Function Block Diagram (FBD)

TM241

Perfection in Automation

www.br-automation.com

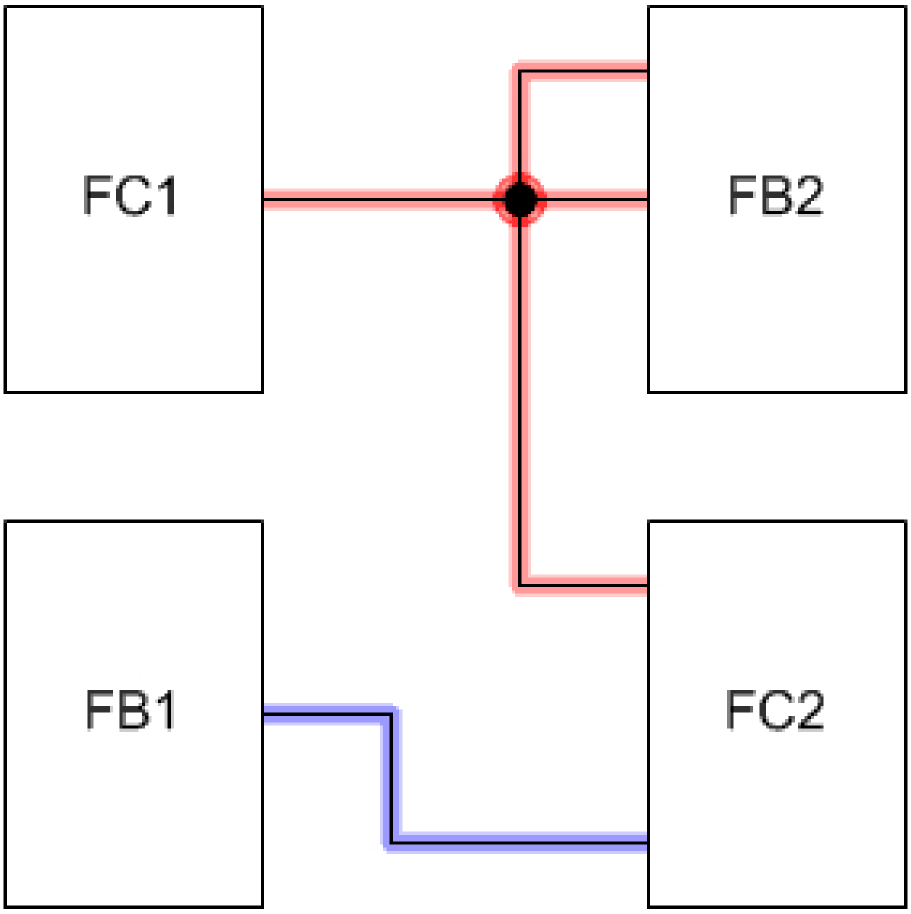
Requirements

Training modules:	TM210 – The Basics of Automation Studio TM223 – Automation Studio Diagnostics
Software	None
Hardware	None

TABLE OF CONTENTS

1 INTRODUCTION

Function Block Diagram (FBD) is a visual programming language that is intuitive and easy to use. Because of this, it is suitable for both experienced and beginning programmers.



Overview of programming using function block diagrams

In this training module, you will learn how to best use the Function Block Diagram (FBD) programming language. Examples will be provided to help explain individual functions.

The basic elements of FBD will be described in general terms in this training module. After each explanation, please refer to the links to the Automation Studio online help documentation for information about using the editor.

1.1 Training module objectives

This training module provides an overview of the...

- ... Different possibilities available when programming with Function Block Diagram
- ... Basic elements and how they are used
- ... Different arrangement options in FBD
- ... Various options available for controlling program flow
- ... Diagnostic options

2 FUNCTION BLOCK DIAGRAM

2.1 General information

FBD is a visual programming language defined in the IEC 61131-3 standard.

The pre-programmed functions and function blocks it contains make it possible to solve many different kinds of tasks.

Function Block Diagram has the following features:

- Visual programming
- Simple and clear programming
- Self-explanatory
- Clear diagnostic functions
- Standardized in IEC 61131-3

The Function Block Diagram programming language provided with Automation Studio offers the following possibilities:

- Operation of digital and analog inputs/outputs
- Logical operations
- Logical comparison expressions
- Arithmetic operations
- Calling function blocks and functions
- Control of program execution order using jumps
- Calling IEC actions with action blocks
- Diagnostic tools

Programming \ Programs \ Function Block Diagram (FBD)

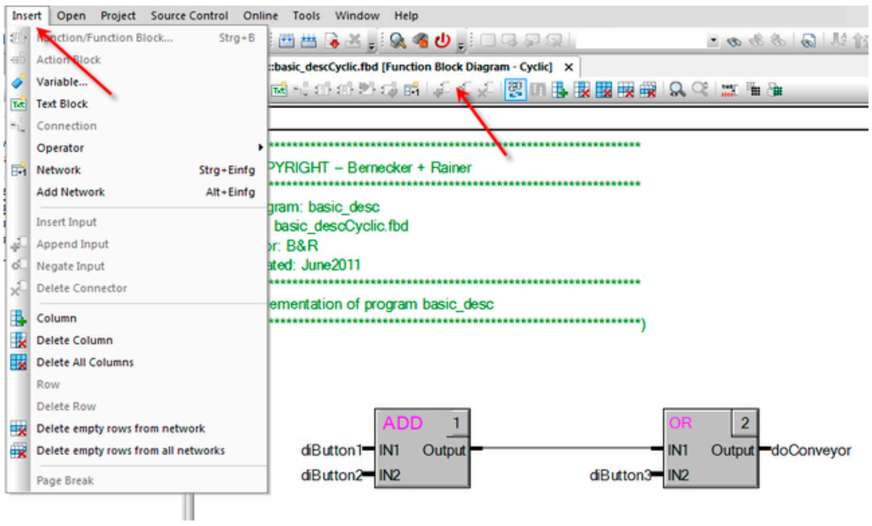
Programming \ Libraries \ IEC 61131-3 functions

2.2 Editor functions in Function Block Diagram

The function block diagram editor is a visual editor. Blocks are placed in networks and connected to each other. The editor supports automatic layout for the networks.

The editor is operated using the following elements:

- Main menu: Insert
- Editor toolbar of the Function Block Diagram editor
- Shortcut menu for function block diagrams
- Shortcut menu for blocks



Overview of the Function Block Diagram editor

Powerflow can be used to diagnose function block diagrams. (see Diagnostics – Powerflow)

Toolbar in the function block diagram editor

Connections

To connect two or more block contacts, the contacts themselves must first be selected. The connection can then be performed using the toolbar. Another option is to hold down the **<CTRL>** key while selecting the contacts; once released, they will be connected together.

Programming \ Editors \ Graphic editors \ Function Block Diagram editor

Programming \ Editors \ Graphic editors \ Function Block Diagram editor \ Toolbar

Project management \ The workspace \ Toolbars \ Zoom

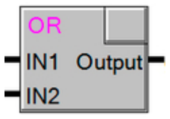
3 FUNCTIONS AND FUNCTION BLOCKS

There are two types of blocks that can be used in FBD:

- Functions
- Function blocks

3.1 Functions

A function is a program organization unit (POU) that can return only a single value. Any number of parameters can be passed. The return value is passed back immediately after the function is called.



Function with two parameters

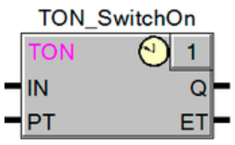
Programming \ Functions and function blocks \ Functions

3.2 Function blocks

A function block is a program organization unit (POU) that can return one or more values. Therefore, it can have one or more inputs and outputs.

When using a function block, an instance variable must be declared first. This is essentially a data structure that contains all of the parameters used by the function block.

Individual function block instances can be called with various parameters.



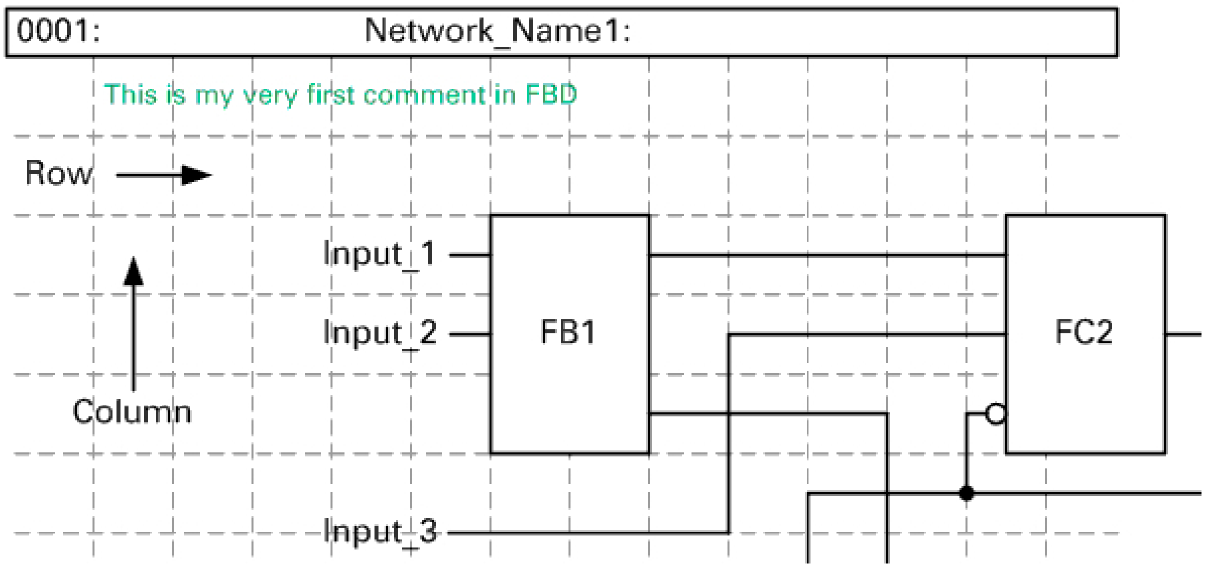
Function with two return values and instance name

Programming \ Functions and function blocks \ Function blocks

4 NETWORKS

4.1 General

A network is a cohesive part of the program where functions, blocks and actions are connected together.



Basic structure of a network

Comments can be added to describe networks. There can be any number of comments at any length in a network, and they can be inserted at any location whatsoever.

When a function block diagram program is being created, each network is assigned a unique network number. This is used to identify the network. Network names can also be changed as needed.

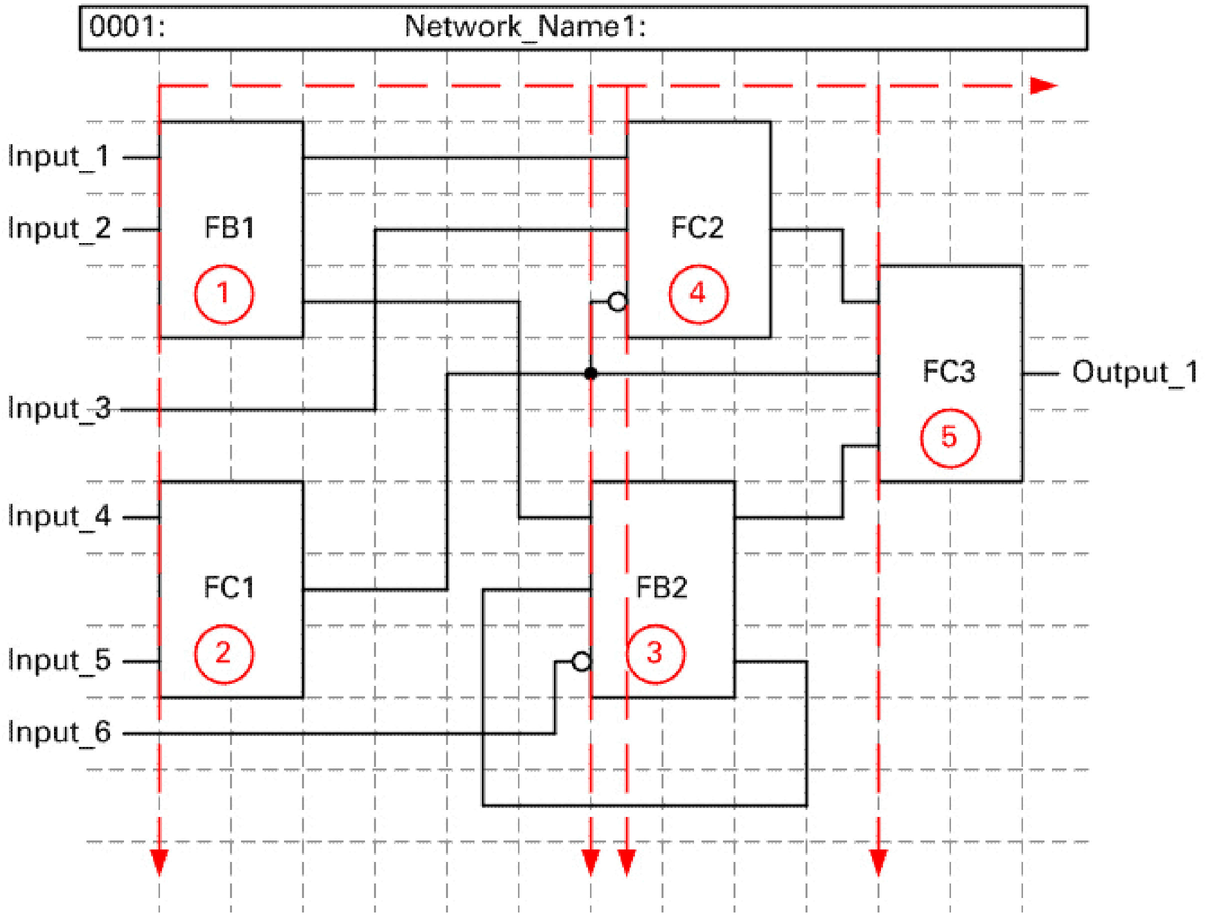
If it is not possible to add blocks in the network, it is probably because the automatic layout feature has been enabled. This can be disabled in the editor's toolbar.

Programming \ Programs \ Function Block Diagram (FBD) \ Network

Programming \ Editors \ Graphic editors \ Function Block Diagram editor \ Networks

4.2 Order of execution

When a network is created and blocks are placed into it, the position of these individual components results in a certain order of execution. This order determines the function of the network. Execution in a network takes place by column from top to bottom starting with the column on the left. This means that in the image below, "FB2" is executed before "FC2" since it is positioned further left.



Order of execution within a network

The blocks contained in a network are numbered automatically by the Function Block Diagram editor. These numbers can be changed manually by clicking and editing them.

It is important to make sure that the signal never flows backwards since this causes errors when the compiler attempts to interpret the function block diagram. The following warning is output:

Error 1366: Result used before it has been calculated.

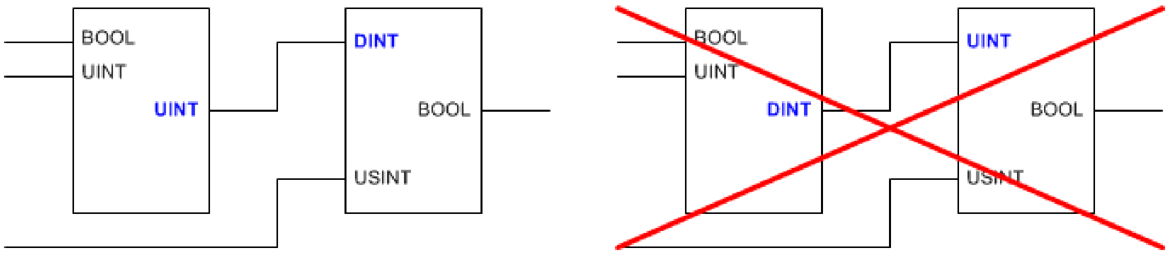
Programming \ Programs \ Function Block Diagram (FBD) \ Execution order

Programming \ Editors \ Graphic editors \ Function Block Diagram editor \ Execution numbers

4.3 Connections

Individual blocks are connected together in a network.

Connections can only exist between identical data types, except in cases when the target data type is larger than the original (automatic conversion).

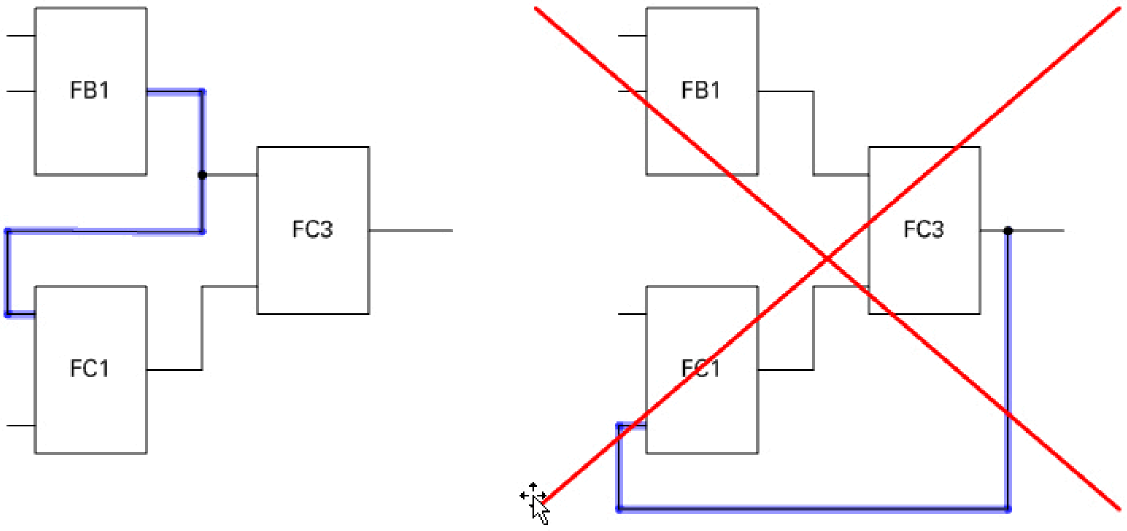


Connections - Data type conversion

The compiler outputs the following error message when attempting to assign unequal data types:

```
Error 1140: Incompatible data types: Cannot convert DINT to UINT
```

Function blocks can only be linked together in the direction in which execution takes place; otherwise, the data flow behavior is undefined.



Connections - Direction

Connections where the signal flows backwards are prevented by the editor.

Programming \ Programs \ Function Block Diagram (FBD) \ Connections
Programming \ Editors \ Graphic editors \ Function Block Diagram editor \ Connections

4.4 Intersections

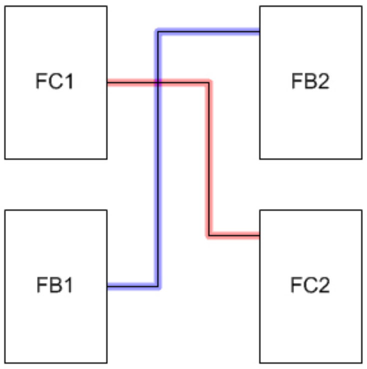
In most cases, networks contain blocks with intersecting connections.

There are two types of intersections:

- Connectionless intersections
- Connected intersections

Connectionless intersections

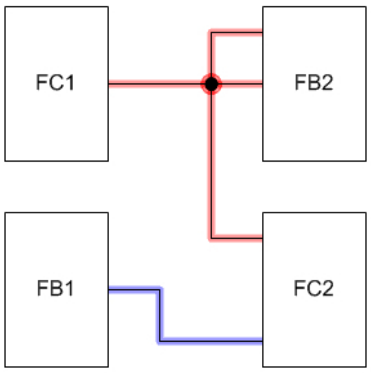
Connectionless intersections are those where one connection simply continues past another. The data flow is not changed by a connectionless intersection.



Connectionless intersection

Connected intersections

Connected intersections change the data flow by passing on the original data to the different connection at the intersection. A connected intersection can be identified by the intersection connection point.



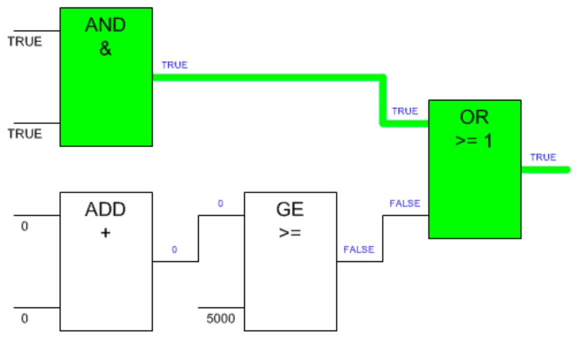
Connected intersection

Programming \ Programs \ Function Block Diagram (FBD) \ Connections
Programming \ Editors \ Graphic editors \ Function Block Diagram editor \ Connections

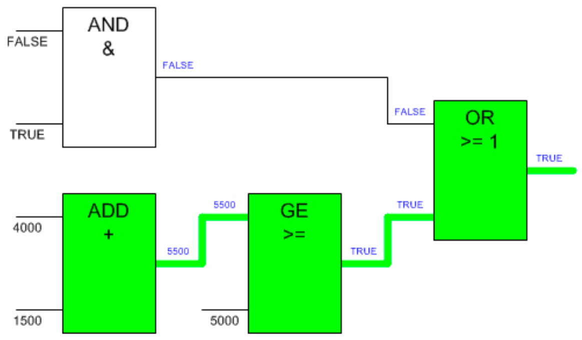
4.5 Diagnostics – Powerflow

Powerflow can be enabled in monitor mode to check the program being configured. With Powerflow, all active connection lines and blocks are highlighted. This makes it relatively easy to diagnose the status of a particular network.

Active elements are highlighted in color.



Powerflow example 1



Powerflow example 2

Diagnostics and service \ Diagnostic tools \ Monitors \ Function Block Diagram editor in monitor mode

Diagnostics and service \ Diagnostic tools \ Monitors \ Programming languages in monitor mode \ Powerflow \ Powerflow in Function Block Diagram

Task: Get to know the Function Block Diagram editor

This task is designed to familiarize yourself with the editor.

1) Insert an ADD block.
2) Declare variables "value1" and "value2" with data type USINT.
3) Connect the two variables with the ADD block.
4) Transfer the program to the controller.
5) Test the results using monitor mode and Powerflow.

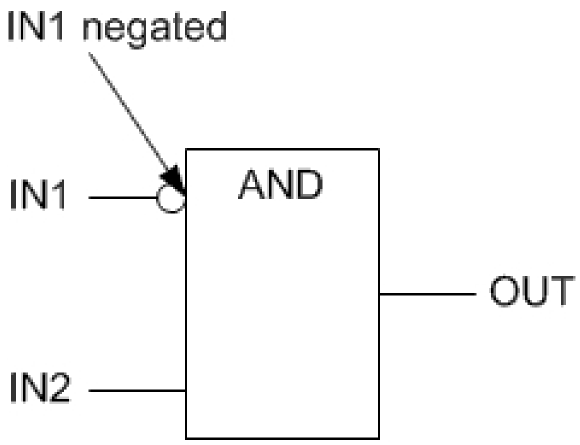
5 LOGIC FUNCTIONS

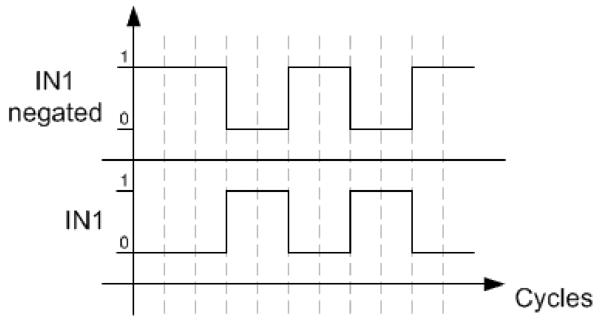
Standard functions inserted directly in the editor are used in Function Block Diagram to configure binary logic operations.

5.1 Negation

In some cases, it is necessary to use the inverse value of a signal on a logic function block. This is where the negation property comes in handy.

Negation is represented by a circle and can be used for all function block inputs whose data type is BOOL.





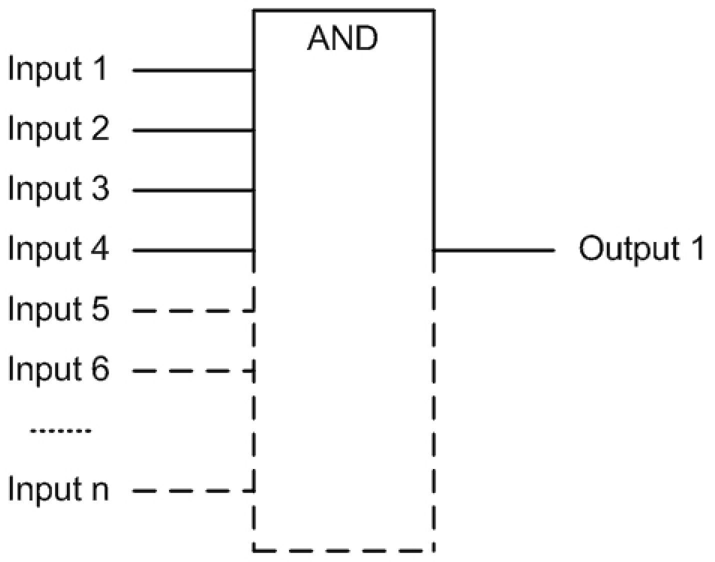
Negation - Status diagram

Negation

Programming \ Editors \ Graphic editors \ Function Block Diagram editor \ Toolbar

5.2 Extensible functions

If a function block requires more inputs to carry out binary operations than are defined in the standard, then the number of inputs can be expanded. Inputs can be easily added or removed in the block's shortcut menu.



Extensible function AND

The IEC 61131-3 standard defines which function blocks have this ability and includes the following functions: ADD, MUL, AND, OR, XOR, MIN, MAX, MUX, CONCAT

Programming \ Programs \ Function Block Diagram (FBD) \ Blocks

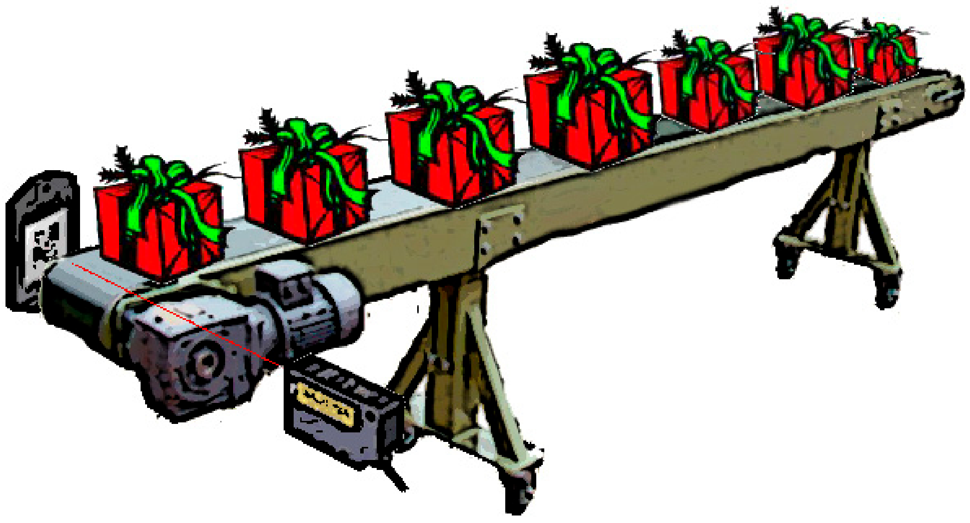
Programming \ Libraries \ IEC 61131-3 functions

Programming \ Editors \ Graphic editors \ Function Block Diagram editor \ Working with blocks

Task: Conveyor belt, Part I

In this training module, we will be creating an application for controlling a conveyor belt in three steps.

In its final state, the program will be able to handle several operating modes while utilizing most of the features available in Function Block Diagram.



Conveyor belt

1) Create the manual operating mode:

   Create a program for manual mode where the command variable **"cmdManual"** can be switched on and off using only a single input, **"gDiSwitchManual"**.

2) Use negation, SET and RESET:

   To complete this task, use negation, set and reset blocks.

Possible solution: Solution - Conveyor belt, Part I

5.3 SET and RESET

SET function

The SET function is a standard function in FBD. It sets an output signal when there is a rising edge on an input signal. The output signal remains set until it is reset, e.g. with the RESET function.

RESET function

The RESET function is also a standard function in FBD. It resets an output signal when there is a rising edge on an input signal.

For both of these functions, the connected data types have to be of type BOOL. Both functions can be inserted from the editor's toolbar.

?

Programming \ Editors \ Graphic editors \ Function Block Diagram editor \ Toolbar

Task: Conveyor belt, Part II

At this point, it is possible to operate the command in manual mode. We will now add a few functions to handle automatic operation and to control the motor (**"gDoMotor"**).

Start the conveyor belt:

- If no material is detected by the sensor at the end of the **"gDiEnd"** conveyor belt.
- If material is detected by the sensor at the end of the conveyor belt and the machine requests more material with the **"gDiMoreMaterial"** digital input.

Stop the conveyor belt:

- If material is detected by the sensor at the end of the conveyor belt and the machine is not requesting any more material.

The digital input "gDiAutoMode" is used to to set the operating mode.

- gDiAutoMode = TRUE: Automatic operation
- gDiAutoMode = FALSE: Manual operation

For safety reasons, it is only possible to change from manual to automatic mode when the motor is switched off.

Possible solution: Solution - Conveyor belt, Part II

6 CONTROLLING THE PROGRAM FLOW

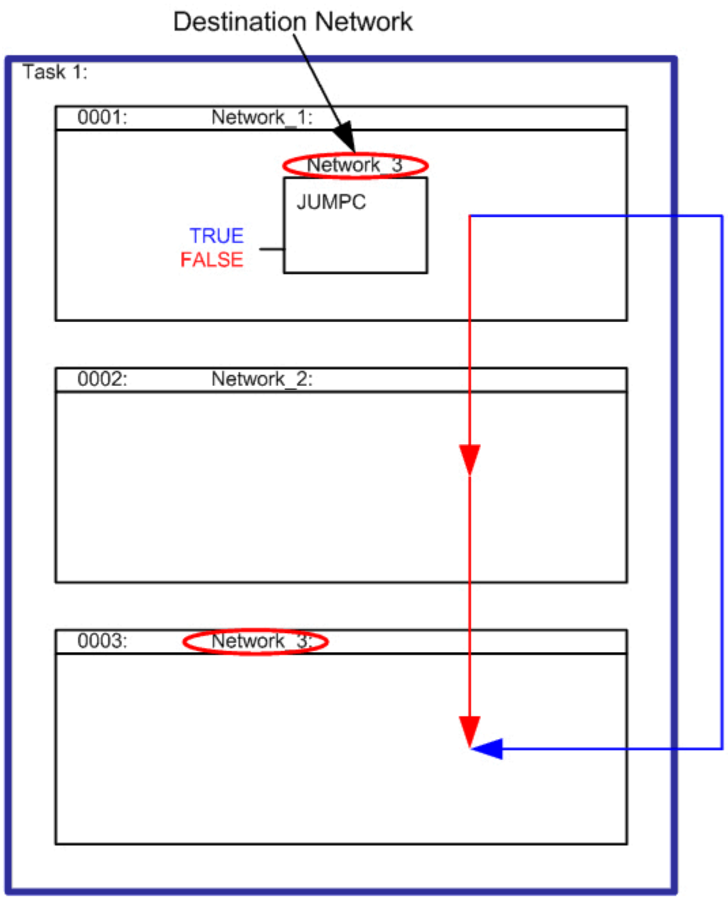
The program flow always moves from the highest to the lowest network (lowest to highest network number). Usually, each network is cycled through. It is also possible to steer the program flow as needed in the event that not all program parts have to be processed. This can be done using the jump and return blocks.

6.1 Jump

Jumps are used to skip from one network to another.

They are only triggered when the logical value TRUE is present on the input. Since a variable or the output of a block can be attached to the input, a jump is also referred to as a "conditional jump".

To define which network to jump to, the name of the target network is added to the function.



Execution with and without a jump

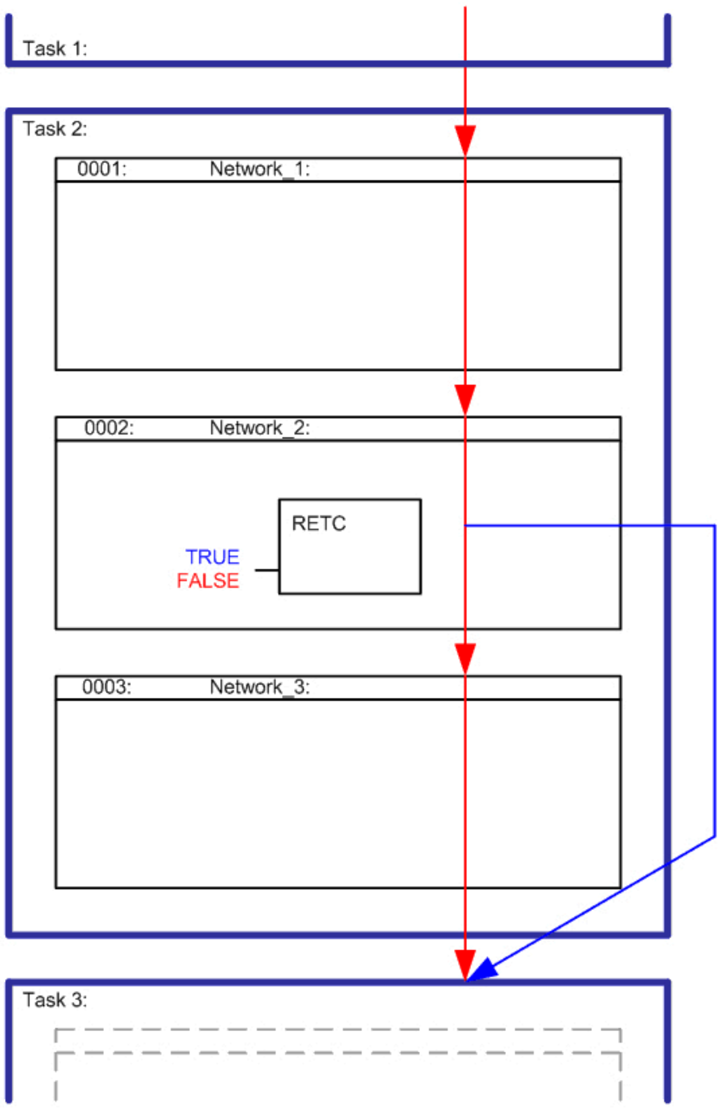
Programming \ Programs \ Function Block Diagram (FBD) \ Jump / Jump return

6.2 Return

A return block will terminate the program wherever it is located, i.e. the underlying networks are no longer called.

The return is only executed when the logical value TRUE is present on the input.

Since a variable or the output of a block can also be attached to the input, a return is also referred to as a "conditional return".



Order of execution with and without a return

Programming \ Programs \ Function Block Diagram (FBD) \ Jump / Jump return

Task: Conveyor belt, Part III

The program can now handle two different operating modes.

Controlling the program flow

Since the conveyor belt can only be operated in one mode at a time, there is always a part of the software that is not needed. This depends on the mode that is selected.

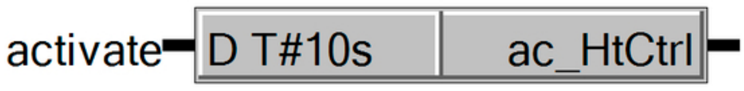
Now expand your existing program so that the parts of the software that are not required are skipped.

Possible solution: Solution - Conveyor belt, Part III

7 ACTION BLOCKS

Action blocks make it possible to call IEC actions.

In addition, qualifiers can be defined to determine how the actions should be called. For example, it is possible to specify that an action should be delayed by a certain amount of time when the block is activated.



Action block with delayed action

The actual content of an IEC action is programmed in a separate program module, e.g. in Structured Text. IEC actions can be used multiple times, therefore making it possible to assign them to several action blocks with different qualifiers.

It is also possible to use Boolean actions. In this case, a variable of data type BOOL becomes TRUE when the action is active.

The following functions are made possible by important qualifiers:

- Calling an action cyclically (N)
- Limiting an action temporally (L)
- Delaying an action (D)
- Setting an action conditionally (S)
- Resetting an action (R)

For a complete list of possible qualifiers, please refer to the help documentation.

Action blocks in Function Block Diagram are executed according to the logic of the "final scan". Actions are always called at least twice.

The cycle in which the action is currently being processed can be seen with the action's instance structure. (see 7.1 "Evaluating the "final scan"")

In addition, disabled actions are called at the end of the program in alphabetical order; then newly enabled actions are called, also in alphabetical order.

Programming \ Actions

Programming \ Programs \ Function Block Diagram (FBD) \ Blocks

Programming \ Actions \ Action block - Qualifiers

Programming \ Actions \ Using actions in the various programming languages

7.1 Evaluating the "final scan"

The "final scan" of an IEC action can be evaluated from the instance structure of the action itself. It is then possible, for example, when using the "P" qualifier to only call a particular program section once despite the actual logic used in the final scan. The subsequent action has been programmed in Structured Text (ST).

Program code

```
ACTION ac_HtCtrl:
    IF  ac_HtCtrl.x AND NOT ac_HtCtrl._x THEN
        (*final scan*)
    ELSE
        OnlyOnce :=  TRUE;
    END_IF ;
END_ACTION
```

Table: Evaluating the "final scan" in Structured Text

Task: Evaluating the final scan

When a button is pressed, it should switch on an output; when pressed again, it should switch the output back off.

For this, an action block with the qualifier "P" can be called. The last scan in the action's program code must be intercepted since otherwise the output would be switched on/off every time a pulse occurs.

8 EXERCISES

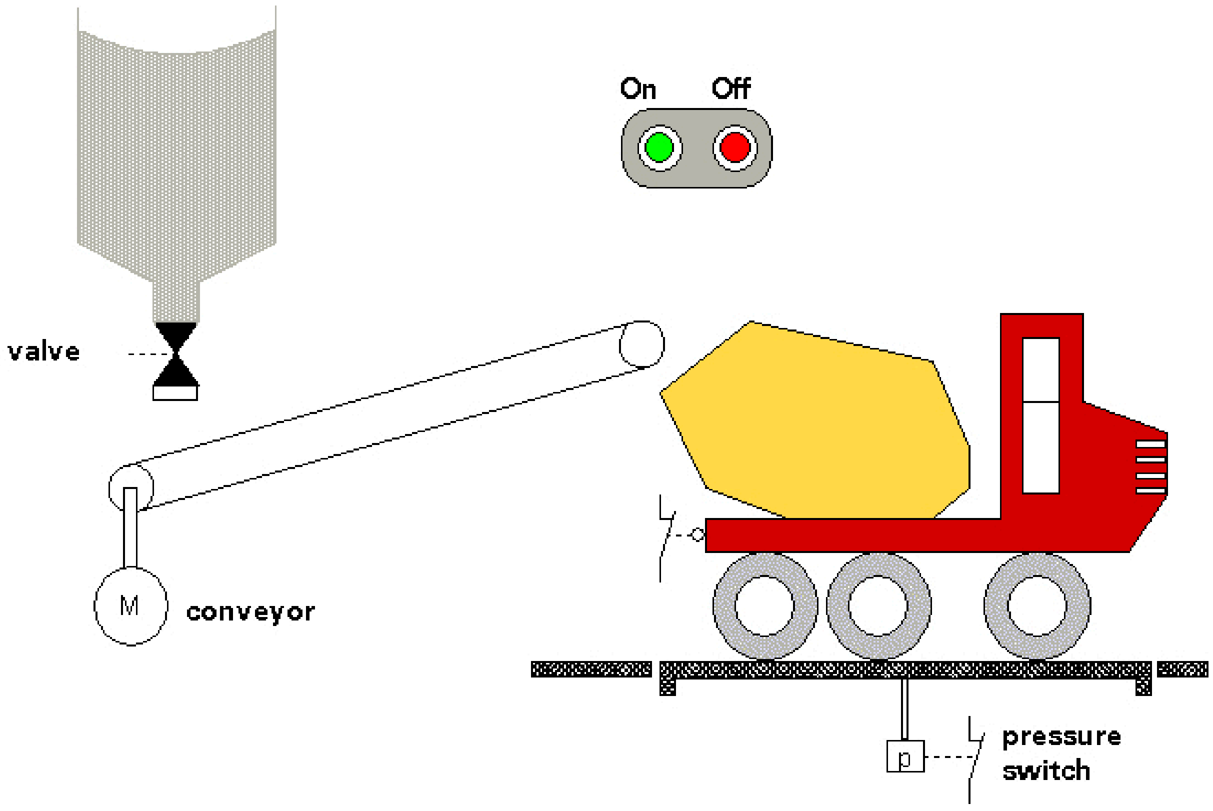
8.1 Concrete filling

In a concrete mixing system, concrete is loaded into a truck via a conveyor belt.

This filling operation is begun by pressing the On button (**btnOn**). However, the hydraulic system controlled by a solenoid valve **(doValve)** cannot be opened until the conveyor has been running for 5 seconds and a truck is located beneath the belt **(diTruck)**.

The solenoid valve is shut off as soon as the total permissible weight of the truck has been reached **(diPressure)**. The conveyor belt should continue to run for an additional 5 seconds, however.

If the off button (**btnOff**) is pressed, then the entire system is switched off immediately. If a malfunction occurs on the conveyor belt (**diConveyorMotorProtection**), then the solenoid valve and the conveyor belt (**doConveyor**) should be shut down immediately. If a malfunction occurs on the solenoid valve (**diValveProtection**), then it is closed immediately, but the belt should continue to run for another 5 seconds.



Schematic illustration of the exercise

Task: Implement the concrete filling system

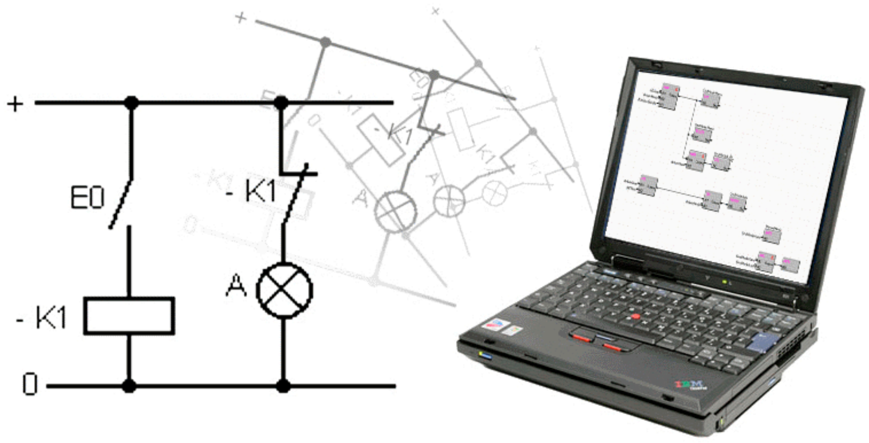
1) Define the individual system functions.
2) Determine the networks.
3) Program the individual networks.

4) Test the specifications using Powerflow and the variable monitor.

One possible solution can be found here: Possible solution - Concrete filling system

9 SUMMARY

Function Block Diagram is a network-oriented visual programming language. It allows function blocks, functions and action blocks to be connected together using lines.



Summary

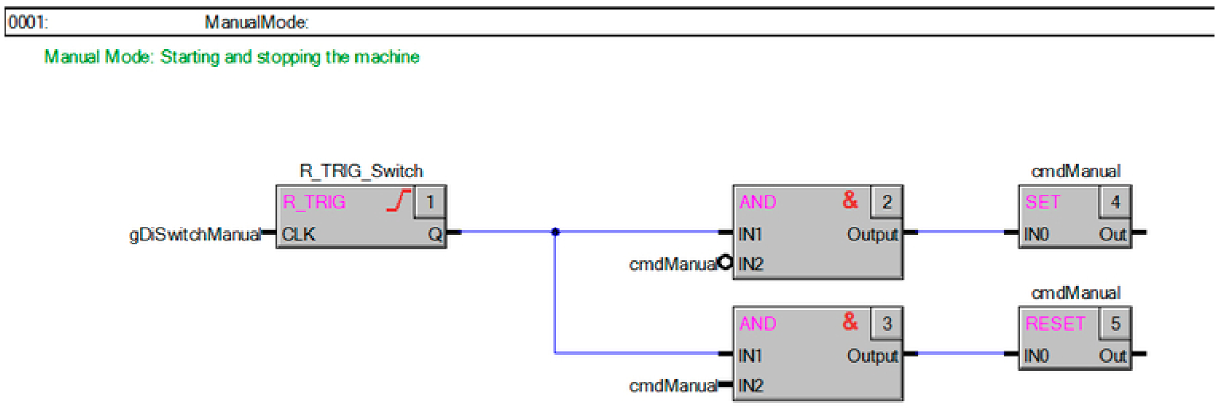
Binary logic can be clearly programmed through the use of SET/RESET blocks as well as AND/OR operations.

Additional program flow control elements as well as the ability to call any function blocks and IEC actions make it possible to configure complex programs and sequences.

10 APPENDIX: SAMPLE SOLUTIONS

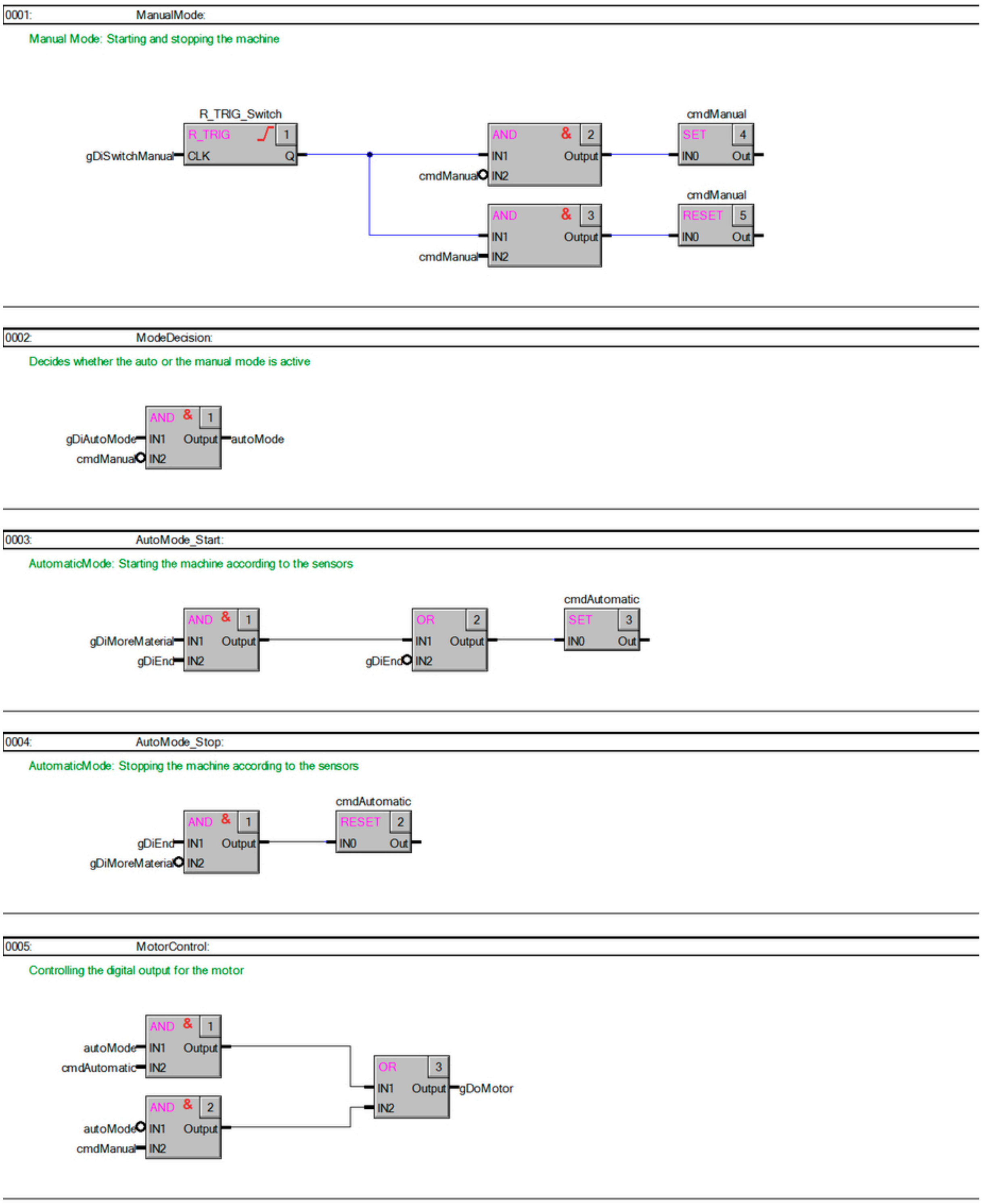
10.1 Solution - Conveyor belt

10.1.1 Solution - Conveyor belt, Part I



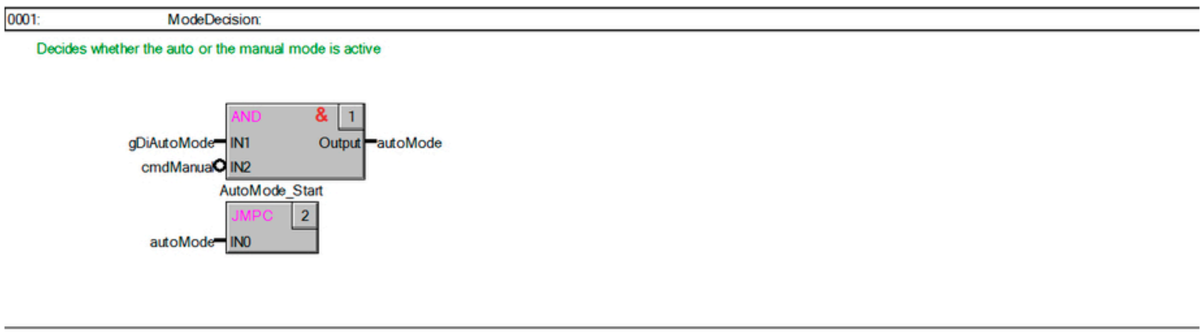
Switching the operating mode

10.1.2 Solution - Conveyor belt, Part II

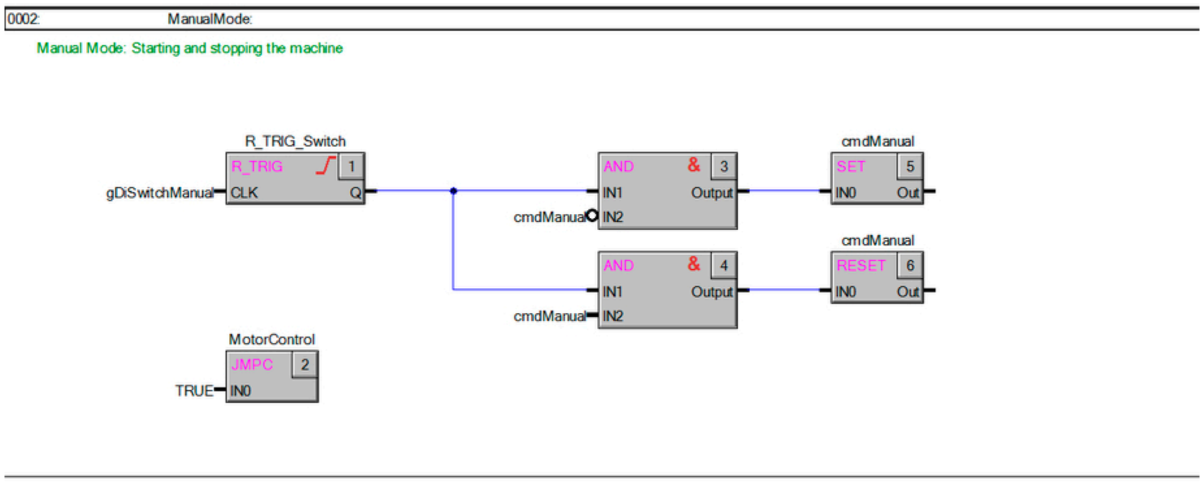


Manual and automatic mode

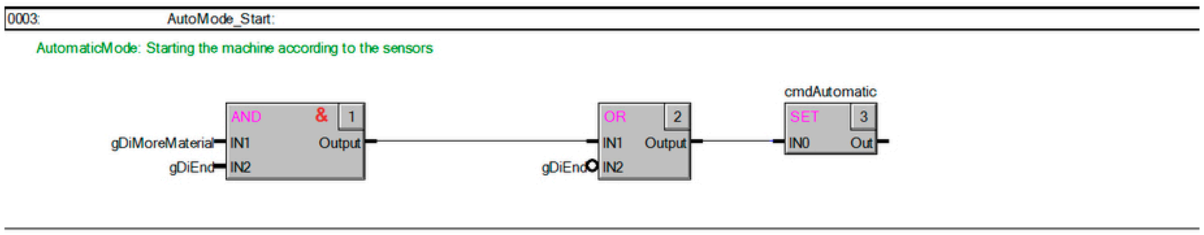
10.1.3Solution - Conveyor belt, Part III


0001: ModeDecision:

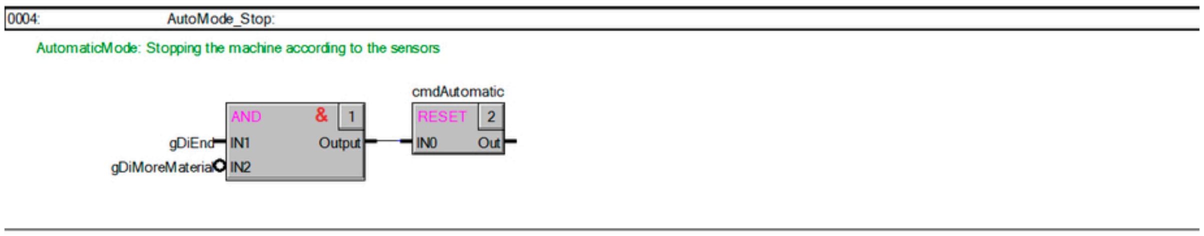
Decides whether the auto or the manual mode is active


0002: ManualMode:

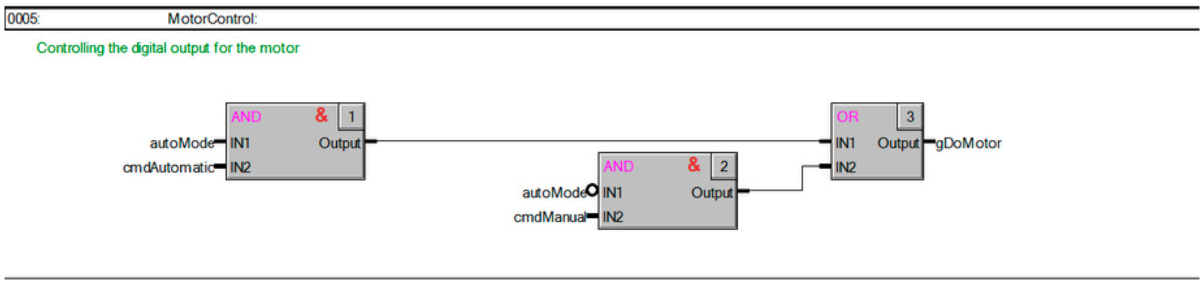
Manual Mode: Starting and stopping the machine


0003: AutoMode_Start:

AutomaticMode: Starting the machine according to the sensors


0004: AutoMode_Stop:

AutomaticMode: Stopping the machine according to the sensors

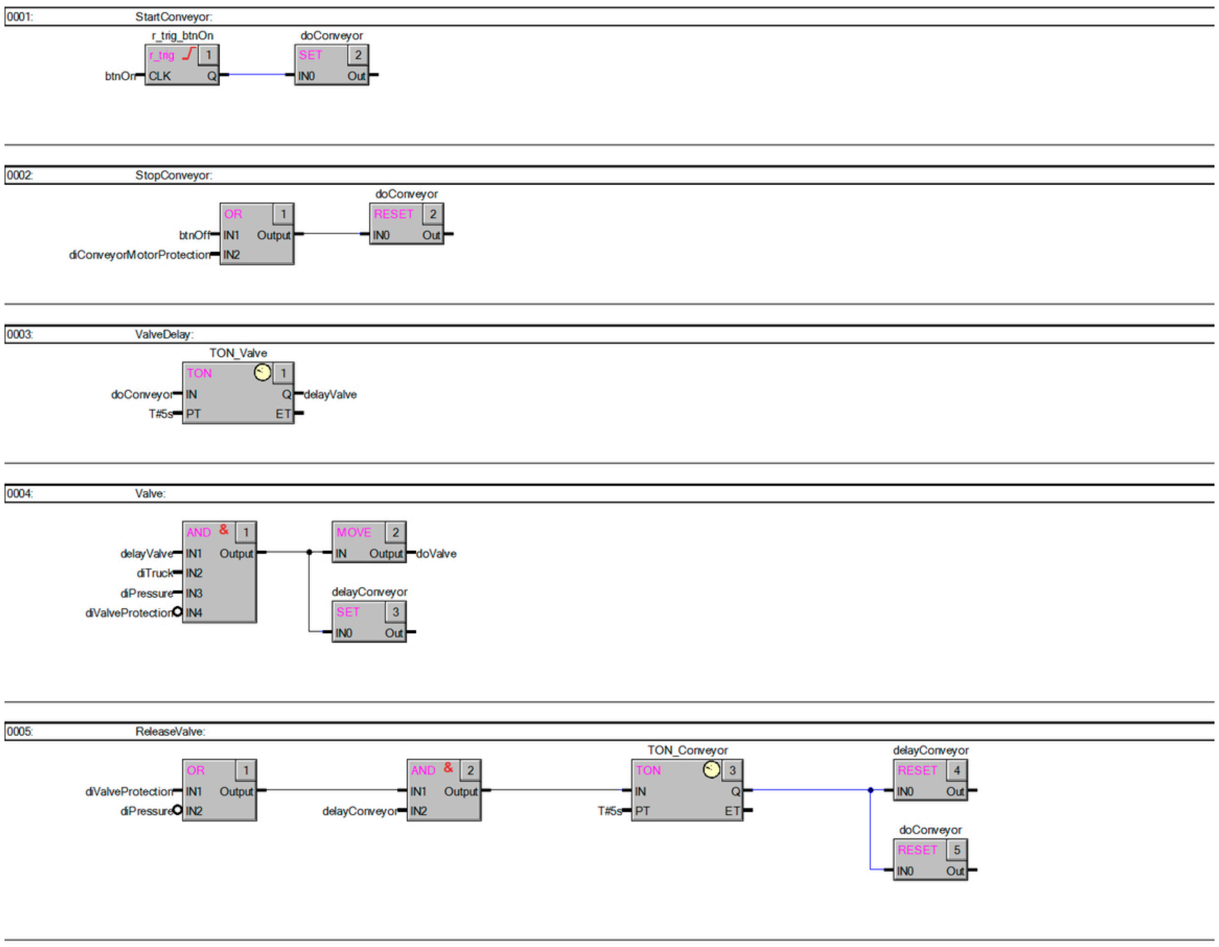

0005: MotorControl:

Controlling the digital output for the motor

Unnecessary networks are skipped.

10.2 Possible solution - Concrete filling system

This exercise can be solved as follows:



Possible solution

TRAINING MODULES

TM000 – eMPTY
TM210 – The Basics of Automation Studio
TM213 – Automation Runtime
TM220 – The Service Technician on the Job
TM223 – Automation Studio Diagnostics
TM230 – Structured Software Generation
TM240 – Ladder Diagram (LD)
TM241 – Function Block Diagram (FBD)
TM242 – Sequential Function Chart (SFC)
TM246 – Structured Text (ST)
TM250 – Memory Management and Data Storage
TM261 – Closed Loop Control with LOOPCONR
TM400 – The Basics of Drive Technology
TM410 – ASiM Basis
TM440 – ASiM Basic Functions
TM441 – ASiM Multi-Axis Functions
TM450 – ACOPOS Control Concept and Adjustment
TM460 – Starting up Motors
TM480 – Basics of Hydraulic Control
TM500 – Basics of Integrated Safety Technology
TM510 – ASiST SafeDESIGNER
TM600 – The Basics of Visualization
TM610 – ASiV Basis
TM640 – ASiV Alarms, Trends and Diagnostics
TM670 – Visual Components Advanced
TM810 – APROL Setup, Configuration and Recovery
TM811 – APROL Runtime System
TM812 – APROL Operator Management
TM813 – APROL XML Queries and Audit Trail
TM830 – APROL Project Engineering
TM890 – The Basics of LINUX