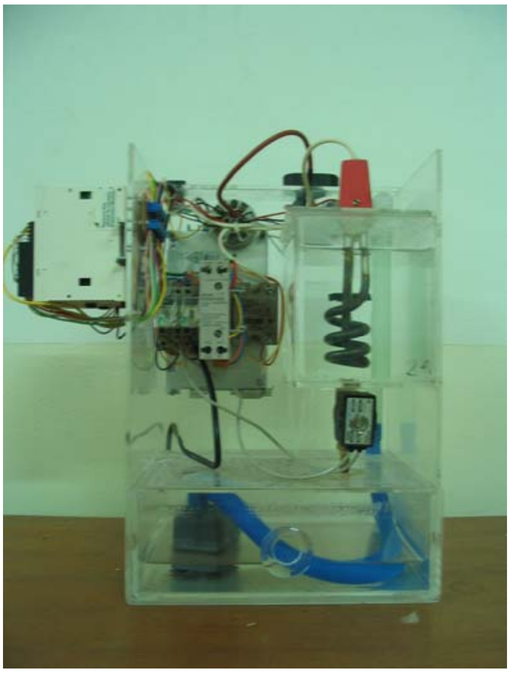
LAMPIRAN A

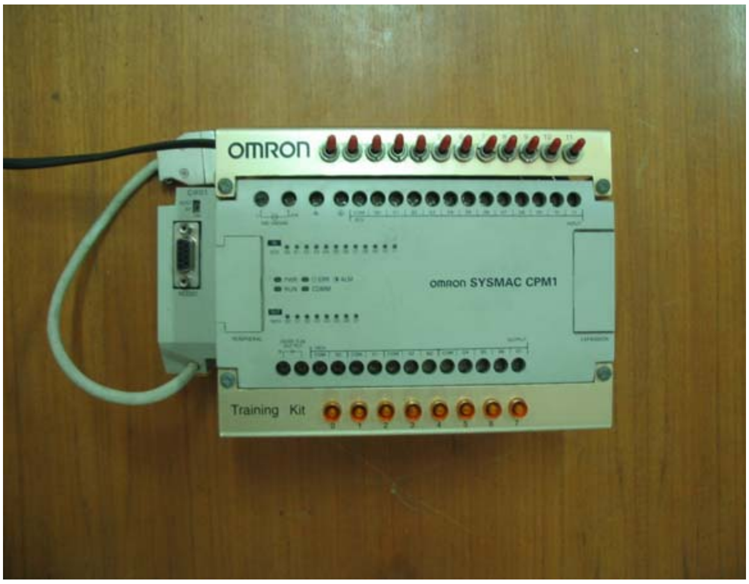
FOTO ALAT

Plant Simulasi Pengendalian Temperatur



Plant Simulasi Pengendalian Pencampuran

Training Kit Omron



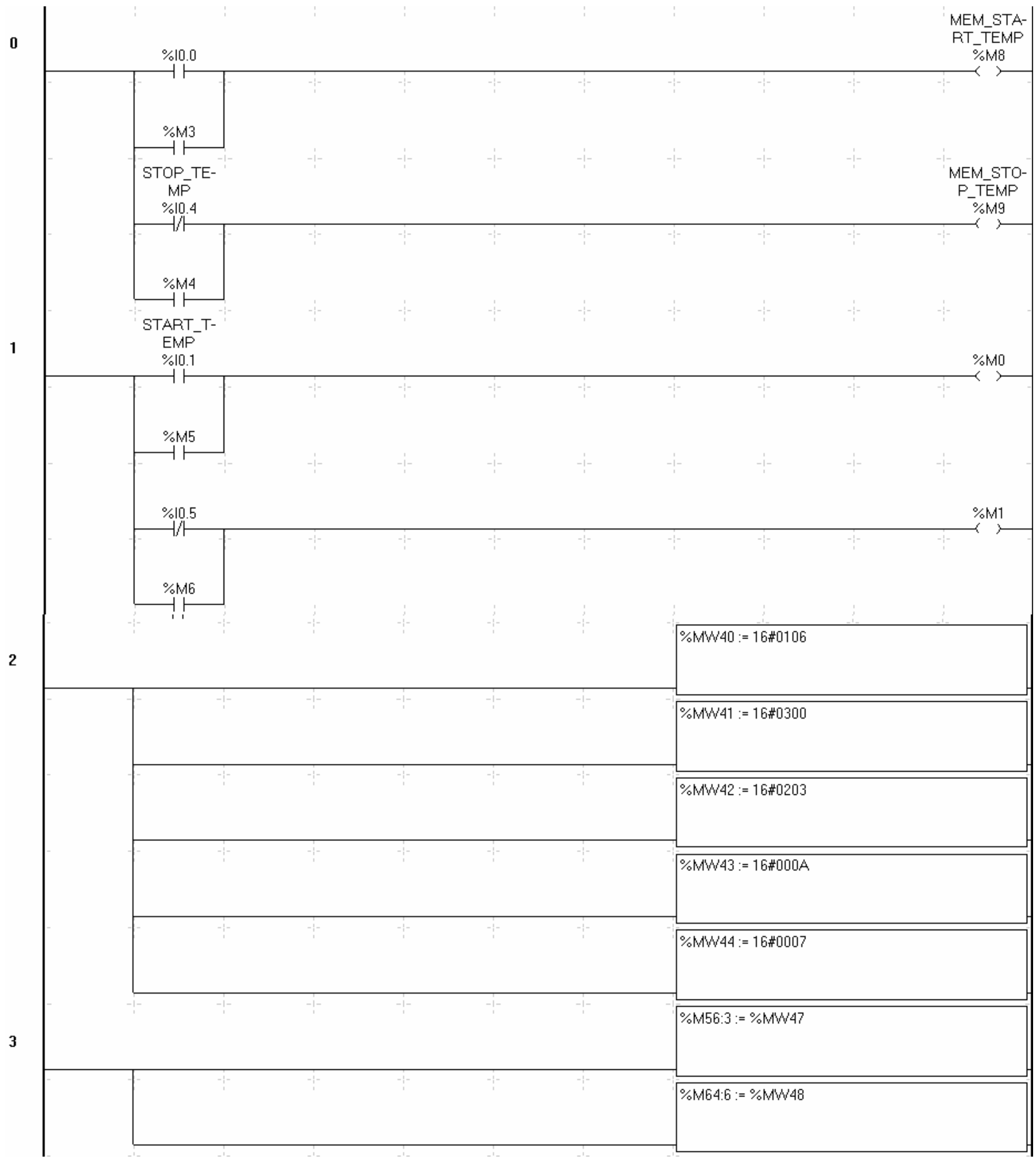
LAMPIRAN B

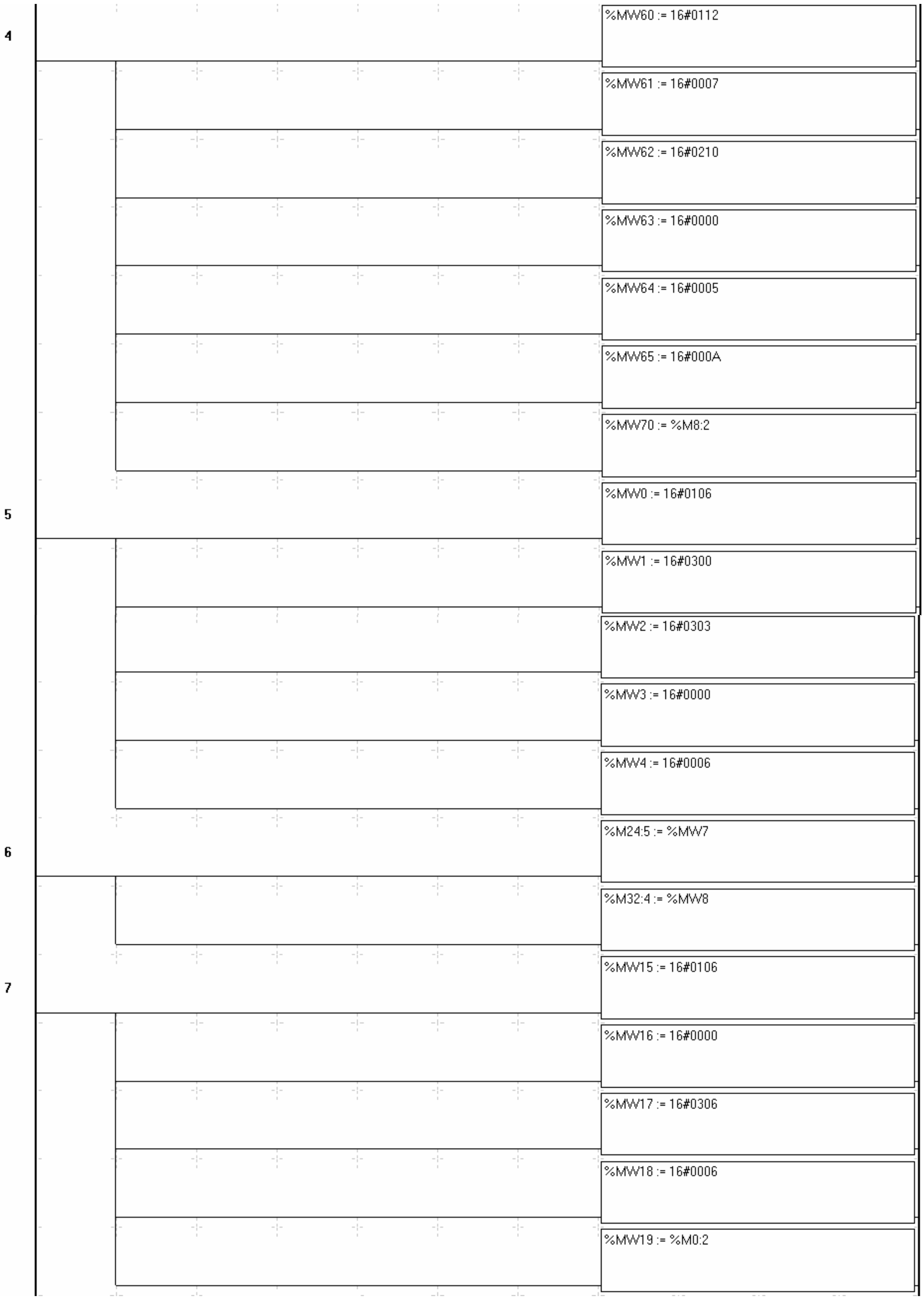
PERANGKAT LUNAK

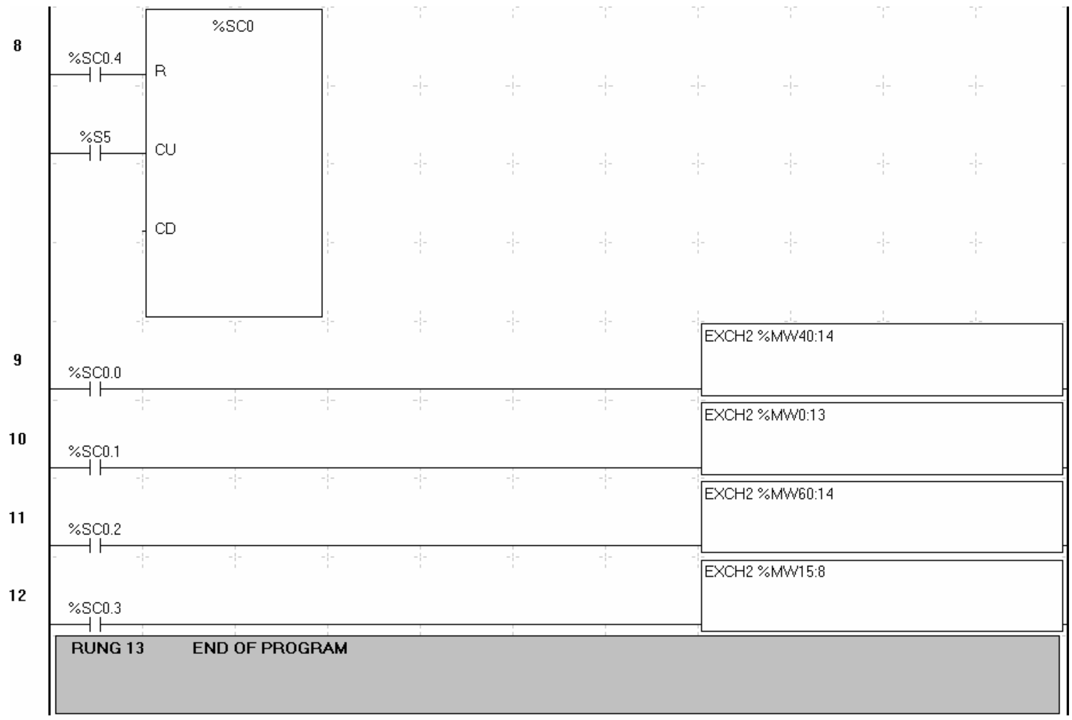
LADDER DIAGRAM

PLC MASTER TWIDO

Ladder Diagram



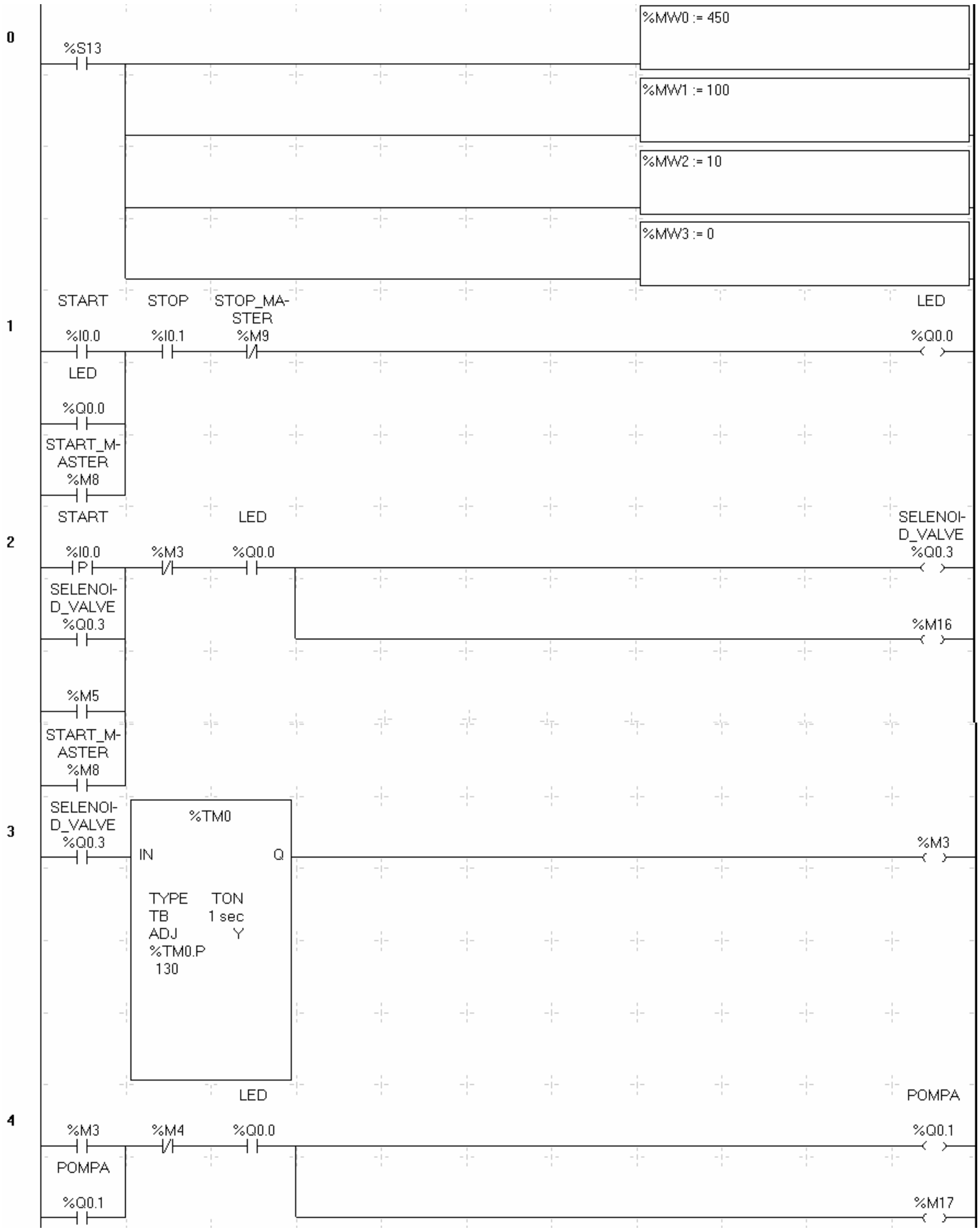

4
%MW60 := 16#0112
%MW61 := 16#0007
%MW62 := 16#0210
%MW63 := 16#0000
%MW64 := 16#0005
%MW65 := 16#000A
%MW70 := %M8:2
5
%MW0 := 16#0106
%MW1 := 16#0300
%MW2 := 16#0303
%MW3 := 16#0000
%MW4 := 16#0006
6
%M24:5 := %MW7
%M32:4 := %MW8
7
%MW15 := 16#0106
%MW16 := 16#0000
%MW17 := 16#0306
%MW18 := 16#0006
%MW19 := %M0:2

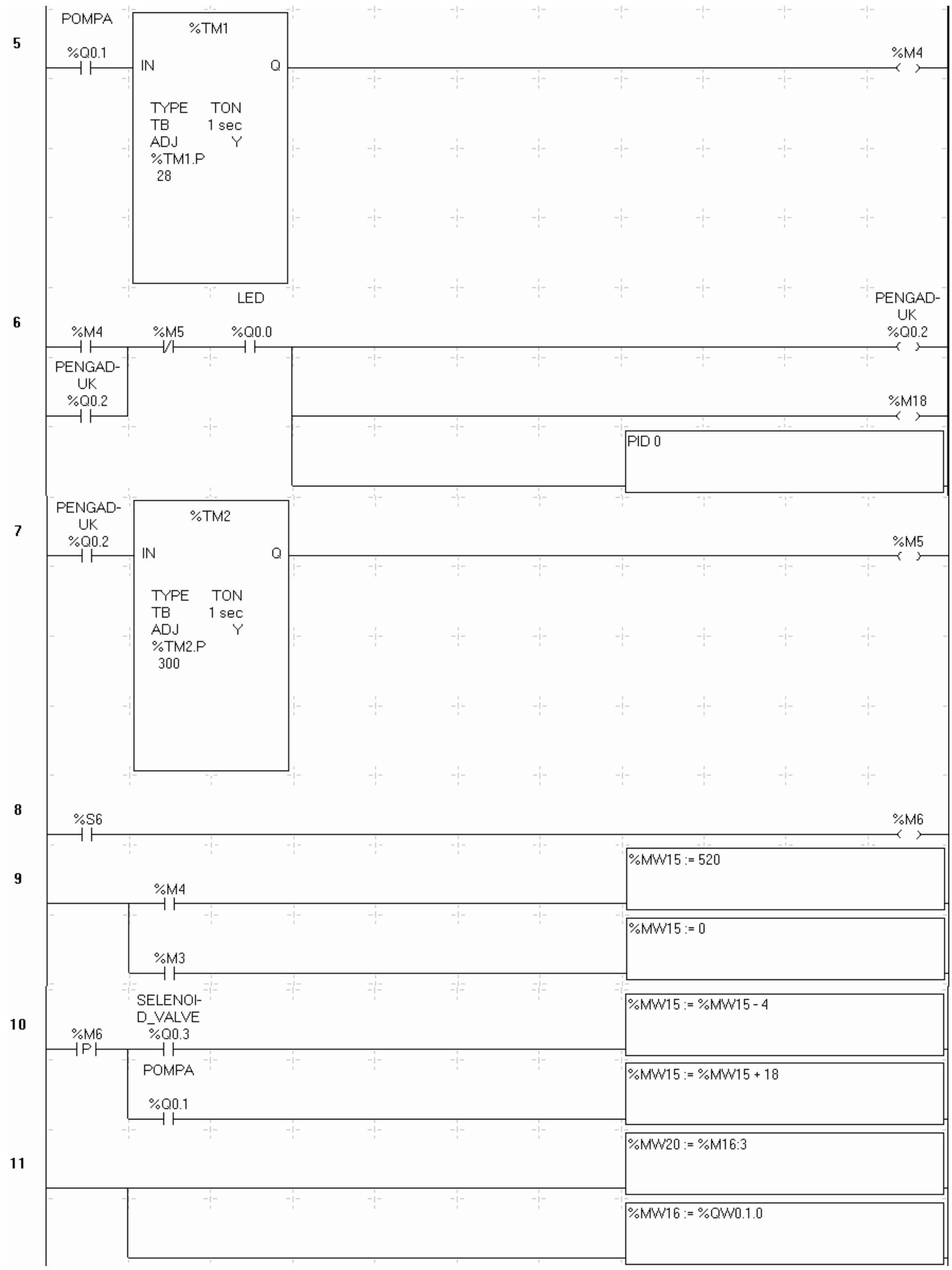

8
%SC0.4
R
%S5
CU
CD
%SC0
9
%SC0.0
EXCH2 %MW40:14
10
%SC0.1
EXCH2 %MW0:13
11
%SC0.2
EXCH2 %MW60:14
12
%SC0.3
EXCH2 %MW15:8
RUNG 13
END OF PROGRAM

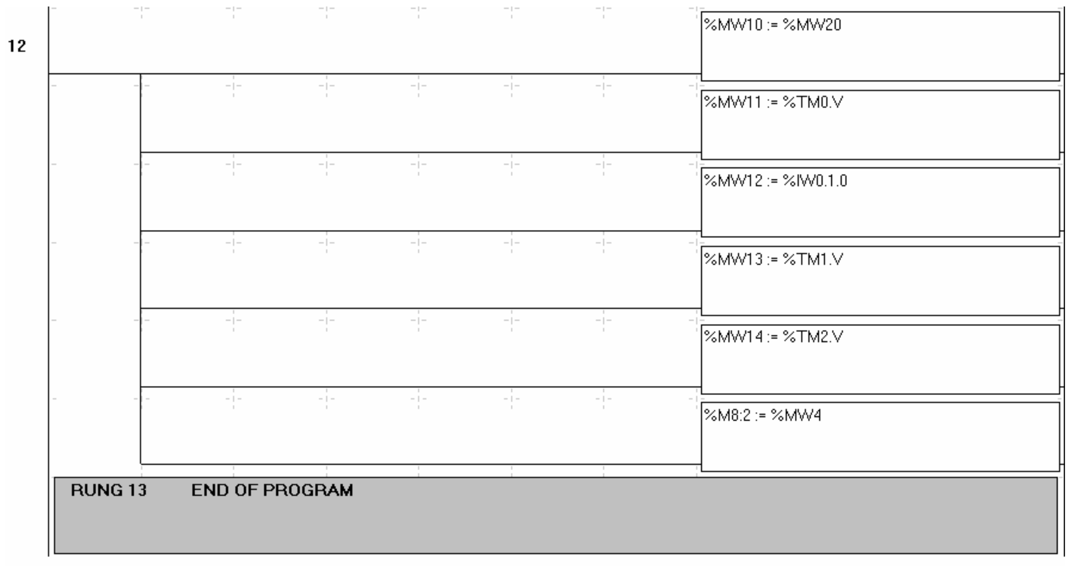
LADDER DIAGRAM

PLANT SIMULASI PENGENDALIAN TEMPERATUR

Ladder Diagram



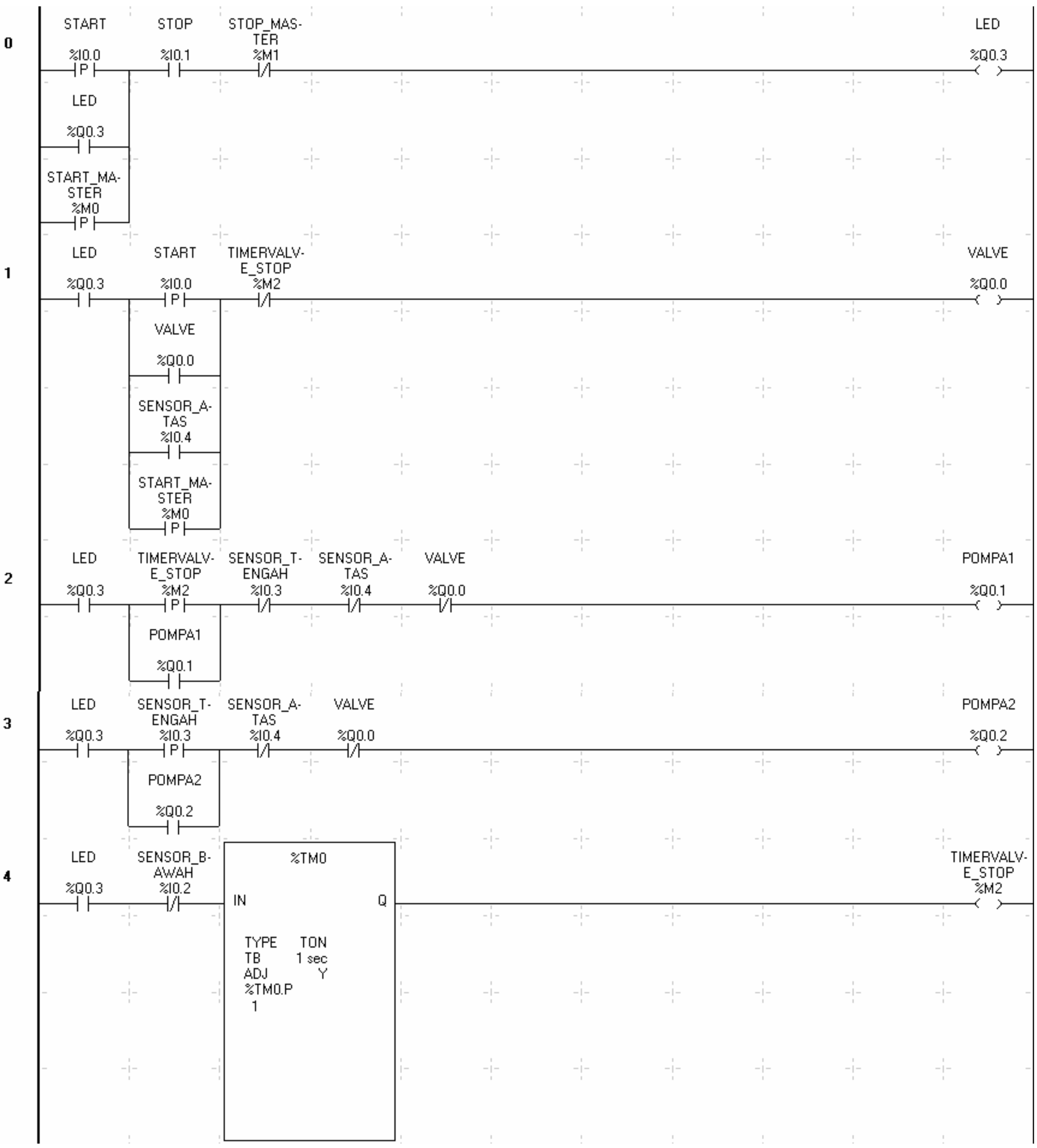

5
POMPA
%Q0.1
%TM1
IN
Q
TYPE TON
TB 1 sec
ADJ Y
%TM1.P
28
%M4
6
LED
%M4
%M5
%Q0.0
PENGAD-UK
%Q0.2
PENGAD-UK
%Q0.2
%M18
PID 0
7
PENGAD-UK
%Q0.2
%TM2
IN
Q
TYPE TON
TB 1 sec
ADJ Y
%TM2.P
300
%M5
8
%S6
%M6
9
%M4
%MW15 := 520
%M3
%MW15 := 0
10
%M6
P
SELENOID_VALVE
%Q0.3
%MW15 := %MW15 - 4
POMPA
%Q0.1
%MW15 := %MW15 + 18
11
%MW20 := %M16:3
%MW16 := %QW0.1.0

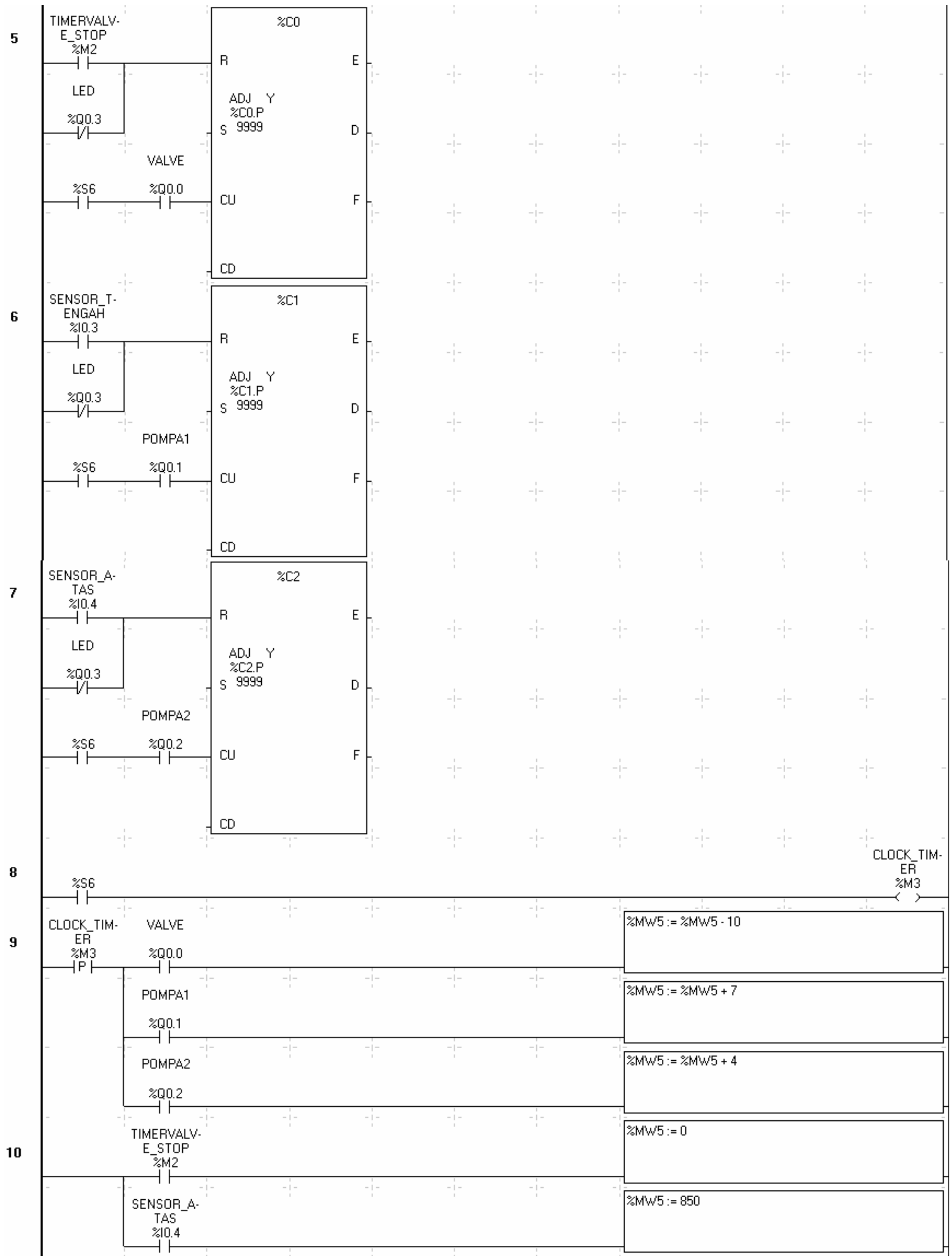

12
%MW10 := %MW20
%MW11 := %TM0.V
%MW12 := %IW0.1.0
%MW13 := %TM1.V
%MW14 := %TM2.V
%M8:2 := %MW4
RUNG 13 END OF PROGRAM

LADDER DIAGRAM

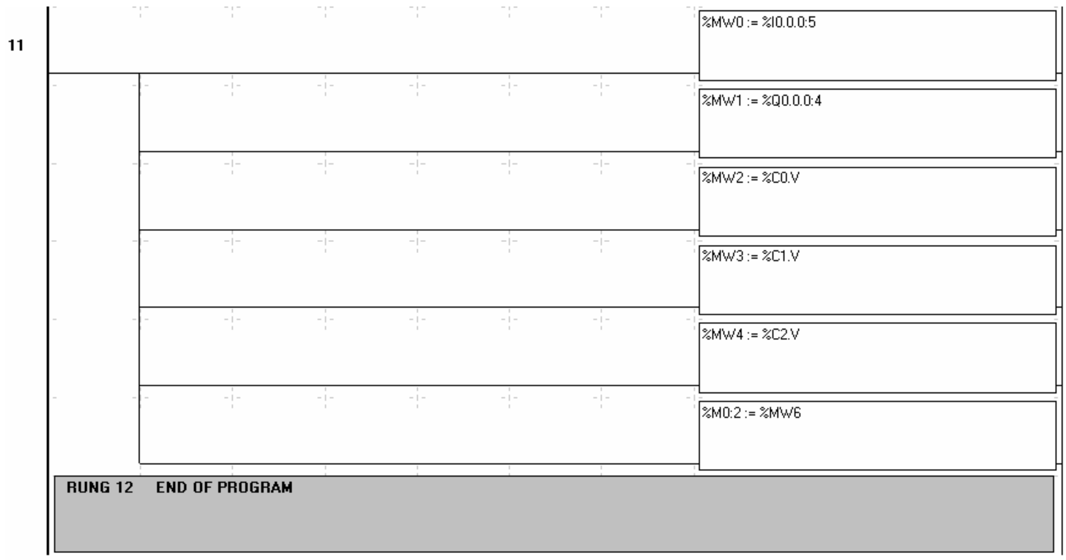
PLANT SIMULASI PENGENDALIAN PENCAMPURAN

Ladder Diagram



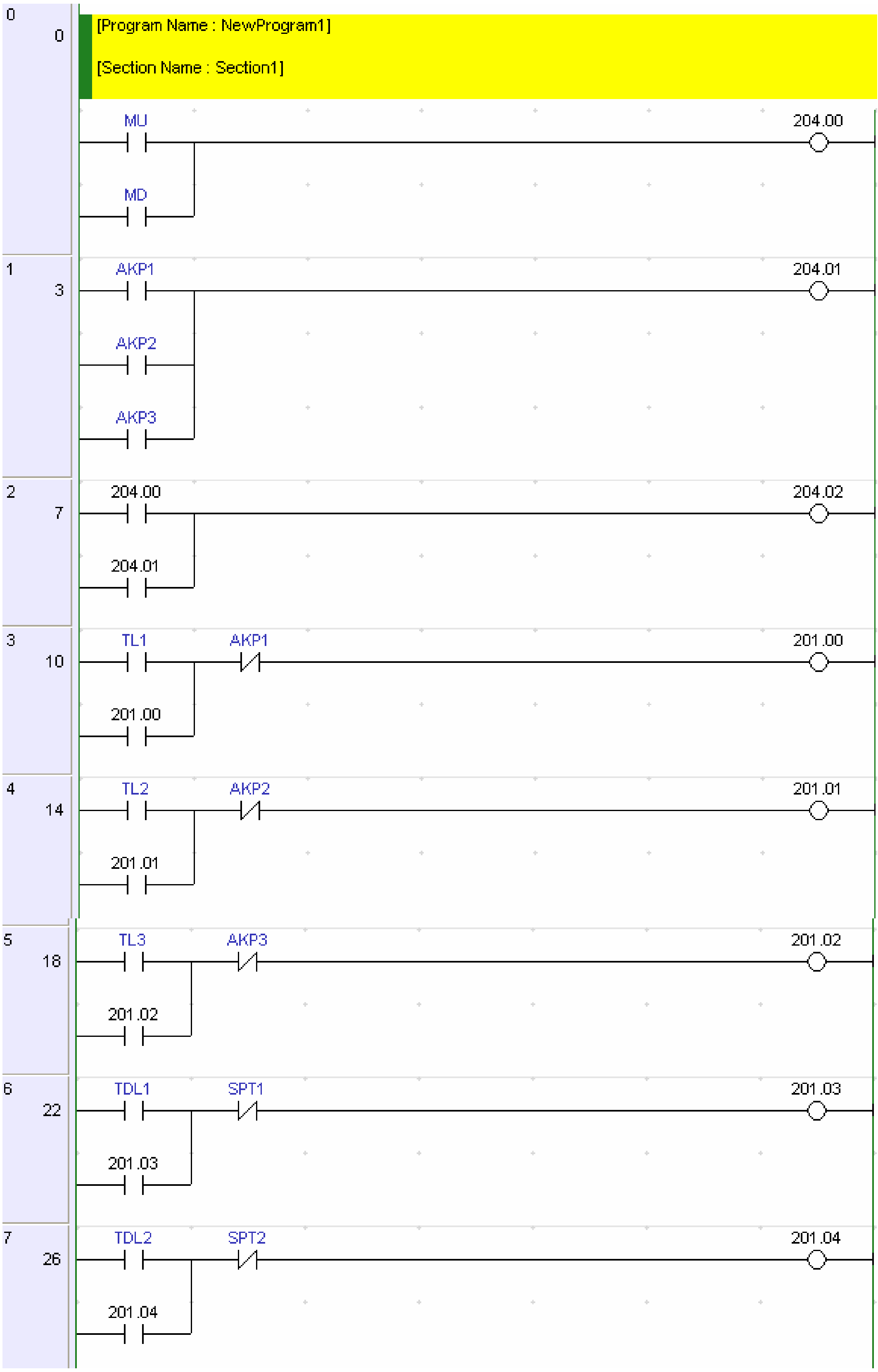


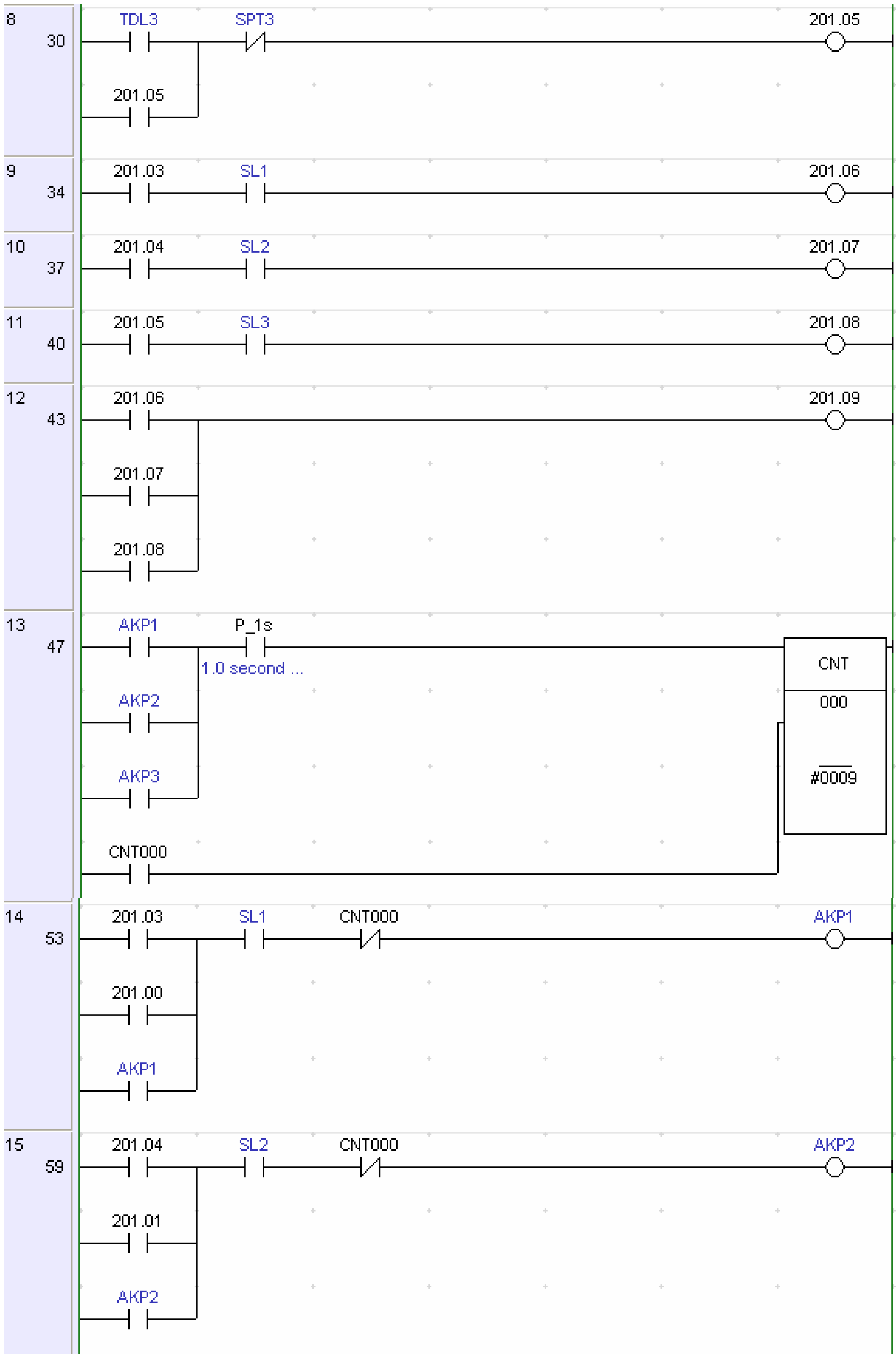
5
TIMERVALVE_STOP %M2
LED %Q0.3
VALVE
%S6 %Q0.0
%C0
R E
ADJ Y
%C0.P
S 9999 D
CU F
CD
6
SENSOR_TENGAH %I0.3
LED %Q0.3
POMPA1
%S6 %Q0.1
%C1
R E
ADJ Y
%C1.P
S 9999 D
CU F
CD
7
SENSOR_ATAS %I0.4
LED %Q0.3
POMPA2
%S6 %Q0.2
%C2
R E
ADJ Y
%C2.P
S 9999 D
CU F
CD
8
%S6
CLOCK_TIMER %M3
9
CLOCK_TIMER %M3 P
VALVE %Q0.0
POMPA1 %Q0.1
POMPA2 %Q0.2
%MW5 := %MW5 - 10
%MW5 := %MW5 + 7
%MW5 := %MW5 + 4
10
TIMERVALVE_STOP %M2
SENSOR_ATAS %I0.4
%MW5 := 0
%MW5 := 850

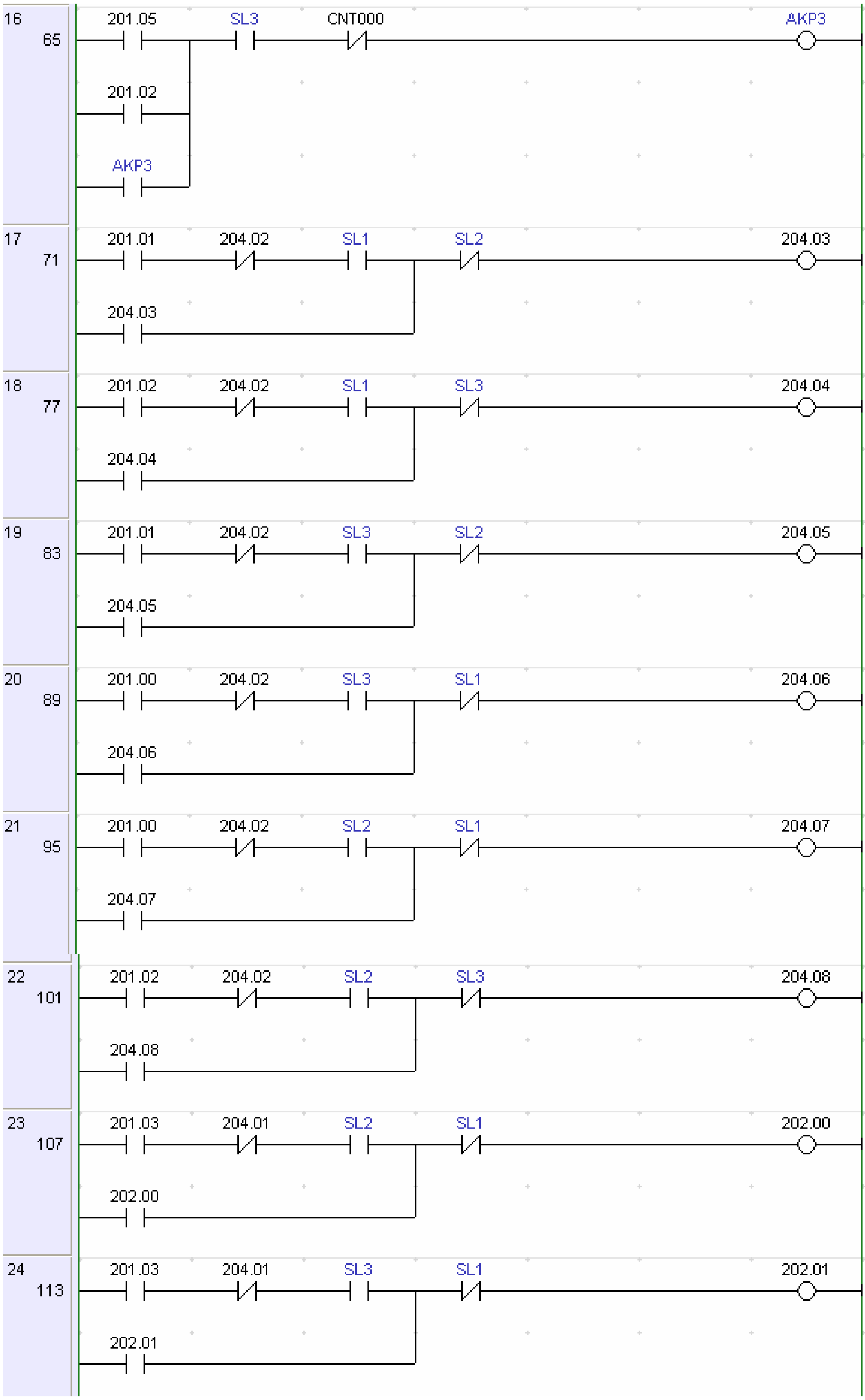

11
%MW0 := %I0.0.0:5
%MW1 := %Q0.0.0:4
%MW2 := %C0.V
%MW3 := %C1.V
%MW4 := %C2.V
%M0:2 := %MW6
RUNG 12 END OF PROGRAM

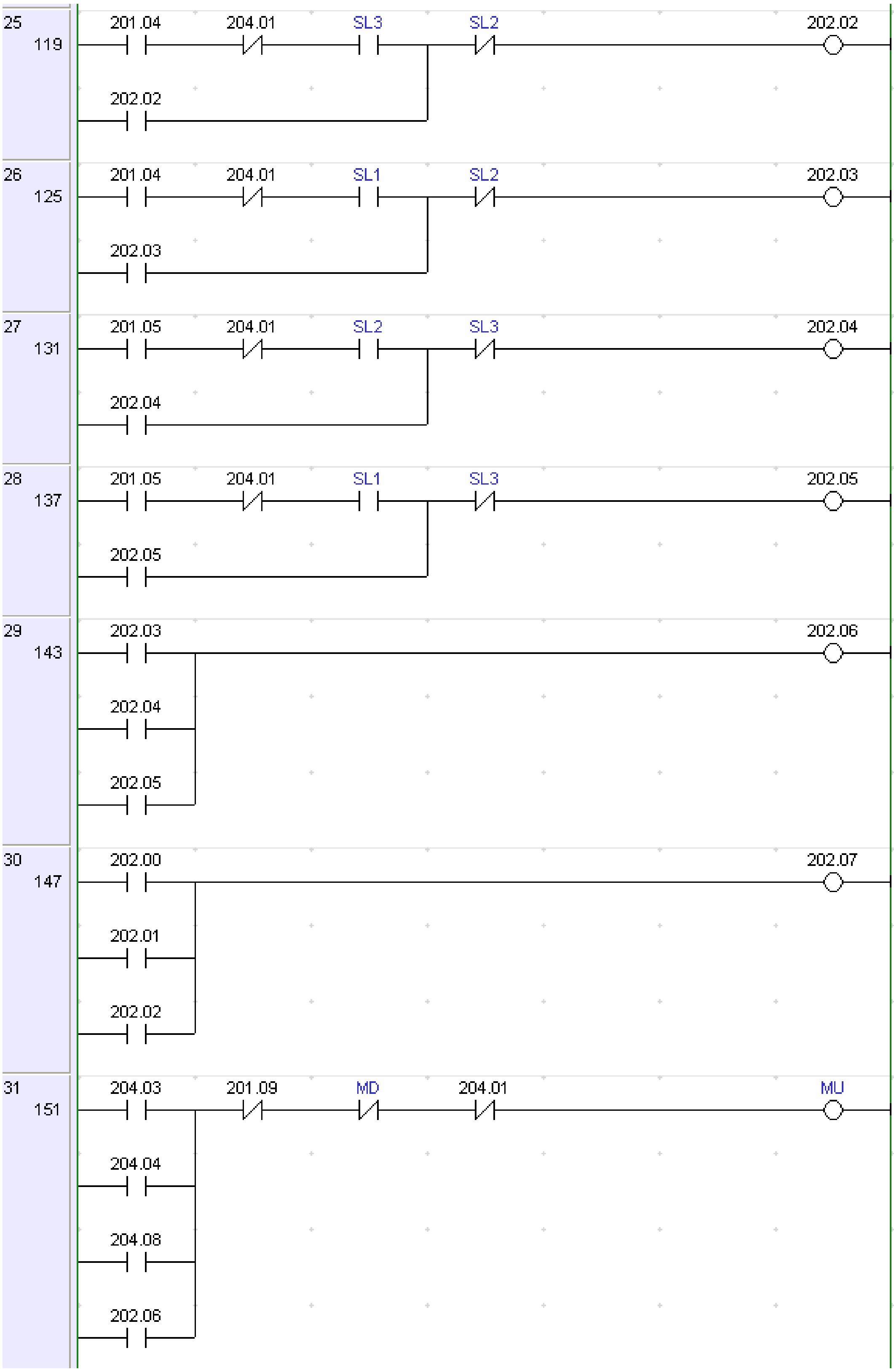
LADDER DIAGRAM

PLANT SIMULASI LIFT

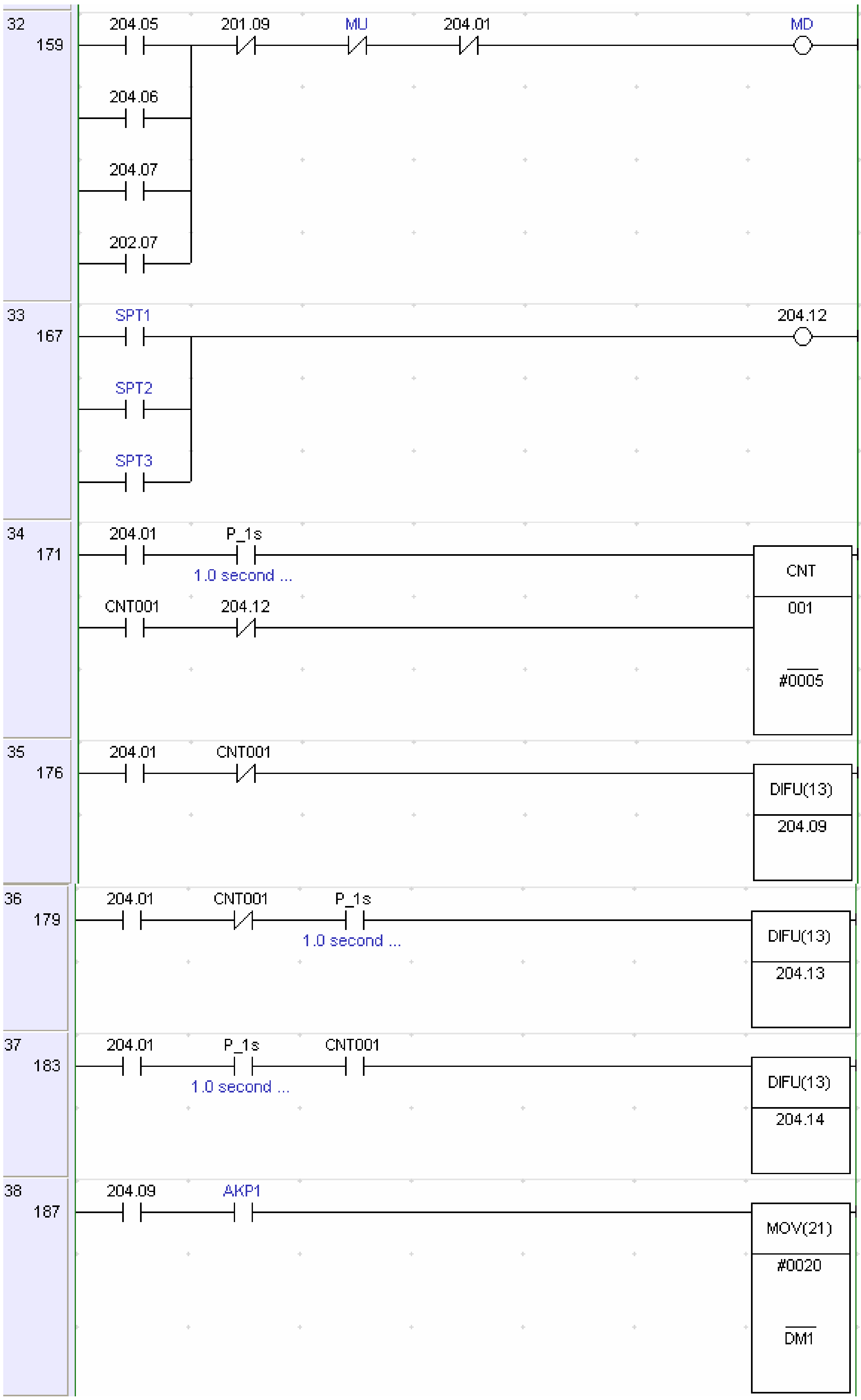

0
0
[Program Name : NewProgram1]
[Section Name : Section1]
MU
MD
204.00
1
3
AKP1
AKP2
AKP3
204.01
2
7
204.00
204.01
204.02
3
10
TL1
AKP1
201.00
201.00
4
14
TL2
AKP2
201.01
201.01
5
18
TL3
AKP3
201.02
201.02
6
22
TDL1
SPT1
201.03
201.03
7
26
TDL2
SPT2
201.04
201.04

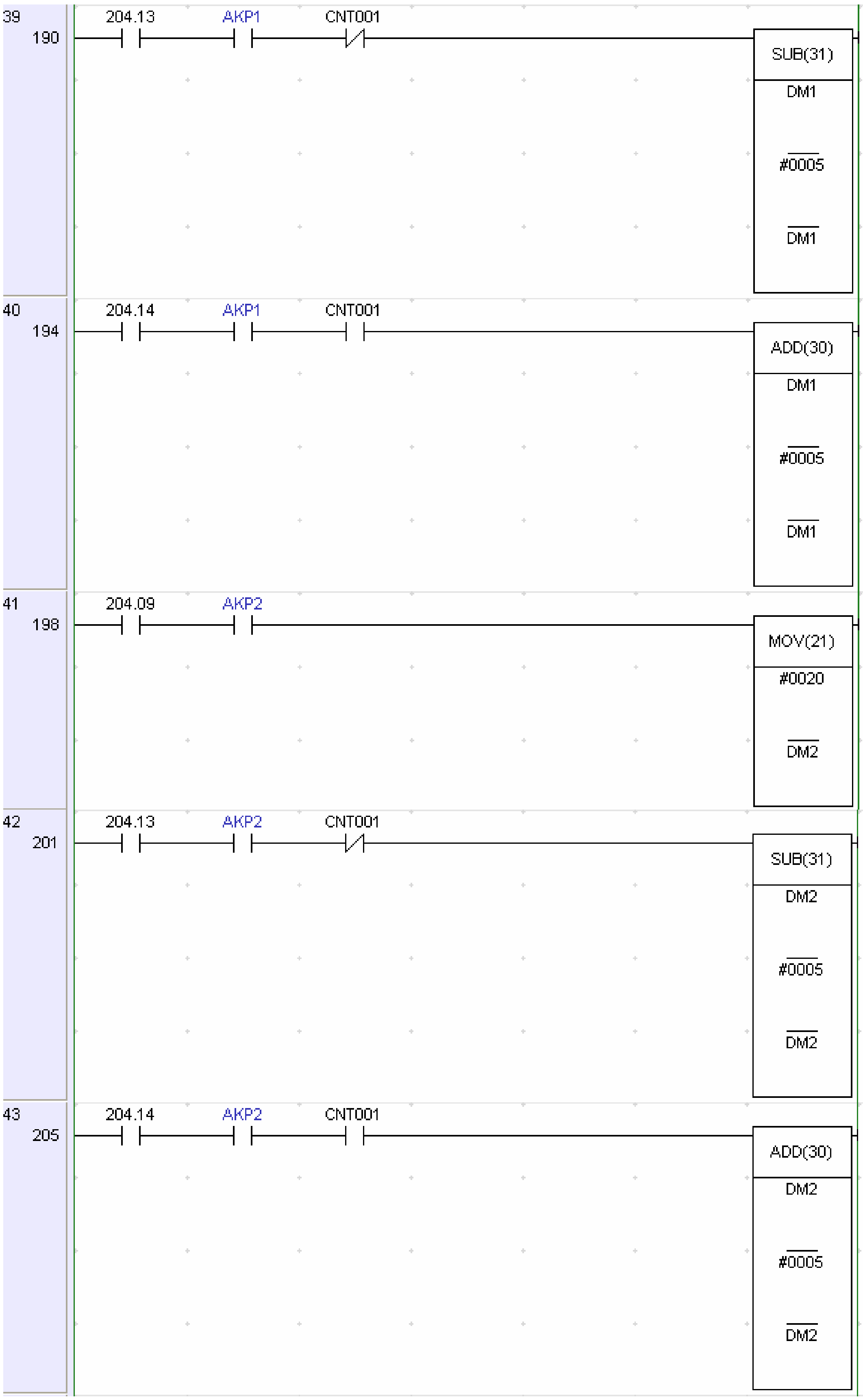

8
30
TDL3
SPT3
201.05
201.05
9
34
201.03
SL1
201.06
10
37
201.04
SL2
201.07
11
40
201.05
SL3
201.08
12
43
201.06
201.09
201.07
201.08
13
47
AKP1
P_1s
1.0 second ...
CNT
000
#0009
AKP2
AKP3
CNT000
14
53
201.03
SL1
CNT000
AKP1
201.00
AKP1
15
59
201.04
SL2
CNT000
AKP2
201.01
AKP2


16
65
201.05
SL3
CNT000
AKP3
201.02
AKP3
17
71
201.01
204.02
SL1
SL2
204.03
204.03
18
77
201.02
204.02
SL1
SL3
204.04
204.04
19
83
201.01
204.02
SL3
SL2
204.05
204.05
20
89
201.00
204.02
SL3
SL1
204.06
204.06
21
95
201.00
204.02
SL2
SL1
204.07
204.07
22
101
201.02
204.02
SL2
SL3
204.08
204.08
23
107
201.03
204.01
SL2
SL1
202.00
202.00
24
113
201.03
204.01
SL3
SL1
202.01
202.01

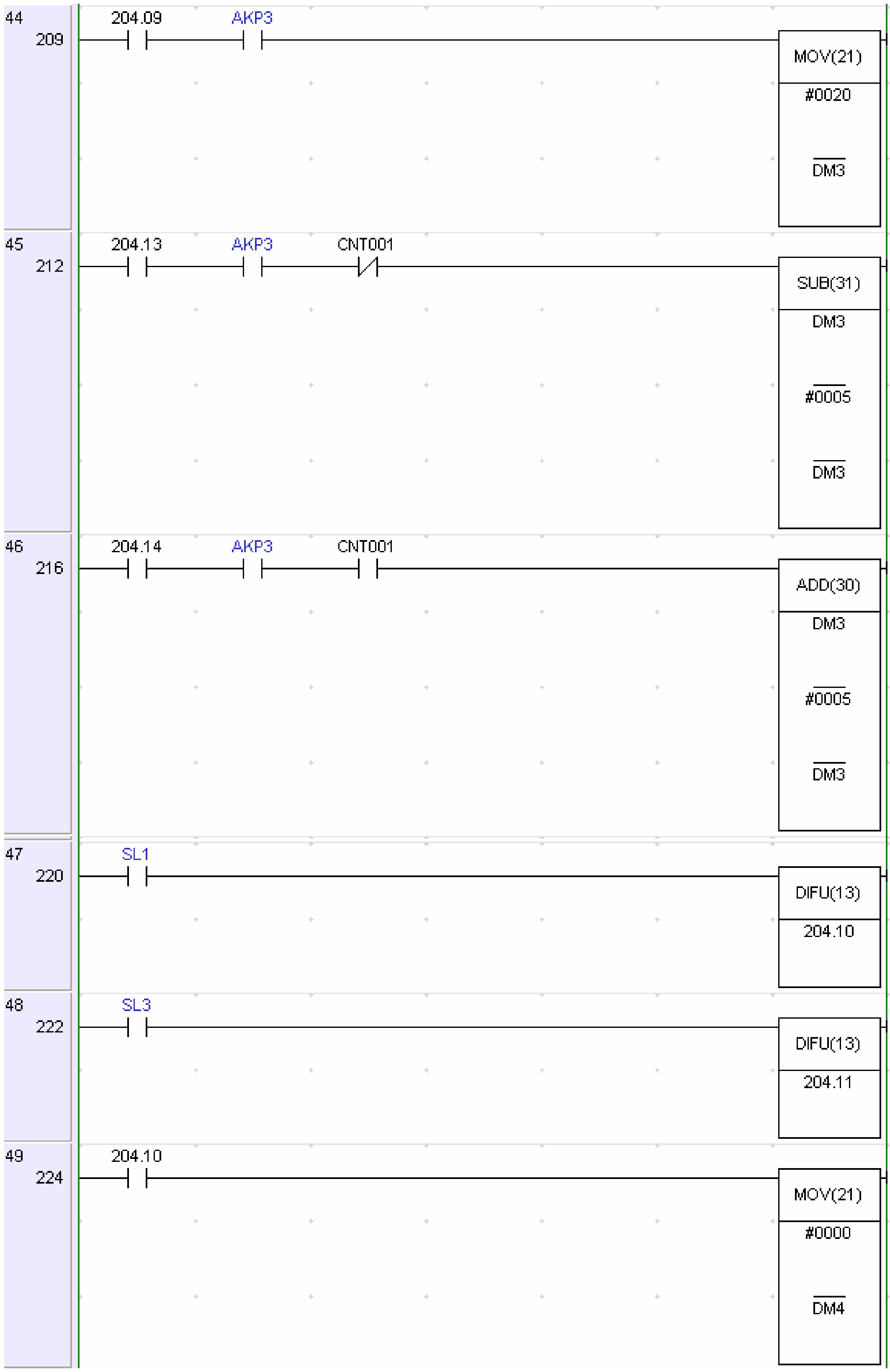


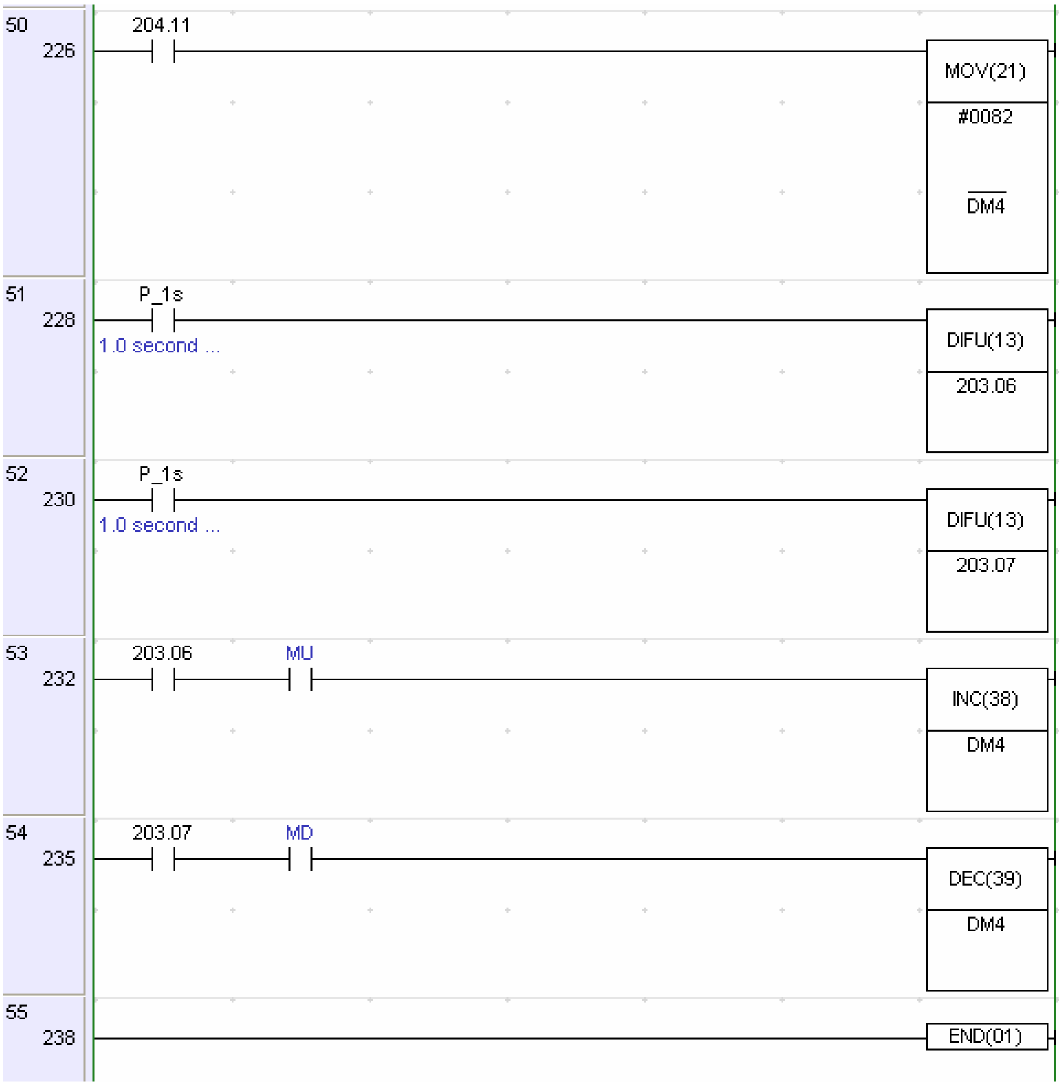
25
119
201.04
204.01
SL3
SL2
202.02
202.02
26
125
201.04
204.01
SL1
SL2
202.03
202.03
27
131
201.05
204.01
SL2
SL3
202.04
202.04
28
137
201.05
204.01
SL1
SL3
202.05
202.05
29
143
202.03
202.06
202.04
202.05
30
147
202.00
202.07
202.01
202.02
31
151
204.03
201.09
MD
204.01
MU
204.04
204.08
202.06


32
159
204.05
201.09
MU
204.01
MD
204.06
204.07
202.07
33
167
SPT1
204.12
SPT2
SPT3
34
171
204.01
P_1s
1.0 second ...
CNT
001
CNT001
204.12
#0005
35
176
204.01
CNT001
DIFU(13)
204.09
36
179
204.01
CNT001
P_1s
1.0 second ...
DIFU(13)
204.13
37
183
204.01
P_1s
CNT001
1.0 second ...
DIFU(13)
204.14
38
187
204.09
AKP1
MOV(21)
#0020
DM1



39
190
204.13
AKP1
CNT001
SUB(31)
DM1
#0005
DM1
40
194
204.14
AKP1
CNT001
ADD(30)
DM1
#0005
DM1
41
198
204.09
AKP2
MOV(21)
#0020
DM2
42
201
204.13
AKP2
CNT001
SUB(31)
DM2
#0005
DM2
43
205
204.14
AKP2
CNT001
ADD(30)
DM2
#0005
DM2


44
209
204.09
AKP3
MOV(21)
#0020
DM3
45
212
204.13
AKP3
CNT001
SUB(31)
DM3
#0005
DM3
46
216
204.14
AKP3
CNT001
ADD(30)
DM3
#0005
DM3
47
220
SL1
DIFU(13)
204.10
48
222
SL3
DIFU(13)
204.11
49
224
204.10
MOV(21)
#0000
DM4


50
226
204.11
MOV(21)
#0082
DM4
51
228
P_1s
1.0 second ...
DIFU(13)
203.06
52
230
P_1s
1.0 second ...
DIFU(13)
203.07
53
232
203.06
MU
INC(38)
DM4
54
235
203.07
MD
DEC(39)
DM4
55
238
END(01)

LAMPIRAN C
DATASHEET

Presentation of the different types of communication

At a Glance

Twido provides one or two serial communications ports used for communications to remote I/O controllers, peer controllers, or general devices. Either port, if available, can be used for any of the services, with the exception of communicating with Twido Soft, which can only be performed using the first port. Three different base protocols are supported on each Twido controller: Remote Link, ASCII, or Modbus (modbus master or modbus slave).
Moreover, the TWDLCAE40DRF compact controller provides one RJ-45 Ethernet communications port. It supports the Modbus TCP/IP client/server protocol for peer-to-peer communications between controllers over the Ethernet network.

Remote Link

The remote link is a high-speed master/slave bus designed to communicate a small amount of data between the master controller and up to seven remote (slave) controllers. Application or I/O data is transferred, depending on the configuration of the remote controllers. A mixture of remote controller types is possible, where some can be remote I/O and some can be peers.

ASCII

The ASCII protocol is a simple half-duplex character mode protocol used to transmit and/or receive a character string to/from a simple device (printer or terminal). This protocol is supported only via the "EXCH" instruction.

Modbus

The Modbus protocol is a master/slave protocol that allows for one, and only one, master to request responses from slaves, or to act based on the request. The master can address individual slaves, or can initiate a broadcast message to all slaves. Slaves return a message (response) to queries that are addressed to them individually. Responses are not returned to broadcast queries from the master.
Modbus master - The modbus master mode allows the Twido controller to send a modbus query to a slave and await its reply. The modbus master mode is supported only via the "EXCH" instruction. Both Modbus ASCII and RTU are supported in modbus master mode.
Modbus Slave - The modbus slave mode allows the Twido controller to respond to modbus queries from a modbus master, and is the default communications mode if no other type of communication is configured. The Twido controller supports the standard modbus data and control functions and service extensions for object access. Both Modbus ASCII and RTU are supported in modbus slave mode.

Note: 32 devices (without repeaters) can be installed on an RS-485 network (1 master and up to 31 slaves), the addresses of which can be between 1 and 247.

Modbus TCP/IP

Note: Modbus TCP/IP is solely supported by TWDLCAE40DRF series of compact controllers with built-in Ethernet network interface.

The following information describes the Modbus Application Protocol (MBAP).
The Modbus Application Protocol (MBAP) is a layer-7 protocol providing peer-to-peer communication between programmable logic controllers (PLCs) and other nodes on a LAN.
The current Twido controller TWDLCAE40DRF implementation transports Modbus Application Protocol over TCP/IP on the Ethernet network. Modbus protocol transactions are typical request-response message pairs. A PLC can be both client and server depending on whether it is querying or answering messages.

Modbus Communications

Introduction

The Modbus protocol is a master-slave protocol that allows for one, and only one, master to request responses from slaves, or to act based on the request. The master can address individual slaves, or can initiate a broadcast message to all slaves. Slaves return a message (response) to queries that are addressed to them individually. Responses are not returned to broadcast queries from the master.

	CAUTION UNEXPECTED EQUIPMENT OPERATION Be sure that there is only one Modbus master controller on the bus and that each Modbus slave has a unique address. Failure to observe this precaution may lead to corrupted data or unexpected and ambiguous results. Be sure that all Modbus slaves have unique addresses. No two slaves should have the same address. Failure to observe this precaution may lead to corrupted data or unexpected and ambiguous results. Failure to follow this instruction can result in injury or equipment damage.

Hardware Configuration

A Modbus link can be established on either the EIA RS-232 or EIA RS-485 port and can run on as many as two communications ports at a time. Each of these ports can be assigned its own Modbus address, using system bit %S101 and system words %SW101 and %SW102. .

The table below lists the devices that can be used:

Remote	Port	Specifications
TWDLC•A10/16/24DRF, TWDLCA•40DRF, TWDLMDA20/40DTK, TWDLMDA20DRT	1	Base controller supporting a 3-wire EIA RS-485 port with a miniDIN connector.
TWDNOZ232D	2	Communication module equipped with a 3-wire EIA RS-232 port with a miniDIN connector. Note: This module is only available for the Modular controllers. When the module is attached, the controller cannot have an Operator Display expansion module.
TWDNOZ485D	2	Communication module equipped with a 3-wire EIA RS-485 port with a miniDIN connector. Note: This module is only available for the Modular controllers. When the module is attached, the controller cannot have an Operator Display expansion module.
TWDNOZ485T	2	Communication module equipped with a 3-wire EIA RS-485 port with a terminal. Note: This module is only available for the Modular controllers. When the module is attached, the controller cannot have an Operator Display expansion module.
TWDNAC232D	2	Communication adapter equipped with a 3-wire EIA RS-232 port with a miniDIN connector. Note: This adapter is only available for the Compact 16, 24 and 40 I/O controllers and the Operator Display expansion module.
TWDNAC485D	2	Communication adapter equipped with a 3-wire EIA RS-485 port with a miniDIN connector. Note: This adapter is only available for the Compact 16, 24 and 40 I/O controllers and the Operator Display expansion module.
TWDNAC485T	2	Communication adapter equipped with a 3-wire EIA RS-485 port with a terminal connector. Note: This adapter is only available for the Compact 16, 24 and 40 I/O controllers and the Operator Display expansion module.
TWDXCPODM	2	Operator Display expansion module equipped with a 3-wire EIA RS-232 port with a miniDIN connector, a 3-wire EIA RS-485 port with a miniDIN connector and a 3-wire EIA RS-485 port with a terminal.

		Note: This module is only available for the Modular controllers. When the module is attached, the controller cannot have a Communication expansion module.

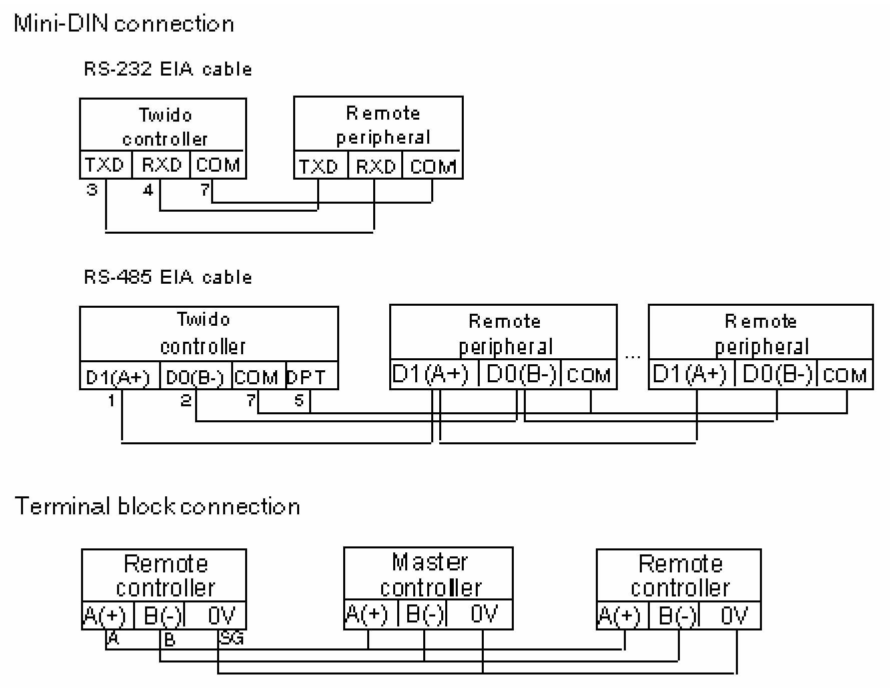
Note: The presence and configuration (RS232 or RS485) of Port 2 is checked at power-up or at reset by the firmware executive program.

Nominal Cabling

Nominal cable connections are illustrated below for both the EIA RS-232 and the EIA RS-485 types.

Note: If port 1 is used on the Twido controller, the DPT signal on pin 5 must be tied to the circuit common (COM) on pin 7. This signifies to the Twido controller that the communications through port 1 is Modbus and is not the protocol used to communicate with the TwidoSoft software.

The cable connections made to each remote device are shown below.



EIA RS-485 Line Polarization on TWDLCA•40DRFControllers

There is no internal pre-polarization in TWDLCA•40DRF controllers. Therefore, external line polarization is required when connecting the TWDLCA•40DRF Modbus master controller to the EIA-485 Modbus network.
(When there is no data activity on an EIA-485 balanced pair, the lines are not driven and, therefore, susceptible to external noise or interference. To ensure that its receiver stays in a constant state, when no data signal is present, the Modbus

master device needs to bias the network via external line polarization.)

Note: EIA RS-485 external line polarization must be implemented on the Modbus Master controller only; you must not implement it on any slave device.

The external line polarization assembly on the TWDLCA•40DRF mini-DIN RS-485 EIA line shall consist in:

One pull-up resistor to a 5V voltage on D1(A+) circuit,

One pull-down resistor to the common circuit on D0(B-) circuit.

The following figure illustrates the external line polarization assembly on the TWDLCA•40DRF mini-DIN RS-485 EIA line:

External polarization can be performed in any of two ways:

Connecting externally the user-provided polarization assembly via mini-DIN fly cable. (Please refer to pin definition for connector.)

Using a polarization tap (configured for 2-wire polarization) and polarization assembly (available soon from the catalog).

Software Configuration

To configure the controller to use a serial connection to send and receive characters using the Modbus protocol, you must:

Step	Description
1	Configure the serial port for Modbus using TwidoSoft.
2	Create in your application a transmission/reception table that will be used by the EXCHx instruction.

Configuring the Port

A Twido controller can use its primary port 1 or an optionally configured port 2 to use the Modbus protocol. To configure a serial port for Modbus:

Step	Action
1	Define any additional communication adapters or modules configured to the base.
2	Right-click on the port and click Edit Controller Comm Setup... and change serial port type to "Modbus".
3	Set the associated communication parameters.

Modbus Master

Modbus master mode allows the controller to send a Modbus query to a slave, and to wait for the response. The Modbus Master mode is only supported via the EXCHx instruction. Both Modbus ASCII and RTU are supported in Modbus Master mode.

The maximum size of the transmitted and/or received frames is 250 bytes. Moreover, the word table associated with the EXCHx instruction is composed of the control, transmission and reception tables.

	Most significant byte	Least significant byte
Control table	Command	Length (Transmission/Reception)
	Reception offset	Transmission offset
Transmission table	Transmitted Byte 1	Transmitted Byte 2
	...	...
	...	Transmitted Byte n
	Transmitted Byte n+1	
Reception table	Received Byte 1	Received Byte 2
	...	...
	...	Received Byte p
	Received Byte p+1	

Note: In addition to queries to invidual slaves, the Modbus master controller can initiate a broadcast query to all slaves. The command byte in case of a boradcast query must be set to 00, while the slave address must be set to 0.

Control table

The Length byte contains the length of the transmission table (maximum 250 bytes), which is overwritten by the number of characters received at the end of the reception, if reception is requested.
This parameter is the length in bytes of the transmission table. If the Tx Offset parameter is equal to 0, this parameter will be equal to the length of the transmission frame. If the Tx Offset parameter is not equal to 0, one byte of the transmission table (indicated by the offset value) will not be transmitted and this parameter is equal to the frame length itself plus 1.
The Command byte in case of Modbus RTU request (except for broadcast) must always equal to 1 (Tx and Rx).
The Tx Offset byte contains the rank (1 for the first byte, 2 for the second byte, and so on) within the Transmission Table of the byte to ignore when transmitting the bytes. This is used to handle the issues associated with byte/word values within the Modbus protocol. For example, if this byte contains 3, the third byte would be ignored, making the fourth byte in the table the third byte to be transmitted.
The Rx Offset byte contains the rank (1 for the first byte, 2 for the second byte, and so on) within the Reception Table to add when transmitting the packet. This is used to handle the issues associated with byte/word values within the Modbus protocol. For example, if this byte contains 3, the third byte within the table would be filled with a ZERO, and the third byte was actually received would be entered into the fourth location in the table.

Transmission/reception tables

When using either mode (Modbus ASCII or Modbus RTU), the Transmission table is filled with the request prior to executing the EXCHx instruction. At execution time, the controller determines what the Data Link Layer is, and performs all conversions necessary to process the transmission and response. Start, end, and check characters are not stored in the Transmission/Reception tables.

Once all bytes are transmitted, the controller switches to reception mode and waits to receive any bytes.

Reception is completed in one of several ways:

- timeout on a character or frame has been detected,
- end of frame character(s) received in ASCII mode,
- the Reception table is full.

Transmitted byte X entries contain Modbus protocol (RTU encoding) data that is to be transmitted. If the communications port is configured for Modbus ASCII, the correct framing characters are appended to the transmission. The first byte contains the device address (specific or broadcast), the second byte contains the function code, and the rest contain the information associated with that function code.

> Note: This is a typical application, but does not define all the possibilities. No validation of the data being transmitted will be performed.

Received Bytes X contain Modbus protocol (RTU encoding) data that is to be received. If the communications port is configured for Modbus ASCII, the correct framing characters are removed from the response. The first byte contains the device address, the second byte contains the function code (or response code), and the rest contain the information associated with that function code.

> Note: This is a typical application, but does not define all the possibilities. No validation of the data being received will be performed, except for checksum verification.

Modbus Slave

Modbus slave mode allows the controller to respond to standard Modbus queries from a Modbus master.

When TSXPCX1031 cable is attached to the controller, TwidoSoft communications are started at the port, temporarily disabling the communications mode that was running prior to the cable being connected.

The Modbus protocol supports two Data Link Layer formats: ASCII and RTU. Each is defined by the Physical Layer implementation, with ASCII using 7 data bits, and RTU using 8 data bits.

When using Modbus ASCII mode, each byte in the message is sent as two ASCII characters. The Modbus ASCII frame begins with a start character (':'), and can end with two end characters (CR and LF). The end of frame character defaults to 0x0A (line feed), and the user can modify the value of this byte during

configuration. The check value for the Modbus ASCII frame is a simple two's complement of the frame, excluding the start and end characters.
Modbus RTU mode does not reformat the message prior to transmitting; however, it uses a different checksum calculation mode, specified as a CRC.
The Modbus Data Link Layer has the following limitations:

Address 1-247

Bits: 128 bits on request

Words: 125 words of 16 bits on request

Message Exchange

The language offers two services for communication:

EXCHx instruction: to transmit/receive messages

%MSGx Function Block: to control the message exchanges.

The Twido controller uses the protocol configured for that port when processing an EXCHx instruction.

> Note: Each communications port can be configured for different protocols or the same. The EXCHx instruction or %MSGx function block for each communications port is accessed by appending the port number (1 or 2).

EXCHx Instruction

The EXCHx instruction allows the Twido controller to send and/or receive information to/from Modbus devices. The user defines a table of words (%MWi:L) containing control information and the data to be sent and/or received (up to 250 bytes in transmission and/or reception). The format for the word table is described earlier.
A message exchange is performed using the EXCHx instruction:

```
Syntax: [EXCHx %MWi:L]
 where: x = port number (1 or 2)
        L = number of words in the control words, transmission and reception tables
```

The Twido controller must finish the exchange from the first EXCHx instruction before a second can be launched. The %MSGx function block must be used when sending several messages.
The processing of the EXCHx list instruction occurs immediately, with any transmissions started under interrupt control (reception of data is also under interrupt control), which is considered background processing.

%MSGx Function Block

The use of the %MSGx function block is optional; it can be used to manage data exchanges. The %MSGx function block has three purposes:

Communications error checking

The error checking verifies that the parameter L (length of the Word table) programmed with the EXCHx instruction is large enough to contain the length of the message to be sent. This is compared with the length programmed in the least significant byte of the first word of the word table.

Coordination of multiple messages

To ensure the coordination when sending multiple messages, the %MSGx function block provides the information required to determine when transmission of a previous message is complete.

Transmission of priority messages

The %MSGx function block allows current message transmissions to be stopped in order to allow the immediate sending of an urgent message.

The %MSGx function block has one input and two outputs associated with it:

Input/Output	Definition	Description
R	Reset input	Set to 1: re-initializes communication or resets block (%MSGx.E = 0 and %MSGx.D = 1).
%MSGx.D	Communication complete	0: request in progress. 1: communication done if end of transmission, end character received, error, or reset of block.
%MSGx.E	Error	0: message length OK and link OK. 1: if bad command, table incorrectly configured, incorrect character received (speed, parity, and so on.), or reception table full.

Limitations

It is important to note the following limitations:

- Port 2 presence and configuration (RS232 or RS485) is checked at power-up or reset
- Any message processing on Port 1 is aborted when the TwidoSoft is connected
- EXCHx or %MSG can not be processed on a port configured as Remote Link
- EXCHx aborts active Modbus Slave processing
- Processing of EXCHx instructions is not re-tried in the event of an error
- Reset input (R) can be used to abort EXCHx instruction reception processing
- EXCHx instructions can be configured with a time out to abort reception
- Multiple messages are controlled via %MSGx.D

Error and Operating Mode Conditions

If an error occurs when using the EXCHx instruction, bits %MSGx.D and

%MSGx.E are set to 1 and system word %SW63 contains the error code for Port 1, and %SW64 contains the error code for Port 2.

System Words	Use
%SW63	EXCH1 error code: 0 - operation was successful 1 – number of bytes to be transmitted is too great (> 250) 2 - transmission table too small 3 - word table too small 4 - receive table overflowed 5 - time-out elapsed 6 - transmission 7 - bad command within table 8 - selected port not configured/available 9 - reception error 10 - can not use %KW if receiving 11 - transmission offset larger than transmission table 12 - reception offset larger than reception table 13 - controller stopped EXCH processing
%SW64	EXCH2 error code: See %SW63.

Master Controller Restart

If a master/slave controller restarts, one of the following events happens:

A cold start (%S0 = 1) forces a re-initialization of the communications.

A warm start (%S1 = 1) forces a re-initialization of the communications.

In Stop mode, the controller stops all Modbus communications.

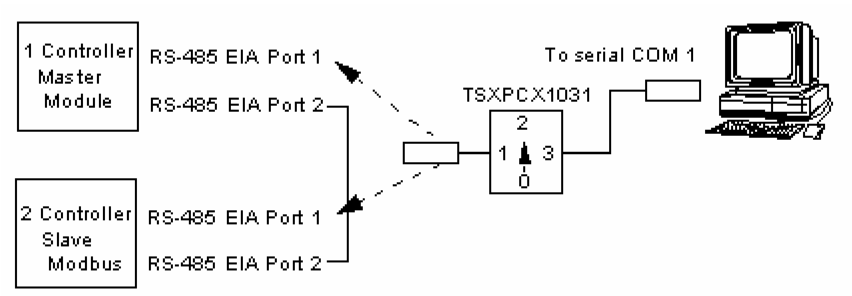
Modbus Link Example 1

To configure a Modbus Link, you must:

1. Configure the hardware.
2. Connect the Modbus communications cable.
3. Configure the port.
4. Write an application.
5. Initialize the Animation Table Editor.

The diagrams below illustrate the use of Modbus request code 3 to read a slave’s output words. This example uses two Twido Controllers.

Step 1: Configure the Hardware:



The hardware configuration is two Twido controllers. One will be configured as the Modbus Master and the other as the Modbus Slave.

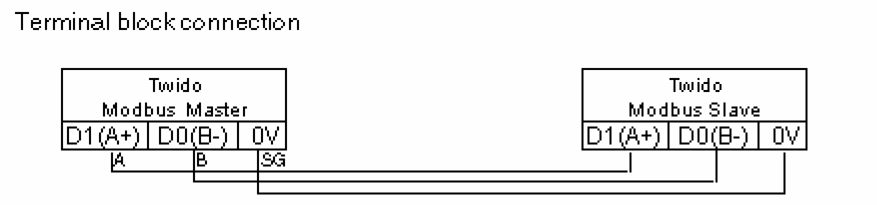
Note: In this example, each controller is configured to use EIA RS-485 on Port 1 and an optional EIA RS-485 Port 2. On a Modular controller, the optional Port 2 can be either a TWDNOZ485D or a TWDNOZ485T, or if you use TWDXCPODM, it can be either a TWDNAC485D or a TWDNAC485T. On a Compact controller, the optional Port 2 can be either a TWDNAC485D or a TWDNAC485T.

To configure each controller, connect the TSXPCX1031 cable to Port 1 of the controller.

Note: The TSXPCX1031 can only be connected to one controller at a time, on RS-485 EIA port 1 only.

Next, connect the cable to the COM 1 port of the PC. Be sure that the cable is in switch position 2. Download and monitor the application. Repeat procedure for second controller.

Step 2: Connect the Modbus Communications Cable:



The wiring in this example demonstrates a simple point to point connection. The three signals D1(A+), D0(B-), and COM(0V) are wired according to the diagram.

If using Port 1 of the Twido controller, the DPT signal (pin 5) must be tied to circuit common (pin 7). This conditioning of DPT determines if TwidoSoft is connected. When tied to the ground, the controller will use the port configuration set in the application to determine the type of communication.

Step 3: Port Configuration:

Hardware -> Add Option
TWDNOZ485-

Hardware => Controller Comm. Setting	
Serial Port 2	
Protocol	Modbus
Address	1
Baud Rate	19200
Data Bits	8 (RTU)
Parity	None
Stop Bit	1
Response Timeout (x100ms)	10
Time between frames (ms)	10

Hardware -> Add Option
TWDNOZ485-

Hardware => Controller Comm. Setting	
Serial Port 2	
Protocol	Modbus
Address	2
Baud Rate	19200
Data Bits	8 (RTU)
Parity	None
Stop Bit	1
Response Timeout (x100ms)	100
Time between frames (ms)	10

In both master and slave applications, the optional EIA RS-485 ports are configured. Ensure that the controller's communication parameters are modified in Modbus protocol and at different addresses.
In this example, the master is set to an address of 1 and the slave to 2. The number of bits is set to 8, indicating that we will be using Modbus RTU mode. If this had been set to 7, then we would be using Modbus-ASCII mode. The only other default modified was to increase the response timeout to 1 second.

> Note: Since Modbus RTU mode was selected, the "End of Frame" parameter was ignored.

Step 4: Write the application:

```
LD 1
[%MW0 := 16#0106 ]
[%MW1 := 16#0300 ]
[%MW2 := 16#0203 ]
[%MW3 := 16#0000 ]
[%MW4 := 16#0004 ]
LD 1
AND  %MSG2.D
[EXCH2  %MW0:11]
LD  %MSG2.E
ST  %Q0.0
END
```

```
LD 1
[%MW0 := 16#6566 ]
[%MW1 := 16#6768 ]
[%MW2 := 16#6970 ]
[%MW3 := 16#7172 ]
END
```

Using TwidoSoft, an application program is written for both the master and the slave. For the slave, we simply write some memory words to a set of known values. In the master, the word table of the EXCHx instruction is initialized to read 4 words from the slave at Modbus address 2 starting at location %MW0.

> Note: Notice the use of the RX offset set in %MW1 of the Modbus master. The offset of three will add a byte (value = 0) at the third position in the reception area of the table. This aligns the words in the master so that they fall correctly on word boundaries. Without this offset, each word of data would be split between two words in the exchange block. This offset is used for convenience.

Before executing the EXCH2 instruction, the application checks the communication bit associated with %MSG2. Finally, the error status of the %MSG2 is sensed and stored on the first output bit on the local base controller I/O. Additional error checking using %SW64 could also be added to make this more accurate.

Step 5:Initialize the animation table editor in the master:

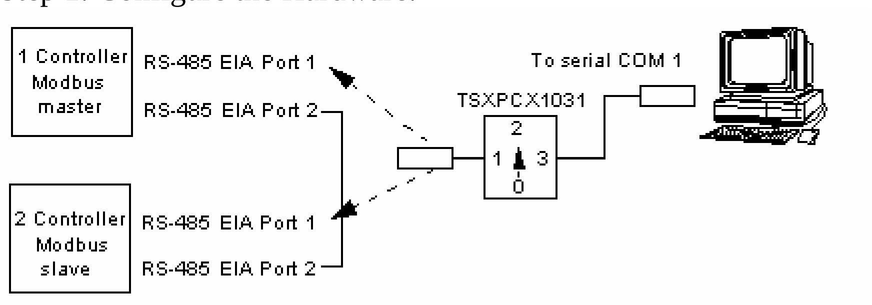
	Address	Current	Retained	Format
1	%MW5	0203	0000	Hexadecimal
2	%MW6	0008	0000	Hexadecimal
3	%MW7	6566	0000	Hexadecimal
4	%MW8	6768	0000	Hexadecimal
5	%MW9	6970	0000	Hexadecimal
6	%MW10	7172	0000	Hexadecimal

After downloading and setting each controller to run, open an animation table on the master. Examine the response section of the table to check that the response code is 3 and that the correct number of bytes was read. Also in this example, note that the words read from the slave (beginning at %MW7) are aligned correctly with the word boundaries in the master.

Modbus Link Example 2

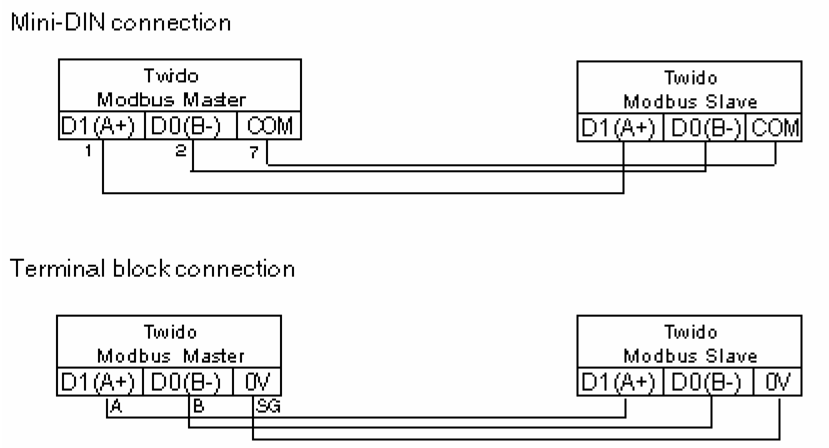
The diagram below illustrates the use of Modbus request 16 to write output words to a slave. This example uses two Twido Controllers.

Step 1: Configure the Hardware:



The hardware configuration is identical to the previous example.

Step 2: Connect the Modbus Communications Cable (RS-485):



The Modbus communications cabling is identical to the previous example.

Step 3: Port Configuration:

Hardware -> Add Option
TWDNOZ485-

Hardware => Controller Comm. Setting

Serial Port 2

Protocol	Modbus
Address	1
Baud Rate	19200
Data Bits	8 (RTU)
Parity	None
Stop Bit	1
Response Timeout (x100ms)	10
Time between frames (ms)	10

Hardware -> Add Option
TWDNOZ485-

Hardware => Controller Comm. Setting

Serial Port 2

Protocol	Modbus
Address	2
Baud Rate	19200
Data Bits	8 (RTU)
Parity	None
Stop Bit	1
Response Timeout (x100ms)	100
Time between frames (ms)	10

The port configurations are identical to those in the previous example.

Step 4: Write the application:

```
LD 1
[%MW0 := 16#010C ]
[%MW1 := 16#0007 ]
[%MW2 := 16#0210 ]
[%MW3 := 16#0010 ]
[%MW4 := 16#0002 ]
[%MW5 := 16#0004 ]
[%MW6 := 16#6566 ]
[%MW7 := 16#6768 ]
LD 1
AND %MSG2.D
[EXCH2 %MW0:11]
LD %MSG2.E
ST %Q0.0
END
```

```
LD 1
[%MW18 := 16#FFFF ]
END
```

Using TwidoSoft, an application program is created for both the master and the slave. For the slave, write a single memory word %MW18. This will allocate space on the slave for the memory addresses from %MW0 through %MW18. Without allocating the space, the Modbus request would be trying to write to locations that did not exist on the slave.

In the master, the word table of the EXCH2 instruction is initialized to read 4 bytes to the slave at Modbus address 2 at the address %MW16 (10 hexadecimal).

Note: Notice the use of the TX offset set in %MW1 of the Modbus master application. The offset of seven will suppress the high byte in the sixth word (the value 00 hexadecimal in %MW5). This works to align the data values in the transmission table of the word table so that they fall correctly on word boundaries.

Before executing the EXCH2 instruction, the application checks the communication bit associated with %MSG2. Finally, the error status of the %MSG2 is sensed and stored on the first output bit on the local base controller I/O. Additional error checking using %SW64 could also be added to make this more accurate.

Step 5:Initialize the Animation Table Editor:

Create the following animation table on the master:

	Address	Current	Retained	Format
1	%MW0	010C	0000	Hexadecimal
2	%MW1	0007	0000	Hexadecimal
3	%MW2	0210	0000	Hexadecimal
4	%MW3	0010	0000	Hexadecimal
5	%MW4	0002	0000	Hexadecimal
6	%MW5	0004	0000	Hexadecimal
7	%MW6	6566	0000	Hexadecimal
8	%MW7	6768	0000	Hexadecimal
9	%MW8	0210	0000	Hexadecimal
10	%MW9	0010	0000	Hexadecimal
11	%MW10	0004	0000	Hexadecimal

Create the following animation table on the slave:

	Address	Current	Retained	Format
1	%MW16	6566	0000	Hexadecimal
2	%MW17	6768	0000	Hexadecimal

After downloading and setting each controller to run, open an animation table on the slave controller. The two values in %MW16 and %MW17 are written to the slave. In the master, the animation table can be used to examine the reception table portion of the exchange data. This data displays the slave address, the response code, the first word written, and the number of words written starting at %MW8 in the example above.

Standard Modbus Requests

T

Introduction

These requests are used to exchange memory words or bits between remote devices. The table format is the same for both RTU and ASCII modes.

Bit	%Mi
Word	%MWi

Modbus Master: Read N Bits

The following table represents requests 01 and 02.

		Table Index	Most significant byte	Least significant byte
Control table		0	01 (Transmission/reception)	06 (Transmission length) (*)
		1	03 (Reception offset)	00 (Transmission offset)
Transmission table		2	Slave@(1..247)	01 or 02 (Request code)
		3	Address of the first bit to read	
		4	N1 = Number of bits to read	
Reception table (after response)		5	Slave@(1..247)	01 or 02 (Response code)
		6	00 (byte added by Rx Offset action)	N2 = Number of data bytes to read = [1+(N1-1)/8], where [] means integral part
		7	Value of the 1st byte (value = 00 or 01)	Value of the 2nd byte (if N1>1)
		8	Value of the 3rd byte (if N1>1)	
		...		
	(N2/2)+6 (if N2 is even) (N2/2+1)+6 (if N2 is odd		Value of the N2th byte (if N1>1)	

(*) This byte also receives the length of the string transmitted after response

---

Modbus Master: Read N Words

The following table represents requests 03 and 04.

	Table Index	Most significant byte	Least significant byte
Control table	0	01 (Transmission/reception)	06 (Transmission length) (*)
	1	03 (Reception Offset)	00 (Transmission offset)
Transmission	2	Slave@(1..247)	03 or 04 (Request

table			code)
	3	Address of the first word to read	
	4	N = Number of words to read	
Reception table (after response)	5	Slave@(1..247)	03 or 04 (Response code)
	6	00 (byte added by Rx Offset action)	2*N (number of bytes read)
	7	First word read	
	8	Second word read (if N>1)	
	...		
	N+6	Word N read (if N>2)	

(*) This byte also receives the length of the string transmitted after response

Note: The Rx offset of three will add a byte (value = 0) at the third position in the reception table. This ensures a good positioning of the number of bytes read and of the read words' values in this table.

Modbus Master: Write Bit

This table represents Request 05.

	Table Index	Most significant byte	Least significant byte
Control table	0	01 (Transmission/reception)	06 (Transmission length) (*)
	1	00 (Reception offset)	00 (Transmission offset)
Transmission table	2	Slave@(1..247)	05 (Request code)
	3	Address of the bit to write	
	4	Bit value to write	
Reception table (after response)	5	Slave@(1..247)	05 (Response code)
	6	Address of the bit written	
	7	Value written	

(*) This byte also receives the length of the string transmitted after response

Note:

This request does not need the use of offset.

The response frame is the same as the request frame here (in a normal case).

For a bit to write 1, the associated word in the transmission table must contain the value FF00H, and 0 for the bit to write 0.

Modbus Master: Write Word

This table represents Request 06.

	Table Index	Most significant byte	Least significant byte
Control table	0	01 (Transmission/reception)	06 (Transmission length) (*)
	1	00 (Reception offset)	00 (Transmission offset)
Transmission table	2	Slave@(1..247)	06 (Request code)
	3	Address of the word to write	
	4	Word value to write	
Reception table (after response)	5	Slave@(1..247)	06 (Response code)
	6	Address of the word written	
	7	Value written	

(*) This byte also receives the length of the string transmitted after response

Note:

This request does not need the use of offset.

The response frame is the same as the request frame here (in a normal case).

Modbus Master: Write of N Bits

This table represents Request 15.

	Table Index	Most significant byte	Least significant byte
Control table	0	01 (Transmission/reception)	8 + number of bytes (transmission)
	1	00 (Reception Offset)	07 (Transmission offset)
Transmission table	2	Slave@(1..247)	15 (Request code)
	3	Number of the first bit to write	
	4	N1 = Number of bits to write	
	5	00 (byte not sent, offset effect)	N2 = Number of data bytes to write = [1+(N1-1)/8], where [] means integral part
	6	Value of the 1st byte	Value of the 2nd byte
	7	Value of the 3rd byte	Value of the 4th byte

	...		
	(N2/2)+5 (if N2 is even) (N2/2+1)+5 (if N2 is odd	Value of the N2th byte	
Reception table (after response)		Slave@(1..247)	15 (Response code)
		Address of the 1st bit written	
		Address of bits written (= N1)	

Note:

The Tx Offset=7 will suppress the 7th byte in the sent frame. This also allows a good correspondence of words' values in the transmission table.

Modbus Master: Write of N Words

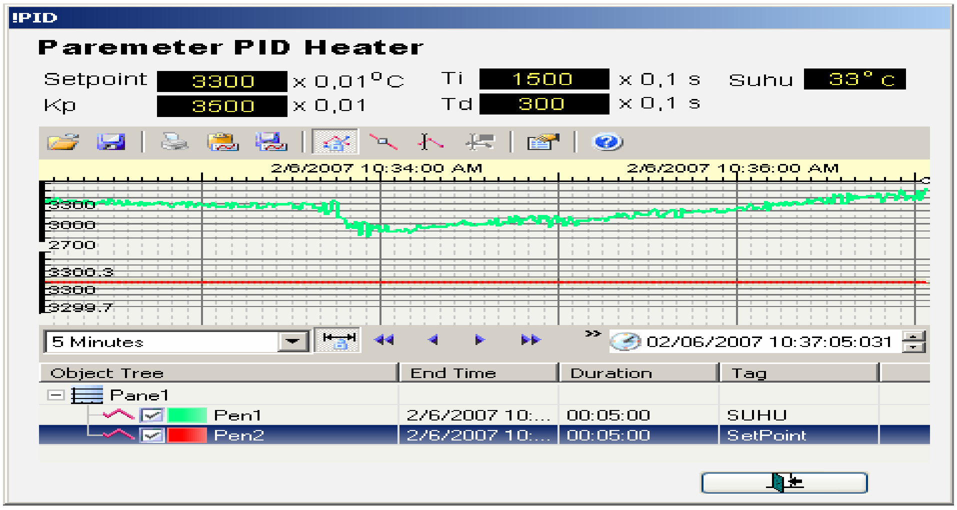
This table represents Request 16.

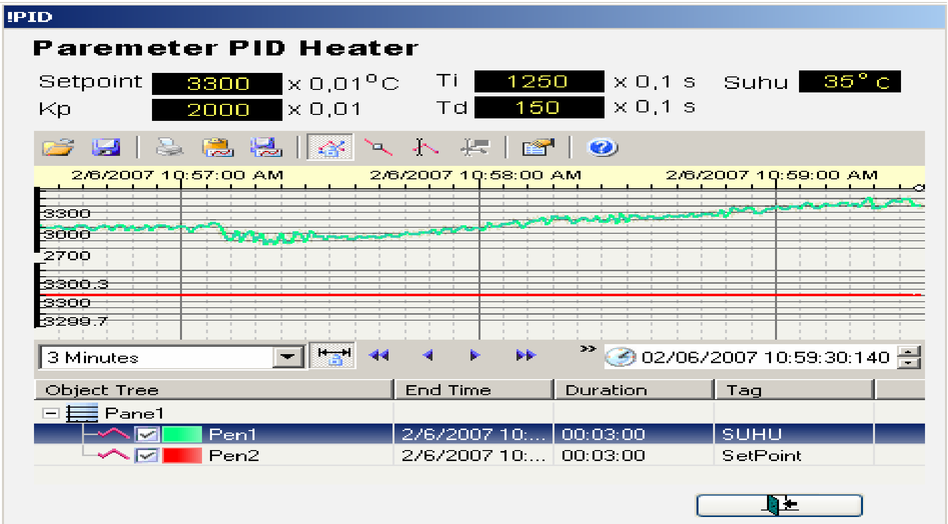
	Table Index	Most significant byte	Least significant byte
Control table	0	01 (Transmission/reception)	8 + (2*N) (Transmission length)
	1	00 (Reception offset)	07 (Transmission offset)
Transmission table	2	Slave@(1..247)	16 (Request code)
	3	Address of the first word to write	
	4	N = Number of words to write	
	5	00 (byte not sent, offset effect)	2*N = Number of bytes to write
	6	First word value to write	
	7	Second value to write	
	...		
	N+5	N values to write	
Reception table (after response)	N+6	Slave@(1..247)	16 (Response code)
	N+7	Address of the first word written	
	N+8	Address of words written (= N)	

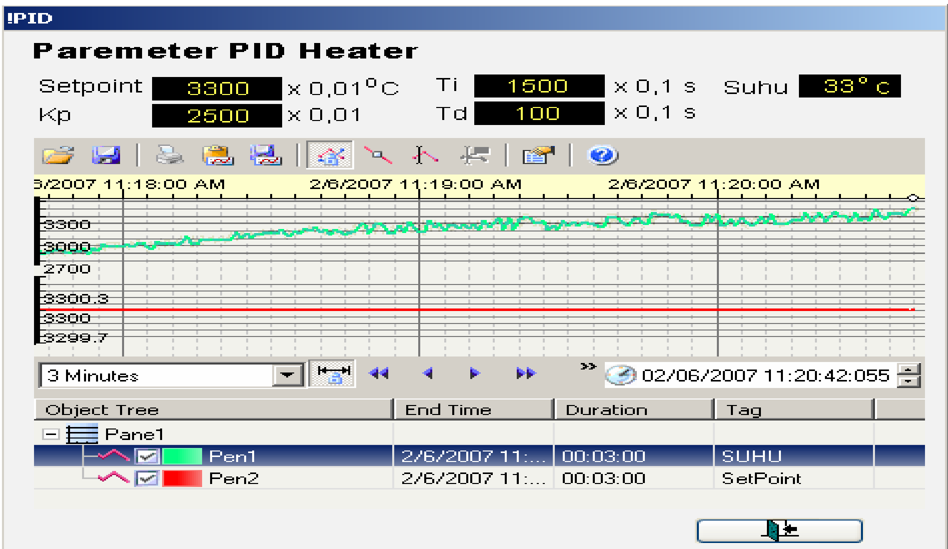
Note: The Tx Offset = 7 will suppress the 5th MMSB byte in the sent frame. This also allows a good correspondence of words' values in the transmission table.

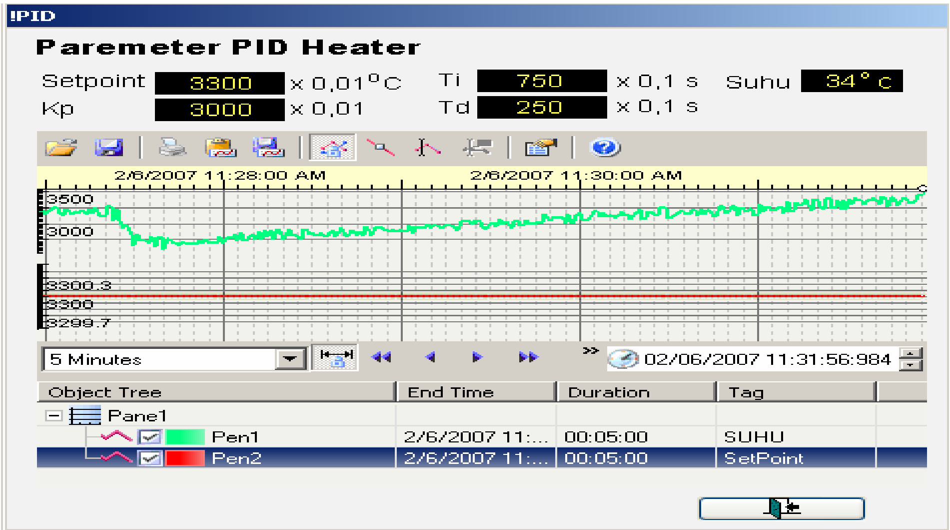
LAMPIRAN D

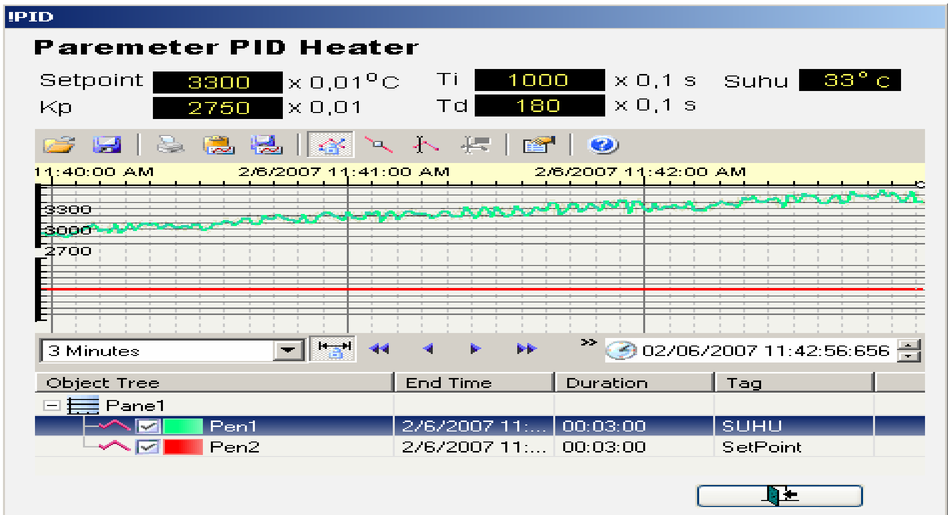
GAMBAR TRENDING PARAMETER PID PLANT SIMULASI PENGENDALIAN TEMPERATUR

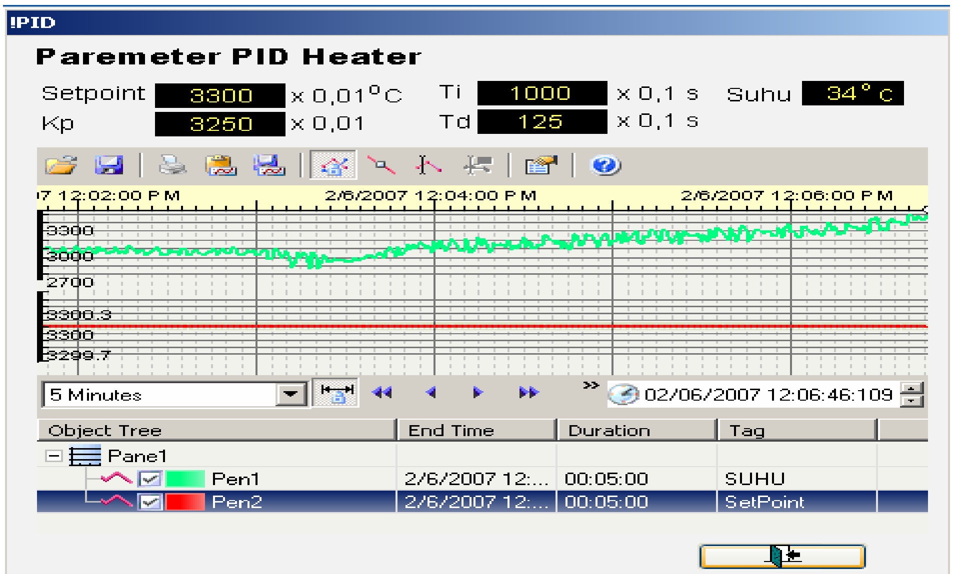

PID
Paremeter PID Heater
Setpoint 3300 x 0,01°C
Kp 3500 x 0,01
Ti 1500 x 0,1 s
Td 300 x 0,1 s
Suhu 33° c
2/6/2007 10:34:00 AM
2/6/2007 10:36:00 AM
3300
3000
2700
3300.3
3300
3299.7
5 Minutes
02/06/2007 10:37:05:031
Object Tree
End Time
Duration
Tag
Pane1
Pen1
2/6/2007 10:...
00:05:00
SUHU
Pen2
2/6/2007 10:...
00:05:00
SetPoint

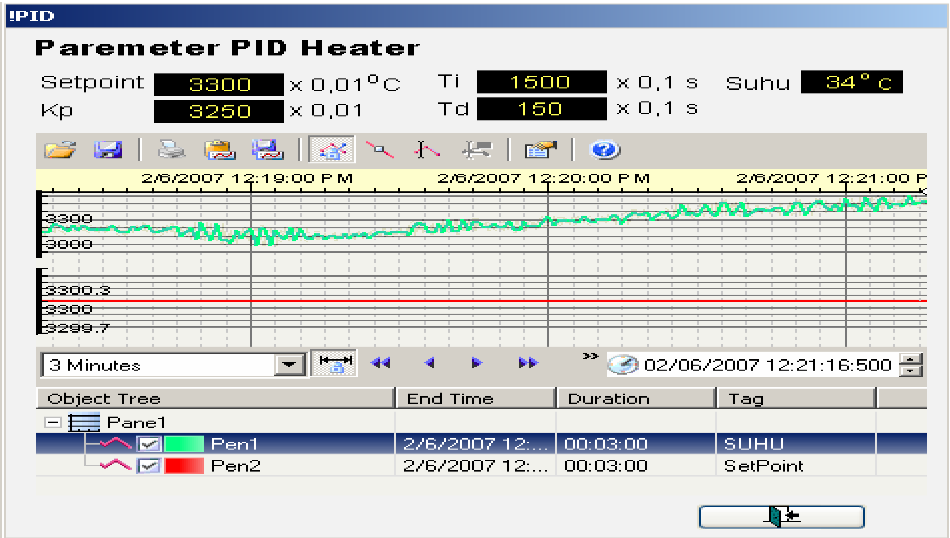

PID
Paremeter PID Heater
Setpoint 3300 x 0,01°C
Kp 2000 x 0,01
Ti 1250 x 0,1 s
Td 150 x 0,1 s
Suhu 35° c
2/6/2007 10:57:00 AM
2/6/2007 10:58:00 AM
2/6/2007 10:59:00 AM
3300
3000
2700
3300.3
3300
3299.7
3 Minutes
02/06/2007 10:59:30:140
Object Tree
End Time
Duration
Tag
Pane1
Pen1
2/6/2007 10:...
00:03:00
SUHU
Pen2
2/6/2007 10:...
00:03:00
SetPoint

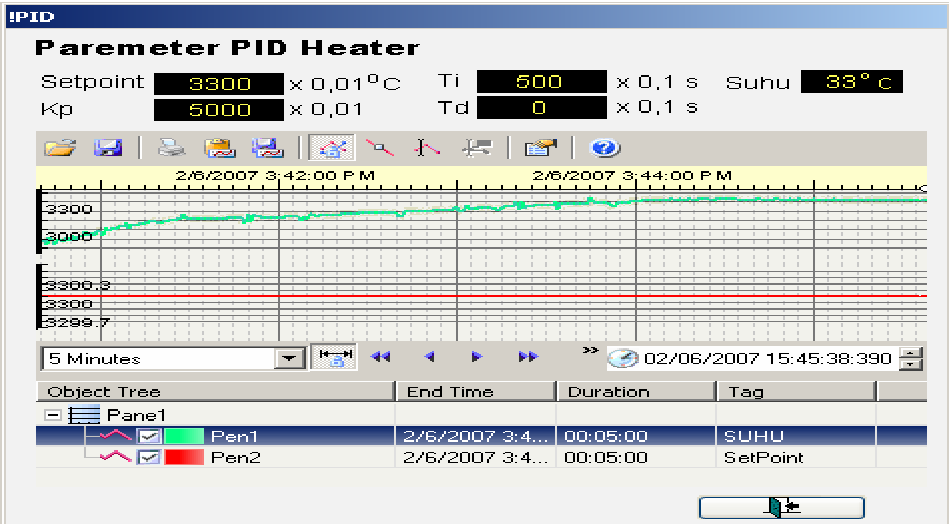

PID
Paremeter PID Heater
Setpoint 3300 x 0,01°C
Kp 2500 x 0,01
Ti 1500 x 0,1 s
Td 100 x 0,1 s
Suhu 33° c
/2007 11:18:00 AM
2/6/2007 11:19:00 AM
2/6/2007 11:20:00 AM
3300
3000
2700
3300.3
3300
3299.7
3 Minutes
02/06/2007 11:20:42:055
Object Tree
End Time
Duration
Tag
Pane1
Pen1
2/6/2007 11:...
00:03:00
SUHU
Pen2
2/6/2007 11:...
00:03:00
SetPoint

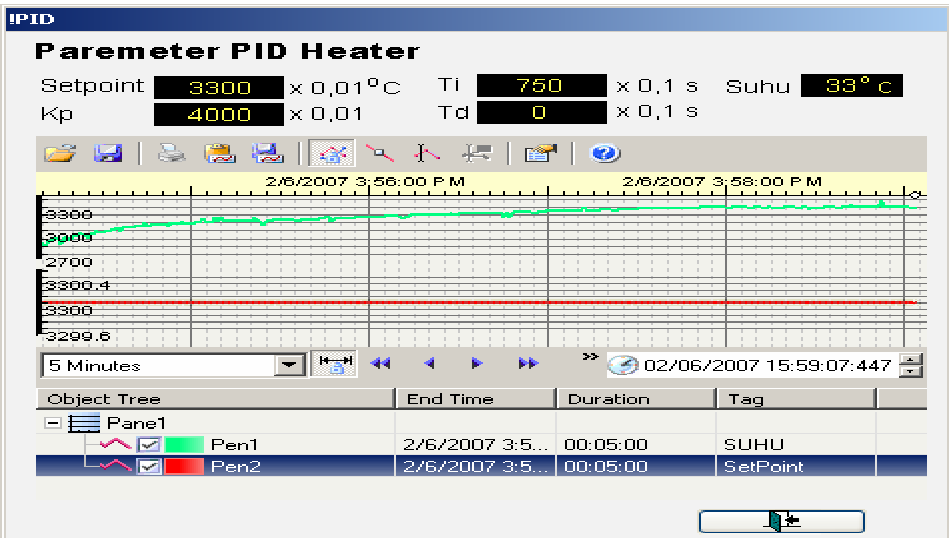

PID
Paremeter PID Heater
Setpoint 3300 x 0,01°C
Kp 3000 x 0,01
Ti 750 x 0,1 s
Td 250 x 0,1 s
Suhu 34° c
2/6/2007 11:28:00 AM
2/6/2007 11:30:00 AM
3500
3000
3300.3
3300
3299.7
5 Minutes
02/06/2007 11:31:56:984
Object Tree
End Time
Duration
Tag
Pane1
Pen1 2/6/2007 11:... 00:05:00 SUHU
Pen2 2/6/2007 11:... 00:05:00 SetPoint

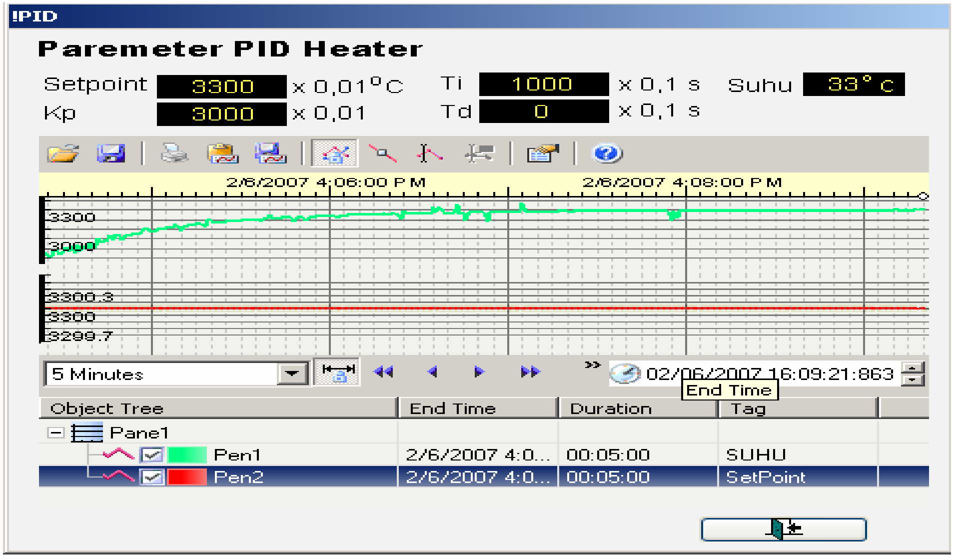

PID
Paremeter PID Heater
Setpoint 3300 x 0,01°C
Kp 2750 x 0,01
Ti 1000 x 0,1 s
Td 180 x 0,1 s
Suhu 33° c
11:40:00 AM
2/6/2007 11:41:00 AM
2/6/2007 11:42:00 AM
3300
3000
2700
3 Minutes
02/06/2007 11:42:56:656
Object Tree
End Time
Duration
Tag
Pane1
Pen1 2/6/2007 11:... 00:03:00 SUHU
Pen2 2/6/2007 11:... 00:03:00 SetPoint

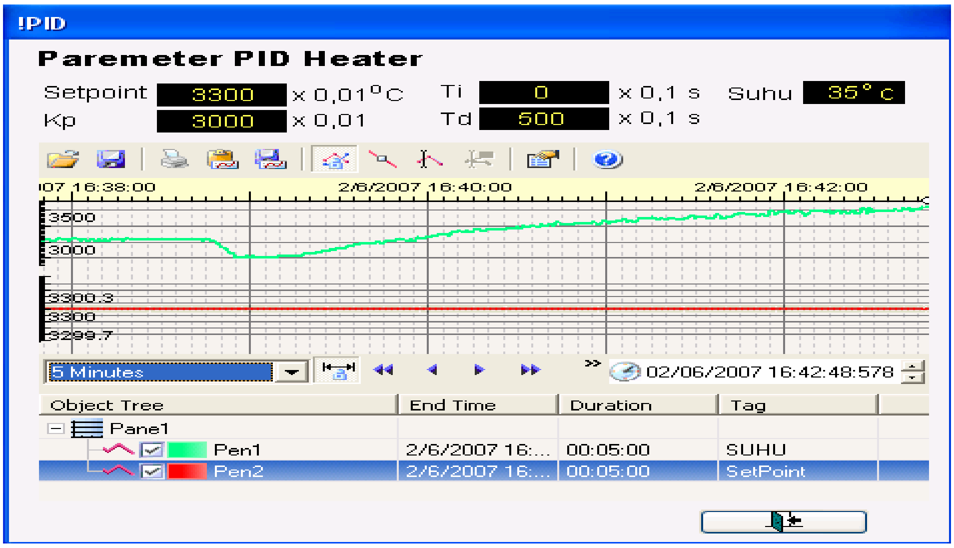

PID
Paremeter PID Heater
Setpoint 3300 x 0,01°C
Kp 3250 x 0,01
Ti 1000 x 0,1 s
Td 125 x 0,1 s
Suhu 34° c
7 12:02:00 PM
2/6/2007 12:04:00 PM
2/6/2007 12:06:00 PM
3300
3000
2700
3300.3
3300
3299.7
5 Minutes
02/06/2007 12:06:46:109
Object Tree
End Time
Duration
Tag
Pane1
Pen1 2/6/2007 12:... 00:05:00 SUHU
Pen2 2/6/2007 12:... 00:05:00 SetPoint

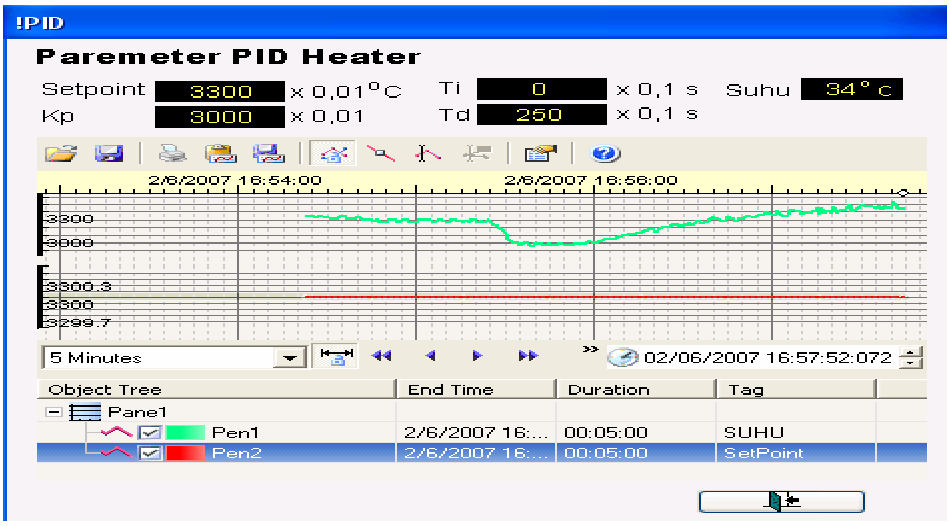

PID
Paremeter PID Heater
Setpoint 3300 x 0,01°C
Kp 3250 x 0,01
Ti 1500 x 0,1 s
Td 150 x 0,1 s
Suhu 34° c
3 Minutes
02/06/2007 12:21:16:500
Object Tree
End Time
Duration
Tag
Pane1
Pen1
2/6/2007 12:...
00:03:00
SUHU
Pen2
2/6/2007 12:...
00:03:00
SetPoint

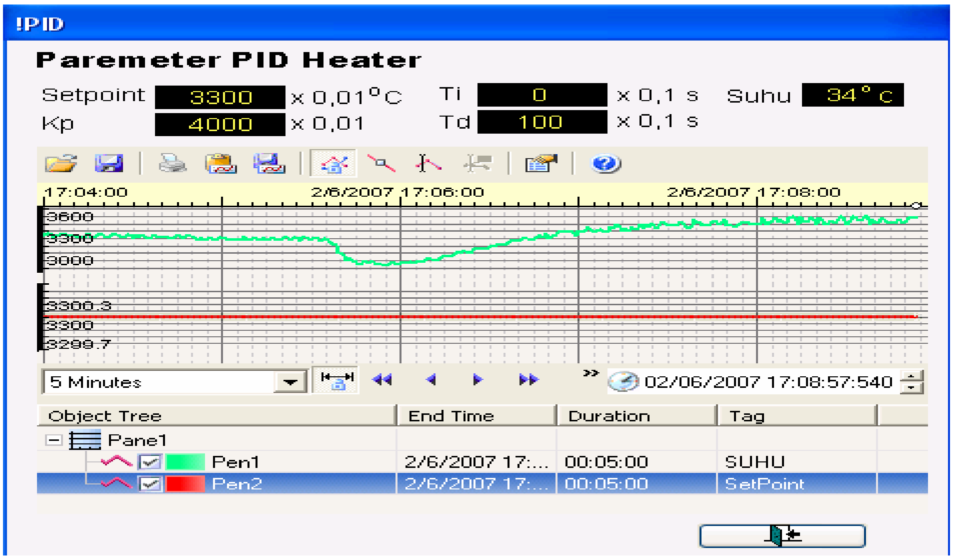

PID
Paremeter PID Heater
Setpoint 3300 x 0,01°C
Kp 5000 x 0,01
Ti 500 x 0,1 s
Td 0 x 0,1 s
Suhu 33° c
5 Minutes
02/06/2007 15:45:38:390
Object Tree
End Time
Duration
Tag
Pane1
Pen1
2/6/2007 3:4...
00:05:00
SUHU
Pen2
2/6/2007 3:4...
00:05:00
SetPoint

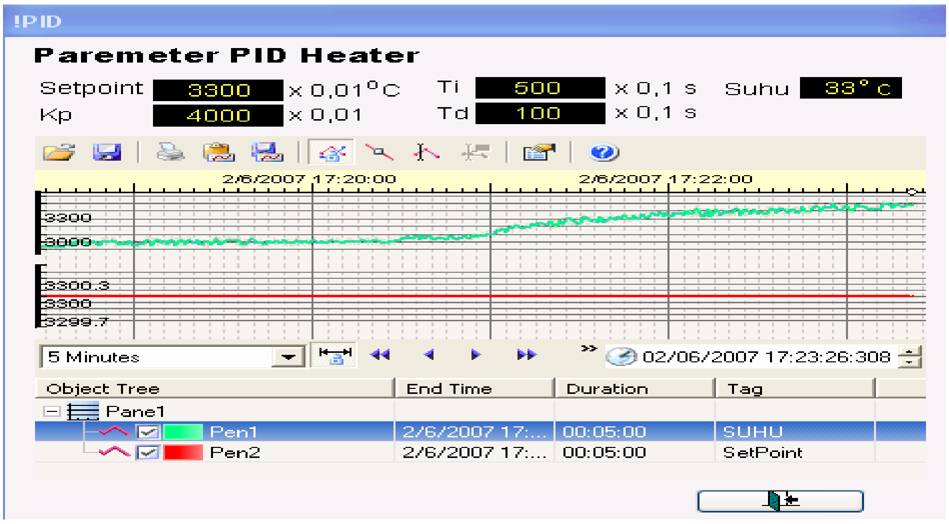

PID
Paremeter PID Heater
Setpoint 3300 x 0,01°C
Kp 4000 x 0,01
Ti 750 x 0,1 s
Td 0 x 0,1 s
Suhu 33° c
5 Minutes
02/06/2007 15:59:07:447
Object Tree
End Time
Duration
Tag
Pane1
Pen1
2/6/2007 3:5...
00:05:00
SUHU
Pen2
2/6/2007 3:5...
00:05:00
SetPoint

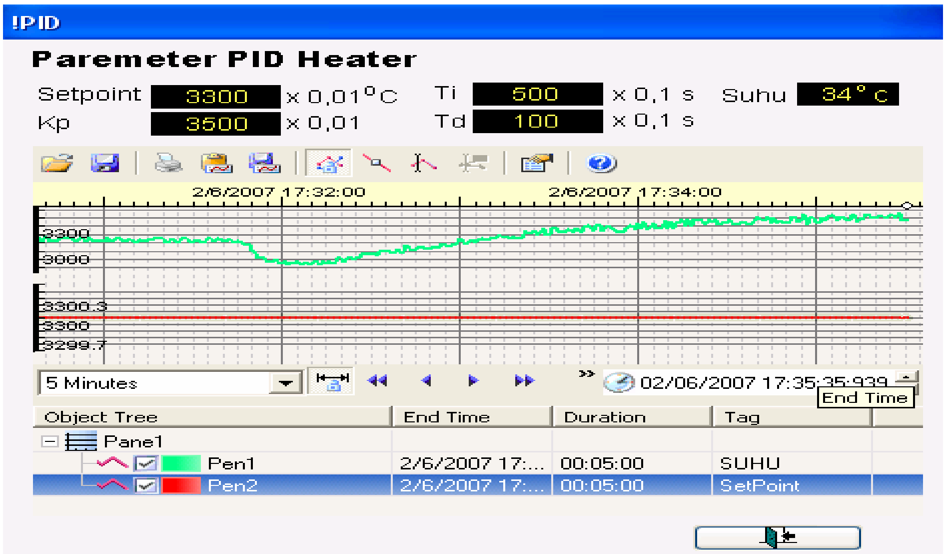

!PID
Paremeter PID Heater
Setpoint 3300 x 0,01°C
Kp 3000 x 0,01
Ti 1000 x 0,1 s
Td 0 x 0,1 s
Suhu 33° c
2/6/2007 4:06:00 PM
2/6/2007 4:08:00 PM
5 Minutes
02/06/2007 16:09:21:863
End Time
Object Tree
End Time
Duration
Tag
Pane1
Pen1
2/6/2007 4:0...
00:05:00
SUHU
Pen2
2/6/2007 4:0...
00:05:00
SetPoint


!PID
Paremeter PID Heater
Setpoint 3300 x 0,01°C
Kp 3000 x 0,01
Ti 0 x 0,1 s
Td 500 x 0,1 s
Suhu 35° c
2/6/2007 16:40:00
2/6/2007 16:42:00
5 Minutes
02/06/2007 16:42:48:578
Object Tree
End Time
Duration
Tag
Pane1
Pen1
2/6/2007 16:...
00:05:00
SUHU
Pen2
2/6/2007 16:...
00:05:00
SetPoint


!PID
Paremeter PID Heater
Setpoint 3300 x 0,01°C
Kp 3000 x 0,01
Ti 0 x 0,1 s
Td 250 x 0,1 s
Suhu 34° c
2/6/2007 16:54:00
2/6/2007 16:56:00
5 Minutes
02/06/2007 16:57:52:072
Object Tree
End Time
Duration
Tag
Pane1
Pen1
2/6/2007 16:...
00:05:00
SUHU
Pen2
2/6/2007 16:...
00:05:00
SetPoint


PID
Paremeter PID Heater
Setpoint 3300 x 0,01°C
Kp 4000 x 0,01
Ti 0 x 0,1 s
Td 100 x 0,1 s
Suhu 34° c
5 Minutes
02/06/2007 17:08:57:540
Object Tree
End Time
Duration
Tag
Pane1
Pen1
Pen2
SUHU
SetPoint


PID
Paremeter PID Heater
Setpoint 3300 x 0,01°C
Kp 4000 x 0,01
Ti 500 x 0,1 s
Td 100 x 0,1 s
Suhu 33° c
5 Minutes
02/06/2007 17:23:26:308
Object Tree
End Time
Duration
Tag
Pane1
Pen1
Pen2
SUHU
SetPoint


PID
Paremeter PID Heater
Setpoint 3300 x 0,01°C
Kp 3500 x 0,01
Ti 500 x 0,1 s
Td 100 x 0,1 s
Suhu 34° c
5 Minutes
Object Tree
End Time
Duration
Tag
Pane1
Pen1
Pen2
SUHU
SetPoint