Automation with PLC

Laboratory report 04

Microcontroller Laboratory Exercise

Student: Kormoua Khongmeng – I3MLPQ
Muneeb Ali – ZG0KE2
Instructor: Kovács Gábor

Budapest University of Technology and Economics
2022

CONFIGURATION MICRO830

The detailed description of the configuration goes here

In this Laboratory we want to create a control for a manufacturing line model shown below with PLC.

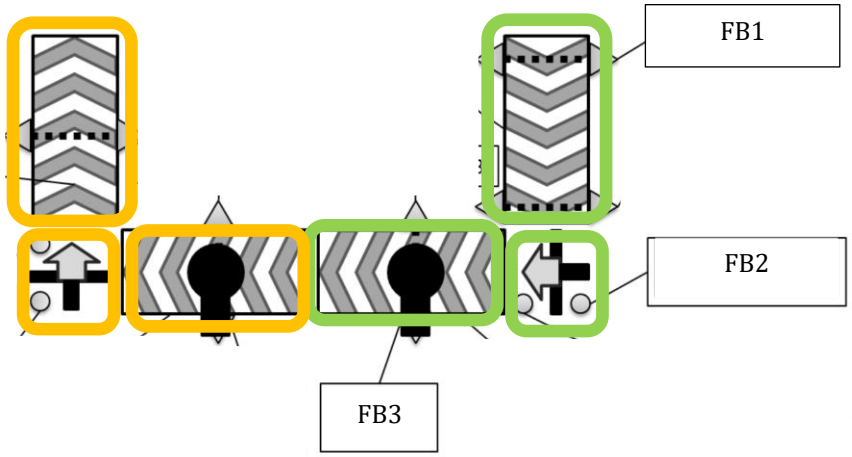


Figure 1: manufacturing line model

For that, we created a control system to drive a manufacturing line model having a main program function call “Prog1” which will control the manufacturing line from start till the end. This Program will call another function block that have an implementation to drive each part of the manufacturing line. Handle each part of the model separately (as shown in the figure above separated by orange and green triangle) makes a task more transparent, easier to implement and read. We will create a control for this manufacturing line using the principle of state machine. We created a state machine for each function block and for each of them the output mapping is also created. For our Lab, we had time just to finish 3 blocks indicated by a green rectangle shown in figure 1

PROG1 BLOCK:

The first one is the main program “Prog1” which as we said, it is the main function that drive the whole manufacturing line by using a sub function block again. For this task a given block diagram below will show the basic working principle of the whole manufacturing line. Again, green rectangles indicate the block diagram that we have already implemented and tested.

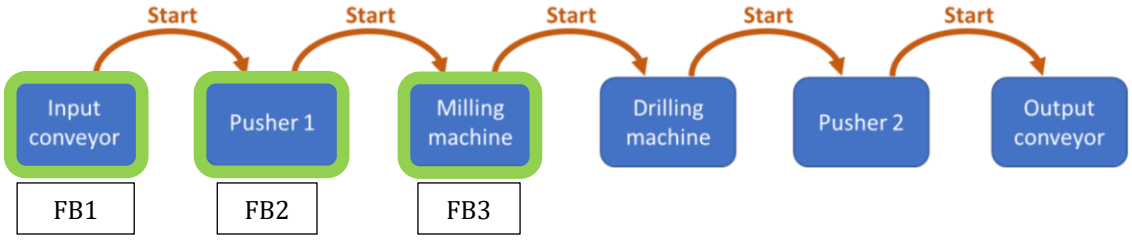


Figure 2: block diagram of the whole manufacturing line model

For this program “Prog1”, basically, we just want to add a sub function block here and let them work in cyclic manner. That is why we first add our FB1 function block (Input conveyor) in a rung here and follow up with FB2 (Pusher 1) and FB3 (Milling machine).

FB1 FUNCTION BLOCK:

This function block responsible for our first conveyor to move. We will divide this function block into 3 states:

1. IDLE (No object is detected, no need to do anything)
2. driveConv1 (the conveyor is moving)
3. stopConv1 (the conveyor move for 2 more seconds)

As we mentioned earlier, we solved this by state machine. So here we created a state machine diagram as shown in figure 3 below.

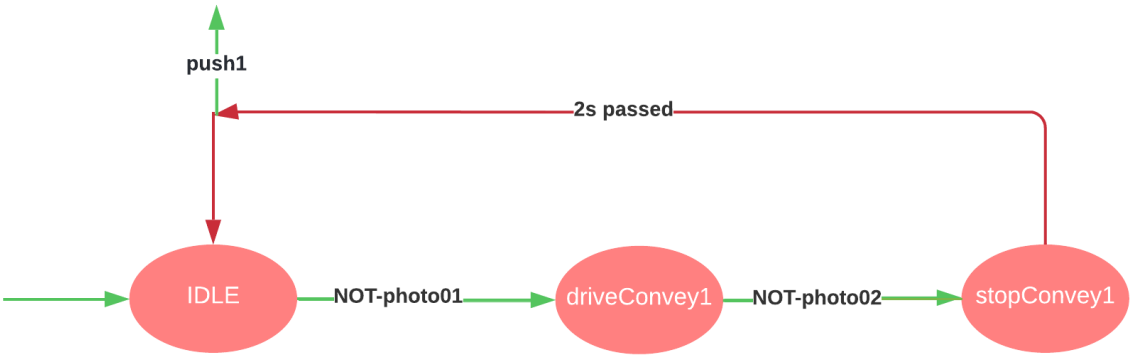


Figure 3: State Machine diagram for FB1

As well as the State machine diagram, we also need to create a output mapping table for each Function Block that we have and here is shown below as table 1.

State	IDLE	driveConvey1	stopConvey1
convey1	0	1	1

Table 1: output mapping table for FB1

Based on the state machine diagram and the output mapping table as shown in Figure 3 and Table 1, we can create our function block for driving the first conveyor as shown in the Ladder diagram “Controller.Micro830.Micro830.FB1” below.

For this block we will have an input and output as:

- photo01 (an input connected to photo1, this is an address of the photo sensor 1 in the manufacturing line model)
- photo02 (an input connected to photo2, this is an address of the photo sensor 2 in the manufacturing line model)
- push1 (an output that will be used to drive the next function block “FB2”)
- cov1 (an output connected to conv1, this is an address of the conveyor 1 in the manufacturing line model)

Initially we when the main program is run, we will be brought to the “IDLE” state which basically does nothing, just stay in stand-by mode. After the object is detected by a first photo sensor “photo01” by using a reverse contact, this function block goes to the next state which is

"driveConvey1" state using set coil and reset coil for reset "IDLE" state. In this state, what it does is just drive a true signal as an output of this function block to the main program to drive the conveyor 1 by using direct coil. When the object reaches the second photo sensor "photo02" by using a reverse contact, this function block will enter the next state which is a ready to stop state name "stopConvey1" by using set coil and reset "driveConvey1" state by reset coil. This state, basically run 2s more and stop; we used TON function block to be able to achieve this working behavior. Together when we set this function block to the IDLE state again by set coil and reset the previous state by reset coil, we will set an output from this function block "push1" to drive the next function block that will come later.

We need to keep in mind that the photo sensor works in reverse logic manner, therefore we always use a reverse contact for the photo sensor and before moving to the next state, we always need to make sure that the current state is running, therefore, we also have a direct contact for the current state before change to the next state. And to have a consistency program, for a particular function block, we implemented in the way that only 1 state can be active at the time.

FB2 FUNCTION BLOCK:

This function block is responsible for pushing the object to the next conveyor. For this function block we will divide into 3 states:

1. idle (does nothing, just be stand-by for operate)
2. runf (the pusher moves in forward direction)
3. runb (the pusher moves in backward direction)

Here we also created a state machine diagram for this function block.

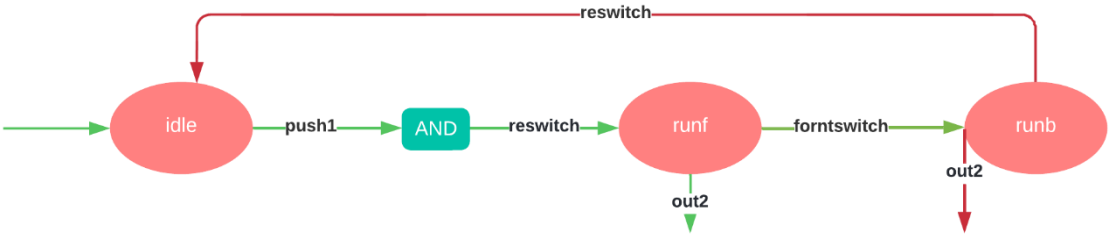


Figure 4: State Machine diagram for FB2

Based on the State machine diagram, a following output mapping table is created as below.

State	idle	runf	runb
motor	0	1	1
fdir	0	1	0

Table 2: Output mapping table for FB2

Based on these state machine diagram "Figure 4" and the output mapping table "Table 2", we can create a ladder diagram as shown in the function block "Controller.Micro830.Micro830.FB2" below.

For this function block we will have an input and output as:

- push1 (an input connect to an output from FB1, take as a start signal)
- reswitch (an input connect to push1r, this is an address of the rear switch of the the pusher 1 in the manufacturing line model)
- forntswitch (an input connect to push1f, this is an address of the front switch of the pusher 1 in the manufacturing line model)

- out2 (an output, this will be a signal to drive the next function block "FB3")
- motor (an output connect to push1motor, this is an address of the motor to drive the pusher 1 of the manufacturing line model)
- fdir (an output connect to push1dir, this is an address of the direction control of the pusher 1 in the the manufacturing line model)

This function block initially is in the "idle" state and just wait for the input signal "push1" from the first function block "FB1" to start operates. When "push1" signal is received from "FB1" by using a direct contact, before going to the next state "runf" we also need to check if the rear switch is active; for this we also use a direct contact since they are not working in reverse logic manner like photo sensor. When this function block is in "runf" state, it should send the output of this function block "out2" to the next function block "FB3" to tell the next one to start operate but this output signal will be turned off when the pusher start to move backward for this we used a set coil to set the output of this function block, another 2 outputs "motor" and "fdir" also need to be sent to the main program to control the milling machine. When the pusher reach to the front switch "forntswitch", at this point we will set the output signal "out2" back to low again by the reset coil and then jump to the next state "runb". In this state the pusher move backward until it reach the rear switch which can be checked by direct contact then this function block can move to the "idle" state again by set coil and reset coil for the "runb" state. During the "runf" and "runb" states this function block drive a signal to the motor by direct coil. "runf" state will drive a high level signal for "fdir" which means move forward by direct coil, while "runb" state will drive a low level signal for "fdir" which means move backward by reverse coil.

FB3 FUNCTION BLOCK:

This function block "FB3" is responsible for the milling machine (**in the CCW, we might name some variable wrong to drilling machine but we in this function block "FB3" we meant milling machine. We apologize for that**). For this function block "FB3" we divide it into 3 states:

1. idle (does nothing, stay stand-by waiting for input signal to start the process)
2. conv2 (the conveyor 2 move until the photo sensor 3 is sensed)
3. drilling (the conveyor 2 stop and the milling machine should start operate for a given amount of time)
4. conv22 (the conveyor 2 move for 3 more seconds)

Here is the state machine diagram for this function block "FB3".

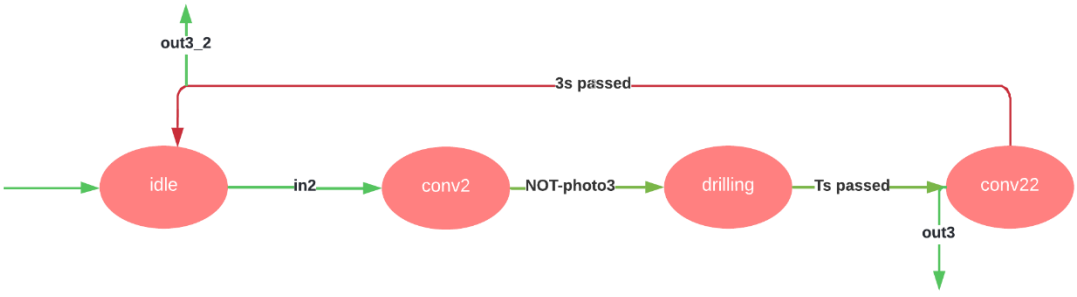


Figure 5: state machine diagram for FB3

Based on the State machine diagram, a following output mapping table is created as below.

State	idle	conv2	drilling	conv22
movcc	0	1	0	1
drillMac	0	0	1	0

Table 3: Output mapping table for FB3

Based on these state machine diagram “Figure 5” and the output mapping table “Table 3”, we can create a ladder diagram as shown in the function block “Controller.Micro830.Micro830.FB3” below.

For this function block we will have an input and output as:

- in2 (an input is connected to an output from previous function block “FB2”)
- inpuT (an input variable where we can specify the time of miilling)
- photo3 (an input connected to photo3, this is the address of the photo sensor 3 in the manufacturing line model)
- movcc (an output connected to conv2, this is the address of the conveyor 2 in the manufacturing line model)
- drillMac (an output connected to milmachine, this is the address of the milling machine in the manufacturing line model)

This function block initially is in “idle” state. Then when the start signal comes “in2” by the direct coil, this will move to the next state by set coil for “conv2” state and reset coil for “idle” state. When the photo sensor 3 is sense by reverse contact, this function block shall move to the next state by set coil for “drilling” state and reset coil for “conv2” state. In “drilling” state, we want it to operate for a given amount of time, in this case we use a TON function block “TON_2” which PT will be connected to “inpuT” from input of this function block “FB3”. After a given amount of time passed, this function block will move to the next state “conv22” by set coil and reset “drilling” state by reset coil. On this state, we want it to operate for exactly 3s, we can again use timer TON “TON_1” which specify PT for 3s; after 3s is passed, this function block will move to “idle” state again by set coil and reset coil for “conv22” state. The output “movcc” which will move the conveyor 2 will be active only on “conv2” state and “conv22” state, therefore, we can have a parallel of a direct contact of these two states and connect to “movcc” output by direct coil. “drillMac” output will be active only during “drilling” state, so we can have a direct contact of “drilling” to a direct coil of “drillMac”.

RESOURCE MICRO830

(* *)

Status: Readable, Modifiable, Deletable

The resource defines 45 variable(s).

VARIABLE _IO_EM_DO_00

(* *)
Direction: VarDirectlyRepresented
Alias: conv1
Data type: BOOL
Attribute: Read/Write
Direct variable (Channel): %QX0.0.0

VARIABLE _IO_EM_DO_01

(* *)
Direction: VarDirectlyRepresented
Alias: conv2

Data type: BOOL
Attribute: Read/Write
Direct variable (Channel): %QX0.0.1

VARIABLE _IO_EM_DO_02

(* *)
Direction: VarDirectlyRepresented
Alias: conv3
Data type: BOOL
Attribute: Read/Write
Direct variable (Channel): %QX0.0.2

VARIABLE _IO_EM_DO_03

(* *)
Direction: VarDirectlyRepresented
Alias: conv4
Data type: BOOL
Attribute: Read/Write
Direct variable (Channel): %QX0.0.3

VARIABLE _IO_EM_DO_04

(* *)
Direction: VarDirectlyRepresented
Alias: push1motor
Data type: BOOL
Attribute: Read/Write
Direct variable (Channel): %QX0.0.4

VARIABLE _IO_EM_DO_05

(* *)
Direction: VarDirectlyRepresented
Alias: push1dir
Data type: BOOL
Attribute: Read/Write
Direct variable (Channel): %QX0.0.5

VARIABLE _IO_EM_DO_06

(* *)
Direction: VarDirectlyRepresented
Alias: push2motor

Data type: BOOL
Attribute: Read/Write
Direct variable (Channel): %QX0.0.6

VARIABLE _IO_EM_DO_07

(* *)
Direction: VarDirectlyRepresented
Alias: pusher2dir
Data type: BOOL
Attribute: Read/Write
Direct variable (Channel): %QX0.0.7

VARIABLE _IO_EM_DO_08

(* *)
Direction: VarDirectlyRepresented
Alias: milmachine
Data type: BOOL
Attribute: Read/Write
Direct variable (Channel): %QX0.0.8

VARIABLE _IO_EM_DO_09

(* *)
Direction: VarDirectlyRepresented
Alias: drilmachine
Data type: BOOL
Attribute: Read/Write
Direct variable (Channel): %QX0.0.9

VARIABLE _IO_EM_DI_00

(* *)
Direction: VarDirectlyRepresented
Alias: photo1
Data type: BOOL
Attribute: Read
Direct variable (Channel): %IX0.1.0

VARIABLE _IO_EM_DI_01

(* *)
Direction: VarDirectlyRepresented
Alias: photo2

Data type: BOOL
Attribute: Read
Direct variable (Channel): %IX0.1.1

VARIABLE _IO_EM_DI_02

(* *)
Direction: VarDirectlyRepresented
Alias: photo3
Data type: BOOL
Attribute: Read
Direct variable (Channel): %IX0.1.2

VARIABLE _IO_EM_DI_03

(* *)
Direction: VarDirectlyRepresented
Alias: photo4
Data type: BOOL
Attribute: Read
Direct variable (Channel): %IX0.1.3

VARIABLE _IO_EM_DI_04

(* *)
Direction: VarDirectlyRepresented
Alias: photo5
Data type: BOOL
Attribute: Read
Direct variable (Channel): %IX0.1.4

VARIABLE _IO_EM_DI_05

(* *)
Direction: VarDirectlyRepresented
Alias: push1f
Data type: BOOL
Attribute: Read
Direct variable (Channel): %IX0.1.5

VARIABLE _IO_EM_DI_06

(* *)
Direction: VarDirectlyRepresented
Alias: push1r

Data type: BOOL
Attribute: Read
Direct variable (Channel): %IX0.1.6

VARIABLE _IO_EM_DI_07

(* *)
Direction: VarDirectlyRepresented
Alias: push2f
Data type: BOOL
Attribute: Read
Direct variable (Channel): %IX0.1.7

VARIABLE _IO_EM_DI_08

(* *)
Direction: VarDirectlyRepresented
Alias: push2r
Data type: BOOL
Attribute: Read
Direct variable (Channel): %IX0.1.8

VARIABLE _IO_EM_DI_09

(* *)
Direction: VarDirectlyRepresented
Data type: BOOL
Attribute: Read
Direct variable (Channel): %IX0.1.9

VARIABLE _IO_EM_DI_10

(* *)
Direction: VarDirectlyRepresented
Data type: BOOL
Attribute: Read
Direct variable (Channel): %IX0.1.10

VARIABLE _IO_EM_DI_11

(* *)
Direction: VarDirectlyRepresented
Data type: BOOL
Attribute: Read
Direct variable (Channel): %IX0.1.11

VARIABLE _IO_EM_DI_12

(* *)
Direction: VarDirectlyRepresented
Data type: BOOL
Attribute: Read
Direct variable (Channel): %IX0.1.12

VARIABLE _IO_EM_DI_13

(* *)
Direction: VarDirectlyRepresented
Data type: BOOL
Attribute: Read
Direct variable (Channel): %IX0.1.13

VARIABLE __SYSVA_CYCLECNT

(* Cycle counter *)
Direction: VarGlobal
Data type: DINT
Attribute: Read

VARIABLE __SYSVA_CYCLEDATE

(* Timestamp of the beginning of the cycle in milliseconds (ms) *)
Direction: VarGlobal
Data type: TIME
Attribute: Read

VARIABLE __SYSVA_KVBPERR

(* Kernel variable binding producing error (production error) *)
Direction: VarGlobal
Data type: BOOL
Attribute: Read

VARIABLE __SYSVA_KVBCERR

(* Kernel variable binding consuming error (consumption error) *)
Direction: VarGlobal
Data type: BOOL
Attribute: Read/Write

VARIABLE __SYSVA_RESNAME

(* Resource name (max length=255) *)
Direction: VarGlobal
Data type: STRING
Attribute: Read

VARIABLE __SYSVA_SCANCNT

(* Input scan counter *)
Direction: VarGlobal
Data type: DINT
Attribute: Read

VARIABLE __SYSVA_TCYCYCTIME

(* Programmed cycle time *)
Direction: VarGlobal
Data type: TIME
Attribute: Read/Write

VARIABLE __SYSVA_TCYCURRENT

(* Current cycle time *)
Direction: VarGlobal
Data type: TIME
Attribute: Read

VARIABLE __SYSVA_TCYMAXIMUM

(* Maximum cycle time since last start *)
Direction: VarGlobal
Data type: TIME
Attribute: Read

VARIABLE __SYSVA_TCYOVERFLOW

(* Number of cycle overflows *)
Direction: VarGlobal
Data type: DINT
Attribute: Read

VARIABLE __SYSVA_RESMODE

(* Resource execution mode *)

Direction: VarGlobal
Data type: SINT
Attribute: Read

VARIABLE __SYSVA_CCEXEC

(* Execute one cycle when application is in cycle to cycle mode *)
Direction: VarGlobal
Data type: BOOL
Attribute: Read/Write

VARIABLE __SYSVA_REMOTE

(* Remote status *)
Direction: VarGlobal
Data type: BOOL
Attribute: Read

VARIABLE __SYSVA_SUSPEND_ID

(* Last Suspend ID *)
Direction: VarGlobal
Data type: UINT
Attribute: Read

VARIABLE __SYSVA_TCYWDG

(* Software Watchdog *)
Direction: VarGlobal
Data type: UDINT
Attribute: Read/Write

VARIABLE __SYSVA_MAJ_ERR_HALT

(* Major Error Halted status *)
Direction: VarGlobal
Data type: BOOL
Attribute: Read

VARIABLE __SYSVA_ABORT_CYCLE

(* Aborting Cycle *)
Direction: VarGlobal
Data type: BOOL
Attribute: Read

VARIABLE __SYSVA_FIRST_SCAN

(* First scan bit *)
Direction: VarGlobal
Data type: BOOL
Attribute: Read

VARIABLE __SYSVA_USER_DATA_LOST

(* User data lost *)
Direction: VarGlobal
Data type: BOOL
Attribute: Read/Write

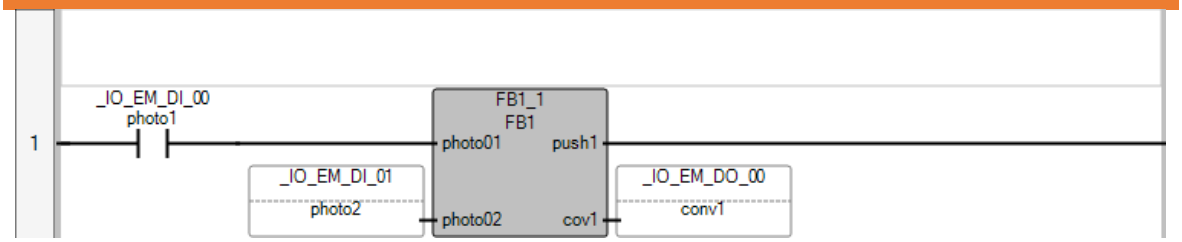
VARIABLE __SYSVA_POWERUP_BIT

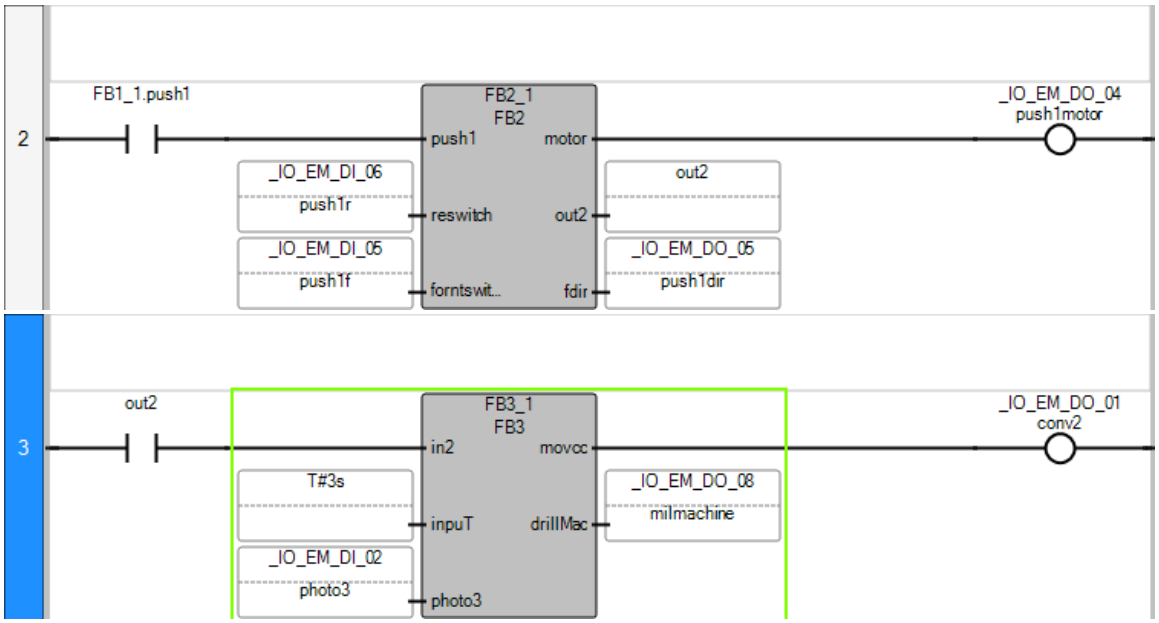
(* Power-up bit *)
Direction: VarGlobal
Data type: BOOL
Attribute: Read

VARIABLE __SYSVA_PROJ_INCOMPLETE

(* Project Incomplete *)
Direction: VarGlobal
Data type: UDINT
Attribute: Read

CONTROLLER.MICRO830.MICRO830.PROG1





POU PROG1

The POU defines 4 variable(s).

VARIABLE FB1_1

(* *)

Direction: Var
Data type: FB1
Attribute: Read/Write

VARIABLE FB2_1

(* *)

Direction: Var
Data type: FB2
Attribute: Read/Write

VARIABLE OUT2

(* *)

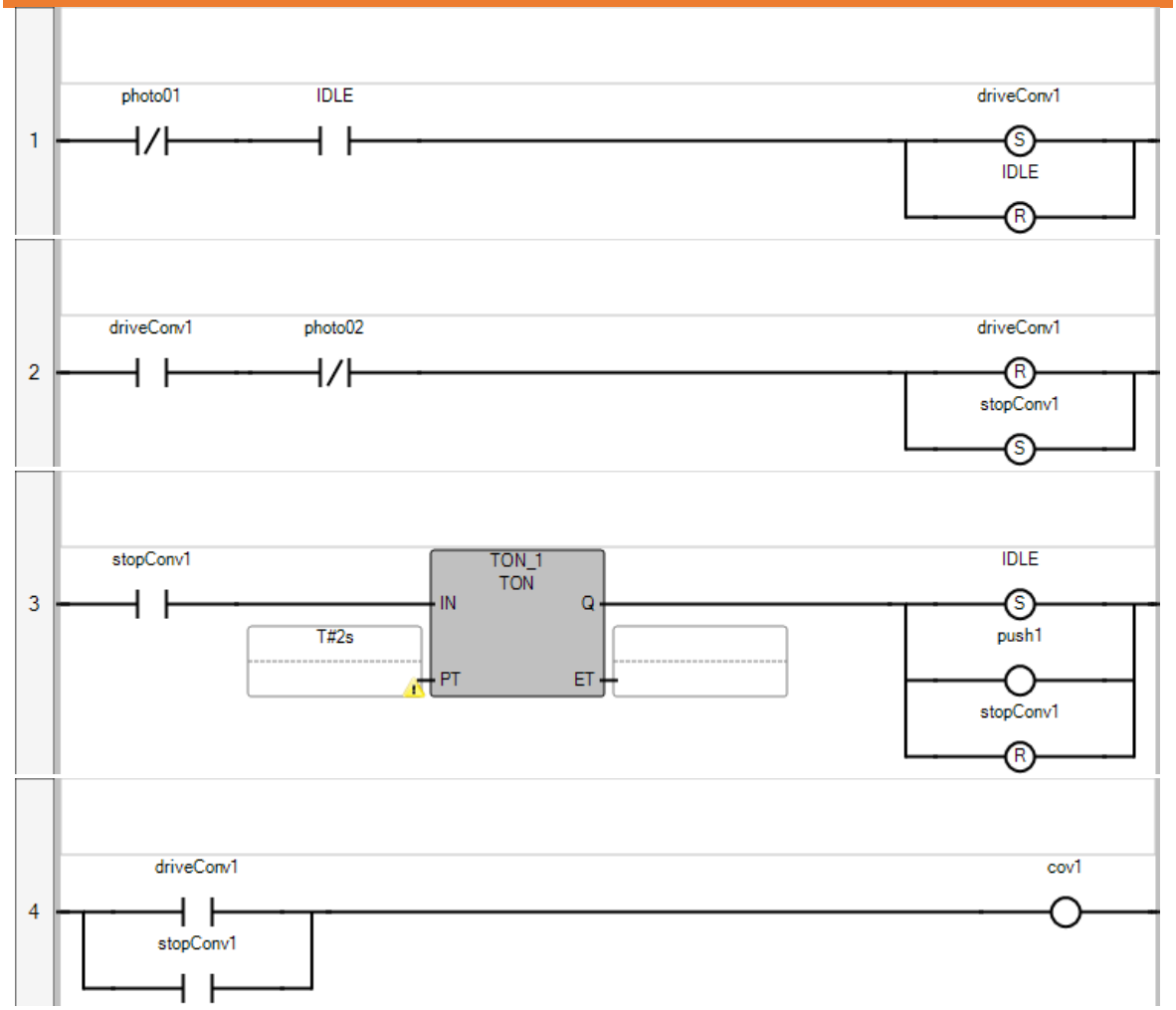
Direction: Var
Data type: BOOL
Attribute: Read/Write

VARIABLE FB3_1

(* *)

Direction: Var
Data type: FB3
Attribute: Read/Write

CONTROLLER.MICRO830.MICRO830.FB1



POU FB1

The POU defines 8 variable(s).

VARIABLE PHOTO01

(* *)
Direction: VarInput
Data type: BOOL
Attribute: Read

VARIABLE PHOTO02

(* *)

Direction: VarInput
Data type: BOOL
Attribute: Read

VARIABLE PUSH1

(* *)

Direction: VarOutput
Data type: BOOL
Attribute: Write

VARIABLE COV1

(* *)

Direction: VarOutput
Data type: BOOL
Attribute: Write

VARIABLE IDLE

(* *)

Direction: Var
Data type: BOOL
Attribute: Read/Write

VARIABLE DRIVECONV1

(* *)

Direction: Var
Data type: BOOL
Attribute: Read/Write

VARIABLE STOPCONV1

(* *)

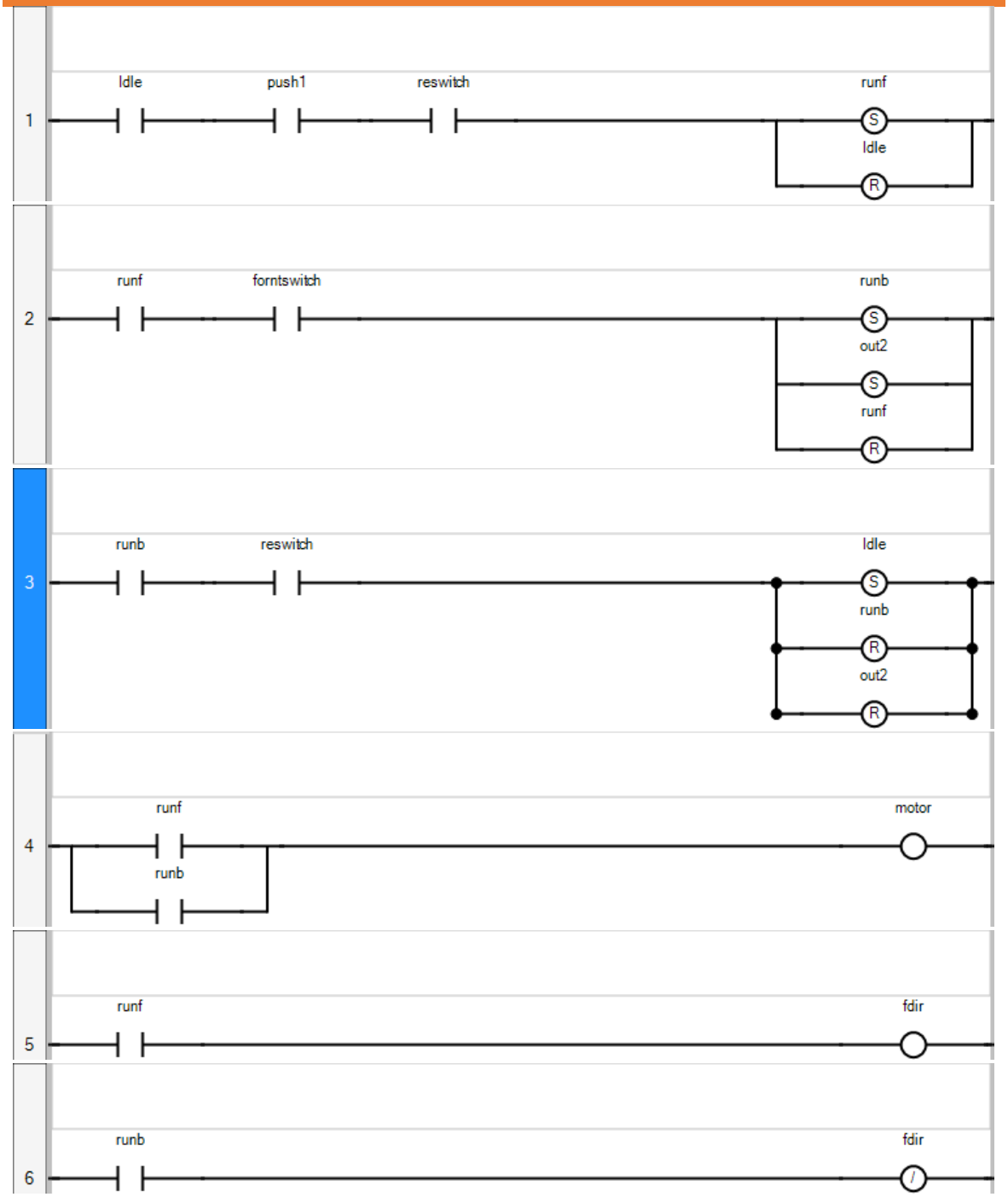
Direction: Var
Data type: BOOL
Attribute: Read/Write

VARIABLE TON_1

(* *)

Direction: Var
Data type: TON
Attribute: Read/Write

CONTROLLER.MICRO830.MICRO830.FB2



POU FB2

The POU defines 9 variable(s).

VARIABLE PUSH1

(* *)

Direction: VarInput
Data type: BOOL
Attribute: Read

VARIABLE RESWITCH

(* *)

Direction: VarInput
Data type: BOOL
Attribute: Read

VARIABLE FORNTSWITCH

(* *)

Direction: VarInput
Data type: BOOL
Attribute: Read

VARIABLE MOTOR

(* *)

Direction: VarOutput
Data type: BOOL
Attribute: Write

VARIABLE OUT2

(* *)

Direction: VarOutput
Data type: BOOL
Attribute: Write

VARIABLE FDIR

(* *)

Direction: VarOutput
Data type: BOOL
Attribute: Write

VARIABLE IDLE

(* *)

Direction: Var
Data type: BOOL
Attribute: Read/Write

VARIABLE RUNF

(* *)

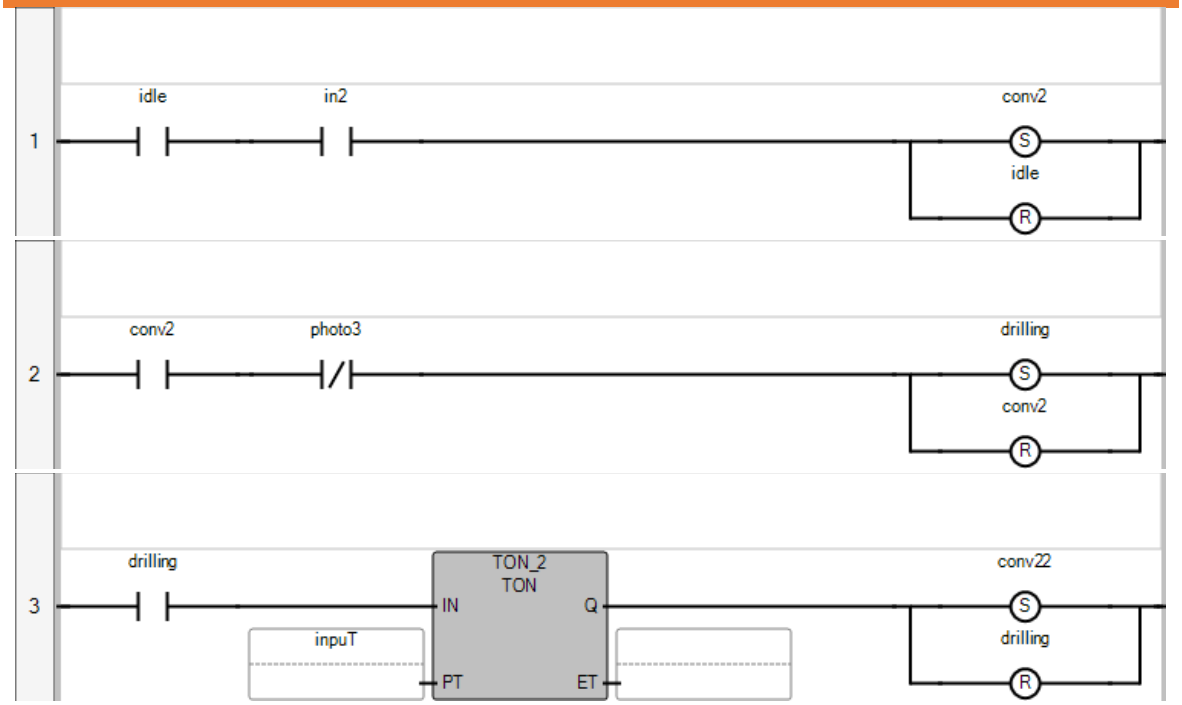
Direction: Var
Data type: BOOL
Attribute: Read/Write

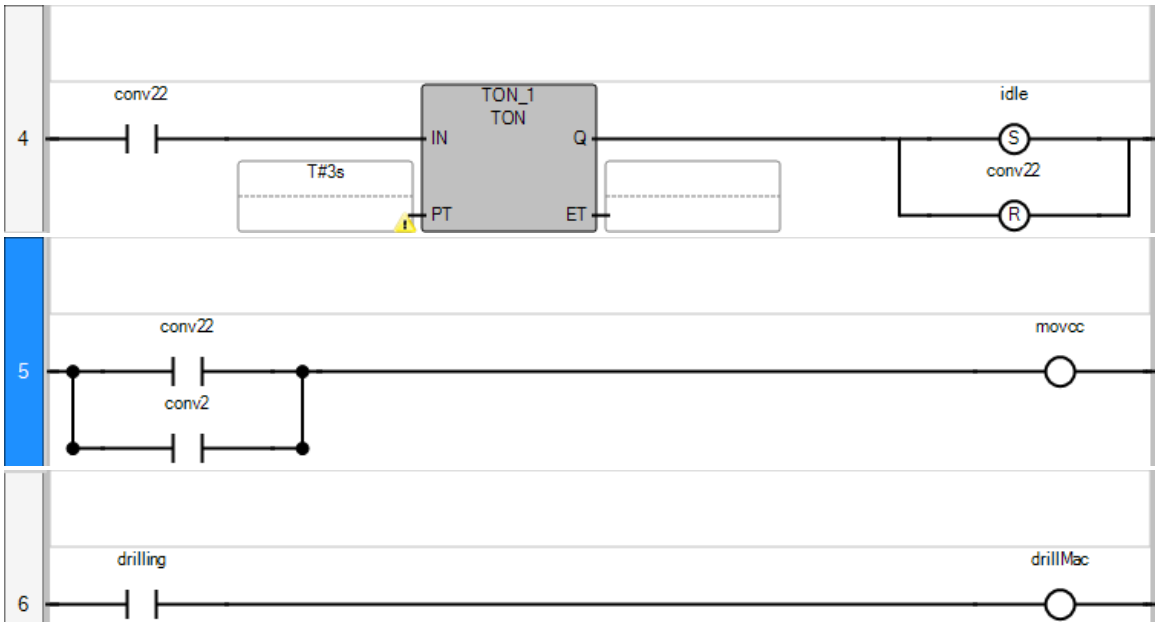
VARIABLE RUNB

(* *)

Direction: Var
Data type: BOOL
Attribute: Read/Write

CONTROLLER.MICRO830.MICRO830.FB3





POU FB3

The POU defines 11 variable(s).

VARIABLE IN2

(* *)

Direction: VarInput
Data type: BOOL
Attribute: Read

VARIABLE INPUT

(* *)

Direction: VarInput
Data type: TIME
Attribute: Read

VARIABLE PHOTO3

(* *)

Direction: VarInput
Data type: BOOL
Attribute: Read

VARIABLE MOVCC

(* *)

Direction: VarOutput
Data type: BOOL
Attribute: Write

VARIABLE DRILLMAC

(* *)
Direction: VarOutput
Data type: BOOL
Attribute: Write

VARIABLE IDLE

(* *)
Direction: Var
Data type: BOOL
Attribute: Read/Write

VARIABLE CONV2

(* *)
Direction: Var
Data type: BOOL
Attribute: Read/Write

VARIABLE DRILLING

(* *)
Direction: Var
Data type: BOOL
Attribute: Read/Write

VARIABLE CONV22

(* *)
Direction: Var
Data type: BOOL
Attribute: Read/Write

VARIABLE TON_1

(* *)
Direction: Var
Data type: TON
Attribute: Read/Write

VARIABLE TON_2

(* *)

Direction: Var

Data type: TON

Attribute: Read/Write