TTE-50200 Tuotantoautomaatio

BALLGAME SPECIFICATION AND WORKING WITH TWINCAT (TwinCAT Guide)

1 PRACTICAL APPLICATION

1.1 INTRODUCTION

The practical application deals with configuring the hardware; designing and implementing the control program of an assorting system. The system is shown in the pictures below.

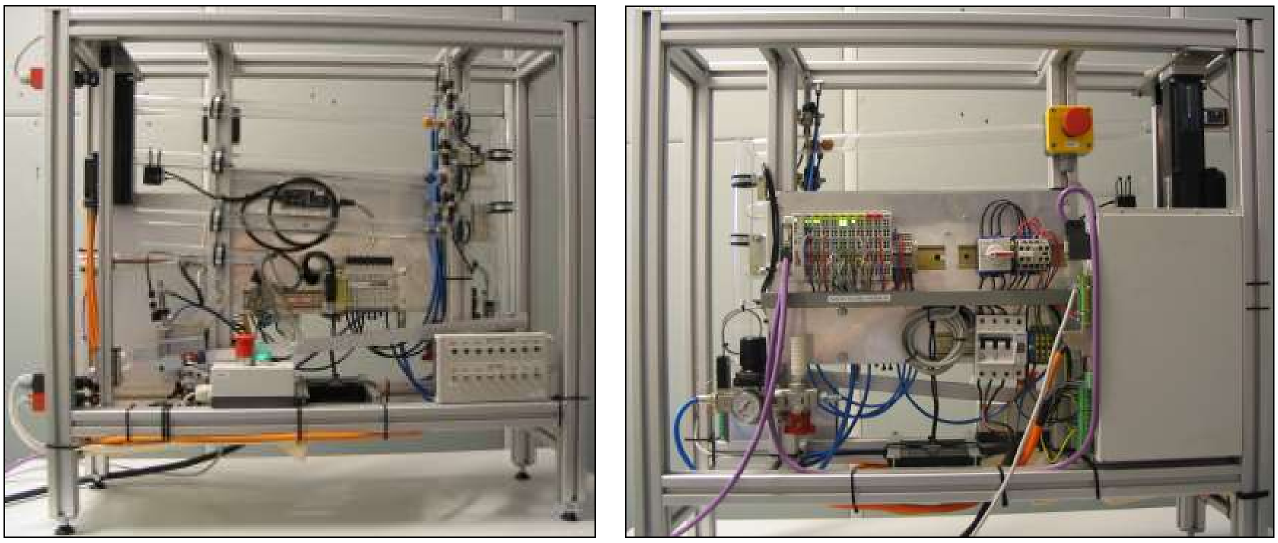


Figure 1 Overview of the system

The aim of the device is to sort different kind of balls from magazine to correct pipes:

Up most pipe = Yellow, wooden balls
Middle pipe = Blue, wooden balls
Lowest pipe = Metal balls

After sorting the sorted balls are brought back into the magazine by opening the pipe gates. Sorting starts again from the beginning.

The information from three (3) different sensors can be used for sorting the balls:

1. Optical gate = optical sensor
2. Inductive sensor
3. Colour sensor

1.2 WORKING PRINCIPLE & REQUIREMENTS

The position of the sensors and the actuators in the system is presented in the following schema:

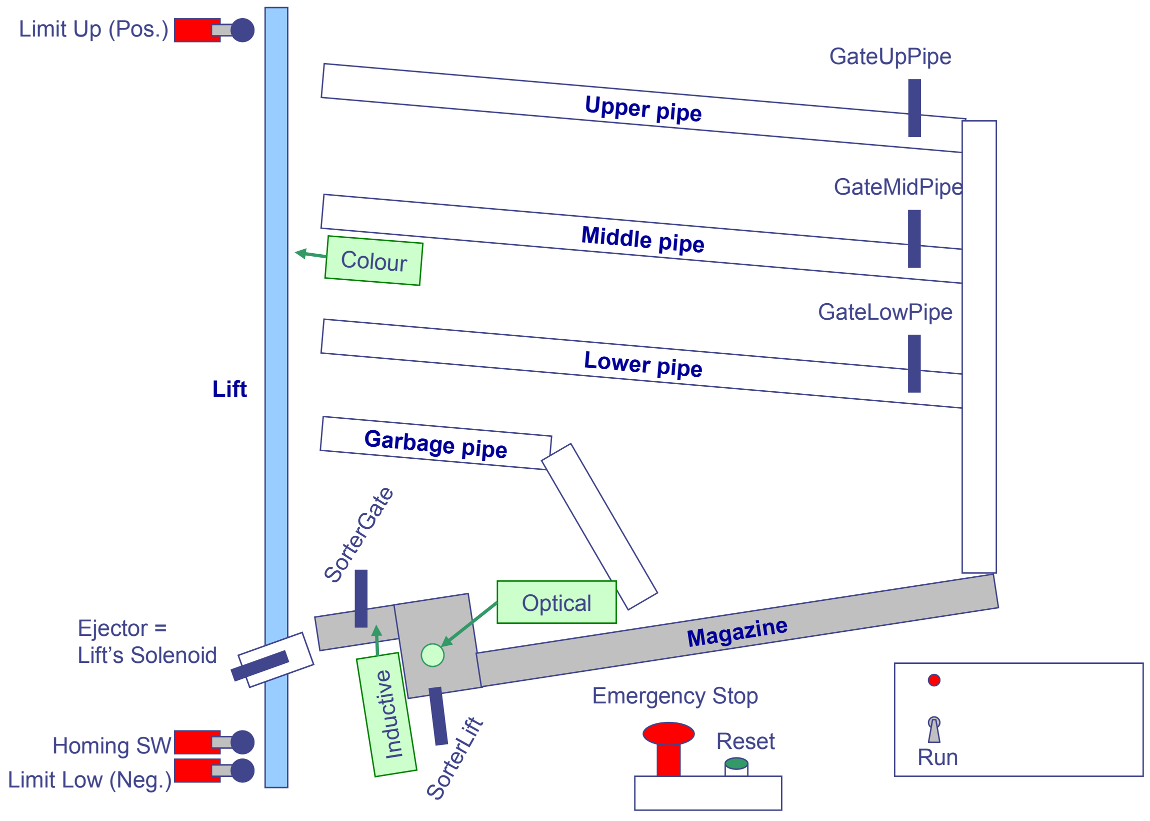


Figure 2 Functional parts of BallGame

The starting point for the system is the two sensors located in the sorter (in the end of the ball magazine). The sensor located at the end of magazine just in the slot of the sorter lift is optical sensor or optical gate. It will recognise if there is some object between the open ends of the fibers, which take the light to the actual sensing device (SW10). The sensor located closer to the sorter gate is an Inductive Sensor (SW11). It will recognise if the object in front of it is a metallic one.

The sorter lift is used to take one ball at time to the 2nd level of the sorter. At front of the sorter gate there is room for only one ball at time. From the sorter the ball is rolled to the lift by opening the sorter gate. One ball at the time should let through from the sorter gate.

Lift shall take the ball to:

a) garbage pipe
b) lower pipe
c) middle pipe
d) upper pipe
e) colour sensor

Lift could take the ball to visit one or more locations before it is ejected to proper pipe according the desired sorting algorithm.

For recognizing the colour of the ball it must be taken in front of the colour sensor (SW13) with the lift. If the colour is recognised one of the predefined ones (i.e. taught to the sensor) the sensor will tell it with one of the four signals, each representing one degree of colouring.

After making the decision the ball is taken to proper pipe and it is ejected into the pipe with the solenoid located in the lift. Next ball is taken into the sorting cycle.

When all balls have been sorted the storage pipes are emptied one after one in desired order, and balls are taken back to the magazine. The storage pipes should not be opened at the same time, because jam may occur.

Other requirements are that:

- Preconditions must be taken into account. (i.e. powers are connected & available, actuators are in correct positions, emergency stop loop is neutralized, etc.)
- Initialization cycle will bring the system to known state before the normal sorting cycle can start. (E.g. homing the lift, driving it to position, …)
- Run switch will be used to start and stop the system. However the stopping must be done in “nice” phase. This means that stopping is not done right away, but in the end of current working cycle.
- Pressing the emergency stop will stop the system immediately, without intervention of the control program. However this will not stop the control program itself, but it must handle itself properly after recognition of emergency stop activation.

Optional & Additional requirements:

- The amount of sorted balls are counted and displayed. Also the number of balls in tubes is illustrated.
- Additional manual actions (e.g. emptying pipes, resetting counters, …)
- Error diagnostics and alarm indications.
- Implementation of HMI

The available I/Os for the control program are presented in the chapter 1.4.

1.3 System Components

1.3.1 Controller architecture and connections

In this exercise control of the Ball Game –device is done directly with PC. Inside PC is running a software based PLC, usually called softPLC. This software will provide real-time controls via fieldbus interface.

In this exercise is used Profibus DP as fieldbus. Embedded PC has Profibus master module that will offer the controller an access to real world’s devices. Sensors and different kinds of outputs are connected to I/O terminal modules, which are further connected to Profibus slave terminal i.e. buscoupler (BK3000). These devices are coming from Beckhoff and WAGO. It is the first slave in our fieldbus network.

Second slave in our network is the Indramat position controller / Servo drive.

Each slave does have a specific configuration file in case of Profibus, which are called GSD-files. This will tell essential configuration information to the fieldbus master and to fieldbus configurator. In this case the GSD –files for the I/O terminals are integrated beforehand to the configurator

(TwinCAT System Manager), but e.g. Indramat drive's GSD –file must be available for making the working configuration.

These files should be introduced to the configurator, when the hardware configuration is done. Because the buscoupler is made by the same vendor than the softPLC itself, the configuration can be done by just selecting the proper controller from list. Same procedure can be continued when introducing I/O terminals.

For the case of Indramat drive we need to add "Generic Profibus Box" and assign proper GSD –file into it.

1.3.2 Safety circuit

The following safety circuitry is used to protect the humans and device.

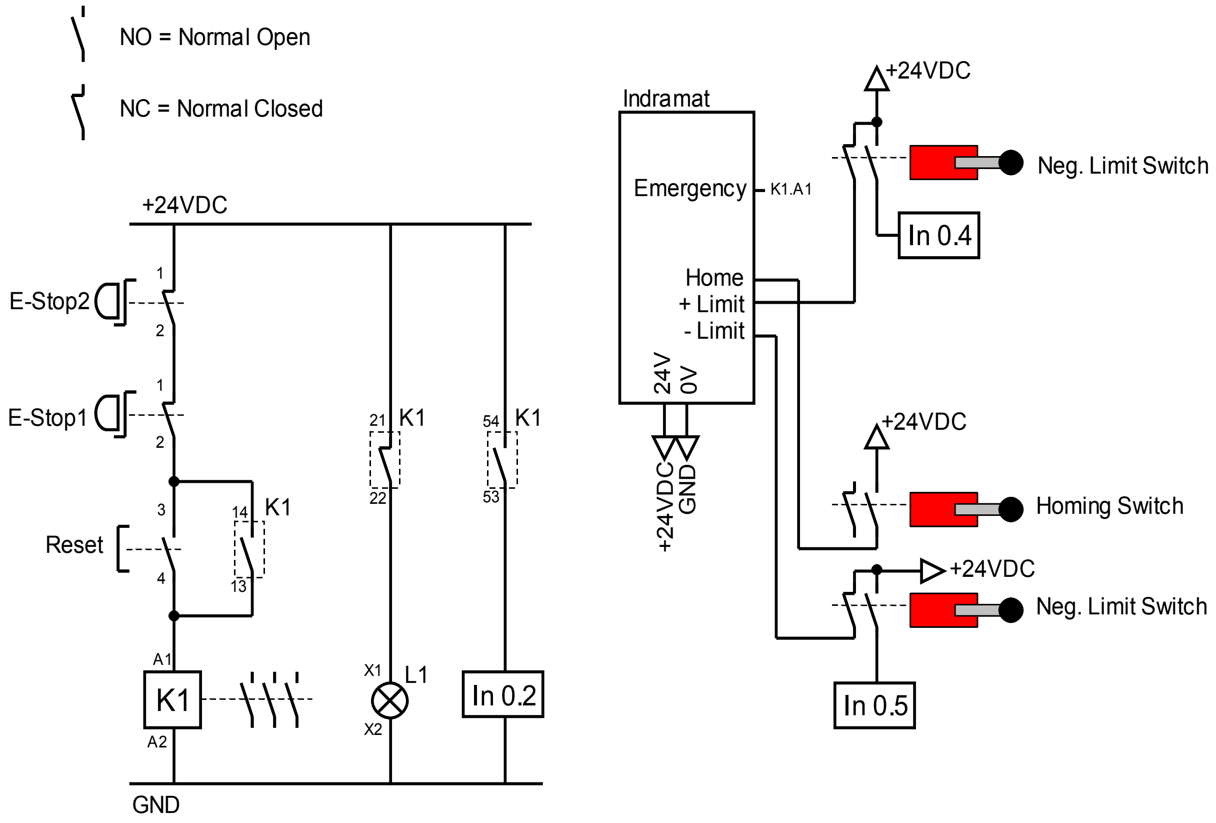


Figure 3 Safety circuit of BallGame

1.3.3 INDRAMAT POSITION CONTROLLER

NOTE!: The description of the detailed content of drive control words and functions MUST be read from the drive's manual: ECODRIVE DKC03.1 Drive Controller.

The Position Controller has its own Profibus connector. This is the most complicated component of system. It needs special software for the configuration and programming of the movements of the drive. Information of those movements are stored in Process Blocks into the Drives memory. Assistant has done configuration and programming beforehand, so those are out of the basic scope of this exercise.

Movements' parameters are like: target position, speed, acceleration, jerk, etc. In the PLC program you call those Process Blocks by the numbers given in the configuration of the position controller.

The actuator for the movement of the lift is a servo motor that is connected with tooth belt to a ball screw. At the end of the servo is integrated resolver that provides the feedback information to the drive controller.

1.1.1.1 Working principle

The lift is using absolute positioning. This means that first thing to do is homing procedure. With the homing sequence the drive is taken to 0–position. All positions are calculated comparing this position. The 0-position is found with external sensor – homing switch (SW 4). After homing procedure has been run the 0 mark is set, drive is homed and it is ready for making movements.

All movements are performed by calling predefined Process blocks. Each Process block has been defined its own target with set of related parameters. These parameters are like target position, used velocity and acceleration, etc.

Drive is controlled from the PLC program with two words: *ODriveControl* and *IDriveStatus*. The former is for commanding the drive from PLC program and the later is for receiving feedback from the drive. These words are used bit by bit in the control program. See Table 1.

Table 1 Drive controller I/Os

I/Os	Components	Value	Meaning
IDriveStatus	Drive Controller Status Word (Output from Drive)	`XXXX XXXX XXXX X1XX (xxx4h)`	Drive is Homed
		`XXXX XXXX XXXX 1XXX (xxx8h)`	Drive is in Motion
		`XXXX XXXX XXX1 XXXX (xx1xh)`	Drive is in Desired Position
		`XXXX XXXX XX1X XXXX (xx2xh)`	Drive OK, no error
		`XXXX XXXX X1XX XXXX (xx4xh)`	Drive Ready
		`XXXX XXXX 1XXX XXXX (xx8xh)`	Drive Power is ON
ODriveControl	Drive Controller Control Word (Input to Drive)	`XXXX XXXX XXXX XXX1 (xxx1h)`	Drive enabled
		`XXXX XXXX XXXX XX1X (xxx2h)`	Start movement/drive
		`XXXX XXXX XXXX X1XX (xxx4h)`	Start homing
		`0000 1010 XXXX XXXX (0Axxh)`	Target is loading position
		`0000 1011 XXXX XXXX (0Bxxh)`	Target is lower pipe
		`0000 1100 XXXX XXXX (0Cxxh)`	Target is middle pipe
		`0000 1101 XXXX XXXX (0Dxxh)`	Target is upper pipe
		`0000 1110 XXXX XXXX (0Exxh)`	Target is garbage pipe
		`0000 1111 XXXX XXXX (0Fxxh)`	Target is colour sensor

1 Word = 2 Bytes = 4 Hex values = 16 bits

1.1.1.2 Homing sequence

1. Make sure that drive has all powers on and
   emergency stop circuit is reseted and
   there is not any error active in the drive.
2. Set Enable and Start Homing-bit ON in *DriveControl* word
3. Wait 500ms (>>2ms)
4. Set Start –bit ON
5. Homing procedure starts
6. Check the bit “Drive is homed” in *DriveStatus* word for finding out the point when the homing procedure has been done
7. Turn Start –bit OFF.

1.1.1.3 Calling Process blocks = Performing a motion with lift

1. Precondition: Homing sequence has been performed
2. Set Enable bit ON and the desired target to *DriveControl* word. Make sure the Start bit is turned OFF.
   Target is selected by selecting the right process block from the drive. The process block will perform the movement to correct absolute position.
   Desired target is set to the higher byte of the control word (I.e. the bits 13-8). See Table 2.
3. Wait 500ms (>>2ms)
4. Set Start–bit ON and start delay
5. Movement starts
6. After delay has expired…
7. Check the bit “In positioning window” in *DriveStatus* word for finding out the point when the target position has been reached.
8. Turn Start –bit OFF.

Detailed timing of the movement is presented in figure below.

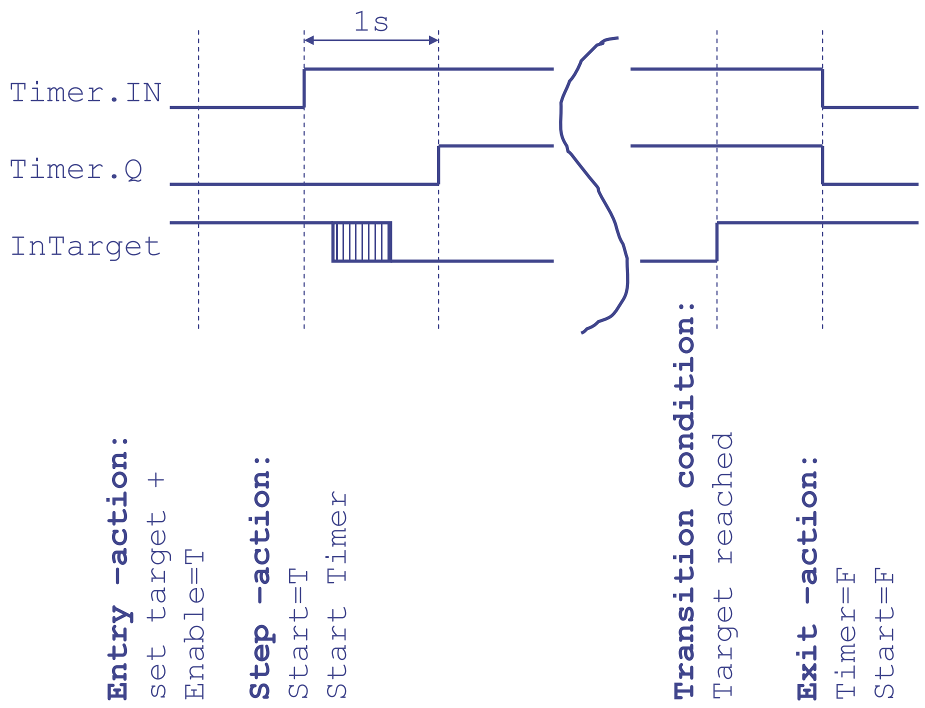


Figure 4 Detailed timing diagram of the movement

Table 2 Available Process blocks = Targets

Process Block #	Same in hexadecimal of Control Word	Description
10	0Axxh	Lift is moved to loading position
11	0Bxxh	Lift is moved to lower pipe
12	0Cxxh	Lift is moved to middle pipe
13	0Dxxh	Lift is moved to upper pipe
14	0Exxh	Lift is moved to garbage pipe
15	0Fxxh	Lift is moved to colour sensor

xx = any bits

Example:
If you want drive the lift to the front of the lower assorting pipe you select Process block number 11 (0Bxxh), and when you want the lift to go back down to the initial position from the current position you call the Process block number 10 (0Axxh).

The numbers of the Process blocks are given in the first byte of the Word. For driving the motor you must send the number of the movement and the enable bit and then after short delay the start bit.

To give the drive a command of selecting the process block #11 (0Bh) and set the Enable –bit TRUE the *ODriveControl* is written the following value:
0000 1011 0000 0001b (0B01h)

To start the movement prepared in the previous chapter we need in addition to turn the Start –bit TRUE. We do this writing the following value to the *ODriveControl* word:
0000 1011 0000 0011b (0B03h).

From the *Drive Controller Status Word* you can see when motor is moving, when it is stopped or other kind of information. So with that *word* you know when the movement is finished and then you can perform the next action.

1.3.4 Motion control block (Optional aid)

In some courses where this piece of equipment is used there is ready made function block supplied for students. That function block helping the implementation of motion control is defined in this chapter.

Introduction

The function block has a set of inputs and outputs, which must be assigned. The block is taking care all funcitions needed for moving the lift i.e. controlling the drive. It hides from user issues like checking the precondition, making of homing sequence, and sequences needed for starting the drive and move the lift to desired destination.

Functional description

Signals connected to the Motion control Function Block used for controlling the indramat drive are presented in the Figure 5.

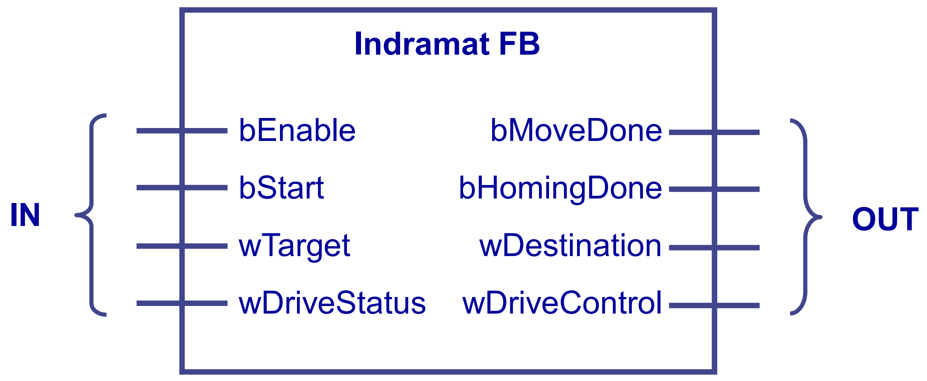


Figure 5 Motion Control FB

Table 3 Signals of the Motion Control FB

Directio n	Name	Type	Description
IN	bEnable	Boolean	Enabling the drive for operation. Must be true all the time during operation. If signal comes false drive will stop immediatelly.
IN	bStart	Boolean	Starting the movement towards the destination. Must be true during the entire movement. If signal comes false drive will stop immediatelly.
IN	wTarget	Word	Destination where the drive is commanded to move. Is actually the number of drive's Process Block that has preprogrammed function. For allowed block numbers see Table 2 on page 9.
IN	wDriveStatus	Word	The status word for reporting the status of the drive. Real input from the drive to the control system via fieldbus. Content and bits of this word are defined in the user manual of the Indramat drive. **Special notice!!!** Make sure the byte & bit order of this word corresponds the order defined in the user manual!!
OUT	bMoveDone	Boolean	Indicates that the drive has reached the destination it has been commanded. Requires the bStart signal to be true.
OUT	bHomingDone	Boolean	Drive has made properly the homing cycle. I.e. it has moved to the homing sensor and found the reference position. Drive is able to start absolute positioning.
OUT	PowerOn	Boolean	Indicates that both high and control voltages are supplied for the controller. Please notice the positions of main switches and fuses in the instrumentation.
OUT	Enabled	Boolean	Indicates that drive is enabled. Preconditions: PowerOn is true and drive initialization cycle is properly done after Enable signal.
OUT	InMotion	Boolean	Indicates when the drive is physically moving.
OUT	wDriveControl	Word	The control word for commanding the drive. Real output from the control system to the drive via fieldbus. Content and bits of this word are defined in the user manual of the Indramat drive. **Special notice!!!** Make sure the byte & bit order of this word corresponds the order defined in the user manual!!

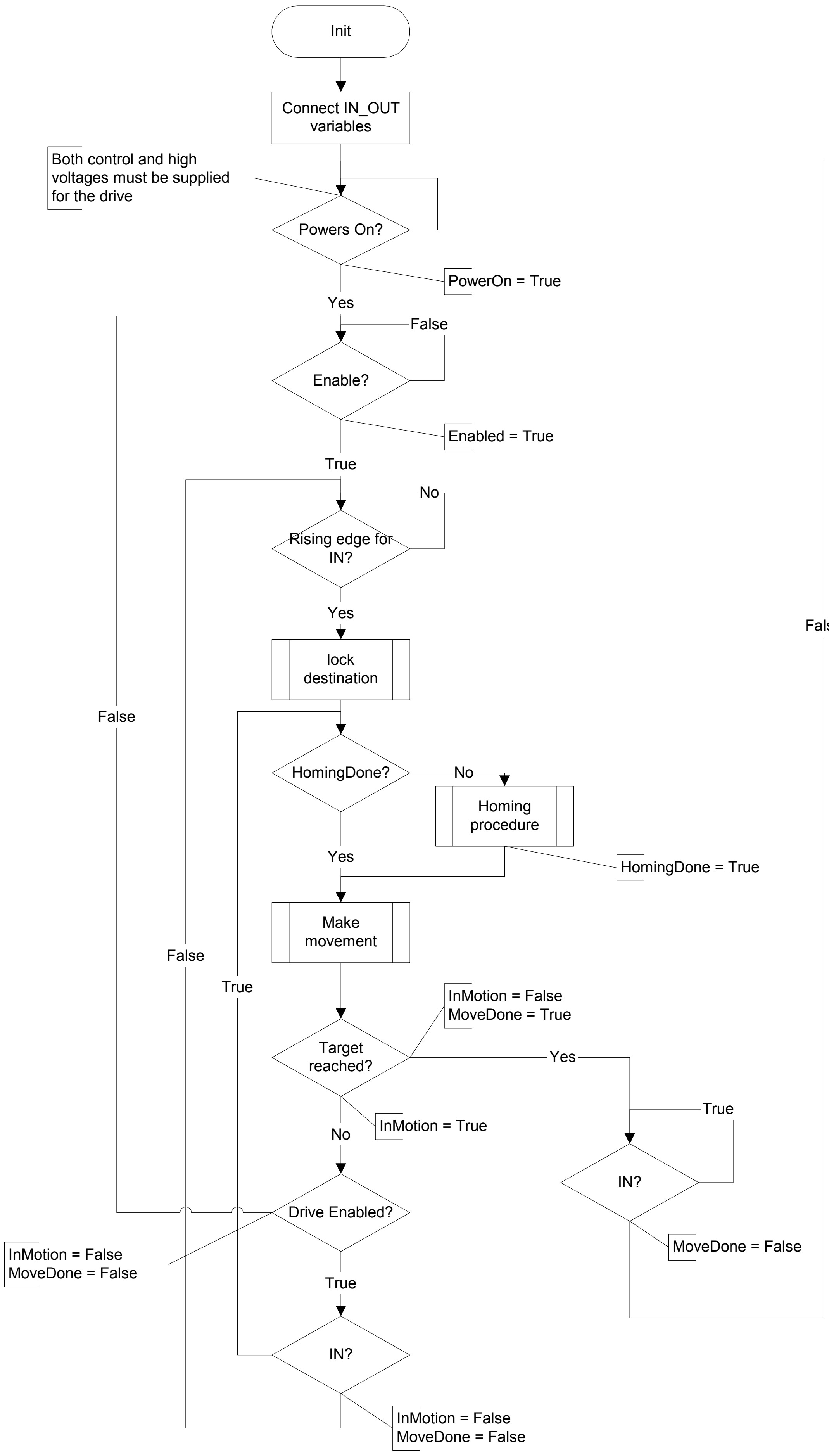

Init
Connect IN_OUT variables
Both control and high voltages must be supplied for the drive
Powers On?
PowerOn = True
Yes
False
Enable?
Enabled = True
True
No
Rising edge for IN?
Yes
False
lock destination
False
HomingDone?
No
Homing procedure
HomingDone = True
Yes
Make movement
False
True
InMotion = False
MoveDone = True
Target reached?
Yes
True
InMotion = True
No
IN?
Drive Enabled?
MoveDone = False
InMotion = False
MoveDone = False
True
IN?
InMotion = False
MoveDone = False

Figure 6 Flow chart diagram of FB operation

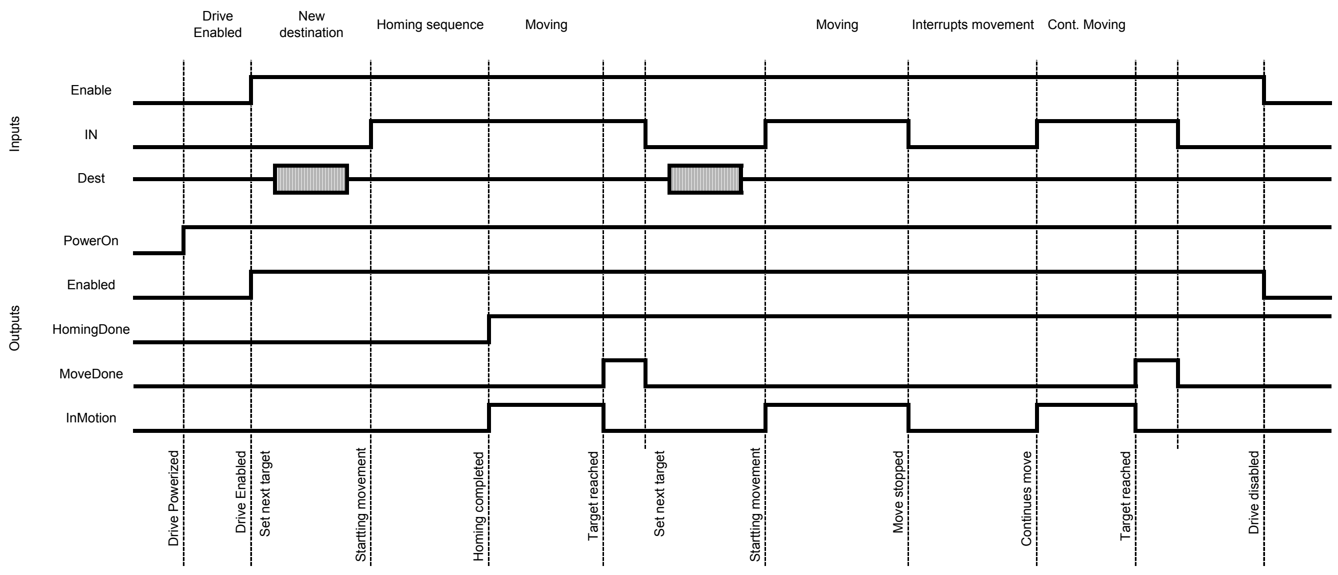


Figure 7 Timing diagram of the motion conterol FB operations

The functional operation of the motion control FB is presented in the Figure 6 and Figure 7.

The former presents the working principle of the Function Block including the variable values and the states of the output variables in the specific points of execution of the FB. The later presentes the timing diagram and relations of signals states to each other.

Important is to notice that at any point during the movement either *bEnable* or *bStart* signal is changed to false state the movement will immediatelly stop. The ongoing movement is continued when both mentioned signals are again true.

Another point to notice is that *bMoveDone* signal will drop down to false state at the same moment as the *bStart* signal is brought to the state of false.

Homing sequence is done only once before the first movement or it might be even left totally out of the sequence. This will be the case if the voltage has been supplied to the drive controller nonstop and the motor have not been moved by other than the drive itself (manually) since last homing cycle. In this case the previous homing cycle is still valid and the absolute position is known for the drive.

1.4 I/Os AND VARIABLES OF THE SYSTEM

1.4.1 I/O Variables

id	Name of I/O	Address	Components	TRUE	FALSE
			BIT, Input		
SW 10	ISensorOpt	AT %IX0.0	Optical sensor. Located in the sorter lift	Object in front of sensor.	No object in front of sensor.
SW 11	ISensorInd	AT %IX0.1	Inductive Sensor. Located in the 2nd level of sorter.	Non metallic	Metallic object
	IEmergencyClear	AT %IX0.2	State of the Emergency stop circuit	Emergency Stop Deactivated → Everything OK! (Note signalling, if wire cuts!!!)	Emergency Stop Pushed.
	IRun	AT %IX0.3	System should be running. State of RUN switch	ON. System is running.	OFF. Stop the system in next NICE position.
SW 3	ILimitUp	AT %IX0.4	Upper Limit Switch	Lift out of range.	Lift working normally
SW 2	ILimitLow	AT %IX0.5	Lower Limit Switch	Lift out of range.	Lift working normally
SW 20	IGateClosedUp	AT %IX0.6	State of upper gate	Closed	Not closed
SW 21	IGateClosedMid	AT %IX0.7	State of middle gate	Closed	Not closed
SW 22	IGateClosedLow	AT %IX1.0	State of lower gate	Closed	Not closed
SW 23	ISorterGateClosed	AT %IX1.1	State of sorter gate	Closed	Not closed
SW 24	ISorterLiftUp	AT %IX1.2	State of sorter lift	Up position	Not up
SW 25	ISorterLiftDown	AT %IX1.3		Down position	Not down
SW 12	IPressureApplied	AT %IX1.4	Pneumatic pressure available.	Pressurized	No pressure
	TEMP1	AT %IX1.5	Not used		
SW 13	ISenColourYellow	AT %IX1.6	Colour of the ball in front of Colour sensor	Yellow ball	Not yellow
SW 13	ISenColourBlue	AT %IX1.7		Blue ball	Not blue
SW 13	ISenColourGreen	AT %IX2.0		Green ball	Not green
SW 13	ISenColourRed	AT %IX2.1		Red ball	Not red
			BIT, Output		
Y1	OGateUp	AT %QX0.0	Gate valve + cylinder for upper pipe	Gate open	Gate closed
Y2	OGateMid	AT %QX0.1	Gate valve + cylinder for middle pipe	Gate open	Gate closed
Y3	OGateLow	AT %QX0.2	Gate valve + cylinder for lower pipe	Gate open	Gate closed
Y4	OSorterGate	AT %QX0.3	Valve + cylinder for sorter gate	Gate open	Gate closed
Y8	OSorterLift	AT %QX0.4	Valve + cylinder for sorter lift	Sorter lift up	Sorter lift down
	OEjector	AT %QX0.5	Solenoid in the lift used for ejecting the ball from lift.	Solenoid out (Lift ejects the ball).	Solenoid in (Lift carries ball)
			WORD		
	IDriveStatus	AT%IW3	Drive Controller Status Word (Output from Drive)		
	ODriveControl	AT%QW3	Drive Controller Control Word (Input to Drive)		

1.4.2 Global variables

id	Name of var	Type	(initial) value	Description
	CountBallsSortedMetal	INT	0	Balls sorted: Metal
	CountBallsSortedWooden	INT	0	Balls sorted: Wood
	CountBallsSortedYellow	INT	0	Balls sorted: Wood, Yellow
	CountBallsSortedBlue	INT	0	Balls sorted: Wood, Blue
	fbDrive	DriveMovement2		

1.4.3 Global Constants

id	Name of var	Type	(initial) value	Description
	GPOS_LOADING	WORD	16#0A	Drive to loading level
	GPOS_PIPE_LOW	WORD	16#0B	Drive to lower pipe
	GPOS_PIPE_MID	WORD	16#0C	Drive to middle pipe
	GPOS_PIPE_UP	WORD	16#0D	Drive to upper pipe
	GPOS_GARBAGE	WORD	16#0E	Drive to garbage pipe
	GPOS_COLOR_SEN	WORD	16#0F	Drive to colour sensor

1.5 PROGRAM SEQUENCE

The program for controlling the system can be made by using only Main Program. The Main Program is controlling I/Os and the drive control. Additional Safety Program can be made for error situations and error recovery. This additional part is out of basic requirements. Drive control can be alternatively made as separate block.

1.5.1 MAIN PROGRAM

The easiest way to construct the program is to use SFC (=Structured Flow Charting) as the organisational language and then ST (= Structured Text) for programming the different Steps and Transitions. Also it is a good idea to split the SFC diagram into several hierarchical levels.

It could be wise to divide program into few functional blocks, which perform a certain specific task. (e.g. initialisation, …)

The program starts running by resetting the emergency circuitry (**IEmergencyClear**) and switching **IRun** signal on.

0: The initial state

Initialisation cycle is executed at the beginning of program to bring the system into known initial state. After this the system will be up and can be started to run. This cycle also makes all positions where a ball might be into a known state.

Note that all locations do not have sensing capabilities.

Think case when execution has been stopped with braking emergency stop circuit.
Things to be checked and actions to be made

- Powers are supplied both pneumatic &electrical.
- All gates are in down position (balls in the system are stopped)
- For lift is made homing procedure E.g. position of lift will be known.
- If there are balls in unknown/undefined places they are moved in controlled way into proper (=known) places.
- …

1: Ball in front of the Optical sensor & Sorter lift
If there is an object in front of the Optical sensor you lift it to the 2nd level of sorter. Sorter lift is capable of lifting one single ball at time. After lifting you need to wait a moment (e.g. 1 sec) for letting the ball to roll from the sorter lift before you can move it back to the down position.

2: Ball in front of the Inductive sensor & Sorter gate
With the Inductive sensor you will identify whether the material of the ball is metallic or non-metallic. If sensor's value is FALSE the ball is a metallic one, if it is TRUE then the ball is a non-metallic one. You can use some variable (e.g. the internal variable F_BallType) for storing that value for the later use. After that you drive up Sorter gate and let the ball roll into the lift. Make sure that lift is in right level before opening the gate.

Then close the Sorter gate after a delay (e.g. 1.5 sec). You need that delay, because the wooden balls are rolling very slowly, so there must be enough time for those to roll in to the lift.

At the same time, you can you send to the Drive Controller the number of the Process block performing the movement you wish to make next and the enable bit for the drive. Indeed the Enable bit shall be TRUE all the time.

3: Ball is in the lift
You have to send to the Drive Controller the Start bit. See chapter 1.3.3/1.3.4 for detailed use of the drive.

4: Colour sensor
Colour sensor identifies the colour of the ball. Use the lift to drive a single ball at front of the colour sensor. After the lift is stopped you can read the colour of the ball with four inputs connected to the colour sensor (SW 13). If the ball is yellow the 1st input is TRUE, if the ball is blue the 2nd input is TRUE, etc. If colour of the ball does not correspond any of the four taught hues, none of those inputs is coming TRUE.
According the result of the sensor information the lift is driven to desired pipe and ball is ejected.

5: Lift is driven up at front of a pipe
Push the ball out from the lift into the pipe with the lift's solenoid i.e. Ejector. At the same moment you send to the Drive Controller the new process block number and the enable bit. After a delay (e.g. 1 sec) you can turn the start bit on and lift starts the movement. At the same moment you release ejector.

6: The lift has been driven back down to the initial position
We are again at the initial step and the system is ready to sort the next ball.

7: Emptying pipes

After all balls are sorted into the pipes they must be emptied. They are emptied pipe after pipe. I.e. Open the gate of a pipe for few seconds, close the gate, open the next gate,…

1.5.2 SAFETY PROGRAM (Optional)

The Safety Program is needed for controlling the Lift in unexpected or faulty conditions.

- Resetting the program after emergency stop.
- E.g. for some reason the drive is driven against the limit switch → Recovery from this situation?

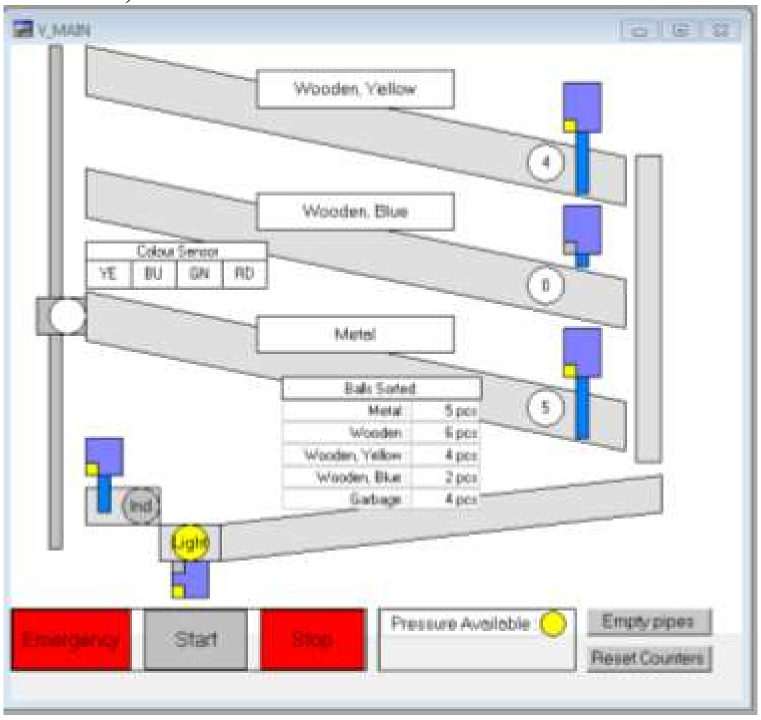
If the Emergency Stop is pushed or if the lift is touching one of the Limit Switches, the program should stop the motor by sending to the *Drive Controller Control Word* a signal to disable and stop the drive (for example 0000 0000 0000 00**00**).

The execution of this program is simultaneous to the execution of the Main Program, so it has to be configured a new task in PLC Control and assigned to this Safety Program.

1.6 Human Machine Interface (HMI)

Human Machine Interface (HMI) is made in this case with TwinCAT from Beckhoff. This is the easiest method for making the HMI. Other possibilities are: Dedicated application programmed with e.g. C#/C++/C, VisualBasic, Java, …; or Web based HMI application.

One possible HMI is illustrated in Figure 8. Notice gate and ejector actuation, illustration of sensor states by lights (yellow vs. grey), number of balls in pipes and sorted, lift position, running state of device, buttons for user action, …



Balls Sorted	
Metal	5 pcs
Wooden	6 pcs
Wooden, Yellow	4 pcs
Wooden, Blue	2 pcs
Garbage	4 pcs

Figure 8 Illustrative HMI Example. Main Screen (Running)

Components used for making the HMI:

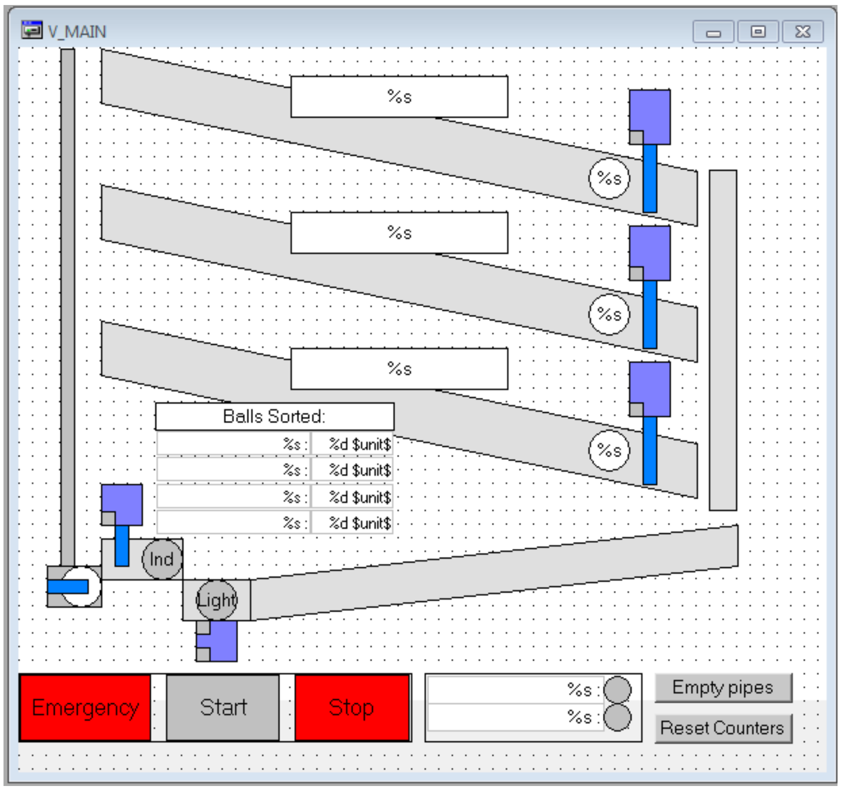


Figure 9 Main Screen (Design view)

Some visualisation components:

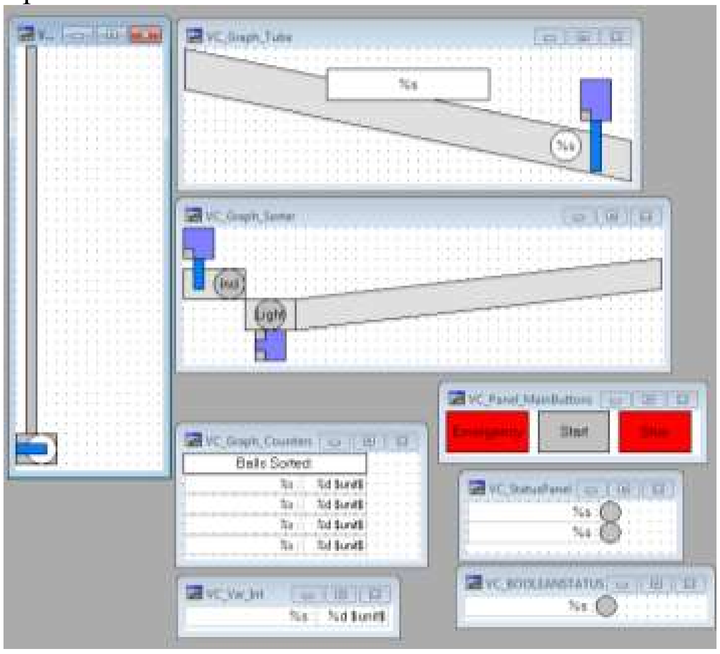


Figure 10 Visualisation components

1.7 Components used

The components of the system are:

Location	***Vendor***	***Device***	***Quantity***	***Remarks/Features***

Location	***Vendor***	***Device***	***Quantity***	***Remarks/Features***
	~~Beckhoff~~	~~FC3101~~	~~1~~	~~Profibus-DP fieldbus card~~
Main ctrl	Beckhoff	CX8090	1	Controller PLC (Embedded PC for Ethernet)
Main ctrl	Beckhoff	EL6731	1	Profibus-DP master module
Main ctrl	Beckhoff	EL9011	1	Endcap
IO mod	Beckhoff	BK 3000	1	Profibus-DP buscoupler
IO mod	Wago	750-401 comp. KL1012	3	Input Terminals 2Ch. 0,2ms
IO mod	Wago	750-403 comp. KL1114	3	Input Terminals 4Ch. 0,2ms
IO mod	Wago	740-504 comp. KL2114	2	Output Terminals 4Ch. 0,5A
IO mod	Beckhoff	KL9010	1	End Terminal
Drive	Indramat	Ecodrive DKC03.1-040-7	1	Drive controller with Profibus connection
Drive	Indramat	MKD025B-144-GG0	1	Digital AC Servo Motor
Sensors	Omron	E3MC-MX41	1	Colour sensor (4 teachable colours)
Sensors	Omron	E3X-NT41	1	Fiberoptic gate switch
Actuators	SMC		1	Valve terminal base
Actuators	SMC	VQ1101N-5	6	Pneumatic valve 2/5
Actuators	SMC	VQ1201N-5	2	Pneumatic valve 2x 2/3
Actuators	Festo	DSNU-10-20-P-A	5	Pneumatic cylinders
Actuators	Festo	SMEO-4U-S-LED-24B	6	Proximity sensors for cylinders
Sensors	Turck	XS1N08P8340S	1	Inductive sensor
	Mascot	Power Supply 24V	1	Power Supplies for I/O modules

1.8 ADDITIONAL INFORMATION

Profibus Cable
Transmission Speeds available: 9.6 to 500 kbit/s and 1.5 to 12 Mbit/s
Typical Reaction Time: 3msec
Maximun number of I/O terminals per buscoupler: 64 (128 I/Os)

TwinCAT
Beckhoff TwinCAT is an automation system offering a fieldbus independent I/O interface and integration in Windows operating system.

TwinCAT 2.8 demo version for Windows NT/2000 can be downloaded for free from the site *http://www.beckhoff.com/english.htm*

1.9 MORE INFO

TwinCAT Soft PLC	http://www.beckhoff.com
Fieldbus card	http://www.hilscher.com http://www.beckhoff.com
Buscouplers and I/O terminals	http://www.beckhoff.com http://www.wago.com
Drive Controllers and Servo Motors	http://www.indramat.com
IEC 61131-3 Programming Languages	http://www.plcopen.org/

2 THE PLC STANDARD IEC 61131-3

Five different languages conforming to the IEC 61131-3 are available for creating PLC programmes. These languages could be divided in 2 textual ones (IL and ST), 2 graphical ones (LD and FBD) and 1 organisational language (SFC).

2.1 INSTRUCTION LIST (IL)

The instruction list is very similar to the STEP 5 programming language for Siemens PLCs. Every instruction begins in a new line and contains one operator and one or several operands. An instruction may be preceded by a label, followed by a colon. A comment must be the last element in one line.

Example:

```
Start:      LD    Basin_level    (* Load level *)
            GE    13             (* Limit reached? *)
            JMPC  Pump_on
            R     Pump_control   (* Pump off *)
            JMP   End
Pump_on:    S     Pump_control   (* Pump on *)
End:
```

2.2 STRUCTURED TEXT (ST)

In the case of this programming language, we also speak of a higher-level programming language because it is not "machine-oriented" commands that are used. Instead, power command strings can be created by way of abstract commands. From the area of the PC, Basic, PASCAL and C are comparable higher-level programming languages.

Example:

```
CASE Temperature_furnace OF             (* Control heating output *)
        60..99:     Heating := 80;      (* 80% *)
        100..149:   Heating := 60;      (* 60% *)
        150..199:   Heating := 35;      (* 35% *)
        200..250:   Heating := 10;      (* 10% *)
        ELSE:       Alarm := TRUE;      (* Set alarm *)
END_CASE;
```

2.3 LADDER DIAGRAM (LD)

The representation of logical sequences in the form of the Ladder Diagram originates from the area of electrotechnical plant engineering. This mode of representation is particularly suitable for implementing relay switching operations in PLC programmes.

Example:

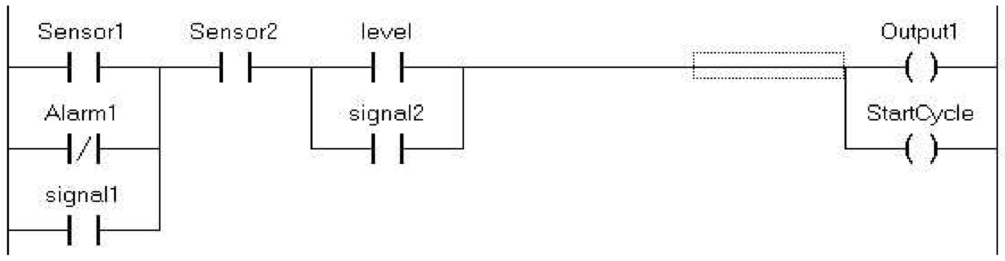


Figure 11 Example Ladder Diagram (LD)

2.4 FUNCTION BLOCK DIAGRAM (FBD)

The basic idea behind PLC programming with the Function Block Diagram is that the programme is structured in function-oriented logical sequence cascades (networks). Within one network, the executed direction is always from left to right. All input values must always have been generated before execution of a function block. Evaluation of a network is not concluded until the output values of all elements have been calculated.

Example:

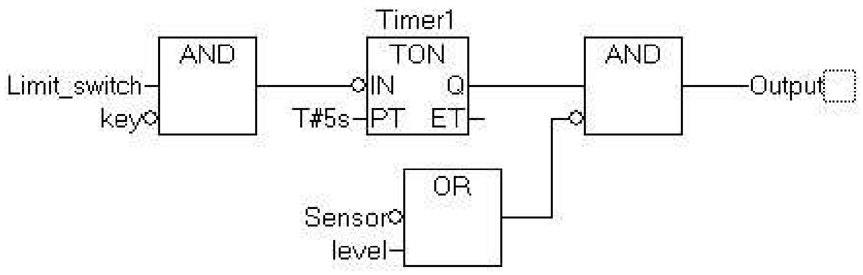


Figure 12 Example Function Block Diagram (FBD)

2.5 SEQUENTIAL FUNCTION CHART (SFC)

The SFC is expedient wherever sequencer programming is necessary. Complex tasks are split into clearly arranged portions of programme (steps). The sequence between these steps is then defined graphically. The steps themselves are created in a different programming language (ST, IL, etc) or can be also represented in SFC again.

SFC programmes essentially consist of steps, transitions and their links. Each step is assigned a set of commands. These commands are executed when the step is active. A transition must be fulfilled to ensure that the next step is executed. The steps and the transitions can be formulated in any chosen language.

Example:

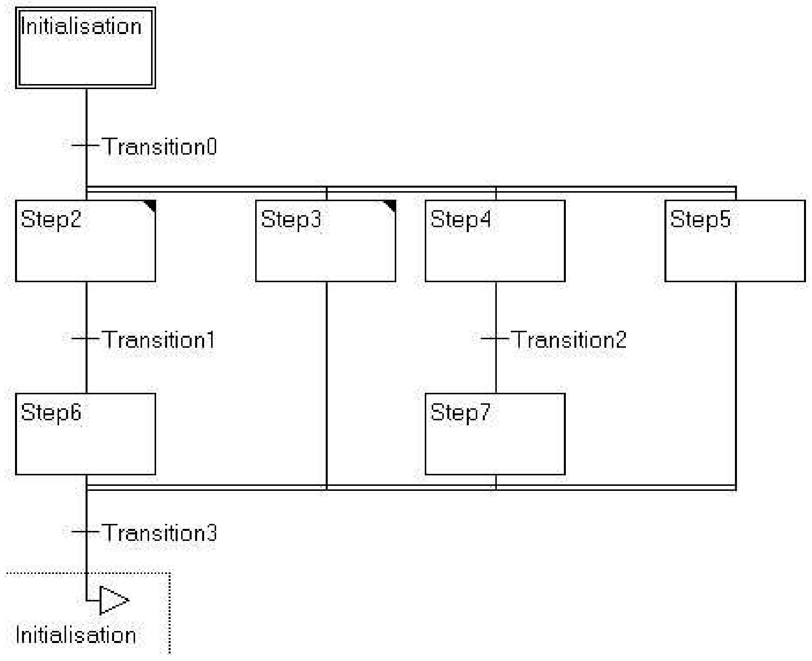


Figure 13 Example Sequential Function Chart (SFC)

3 INTRODUCTION TO WORKING WITH TWINCAT

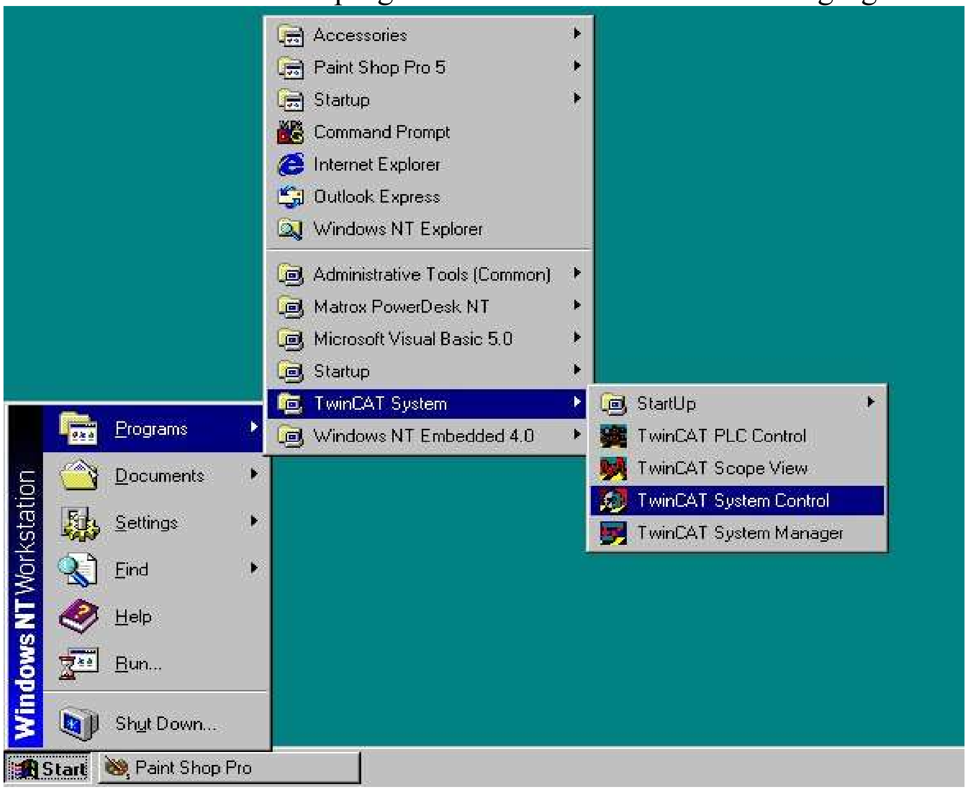
3.1 FIRST STEPS

3.1.1 SYSTEM REQUIREMENTS

To operate TwinCAT it is needed a PC with a 486 type processor or higher, and the computer must contain at least 16 MB of RAM. TwinCAT as most of the PC-Based control packages on the market only runs at the moment under the Windows NT v4.0 Operating System.

3.1.2 STARTING THE PROGRAMME

TwinCAT is started from the menu of programmes as shown in the following figure.

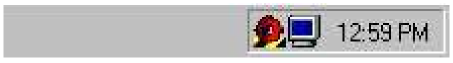


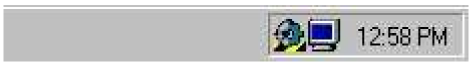
TwinCAT is formed by different software programmes. The modules are:

- *TwinCAT PLC Control*: This is the software development kit for IEC 61131-3. The PLC programmes are written, managed and tested here.
- *TwinCAT System Manager*: Used to assign the physical I/O addresses (field bus) to the logical process variables (PLC programme, NC/CNC). This is also referred as mapping.
- *TwinCAT System Control*: Apart from the visible programmes, there are also hidden tasks and drivers that run in the background. TwinCAT System Control administrates these programmes.
- *TwinCAT Scope View*: With the help of this module, process values can be displayed graphical in real time.

3.1.3 TWINCAT IN WINDOWS NT

When the System is started, the TwinCAT Real Time server icon is displayed on the right of the Windows NT Taskbar as shown in the figures below.

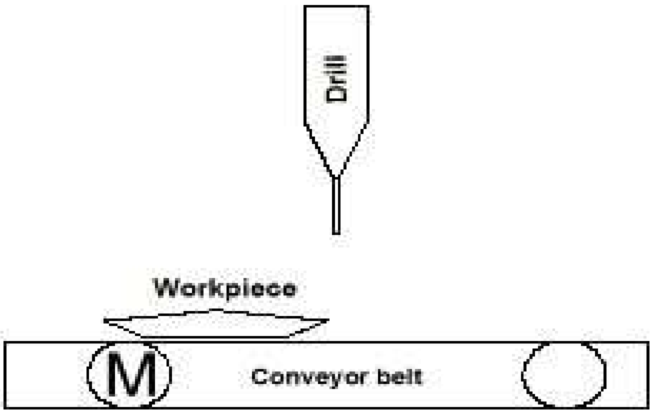




The color indicates the general operating state of the system. There are 3 different possible states: "Started" (green), "Starting" (yellow) and "Stopped" (Red). By clicking the icon, a pop up menu opens in which you can define further system settings. Also the TwinCAT server can be stopped and started in this menu.

3.2 EXAMPLE PROGRAMME

The creation of applications using TwinCAT will be explained with reference to an example programme. This programme represents a machine tool for any chosen workpieces. This project can be found in the directory "\TwinCAT\Samples\FirstSteps", it has the name "Maschine.pro".

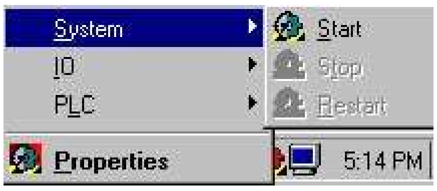


Description:

1- The conveyor belt is moved by 25 steps
2- The drill is moved down for 2 sec.
3- The drill is moved up for 2 sec.
4- Begin at step 1 again.

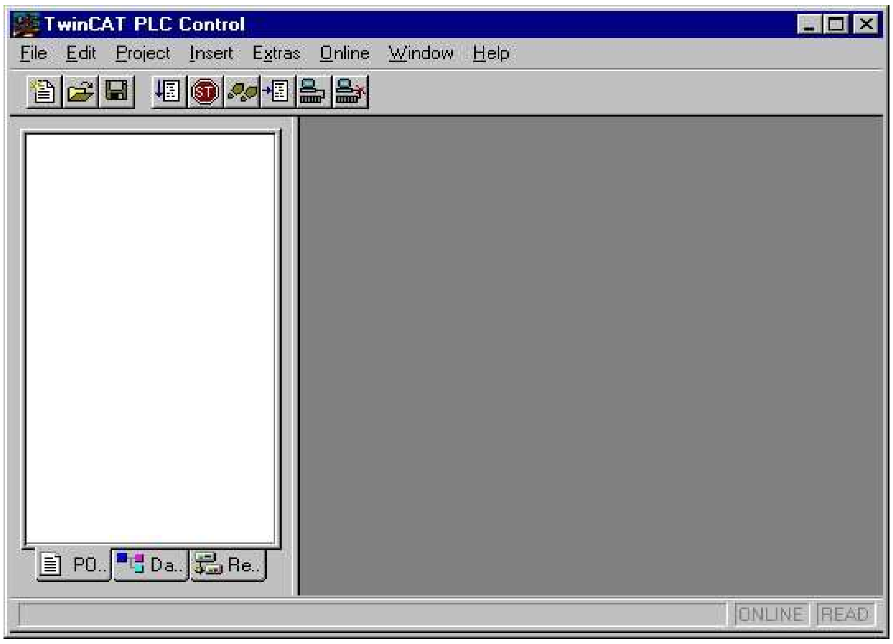
3.2.1 STARTING TWINCAT

Before you are able to execute the programme, you must activate the TwinCAT real time server. For doing this, click on the TwinCAT real time server icon and activate the "Start" command from the "System" menu. The color of the icon changes through yellow to green, which means that TwinCAT's real time kernel is active.



3.2.2 STARTING TWINCAT PLC CONTROL

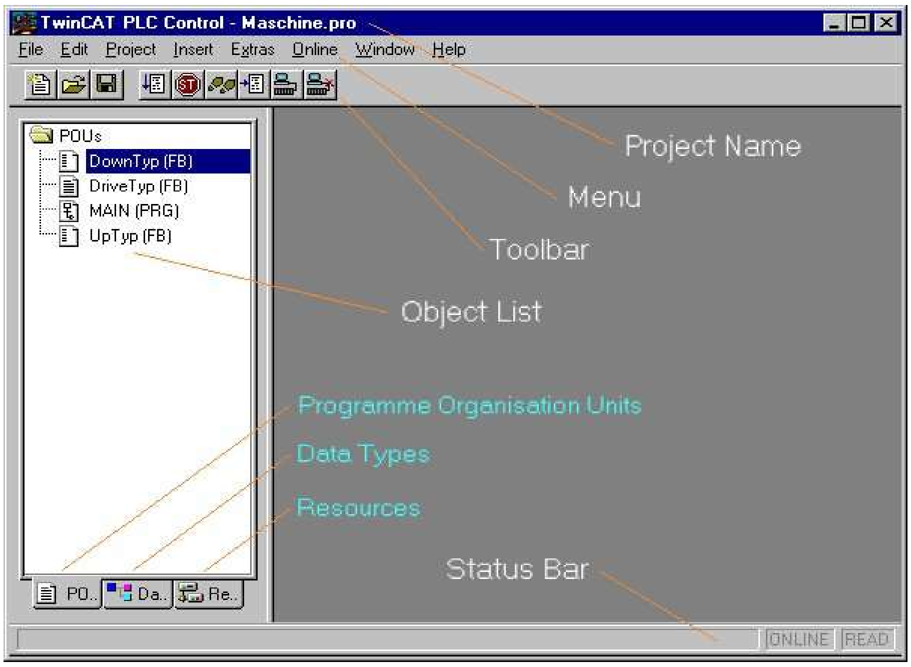
Start now the programming surface from the TwinCAT PLC. For doing this click with the mouse "Start", "Programs", "TwinCAT system" and finally "TwinCAT PLC control".



3.2.3 OPENING A PROJECT

A PLC project is stored in a file on the hard disk or on a diskette that bears the name of the project. To open a project, select the "File" menu and then the "Open" command. Then, switch to the directory specified above (\TwinCAT\Samples\FirstSteps) and select the "Maschine.pro" project.

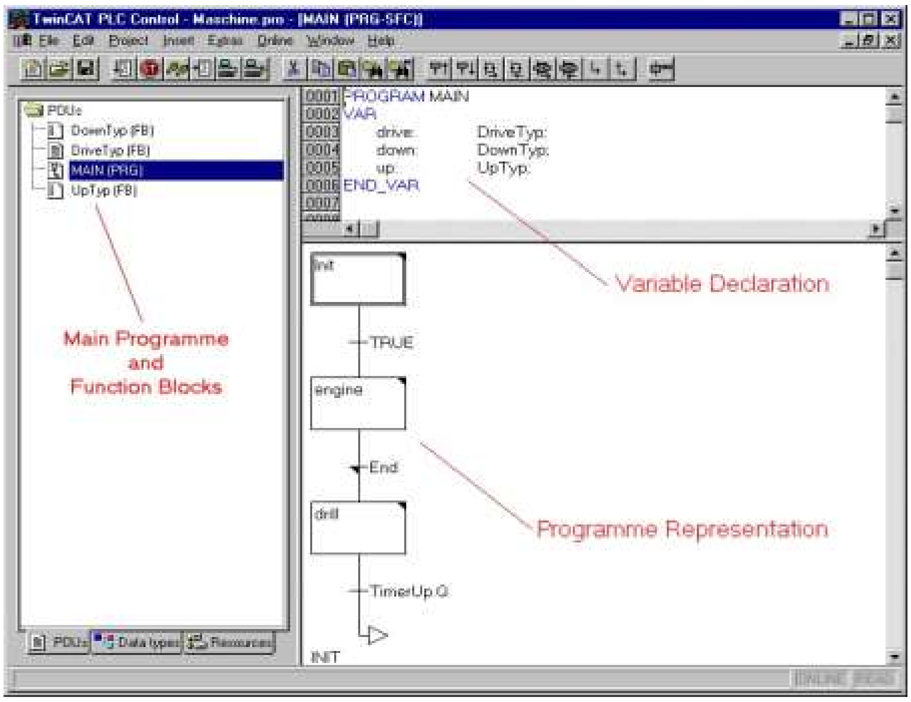
3.2.4 ITEMS OF THE PLC-CONTROL



TwinCAT distinguishes 3 kinds of basic objects in a project:

- Programme Organisation Units (Programme Blocks)
- Data Types
- Resources

To select a programme object, open the pick list and select "function blocks" (this may already be set as the default. All programme objects belonging to the project are now listed. Then double click in the "MAIN (PRG)" entry, and the contents of the "MAIN" programme object are displayed.



```
PROGRAM MAIN
VAR
        drive:      DriveTyp;
        down:       DownTyp;
        up:         UpTyp;
END_VAR
```

Variable Declaration

A PLC programme stores its data in variables. Variables are comparable to flag words or data words. Before a variable can be used, it must be declared, i.e. its affiliation to a specific data type (e.g. BYTE or REAL) must be made known. Declaration also involves defining specific attributes such as battery buffering, initial values or affilation to physical addresses.

If a variable is not needed in the input or output image, i.e. only within the PLC programme, the PLC programmer need not worry about the storage location of the data. This is taken care of by TwinCAT. This avoids unintentional overlapping of flag words/data words as was possible in previous systems (side effects).

As in the case of variables, function blocks also have to be declared (instance). In the example, one instance each (drive, down and up) of the 3 functions "DriveType", "DownType" and "UpType" are created. After instancing, the instances can be used and can be activated.

Programme Representation

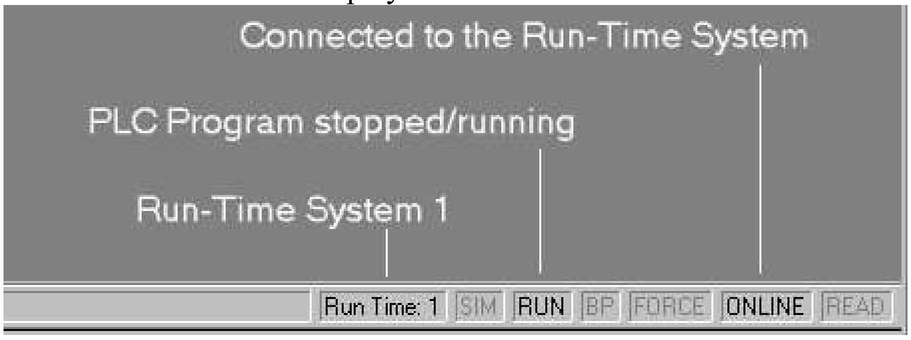
The actual PLC programme is entered and represented in this area of TwinCAT PLC control. As explained further above, TwinCAT supports 5 different languages, which are standardised by IEC 61131-3.

3.2.5 EXECUTION OF THE PROGRAMME

Logging In

You have now loaded the PLC programme in TwinCAT PLC control and you are able to execute it. Make sure the TwinCAT real time server is active. This is recognizable by the fact that the TwinCAT real time server icon is displayed in green on the bottom right of the screen. Before you start a PLC programme, you must link TwinCAT PLC control to the run time system, i.e. you must "log in" with the control system. Execute the "Log In" command in the "Online" menu. As there is still no PLC programme in the run time system, you are then prompted to specify whether or not you wish to download the programme to the control system. Answer "Yes" in response to this prompt.

The current status of the connection is displayed in the status line:



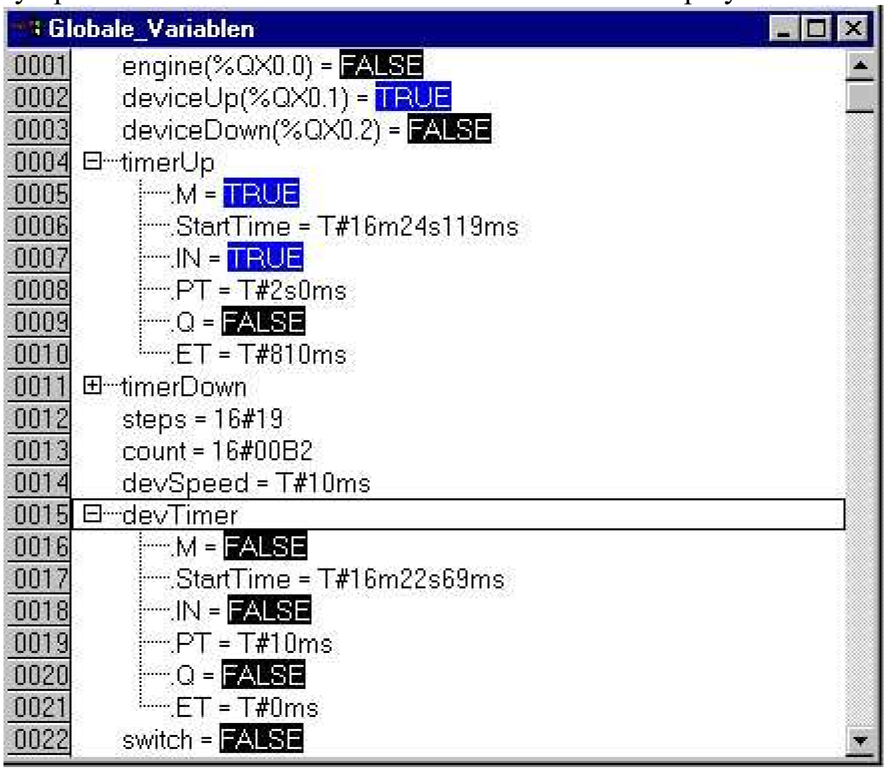
Start PLC programme

You start the PLC programme in the TwinCAT real time server by selecting the "Start" command from the "Online" menu. The word "RUN" displayed in the status line darkens. You should also see that individual steps within the SFC are temporarily displayed in blue. A step shown in blue is currently being executed, i.e. it is the active step.

Tracing the PLC programme sequence

When you activate the "Global Variables" window by clicking the "Resources" tab at the bottom of the "Object List" window, and then double clicking the "Global Variables" object, you will see all globally declared variables. Global variables can be used jointly by all programme objects (POUs).

Besides the variables, the "timerUp", "timerDown" and "devTimer" function blocks are also displayed there. A lozenge is visible before the function names. When you double click the lozenge, a tree-like display opens in which all variables of the function are displayed.



Globale_Variablen

```
engine(%QX0.0) = FALSE
deviceUp(%QX0.1) = TRUE
deviceDown(%QX0.2) = FALSE
⊟timerUp
    .M = TRUE
    .StartTime = T#16m24s119ms
    .IN = TRUE
    .PT = T#2s0ms
    .Q = FALSE
    .ET = T#810ms
⊞timerDown
steps = 16#19
count = 16#00B2
devSpeed = T#10ms
⊟devTimer
    .M = FALSE
    .StartTime = T#16m22s69ms
    .IN = FALSE
    .PT = T#10ms
    .Q = FALSE
    .ET = T#0ms
switch = FALSE
```

Online Monitoring/debugging
When program is running program flow (which lines are executed on this cycle) can be highlighted for the debug task from (Online→Display Flow Control. Shortcut: Ctrl+F11)

Break points can be assigned (Online →Toggle breakpoint. Shortcut: F9)

Stop the PLC programme
You have now loaded a PLC programme into TwinCAT PLC control and you have executed it on the TwinCAT PLC server (run time system). Now for ending the PLC programme, select the "Stop" command in the "Online" menu.

Logging Out
For adding or modifying the PLC programme is necessary to log out first from the TwinCAT PLC server. To do this, execute the "Log Out" command in the "Online" menu.

3.2.6 VIEWING PROGRAMME COMPONENTS

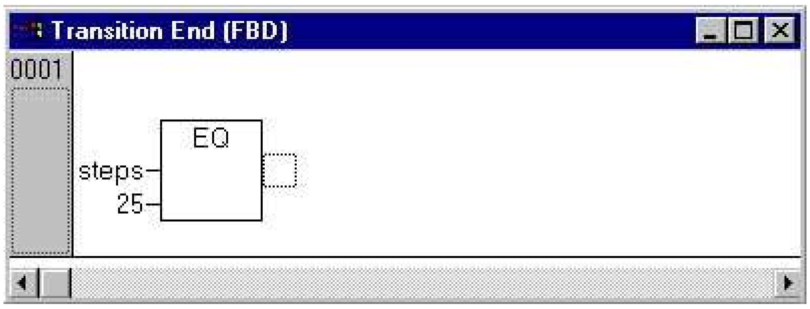
Viewing programme text
To look at the code for the MAIN program click the POU tab of the "Object List", and double click the object labeled "MAIN". This example was programmed in the various IEC 61131-3 programming languages. The main part of the programme was created in the SFC. It contains the steps "Init", "Engine" and "Drill", and the transitions "TRUE", "End" and "TimerUp.Q".

Viewing transitions
The "TRUE" transition is constantly fulfilled because the "TRUE" key word is a system-wide constant and is permanently fulfilled. The "Engine" step is executed unconditionally after the "INIT" step.

TimerUp.Q means that the variable "Q" in the "UP" function must be TRUE for this transition to be fulfilled.

"End" is a transition that contains further programme text. When you double click the transition, a further window opens in which the corresponding programme text is displayed.

In the "End" transition a comparison is made as to whether the 25 steps of the motor have already been reached. If this is the case, the programme changes in the next cycle from the "engine" step to the "drill" step.

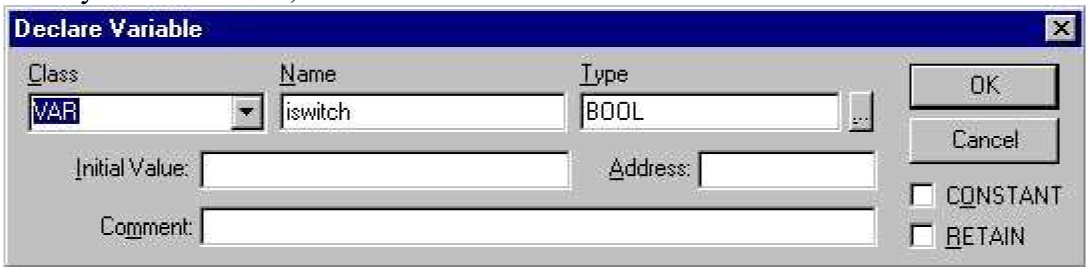


3.2.7 MODIFYING THE PLC PROGRAMME

Switch back to the "MAIN" window. At this point, you should modify the PLC programme in such a way that the motor's cycle speed can be modified in 2 stages (fast/slow) by way of a variable.

Open the "Drive Type" function block in the object list by double clicking it. Move the input cursor to the first line and enter the following text:
IF iswitch = TRUE THEN

When the "Enter" (Return) key is pressed, a dialog box appears which you must fill out as shown below. When you select "OK", the "iswitch" variable is added to the variable list of MAIN.

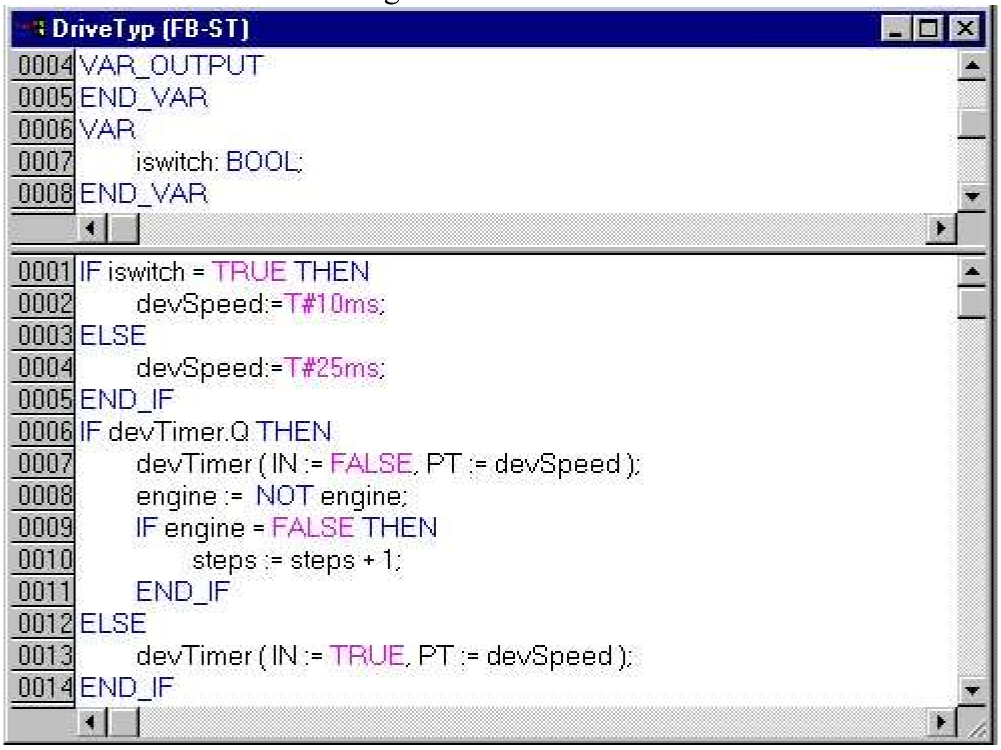

Declare Variable

Class: VAR
Name: iswitch
Type: BOOL
Initial Value:
Address:
Comment:
OK
Cancel
CONSTANT
RETAIN

Then also enter the following programme lines:

```
        devSpeed := T#10ms;
ELSE
        devSpeed := T#25ms;
END_IF
```

The window must then have the following contents:


DriveTyp (FB-ST)

```
VAR_OUTPUT
END_VAR
VAR
    iswitch: BOOL;
END_VAR
```

```
IF iswitch = TRUE THEN
    devSpeed:=T#10ms;
ELSE
    devSpeed:=T#25ms;
END_IF
IF devTimer.Q THEN
    devTimer ( IN := FALSE, PT := devSpeed );
    engine := NOT engine;
    IF engine = FALSE THEN
        steps := steps + 1;
    END_IF
ELSE
    devTimer ( IN := TRUE, PT := devSpeed );
END_IF
```

If the "iswitch" variable is set, the "devSpeed" variable is set to 25ms, or otherwise to 10ms. The result of this is that the pulse and pause duration of the clock pulse generator in the following programme lines either amount to 25ms or 10ms.

Saving the programme

Save the modified programme by selecting the "Save As" command in the "File" menu. Enter a different name than "Maschine.pro", so that the original programme is not overwritten.

Compiling and starting the programme

Before a programme can be transferred to the TwinCAT PLC server, it must be compiled, i.e. it must be converted from its textual or graphical representations to a form that is understandable to the control system. To do this, "log in" with the control system and choose "OK" for "Rebuilding the programme". Then you can start the PLC programme.

Modifying variable values

You have the possibility of modifying values of variables while the PLC programme is running. Open the "Global variables" window and double click the "iswitch" entry. The display changes from FALSE to TRUE and the lettering turns red. Up to this time, however, the value in the TwinCAT PLC server has not yet changed. To do this, you must execute the "Write values" command in the "Online" menu item. The lettering turns black again and the "devSpeed" variable changes to 10ms.

4 TWINCAT SYSTEM MANAGER

<u>NOTE</u>: It is not possible to build a bus configuration by using the TwinCAT System Manager in the demo version installed in the computers of the PC class, because these computers do not have the needed hardware attached. Only it can be partly simulated. This step will be done completely during the laboratory exercise.

By means of the TwinCAT System Manager, all input/output interface connections are managed and addressed, and also they are assigned to the I/O data. Each I/O channel can be addressed by a logical name.

As an example, a PLC program developed with TwinCAT PLC control (the program "Maschine.pro") will be connected to the bus terminals that are controlled via the Profibus fieldbus system.

The hardware required for building up this example is:
- PC interface card for Profibus DP and Bus coupler for Profibus DP
- 2 bus terminals with 2 digital outputs each one and 1 bus end terminal (KL9010)
- 24V power supply unit

4.1 VARIABLE DECLARATION

The storage locations (addresses) of variables are managed internally by the system. The programmer does not need to bother about memory management. The PLC programmes operate with symbolic variable names, thus preventing the occurrence of side effects (overlaps) when using variables.

To access the Input/Output level, it is necessary for the programmer to be able to assign a fixed address to individual variables. This is achieved by means of the key word "AT", which must always be specified when declaring a variable. The key word "AT" is followed by several parameters which provide information about the data location (input/output or flag area) and the width of the data (BIT, BYTE, WORD or DWORD).

The variable declaration for the "Maschine.pro" example has the following structure:

```
VAR_GLOBAL
        Engine          AT%QX0.0:       BOOL;
        DeviceUp        AT%QX0.1:       BOOL;
        DeviceDown      AT%QX0.2:       BOOL;
        TimerUp:                        TON;
        TimerDown:                      TON;
        Steps:                          BYTE;
        Count:                          UINT:=0;
        DevSpeed:                       TIME:=t#10ms;
        DevTimer:                       TP;
        Switch:                         BOOL;
END_VAR
```

Where:

	Data Location	*Data Width*	*Meaning*
%			Start of I/O definition
	I		Input
	Q		Output
	M		Flag
		X	Bit (1 bit)
		B	Byte (8 bits)
		W	Word (16 bits)
		D	Double Word (32 bits)

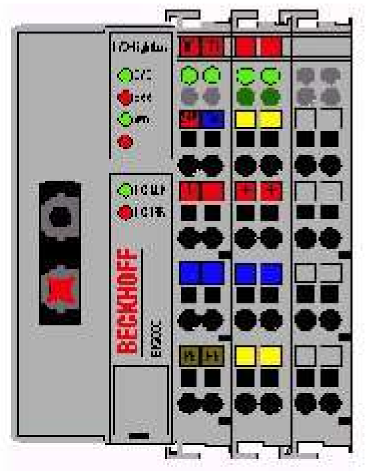
The digits after the data width specify the address of the variable. For bit variables, the address must be specified in the format x.y (ex: 0.1 or 0.2), or simply x for byte, word and double word. The input and outputs are in different storage areas, therefore may contain the same address.

For example, the following variables are located in different addresses:

```
Engine_IN      AT%IX0.0:      BOOL;
Engine_OUT     AT%QX0.0:      BOOL;
```

4.2 SETTING UP THE PROFIBUS BUS TERMINALS

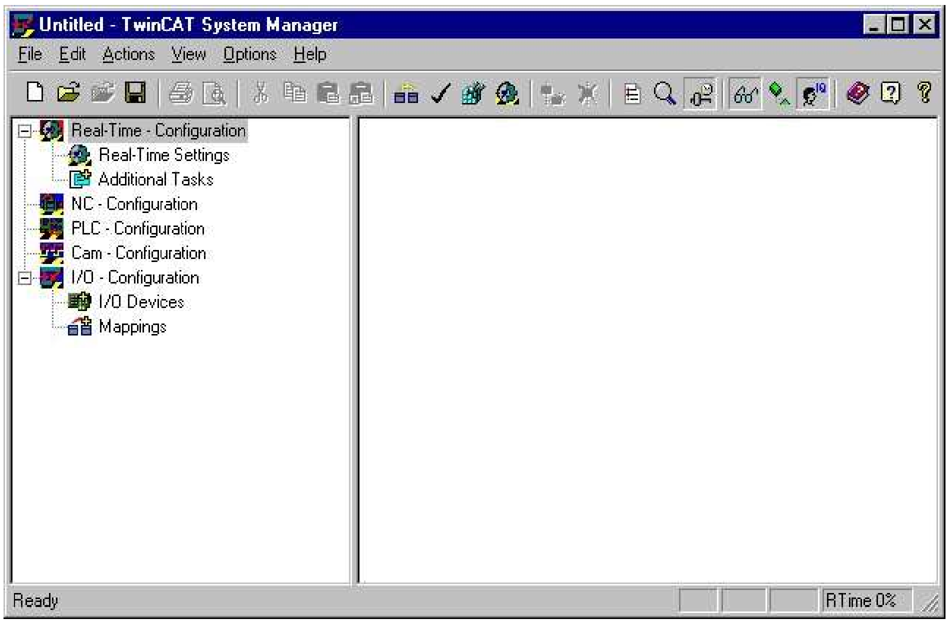
Set up the bus coupler and the bus terminals as shown in the drawing below:



Connect the bus coupler to the Profibus card and power the bus coupler with 24V. Now start the TwinCAT realtime server, unless you have already done so. Once the system has been started, the colour of the icon should be green.

4.3 STARTING THE TWINCAT I/O MANAGER

Start the TwinCAT System Manager by selecting "Start", "Programs", "TwinCAT system" and "TwinCAT System Manager".

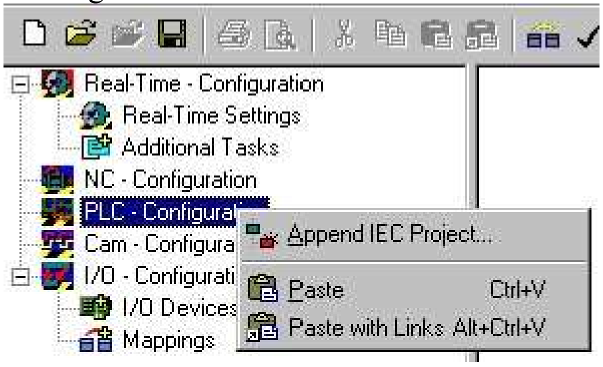


The system configuration is shown as a tree structure on the left side of the System Manager. It consists of the following 4 main points:

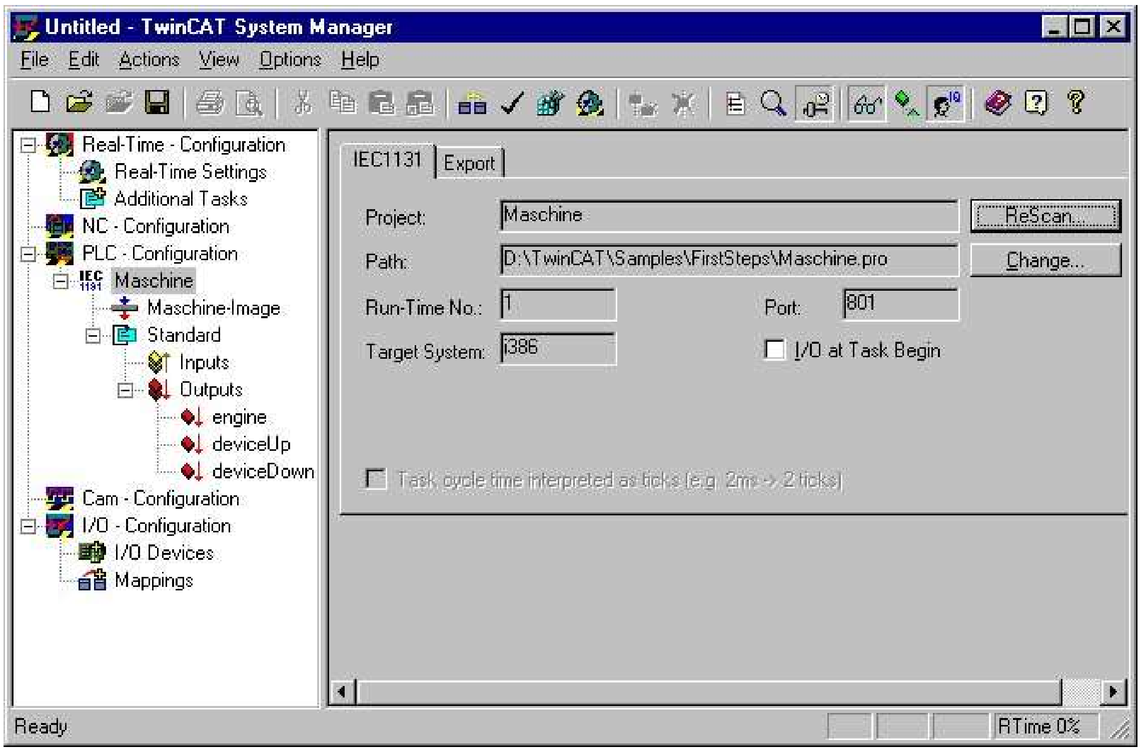
- *RealTime-Configuration*: Set the Real Time Parameters
- *NC-Configuration*: Set the NC (Numerical Control) Parameters
- *PLC-Configuration*: All PLC projects that are required to be configured
- *I/O-Configuration*: In order to link the controller to the process level, the system needs interfaces. This entry offers a list of all interfaces.

4.4 ADDING A PLC PROJECT

The individual PLC projects must be made known to the System Manager so that TwinCAT can access the variables of the PLC programmes. To do this, press the right mouse button while the mouse pointer is over "PLC configuration".



A context menu opens, in which you must select the "Append IEC Project…" entry. Then switch to the "\TwinCAT\Samples\FirstSteps\" directory and select the "maschine.pro" file.

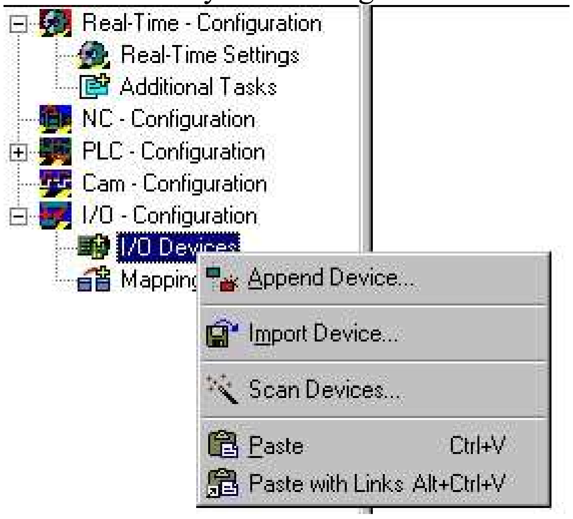


Project: Maschine
Path: D:\TwinCAT\Samples\FirstSteps\Maschine.pro
Run-Time No.: 1
Port: 801
Target System: i386

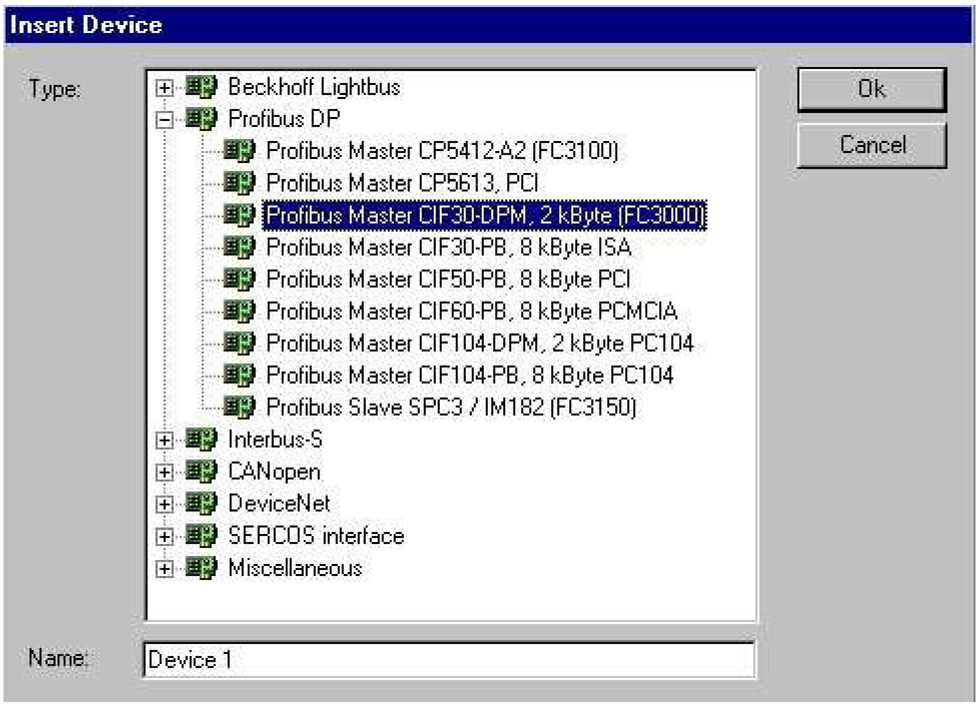
A further point has been added under "PLC configuration" which bears the name of the PLC project. All the variables that have also been declared as inputs/outputs in the PLC program are now listed under the "Inputs" and "Outputs" points.

4.5 ADDING AN I/O DEVICE

Once the PLC project has been added to the PLC configuration, it is necessary to specify the I/O configuration. Select the "I/O Devices" entry with the right mouse button.



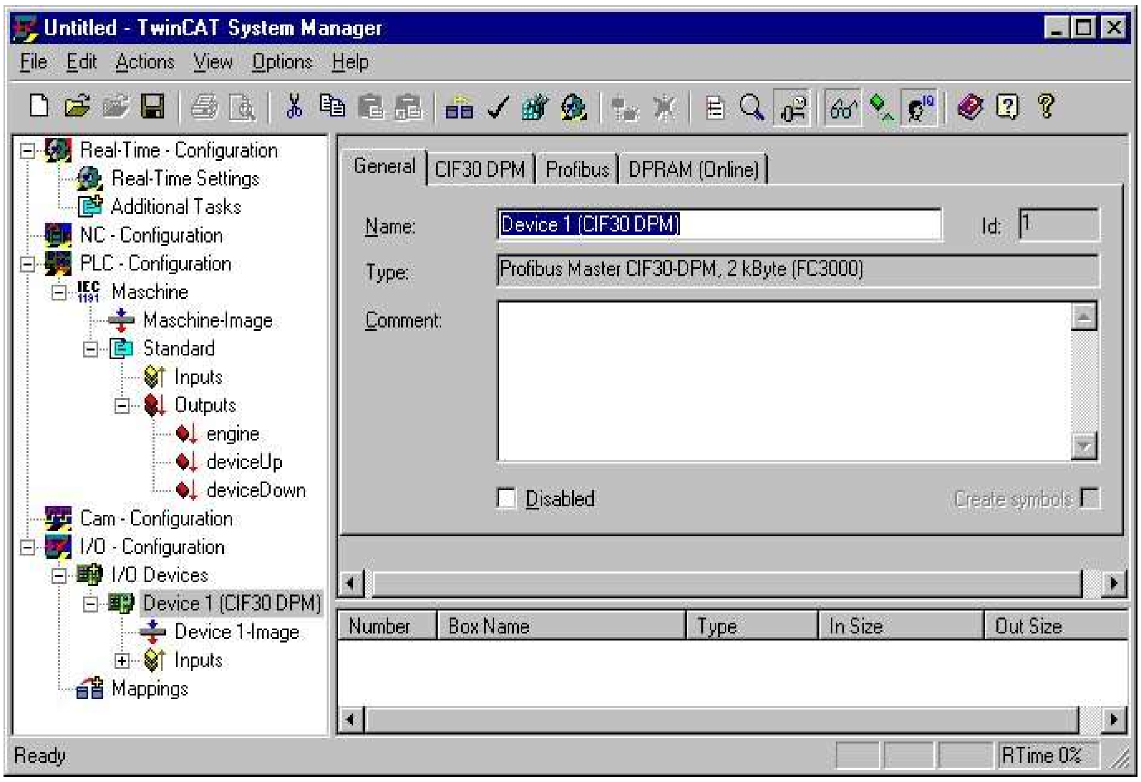
A context menu opens, in which you must select the "Append Device" entry.


Insert Device

Type:
- Beckhoff Lightbus
- Profibus DP
  - Profibus Master CP5412-A2 (FC3100)
  - Profibus Master CP5613, PCI
  - Profibus Master CIF30-DPM, 2 kByte (FC3000)
  - Profibus Master CIF30-PB, 8 kByte ISA
  - Profibus Master CIF50-PB, 8 kByte PCI
  - Profibus Master CIF60-PB, 8 kByte PCMCIA
  - Profibus Master CIF104-DPM, 2 kByte PC104
  - Profibus Master CIF104-PB, 8 kByte PC104
  - Profibus Slave SPC3 / IM182 (FC3150)
- Interbus-S
- CANopen
- DeviceNet
- SERCOS interface
- Miscellaneous

Name: Device 1

Select the device type, in this case "Profibus Master CIF30-DPM". You can choose the device name freely.


Untitled - TwinCAT System Manager

- Real-Time - Configuration
  - Real-Time Settings
  - Additional Tasks
- NC - Configuration
- PLC - Configuration
  - Maschine
    - Maschine-Image
    - Standard
      - Inputs
      - Outputs
        - engine
        - deviceUp
        - deviceDown
- Cam - Configuration
- I/O - Configuration
  - I/O Devices
    - Device 1 (CIF30 DPM)
      - Device 1-Image
      - Inputs
  - Mappings

General | CIF30 DPM | Profibus | DPRAM (Online)

Name: Device 1 (CIF30 DPM) Id: 1

Type: Profibus Master CIF30-DPM, 2 kByte (FC3000)

Comment:

Disabled Create symbols

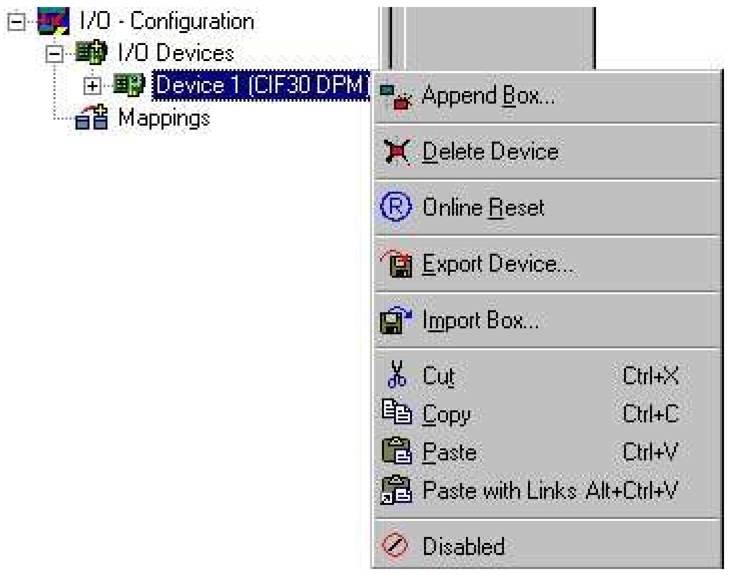
Number	Box Name	Type	In Size	Out Size

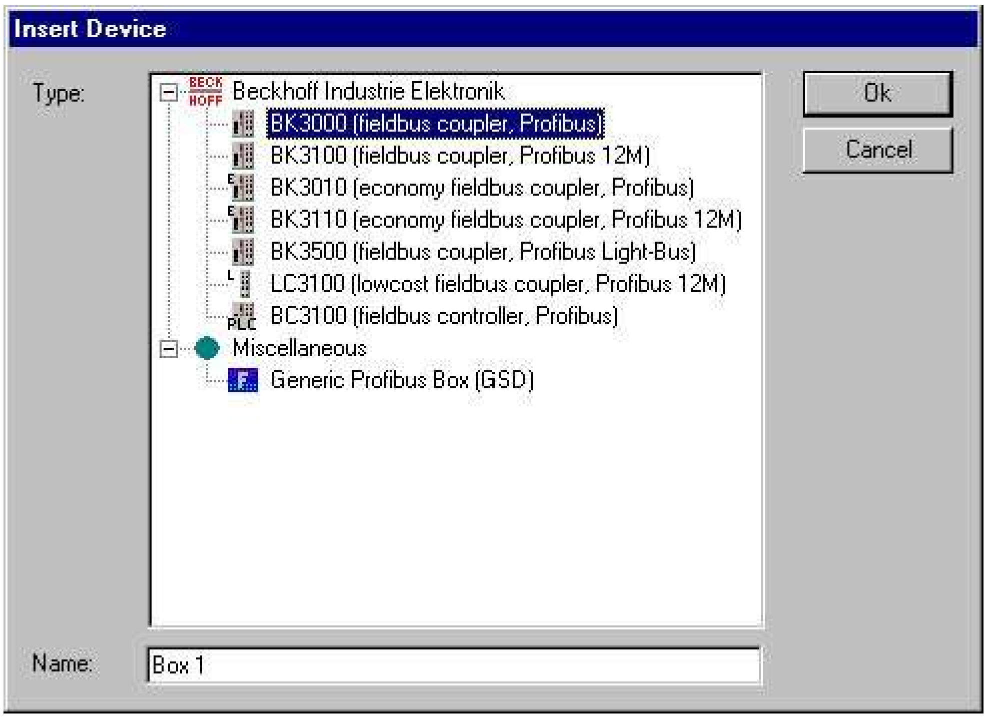
Ready RTime 0%

On the right hand side, a dialog box now opens in which you can specify the configuration of the interface card. One important setting under the "CIF30 DPM" slider, for example, is the I/O address of the card.

4.5.1 ADDING A BUS COUPLER

Open the context menu of the CIF30 DPM card (device 1) and select the "Append box…" command.

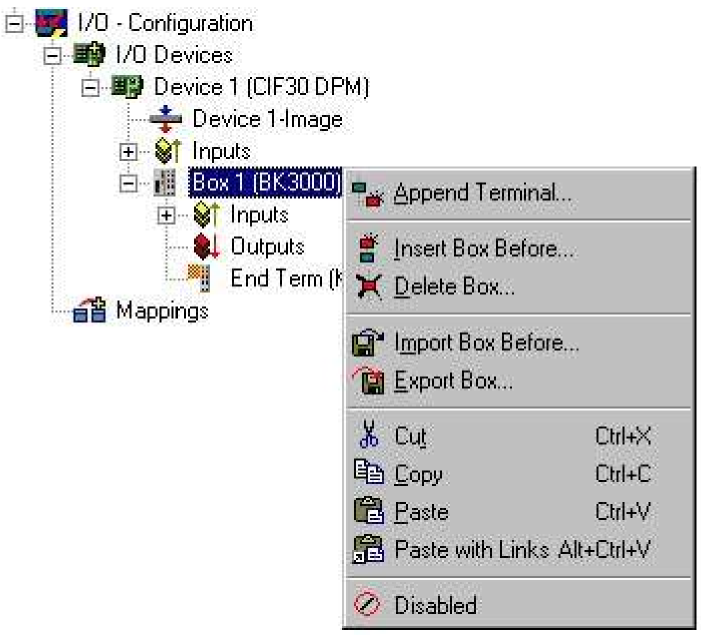


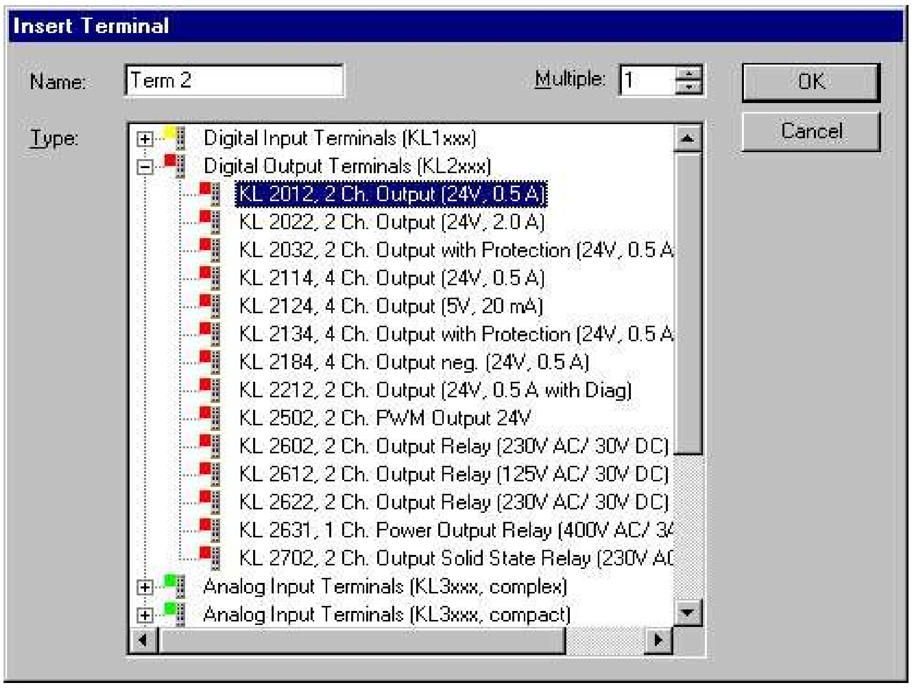

Insert Device
Type:
Beckhoff Industrie Elektronik
BK3000 (fieldbus coupler, Profibus)
BK3100 (fieldbus coupler, Profibus 12M)
BK3010 (economy fieldbus coupler, Profibus)
BK3110 (economy fieldbus coupler, Profibus 12M)
BK3500 (fieldbus coupler, Profibus Light-Bus)
LC3100 (lowcost fieldbus coupler, Profibus 12M)
BC3100 (fieldbus controller, Profibus)
Miscellaneous
Generic Profibus Box (GSD)
Name: Box 1
Ok
Cancel

Select the field bus module type, in this case "BK3000". You can choose the field bus module name freely. It is also possible using a module from a different vendor just by choosing "Generic Profibus Box". In this case they are necessary the GSD files from those modules.

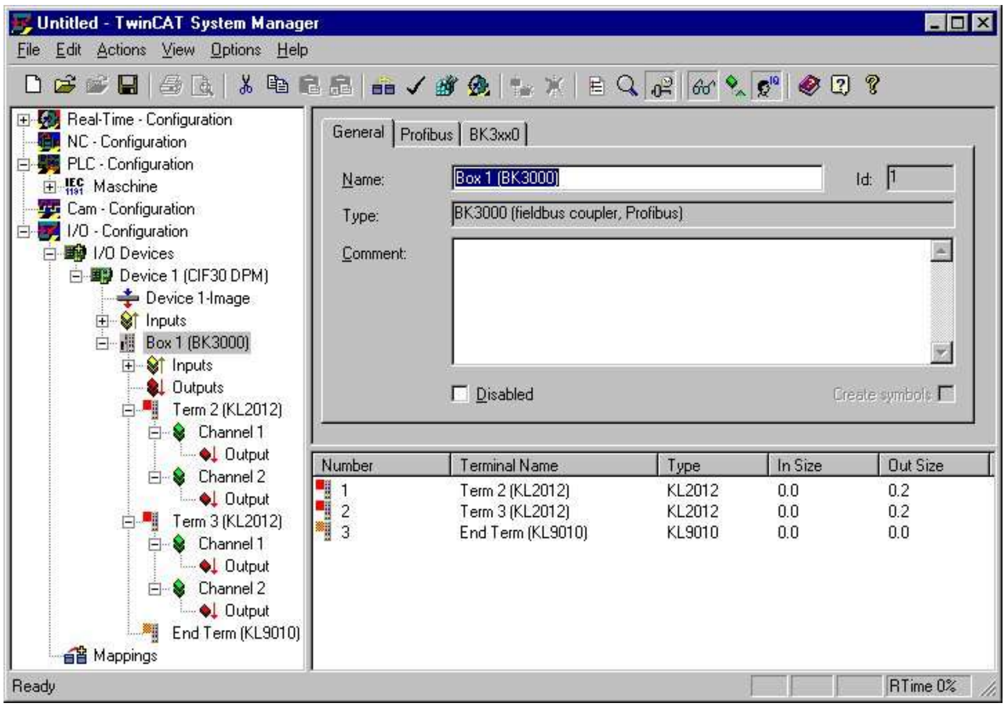
4.5.2 ADDING A BUS TERMINAL

Open the context menu (press the right mouse button) on the BK3000 (box1) and select the "Append Terminal…" command.




Insert Terminal
Name: Term 2
Multiple: 1
OK
Cancel
Type:
Digital Input Terminals (KL1xxx)
Digital Output Terminals (KL2xxx)
KL 2012, 2 Ch. Output (24V, 0.5 A)
KL 2022, 2 Ch. Output (24V, 2.0 A)
KL 2032, 2 Ch. Output with Protection (24V, 0.5 A
KL 2114, 4 Ch. Output (24V, 0.5 A)
KL 2124, 4 Ch. Output (5V, 20 mA)
KL 2134, 4 Ch. Output with Protection (24V, 0.5 A
KL 2184, 4 Ch. Output neg. (24V, 0.5 A)
KL 2212, 2 Ch. Output (24V, 0.5 A with Diag)
KL 2502, 2 Ch. PWM Output 24V
KL 2602, 2 Ch. Output Relay (230V AC/ 30V DC)
KL 2612, 2 Ch. Output Relay (125V AC/ 30V DC)
KL 2622, 2 Ch. Output Relay (230V AC/ 30V DC)
KL 2631, 1 Ch. Power Output Relay (400V AC/ 3
KL 2702, 2 Ch. Output Solid State Relay (230V A
Analog Input Terminals (KL3xxx, complex)
Analog Input Terminals (KL3xxx, compact)

Select the terminal that bears the designation KL2012. Make sure that the terminals are added in the right order. The configuration then has the following breakdown:



Name: Box 1 (BK3000) Id: 1

Type: BK3000 (fieldbus coupler, Profibus)

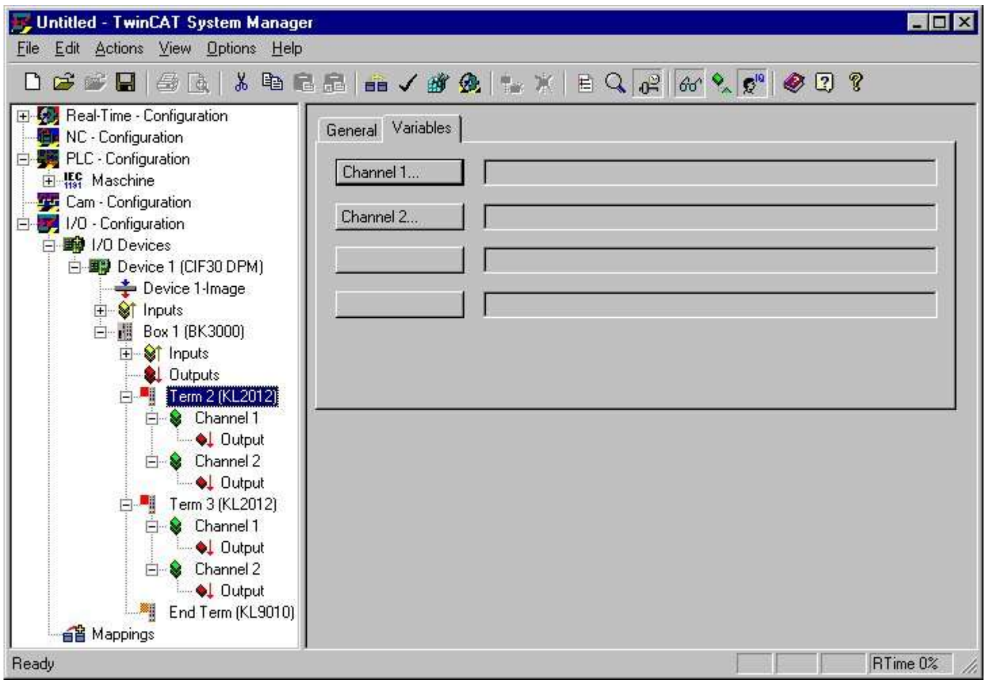
Number	Terminal Name	Type	In Size	Out Size
1	Term 2 (KL2012)	KL2012	0.0	0.2
2	Term 3 (KL2012)	KL2012	0.0	0.2
3	End Term (KL9010)	KL9010	0.0	0.0

It is possible to rename the standard designations (device 1, box 1, terminal 1, etc). To do this, slowly double click the corresponding name and enter the new designation.

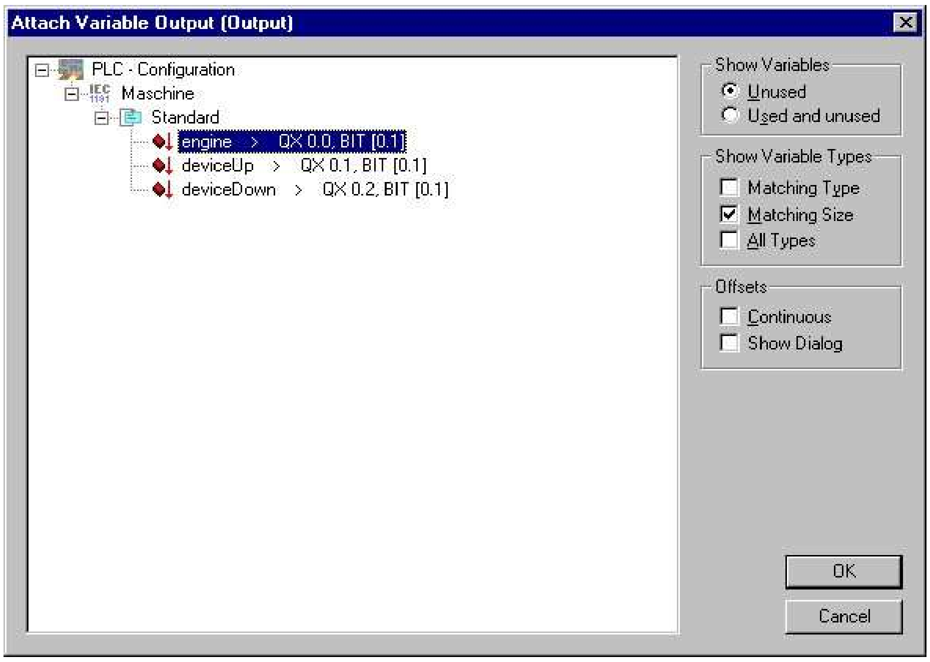
4.6 ASSIGNING VARIABLES TO THE I/O CHANNELS

Up to this point, the complete hardware needed for the example program has been configured. Next, the individual variables from the PLC project must be assigned to the individual I/O channels.

To do this, mark the terminal you wish to configure. In the case of terminal 1 (2 digital outputs), a dialog box containing the "General" and "Variables" sliders opens on the right. Select the "Variables" tab.



You now see a list of the 2 output channels, but they are still all free. To configure channel 1, select the corresponding button ("channel 1…"). The following dialog box opens:



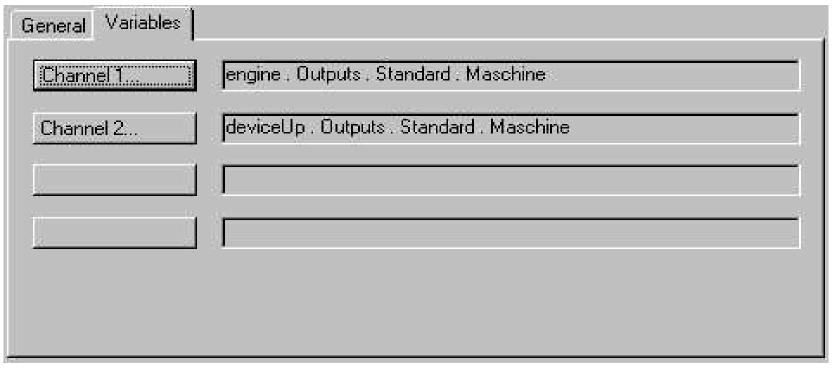
engine > QX 0.0, BIT [0.1]
deviceUp > QX 0.1, BIT [0.1]
deviceDown > QX 0.2, BIT [0.1]

Show Variables: [✓ Unused] Used and unused

Show Variable Types: Matching Type [✓ Matching Size] All Types

Offsets: Continuous, Show Dialog

All output variables are now listed in this dialog box. Select the first variable (engine) and confirm your entry by clicking "OK". Proceed analogously with the other output variables. As there are 3 output variables, 2 should be assigned in "Terminal 2" and one should be assigned in "Terminal 3".



Channel 1... engine . Outputs . Standard . Maschine

Channel 2... deviceUp . Outputs . Standard . Maschine

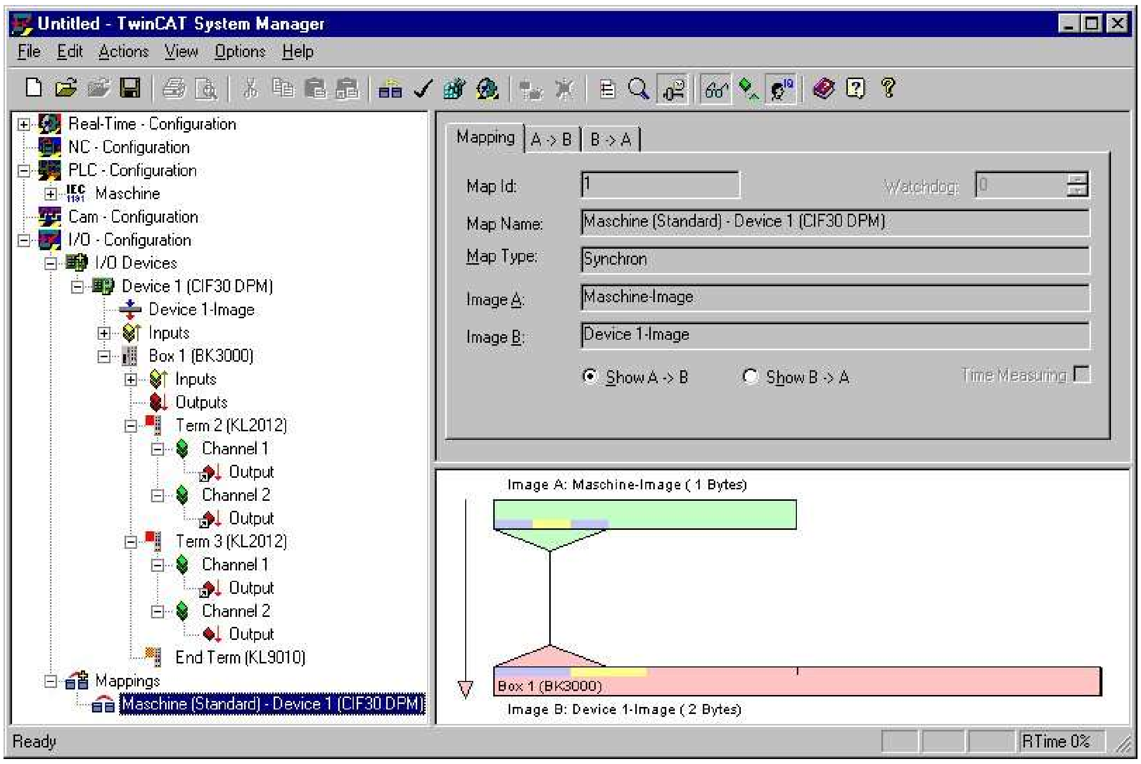
<u>NOTE</u>: The following steps about using TwinCAT's System Manager SHOULD NOT be done during the exercises carried out in the PC class. They should be done only during the laboratory exercise.

4.7 SAVING THE PROJECT

You should save the configuration at this point to make sure you can access it later on. To do this, run the "Save as…" command from the "File" menu. Then switch to the "\TwinCAT\Samples\FirstSteps\" directory and enter the file name. Confirm your entry by selecting "Save".

4.8 MAPPING VARIABLES

You have now configured the complete system for the above example program. Now you must create the allocation. To do this, go to the "Generate Mappings" command in the "Actions" menu. Under the "Mappings" tree entry, you now see "Standard device 1". Click this entry.



Map Id: 1
Watchdog: 0
Map Name: Maschine (Standard) - Device 1 (CIF30 DPM)
Map Type: Synchron
Image A: Maschine-Image
Image B: Device 1-Image
[✓ Show A -> B] Show B -> A
Time Measuring ☐

In the dialog box, you can define whether the data flow from A to B or from B to A is to be displayed. In this case, Image A corresponds to the process image of the PLC variables, i.e. the I/O variables. Image B corresponds to the process image of the I/O devices, in this case of the bus coupler BK3000.

Each variable or bus terminal is colour highlighted in the process image. If you stop on one of these areas with the mouse, a small display box appears in which the precise designation is shown.

4.9 WRITING THE CONFIGURATION TO THE REGISTRY

As the last step, you must save the configuration to Windows NT registry because the information stored there is evaluated when you start TwinCAT. Run the "Save To Registry…" command from the "Actions" menu. If an older configuration is already stored there, a safety prompt will appear, which you must confirm.

Failuring to register may cause problems or old configurations to be processed. After that, TwinCAT automatically will ask for restarting the TwinCAT system so the change will be accepted. The individual PLC variables are now outputs on the modules.

5 APPENDIX

5.1 ACRONYMS AND SYMBOLS

AS-Interface	Actuator-Sensor Interface

CNC	Computer Numerical Control
DDE	Dynamic Data Exchange
FB	Function Block
FBD	Function Blocks Diagram
FUN	Function
GUI	Graphical User Interface
HMI	Human Machine Interface
I/O	Input/Output
IEC	International Electrotechnical Commission
IEC 61131-3	International Standard for Industrial Programming Languages
IL	Instruction List
LD	Ladder Diagram
ms	Millisecond
NC	Numerical Control
PC	Personal Computer
PLC	Programmable Logic Controller
POU	Programme Organisation Unit
PRG	Programme
Profibus	Process Fieldbus
s	Second
SCADA	Supervisory Control and Data Acquisition
SFC	Sequential Function Chart
Soft PLC	PC-Based Control Control Engine in a PC Environment for Industrial Automation
ST	Structured Text

5.2 BIBLIOGRAPHY

- *Lewis, R. W.* Programming Industrial Control Systems using IEC 1131-3
  ISBN: 0-852-96827-2. IEE Control Engineering Series 50
- *Bonfatti, F., & Monari, F.* 1997. IEC 1131-3 Programming Methodology. CJ International.
- IEC 61131-3, Second Edition. 1998. Programmable Controllers - Programming Languages. International Electrotechnical Commission, Technical Committee No. 65, Sub-Committee 65b: Devices, Working Group 7: Programmable Controllers.
- PLCOpen, Standardisation in Industrial Control Programming. http://www.plcopen.org/
- Beckhoff Industrie Elektronik, TwinCAT Soft PLC. http://www.beckhoff.com/