, **EnableIn** is false, the block does not execute and the outputs are not updated.

Output

EN — Enable Output
off (default) | on

By default, **EnableOut** follows the state of **EnableIn**. If the **EnableIn** input to the block is false, the logic implemented by the block is not executed and **EnableOut** signal is set to false.

Parameters

Operand Tag To Reset — Structure to Reset
A (default) | character vector

Specify the tag to reset. In Ladder Diagrams, tags (variables) are used for representing all inputs, outputs, and internal memory with attributes such as Data Type, Initial Value, and size. To change the attributes of the Operand Tag To Reset, open the **Program Variables** table within the Ladder Diagram Program block.

Programmatic Use
Block Parameter: PLCTagToReset
Type: character vector
Value: character vector
Default: 'A'

Tag Structure — Structure Type
'TIMER or COUNTER' (default) | 'CONTROL'

Specify the structure type to reset.

Type	Description
TIMER or COUNTER	Clears the ACC field of the structure.
CONTROL	Clears the POS field of the structure.

Programmatic Use
Block Parameter: PLCTagDataType
Type: character vector
Value: 'TIMER or COUNTER'|'CONTROL'
Default: 'TIMER or COUNTER'

See Also
CTD | CTU | RTO | TOF | TON

Introduced in R2019a

RET

Return
Library:

Description

The RET block implements the RET instruction. The blocks is used to return the control of execution from a subroutine. It can be used only inside a subroutine. This block does not support parameters to be passed from the subroutine.

Parameters

Sample time (-1 for inherited) — Discrete interval between sample time hits
-1 (default) | scalar

Enter the discrete interval between sample time hits or specify -1 to inherit the sample time

See also "Specify Sample Time".

Programmatic Use
Block Parameter: SampleTime
Type: character vector
Value: real scalar
Default: '-1'

See Also

Introduced in R2019a

RTO

Retentive Timer On
Library:

Description

The RTO block implements the RTO ladder logic instruction. When the rung conditions are true, the block accumulates time until:

- The timer is disabled
- The timer completes

Ports

Input

EnableIn — Enable Input
off (default) | on

Controls the execution of the block. **EnableIn** reflects the rung state preceding the block. If the rung state preceding the block is false, **EnableIn** is false, the block does not execute and the outputs are not updated.

Output

EN — Enable Output
off (default) | on

By default, **EnableOut** follows the state of **EnableIn**. If the **EnableIn** input to the block is false, the logic implemented by the block is not executed and **EnableOut** signal is set to false.

Parameters

Timer Tag — Timer Structure
T (default) | character vector

Specify the timer parameters in the format of `tags`. In Ladder Diagrams, tags (variables) are used for representing all inputs, outputs, and internal memory with attributes such as `Data Type`, `Initial Value`, and `size`. To change the attributes of the `Operand Tag`, open the **Program Variables** table within the Ladder Diagram Program block. The `Data Type` of the timer tag is of the `Bus:TIMER` type with its initial value specified as a structure containing the following fields:

Field	Description	Default Value
PRE	The preset value specifies the value (1 millisecond units) which the accumulated value must reach before the instruction indicates it is done	0
ACC	The accumulated value specifies the number of milliseconds that have elapsed since the RTO instruction was enabled.	0
EN	The enable bit contains rung-condition-in when the instruction was last executed.	0
TT	The timing bit when set indicates the timing operation is in process.	0
DN	The done bit when set indicates the timing operation is complete (or paused).	0

Programmatic Use
Block Parameter: PLCOperandTag
Type: character vector
Value: character vector
Default: 'T'

See Also

CTD | CTU | RES | TOF | TON

Introduced in R2019a

RungTerminal

Rung Terminal
Library:

Description

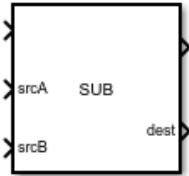The block is for reference purpose only. Do not use this block for ladder diagram modeling.

See Also
Junction | Power Rail Terminal | Power Rail Start

Introduced in R2019a

SUB

Subtract inputs
Library:

Description

The SUB block implements the SUB ladder logic instruction. When the rung conditions are true, the block subtracts source B (srcB) from source A (srcA) and outputs the result to the destination (dest).

Ports

Input

EnableIn — Enable Input
off (default) | on

Controls the execution of the block. **EnableIn** reflects the rung state preceding the block. If the rung state preceding the block is false, **EnableIn** is false, the block does not execute and the outputs are not updated.

srcA — First input signal
scalar

The first input signal to the subtraction operation.

Data Types: int8 | int16 | int32 | single

srcB — Second input signal
scalar

The second input signal to the subtraction operation.

Data Types: int8 | int16 | int32 | single

Output

EnableOut — Enable Output
off (default) | on

By default, **EnableOut** follows the state of **EnableIn**. If the **EnableIn** input to the block is false, the logic implemented by the block is not executed and **EnableOut** signal is set to false.

dest — Output signal
scalar

Output signal resulting from the subtraction operation.

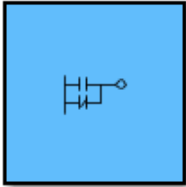Data Types: `int8` | `int16` | `int32` | `single`

See Also
ADD | CPT | DIV | MUL

Introduced in R2019a

Task

Task
Library:

Description

The task block is placed inside a PLC controller. There can be multiple tasks inside a controller. Each task consists of a program or multiple programs.

You can use the `get_param` and `set_param` functions to control the settings of the Task block.

Property Name	Description
`'Name'`	Task name
`'SystemSampleTime'`	Task rate
`'PLCTaskWatchDog'`	Watch dog timer value
`'Priority'`	Task priority
`'Description'`	Task Description

See Also

AOI Runner | Ladder Diagram Program | Ladder Diagram Subroutine | Ladder Diagram Function Block (AOI) | PLC Controller | Task

Introduced in R2019a

TND

Temporary End
Library:

Description

The TND block implements the TND ladder logic instruction. When the rung conditions are true, the block acts as the end of ladder diagram execution.

Ports

Input

EnableIn — Enable Input
off (default) | on

Controls the execution of the block. **EnableIn** reflects the rung state preceding the block. If the rung state preceding the block is false, **EnableIn** is false, the block does not execute and the outputs are not updated.

Output

EN — Enable Output
off (default) | on

By default, **EnableOut** follows the state of **EnableIn**. If the **EnableIn** input to the block is false, the logic implemented by the block is not executed and **EnableOut** signal is set to false.

See Also
AFI | JMP | LBL | MCR | NOP

Introduced in R2019a

TOF

Timer Off Delay
Library:
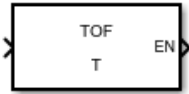
Description

The TOF block implements the TOF ladder logic instruction. When the rung conditions are true, the block accumulates time until:

- The timer is disabled
- The timer completes

Ports

Input

EnableIn — Enable Input
off (default) | on

Controls the execution of the block. **EnableIn** reflects the rung state preceding the block. If the rung state preceding the block is false, **EnableIn** is false, the block does not execute and the outputs are not updated.

Output

EN — Enable Output
off (default) | on

By default, **EnableOut** follows the state of **EnableIn**. If the **EnableIn** input to the block is false, the logic implemented by the block is not executed and **EnableOut** signal is set to false.

Parameters

Timer Tag — Timer Structure
T (default) | character vector

Specify the timer parameters in the format of `tags`. In Ladder Diagrams, tags (variables) are used for representing all inputs, outputs, and internal memory with attributes such as `Data Type`, `Initial Value`, and `size`. To change the attributes of the `Operand Tag`, open the **Program Variables** table within the Ladder Diagram Program block. The `Data Type` of the timer tag is of the `Bus:TIMER` type with its initial value specified as a structure containing the following fields:

Field	Description	Default Value
PRE	The preset value specifies the value (1 millisecond units) which the accumulated value must reach before the instruction indicates it is done	0
ACC	The accumulated value specifies the number of milliseconds that have elapsed since the TOF instruction was enabled.	0
EN	The enable bit contains rung-condition-in when the instruction was last executed.	0
TT	The timing bit when set indicates the timing operation is in process.	0
DN	The done bit when set indicates the timing operation is complete (or paused).	0

Programmatic Use
Block Parameter: PLCOperandTag
Type: character vector
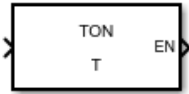**Value**: character vector
Default: 'T'

See Also
CTD | CTU | RES | RTO | TON

Introduced in R2019a

TON

Timer On Delay
Library:

Description

The TON block implements the TON ladder logic instruction. When the rung conditions are true, the block accumulates time until:

- The timer is disabled
- The timer completes

Ports

Input

EnableIn — Enable Input
off (default) | on

Controls the execution of the block. **EnableIn** reflects the rung state preceding the block. If the rung state preceding the block is false, **EnableIn** is false, the block does not execute and the outputs are not updated.

Output

EN — Enable Output
off (default) | on

By default, **EnableOut** follows the state of **EnableIn**. If the **EnableIn** input to the block is false, the logic implemented by the block is not executed and **EnableOut** signal is set to false.

Parameters

Timer Tag — Timer Structure
T (default) | character vector

Specify the timer parameters in the format of `tags`. In Ladder Diagrams, tags (variables) are used for representing all inputs, outputs, and internal memory with attributes such as `Data Type`, `Initial Value`, and `size`. To change the attributes of the `Operand Tag`, open the **Program Variables** table within the Ladder Diagram Program block. The `Data Type` of the timer tag is of the `Bus:TIMER` type with its initial value specified as a structure containing the following fields:

Field	Description	Default Value
PRE	The preset value specifies the value (1 millisecond units) which the accumulated value must reach before the instruction indicates it is done	0
ACC	The accumulated value specifies the number of milliseconds that have elapsed since the TON instruction was enabled.	0
EN	The enable bit contains rung-condition-in when the instruction was last executed.	0
TT	The timing bit when set indicates the timing operation is in process.	0
DN	The done bit when set indicates the timing operation is complete (or paused).	0

Programmatic Use
Block Parameter: `PLCOperandTag`
Type: character vector
Value: character vector
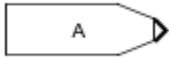**Default**: `'T'`

See Also

CTD | CTU | RES | RTO | TOF

Introduced in R2019a

Variable Read

Variable Read
Library:

Description

The block is used as an input variable assignment.

Parameters

Data Tag — Data to be read
A (default) | boolean

Specify the data to be read, specified in the format of `tags`. In Ladder Diagrams, tags (variables) are used for representing all inputs, outputs, and internal memory with attributes such as `Data Type`, `Initial Value`, and `size`. To change the attributes of the `Data Tag`, open the **Program Variables** table within the Ladder Diagram Program block.

Programmatic Use
Block Parameter: PLCOperandTag
Type: character vector
Value: character vector
Default: 'A'

See Also
Variable Write

Introduced in R2019a

Variable Write

Variable Write
Library:

Description

The block is used as an output variable assignment.

Parameters

Data Tag — Data to be written
A (default) | boolean

Specify the data bits to be written, specified in the format of `tags`. In Ladder Diagrams, tags (variables) are used for representing all inputs, outputs, and internal memory with attributes such as `Data Type`, `Initial Value`, and `size`. To change the attributes of the `Data Tag`, open the **Program Variables** table within the Ladder Diagram Program block.

Programmatic Use
Block Parameter: PLCOperandTag
Type: character vector
Value: character vector
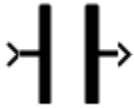**Default**: 'A'

See Also
Variable Read

Introduced in R2019a

XIC

Examine If Closed
Library:

Description

XIC is one of the building blocks of a ladder diagram. The XIC block implements the XIC ladder logic instruction. The block examines the operand tag (data bit) for an on condition. If the bit is set, the rung is enabled. If the bit is clear, the rung is disabled.

Parameters

`Operand Tag` — **Bit to be tested**
A (default) | boolean

Specify the data bits to be tested. Data bits are specified in the format of `tags`. In Ladder Diagrams, tags (variables) are used for representing all inputs, outputs, and internal memory with attributes such as `Data Type`, `Initial Value`, and `size`. To change the attributes of the `Operand Tag`, open the **Program Variables** table within the Ladder Diagram Program block.

Programmatic Use
Block Parameter: PLCOperandTag
Type: character vector
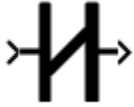**Value**: character vector
Default: 'A'

See Also
OTE | OTL | OTU | XIO

Introduced in R2019a

XIO

Examine If Open
Library:

Description

XIO is one of the building blocks of a ladder diagram. The XIO block implements the XIO ladder logic instruction. The block examines the operand tag (data bit) for an off condition. If the bit is clear, the rung is enabled. If the bit is set, the rung is disabled.

Parameters

`Operand Tag` — Bit to be tested
A (default) | boolean

Specify the data bits to be tested. Data bits are specified in the format of `tags`. In Ladder Diagrams, tags (variables) are used for representing all inputs, outputs, and internal memory with attributes such as `Data Type`, `Initial Value`, and `size`. To change the attributes of the `Operand Tag`, open the **Program Variables** table within the Ladder Diagram Program block.

Programmatic Use
Block Parameter: PLCOperandTag
Type: character vector
Value: character vector
Default: 'A'

See Also
OTE | OTL | OTU | XIC

Introduced in R2019a

Functions

plccoderdemos

Product examples

Syntax

`plccoderdemos`

Description

`plccoderdemos` displays the Simulink PLC Coder examples.

Examples

Display List of Simulink PLC Coder Examples

To display a list of examples, at the command prompt enter:

`plccoderdemos`

See Also
`plcopenconfigset`

Introduced in R2010a

plccoderpref

Manage user preferences

Syntax

```
plccoderpref

plccoderpref('plctargetide')

plccoderpref('plctargetide', preference_value )

plccoderpref('plctargetide', 'default')

plccoderpref('plctargetidepaths')

plccoderpref('plctargetidepaths','default')

plccoderpref('plctargetlist')

plccoderpref('plctargetlist',targetlist)
```

Description

`plccoderpref` displays the current set of user preferences, including the default target IDE.

`plccoderpref('plctargetide')` returns the current default target IDE. This default can be the target IDE set previously, or the factory default. The factory default is `'codesys23'`.

`plccoderpref('plctargetide', preference_value )` sets the default target IDE to the one that you specify in *preference_value* . This command sets the *preference_value* to persist as the default target IDE for future MATLAB® sessions.

`plccoderpref('plctargetide', 'default')` sets the default target IDE to the factory default target IDE (`'codesys23'`).

`plccoderpref('plctargetidepaths')` returns a 1-by-1 structure of the installation paths of supported target IDEs.

`plccoderpref('plctargetidepaths','default')` sets the contents of the 1-by-1 structure of the installation paths to the default values.

`plccoderpref('plctargetlist')` displays the target IDEs that appear in the reduced **Target IDE** list in the Simulink Configuration Parameters dialog box. For more information, see "Target IDE" and "Show Full Target List".

`plccoderpref('plctargetlist',targetlist)` sets the target IDEs that appear in the reduced **Target IDE** list to the values that you specify in *targetlist*.

Examples

Return Current Default Target IDE

- ```
  plccoderpref('plctargetide')
  ```

  ```
  ans =
  'rslogix5000'
  ```

Set `rslogix5000` as New Default Target IDE

- ```
  plccoderpref('plctargetide', 'rslogix5000')
  ```

  ```
  ans =
  'rslogix5000'
  ```

See Installation Paths of Supported Target IDEs

- ```
  plccoderpref('plctargetidepaths')
  ```

  ```
  ans = struct with fields:
                codesys23: 'C:\Program Files (x86)\3S Software'
                codesys33: 'C:\Program Files\3S CoDeSys'
                codesys35: 'C:\Program Files\3S CoDeSys'
               studio5000: ''
       studio5000_routine: ''
               rslogix5000: ''
       rslogix5000_routine: ''
            brautomation30: 'C:\Program Files\BrAutomation'
            brautomation40: 'C:\Program Files\BrAutomation'
               multiprog50: 'C:\Program Files\KW-Software\MULTIPROG 5.0'
                  pcworx60: 'C:\Program Files\Phoenix Contact\Software Suite 150'
                     step7: 'C:\Program Files\Siemens'
                  tiaportal: 'C:\Program Files\Siemens\Automation'
           tiaportal_double: 'C:\Program Files\Siemens\Automation'
                   plcopen: ''
                 twincat211: 'C:\TwinCAT'
                   twincat3: 'C:\TwinCAT'
                   generic: ''
                 indraworks: ''
                     omron: ''
                  selectron: ''
  ```

Customize Reduced Target IDE List

- ```
  targetlist = {'codesys23','rslogix5000'};
  plccoderpref('plctargetlist',targetlist)
  ```

  ```
  ans = 1×2 cell
      {'codesys23'}    {'rslogix5000'}
  ```

Reset Reduced Target IDE List

- `plccoderpref('plctargetlist','default')`

  ```
  ans = 1×5 cell
      {'codesys23'}    {'studio5000'}    {'step7'}    {'omron'}    {'plcopen'}
  ```

Append Another IDE to Default Reduced Target IDE List

- `plccoderpref('plctargetlist', [plccoderpref('plctargetlist', 'default') 'codesys35'])`

  ```
  ans = 1×6 cell
      {'codesys23'}    {'studio5000'}    {'step7'}    {'omron'}    {'plcopen'}    {'codesys35'}
  ```

Append Another IDE to Current Reduced Target IDE List

- `plccoderpref('plctargetlist', [plccoderpref('plctargetlist') 'codesys35'])`

  ```
  ans = 1×8 cell
      {'codesys23'}    {'studio5000'}    {'step7'}    {'omron'}    {'plcopen'}    {'codesys35'}
  ```

Input Arguments

plctargetide — Name of the target IDE
character vector

String directive that specifies the default target IDE.

Value	Description
`codesys23`	3S-Smart Software Solutions CoDeSys Version 2.3 (default) target IDE
`codesys33`	3S-Smart Software Solutions CoDeSys Version 3.3 target IDE
`codesys35`	3S-Smart Software Solutions CoDeSys Version 3.5 target IDE
`brautomation30`	B&R Automation Studio® 3.0 target IDE
`brautomation40`	B&R Automation Studio 4.0 target IDE
`generic`	Generic target IDE
`indraworks`	Rexroth IndraWorks version 13V12 IDE
`multiprog50`	PHOENIX CONTACT (previously KW) Software MULTIPROG® 5.0 or 5.50 target IDE
`omron`	OMRON® Sysmac® Studio
`plcopen`	PLCopen XML target IDE
`pcworx60`	Phoenix Contact® PC WORX™ 6.0

Value	Description
`rslogix5000`	Rockwell Automation RSLogix™ 5000 Series target IDE for AOI format
`rslogix5000_routine`	Rockwell Automation RSLogix 5000 Series target IDE for routine format
`step7`	Siemens® SIMATIC® STEP® 7 Version 5 target IDE
`studio5000`	Rockwell Studio 5000 Logix Designer target IDE for AOI format
`studio5000_routine`	Rockwell Studio 5000 Logix Designer target IDE for routine format
`twincat211`	Beckhoff® TwinCAT® 2.11 target IDE
`twincat3`	Beckhoff TwinCAT 3 target IDE
`tiaportal`	Siemens TIA Portal
`tiaportal_double`	Siemens TIA Portal with support for double precision (LREAL type)

`plctargetidepaths` — Target IDE installation path
character vector

String that specifies the target IDE installation path. Contains a 1-by-1 structure of the installation paths of supported target IDEs.

```
codesys23: 'C:\Program Files\3S Software'
codesys33: 'C:\Program Files\3S CoDeSys'
codesys35: 'C:\Program Files\3S CoDeSys'
studio5000: 'C:\Program Files\Rockwell Software'
studio5000_routine: 'C:\Program Files\Rockwell Software'
rslogix5000: 'C:\Program Files\Rockwell Software'
rslogix5000_routine: 'C:\Program Files\Rockwell Software'
brautomation30: 'C:\Program Files\BrAutomation'
brautomation40: 'C:\Program Files\BrAutomation'
multiprog50: 'C:\Program Files\KW-Software\MULTIPROG 5.0'
pcworx60: 'C:\Program Files\Phoenix Contact\Software Suite 150'
step7: 'C:\Program Files\Siemens'
tiaportal: 'C:\Program Files\Siemens\Automation'
tiaportal_double: 'C:\Program Files\Siemens\Automation'
plcopen: ''
twincat211: 'C:\TwinCAT'
twincat3: 'C:\TwinCAT'
generic: ''
indraworks: ''
omron: ''
```

`plctargetlist` — List of your target IDEs
cell array of string

Cell array of strings. Each string specifies a target IDE. You can specify any target IDE that is available for the `plctargetide` argument.

Use the string `default` to reset the reduced **Target IDE** list.

Tips

Use the Simulink Configuration Parameters dialog box to change the installation path of a target IDE (**Target IDE Path**).

Introduced in R2010a

plcgeneratecode

Generate structured text or ladder diagram (L5X) for the atomic subsystem

Syntax

```
generatedfiles= plcgeneratecode(system)
```

Description

`generatedfiles= plcgeneratecode(system)` generates Structure Text or Ladder Diagram for the specified atomic subsystem in a model.

- Structured Text is generated for the specified atomic subsystem in a model. Argument `system` is the fully qualified path name of the atomic subsystem. `generatedfiles` is a cell array of the generated file names. You must first load or start the model.
- Ladder Diagram (L5X) file is generated for the specified system in the model. Argument `system` is the fully qualified path name of the top organizational unit in the Simulink model. The `system` should either be a PLC Controller block, Ladder Diagram Function (AOI) block or an AOI Runner block. `generatedfiles` is a cell array of the generated file names. You must first load or start the Simulink model.

Examples

Generate Structured Text Code for Subsystem

Open or load the model containing the subsystem.

```
plcdemo_simple_subsystem
```
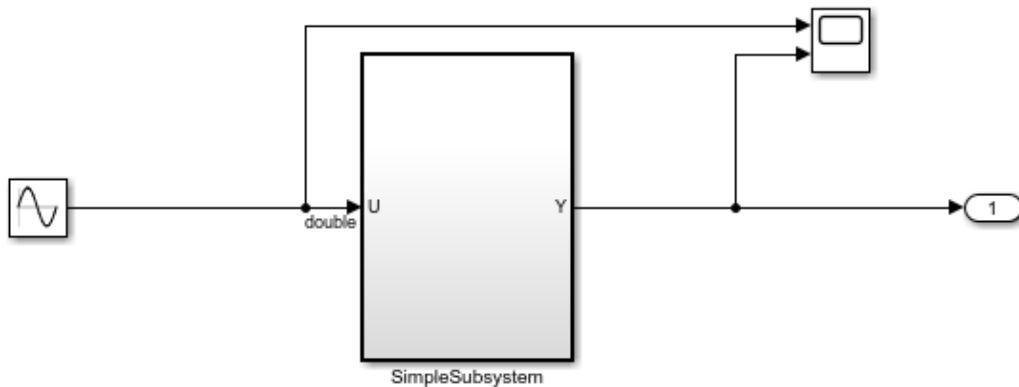
SimpleSubsystem

This introductory model shows the code generated for a simple subsystem consisting of a few basic Simulink blocks. To build the subsystem, right-click on the subsystem block and select PLC Code > Generate Code for Subsystem.

The Diagnostic Viewer displays hyperlinks to the generated code files, click the links to view the generated files.
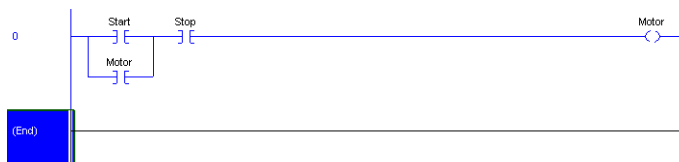
Copyright 2009-2019 The MathWorks, Inc.

Generate code for the subsystem, `plcdemo_simple_subsystem/SimpleSubsystem`.

```
generatedFiles = plcgeneratecode('plcdemo_simple_subsystem/SimpleSubsystem');
```

Generate PLC Code for Ladder diagram model

The following example demonstrates how to import a simple ladder diagram from an L5X file (`simpleController.L5X`) into the Simulink environment and then generate Ladder Diagram (L5X) from the imported model. The ladder L5X file was created using RSLogix 5000 IDE and contains contacts and coils representing switches and motor. The following is a snapshot of the ladder structure.

Use the `plcladderimport` function to import the ladder into Simulink.

```
[mdlName,mdlLib,busScript] = plcimportladder('simpleController.L5X','OpenModel','On')
```

The imported model contains a PLC Controller block named `simpleController`, followed by a Task block named `MainTask` and finally a Ladder Diagram Program block named `MainProgram`. The model imported into Simulink has blocks that implement the functionality of the contacts and coils.
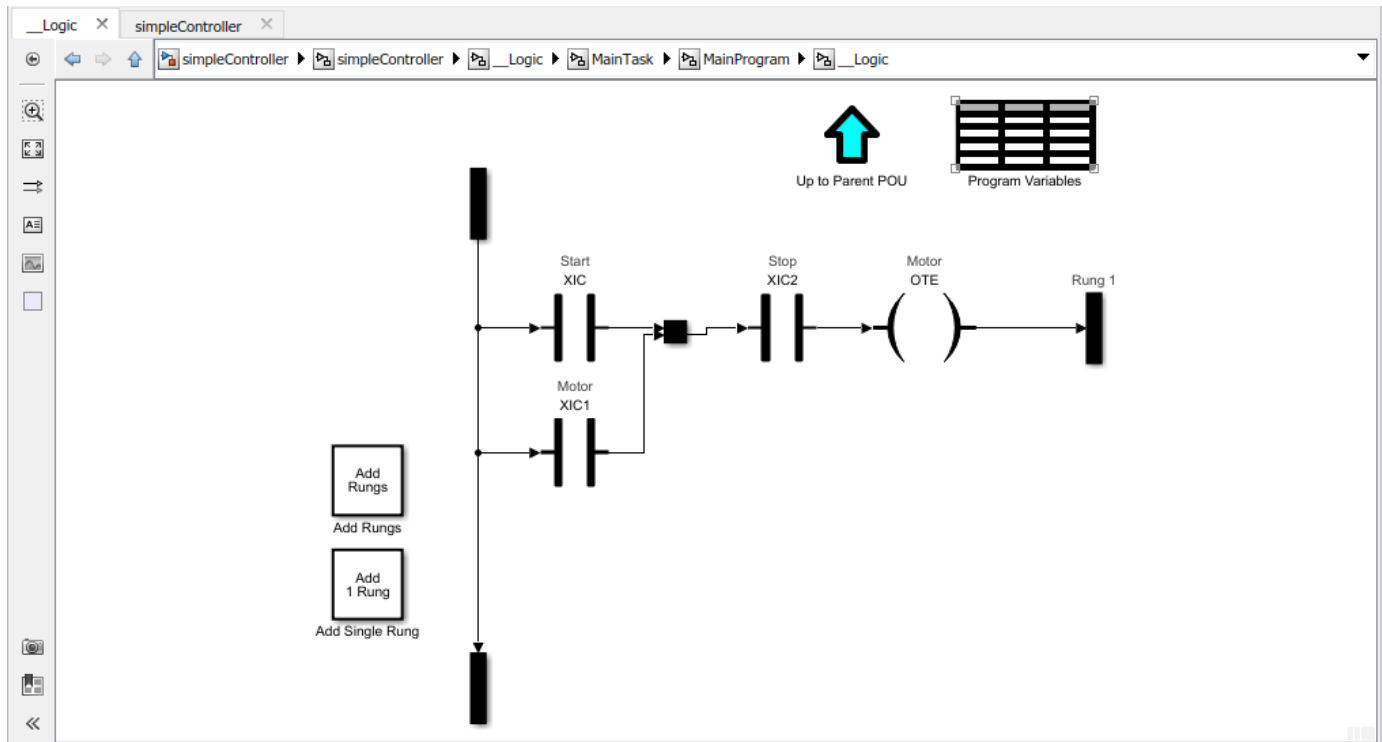
Generate code for the subsystem, `simpleController/simpleController`.

`generatedFiles = plcgeneratecode('simpleController/simpleController');`

`PLC code generation successful for 'simpleController/simpleController'.`

`Generated ladder files:`
`plcsrc\simpleController_gen.L5X`

Input Arguments

system — Full file path
character vector

For Structured Text, system specifies the relative or absolute path to the atomic subsystem in the Simulink model.

For Ladder Diagram, system specifies the relative or absolute path to the Simulink model, imported from the Ladder L5X file or a manually created model.

Output Arguments

generatedfiles — Array of generated file names
character array

Specifies the name of the generated code and testbench files.

See Also

plcgeneratecode | plcgeneraterunnertb | plcimportladder | plcladderlib | plcladderoption | plcopenconfigset

Introduced in R2010a

plcopenconfigset

Open Configuration Parameters dialog box for subsystem

Syntax

```
plcopenconfigset('subsystem')
```

Description

`plcopenconfigset('subsystem')` opens the Configuration Parameters dialog box for the specified atomic subsystem in the model. *subsystem* is the fully qualified path name of the atomic subsystem.

Examples

Open Configuration Parameters for Subsystem

This example shows you how to retrieve the configuration parameters for the subsystem from which you generate structured text code.

Open the model containing the subsystem.

```
open_system('plcdemo_simple_subsystem')
```

Open the Configuration Parameters dialog box for the subsystem, `plcdemo_simple_subsystem/SimpleSubsystem`.

```
plcopenconfigset('plcdemo_simple_subsystem/SimpleSubsystem')
```

Introduced in R2010a

plccheckforladder

(Has been removed) Check whether Stateflow chart is ready for Ladder Diagram code generation

---

Note plccheckforladder will be removed in a future release. For more information, see "Compatibility Considerations".

Syntax

plccheckforladder(chartPath)

Description

plccheckforladder(chartPath) checks whether a Stateflow chart (Stateflow) is ready for Ladder Diagram code generation. If the chart has properties that do not allow Ladder Diagram code generation, the function shows errors in the Diagnostic Viewer window.

Examples

Preparation of Stateflow Chart for Ladder Diagram Code Generation

Open the model plcdemo_ladder_three_aspect.

open_system('plcdemo_ladder_three_aspect')

The model contains a subsystem Subsys, which contains a Stateflow chart, 3Aspect. Save the model elsewhere with the name plcdemo_ladder_three_aspect_copy.

Enable super step semantics for the chart. In the chart properties, select **Enable Super Step Semantics**.

Check whether the Stateflow chart is ready for Ladder Diagram code generation.

plccheckforladder('plcdemo_ladder_three_aspect_copy/Subsys/3Aspect')

You see the following error message in the Diagnostic Viewer window:

Chart must not have superstep semantics enabled in Objects: 'Subsys/3Aspect'

Prepare the chart for Ladder Diagram code generation.

plcprepareforladder('plcdemo_ladder_three_aspect_copy/Subsys/3Aspect')

Check again whether the chart is ready for Ladder Diagram code generation.

plccheckforladder('plcdemo_ladder_three_aspect_copy/Subsys/3Aspect')

There are no more error messages. The function `plcprepareforladder` has disabled super step semantics for the chart.

Input Arguments

`chartPath` — Full path name of the Stateflow chart
character vector

Full path name of the Stateflow chart relative to the top level Simulink model, specified as a character vector. To obtain the full path, select the Stateflow chart in your model and use the `gcb` function.

Example: `gcb, 'ThreeAspectAutoSignal/Subsystem/AutoSignalChart'`

Compatibility Considerations

plccheckforladder will be removed
Not recommended starting in R2019a

plccheckforladder will be removed in a future release. Use Simulink model for ladder logic instead of Stateflow charts. Use the blocks from the PLC Ladder library to create a model that is compatible with ladder diagram generation. To open this library, type `plcladderlib` at the MATLAB command prompt.

See Also

Topics
"Prepare Chart for Simulink PLC Coder Ladder Diagram Code Generation"
"Generate Simulink PLC Coder Ladder Diagram Code from Stateflow Chart"
"Import Ladder Diagram Code to CODESYS 3.5 IDE and Validate Diagram"
"Simulink PLC Coder Ladder Diagram Code Generation"
"Supported IDE Platforms"

Introduced in R2016b

plcprepareforladder

(Has been removed) Change some Stateflow chart properties to enable Ladder Diagram code generation

---

Note plcprepareforladder will be removed in a future release. For more information, see "Compatibility Considerations".

Syntax

```
plcprepareforladder(chartPath)
```

Description

plcprepareforladder(chartPath) changes certain properties of a Stateflow chart (Stateflow) so that the chart is ready for Ladder Diagram code generation. The following properties are changed:

- The data types of inputs and outputs are changed to Boolean.
- The action language of the chart is changed to C.
- Super step semantics and chart initialization at execution are disabled.

Examples

Preparation of Stateflow Chart for Ladder Diagram Code Generation

Open the model plcdemo_ladder_three_aspect.

```
open_system('plcdemo_ladder_three_aspect')
```

The model contains a subsystem Subsys, which contains a Stateflow chart, 3Aspect. Save the model elsewhere with the name plcdemo_ladder_three_aspect_copy.

Enable super step semantics for the chart. In the chart properties, select **Enable Super Step Semantics**.

Check whether the Stateflow chart is ready for Ladder Diagram code generation.

```
plccheckforladder('plcdemo_ladder_three_aspect_copy/Subsys/3Aspect')
```

You see the following error message in the Diagnostic Viewer window:

```
Chart must not have superstep semantics enabled in Objects: 'Subsys/3Aspect'
```

Prepare the chart for Ladder Diagram code generation.

```
plcprepareforladder('plcdemo_ladder_three_aspect_copy/Subsys/3Aspect')
```

Check again whether the chart is ready for Ladder Diagram code generation.

```
plccheckforladder('plcdemo_ladder_three_aspect_copy/Subsys/3Aspect')
```

There are no more error messages. The function `plcprepareforladder` has disabled super step semantics for the chart.

Tips

- Before you use this function, make a backup copy of your model because the function changes chart properties.
- The function does not change all properties that would allow for Ladder Diagram code generation. You must explicitly change certain properties. For the full list of chart properties that are not allowed, see "Restrictions on Stateflow Chart for Ladder Diagram Generation".

Input Arguments

`chartPath` — Full path name of the Stateflow chart
character vector

Full path name of the Stateflow chart relative to the top level Simulink model, specified as a character vector. To obtain the full path, select the Stateflow chart in your model and use the `gcb` function.

Example: `gcb, 'ThreeAspectAutoSignal/Subsystem/AutoSignalChart'`

Compatibility Considerations

plcprepareforladder will be removed
Not recommended starting in R2019a

plcprepareforladder will be removed in a future release.Use Simulink model for ladder logic instead of Stateflow charts.

See Also

Topics
"Prepare Chart for Simulink PLC Coder Ladder Diagram Code Generation"
"Generate Simulink PLC Coder Ladder Diagram Code from Stateflow Chart"
"Import Ladder Diagram Code to CODESYS 3.5 IDE and Validate Diagram"
"Simulink PLC Coder Ladder Diagram Code Generation"
"Supported IDE Platforms"

Introduced in R2016b

plcgenerateladder

(Has been removed) Generate Ladder Diagram code from Stateflow chart

Note plcgenerateladder will be removed in a future release. For more information, see "Compatibility Considerations".

Syntax

```
plcgenerateladder(chartPath)
plcgenerateladder(chartPath,Name,Value)
```

Description

`plcgenerateladder(chartPath)` generates code from a Stateflow chart (Stateflow) that you can import to an IDE such as CODESYS 3.5 and view as a ladder diagram.

`plcgenerateladder(chartPath,Name,Value)` uses additional options specified by one or more `Name,Value` pair arguments. For instance, you can create a validation model or test bench to compare the generated Ladder Diagram code against the original Stateflow chart.

Input Arguments

### chartPath — Full path name of the Stateflow chart
character vector

Full path name of the Stateflow chart relative to the top level Simulink model, specified as a character vector. To obtain the full path, select the Stateflow chart in your model and use the `gcb` function.

The Stateflow chart must have these properties:

- The inputs and outputs to the chart must be Boolean. These inputs and outputs correspond to the input and output terminals of your PLC.
- Each state of the chart must correspond to a chart output.
- The expressions controlling the transition between states must involve only Boolean operations between the inputs.

For instance, in the following chart, `c1`, `c2`, `c3`, and `c4` are Boolean inputs to the model. `A1`, `A2`, `A3`, and `A4` are Boolean outputs from the model.
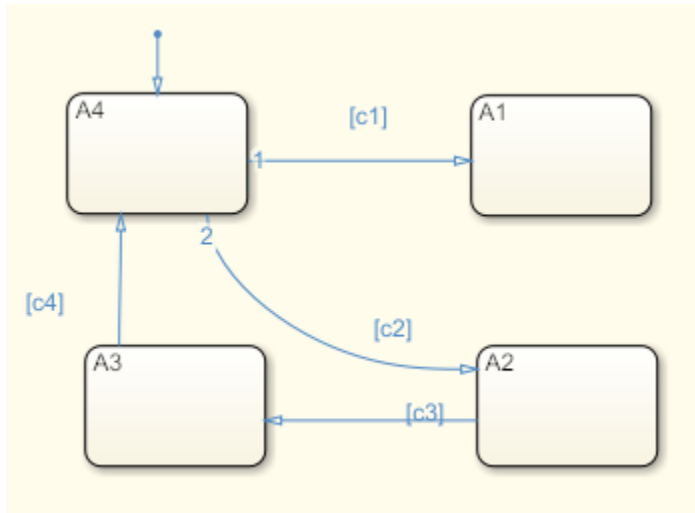
Some advanced Stateflow features are not supported because of inherent restrictions in ladder logic semantics. See the full list of unsupported features.

Example: `gcb, 'ThreeAspectAutoSignal/Subsystem/AutoSignalChart'`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'GenerateTestBench','on','PLC_OutputDir','laddereqn'` generates test bench code in addition to the ladder diagram and places the generated files in the subfolder `laddereqn` of the current working folder.

GenerateTestBench — Generate test bench for validation
`'off'` (default) | `'on'`

Specify whether a test bench must be generated.

You can import the Ladder Diagram code and the test bench together to a target IDE such as CODESYS 3.5. In the IDE, you can validate the ladder diagram against the test bench.

InsertGuardResets — Add reset coils to safeguard against multiple active states
`'off'` (default) | `'on'`

In the ladder diagram, when the output coil corresponding to the active state is turned on, reset coils can be used to force deactivation of other states. The reset coils act as a safeguard against multiple states being simultaneously active. Specify whether the reset coils must be generated.

- If you do not enable this option, each output is a coil that represents a state in the chart.

  The following figure shows an output of the diagram when imported into the CODESYS 3.5 IDE. The output coil represents a state A1 in the chart. When the state is active, the coil receives power.
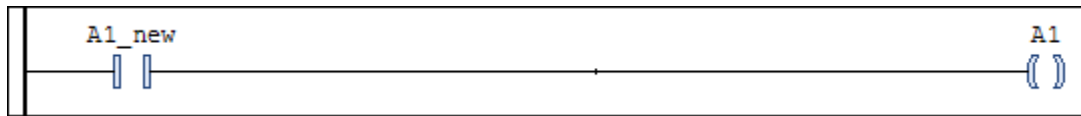
- If you enable this option, each output is a coil that represents a state of the chart. The output is also coupled with reset coils that represent the other states. When a particular state is active, the reset coils force deactivation of the other states.

  The following figure shows an output in the ladder diagram when viewed in the CODESYS 3.5 IDE. The output coil represents a state A1. To avoid multiple states from being simultaneously active, the signal that turns the coil on also turns on the reset coils associated with the other states A2, A3, and A4.
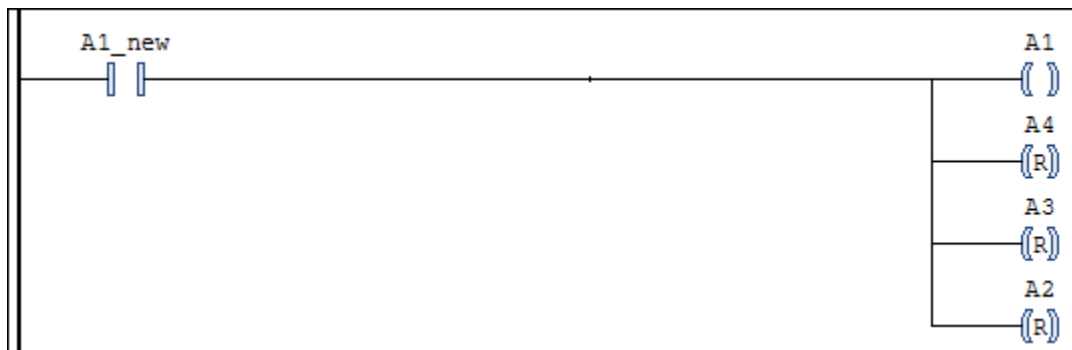
GenerateValidationModel — Generate model with Ladder Diagram code for validation
`'off'` (default) | `'on'`

Specify whether a validation model must be generated. You can use the validation model to compare the generated Ladder Diagram code against the original Stateflow chart.

The validation model has two Subsystem blocks:

- The first block has the original Stateflow Chart.
- The second block has the Ladder Diagram code in a MATLAB Function block.

When you simulate this validation model, for all inputs, the software verifies the output of the second block against the first block. If the output of the second Subsystem block does not match the first, the simulation fails.

PLC_OutputDir — Path relative to current folder where generated files are placed
`'plcsrc'` (default) | character vector

Path relative to the current folder, specified as a character vector. The generated code files are placed in this subfolder. If you do not specify a value, the subfolder `plcsrc` is used.

The output folder must not have the same name as the current folder. For instance, if you do not specify an output folder, `plcsrc` is used. If the current folder is also `plcsrc`, an error occurs.

Example: `'out\plccode'`

Compatibility Considerations

plcgenerateladder will be removed
Not recommended starting in R2019a

plcgenerateladder will be removed in a future release.Use Simulink model for ladder logic instead of Stateflow charts.

See Also

Topics
"Prepare Chart for Simulink PLC Coder Ladder Diagram Code Generation"
"Generate Simulink PLC Coder Ladder Diagram Code from Stateflow Chart"
"Import Ladder Diagram Code to CODESYS 3.5 IDE and Validate Diagram"
"Simulink PLC Coder Ladder Diagram Code Generation"
"Supported IDE Platforms"

Introduced in R2016b

plcimportladder

Import ladder diagram into a Simulink subsystem

Syntax

```
mdlname = plcimportladder(filename)
```

```
[mdlname,mdllib,genbusscript] = plcimportladder(filename,Name,Value)
```

Description

`mdlname = plcimportladder(filename)` generates a Simulink representation of the ladder diagram in the L5X file created using Rockwell Automation IDEs such as RSLogix 5000 and Studio 5000.

`[mdlname,mdllib,genbusscript] = plcimportladder(filename,Name,Value)` generates a Simulink representation of the ladder diagram in the L5X file with properties specified using one or more `Name,Value` pair arguments.

Examples

Import Simple Ladder Diagram into Simulink

The following example demonstrates how to import a simple ladder diagram from an L5X file (`simpleController.L5X`) into the Simulink environment. Create the `simpleController.L5X` by using RSLogix 5000 IDE. The ladder diagram contains contacts and coils representing switches and motor. The following is a snapshot of the ladder diagram structure.
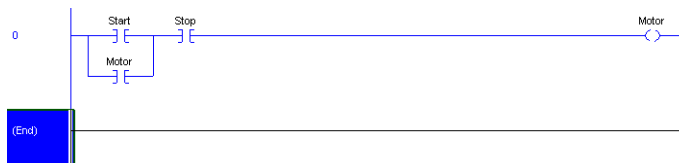
Use the `plcladderimport` function to import the created ladder diagram into Simulink. For this example, the program `Name` of the ladder is `MainProgram` and the `MainRoutineName` is `MainRoutine`.

```
[mdlName,mdlLib,busScript] = plcimportladder('simpleController.L5X','OpenModel','On')
```
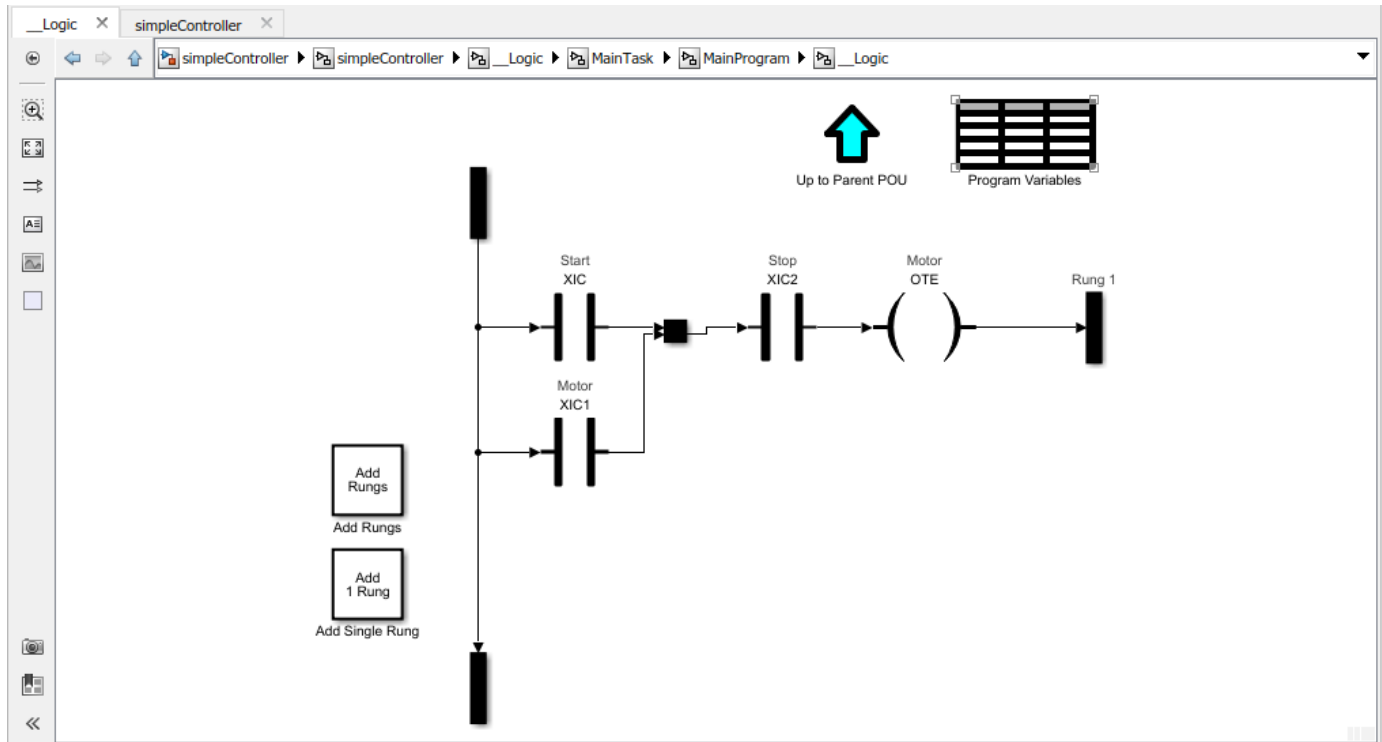
```
mdlName =

    'simpleController'


mdlLib =

    'simpleController_lib'
```

```
busScript =

     []
```

The imported model contains a PLC Controller block named `simpleController`, followed by a Task block named `MainTask` and finally a Ladder Diagram Program block named `MainProgram`. The model imported into Simulink has blocks that implement the functionality of the contacts and coils.

You can traverse up these organizational blocks by double-clicking the `Up to Parent POU` element. The `Program Variables` spreadsheet contains the variables `Start`, `Stop`, and `Motor`. You can use this table to modify attributes of the variables such as `Data Type`, `Initial Value`, and `size`

Input Arguments

`filename` — Name of the ladder file
character vector

Specifies the name of the ladder file to read. Depending on the location of your file, you can either specify the name of the file or provide the full or relative path. name.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `[mdlName,mdlLib,genBusScript] = plcimportladder('sampleLadder.L5X','OpenModel','on','TopAOI','sampleAOI')`

OpenModel — Keep the model open
off (default) | on

At the end of import process, the model is hidden by default. To keep the model open at the end of import, set the value of OpenModel to on.

TopAOI — Top AOI name
character vector

Specify the function blocks that is to be imported. The software imports it into a ladder diagram 'runner' block.

Output Arguments

mdlname — Simulink model name
character array

Specifies the name of the generated Simulink model.

mdllib — Simulink library name
character array

Specifies the name of the generated Simulink model library.

genbusscript — Name of the bus script
character array

Specifies the name of the generated bus script.

See Also

plccleartypes | plcgeneratecode | plcgeneraterunnertb | plcladderlib | plcladderoption | plcloadtypes

Topics
"Supported Features in Ladder Diagram"
"Import L5X Ladder Files into Simulink"
"Modeling and Simulation of Ladder Diagrams in Simulink"
"Generating Ladder Diagram Code from Simulink"
"Verify Generated Ladder Diagram Code"

Introduced in R2018a

plcdispextmodedata

Display the external mode logging data

Syntax

```
plcdispextmodedata(filename)
```

Description

plcdispextmodedata(`filename`) displays logging data information contained in the `filename` MAT-file on the MATLAB command window. The OPC Toolbox™ is required to run the external mode visualization.

Examples

Display Logging Data Information

The following example reads the logging data information stored in `plc_log_data.mat` and displays it on the command window.

```
plcdispextmodedata('plc_log_data.mat')
```

```
Log data:
#1: Y1: LREAL
#2: Y2: LREAL
#3: Y3: LREAL
#4: io_Chart.out: DINT
#5: io_Chart.ChartMode: DINT
#6: io_Chart.State_A: BOOL
#7: io_Chart.State_B: BOOL
#8: io_Chart.State_C: BOOL
#9: io_Chart.State_D: BOOL
#10: io_Chart.is_active_c3_Subsystem: USINT
#11: io_MATLABFunction.y: LREAL
#12: io_MATLABFunction.i: LREAL
#13: io_S1.y: LREAL
#14: io_S1.UnitDelay_DSTATE: LREAL
#15: i1_S1.y: LREAL
#16: i1_S1.UnitDelay_DSTATE: LREAL
```

Input Arguments

`filename` — Name of the MAT-file
character vector

Name of the MAT-file containing the logging information.

See Also
plcgeneratecode | plcrunextmode

Topics
"External Mode Logging"
"Generate Structured Text Code That Has Logging Instrumentation"
"Visualize and Monitor Logging Data by using Simulation Data Inspector"

Introduced in R2018a

plcladderinstructions

Lists ladder instructions identified by Simulink PLC Coder

Syntax

```
out = plcladderinstructions
```

Description

`out = plcladderinstructions` returns the ladder instructions identified by Simulink PLC Coder. You can use these instructions to check if a customized, user-defined instruction has been identified by Simulink PLC Coder.

Examples

Verify Simulink PLC Coder Identified Instructions

To verify if Simulink PLC Coder has identified your newly created instruction, at the MATLAB command line, enter:

```
plcladderinstructions
```

This command lists the instructions that Simulink PLC Coder can use. The supported instructions displayed in the output now includes your newly created instructions.

Output Arguments

`out` — Ladder instructions identified by Simulink PLC Coder
character array

List of ladder instructions identified by Simulink PLC Coder.

See Also
Custom Instruction | `plcimportladder` | `plcladderlib`

Topics
"Create Custom Instruction in PLC Ladder Diagram Models"
"Import L5X Ladder Files into Simulink"
"Modeling and Simulation of Ladder Diagrams in Simulink"
"Generating Ladder Diagram Code from Simulink"

Introduced in R2020a

plcrunextmode

Run external mode visualization

Syntax

```
plcrunextmode(opc_host,target_ide,mdl_name,log_file)
plcrunextmode( ___ ,idx_list)
plcrunextmode( ___ ,name_list)
```

Description

plcrunextmode(opc_host,target_ide,mdl_name,log_file) runs external mode visualization using the settings specified in the arguments. All the logged signals are displayed in the Simulation Data Inspector.

plcrunextmode(___ ,idx_list) runs external mode visualization and displays only the logged signals identified by the indices in the idx_list .
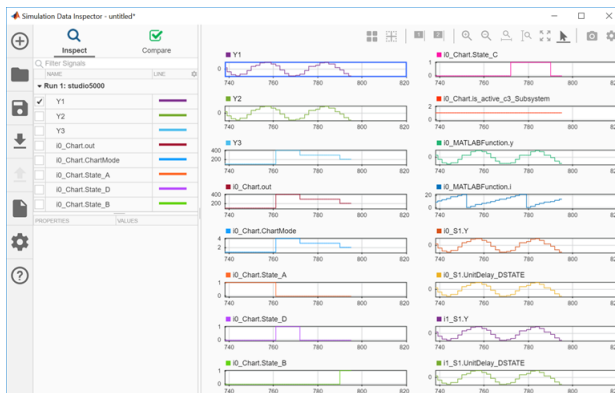
plcrunextmode(___ ,name_list) runs external mode visualization and displays only the logged signals identified by the names in the name_list.

Examples

Visualize Logging Data

The following example uses plcrunextmode to connect to an OPC server and stream log data in to Simulation Data Inspector.

```
plcrunextmode ('localhost', 'studio5000', 'ext_demo1', 'plc_log_data.mat');
```


Input Arguments

opc_host — Host address
character vector

Host address of the OPC server.

Example: `'localhost'`

`target_ide` — Target IDE string
character vector

Specifies the name of the PLC target IDE.

Example: `'studio5000'`

`mdl_name` — Simulink model name
character vector

Specifies the Simulink model for which the code was generated with logging instrumentation.

Example: `'extmode_demo'`

`log_file` — Logging data MAT-file
character vector

Full file path of the logging data MAT-file.

Example: `'C:\plc_log_data.mat'`

`idx_list` — Index list of logged data
integer vector

The index vector specifying the indices of the logged data signals to display. This argument is optional.

Example: `[1 2 3]`

`name_list` — Name list of the logged data
vector

The name vector specifying the names of the logged data signals to display. This argument is optional.

Example: `{'Y1', 'Y2', 'i0_S1.Y'}`

See Also
`plcdispextmodedata` | `plcgeneratecode`

Topics
"External Mode Logging"
"Generate Structured Text Code That Has Logging Instrumentation"
"Visualize and Monitor Logging Data by using Simulation Data Inspector"

Introduced in R2018a

plcladderlib

Open the Simulink PLC Coder Ladder Library
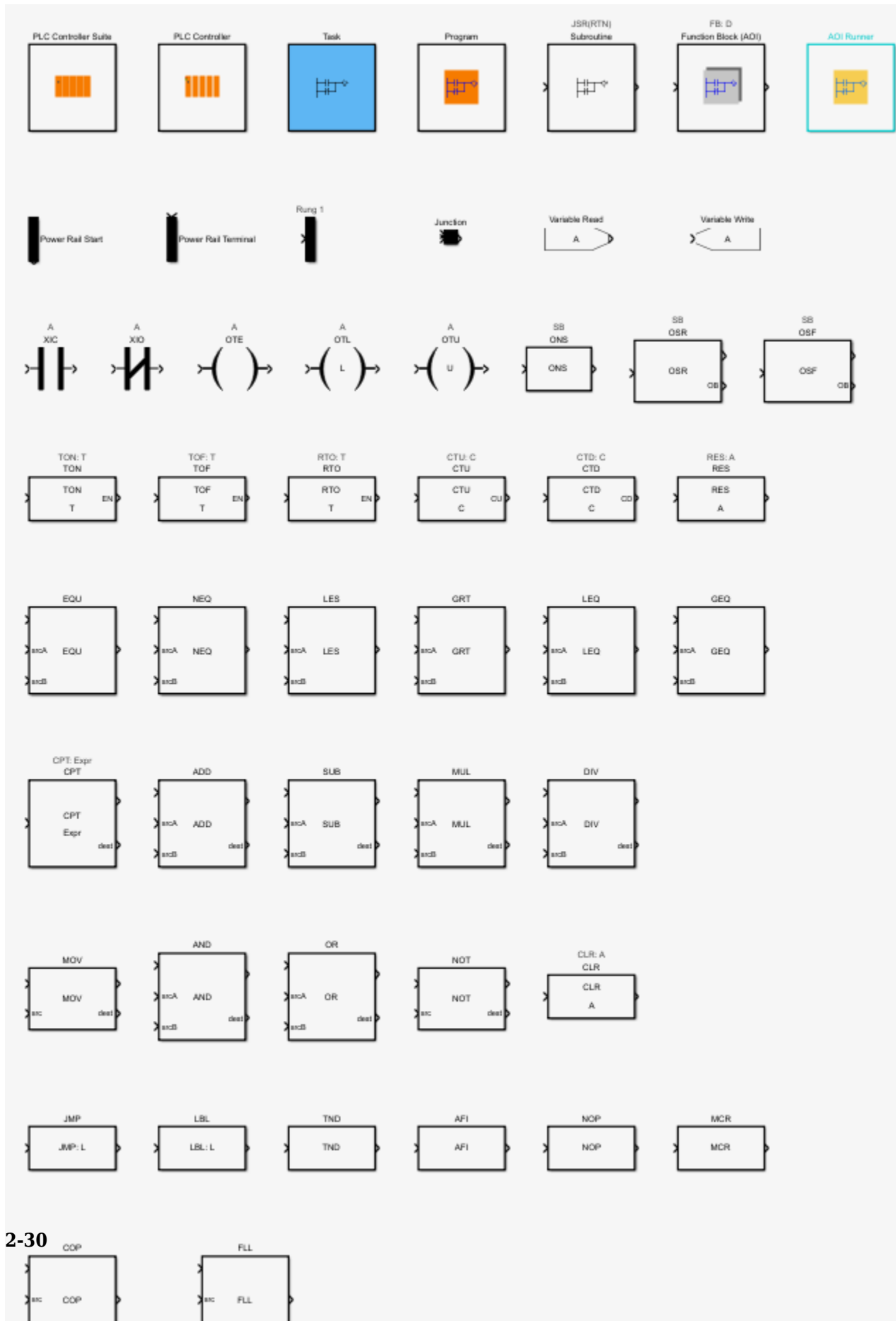
Syntax

```
plcladderlib
```

Description

`plcladderlib` opens the Simulink PLC Coder Ladder Library.

Examples

Open the Ladder Library

```
plcladderlib
```

PLC Controller Suite

PLC Controller

Task

Program

JSR(RTN)
Subroutine

FB: D
Function Block (AOI)

AOI Runner

Power Rail Start

Power Rail Terminal

Rung 1

Junction

Variable Read
A

Variable Write
A

A
XIC

A
XIO

A
OTE

A
OTL
L

A
OTU
U

SB
ONS
ONS

SB
OSR
OSR
OB

SB
OSF
OSF
OB

TON: T
TON
TON
T
EN

TOF: T
TOF
TOF
T
EN

RTO: T
RTO
RTO
T
EN

CTU: C
CTU
CTU
C
CU

CTD: C
CTD
CTD
C
CD

RES: A
RES
RES
A

EQU
srcA EQU
srcB

NEQ
srcA NEQ
srcB

LES
srcA LES
srcB

GRT
srcA GRT
srcB

LEQ
srcA LEQ
srcB

GEQ
srcA GEQ
srcB

CPT: Expr
CPT
CPT
Expr
dest

ADD
srcA ADD
srcB dest

SUB
srcA SUB
srcB dest

MUL
srcA MUL
srcB dest

DIV
srcA DIV
srcB dest

MOV
MOV
src dest

AND
srcA AND
srcB dest

OR
srcA OR
srcB dest

NOT
NOT
src dest

CLR: A
CLR
CLR
A

JMP
JMP: L

LBL
LBL: L

TND
TND

AFI
AFI

NOP
NOP

MCR
MCR

COP
src COP

FLL
src FLL

`plcladderoption` | `plcloadtypes`

Topics
"Supported Features in Ladder Diagram"
"Import L5X Ladder Files into Simulink"
"Modeling and Simulation of Ladder Diagrams in Simulink"
"Generating Ladder Diagram Code from Simulink"
"Verify Generated Ladder Diagram Code"

Introduced in R2019a

plcloadtypes

Load the data types for Simulink PLC Coder ladder models

Syntax

```
plcloadtypes
```

Description

`plcloadtypes` loads the data types used by PLC Ladder models into the MATLAB workspace.

Examples

Load Data types for Ladder diagram model

The following example demonstrates how to load the data types associated with a Ladder model in Simulink.

Use the `plcloadtypes` function to load the data types associated with the Simulink Ladder Diagram model.

```
plcloadtypes
```

The data types are loaded into the MATLAB workspace.

See Also

`plccleartypes` | `plcgeneratecode` | `plcgeneraterunnertb` | `plcimportladder` | `plcladderlib` | `plcladderoption`

Topics
"Supported Features in Ladder Diagram"
"Import L5X Ladder Files into Simulink"
"Modeling and Simulation of Ladder Diagrams in Simulink"
"Generating Ladder Diagram Code from Simulink"
"Verify Generated Ladder Diagram Code"

Introduced in R2019a

plccleartypes

Clear the data types associated with the Simulink PLC Coder ladder models from the workspace

Syntax

```
plccleartypes
```

Description

`plccleartypes` clears the ladder data types from the MATLAB workspace.

Examples

Clear data types from a Ladder Diagram model

The following example demonstrates how to clear the data types associated with a Ladder model in Simulink.

Use the `plccleartypes` function to clear the data types associated with the Simulink Ladder Diagram model.

```
plccleartypes
```

The data types are cleared from the MATLAB workspace.

See Also

`plcgeneratecode` | `plcgeneraterunnertb` | `plcimportladder` | `plcladderlib` | `plcladderoption` | `plcloadtypes`

Topics
"Supported Features in Ladder Diagram"
"Import L5X Ladder Files into Simulink"
"Modeling and Simulation of Ladder Diagrams in Simulink"
"Generating Ladder Diagram Code from Simulink"
"Verify Generated Ladder Diagram Code"

Introduced in R2019a

plcladderoption

Get or set parameter values associated with Ladder Diagram models

Syntax

```
currentState = plcladderoption(mdlname,Name,Value)
```

Description

`currentState = plcladderoption(mdlname,Name,Value)` sets the parameter to the specified value for the Simulink ladder diagram model. Open or load the Simulink ladder diagram model first. If the `Value` argument is not specified this function returns the value of the specified parameter for the ladder diagram model.

Examples

Set the parameter value for Ladder diagram model

The following example demonstrates how to import a simple ladder diagram from an L5X file (`simpleController.L5X`) into the Simulink environment and set ladder options. The ladder L5X file was created using RSLogix 5000 IDE and contains contacts and coils representing switches and motor. The following is a snapshot of the ladder structure.
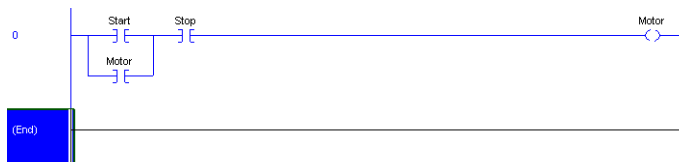
Use the `plcladderimport` function to import the ladder into Simulink. For this example, the program `Name` of the ladder is `MainProgram` and the `MainRoutineName` is `MainRoutine`.

```
[mdlName,mdlLib,busScript] = plcimportladder('simpleController.L5X','OpenModel','On')
```

The imported model contains a PLC Controller block named `simpleController`, followed by a Task block named `MainTask` and finally a Ladder Diagram Program block named `MainProgram`. The model imported into Simulink has blocks that implement the functionality of the contacts and coils.

Use the `plcladderoption` function to enable `FastSim`.

```
currentState = plcladderoption('simpleController','FastSim','on');
```

Input Arguments

`mdlname` — Simulink model name
character array

Specifies the name of the generated Simulink model.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `plcladderoption('simpleController','Animation','on')`

`FastSim` — Enable animation and data display
off (default) | on

When `'on'`, this option disables all animation and data display. If `'FastSim'` is set to `'on'`, then any change made to `'Animation'` and `'DataDisplay'` parameter would not change simulation behavior.

`Animation` — Animation of ladder rungs
on (default) | off

This option enables/disables animation of Ladder Diagram rungs, when `'FastSim'` is in `'off'` state

DataDisplay — Data Display
off (default) | on

This option enables/disables data display of Ladder Diagram rungs, when `'FastSim'` is in `'off'` state

SLDV — Simulink Design Verifier
off (default) | on

This option enables/disables using Simulink Design Verifier features with Ladder Diagrams. Set `'SLDV'` parameter to `'on'` before SLDV analysis. SLDV analysis could be still done without using `'SLDV'` option, by setting `'FastSim'` to `'on'` and `'Prescan'` to `'off'`.

Prescan — Prescan
on (default) | off

This option enables/disables `Prescan` for instructions and AOI blocks explicitly for Simulink Design Verifier analysis.

Output Arguments

currentState — The value of the specified parameter name for the ladder diagram model
off | on

The value of the specified parameter name for the ladder diagram model.

See Also
plccleartypes | plcgeneratecode | plcgeneraterunnertb | plcimportladder | plcladderlib | plcloadtypes

Topics
"Supported Features in Ladder Diagram"
"Import L5X Ladder Files into Simulink"
"Modeling and Simulation of Ladder Diagrams in Simulink"
"Generating Ladder Diagram Code from Simulink"
"Verify Generated Ladder Diagram Code"

Introduced in R2019a

plcgeneraterunnertb

Generate L5X test bench code for specified AOI Runner block and AOI name

Syntax

```
TbCode = plcgeneraterunnertb (runnerBlk)
```

Description

`TbCode = plcgeneraterunnertb (runnerBlk)` returns the generated test bench code for the AOI runner block of the ladder diagram model. Open or load the Simulink Ladder Diagram model first.

Examples

Generate Test bench for AOI runner block

The following example demonstrates how to import a simple ladder diagram from an L5X file (`simpleXIC.L5X`) into the Simulink environment and generate test bench code for it. The ladder L5X file was created using RSLogix 5000 IDE and contains an AOI named `simpleXIC` with contact and coil representing a switch and a light. The following is a snapshot of the ladder structure.
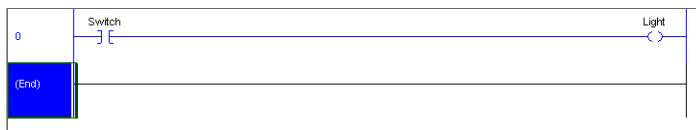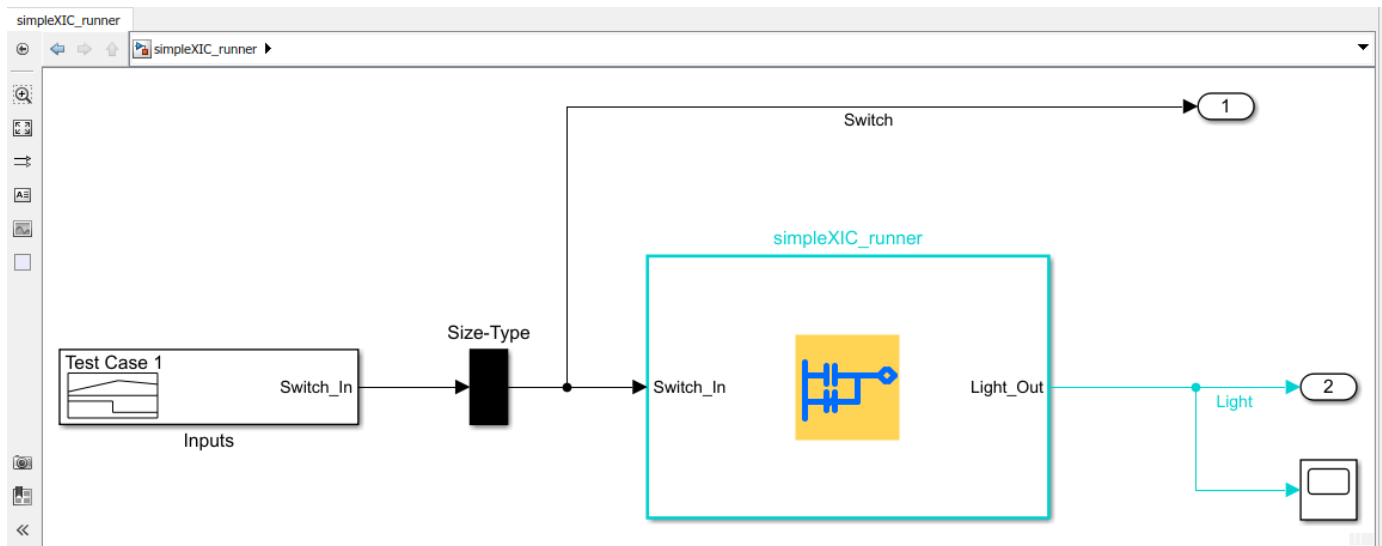

Use the `plcladderimport` function to import the ladder into Simulink.

```
[mdlName,mdlLib,busScript] = plcimportladder('simpleXIC.L5X',...
'OpenModel','On','TopAOI','simpleXIC')
```
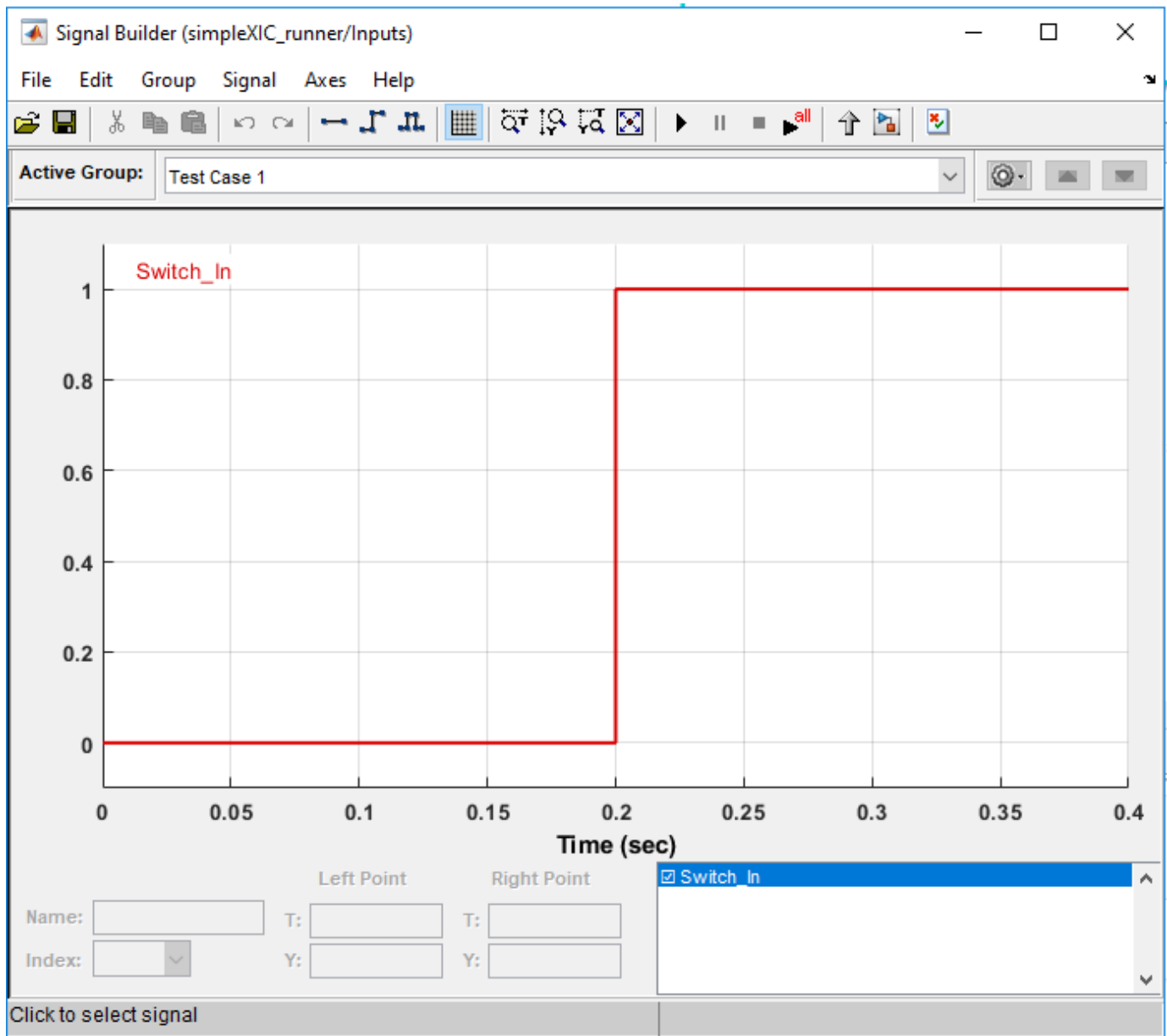
The imported model contains an AOI Runner block named `simpleXIC_runner`, followed by a Ladder Diagram Function (AOI) block named `simpleXIC`.
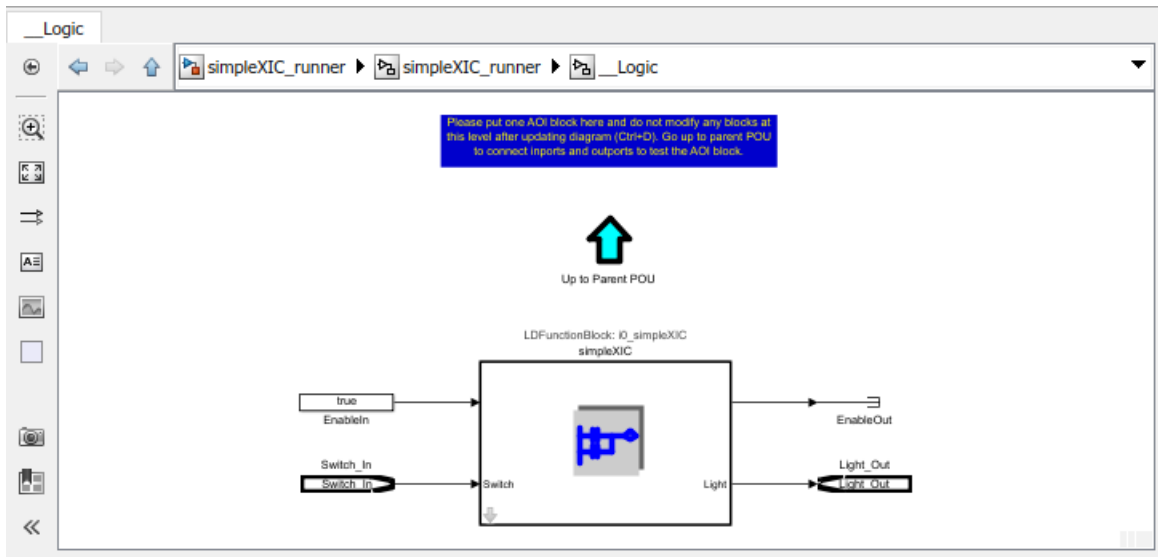
Add Signal Builder input block, Scope and output ports as shown.

Modify the Signal Builder input to mimic a switch operation as shown.

Generate test-bench for the Ladder Diagram model.

```
Tbcode = plcgeneraterunnertb('simpleXIC_runner/simpleXIC_runner')

Tbcode =

  1×1 cell array

    {'C:\runnerTB\simpleXIC_runner.L5X'}
```

If the test-bench code generation is successful, a test-bench file `simpleXIC_runner.L5X` is created. The generated AOI test bench file can be verified on Rockwell Automation IDE.

Note Test-bench generation for ladder diagram models containing timer blocks such as TON, TOF and RTO fails. To generate test-bench code for these models, modify the ladder diagram structure while maintaining the logic.

Input Arguments

runnerBlk — AOI runner block name
character vector

AOI runner block name specified as character vector.

Output Arguments

TbCode — Test-bench Code
character vector

Generated test-bench file name specified as character vector.

See Also
plccleartypes | plcgeneratecode | plcgeneraterunnertb | plcimportladder | plcladderlib | plcladderoption | plcloadtypes

Topics
"Supported Features in Ladder Diagram"
"Import L5X Ladder Files into Simulink"
"Modeling and Simulation of Ladder Diagrams in Simulink"
"Generating Ladder Diagram Code from Simulink"
"Verify Generated Ladder Diagram Code"

Introduced in R2019a