Function Block Diagram (FBD)

TM241

Perfection in Automation

www.br-automation.com

Requirements

Training modules:	TM210 – The Basics of Automation Studio
	TM211 – Automation Studio Online Communication
	TM213 – Automation Runtime
	TM223 – Automation Studio Diagnostics
Software:	None
Hardware:	None

Table of contents

1. INTRODUCTION 4
   1.1 Objectives 5
2. FUNCTION BLOCK DIAGRAM 6
   2.1 General 6
   2.2 Features 6
   2.3 Possibilities 6
3. FUNCTION BLOCKS 7
   3.1 Definition of "function" 7
   3.2 Definition of "function block" 8
4. NETWORKS 10
   4.1 General 10
   4.2 Order of execution 11
   4.3 Connections 12
   4.4 Intersections 13
   4.5 Diagnostics – Signal flow display 14
5. LOGIC 15
   5.1 Negation 15
   5.2 Expandable functions 15
   5.3 Set/Reset 17
6. CONTROLLING THE PROGRAM FLOW 20
   6.1 Jump 20
   6.2 Jump Return 21
7. EXERCISES 23
8. SUMMARY 25
9. APPENDIX 26
   9.1 Functions in accordance to IEC61131-3 26
   9.2 Solutions 30

1. INTRODUCTION

Function Block Diagram is a graphic programming language, favored because of its self-explanatory ease-of-use.

Thanks to its simplicity and clearly organized design, FBD is well-suited for experienced users as well as for beginners.

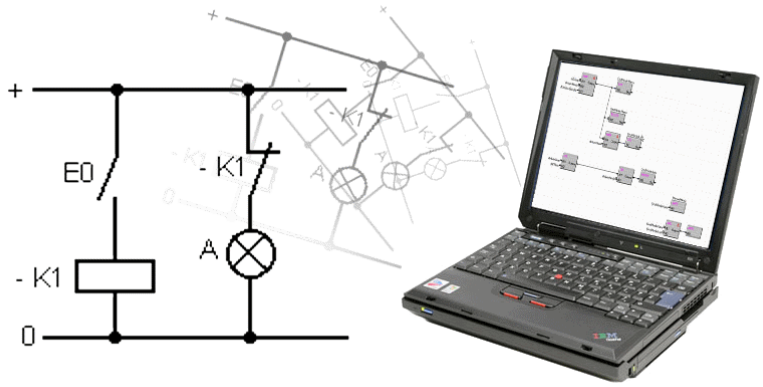


Fig. 1: FBD programming

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

1.1 Objectives

You will get an overview of the possibilities available when programming with Function Block Diagram.

You will learn the fundamental elements of Function Block Diagram and the symbols for logic programming.

You will be able to develop flexible Function Block Diagram programs using the program flow control elements.

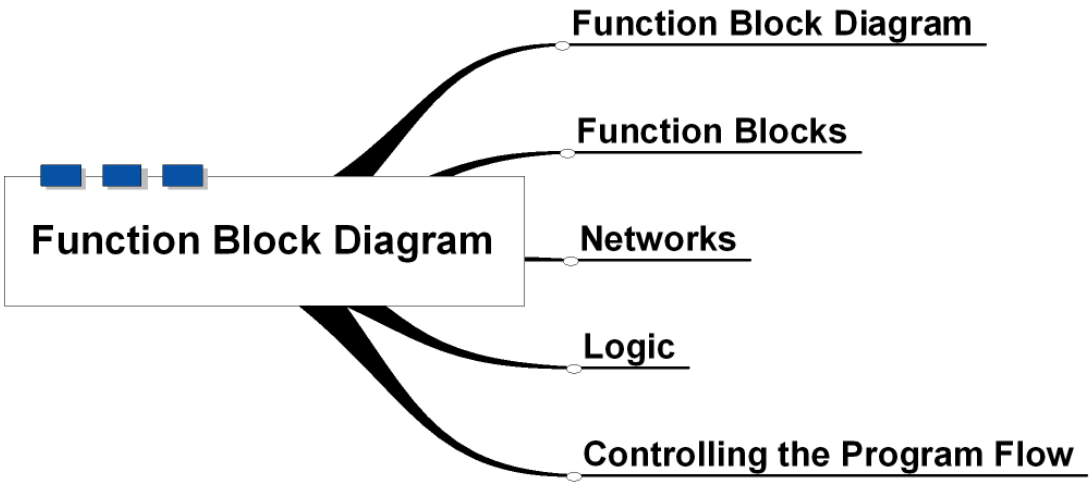


Fig. 2: Objectives

2. FUNCTION BLOCK DIAGRAM

2.1 General

FBD is a graphical programming language defined in the IEC61131-3 standard and adapted by B&R in its entirety.

Pre-programmed functions and function blocks make it possible to solve complex tasks using FBD with minimum previous knowledge of the programming language.

2.2 Features

Function Block Diagram has the following features:

- Graphical programming languages
- Simple and clear programming
- Intuitive to use
- Easy to find errors
- Complies with the IEC 61131-3 standard

2.3 Possibilities

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

- Digital and analog inputs and outputs
- Logic operations
- Logic comparison expressions
- Arithmetic operations
- Function blocks
- Diagnostic tools

3. FUNCTION BLOCKS

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

- Functions (FC)
- Function blocks (FB)

These two types differ in behavior and how they are used.

3.1 Definition of "function"

A function is a program organizational unit which returns exactly one value. Therefore, it has just one output, but can have any number of inputs.

Unlike function blocks, functions **do not have any static memory**. With only a few exceptions (e.g. time and IO – read functions), this means that it always returns the same output value when called repeatedly with the same input parameters.

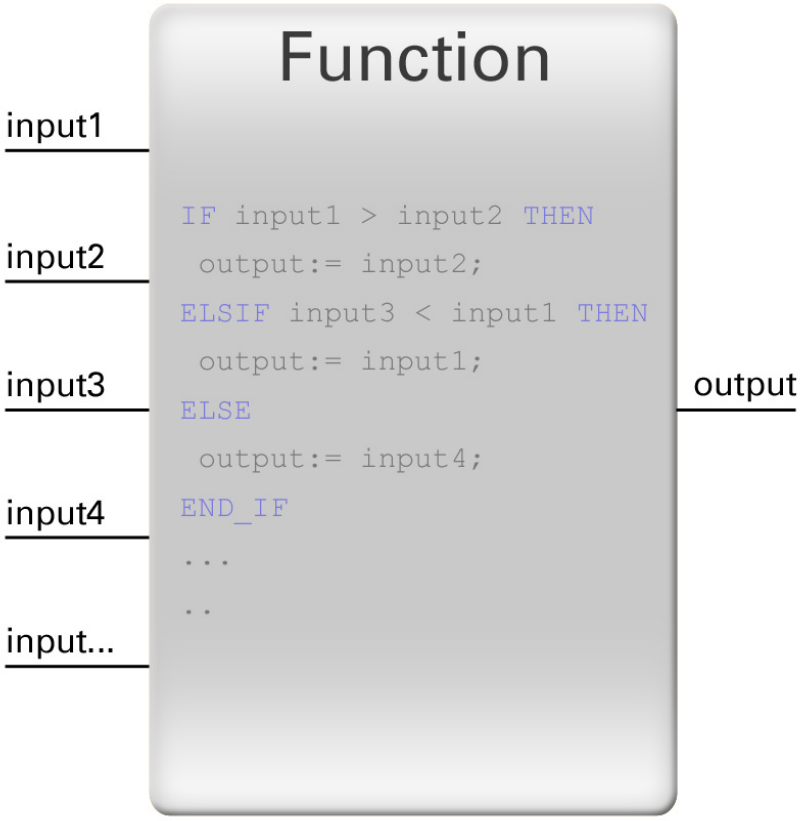


Fig. 3: Function

3.2 Definition of "function block"

A function block is a program organizational unit which can return one or more values. Therefore, it can have one or more inputs and outputs.

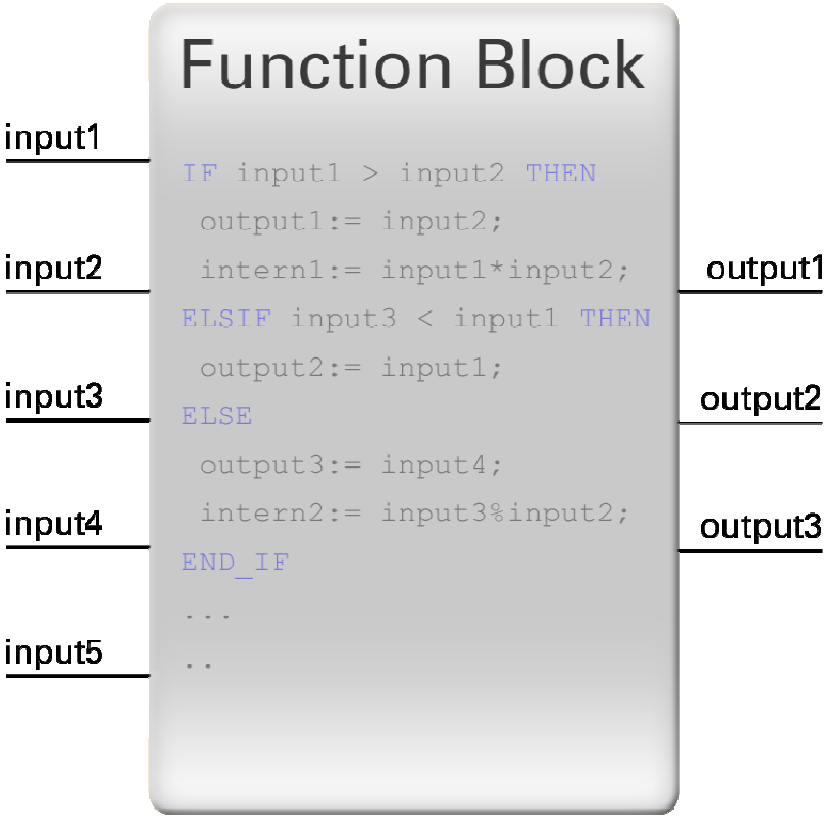


Fig. 4: Function block

An **instance** of a function block must be created before it can be used. This is essentially a data structure, which contains all of the parameters the function block uses (i.e. inputs, outputs, and internal variables).

By using a data structure, function blocks have a **static memory**. When called repeatedly with the same input parameters, the output values can also change.

In some cases, function blocks, which require a great deal of system resources or access hardware, might have to be called repeatedly using multiple cycles. This makes it possible to wait for a response from the hardware and can reduce the load the function block puts on the system.

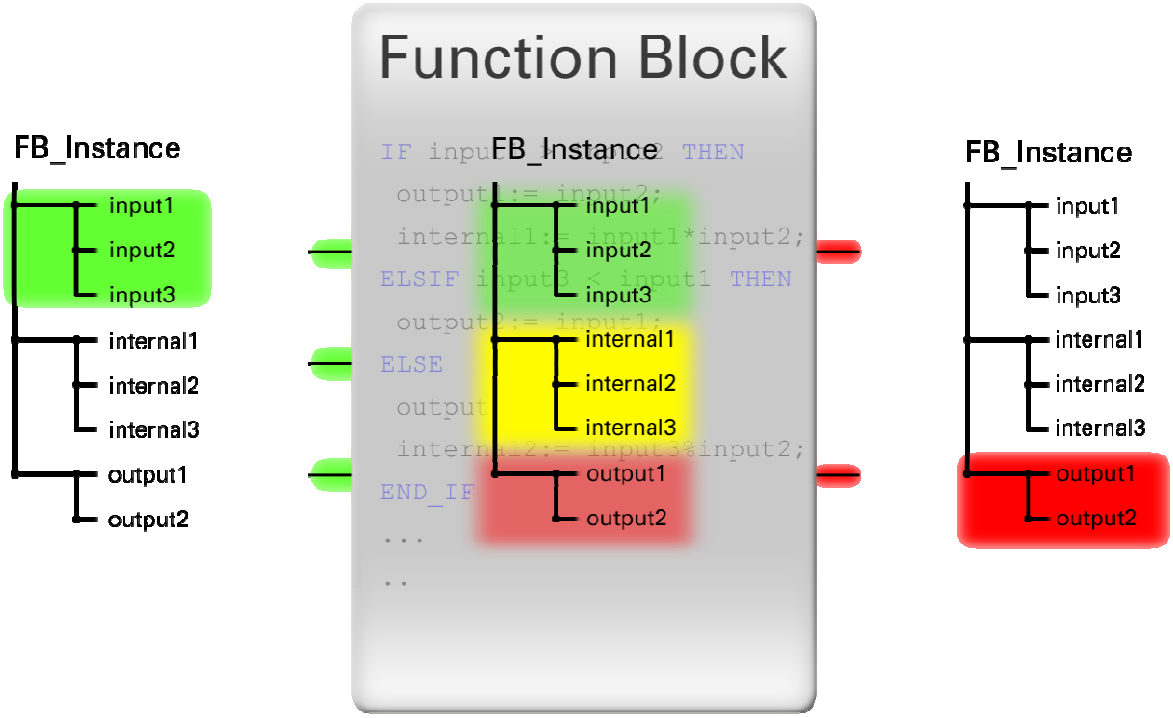


Fig. 5: Function block with instance structure

4. NETWORKS

4.1 General

A network (NW) is the interface where blocks, connections, commands and properties come together.

A network in the function block diagram contains a maximum of 65535 rows and 256 columns.

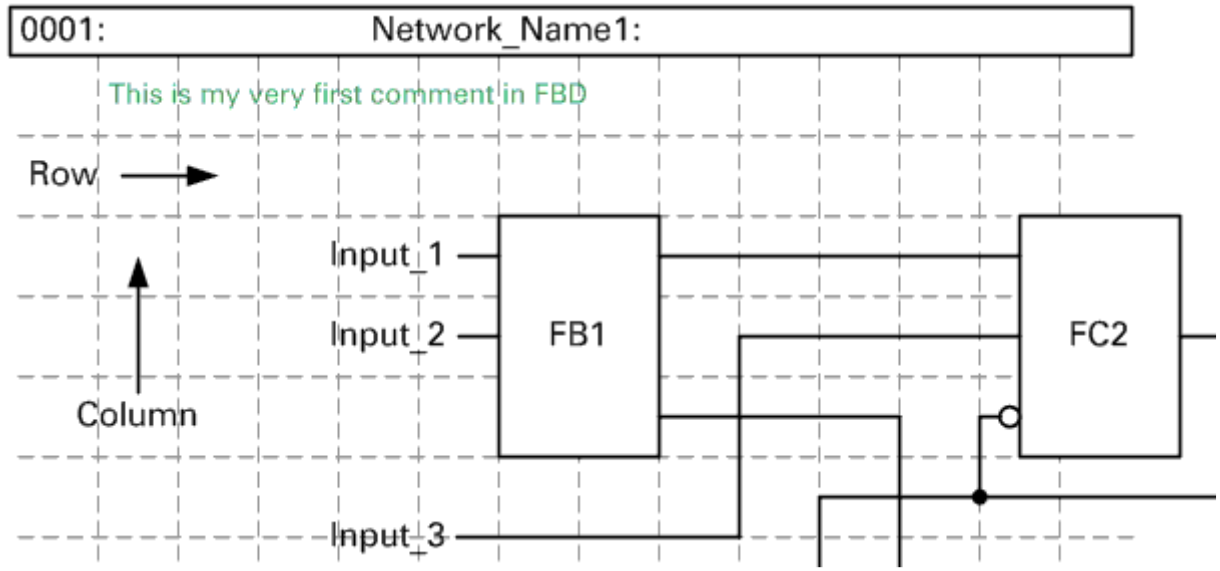


Fig. 6: Network structure

It is possible to insert up to 65535 networks in an application task.

In an extensive application, it is useful to describe each network. **Comments** can be used to do this. The number of and length of comments in a network is unlimited. They can be added anywhere in the network at any time without affecting the software.

A consecutive number is automatically assigned when a network is created. This helps to navigate through the networks, but is not used for network identification.

A **Name** (must be **unique** in the task) can be assigned for uniquely identifying a network. The name can be used for identification and for providing a brief description.

4.2 Order of execution

When filling a network with blocks, not only the connections are important, also the **order** of execution determines a network's functionality. Neglecting this can cause an error which is very difficult to detect.

The order of execution occurs column by column from top to bottom, from left to right.

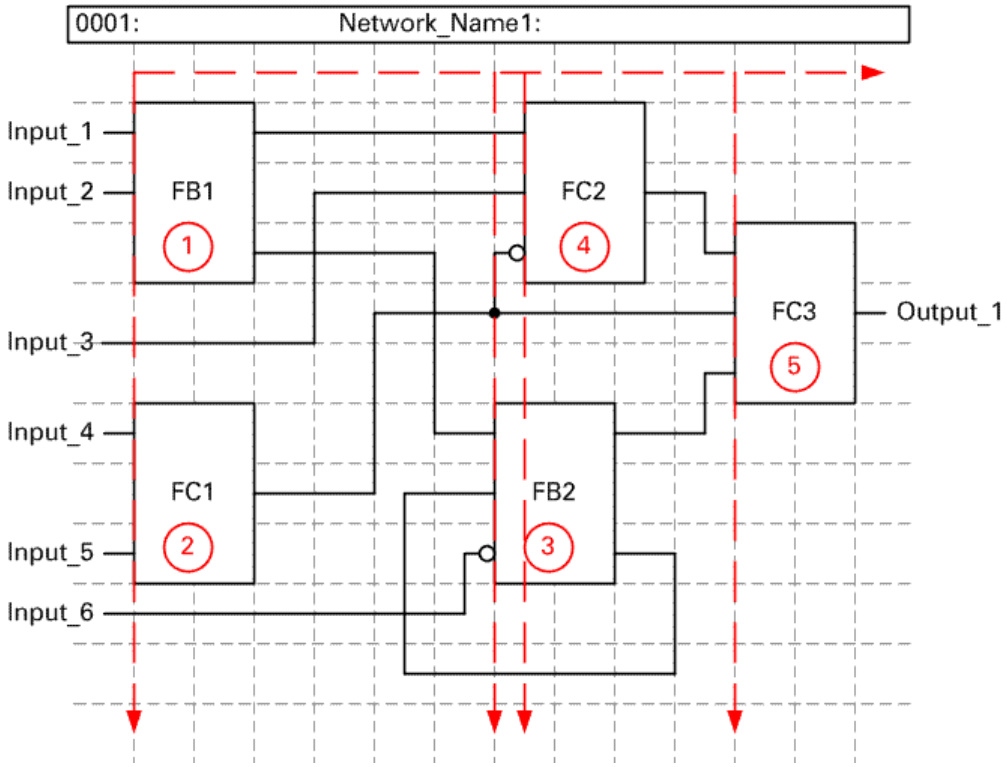


Fig. 7: Order of execution

The example shown in the image illustrates which block is processed when.

4.3 Connections

Blocks must be connected in order to create an application.

Connections can only be made **from output to input**. This makes it impossible to connect two inputs or two outputs.

Connections can only be created between the **same data types**, with the exception that an automatic conversion is made when the target data type is larger than the original.

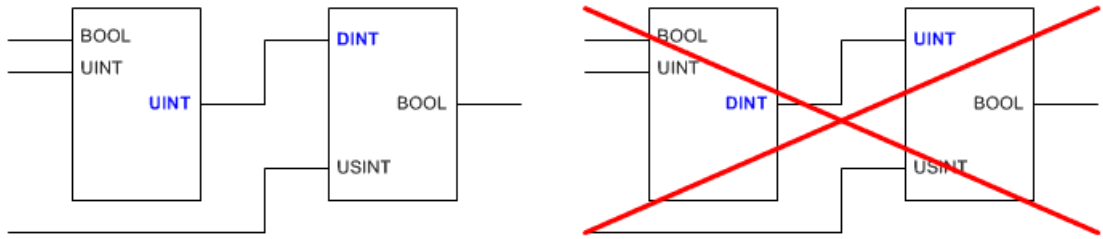


Fig. 8: Connections – data type conversion

Function blocks can only be linked together **in the direction of the execution sequence** because otherwise the data flow behavior is undefined.

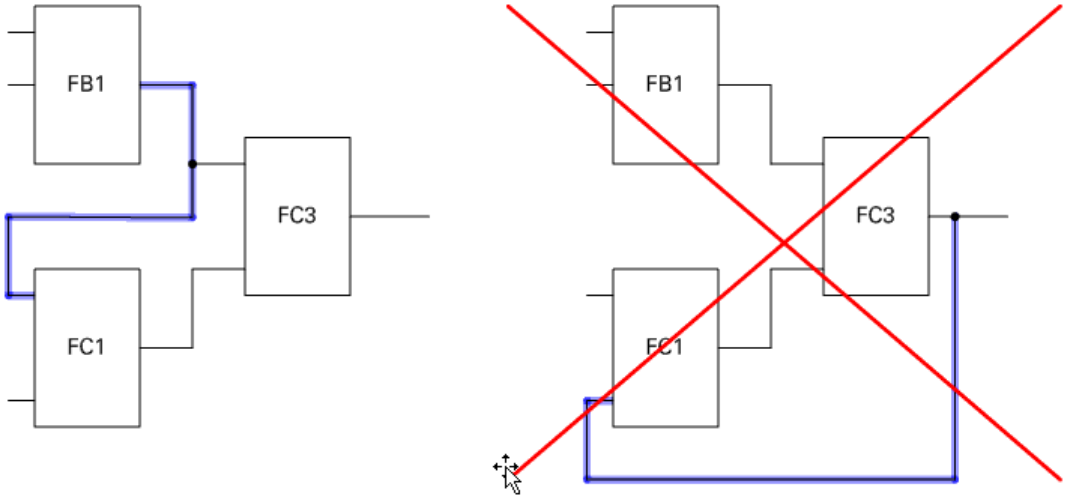


Fig. 9: Connections - direction

4.4 Intersections

In most cases, networks contain blocks with intersecting connections. There are two types of intersections:

- Connectionless intersections
- Connected intersections

4.4.1 Connectionless intersections

Connectionless intersections have the characteristic that a connection is above another connection. This means that the data flow is not changed by a connectionless intersection.

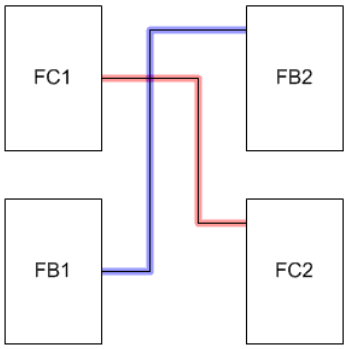


Fig. 10: Connectionless intersection

4.4.2 Connected intersections

Connected intersections change the data flow by forwarding the original data to the other branches of the intersection. A connected intersection can be identified at the intersection connection point.

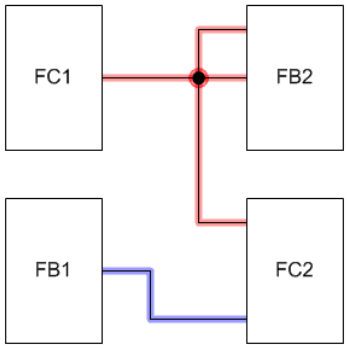


Fig. 11: Connected intersection

4.5 Diagnostics – Signal flow display

FBD contains a very useful tool, the signal flow display, for checking the sequence of the written software and preventing or diagnosing errors.

The signal flow display marks the active connections and blocks in the network. Active means that on blocks the output, and on connections, the transferred value is greater than or equal to 1.

The purpose of the signal flow display is to highlight the elements that are currently affecting the software sequence.

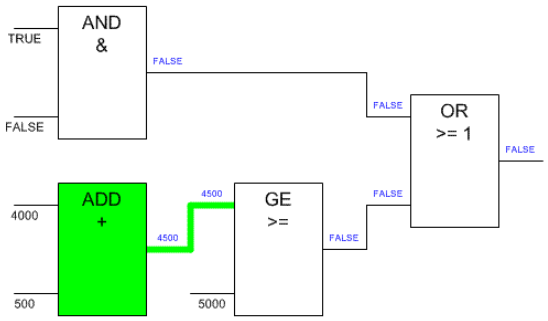


Fig. 12: Signal flow – Example 1

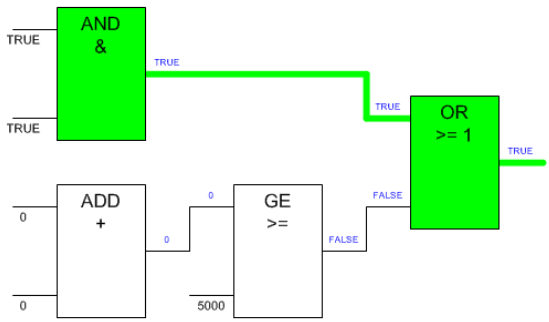


Fig. 13: Signal flow – Example 2

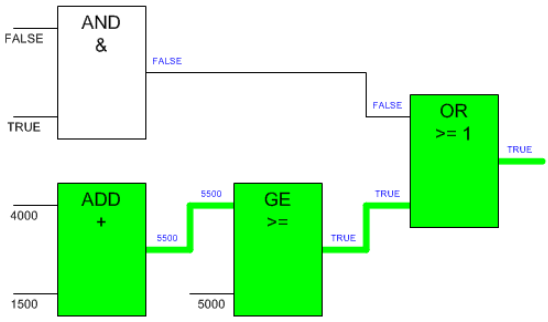


Fig. 14: Signal flow – Example 3

5. LOGIC

5.1 Negation

In some cases, it is necessary to use **the inverse value** of a signal on a logic function block. The Negation feature can be used to do this.

The **Negation is shown as a circle** and can only be used on inputs of the AND, OR and XOR function.

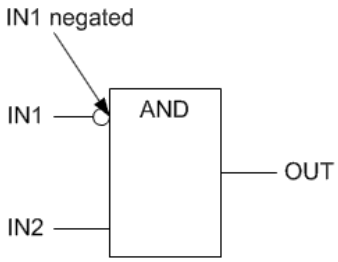


Fig. 15: Negation

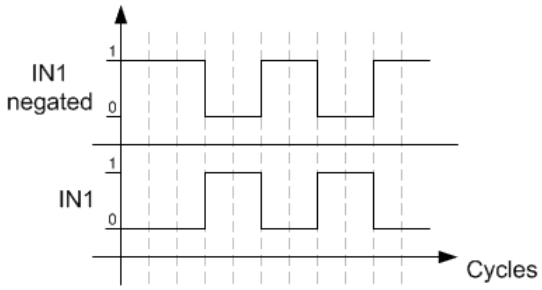


Fig. 16: Negation - Status diagram

5.2 Expandable functions

In certain cases, it is necessary for the function block to use more than the standard number of inputs to to generate an output value. It is possible to expand some blocks to include more inputs in order to avoid having to enter multiple functions in a network to get this value.

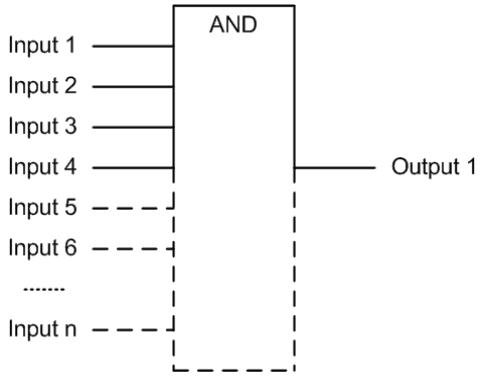


Fig. 17: Expandable Functions

Note:

IEC61131-3 defines which function blocks have this ability and includes the following functions:
ADD, MUL, AND, OR, XOR, MIN, MAX, MUX, CONCAT

Task: Conveyor belt, Part I

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

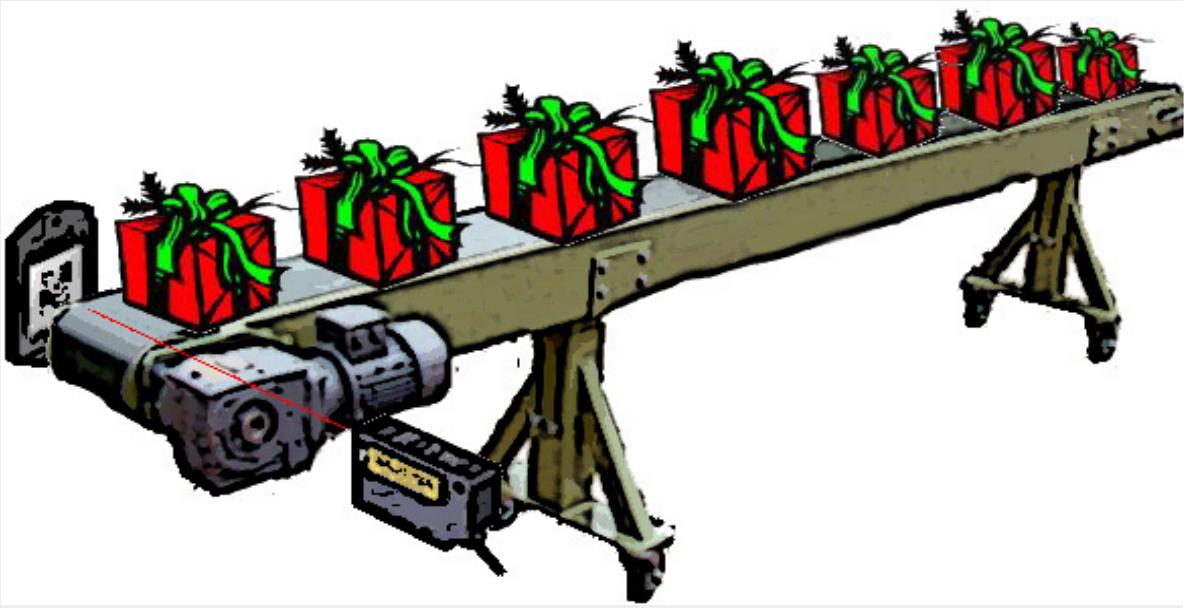


Fig. 18: Conveyor belt

Create a program for manual operation which can be used to switch the "cmdManual" command variable on and off using only the "gDiSwitchManual" input.

5.3 Set/Reset

5.3.1 Set

The SET block is a standard function in FBD.

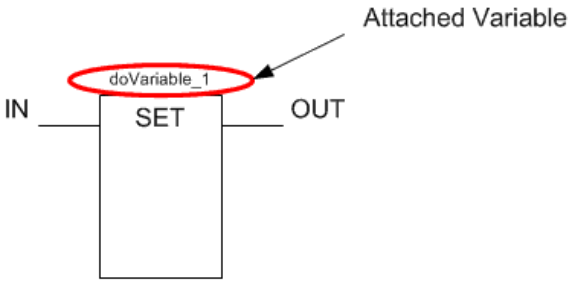


Fig. 19: SET - function

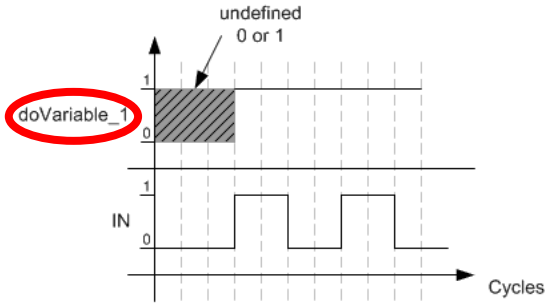


Fig. 20: SET - status diagram

This block writes the boolean value TRUE to the connected variable (doVariable_1), when a positive edge occurs on the input.

This function also has an output (OUT) with the current value of the connected variable (doVariable_1).

The connected variable must be declared as a BOOL data type.

5.3.2 Reset 

The RESET block is the counterpart to the SET block.

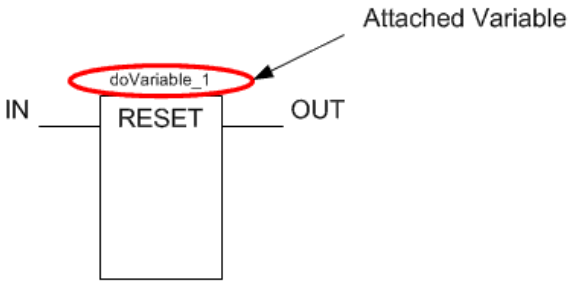


Fig. 21: RESET - function

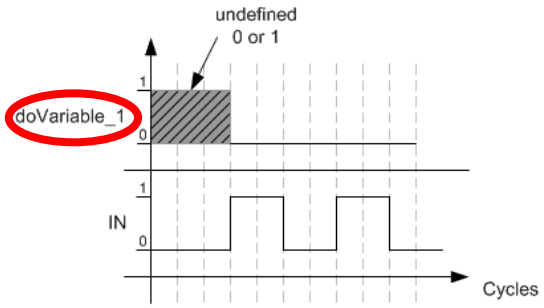


Fig. 22: RESET - Status diagram

This block resets the connected variable (doVariable_1) to the Boolean value FALSE, when a positive edge occurs on the input.

This function also has an output (OUT) with the current value of the connected variable (doVariable_1).

Also here you have to make sure that the connected variable is a BOOL data type.

Task: Conveyor belt, Part II

At this point, it is possible to operate the command in manual mode. Now we will add a few functions for automatic operation and for controlling the "gDoMotor" motor.

Start the conveyor belt:

- If no material is detected by the final sensor of the "**gDiEnd**" conveyor belt.
- If material is detected by the final sensor 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 final sensor 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

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

6. CONTROLLING THE PROGRAM FLOW

The program flow within a task always goes from the **top to the bottom network** (lowest to highest network number), whereby each network is normally run through. It is also possible to control the program in the event that not all program parts have to be run through. This can be done using the jump or jump return block.

6.1 Jump 

The jump is used **to jump from one network to another**.

It is only triggered when the logical value TRUE is on the input. A variable or the output of a block can also be connected to the input, in which case the jump is also called a "conditional jump".

To define which network to jump to, add the name of the network to the function.

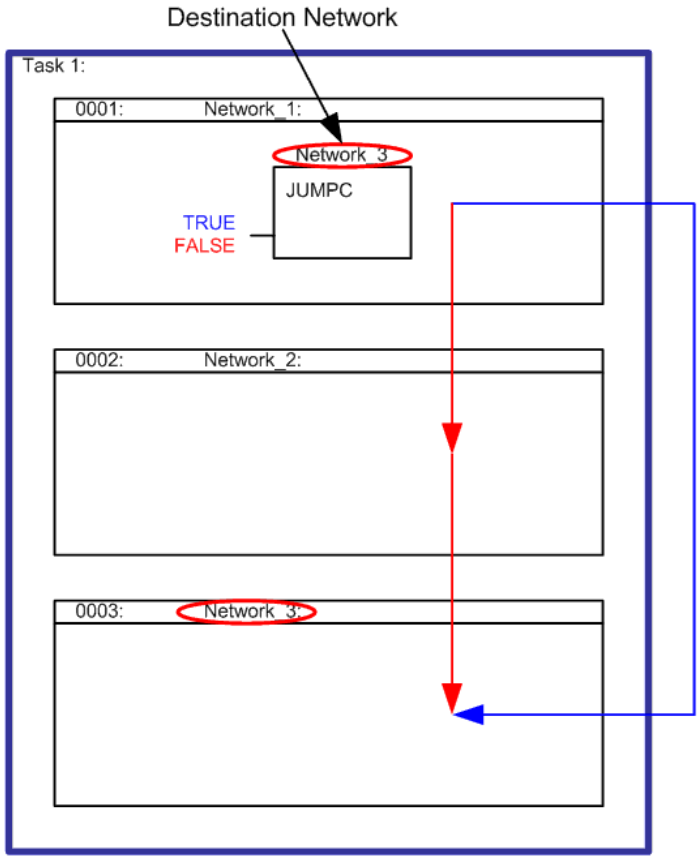


Fig. 23: Execution with and without jump

6.2 Jump Return

The jump return is used to influence when to jump from one task to the next. Processing of the current task is completed and the next program is started.

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

A variable or the output of a block can also be connected to the input, in which case the return is also called a "conditional jump return".

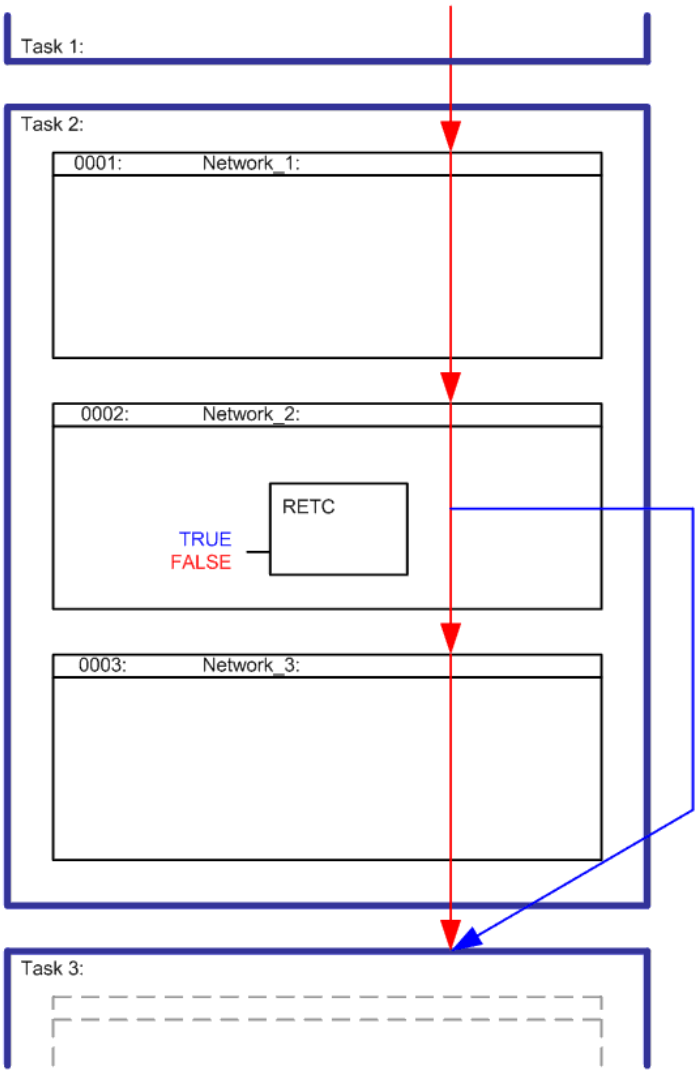


Fig. 24: Order of execution with and without return

Task: Conveyor belt, Part III

The program can now handle two different operating modes.

The conveyor belt can only be operated in one mode. Therefore parts of the software are not required. This depends on the mode that is selected.

Now expand your program so that parts of the software which are not required will be skipped.

7. EXERCISES

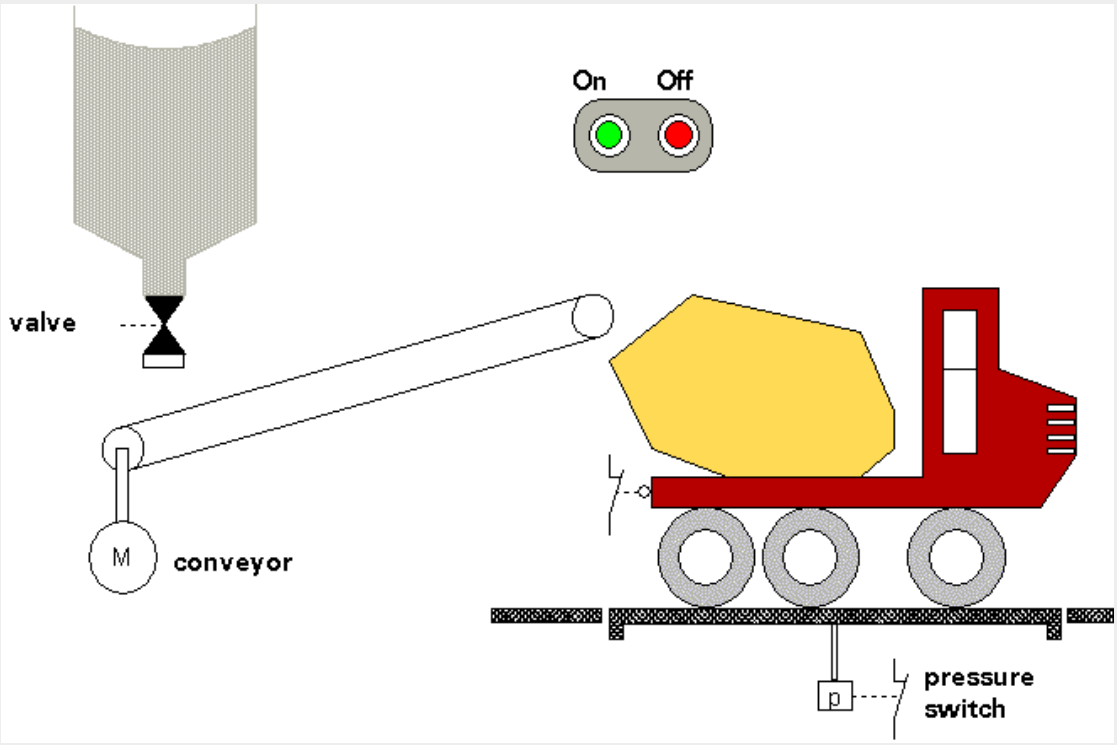
Task: Concrete filling system:

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

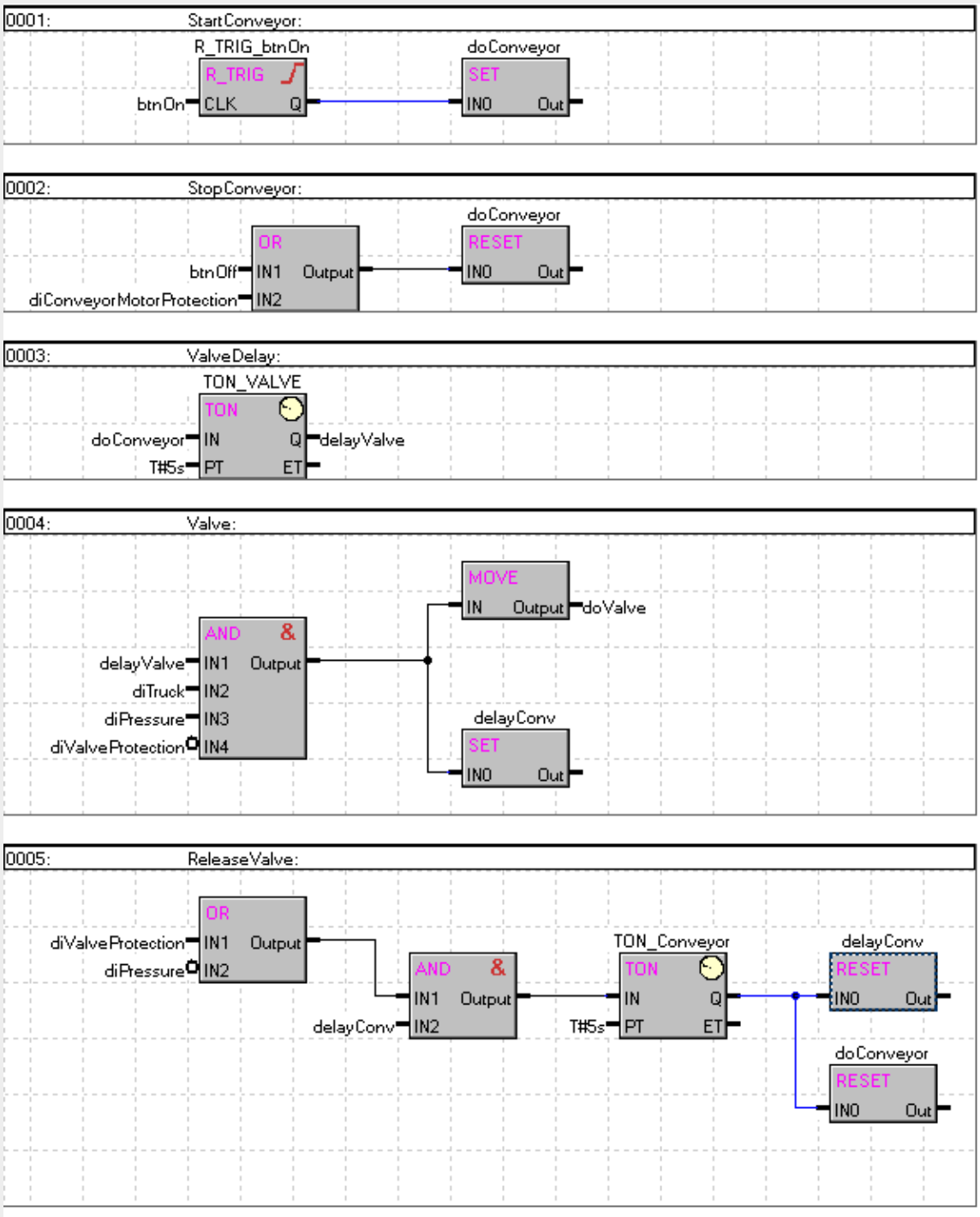
The filling operation is activated using the On-button (btnOn). However, the hydraulic system controlled by a solenoid valve (doValve) cannot be opened until the conveyor has been running for 5 sec. and a truck is located beneath the belt (diTruck).

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

The entire system is immediately shut down if the Off-button (btnOff) is pressed. If there is a disturbance in the conveyor (diConveyorMotorProtection), then the solenoid valve and the conveyor belt (doConveyor) should be shut off immediately. If there is a disturbance in the solenoid valve (diValveProtection), then the solenoid valve is immediately closed, but the belt should run empty for an additional 5 seconds.



The function block diagram could look like this:



8. SUMMARY

Function Block Diagram is an easy-to-learn programming language with a wide range of application.

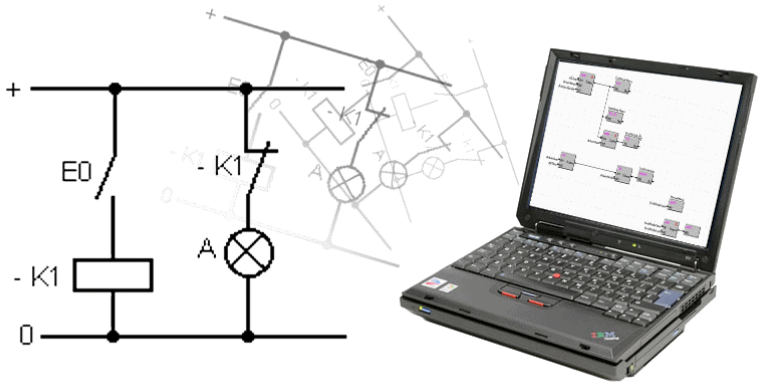


Fig. 25: Summary

You now know what functions and function blocks are and you know how to use and place these blocks in a network. Additional logical elements and functions help you reach your goal sooner and program flow control elements allow you to design your software in a more dynamic manner that utilizes resources more effectively. You are now able to create complex and high-performance applications using the function block diagram.

9. APPENDIX

9.1 Functions in accordance to IEC61131-3

All functions defined in the IEC61131-3 standard can be found in Automation Studio in the "OPERATOR" and "STANDARD" libraries

9.1.1 Arithmetic functions

Name	Description
ADD	Addition
SUB	Subtraction
MUL	Multiplication
DIV	Division
SQRT	Square root
EXPT	Exponential function
SIN	Sine
COS	Cosine
TAN	Tangent
ASIN	Arc sine
ACOS	Arc cosine
ATAN	Arc tangent
LOG	Base 10 logarithm
LN	Natural logarithm
EXP	Natural exponential function
MOD	Generate modulo
TRUNC	Truncate decimal places
ABS	Generate absolute value

9.1.2 Comparison operations

Name	Description
EQ	Equal to (=)
NOT	Not equal to (≠)
GT	Greater than (>)
GE	Greater than or equal to (≥)
LT	Less than (<)
LE	Less than or equal to (≤)

9.1.3 Assign and limitation functions

Name	Description
LIMIT	Limit the output to a Min or Max
MAX	Largest input value as output
MIN	Lowest input value as output
MOVE	Assign
MUX	Select output from multiple inputs
SEL	Select output from two inputs

9.1.4 Address and size functions

Name	Description
ADR	Determine address in the memory
SIZEOF	Size (bytes)

9.1.5 Bit operations

Name	Description
AND	Logical AND
OR	Logical OR
NOT	Logical NOT
XOR	Logical EXCLUSIVE OR
ROL	Bit rotation to the left
ROR	Bit rotation to the right
SHL	Bit shift to the left
SHR	Bit shift to the right

9.1.6 String functions

Name	Description
CONCAT	Group strings
DELETE	Delete character from a string
FIND	Determines the position of a character string within a string
REPLACE	Replaces characters in a string
INSERT	Inserts a string into a string
LEFT	Copies the left character of a string
RIGHT	Copies the right character of a string
MID	Copies the middle character of a string
LEN	Determines the length of a string

9.1.7 Counter functions

Name	Description
CTU	Counts up
CTD	Counts down
CTUD	Counts up and down

9.1.8 Time functions

Name	Description
TON	Switch-on delay
TOF	Switch-off delay
TP	Pulse

9.1.9 Setting / resetting

Name	Description
RS	FlipFlop (reset priority)
SR	FlipFlop (set priority)

9.1.10 Edge functions

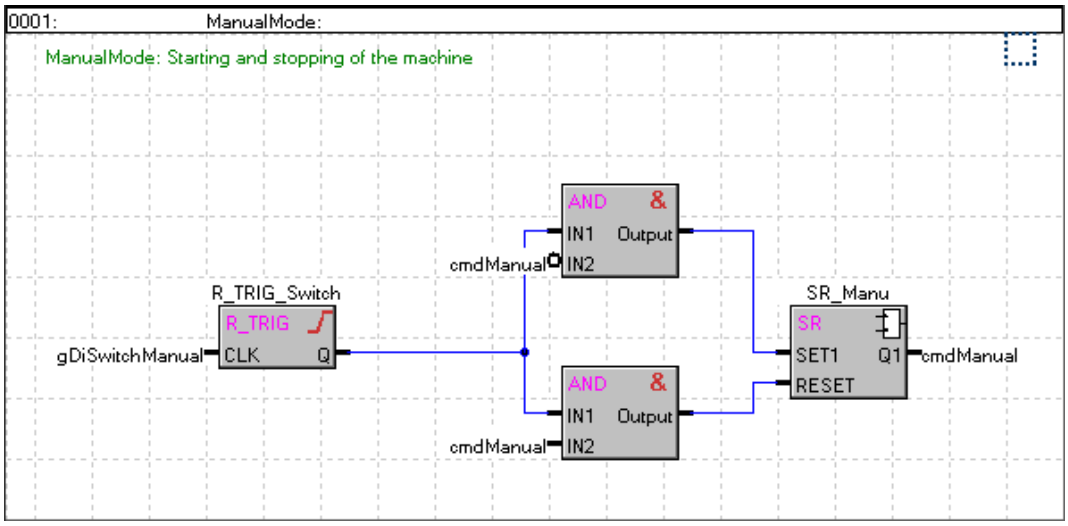
Name	Description
R_TRIG	Detects positive edges
F_TRIG	Detects negative edges
RF_TRIG	Detects positive and negative edges

9.1.11 Semaphore functions

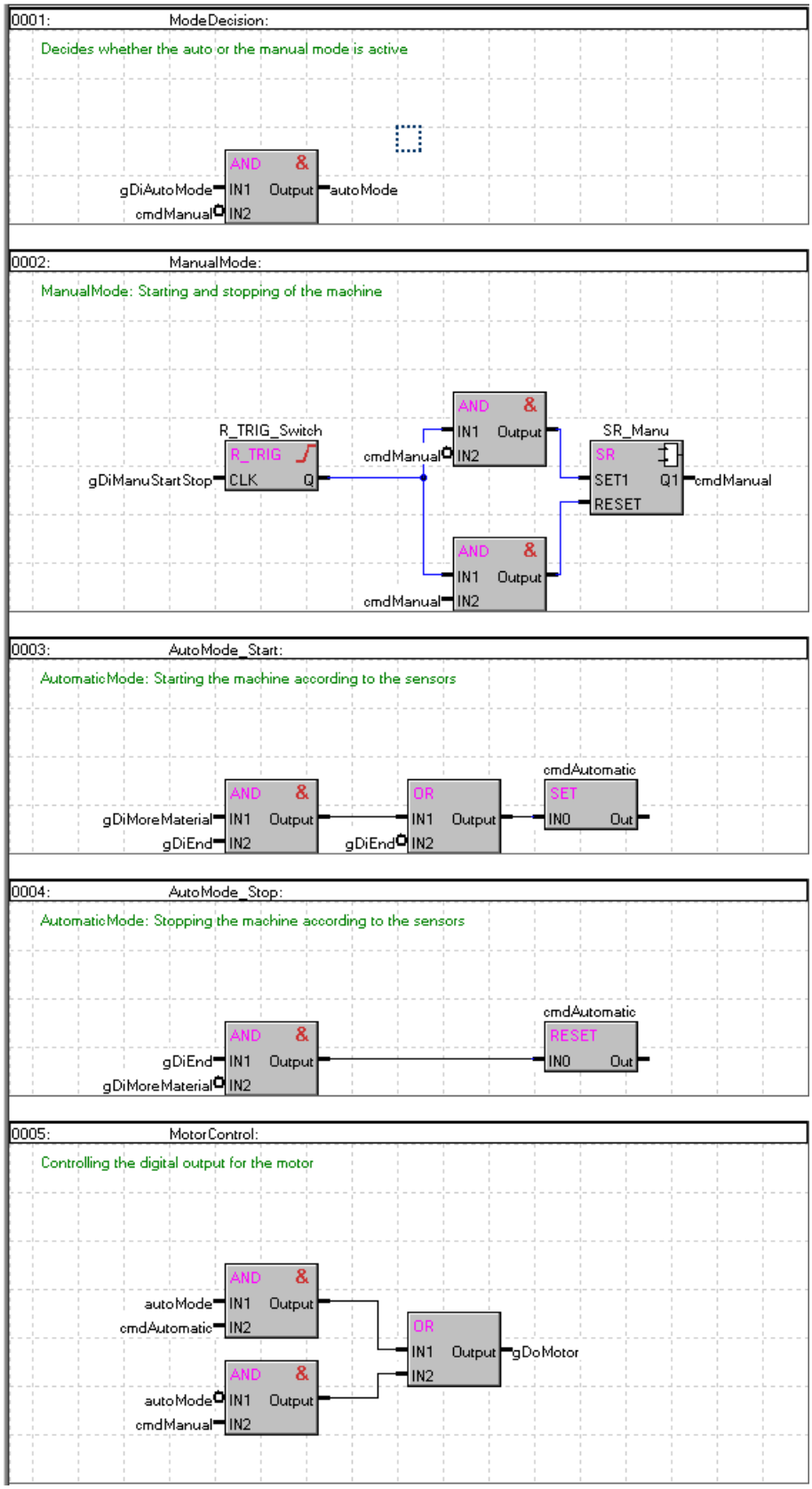
Name	Description
SEMA	Semaphore handling within a task class

9.2 Solutions

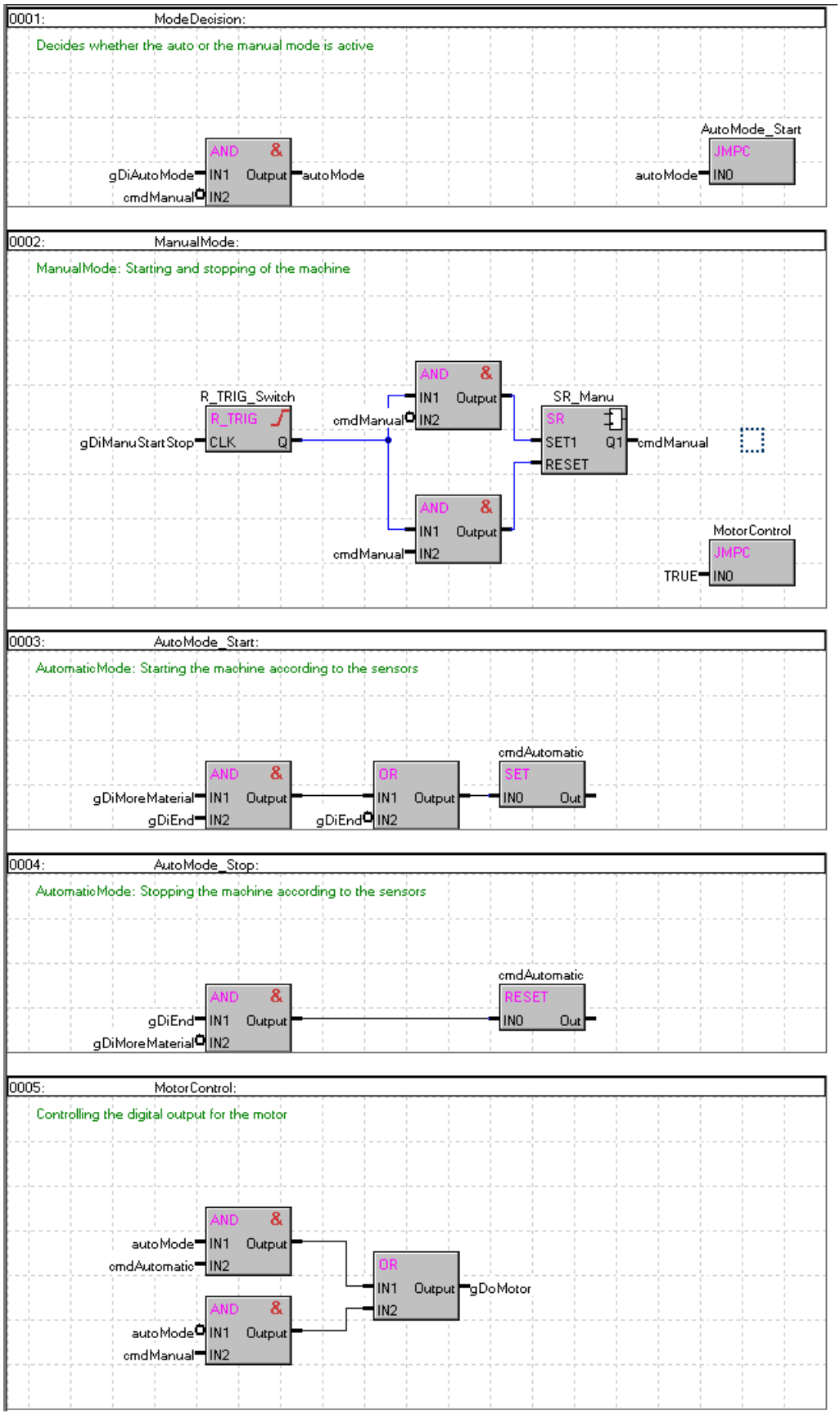
9.2.1 Conveyor belt – Part 1 (Sec. 5.2)



9.2.2 Conveyor belt – Part 2 (Sec. 5.3.2)



9.2.3 Conveyor belt – Part 3 (Sec. 6.2)



Overview of training modules

TM210 – The Basics of Automation Studio
TM211 – Automation Studio Online Communication
TM213 – Automation Runtime
TM220 – The Service Technician on the Job
TM223 – Automation Studio Diagnostics
TM230 – Structured Software Generation
TM240 – Ladder Diagram (LAD)
TM241 – Function Block Diagram (FBD)
TM246 – Structured Text (ST)
TM250 – Memory Management and Data Storage
TM261 – Closed Loop Control with LOOPCONR

TM400 – The Basics of Motion Control
TM410 – The Basics of ASiM
TM440 – ASiM Basic Functions
TM441 – ASiM Multi-Axis Functions
TM445 – ACOPOS ACP10 Software
TM446 – ACOPOS Smart Process Technology
TM450 – ACOPOS Control Concept and Adjustment
TM460 – Starting up Motors
TM480 – Hydraulic Drive Control

TM500 – The Basics of Integrated Safety Technology
TM510 – ASiST SafeDESIGNER
TM540 – ASiST SafeMC

TM600 – The Basics of Visualization
TM610 – The Basics of ASiV
TM630 – Visualization Programming Guide
TM640 – ASiV Alarm System, Trend and Diagnostic
TM670 – ASiV Advanced

TM700 – Automation Net PVI
TM710 – PVI Communication
TM711 – PVI DLL Programming
TM712 – PVIServices
TM730 – PVI OPC

TM800 – APROL System Concept
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
TM840 – APROL Parameter Management and Recipes
TM850 – APROL Controller Configuration and INA
TM860 – APROL Library Engineering
TM865 – APROL Library Guide Book
TM870 – APROL Python Programming
TM890 – The Basics of LINUX

CORPORATE HEADQUARTERS

Bernecker + Rainer Industrie-Elektronik Ges.m.b.H.
B&R Strasse 1
5142 Eggelsberg
Austria
Tel.: +43 (0) 77 48 / 65 86 - 0
Fax: +43 (0) 77 48 / 65 86 - 26
info@br-automation.com
www.br-automation.com

TM241TRE.30-ENG 0907

155 offices in 60 countries - **www.br-automation.com/contact**

Argentina • Australia • Austria • Belarus • Belgium • Botswana • Brazil • Bulgaria • Canada • Chile • China • Colombia • Croatia • Cyprus
Czech Republic • Denmark • Egypt • Emirates • Finland • France • Germany • Greece • Hungary • India • Indonesia • Ireland
Israel • Italy • Japan • Korea • Luxemburg • Kyrgyzstan • Malaysia • Mexico • Mozambique • Namibia • The Netherlands
New Zealand • Norway • Pakistan • Peru • Poland • Portugal • Romania • Russia • Serbia • Singapore • Slovakia • Slovenia • South Africa
Spain • Sweden • Switzerland • Taiwan • Thailand • Turkey • Ukraine • United Kingdom • USA • Venezuela • Vietnam • Zimbabwe