

# Adaptive programming

## Application Guide

The screenshot displays the Adaptive Programming software interface. At the top, there is a window title bar for 'Adaptive Programming {0}{19}' and a toolbar with 'Download to drive' and 'On PC' buttons. Below this is a 'Program tools' panel with icons for Undo, Redo, Open..., Save..., Run program, and Restore. The 'Functional Blocks' panel shows a row of six blocks: Add, Subtract, Multiply, Divide, Abs, and Filter. Each block has input ports (In1, In2 for arithmetic; In for Abs and Filter) and a yellow 'D' icon. Below the Functional Blocks panel is a navigation bar with 'Arithmetic blocks (8)', 'Logical blocks (7)', and 'Selection blocks (7)'. The main workspace shows a selected 'Add' block with inputs 'In1' and 'In2', and a '1' label. On the right, an 'Outputs' panel lists: I/O, Start control, Speed control, Frequency control, Torque control, and Limitations. The 'Inputs' panel on the left lists: Constants, I/O, Actual values, Status, and Data storage.



# Adaptive programming

## Application Guide

Table of contents





# Table of contents

---

## 1 Introduction to the guide

Contents of this chapter .....	11
Applicability .....	11
Compatibility .....	11
Safety instructions .....	11
Target audience .....	12
Related manuals .....	12

## 2 Adaptive programming

Contents of this chapter .....	13
Overview of Adaptive programming .....	13
Base program .....	14
Sequence program .....	14
Connecting the Adaptive program to a drive control program .....	14
Enabling and running the Adaptive program .....	14
Execution of the Adaptive program .....	15
Creating a backup/restore .....	15

## 3 Using PC tool interface

Contents of this chapter .....	17
Adaptive programming user interface .....	17
Base and sequence programs .....	18
Program tools .....	19
Functional blocks .....	19
Inputs .....	19
Editing the input labels .....	19
Outputs .....	20
Sequence states .....	21
State transition .....	21

## 3 Creating an Adaptive program

Contents of this chapter .....	23
Creating a base program .....	23
Creating a sequence program .....	25
Downloading the adaptive program .....	27

## 3 Program elements

Contents of this chapter .....	31
Inputs .....	32
Constants .....	32
Parameters .....	32
I/O .....	32
Actual values .....	33
Status .....	33

---



Data storage .....	34
System outputs .....	35
Parameters .....	35
I/O .....	35
Start control .....	36
Speed control .....	36
Frequency control .....	37
Torque control .....	37
Limitations .....	38
Events .....	38
Data storage .....	39
Process PID .....	39
Function block specifications .....	40
Abs .....	40
Output: .....	40
Input: 1 .....	40
Block function .....	40
Exceptional cases .....	40
Add .....	41
Output: .....	41
Inputs: 2-8 .....	41
Default inputs: 2 .....	41
Block function .....	41
Exceptional cases .....	41
AND .....	42
Output .....	42
Inputs: 2-8 .....	42
Default inputs: 2 .....	42
Block function .....	42
Exceptional cases .....	42
Bit get .....	43
Output .....	43
Inputs: 2-9 .....	43
Block function .....	43
Exceptional cases .....	43
Bitwise AND .....	45
Output .....	45
Inputs: 2-8 .....	45
Block function .....	45
Exceptional cases .....	45
Bitwise OR .....	46
Output .....	46
Inputs: 2-8 .....	46
Block function .....	46
Exceptional cases .....	46
Bitwise XOR .....	47
Output .....	47
Inputs: 2 .....	47
Block function .....	47
Exceptional cases .....	47
Divide .....	48
Output: .....	48



Inputs: 2 .....	48
Block function .....	48
Exceptional cases .....	48
Equal .....	49
Output .....	49
Inputs: 2 .....	49
Block function .....	49
Exceptional cases .....	49
Filter .....	50
Output: .....	50
Inputs: 2 .....	50
Block function .....	50
Where: .....	50
Exceptional cases .....	50
Greater than .....	51
Output .....	51
Inputs: 3 .....	51
Block function .....	51
First: .....	51
Second (if first is not true): .....	51
Third (if neither are true): .....	51
Exceptional cases .....	51
Less than .....	52
Output .....	52
Inputs: 3 .....	52
Block function .....	52
First .....	52
Second (if first isn't true) .....	52
Third (if neither are true) .....	52
Exceptional cases .....	52
Limit .....	53
Output: .....	53
Inputs: 3 .....	53
Block function .....	53
Exceptional cases .....	53
Max .....	54
Output: .....	54
Inputs: 2-8 .....	54
Default inputs: 2 .....	54
Block function .....	54
Exceptional cases .....	54
Min .....	55
Output: .....	55
Inputs: 2-8 .....	55
Default inputs: 2 .....	55
Block function .....	55
Exceptional cases .....	55
Multiply .....	56
Output: .....	56
Inputs: 2-8 .....	56
Default inputs: 2 .....	56
Block function .....	56



Exceptional cases .....	56
NOT .....	57
Output .....	57
Input: 1 .....	57
Block function .....	57
Exceptional cases .....	57
OR .....	58
Output .....	58
Inputs: 2-8 .....	58
Default inputs: 2 .....	58
Block function .....	58
Exceptional cases .....	58
PI .....	59
Output: .....	59
Inputs: 8 .....	59
Block function .....	60
Exceptional cases .....	60
Ramp .....	61
Output .....	61
Inputs: 7 .....	61
Block function .....	61
Exceptional cases .....	62
Select boolean .....	63
Output .....	63
Inputs: 3-9 .....	63
Default inputs: 3 .....	63
Block function .....	63
Exceptional cases .....	63
Select value .....	64
Output .....	64
Inputs: 3-9 .....	64
Default inputs: 3 .....	64
Block function .....	64
Exceptional cases .....	64
Set bits 0-7 .....	65
Output .....	65
Inputs: 9 .....	65
Block function .....	66
Exceptional cases .....	66
Set bits 8-15 .....	67
Output .....	67
Inputs: 9 .....	67
Block function .....	68
Exceptional cases .....	68
Square root .....	69
Output .....	69
Inputs: 1 .....	69
Block function .....	69
Exceptional cases .....	69
SR .....	70
Output .....	70
Input: 2 .....	70





Block function .....	70
Exceptional cases .....	70
Subtract .....	71
Output: .....	71
Inputs: 2 .....	71
Block function .....	71
Exceptional cases .....	71
Switch boolean .....	72
Output: .....	72
Inputs: 3-15 .....	72
Default inputs: 3 .....	73
Block function .....	73
Example: .....	73
Exceptional cases .....	73
Switch value .....	74
Output: .....	74
Inputs: 3-15 .....	74
Default inputs: 3 .....	75
Block function .....	75
Example: .....	75
Exceptional cases .....	75
Timer .....	76
Output .....	76
Inputs: 4-10 .....	76
Default inputs: 4 .....	76
Block function .....	76
Exceptional cases .....	78
Trigger down .....	79
Output .....	79
Input: 1 .....	79
Block function .....	79
Exceptional cases .....	79
Trigger up .....	80
Output .....	80
Input: 1 .....	80
Block function .....	80
Exceptional cases .....	80
T_off .....	81
Output .....	81
Inputs: 2 .....	81
Block function .....	81
Exceptional cases .....	81
T_on .....	82
Output .....	82
Inputs: 2 .....	82
Block function .....	82
Exceptional cases .....	82
XOR .....	83
Output .....	83
Inputs: 2 .....	83
Block function .....	83
Exceptional cases .....	83



10 Table of contents

**Further information**



# 1

## Introduction to the guide

---

### Contents of this chapter

This chapter gives general information on the guide.

### Applicability

This guide applies to Adaptive programming - a standard feature available in the Drive Composer PC tool. These version of the tool include the Adaptive programming:

- Drive Composer Pro 1.9 or later
- Drive Composer entry x.x or later.

### Compatibility

Several control programs for drives and units support Adaptive programming. Refer to the appropriate control program firmware manual.

### Safety instructions

Follow all safety instructions delivered with the drive.

- Read the complete safety instructions before you install, commission, or use the drive. The complete safety instructions are delivered with the drive as either part of the Hardware manual, or, in the case of ACS880 multidrives, as a separate document.
  - Read the software function specific warnings and notes before changing the default settings of the function. For each function, the warnings and notes are given in the Firmware Manual in the subsection describing the related user adjustable parameters.
-

## Target audience

This guide is intended for people who design, commission, or operate the drive system.

## Related manuals

You can find manuals on the Internet. See below for the relevant code/link. For more documentation, go to [www.abb.com/drives/documents](http://www.abb.com/drives/documents)

- Firmware manual of the drive or unit with the Adaptive program.
  - Drive Composer user's manual [3AUA0000094606](#)
-



# Adaptive programming

---

## Contents of this chapter

This chapter provides an overview of Adaptive programming and how to use the Adaptive program.

## Overview of Adaptive programming

Adaptive programming is used to customize the operation of a drive in case the drive parameter setting is not sufficient. The Adaptive program is built with standard function blocks included in the drive firmware. The program consists of the following elements:

- A predefined list of input elements to read values from the drive control program..
- A predefined list of outputs to write values to the drive control program..
- A collection of states in which each state has its own block program, including inputs, outputs and state transition elements.

Standard function blocks (for example ADD, AND) are used to create an executable Adaptive program. The maximum size of an Adaptive program is approximately 20 standard function blocks, depending on the block types used and the number of predefined inputs and outputs utilized in the program. Numerical function blocks use floating point numbers in the calculations.

Adaptive program is created in the drive control program using the Drive composer software with which the program can also be downloaded to the drive and started. By default, Adaptive program is started when the drive is powered On, if the program already exists in the drive.

Adaptive programming execution stops when macro or user set is changed.

---

## Base program

The base program is the actual function block program.

## Sequence program

In addition to the Base program, the Adaptive program can also consist of a collection of states, called a Sequence program. When the program is running, there is always one state active and the corresponding program is executed until another state is active. Execution of sequence states always begins from state 1.

The state changes are controlled with state transition elements that can be connected to function block outputs. State transition takes place after the full execution cycle of the program during which the value of any corresponding output becomes true. In case multiple state transitions are true during a single execution cycle, then the one that is connected to the smallest numbered block is triggered. See the example program execution.

Not all drive control programs support the Sequence program. Refer to the drive firmware manual.

See also [Creating a sequence program \(page 25\)](#) and [Downloading the adaptive program \(page 27\)](#).

## Connecting the Adaptive program to a drive control program

Adaptive program is connected to the drive control program through input and outputs. When a Adaptive program uses a drive parameter as an output, it is reserved for the Adaptive program and write-protected otherwise.

Drive provides the available inputs and outputs and sets the parameter values accordingly based on the created program.

When the predefined output (value/bit pointer parameter) is written to from the Adaptive program, the parameter is write protected and it is not changed in the parameter table. The control panel and Drive composer shows text in the pointer parameter to indicate that the parameter is connected to the Adaptive program.

## Enabling and running the Adaptive program

The Adaptive program function can be enabled or disabled with the drive parameter 96.70 Disable Adaptive program.

Adaptive program is in running when all of these conditions are valid:

1. Run program button in the Adaptive programming window is active (pressed).  
Run program button is on the In drive tab.
2. Adaptive program is enabled.
3. The control unit of the drive or unit is powered
4. The control program of the drive or unit is not performing a parameter restore, application macro change, or other large scale parameter change operation.

When Adaptive program is disabled, the situation is similar to a drive without Adaptive program. The following operations are not possible:

---

- Adaptive program cannot be put to running mode when the drive is powered On
- Adaptive program cannot be edited or put to running mode from Drive composer.

## Execution of the Adaptive program

Adaptive program is executed on firmware time level. The parameter 7.30 Adaptive program status shows the status of the Adaptive program. The program can be edited only when the drive is in Stopped state. While editing the program, the Start inhibit is On, so that the drive cannot be started.

**Note:** For time level actual value, refer firmware manual.

The Adaptive program executes the function blocks in numerical order with all blocks on the same time level. This cannot be changed by the user. The user can only do the following tasks:

- build a program using the standard blocks and connections
- change the numbering of the blocks by moving them to different positions
- select the operation mode of the program (run/edit).

If Adaptive program in the drive is not compatible or corrupted, the fault 64A6h Adaptive program is activated. The extension code of the fault explains the detail of the problem with the Adaptive program.

## Creating a backup/restore

Adaptive program can be saved to the backup file and restored. The program starts automatically if the conditions listed in [Enabling and running the Adaptive program \(page 14\)](#) are valid.

---







## Using PC tool interface

---

### Contents of this chapter

This chapter describes the main user interface elements of PC tool for Adaptive programming.

### Adaptive programming user interface

The Adaptive programming user interface is available in the Drive Composer PC tool. From the interface it is possible to:

- Create an Adaptive program
- Edit the Adaptive program
- Download a new/edited Adaptive program to the drive.

The working area can be used either with tab or floating window. The selection between tab and floating window can be made using Drive composer pro View menu. The figure below shows the user interface with tabbed window.

---

## 18 Using PC tool interface

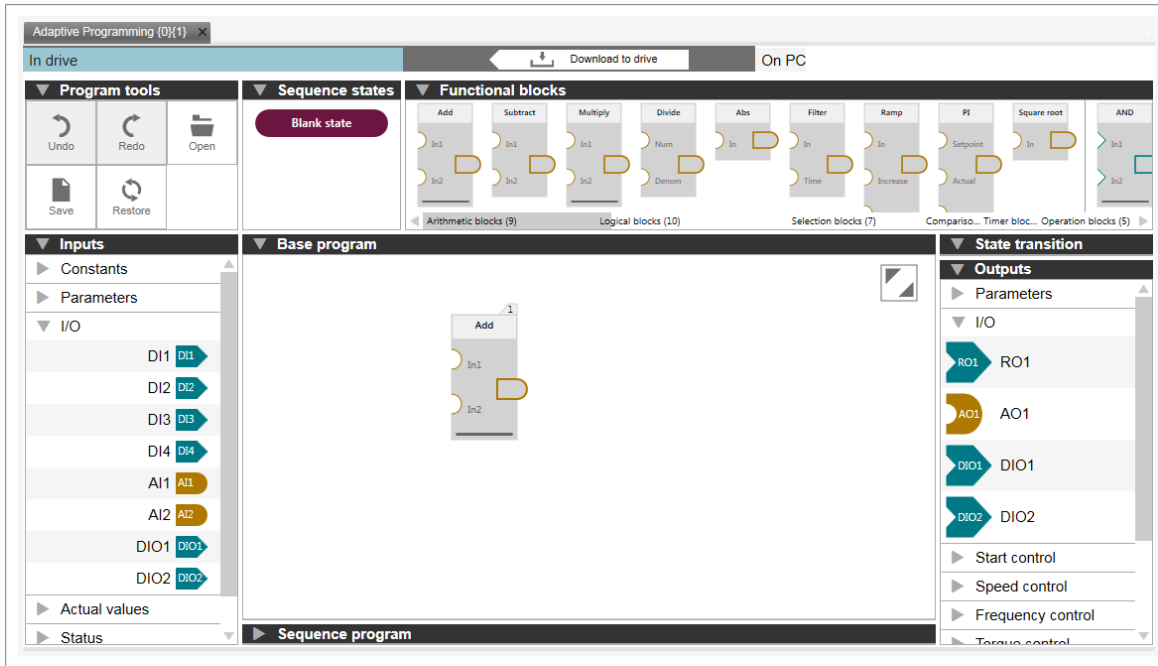


Figure 1. Adaptive programming user interface

### ■ Base and sequence programs

There are separate canvases for creating base and sequence programs. The required canvas can be expanded or collapsed.

- The base program canvas can be used to create a base program with function blocks. The user can drag and drop the desired function blocks to build a base program. See [Creating a base program \(page 23\)](#).
- The sequence program canvas can be used to create a sequence program. The user can drag and drop the desired amount of states to build a sequence program. See [Creating a sequence program \(page 25\)](#).

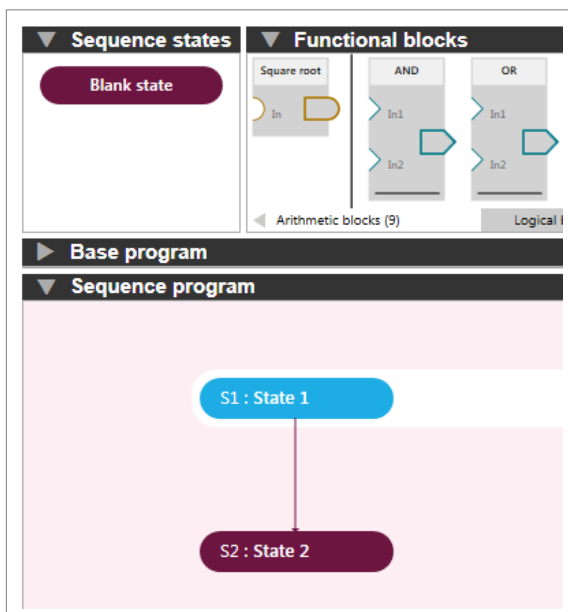


Figure 2. Sequence program user interface

## ■ Program tools

The program tools contains the following options:

- Undo: Erases the last change made and reverts it to an older state
- Redo: Reverses the undo or advances to a more current state
- Open: Opens a program from locally saved file
- Save: Saves the active program to a local file (.dcap format)
- Restore: Restores the default program.

## ■ Functional blocks

Functional blocks of Adaptive programming are grouped into categories and are shown on a horizontal shelf. The scroll bar shows category labels and indicates the current view. The user can drag and drop the required blocks to the canvas.

## ■ Inputs

The inputs are categorized into groups. Note that the available groups and inputs are dependent on the drive type. Typical examples are:

- Constants
- I/O
- Actual values.

The same input can be used multiple times in the same program. Hovering over an input on the shelf highlights every instance of that input on the canvas, so you can easily locate where the input is used in the program.

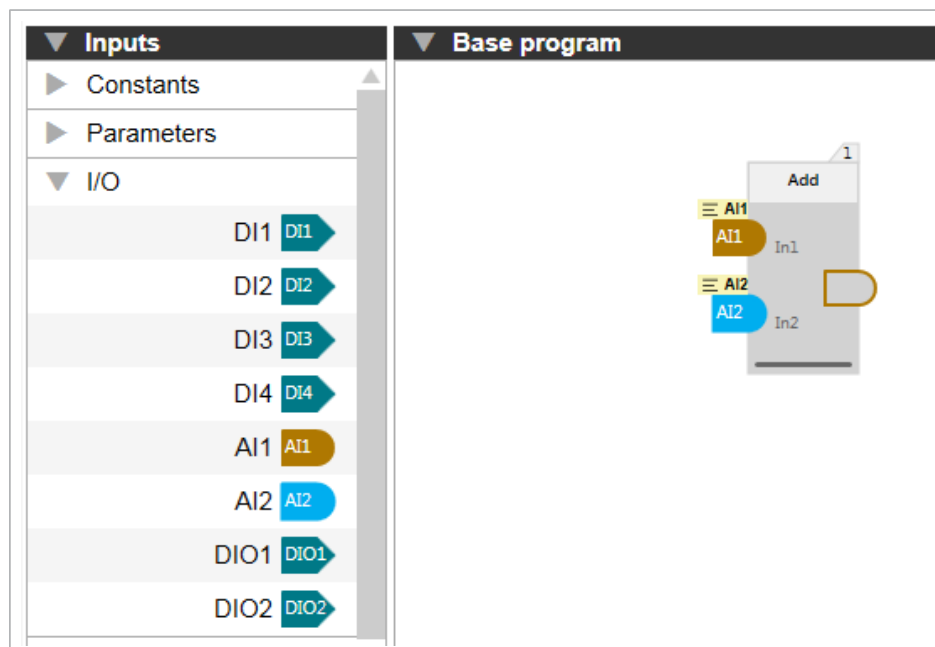



Figure 3. Inputs

### Editing the input labels

You can edit the input labels and add a comment.

1. Click  label in the functional block input.

## 20 Using PC tool interface

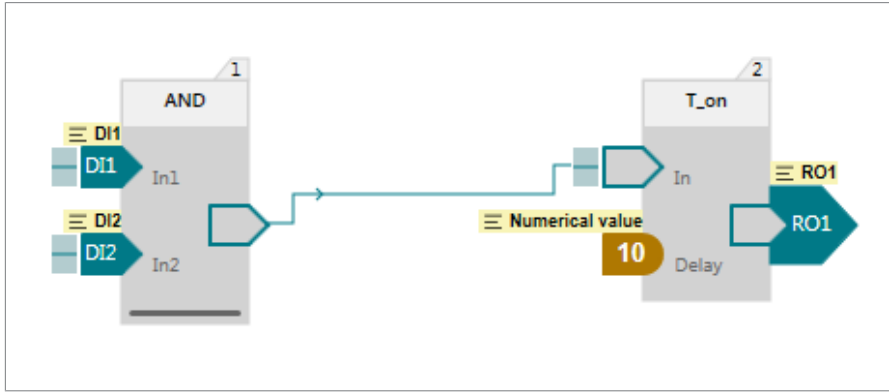


Figure 4. Editing label

2. Edit the label and add the comment as desired.

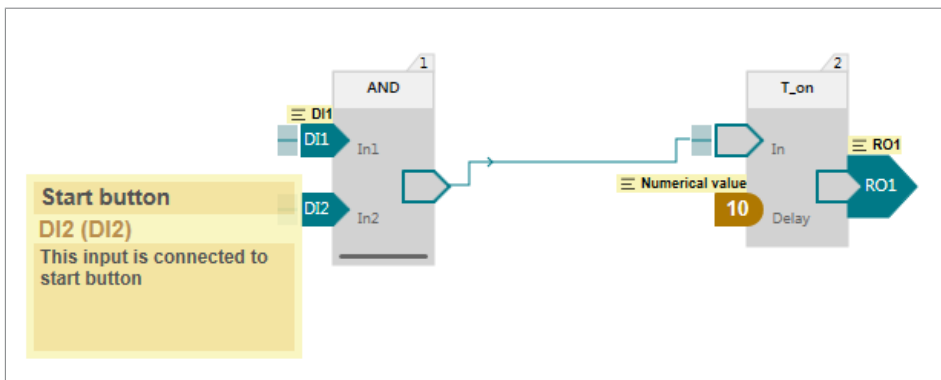


Figure 5. Editing label and comment

### ■ Outputs

The outputs are categorized into groups. Note that the available groups and outputs are dependent on the drive type. Typical examples are:

- Parameters
- I/O
- Start control
- Speed control.

Each output can be used only once in the program. After you drag and drop an output to the canvas, it is faded on the shelf.

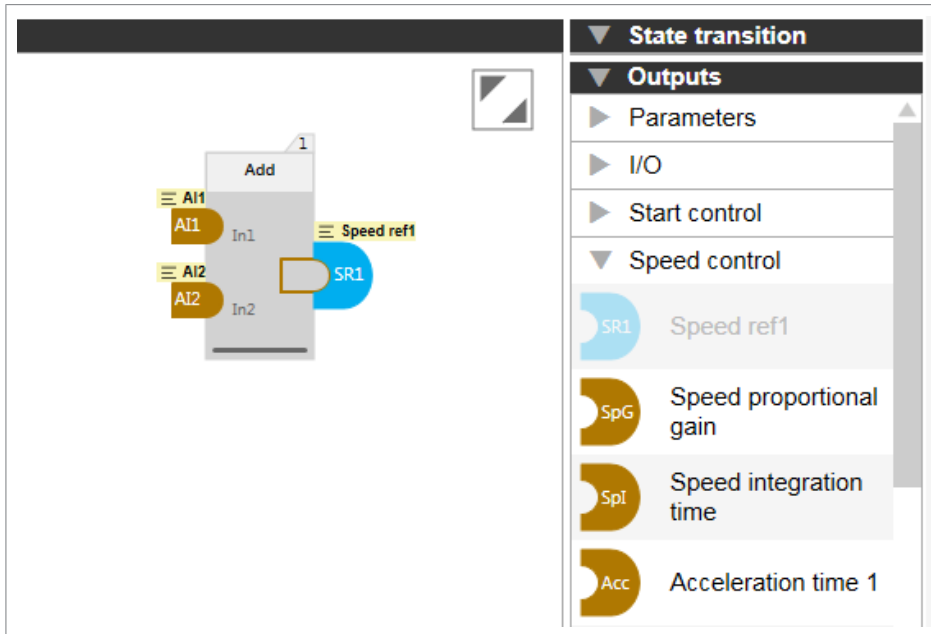


Figure 6. Outputs

For more information on output descriptions, refer firmware to the appropriate firmware manual of the drive or unit.

### ■ Sequence states

The sequence states contains a:

- Blank state: adds a new empty state to the sequence program.  
You can drag-and-drop this empty state any number of times to the sequence program canvas and rename the state in the program.

### ■ State transition

State transition element is used to control the sequence of state transitions when connected to boolean type block outputs. There can be several state transition elements used in a single state.

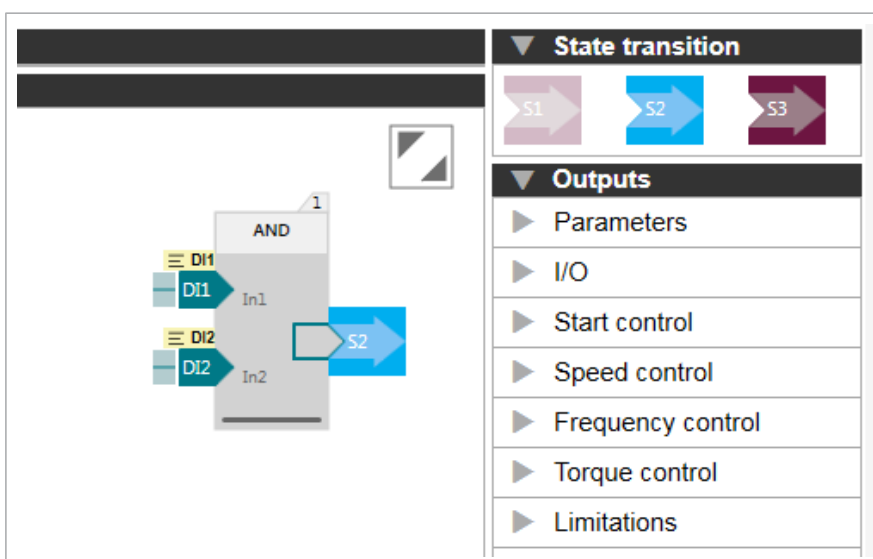


Figure 7. State transition





## Creating an Adaptive program

---

### Contents of this chapter

This chapter describes how to create an Adaptive program and download the program to the drive.

You can do the following:

- Create a base program using function blocks.
- Optionally create a sequence program using states.
- Download the program to the drive.

### Creating a base program

To create a base program using function blocks, proceed as follows:

1. Open the Drive composer PC tool, and make sure that it is connected the drive for which you want to create a base program.
  2. Select ... to open the Adaptive programming interface.
  3. Drag-and-drop the desired function blocks to the base program canvas.
-

## 24 Creating an Adaptive program

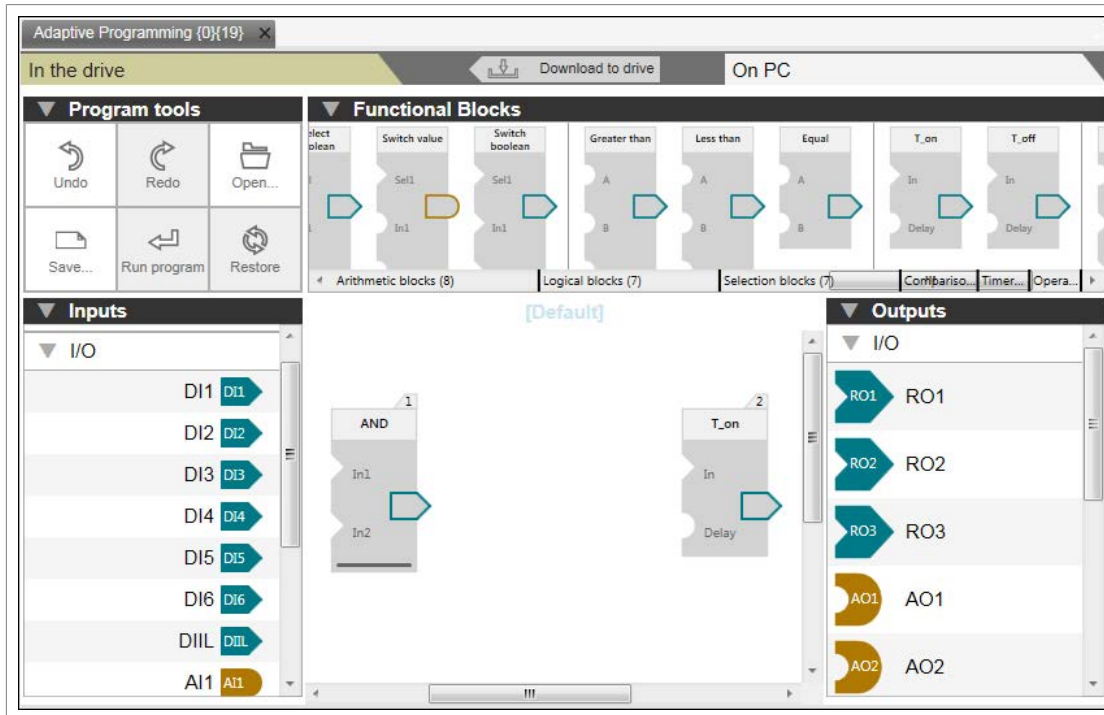


Figure 8. Function block

4. Drag-and-drop the desired input element from the **Inputs** categories to the function block(s). Function block input shape indicates which type of input element it accepts, boolean or numeric. Double-click the input element to select or define the actual input content. For example, select the parameter index for the **Parameter** input element.

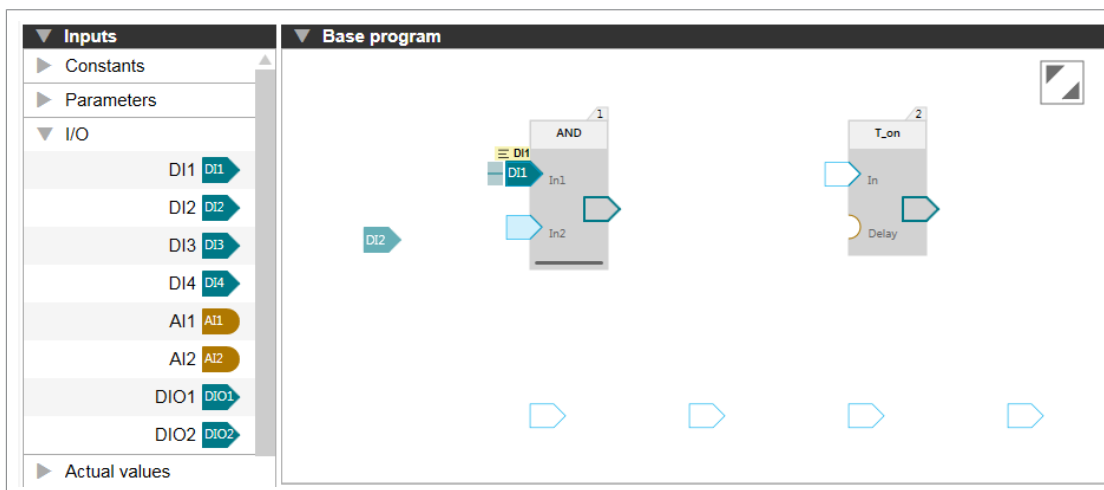


Figure 9. Adding inputs

5. Drag-and-drop the desired connections from the block outputs to other function block(s). Function block output shape indicates which type of output element it accepts, boolean or numeric. Double-click the output element to select or define the actual output content. For example, select the parameter index for the **Parameter** output element.



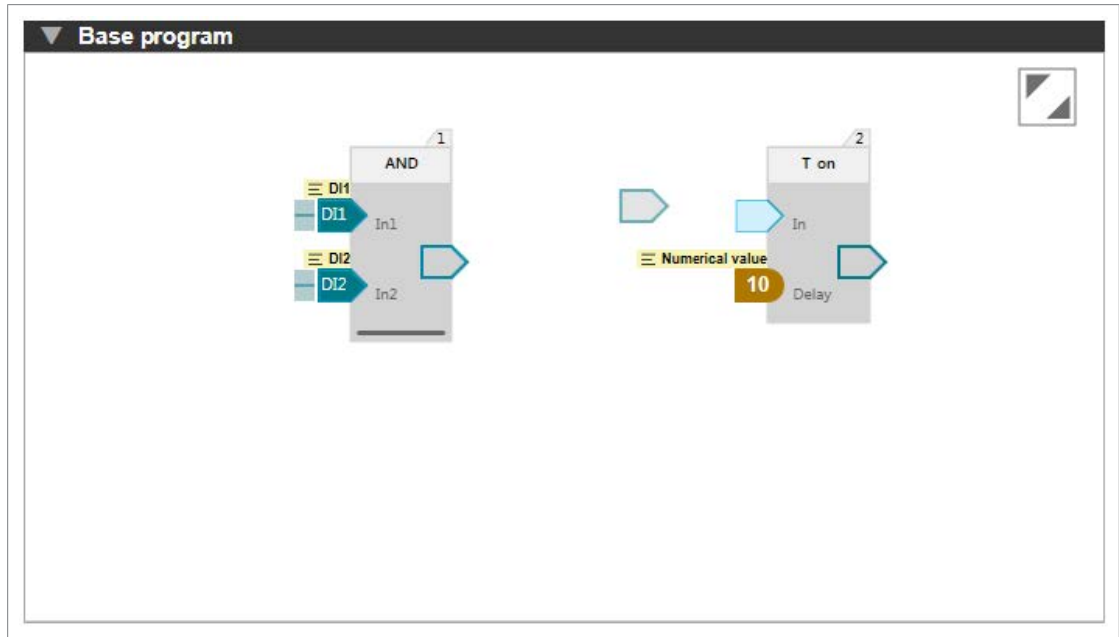


Figure 10. Adding outputs

6. Drag-and-drop the desired output element from the Outputs categories to the function block(s).

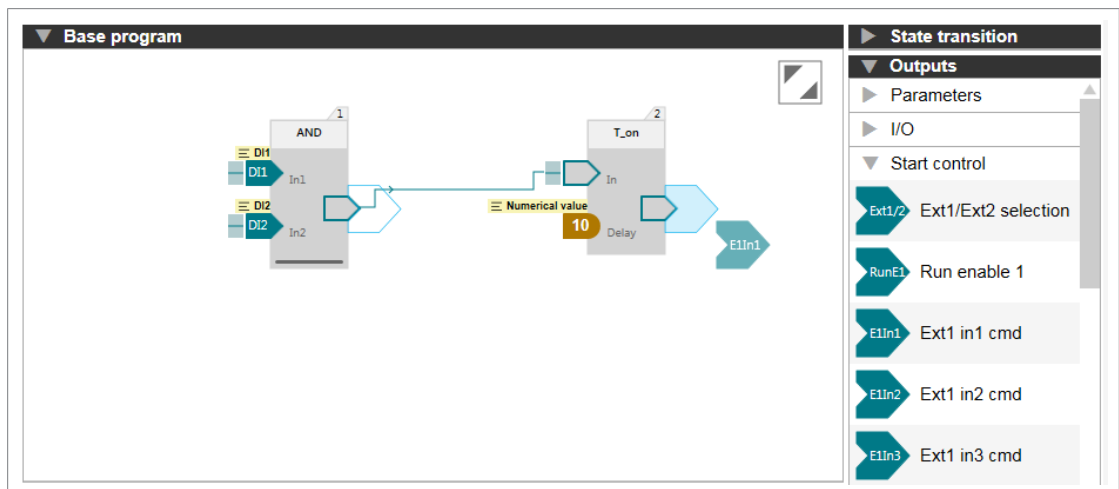


Figure 11. Adding outputs

## Creating a sequence program

To create a sequence program using states, proceed as follows:

1. Open the Drive composer PC tool, and make sure that it is connected the drive for which you want to create an sequence program.
2. Open the Adaptive programming interface.
3. Open the Sequence Program canvas.
4. Drag-and-drop the desired amount of states to the sequence.

## 26 Creating an Adaptive program

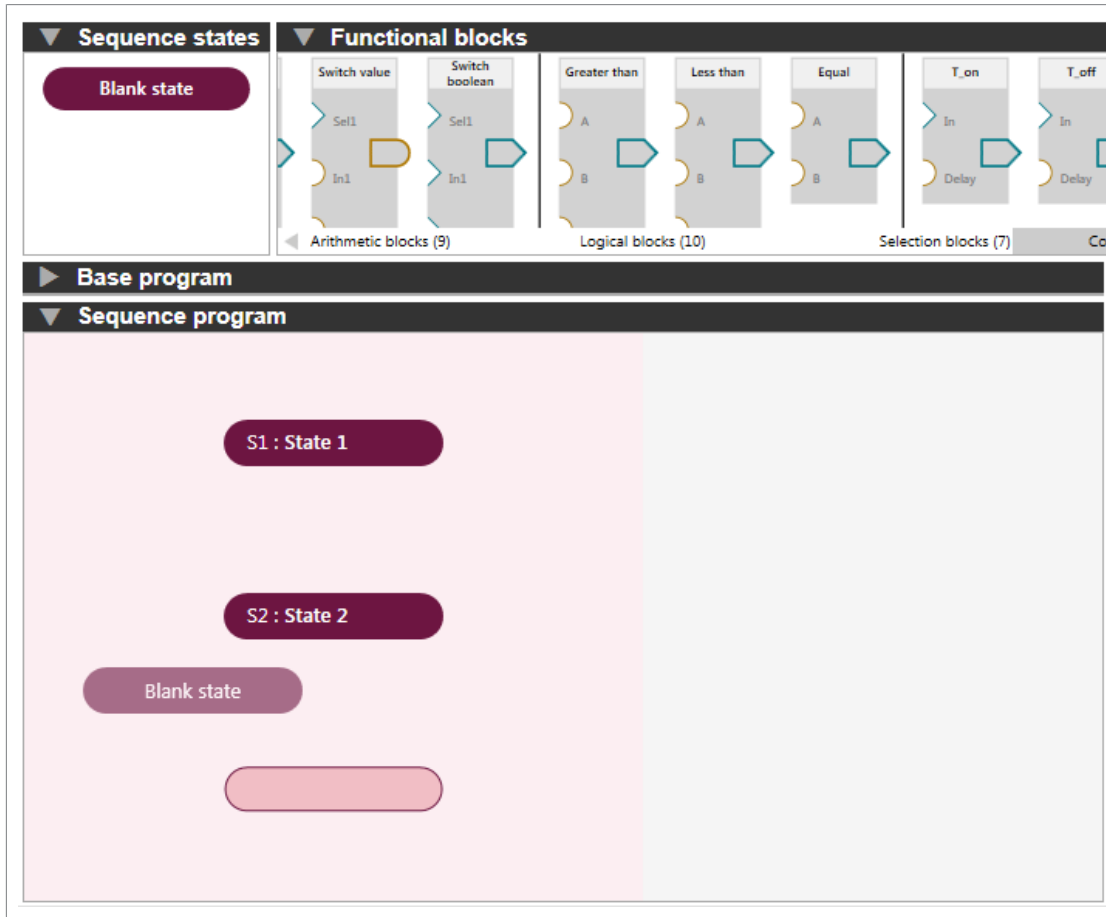


Figure 12. Sequence program states

5. Select the state and create desired block program for each state.

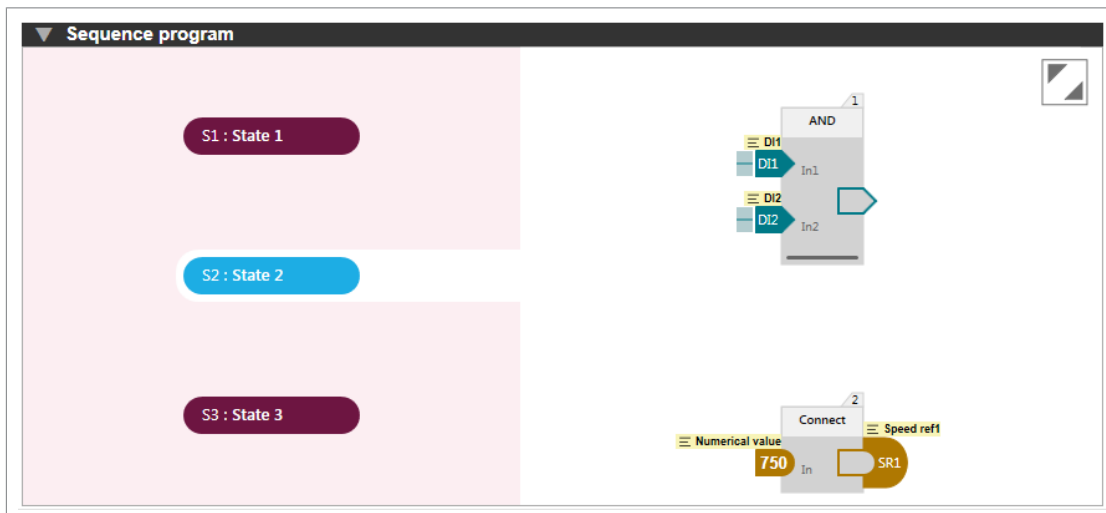


Figure 13. Block program in selected state

6. Drag-and-drop the desired state transitions to each state.

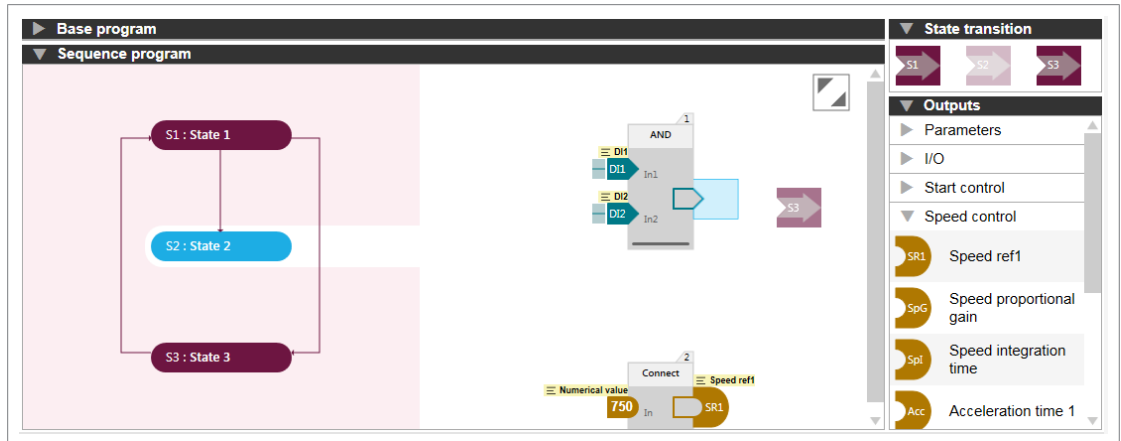


Figure 14. State transitions

## Downloading the adaptive program

After creating a base program and optionally a sequence program, you can download the program to a drive and run the program.

1. Click **Download to drive**.

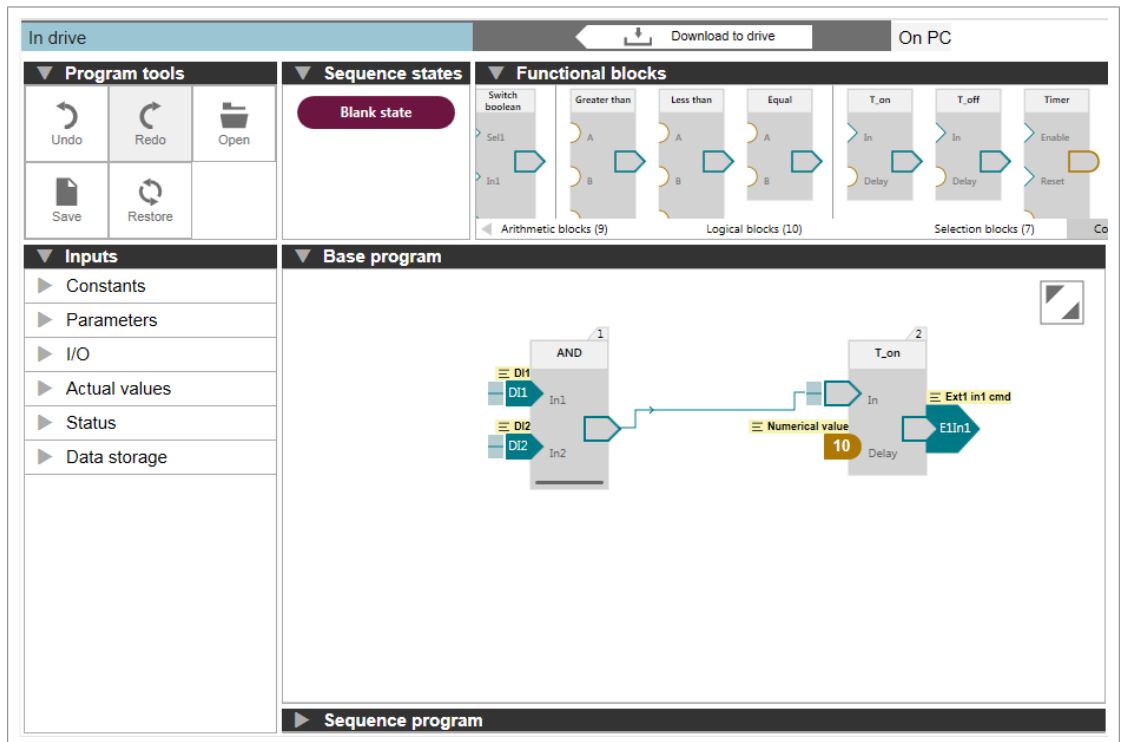


Figure 15. Downloading to drive

The program is downloaded to the drive.

## 28 Creating an Adaptive program

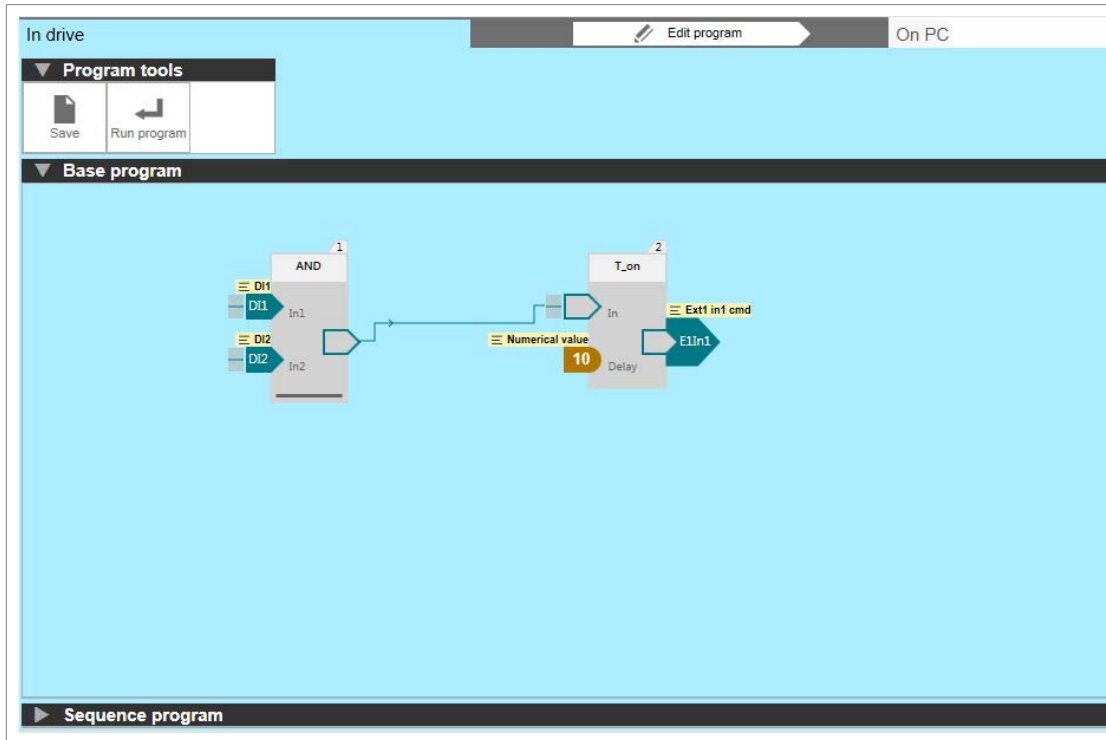


Figure 16. Program downloaded to a drive

2. Go to **In drive** tab. In the Program tools, click **Run program** to start the program.
3. Open the **Sequence program** canvas to view the sequence program.

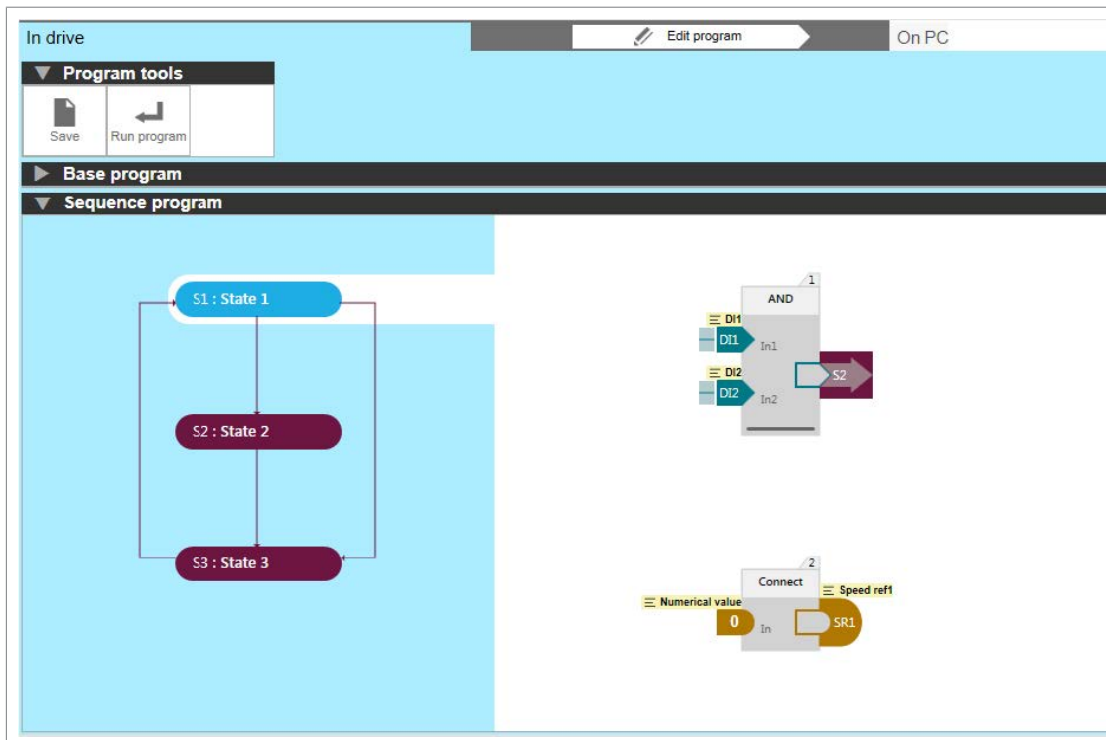


Figure 17. Sequence program

After downloading the program to the drive, you can

- click **Edit program** to stop the program and start editing

or

- click **Save** to save the adaptive program to a local file (.dcap format).



A large, light gray square with rounded corners, centered on the page. Inside the square, the number '3' is written in a large, bold, black font.

## Program elements

---

### Contents of this chapter

This chapter presents the main Adaptive program elements, that is, input and output element listings, and function blocks.

The content of the input/and output element listings vary depending on the control program of the drive or unit. This chapter presents I/O elements for ACS880 primary control program.

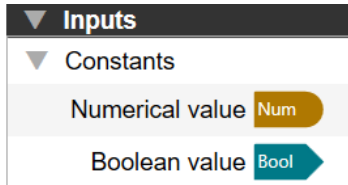
---

## Inputs

### ■ Constants

This input element list contains two elements, **Numerical value** and **Boolean value**:

- Use **Numerical value** to read a constant with a numeric value.
- Use **Boolean value** to read a constant with boolean value.

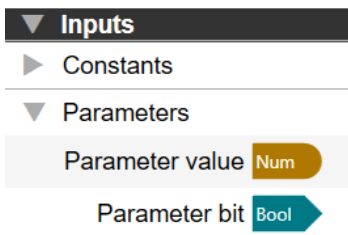


Drag and drop the input element to an input of a function block. Function block input shape indicates which type of input element it accepts, boolean or numeric. Double-click the element to select the constant index

### ■ Parameters

This input element list contains two elements, **Numerical value** and **Boolean value**:

- Use **Numerical value** to read a parameter with a numeric value.
- Use **Boolean value** to read a parameter with boolean value.



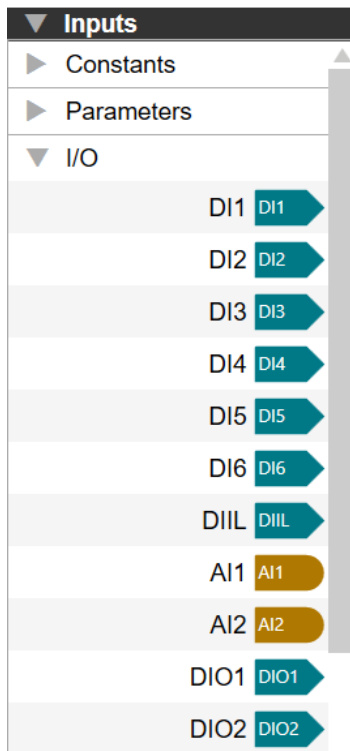
Drag and drop the input element to an input of a function block. Function block input shape indicates which type of input element it accepts, boolean or numeric. Double-click the element to select the parameter index

### ■ I/O

This input element list contains digital input, analog input and digital input/output signals of the drive or unit. For example, **DI1**, etc.

Drag and drop the input element to an input of a function block. Function block input shape indicates which type of input element it accepts, boolean or numeric.

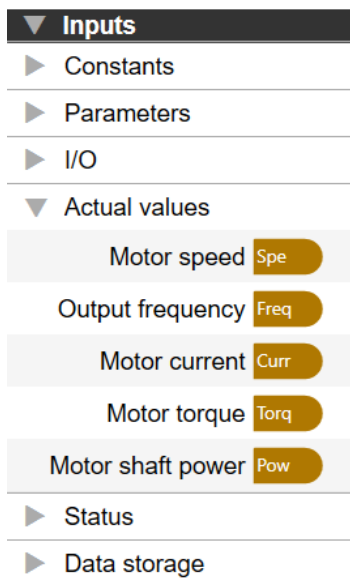




■ **Actual values**

This input element list contains the basic signals for monitoring the operation of the drive or unit. For example, **Motor speed**, etc.

Drag and drop the input element to an input of a function block. Function block input shape indicates which type of input element it accepts, boolean or numeric.



■ **Status**

This input element list contains the basic status signals of the drive or unit. for example, **Enabled**, etc.

## 34 Program elements

▼ Inputs	
▶ Constants	
▶ Parameters	
▶ I/O	
▶ Actual values	
▼ Status	
Enabled	Enab
Inhibited	Inhib
Ready to start	Rea
Tripped	Trip
At setpoint	AtSet
Limiting	Limit
Ext1 active	Ext1
Ext2 active	Ext2

### ■ Data storage

This input element list shows the data storage values of the control program. These are the values of the 32 bit real data storage parameters in the control program. For example, values for parameters 47.01 to 47.08.

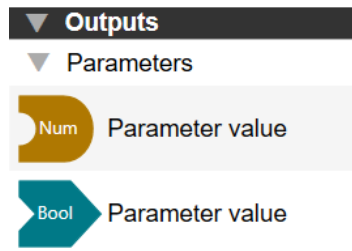
▼ Inputs	
▶ Constants	
▶ Parameters	
▶ I/O	
▶ Actual values	
▶ Status	
▼ Data storage	
Data storage 1 real 32	DS1
Data storage 2 real 32	DS2
Data storage 3 real 32	DS3
Data storage 4 real 32	DS4
Data storage 5 real 32	DS5
Data storage 6 real 32	DS6
Data storage 7 real 32	DS7
Data storage 8 real 32	DS8

## System outputs

### ■ Parameters

This input element list contains two elements, **Numerical value** and **Boolean value**:

- Use **Numerical value** to write a parameter with a numeric value.
- Use **Boolean value** to write a parameter with boolean value.



Drag and drop the output element to an output of a function block. Function block output shape indicates which type of output element it accepts, boolean or numeric. Double-click the element to select the parameter index

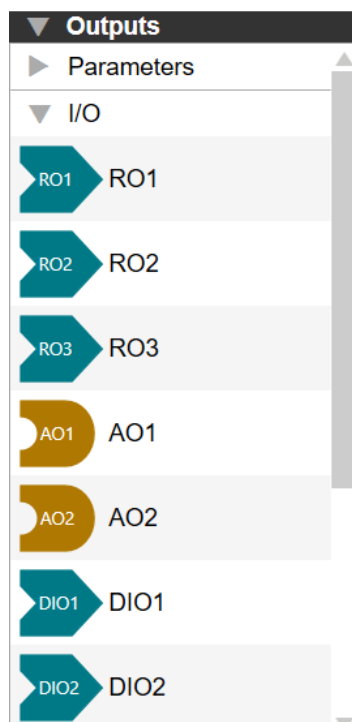
The block output value is written to the parameter only when the value changes. The written parameter values are not saved over power down of the drive.

For efficiency, the parameter reading and writing is made in the internal format. In case of some parameters, it is possible that the block input shows a different value than the corresponding parameter.

### ■ I/O

This output element list contains digital output, analog output and digital input/output signals of the drive or unit. For example, **D11**, etc.

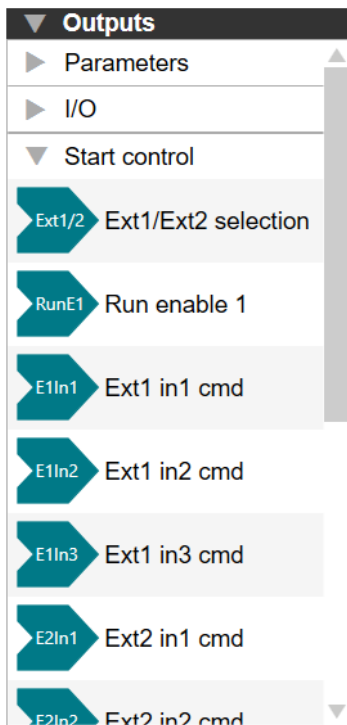
Drag and drop the output element to an output of a function block. Function block output shape indicates which type of output element it accepts, boolean or numeric.



## ■ Start control

This output element list contains signals that have effect on the drive start/stop control. For example:

- **EXT1/EXT2:** Defines the source signal for the external control location selection (parameter 19.11).
- **Run enable 1:** Defines the source for the run enable 1 signal (parameter 20.12).
- **Fault reset:** Defines the source for the external fault reset signal (parameter 31.11)

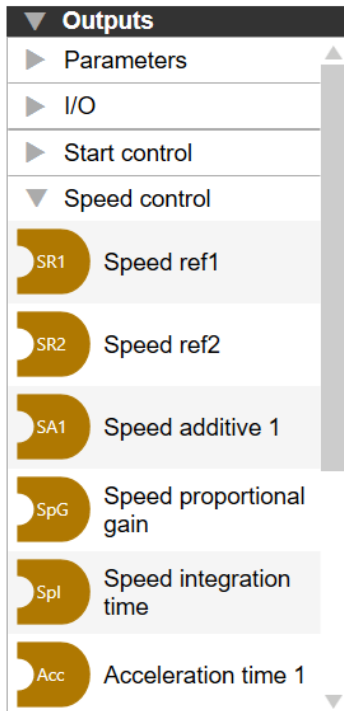


Refer to an appropriate firmware manual for more information.

## ■ Speed control

This output element list shows signals that have effect on the speed control of the drive. For example:

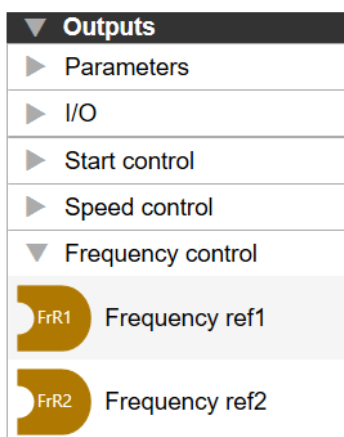
- **Speed ref1:** Defines the value for the Speed reference 1 signal in the speed control chain (parameter 22.11). Refer to the firmware manual for the drive parameter and the location of the signal in the speed reference control chain.



### ■ Frequency control

This output element list shows signals that have effect on the frequency control of the drive. For example:

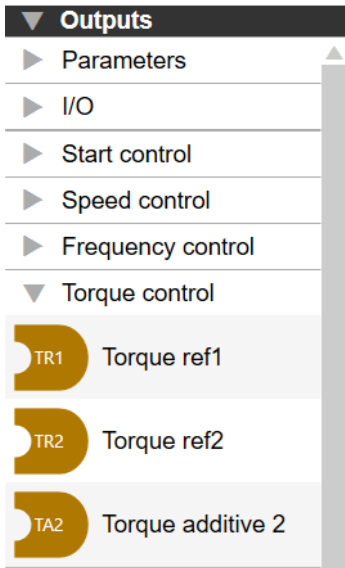
- **Frequency ref1:** Defines the value for the Frequency reference 1 signal in the frequency control chain (parameter 28.11). Refer to the firmware manual for the drive parameter and the location of the signal in the frequency reference control chain.



### ■ Torque control

This output element list shows signals that have effect on the torque control of the drive. For example:

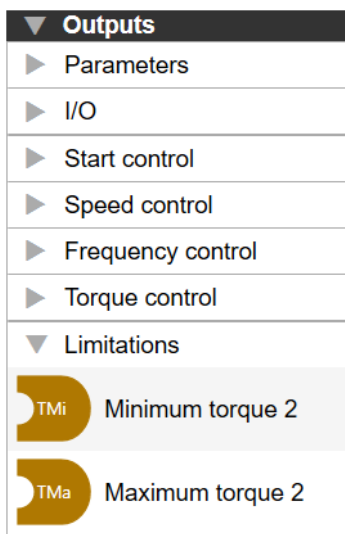
- **Torque ref1:** Defines the value for the Torque reference 1 signal in the torque control chain (parameter 26.11). Refer to the firmware manual for the drive parameter and the location of the signal in the torque reference control chain.



### ■ Limitations

This output element list defines the source of maximum torque limit for the drive. For example:

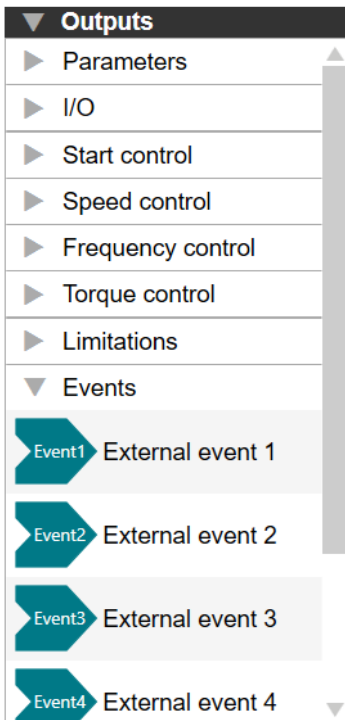
- **Minimum torque 2:** Defines the value for the Minimum torque 2 signal in the torque limit control chain (parameter 30.23). Refer to the firmware manual for the drive parameter and the location of the signal in the torque limit reference control chain.



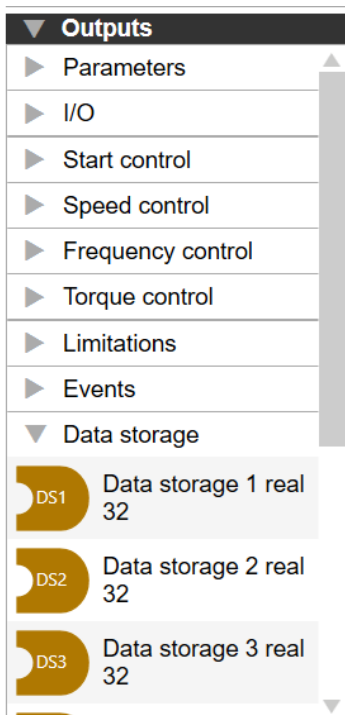
### ■ Events

This output element list shows signals that have effect on the external events of the drive. For example:

- **External event 1:** Defines the value for the External event 1 signal in the external event control chain (parameter 31.1). Refer to the firmware manual for the drive parameter and the location of the external event in the torque reference control chain.



■ **Data storage**



■ **Process PID**

For example: Set 1 setpoint 1, Set 1 feedback 1, Set 1 tracking mode etc. For more information on output descriptions, refer to firmware manual(s) in [Related manuals \(page 12\)](#)

## Function block specifications

You can adjust the number of inputs by dragging the bottom line in the function block.

**Note:** Function blocks which do not contain bottom line cannot be adjusted.

### ■ Abs

Calculates absolute value.



### Output:

Name	Type	Default value
Out	Float	0

### Input: 1

Name	Type	Default value	Function
In	Float	0	Block input

### Block function

Block calculates absolute value of value in input In. Output = | In |.

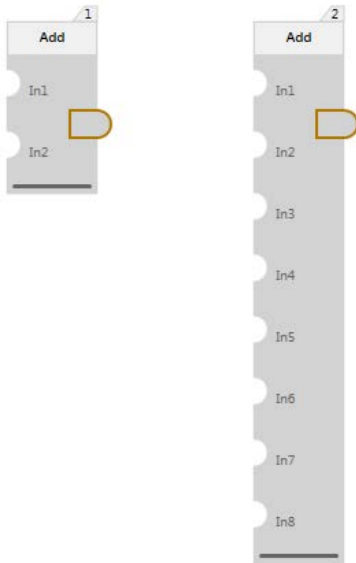
### Exceptional cases

Block input is not connected. Input has a default value.



■ **Add**

Adds **n** inputs and outputs result.



**Output:**

Name	Type	Default value
Out	Float	0

**Inputs: 2-8**

**Default inputs: 2**

Name	Type	Default value	Function
In1 - In8	Float	0	Provides values to add

**Block function**

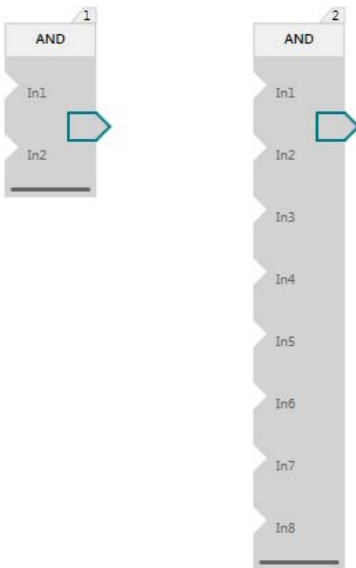
Output = In1 + In2 + ... + In8.

**Exceptional cases**

- Inputs which are not connected are added as default value.
- Overflow to positive side: output is limited to Max float.
- Overflow to the negative side: output is limited to negative Max float.
- Underflow: value 0 is kept at output.

■ **AND**

Performs logic AND.



**Output**

Name	Type	Default value
Out	Boolean	0

**Inputs: 2-8**

**Default inputs: 2**

Name	Type	Default value	Function
In1 - In8	Boolean	N/A	Block inputs

**Block function**

Function block performs logical conjunction operation with inputs.

$$\text{Out} = \text{In1} \& \text{In2} \& \dots \& \text{In8}.$$

The truth table of AND operation is below. Example uses two inputs. Same logic can be applied to other inputs. Output is 1 (true) if and only if all inputs have value 1 (true).

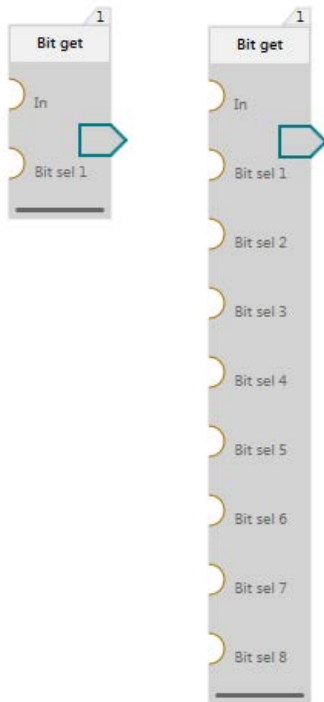
In1	In2	Out
0	0	0
0	1	0
1	0	0
1	1	1

**Exceptional cases**

- Inputs which are not connected have no effect on the output.
- If some inputs are connected and others are not, only the connected inputs are evaluated.

## ■ Bit get

Performs logic OR operation with selected bits from inputs.



### Output

Name	Type	Default value
Out	Boolean	0

### Inputs: 2-9

Name	Type	Default value	Function
In	Float	0	Value to read bits
Bit sel 1 - 8	Float	N/A	Provides number of bits to be selected from input value.

### Block function

Basic functionality of the block is to get the value of the defined bit. In case several bits are defined then values of these bits are retrieved and OR operation is executed with these to get the block output value.

Bits 0 - 15 can be selected.

For example, in case only Bit sel 1 is connected then  $Out = val1$ . If Bit sel 1 and 2 are connected then  $Out = val1 \text{ OR } val2$ , where  $val1$  - value of bit selected by Bit sel 1 input and  $val2$  - value of bit selected by Bit sel 2 input.

### Exceptional cases

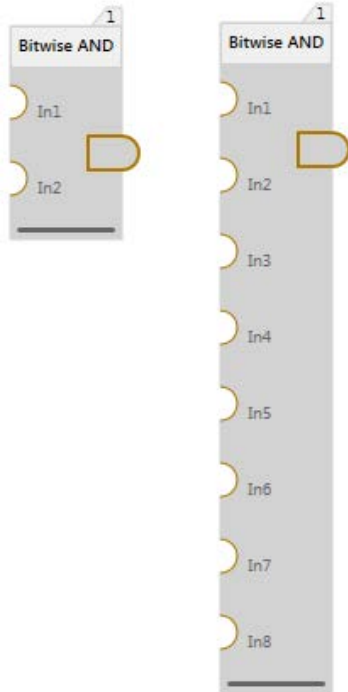
- Bit sel input is not connected. Bit defined by this input is skipped.
- If entered bit sel value > 15, bit 15 is selected.

#### 44 Program elements

- If bit sel < 0 then bit 0 is selected.
  - If input In is not connected, it gets default value.
  - An input In value that is either negative or larger than  $(2^{31})-1$  is set to default value 0.
-

■ **Bitwise AND**

ANDs the lowest 16 separate bits of the input values and outputs the combination as float.



**Output**

Name	Type	Default value
Out	Float	0

**Inputs: 2-8**

Name	Type	Default value	Function
In1 - In8	Float	N/A	Provides an input value.

**Block function**

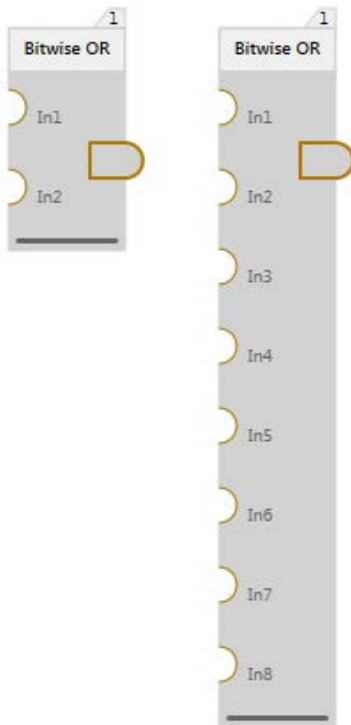
Connected inputs are rounded to the nearest integer after which the AND operation is performed on them. The lowest 16 bits of the result is taken, converted to float and written to output.

**Exceptional cases**

- An input value that is either negative or larger than  $(2^{31})-1$  is set to default value 0.
- If only 1 input is connected then that input is rounded and sent to the output.

## ■ Bitwise OR

ORs the lowest 16 separate bits of the input values and outputs the combination as float.



### Output

Name	Type	Default value
Out	Float	0

### Inputs: 2-8

Name	Type	Default value	Function
In1 - In8	Float	0	Provides an input value.

### Block function

Inputs are rounded to the nearest integer after which the OR operation is performed on them. The lowest 16 bits of the result is taken, converted to float and written to output.

### Exceptional cases

- An input value that is either negative or larger than  $(2^{31})-1$  is set to default value 0.
- If only 1 input is connected then that input is rounded and sent to the output.
- Disconnected inputs have default value 0.

## ■ Bitwise XOR

XORs the lowest 16 separate bits of the input values and outputs the combination as float.



### Output

Name	Type	Default value
Out	Float	0

### Inputs: 2

Name	Type	Default value	Function
In1	Float	0	Provides an input value.
In2	Float	0	Provides an input value.

### Block function

Inputs are rounded to the nearest integer after which the XOR operation is performed on them. The lowest 16 bits of the result is taken, converted to float and written to output.

### Exceptional cases

- An input value that is either negative or larger than  $(2^{31})-1$  is set to default value 0.
- If only 1 input is connected then that input is rounded and sent to the output.

## ■ Divide

Divides block inputs.



### Output:

Name	Type	Default value
Out	Float	0

### Inputs: 2

Name	Type	Default value	Function
Num	Float	0	Dividend
Denom	Float	0	Divisor

### Block function

$$\text{Output} = \text{In1} / \text{In2}$$

Dividing by zero will set block output to zero.

### Exceptional cases

- Inputs which are not connected are assigned with default values.
- Overflow to positive side: output is limited to Max float.
- Overflow to the negative side: output is limited to negative Max float.
- Underflow: value 0 is kept at output.



## ■ Equal

Checks if values at inputs are equal.



### Output

Name	Type	Default value
Out	Boolean	0

### Inputs: 2

Name	Type	Default value	Function
A	Float	0	First comparison value
B	Float	0	Second comparison value

### Block function

Block compares the whole number parts of numbers in A and B. Behavior of the block can be seen in table below.

Condition	Out
A and B are equal	1
A and B are not equal	0

Inputs are rounded before comparison. Only whole number part of the inputs are compared.

For example, if value 70.5 is in input, it will be compared as 71. If value 70.4 is in input it will be compared as 70. Rounding of negative numbers works as illustrated in the following example. -70.4 rounds to -70. -70.5 rounds to -71.

### Exceptional cases

Inputs which are not connected will have a default value.

## ■ Filter

Filters input for a defined length of time and then outputs it.



### Output:

Name	Type	Default value
Out	Float	0

### Inputs: 2

Name	Type	Default value	Function
In	Float	0	Signal to be filtered
Time	Float	0	Filter time constant in seconds

### Block function

This block is a single pole low - pass filter. Input signal *In* is filtered using provided time constant *Time*. The following equation is used for internal calculations.

$$\text{Coefficient} = \text{TimeLevel} / (\text{TimeLevel} + \text{Time})$$

$$\text{Out}[i] = \text{Coefficient} * (\text{In}[i] - \text{Out}[i - 1]) + \text{Out}[i - 1]$$

### Where:

Variable	Function
Out [i]	Current calculated output value
Out [i - 1]	Previous output value of the filter from previous time cycle
In [i]	Current input value
Timelevel	Value of timelevel that the program is running at.

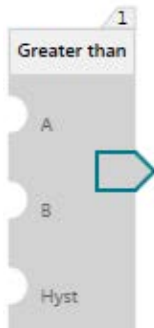
This function is a discrete model for single pole low - pass filter.

### Exceptional cases

- Time constant *Time* < timelevel or negative constant is provided. Filter does not filter input signal. Input is written to output unaltered. Time constant is evaluated to 0.
- *In* is not connected - Input gets default value.
- *Time* constant is not connected - assumed to have default value.

## ■ Greater than

Comparison block. Compares values at its inputs to see if first value is greater than second. Comparison accuracy is set by the user.



### Output

Name	Type	Default value
Out	Boolean	0

### Inputs: 3

Name	Type	Default value	Function
A	Float	0	Provides first comparison value
B	Float	0	Provides second comparison value
Hyst	Float	0	Value B is subtracted

### Block function

Takes two inputs to compare with one another, A and B, and a third input that manipulates input B.

#### First:

- If  $A > B$ , output is set to 1.

#### Second (if first is not true):

- If  $A < (B - \text{Hyst})$  then output is reset to 0.

#### Third (if neither are true):

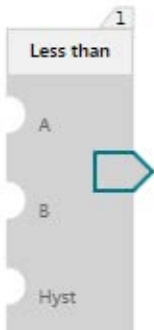
- Previous output value is kept at block output.

### Exceptional cases

- When either A or B input is not connected then output is set to default value 0.
- A disconnected Hyst input has value 0.

## ■ Less than

Comparison block. Compares values at its inputs to see if first value is smaller than second. Comparison accuracy is set by the user.



### Output

Name	Type	Default value
Out	Boolean	0

### Inputs: 3

Name	Type	Default value	Function
A	Float	0	Provides first comparison value
B	Float	0	Provides second comparison value
Hyst	Float	0	Value that is added to B

### Block function

Takes two inputs to compare with one another, A and B, and a third input that manipulates input B.

#### First

- If  $A < B$ , output is set to 1.

#### Second (if first isn't true)

- If  $A > (B + \text{Hyst})$  then output is reset to 0.

#### Third (if neither are true)

- Previous output value is kept at block output.

#### Exceptional cases

- When either A or B input is not connected then output is set to default value 0.
- A disconnected Hyst input has value 0.

## ■ Limit

Takes an input that is limited and outputs the value after limiting it.



### Output:

Name	Type	Default value
Out	Float	0

### Inputs: 3

Name	Type	Default value	Function
In	Float	0	Value to be limited.
Max	Float	3.4028235e+38	Maximum value In is limited
Min	Float	- 3.4028235e+38	Minimum value In is limited.

### Block function

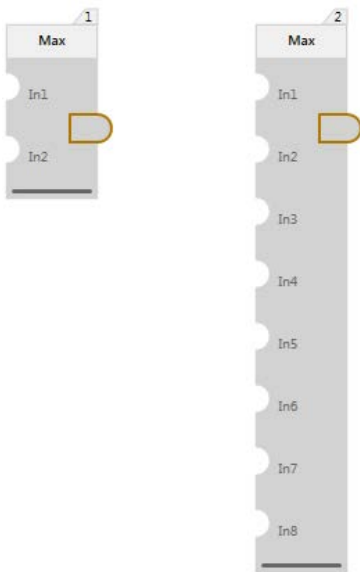
*In* is written to the output as long as it is within the value range of *Max* and *Min*. When *In* exceeds or falls below the respective limit values, it will first be capped to the appropriate limit value and then written to the output. *In* is evaluated first against *Max*. If *Max* is not limiting, then *In* is evaluated against *Min*.

### Exceptional cases

- If *In* is not connected then the block output is zero.
- If *Max* or *Min* input is not connected, then the highest and lowest float values are set as the default values for *Max* or *Min*.

■ **Max**

Compares **n** inputs and outputs the largest input value.



**Output:**

Name	Type	Default value
Out	Float	0

**Inputs: 2-8**

**Default inputs: 2**

Name	Type	Default value	Function
In1 - In8	Float	0	Provides an input value to compare

**Block function**

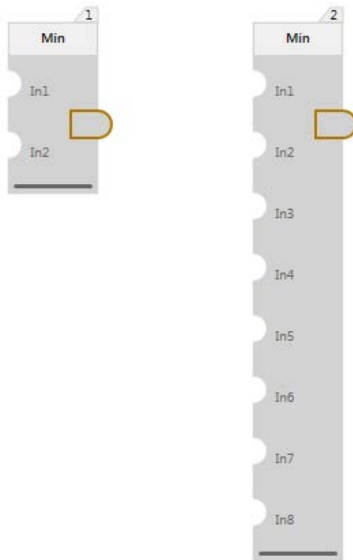
Compares all input values to determine the highest one and outputs it.

**Exceptional cases**

If some inputs are connected and other inputs are not connected, only the connected inputs are evaluated.

■ **Min**

Compares **n** inputs and outputs the smallest input value.



**Output:**

Name	Type	Default value
Out	Float	0

**Inputs: 2-8**

**Default inputs: 2**

Name	Type	Default value	Function
In - In8	Float	0	Provides an input value to be compared

**Block function**

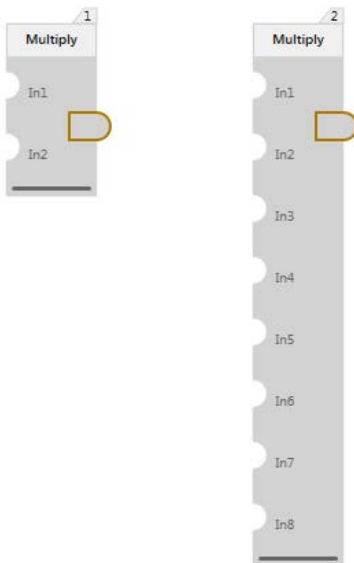
Compares all input values to determine the lowest one and outputs it.

**Exceptional cases**

If some inputs are connected and others are not connected, only the connected inputs are evaluated.

## ■ Multiply

Multiples **n** inputs and outputs the result.



### Output:

Name	Type	Default value
Out	Float	0

### Inputs: 2-8

### Default inputs: 2

Name	Type	Default value	Function
In1 - In8	Float	N/A	Provides values for multiply block to perform multiplication

### Block function

$$\text{Out} = \text{In1} * \text{In2} * \dots * \text{In8}$$

### Exceptional cases

- Inputs which are not connected are not multiplied. If one input is connected, its value is at output.
- All inputs are not connected: output is assigned a default value.
- Overflow to positive side: output is limited to Max float.
- Overflow to the negative side: output is limited to negative Max float.
- Underflow: value 0 is kept at output.



## ■ NOT

Inverts value at input.



### Output

Name	Type	Default value
Out	Boolean	1

### Input: 1

Name	Type	Default value	Function
In	Boolean	0	Block input

### Block function

Function block performs inversion.

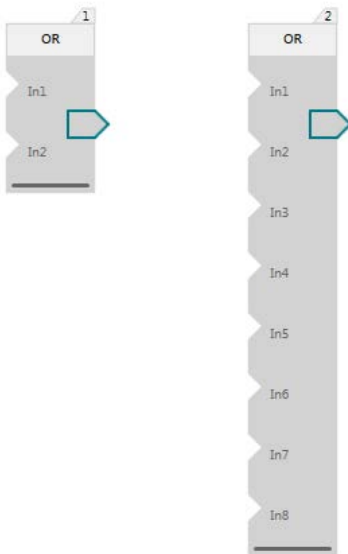
In	Out
0	1
1	0

### Exceptional cases

In case a block input is not connected then its value is set to 0 by default.

## ■ OR

Performs logic *OR*.



### Output

Name	Type	Default value
Out	Boolean	0

### Inputs: 2-8

#### Default inputs: 2

Name	Type	Default value	Function
In1 - In8	Boolean	0	Block inputs

### Block function

Function block performs logical or operation with inputs.  $Out = In1 \vee In2 \vee \dots \vee In8$ .

The truth table of OR operation is below. Example uses two inputs. Same logic can be applied to other inputs. Output has value 1 when one of the inputs have value 1. Output is 0 if and all inputs have value 0.

In0	In1	Out
0	0	0
0	1	1
1	0	1
1	1	1

### Exceptional cases

If some inputs are connected and others are not, only the connected inputs are evaluated.

■ **PI**

PI controller.



**Output:**

Name	Type	Default value
Out	Float	0

**Inputs: 8**

Name	Type	Default value	Function
Setpoint	Float	0	Desired output value
Actual	Float	0	Actual output value
Gain	Float	0	Proportional gain (Kp)
Integration time	Float	0	Integration time in seconds (s)
Track	Boolean	0	Enables tracking mode
Track reference	Float	0	Output value in tracking mode
Min	Float	- 3.4028235e+38	Maximum output value
Max	Float	3.4028235e+38	Minimum output value

### **Block function**

Calculates the P and I terms based on error, proportional gain and an integral coefficient. The sum of P and I is written to the output. Sets output to tracking reference value when tracking is enabled and limits the output when needed. In these cases, the I term value is maintained directly in reference to the tracking reference or limit values to provide smooth transfer/anti-windup. PI output continuous changing from track reference value when track is disabled. In the limitation, the value is evaluated first against *Max* limit. If *Max* is not limiting, then the value is evaluated against *Min* limit.

### **Exceptional cases**

- In case a block input is not connected then its value is set to default value.
  - When either *Setpoint*, *Actual* or *Gain* are not connected then output is set to 0. When *Track* is enabled and *Track reference* is not connected then output is set to 0.
  - When Integration time input is not connected then integral component is reset and PI block functions as a P controller.
  - When *Min* or *Max* is not connected, the default values of these inputs are used.
-

## ■ Ramp

Changes the output value to match the input value at a defined rate of change.



### Output

Name	Type	Default value
Out	Float	0

### Inputs: 7

Name	Type	Default value	Function
In	Float	0	Reference value to ramp to output
Increase	Float	0	The amount of output increased per second
Decrease	Float	0	The amount of output decreased per second
Track	Boolean	0	Enables tracking mode
Track reference	Boolean	0	Output value in tracking mode
Max	Float	3.4028235e+38	Maximum value block output will be limited
Min	Float	- 3.4028235e+38	Minimum value block output will be limited

### Block function

If output value does not equal input reference, then the output value starts changing towards the input value.

---

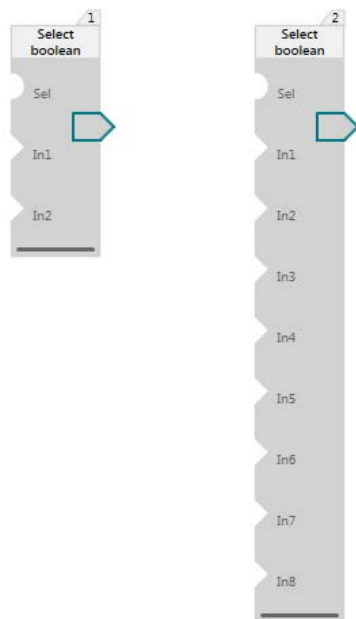
The amount of change per second is defined by the inputs for increasing and decreasing the output. Sets output to track reference value when track is enabled. Output is limited to maximum and minimum limit values. In the limitation, the output is evaluated first against *Max* limit. If *Max* is not limiting, then the output is evaluated against *Min* limit. Ramp output continues changing from tracking reference value when tracking is disabled.

**Exceptional cases**

- In case a block input is not connected, then its value is set to default value.
  - In case, either maximum or minimum limit is disconnected, then their values will be defaulted to the highest and lowest value representable by a float.
  - In case, *Increase* or *Decrease* input is disconnected then Output = *In* when trying to ramp with the disconnected input. If the other input is connected then ramping with it behaves as normal.
  - In case, *In* input is disconnected then Output = 0.
-

## ■ Select boolean

Outputs the Boolean input value that is selected by the selector input.



### Output

Name	Type	Default value
Out	Boolean	0

### Inputs: 3-9

#### Default inputs: 3

Name	Type	Default value	Function
Sel	Float	0	Selects input value to connect to output
In1 - In8	Boolean	0	Provides selectable input value for the block.

### Block function

This is a selector block that can have different input connected to output. Input to be connected is selected by *Sel* input.

When *Sel* = 1 then *Out* = *In1*, when *Sel* = 2 *Out* = *In2* etc.

When *Sel* = 8 *Out* = *In8*.

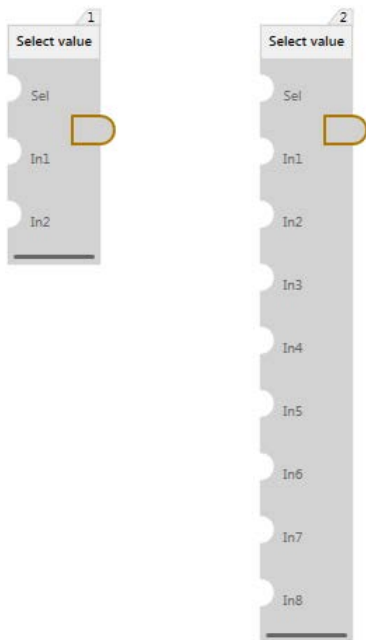
Allowable value range for *Sel* input is  $1 \leq Sel \leq 8$ .

### Exceptional cases

- When *Sel* input is out of its allowable range then *Out* = 0.
- Inputs which are not connected will have a default value.

■ **Select value**

Outputs the float input value that is selected by the selector input.



**Output**

Name	Type	Default value
Out	Float	0

**Inputs: 3-9**

**Default inputs: 3**

Name	Type	Default value	Function
Sel	Float	0	Selects input to be connect to output
In1 - In8	Float	0	Provides selectable input value for the block

**Block function**

This is a selector block that can have different input connected to output. Input to be connected is selected by Sel input.

When,  $Sel = 1$  then  $Out = In1$ , and  $Sel = 2$  then  $Out = In2$  and etc.

When,  $Sel = 8$  then  $Out = In8$ .

Allowable value range for Sel input is  $1 \leq Sel \leq 8$ .

**Exceptional cases**

- When Sel input is out of its allowable range then Output = 0.
- Inputs which are not connected will have a default value.



■ **Set bits 0-7**

Updates bits 0-7 of the input value.



**Output**

Name	Type	Default value
Out	Float	0

**Inputs: 9**

Name	Type	Default value	Function
In	Float	0	Value to be updated
Bit0	Boolean	N/A	Value of bit 0 (lowest)
Bit1	Boolean	N/A	Value of bit 1
Bit2	Boolean	N/A	Value of bit 2
Bit3	Boolean	N/A	Value of bit 3
Bit4	Boolean	N/A	Value of bit 4
Bit5	Boolean	N/A	Value of bit 5
Bit6	Boolean	N/A	Value of bit 6
Bit7	Boolean	N/A	Value of bit 7

### **Block function**

Rounds the float input to closest integer and updates bits 0-7 of the integer value based on the boolean inputs Bit0-Bit7. Takes then the lowest 16 bits of the integer result and converts the value to float and writes it to output.

### **Exceptional cases**

- An input value that is either negative or larger than  $(2^{31})-1$  is set to default value 0. Bits 0-7 of the default value are updated.
  - If Boolean input is not connected, the value of that bit is not updated.
-

■ **Set bits 8-15**

Update bits 8-15 of the input value.



**Output**

Name	Type	Default value
Out	Float	0

**Inputs: 9**

Name	Type	Default value	Function
In	Float	0	Value to be updated
Bit8	Boolean	N/A	Value of bit 8
Bit9	Boolean	N/A	Value of bit 9
Bit10	Boolean	N/A	Value of bit 10
Bit11	Boolean	N/A	Value of bit 11
Bit12	Boolean	N/A	Value of bit 12
Bit13	Boolean	N/A	Value of bit 13
Bit14	Boolean	N/A	Value of bit 14
Bit15	Boolean	N/A	Value of bit 15

### **Block function**

Rounds the float input to closest integer and updates bits 8-15 of the integer value based on the Boolean inputs Bit8-Bit15. Takes then the lowest 16 bits of the integer result and converts the value to float and writes it to output.

### **Exceptional cases**

- An input value that is either negative or larger than  $(2^{31})-1$  is set to default value 0. Bits 8-15 of the default value are updated.
  - If Boolean input is not connected, the value of that bit is not updated.
-

## ■ Square root

Calculates square root of value at input



### Output

Name	Type	Default value
Out	Float	0

### Inputs: 1

Name	Type	Default value	Function
In	Float	0	Block input

### Block function

Block calculates square root of input.  $Out = \sqrt{In}$

### Exceptional cases

- When value at the input is negative ( $In < 0$ ), then  $Out = 0$

## ■ SR

SR trigger is used to store Set value.



### Output

Name	Type	Default value
Out	Boolean	0

### Input: 2

Name	Type	Default value	Function
Set	Boolean	0	Set input
Reset	Boolean	0	Reset

### Block function

This is SR latch. Output keeps its value once set by *Set* input. Value at output is reset to 0 when *Reset* = 1. Value at output depends on previous output value. See truth table.

Previous Out	Reset	Set	Current Out
0	0	0	0
0	0	1	1
x	1	x	0
1	0	0	1
1	0	1	1

### Exceptional cases

- If *Set* is not connected, it is assumed to have default value.
- If *Reset* is not connected, it is assumed to have default value.

## ■ Subtract

Performs subtract.



### Output:

Name	Type	Default value
Out	Float	0

### Inputs: 2

Name	Type	Default value	Function
In1	Float	0	Value to subtract from
In2	Float	0	Value to be subtracted

### Block function

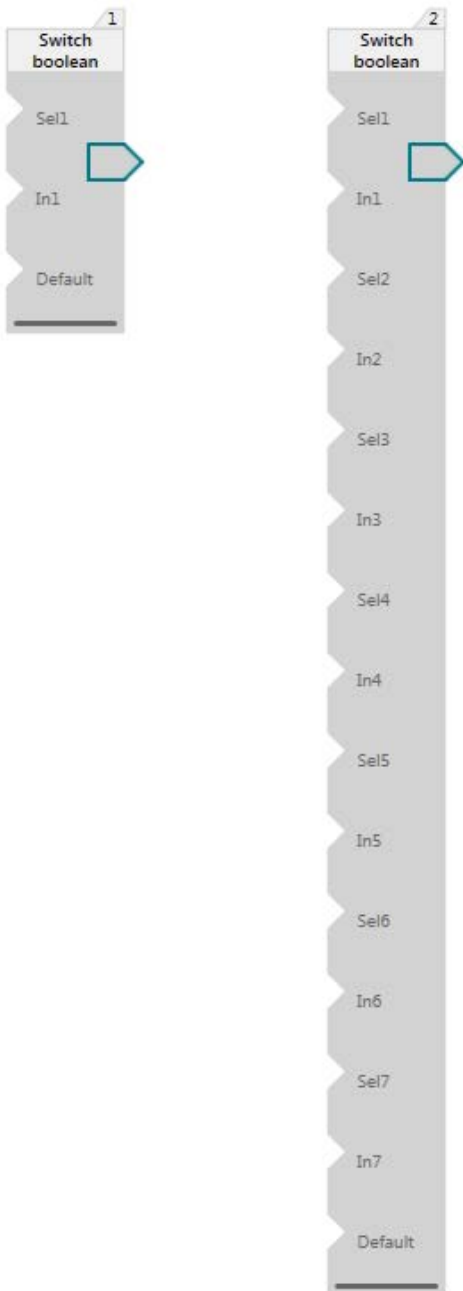
Output = In1 - In2

### Exceptional cases

- In case both inputs are not connected, output has a default value.
- Inputs which are not connected are assigned default value.
- Overflow to positive side: output is limited to Max float.
- Overflow to the negative side: output is limited to negative Max float.
- Underflow: value 0 is kept at output.

### ■ Switch boolean

Outputs the input Boolean value whose enable value is set first.



#### Output:

Name	Type	Default value
Out	Boolean	0

#### Inputs: 3-15



**Default inputs: 3**

Name	Type	Default value	Function
Sel1 - Sel7	Boolean	0	Selects/deselects input value.
In1 - In7	Boolean	0	Provides selectable input value for the block.
Default	Boolean	0	Default output when Sel is not active for any inputs.

**Block function**

The value written to the output is “In X” value whose “Sel X” is set first. If no “Sel X” is set then *Default* input is written to the output.

**Example:**

Multiple Sel inputs have value 1. Inputs are evaluated from top to bottom. In case of multiple *In, Sel* pairs *In1, Sel1* is checked first followed by *In2, Sel2* and etc. In case Multiple Sel inputs are 1 the first one will be connected to output. In this example, if both *Sel1* and *Sel2* are 1 then *In1* is connected to output.

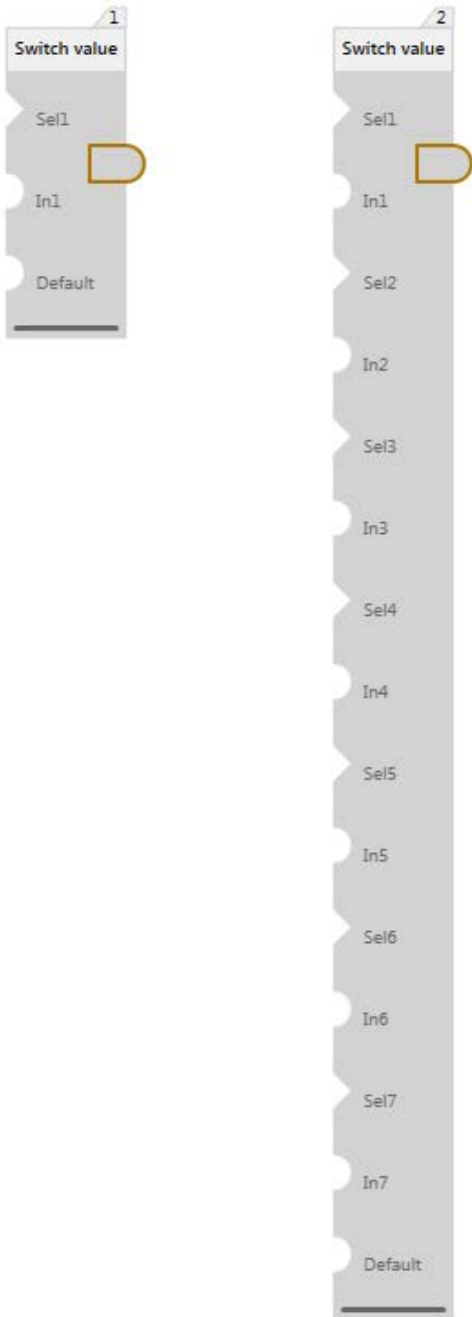
**Exceptional cases**

Inputs which are not connected will have a default value.

---

■ **Switch value**

Outputs the input float value whose enable value is set first.



**Output:**

Name	Type	Default value
Out	Float	0

**Inputs: 3-15**

**Default inputs: 3**

Name	Type	Default value	Function
Sel1 - Sel7	Boolean	0	Selects/deselects input value
In1 - In7	Float	0	Provides selectable input value for the block
Default	Float	0	Default, that is, connected to output when no Sel is 1

**Block function**

The value written to the output is “In X” value whose “Sel X” is set first. If no “Sel X” is set, then the *Default* input is written to the output.

**Example:**

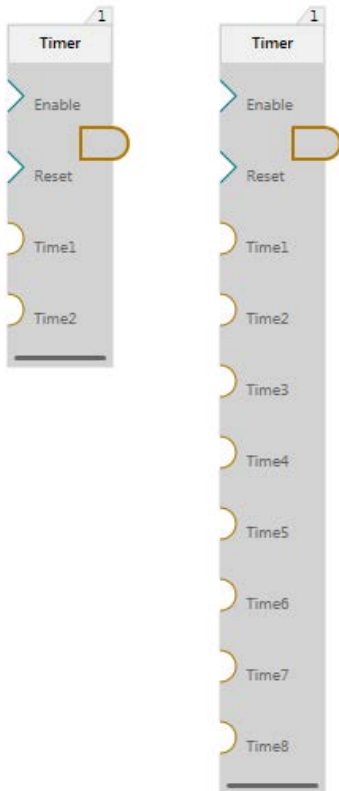
Multiple Sel inputs have value 1. Inputs are evaluated from top to bottom. In case of multiple *In*, Sel pairs *In1*, *Sel1* is checked first followed by *In2*, *Sel2* etc. In case Multiple Sel inputs are 1, the first one will be connected to output. In this example, if both *Sel1* and *Sel2* are 1 then *In1* is connected to output.

**Exceptional cases**

- Inputs which are not connected will have a default value.

■ **Timer**

Runs through states at the speed of timer values defined at the inputs. Outputs the current state. The timers can be paused and the state can be reset.



**Output**

Name	Type	Default value
Out	Float	1

**Inputs: 4-10**

**Default inputs: 4**

Name	Type	Default value	Function
Enable	Boolean	0	Enables/disables timer.
Reset	Boolean	0	Resets time when rising edge is detected on input.
Time1 - Time8	Float	0	Provides time in state, time value is in seconds.

**Block funtion**

Timer block is a state machine that goes through states. The time block stays in each state is specified by time inputs Time1 - Time8. Minimal number of time inputs is 2. When timer starts, it is in state 1 and block output is 1. Timer stays in this state for the time specified in input Time1. When this time is passed, the timer block switches to the next state. This behavior of normal operation is illustrated below. Reset is false,

enable is true. Time values Time1 = 2s, Time2 = 1s and Time3 = 2s are used in all examples below.

Reset	F	F	F	F	F	F	F
Enable	T	T	T	T	T	T	T
Out	1	1	2	3	3	1	1
Time, s	1s	2s	3s	4s	5s	6s	7s

Timer block can be paused by setting enable to false. During which the block stays in the state that it was at the time. When Enable is set to true again, timer resume its work from where it left off. The effect of enable input is illustrated below.

Reset	F	F	F	F	F	F	F
Enable	T	T	T	F	F	T	T
Out	1	1	2	2	2	3	3
Time, s	1s	2s	3s	4s	5s	6s	7s

Timer block can be reset using the reset input. When rising edge is detected at the reset input, block goes to state 1 if it is a valid state. If time in state 1 is specified to be less than the time level that the program is running at, timer block will find the next valid state to go to starting from state 1. If all states have delay times that are less than the time level, block will go to state 1. The reset of the timer block happens also in case the block is not enabled.

The reset behavior under normal circumstances is illustrated below. In this example there are 3 time inputs and they all have valid delay times specified.

## 78 Program elements

Reset	F	F	F	F	F	F	F
Enable	T	T	T	F	F	T	T
Out	1	1	2	2	2	3	3
Time, s	1s	2s	3s	4s	5s	6s	7s

Block only reacts to rising edge. The reset behavior is illustrated below. The rising edge occurs at time 4s. Reset input is left true but this does not interfere with block operation. At time 5s block is in normal operation mode again.

Reset	F	F	F	T	T	F	F
Enable	T	T	T	T	T	T	T
Out	1	1	2	1	1	2	3
Time, s	1s	2s	3s	4s	5s	6s	7s

### Exceptional cases

- Not connected inputs get default values assigned.
- When specified time in a state is smaller than the value of the time level that the program is running, the state will be skipped.
- When all time inputs have times specified that are smaller than the time level value, the block output is set to default value.

## ■ Trigger down

Falling edge detection.



### Output

Name	Type	Default value
Out	Boolean	0

### Input: 1

Name	Type	Default value	Function
In	Boolean	0	Block input

### Block function

Function block performs falling edge detection. Output is 1 when input previous value is 1 and current value is 0. Otherwise output is 0.

### Exceptional cases

- If input *In* is not connected, it will get the default value.
- If input *In* has value 0 at the first execution cycle of the block, the output of the block is set to 0.

## ■ Trigger up

Rising edge detection.



### Output

Name	Type	Default value
Out	Boolean	0

### Input: 1

Name	Type	Default value	Function
In	Boolean	0	Block input

### Block function

Function block performs rising edge detection. Output is 1 when block input previous value is 0 and current value is 1. Otherwise output is 0.

### Exceptional cases

- When input *In* is not connected, it will get the default value.
  - If input *In* has value 1 at the first execution cycle of the block, the output of the block is set to 1.
-



## ■ T\_off

Turns off the delay.



### Output

Name	Type	Default value
Out	Boolean	0

### Inputs: 2

Name	Type	Default value	Function
In	Boolean	0	Provides boolean value
Delay	Float	0	Provides the time value in seconds to delay outputting 0

### Block function

If the value of *In* is 1 then it is written to the output. If the value of *In* is 0 it is written to the output only after a time period is passed which is defined by *Delay*. *Delay* is limited to 2097152 seconds.

### Exceptional cases

In case a block input is not connected, then its value is set to default value.

## ■ T\_on

Turns on the delay.



### Output

Name	Type	Default value
Out	Boolean	0

### Inputs: 2

Name	Type	Default value	Function
In	Boolean	0	Provides boolean value.
Delay	Float	0	Provides time value in seconds to delay outputting 1.

### Block function

If the value of *In* is 0 then it is written to the output. If the value of *In* is 1, it is written to the output only after a time period is passed which is defined by *Delay*. *Delay* is limited to 2097152 seconds.

### Exceptional cases

In case a block input is not connected then its value is set to default value.

## ■ XOR

XOR inputs.



### Output

Name	Type	Default value
Out	Boolean	0

### Inputs: 2

Name	Type	Default value	Function
In1	Boolean	0	Block input
In2	Boolean	0	Block input

### Block function

Function block performs logical XOR operation with inputs.

The truth table of XOR operation:

In1	In2	Out
0	0	0
0	1	1
1	0	1
1	1	0

Output has value 1 when the inputs have different values, otherwise the output is 0.

### Exceptional cases

In case a block input is not connected, the default value of the input is used in the operation.



---

# Further information

## Product and service inquiries

Address any inquiries about the product to your local ABB representative, quoting the type designation and serial number of the unit in question. A listing of ABB sales, support and service contacts can be found by navigating to [www.abb.com/contact-centers](http://www.abb.com/contact-centers).

## Product training

For information on ABB product training, navigate to [new.abb.com/service/training](http://new.abb.com/service/training).

## Providing feedback on ABB manuals

Your comments on our manuals are welcome. Navigate to [new.abb.com/drives/manuals-feedback-form](http://new.abb.com/drives/manuals-feedback-form).

## Document library on the Internet

You can find manuals and other product documents in PDF format on the Internet at [www.abb.com/drives/documents](http://www.abb.com/drives/documents).



[www.abb.com/drives](http://www.abb.com/drives)



3AXD50000028574D