SCE Training Curriculum for the end-to-end automation solution
Totally Integrated Automation (TIA)

TIA Portal Module 030-010
Block Types of SIMATIC S7-300
Suitable SCE trainer packages for these documents

**SIMATIC controllers**
- SIMATIC S7-300 with CPU 314C-2PN/DP  
  Order no.: 6ES7314-6EH04-4AB3
- SIMATIC S7-300 with CPU 314C-2PN/DP (upgrade)  
  Order no.: 6ES7314-6EH04-4AB4
- SIMATIC S7-300 with CPU 315F-2PN/DP  
  Order no.: ES7315-2FH14-4AB1
- SIMATIC ET 200S with CPU IM151-8 F PN/DP  
  Order no.: 6ES7151-8FB00-4AB1

**SIMATIC STEP 7 software for training**
- SIMATIC STEP 7 Professional V11 - Single license  
  Order no.: 6ES7822-1CC01-4YA5
- SIMATIC STEP 7 Professional V11 - Classroom license (up to 12 users)  
  Order no.: 6ES7822-1AA01-4YA5
- SIMATIC STEP 7 Professional V11 - Upgrade license (up to 12 users)  
  Order no.: 6ES7822-1AA01-4YE5
- SIMATIC STEP 7 Professional V11 - Student license (up to 20 users)  
  Order no.: 6ES7822-1AC01-4YA5

Please note that these trainer packages may be replaced by successor trainer packages.  
An overview of the currently available SCE packages is provided under: <siemens.com/sce/tp>

**Advanced training**
Please get in touch with your regional SCE contact for information on regional Siemens SCE advanced training  
<siemens.com/sce/contact>

**Additional information regarding SCE**
<siemens.com/sce>

**Information regarding usage**
This training curriculum for the end-to-end automation solution Totally Integrated Automation (TIA) was prepared for the program "Siemens Automation Cooperates with Education (SCE)" specifically for training purposes for public educational facilities and R&D facilities. Siemens AG does not make any guarantee regarding its contents.

This training curriculum may only be used for initial training on Siemens products/systems. That is, it may be copied in whole or in part and handed out to trainees for use within the context of their training. Distribution and reproduction of this document and disclosure of its contents are permitted within public education and further education facilities for educational purposes.

Any exceptions require written consent from the Siemens AG contact person: Mr. Roland Scheuerer  
roland.scheuerer@siemens.com.

Offenders will be liable for damages. All rights reserved, including those relating to translation and in particular those rights created as a result of a patent being granted or utility model being registered. Use for industry customers is expressly prohibited. We do not consent to any commercial use of the training curriculum.

We would like to thank Michael Dziallas Engineering and all those involved for their support in creating this training curriculum.
1. Preface .................................................................................................................................................. 4
2. Notes on programming for SIMATIC S7-300 ......................................................................................... 6
   2.1 SIMATIC S7-300 automation system ............................................................................................... 6
   2.2 STEP 7 Professional V11 (TIA Portal V11) programming software ................................................ 6
3. Block types for SIMATIC S7-300 ........................................................................................................ 7
   3.1 Linear programming ........................................................................................................................ 7
   3.2 Structured programming .................................................................................................................. 8
   3.2 User blocks for SIMATIC S7-300 ..................................................................................................... 9
       3.2.1 Organization blocks .................................................................................................................... 10
       3.2.2 Functions ................................................................................................................................ 11
       3.2.3 Function blocks ........................................................................................................................ 11
       3.2.4 Data blocks ................................................................................................................................ 12
4. Example task: Function block for belt control ..................................................................................... 13
5. Programming of the belt control for SIMATIC S7-300 ........................................................................ 14
1. Preface

The SCE_EN_030-010 module contents form part of the ‘Advanced functions for PLC programming’ training unit and explain the various blocks that can be used for SIMATIC S7 programming.

Learning objective:

In this module, the reader will become acquainted with the various blocks for SIMATIC S7 programming with the TIA Portal programming tool. This module explains the various block types and describes the procedure for creating a program in a function block in the steps outlined below.

- Creating a function block
- Defining internal tags
- Programming using internal tags in the function block
- Opening and assigning parameters to the function block in OB1

Requirements:

To successfully work through this module, the following knowledge is required:

- Proficiency in working with Windows
- Basic knowledge of PLC programming with the TIA Portal (e.g., module SCE_EN_020-010_R1110_Startup programming with SIMATIC S7-300)
Required hardware and software

1. PC Pentium 4, 1.7 GHz 1 (XP) – 2 (Vista) GB RAM, approx. 2 GB of free hard disk space
   Operating system Windows XP Professional SP3 / Windows 7 Professional / Windows 7 Enterprise / Windows 7 Ultimate / Windows 2003 Server R2 / Windows Server 2008 Premium SP1, Business SP1, Ultimate SP1
3. Ethernet connection between the PC and CPU 315F-2 PN/DP
4. SIMATIC S7-300 PLC, e.g., CPU 315F-2PN/DP with 16DI/16DO signal module. The inputs must be fed out to a control panel.
2. Notes on programming for SIMATIC S7-300

2.1 SIMATIC S7-300 automation system

The SIMATIC S7-300 automation system is a modular microcontroller system for the low and medium performance range. A comprehensive range of modules is available to optimally adapt the system to the automation task.

The S7 controller consists of a power supply, a CPU, and input and output modules for digital and analog signals. If necessary, communication processors and function modules are also used for special tasks such as stepper motor control.

The programmable logic controller (PLC) monitors and controls a machine or a process using the S7 program. The S7 program scans the I/O modules via input addresses (%I) and addresses their output addresses (%Q).

The system is programmed with the STEP 7 software.

2.2 STEP 7 Professional V11 (TIA Portal V11) programming software

The STEP 7 Professional V11 (TIA Portal V11) software is the programming tool for the following automation systems:
- SIMATIC S7-1200
- SIMATIC S7-300
- SIMATIC S7-400
- SIMATIC WinAC

STEP 7 Professional V11 provides the following functions for plant automation:
- Configuration and parameter assignment of the hardware
- Specification of the communication
- Programming
- Testing, commissioning, and servicing with operational/diagnostic functions
- Documentation
- Creation of visualizations for the SIMATIC Basic Panels using the integrated WinCC Basic software.
- Visualization solutions for PCs and other panels can also be created with other WinCC software packages

Support is provided for all functions in a comprehensive online help system.
3. Block types for SIMATIC S7-300

The program for SIMATIC S7-300 is written in so-called blocks. The Main [OB1] organization block is already included as standard. This block represents the interface to the operating system of the CPU and is automatically called and cyclically processed by this operating system. For extensive control tasks, the program is divided into small, manageable program blocks sorted according to functions. These blocks are then called from organization blocks. At the block end, the program jumps back to the calling organization block - specifically, to the line directly after the call.

3.1 Linear programming

With linear programming, the instructions are stored in a block and processed in the order in which they are stored in the program memory. When the program end (block end) is reached, the program execution starts over from the beginning.

This is called cyclic processing. The time required by a device to process all instructions once is called the cycle time. Linear program processing is used only for simple, less extensive controllers and can be implemented in a single OB.

![Linear Programming Diagram](image-url)
3.2 Structured programming

For extensive control tasks, the program is divided into small, manageable program blocks sorted according to functions. The advantage of this approach is that it is possible to individually test program sections and to combine them once they have been found to work properly in order to produce an overall function.

The program blocks must be called from the higher-level block. If the block end (BE) has been detected, the execution of the program continues in the calling block in the line after the call.

- **OB** = Organization block
- **FB** = Function block
- **FC** = Function
- **DB** = Data block

Global DB for all FBs FCs OBs
Instance DB Local data only FB1
3.2. User blocks for SIMATIC S7-300

The following user blocks are available for structured programming:

- **OB (Organization block):**
  An OB is called cyclically by the operating system and thus forms the interface between the user program and the operating system. In this OB, block call commands are used to inform the PLC's sequence control which program blocks it is to process.

- **FB (function block):**
  The FB requires an assigned memory area for each call (instance). When an FB is called, a data block (DB) can be assigned to it as an instance DB, for example. The data in this instance DB is then accessed using the tags of the FB. Various memory areas must be assigned to an FB if it will be called multiple times. It is also possible to call other FBs and FCs within a function block.

- **FC (function):**
  A function does not have an assigned memory area. The local data of a function is lost after the function is processed. It is also possible to call other FBs and FCs within a function.

- **DB (data block):**
  DBs are used to provide memory space for data tags. There are two types of data blocks. Global DBs where all OBs, FBs, and FCs can read the saved data and themselves write data to the DB, and instance DBs that are assigned to a particular FB.

**Note:** If only internal tags were used when programming FCs and FBs, these can be used multiple times in the form of standard blocks. They can then be called as many times as required, whereby a memory area – a so-called instance (e.g., a DB) – must be assigned to the FBs for each call.
3.2.1 Organization blocks

Organization blocks (OBs) form the interface between the operating system and the user program. They are called from the operating system and control the following operations:

- Startup characteristics of the automation system
- Cyclic program execution
- Interrupt-driven program execution
- Error handling

You can program the organization blocks as you wish and thus determine the behavior of the CPU.

Various options are available for using organization blocks in your program:

- **Startup OB, cycle OB, time error OB, and diagnostics OB:**
  It is easy to insert and program these organization blocks in your project. You do not have to call or assign parameters to these organization blocks.

- **Hardware interrupt OB and cyclic interrupt OB:**
  After these organization blocks have been inserted into your program, you need to assign parameters to them. Hardware interrupt OBs can also be attached to an event during run time using the ATTACH instruction or detached again using DETACH.

- **Time-delay interrupt OB:**
  You can insert and program the time-delay interrupt OB in your project. You must additionally call it in the user program using the SRT_DINT instruction. Parameter assignment is not necessary.

**Start information**

At the start of some organization blocks, the operating system outputs information that can be evaluated in the user program.

This can be very helpful for error diagnostics, in particular. The descriptions of the organization blocks contain information on whether information is output and, if so, what information.
3.2.2 Functions

A function contains a program that is executed whenever the function is called from another code block.

Functions (FCs) are code blocks without memory. After the function has been processed, the data in the temporary tags is lost. Functions can use global data blocks in order to save data permanently.

Functions can be used, for example, for the following purposes:
- To return function values to the calling block, e.g., for mathematical functions
- To execute technological functions, e.g., individual controls using binary logic

A function can also be called several times at different points in a program. This facilitates the programming of frequently recurring complex functions.

3.2.3 Function blocks

Function blocks contain subroutines that are executed whenever a function block is called by another code block.

Function blocks are code blocks that store their values permanently in instance data blocks, so that they remain available even after the block has been processed. They store their input, output, and in/out parameters permanently in instance data blocks. Consequently, these parameters are still available after the block has been processed. For this reason, they are also referred to as blocks with "memory".

Function blocks are used for tasks that cannot be implemented with functions:
- In all cases when timers and counters are required in the blocks (see M3 module)
- In all cases when information must be saved in the program. For example: preselection of the operating mode with a button.

A function block can also be called several times at different points in a program. This facilitates the programming of frequently recurring complex functions.

Instances of function blocks

A call of a function block is referred to as an instance. Each instance of a function block is assigned a memory area that contains the data that the function block uses. This memory is made available by data blocks created automatically by the software. It is also possible to provide memory for multiple instances in a data block in the form of a multi-instance.
3.2.4 Data blocks

In contrast to code blocks, data blocks contain no instructions. Rather, they serve as memory for user data. Data blocks thus contain variable data used by the user program.

**Global data blocks** store data that can be used by all other blocks. The maximum size of data blocks varies depending on the CPU. You can define the structure of global data blocks as required.

Application examples are:
- Saving of information about a storage system. “Which product is located where?”
- Saving of recipes for particular products.

Every function block, every function, and every organization block can read the data from a global data block or write data to a global data block. This data is retained in the data block, even when the data block is exited.

The call of a function block is referred to as an instance. An **instance data block** is assigned to every call of a function block that transfers parameters. This instance data block serves as a data memory. It stores the actual parameters and the static data of the function block.

The maximum size of instance data blocks varies depending on the CPU. The tags declared in the function block determine the structure of the instance data block.

A global data block and an instance data block can be open at the same time.
4. Example task: Function block for belt control

When blocks that will function as a sort of "black box" in any program are to be created, they must be programmed using tags. The following rule applies: absolutely addressed inputs/outputs, bit memory, etc. are not permitted to be used in these blocks. Only tags and constants may be used within the block.

In the following example, a function block with tag declaration containing a mode-dependent belt control system is to be created.

In this case, the 'S1' button will enable selection of 'manual' mode and the 'S2' button will enable selection of 'automatic' mode.
In 'manual' mode, the motor will be switched on as long as the 'S3' button is actuated, during which time the 'S4' button must not be actuated.
In 'Automatic' mode, the 'S3' button will switch on the belt motor and the 'S4' button (NC) will switch off the belt motor.

Assignment list:

<table>
<thead>
<tr>
<th>Address</th>
<th>Symbol</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>%I 0.0</td>
<td>S1</td>
<td>Button operation mode manual S1 (NO)</td>
</tr>
<tr>
<td>%I 0.1</td>
<td>S2</td>
<td>Button operation mode automatic S2 (NO)</td>
</tr>
<tr>
<td>%I 0.2</td>
<td>S3</td>
<td>ON button S3 (NO)</td>
</tr>
<tr>
<td>%I 0.3</td>
<td>S4</td>
<td>OFF button S4 (NC)</td>
</tr>
<tr>
<td>%Q 0.2</td>
<td>M01</td>
<td>Belt motor M01</td>
</tr>
</tbody>
</table>

Note: Here, the OFF button S4 is implemented as a normally closed contact in order to ensure wire break protection. This means that the plant will stop automatically in the event of a wire break at this button. Otherwise, it would no longer be possible to stop the plant in the event of a wire break. For this reason, all STOP buttons, OFF buttons, and switches must always be implemented as normally closed contacts in the control technology.
5. **Programming of the belt control for SIMATIC S7-300**

   The ‘**Totally Integrated Automation Portal**’ software is used for project management and programming.

   Components such as control, visualization, and networking of the automation solution are created, assigned parameters, and programmed here using a standard interface. Online tools are available for the error diagnostics.

   The following steps enable you to create a project for the SIMATIC S7-300 and to program the solution for the task:

   1. The central tool is the ‘**Totally Integrated Automation Portal**’, which is opened here with a double-click. (→ TIA Portal V11)
2. Programs for the SIMATIC S7-300 are managed in projects. Start by creating a project in the portal view (→ Create new project → FB_Band → Create).
3. ‘First steps’ for configuring are now suggested. We want to start with ‘Configure a device’.
(→ First steps → Configure a device)
4. The next step is to ‘Add new device’ with device name ‘Program_belt’. Choose the ‘CPU 315F-2 PN/DP’ with the appropriate order number from the catalog.

(→ Add new device → Program_belt → CPU 315F-2 PN/DP → 6ES7 …… → Add)
5. The software now switches automatically to the project view containing the opened hardware configuration. Additional modules can now be added from the hardware catalog (on the right!) and the addresses of the inputs/outputs can be set in the ‘Device overview’. Here, there is one I/O module with 16 inputs (addresses %I0.0 - %I1.7) and 16 outputs (addresses %Q0.0 - %Q1.7). (→ Device overview → DI/DO → DI16/DO16 x 24V/0.5A → 6ES7 323-1BL00-0AA0 → I address: 0…1 → Q address: 0…1)
6. To ensure that the software will access the correct CPU later, the IP address and the subnet mask of the CPU must be set. (→ Properties → General → PROFINET interface [X2] → Ethernet addresses → IP protocol → IP address: 192.168.0.1 → Subnet mask: 255.255.255.0) (see also: SCE_EN_030-010 module on setting the programming interface.)
7. Because modern programming uses tags and not absolute addresses, the **global PLC tags** must be defined here.

These global PLC tags are descriptive names with a comment for each input and output used in the program. The global PLC tags can then be accessed later during programming via their names. These global tags can be used in all blocks anywhere in the program.

In the project tree, select ‘Program_belt [CPU 315F-2 PN/DP]’ and then ‘PLC tags’. Double-click to open the ‘PLC tags’ table and enter the names for the inputs and outputs as shown below. (→ Program_belt [CPU315F-2 PN/DP] → PLC tags → PLC tags)
8. To create function block FB1, select ‘Program_belt [CPU 315F-2 PN/DP]’ and then ‘Program blocks’ in the project tree. Then, double-click ‘Add new block’. (→ Program_belt [CPU315F-2 PN/DP] → Program blocks → Add new block)
9. Select ‘Function block (FB)’ and assign the name ‘Belt’. Specify the ‘FBD’ function block diagram as programming language. The numbering will be automatic. Since this FB1 will be called using its symbolic name later, the number does not play an important role in any case. Click ‘OK’ to accept your entries.

(→ Function block (FB1) → Belt → FBD → OK)
10. The ‘Belt [FB1]’ block is then opened automatically. The interface of the block must be declared before the program can be written. In the interface declaration, the local tags known only in this block are defined.

The tags are divided into two groups:

- **Block parameters** that form the block interface for the call in the program.

<table>
<thead>
<tr>
<th>Type</th>
<th>Designation</th>
<th>Function</th>
<th>Available in</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input parameters</td>
<td>Input</td>
<td>Parameters whose values are read by the block.</td>
<td>Functions, function blocks, and some types of organization blocks</td>
</tr>
<tr>
<td>Output parameters</td>
<td>Output</td>
<td>Parameters whose values are written by the block.</td>
<td>Functions and function blocks</td>
</tr>
<tr>
<td>In/Out parameters</td>
<td>InOut</td>
<td>A parameter whose value is read by the block when it is called and is written back by the block to the same parameter after it is processed.</td>
<td>Functions and function blocks</td>
</tr>
</tbody>
</table>

- **Local data** that is used for saving intermediate results.

<table>
<thead>
<tr>
<th>Type</th>
<th>Designation</th>
<th>Function</th>
<th>Available in</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temporary local data</td>
<td>Temp</td>
<td>Tags that are used to store temporary intermediate results. Temporary data is retained for only one cycle.</td>
<td>Functions, function blocks, and organization blocks</td>
</tr>
<tr>
<td>Static local data</td>
<td>Static</td>
<td>Tags that are used for saving static intermediate results in the instance data block. Static data is retained until it is overwritten, which may be after several cycles.</td>
<td>Function blocks</td>
</tr>
</tbody>
</table>
11. The following tags are required in our example for declaration of the local tags.

**Input:**
- **man**
  - The button for manual mode selection is read in here (normally open)
- **auto**
  - The button for automatic mode selection is read in here (normally open)
- **ON**
  - The ON button is read in here (normally open)
- **OFF**
  - The OFF button is read in here (normally closed)

**Output:**
- **motor**
  - Here, a state is written to the belt motor output

**Static (only available in function blocks FB):**
- **automan**
  - The preselected operating mode is saved here
- **motorauto**
  - If the belt motor was started in automatic mode, this is saved here.

In this case, all of the tags are ‘Bool’ type tags. That is, they are binary tags that can only have the state ‘0’ (false) or ‘1’ (true).

The important thing in this example is that the state must be saved for an extended period of time in both the ‘automan’ and ‘motorauto’ tags. For this reason, the ‘Static’ tag type must be used here. This tag type is also only available in a function block FB.

All local tags should also be provided with a sufficiently descriptive comment for better understanding.
12. Once the local tags have been declared, the program can be entered using the tag names. (Tags are marked by the ‘#’ symbol.) This could appear as follows for our FBD example.

- **Block title:** Control belt conveyor
  - Comment:

- **Network 1:** Accu Operation mode Automatic/Manual selected
  - Comment:
  - Symbols:
    - SR
    - #auto → S
    - #manual → R

- **Network 2:** Accu Control Motor in Automatic
  - Comment:
  - Symbols:
    - #motor
    - #motor
    - SR
    - #motor
    - #motor

- **Network 3:** Control Motor belt conveyor
  - Comment:
  - Symbols:
    - #motor
    - #motor
    - SR
    - #motor
    - #motor
13. The next step is to select the 'Properties' of the cyclically processed 'Main [OB1]' block. Properties of the blocks can be changed. (→ Main [OB1] → Properties)
14. For the ‘Language’, choose the ‘FBD’ function block diagram programming language. (→ FBD → OK)
15. The "Belt" block must now be called from the Main [OB1] program block. Otherwise, the block would not be processed at all. Double-click 'Main [OB1]' to open this block. (→ Main [OB1])
16. The “Belt [FB1]” block can then be moved to Network 1 of the Main [OB1] block using a drag-and-drop operation. Don't forget to document the networks in the Main [OB1] block. (→ Belt [FB1])
17. Because we are dealing with a function block here, memory must be provided for it. Data blocks are available as memory in the SIMATIC S7-300. A data block assigned in this way is called an instance data block. The specification and creation of this instance data block will be ‘Automatic’ here. (→ Automatic → OK)

**Note:** You will find other examples for working with and creating instances in the SCE_EN_030-020 module.
18. Now, interconnect the input tags and the output tag with the PLC tags shown here in OB1.

Then, click 

![Save project](image)

to save the project. (→ “S1” → “S2” → “S3” → “S4” → “M01” → ...)
19. To load your entire program to the CPU, first select the 'Program_belt' folder and then click the Download to device icon. (→ Program_belt → )
20. In the following dialog, select ‘PN/IE’ as the PG/PC interface type and then a suitable network card as the PG/PC interface. After a ‘Refresh’ of the accessible devices, you should see your ‘CPU 315F-2 PN/DP’ with address 192.168.0.1 and be able to select this CPU as the target device. Then, click ‘Load’. (→ Type of the PG/PC interface: PN/IE → PG/PC interface: …… → Refresh → CPU 315F-2 PN/DP → Load)

**Note:** Details on setting the PG/PC interface can be found in the SCE_EN_020-010 module.
21. Then, click ‘Load’ again. The status of the load operation will be displayed in a window. (→ Load)

![Load preview](image)

22. The successful load result is now displayed in a window. To place the CPU 315F-2 PN/DP back in RUN mode, click ‘Start all’ and then ‘Finish’. (→ Start all → Finish)

![Load results](image)
23. Click the Monitoring on/off icon to monitor both the state of the input and output tags on the "Belt" block when testing the program and the program execution in the "Belt" block. (→ Belt [FB1] → )
24. Because our “Belt” block was created according to the rules for standard blocks (no use of global tags within the block!), it can now be used and called as many times as required.

An expanded PLC tag table with inputs/outputs for two belts is shown here.

<table>
<thead>
<tr>
<th>Name</th>
<th>Data type</th>
<th>Address</th>
<th>Retain</th>
<th>Visible</th>
<th>Access</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>SI_belt_conveyor1</td>
<td>Bool</td>
<td>%0.0</td>
<td></td>
<td></td>
<td></td>
<td>BC1 Button Selection mode manual S1 ( )</td>
</tr>
<tr>
<td>S2_belt_conveyor1</td>
<td>Bool</td>
<td>%0.1</td>
<td></td>
<td></td>
<td></td>
<td>BC1 Button Selection mode automatic S...</td>
</tr>
<tr>
<td>S3_belt_conveyor1</td>
<td>Bool</td>
<td>%0.2</td>
<td></td>
<td></td>
<td></td>
<td>BC1 Button ON S3 ( EO)</td>
</tr>
<tr>
<td>S4_belt_conveyor1</td>
<td>Bool</td>
<td>%0.3</td>
<td></td>
<td></td>
<td></td>
<td>BC1 Button OFF S4 ( IC)</td>
</tr>
<tr>
<td>M01_belt_conveyor1</td>
<td>Bool</td>
<td>%0.2</td>
<td></td>
<td></td>
<td></td>
<td>BC1 belt conveyor motor M01</td>
</tr>
<tr>
<td>SI_belt_conveyor2</td>
<td>Bool</td>
<td>%0.4</td>
<td></td>
<td></td>
<td></td>
<td>BC2 Button Selection mode manual S2 ( )</td>
</tr>
<tr>
<td>S2_belt_conveyor2</td>
<td>Bool</td>
<td>%0.5</td>
<td></td>
<td></td>
<td></td>
<td>BC2 Button Selection mode automatic S...</td>
</tr>
<tr>
<td>S3_belt_conveyor2</td>
<td>Bool</td>
<td>%0.6</td>
<td></td>
<td></td>
<td></td>
<td>BC2 Button ON S3 ( EO)</td>
</tr>
<tr>
<td>S4_belt_conveyor2</td>
<td>Bool</td>
<td>%0.7</td>
<td></td>
<td></td>
<td></td>
<td>BC2 Button OFF S4 ( IC)</td>
</tr>
<tr>
<td>M01_belt_conveyor2</td>
<td>Bool</td>
<td>%0.8</td>
<td></td>
<td></td>
<td></td>
<td>BC2 belt conveyor motor M01</td>
</tr>
</tbody>
</table>

<Add more>
25. Now, the "Belt" block can also be called twice in OB1 with different circuitry in each case. A different instance block will be specified for each call.