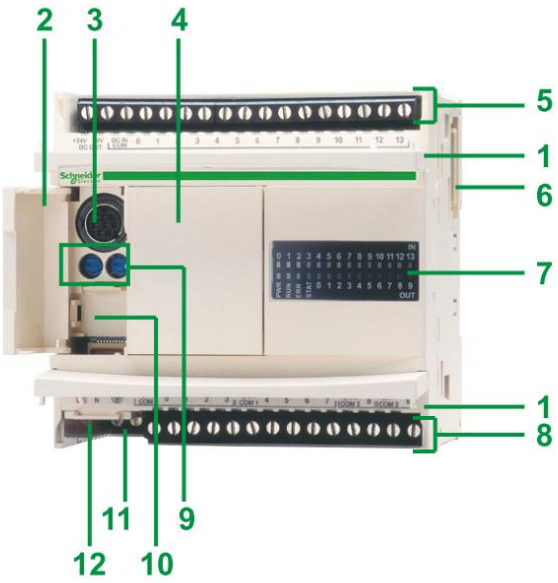
1 TWIDO controllers

TWIDO is the PLC family of Schneider Electric designed for low complexity applications, containing both modular and compact devices. During the measurement, different models of compact controllers will be used.

Being compact devices, PLCs used during the measurement enclose the power supply unit, the CPU and the IOs in the same housing. Parts of a compact TWIDO PLC are shown by Figure 1.



1 Covers of input and output connectors
2 Hinge door
3 RS485 connector (programming interface)
4 Place of operator display
5 Input connectors
6 Connector for extension modules (located at the right side)
7 Input, output and status LEDs
8 Output connectors
9 Setpoint potentiometers
10 Place of second serial port
11 Power connection
12 RTC / EEPROM expansion slot

Figure 1 – Parts of a compact TWIDO PLC

Depending on the model, PLCs might have 24V DC or 230V AC power inputs, from which the power supply unit generates lower voltage supply for the CPU and other modules.

The CPU is responsible for operating the PLC, running the operating system and the user program. It is equipped with 32kBytes of RAM, of which the program memory has the capacity of 3000 instructions. The RAM includes 256 bit registers and (depending on the use of function blocks) at most 3000 word registers. The CPU allows simultaneous use of 128 counters and 128 timers. The RAM is battery-powered to store its contents during power outage, but an external 32kByte EEPROM cartridge can also be used. The controller can also be equipped with a real-time clock (RTC) module.

Depending on the model, compact TWIDO PLCs might have up to 24 digital IOs with 0-24V voltage levels, decoupled from the CPU. Analogue setpoint potentiometers are also available in the base unit. PLCs in the lab have relay outputs, but models with solid-state digital outputs are also available both in sink and source configurations. States of inputs and outputs are displayed on the front panel LEDs. Further inputs and outputs (both digital and alalog ones) can be connected to the base unit in forms of expansion modules.

Every TWIDO PLC is equipped with an RS485 communication interface, serving both as programming interface and a MODBUS communication port for connection to other controllers. Other communication interfaces (ASI, CAN or Ethernet) are available as expansion modules.

2 Basics of ladder diagram programming

Among the PLC programming languages defined by the standard IEC-1131 (and later IEC-61131), the ladder diagram is the most popular one used in simple applications. In the followings, the basic concepts of ladder programming will be presented using the syntax of its TWIDO implementation.

2.1 Basic logic elements

As initially designed to mimic relay logics, ladder diagram consist of rungs, each implementing a logical function. In each scan cycle, the PLC reads the inputs, evaluates the logic functions from the top to the bottom, and updates the outputs. Basic logic elements of ladder diagrams are shown by Table 1.

Contacts		Coils	
—| |—	Normally open (NO) contact	—( )—	Normally opern (NO) coil
—|/|—	Normally closed (NC) contact	—(/)—	Normally closed (NC) coil
—|P|—	Rising edge sensitive contact	—(S)—	Set (latch) coil
—|N|—	Falling edge sensitive contact	—(R)—	Reset (unlatch) coil

Table 1 – Basic logic elements

Inputs of a logical function correspond to the contacts, while its outputs correspond to the coils. The current flow starts from the left side of the rung, and directed towards the right through the contacts and then the coils. The normally open (NO) contact can be interpreted as a simple pushbutton: it conducts if the value of the associated variable (written over the contact in the diagram) is true (1). The normally closed (NC) contact works the opposite way: it conducts if the corresponding variable is false (value 0), so it can be considered as a normally closed pushbutton. Serial interconnection of normally open contacts (resp. normally closed contacts) realize an AND (resp. NAND) operation, while their parallel interconnection implements an OR (resp. NOR) operation.

Normally open (NO) coils set the corresponding variable to 1 if energized, and set it to 0 otherwise. Similarly, normally close (NC, negated) coils set the corresponding variable to 1 if not energized.

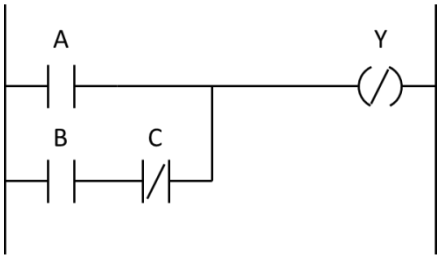


Figure 1 – Implementation of a logical function

The ladder diagram above implements the logical function $Y=\mathrm{NOT}(A \text{ OR } (B \text{ AND NOT } C))$. The top branch on the left conducts if A is true, while the bottom one conducts if B is true and C is false (B AND NOT C). The coil on the right is energized if either of the branches on the left conduct, i.e. if the condition A OR (B AND NOT C) is true. However, since the coil is a normally closed one, the variable C is set to the value $\mathrm{NOT}(A \text{ OR } (B \text{ AND NOT } C))$.

It is important to emphasise that the operation of the rung implements a logical function, which sets the output no matter its condition is satisfied or not. Beginners often make a mistake by associating the above ladder rung with the instruction `IF (A OR (B AND NOT C)) THEN (Y=0)` of high-level programming languages. However, this is not correct: the instruction above does not change the value of Y if the condition is not satisfied. On the other hand, the logical function implemented by the ladder rung sets the output to 1 in that case. Therefore, the logical function should be associated with the instruction `IF (A OR (B AND NOT C)) THEN (Y=0) ELSE (Y=1).`

Variables associated to the coils are set to the value corresponding to the part of the rung left from them in each scan cycle. If we would like to add memory-like behaviour to a variable, Set and Reset coils should be used. The Set coil sets the corresponding variable to 1 if energized, and does not change its value if not energized. Clearing the variable can be achieved by a Reset coil in a similar way.

Due to the cyclic behaviour of PLCs, the user program can not directly set the output bits. Instead, it sets the bits of the output map stored in the memory, which are copied to the physical outputs after the end of the program scan. Therefore, if an output bit is assigned by coils in multiple rungs, the last assignment will overwrite the results of the formers in the output map, so therefore those assignments do not appear at the physical output. It is strongly recommended to write each output only in one single rung of the diagram.

2.2 Operands – language objects

Operands used by different operations are referred to as language objects by the TwidoSuite development environment. These objects can be bit, word, double word or (in case of some PLC models) floating point language objects. During the measurement only bit and word objects will be used. Language objects, except to immediate values, are referenced by their identifier. The identifier starts with the `%` character, followed by the object type (1 or 2 characters) and the number of the object. In case of function blocks, the identifier is extended by the identifier of the function block variable the objects represents.

Bit object include digital inputs and outputs, memory bits and function block outputs. Inputs, outputs and memory bit objects are identified by the characters I, Q and M, respectively. Addresses of inputs and outputs are composed of two numbers delimited by a dot (**.**). The first number corresponds to the module the given port is located at (the base unit, used during the measurement, has the number 0), while the second is the number of the given input or output inside the module. In case of memory bits, the address is one single number from 0 to 255.

Word objects include analog inputs and outputs (`IW` and `QW`), memory words (`MW`) and constants (`KW`), and some function block variables. Referencing input, output and memory word objects is the same as referencing their bit object counterparts.

Variables of function blocks (see next section) can be referenced by adding a dot (**.**) and the variable identifier to the identifier of the function block object (like when referencing a property of an object).

Language objects used during the measurement are shown by Table 2.

Language object	Description	Example
%Ix.y	Digital input y of module x	%I0.4 – Digital input 4 of the base unit
%Qx.y	Digital output y of module x	%Q0.0 – Digital output 0 of the base unit
%Mi	Memory bit number i	%M4 – Memory bit number 4
%MWi	Memory word number i	%MW3 – Memory word number 3
%KWi	Word constant number i	%KW3 – Word constant number 3
%Ci	Counter number i	%C2.V – Value of the counter number 2
%TMi	Timer number i	%TM7.Q – Output of the counter number 7

Table 2 – Language objects

2.3 Basic function blocks

Beside simple logic elements, more sophisticated blocks, the so-called function blocks can also be used in ladder diagram. The most important timer, counter, compare and operation function blocks of TWIDO will be presented in the sequel.

Timers

TWIDO compact PLCs allow the simultaneous use of 128 timer blocks (%TM0…%TM127) which might be of different types (TON, TON, TP). Output Q of the timer block is a delayed copy of the input IN and its actual behaviour depends on the type of the counter. The time delay can be set by the time base (base) and preset parameters. The time base can be 1ms, 10ms, 100ms or 1 min (pay attention that 1 minute is the default setting), and the delay can be expressed as the multiple of the time base defined by the preset value (e.g. using 10ms time base and a preset value of 30 will result in a delay of 300ms): $\tau = t_{base} * n_{preset}$.

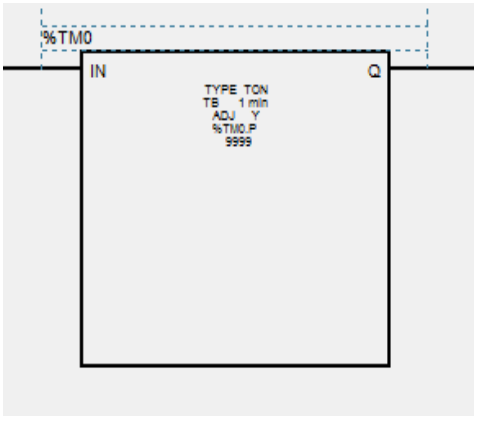


Figure 2 – The Timer function block

The output of the timer (%TMi.Q, read-only), the preset value (%TMi.P) and the current value of the counter (%TMi.V, read-only) can be addressed from the user program.

The On-delay timer (TON) delays the rising of the input IN by a configurable delay, if the input was active during the whole time. The output of the timer will reset immediately upon the input becomes 0 (see Figure 3).

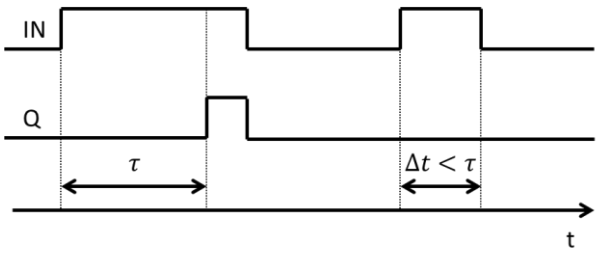


Figure 3 – Timing diagram of a TON-type timer

The output of the Off-delay timer (TOF) follows immediately the rising of the input, but it delays the falling edge of it by a configurable amount of time. The output of the timer is reset only if the input was constantly 0 during the delay (see Figure 4).

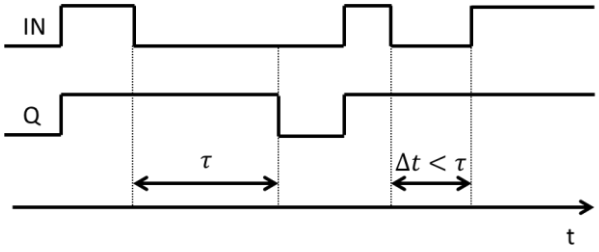


Figure 4- Timing diagram of a TOF-type timer

The output of the Pulse timer (TP) function block is set to 1 upon a rising edge of the input for the predefined delay and then returns to 0, regardless the change of the input value (see Figure 5).

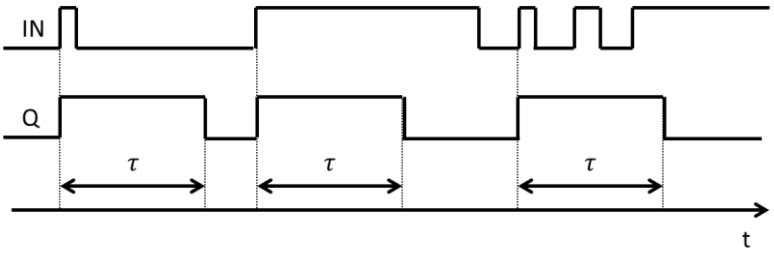


Figure 5 - Timing diagram of a TP-type timer

Other PLCs might implement other types of timers (e.g. retentive ones) also.

Counters

Compact TWIDO PLCs allow the simultaneous use of 128 counter blocks (`%C0…%C127`). These counters are bidirectional, resettable and a preset value can be defined to each of them.

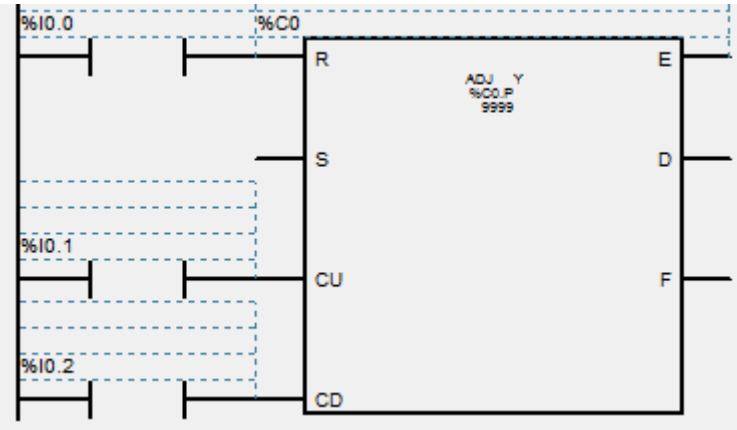


Figure 6 – The Counter function block

The value 1 of the R (Reset) input sets the current value of the counter to 0, while the value 1 of the S (Set) input sets it to the predefined preset value. Rising edges on the CU and CD inputs increment and decrement the counter value, respectively.

The current value of the counter can be in the range of 0 to 9999. Overflow (9999 → 0) is signalled by the value 1 of the output F (Full), while underflow (0 → 9999) is signalled by the output E (Empty). The value of the output D (Done) is set to 1 if the current value equals the preset value. These outputs can be referenced as language objects, e.g. as %Ci.D.

However the current value and preset value of the counter do not appear as outputs (as being not logical values), they can be accessed through compare and operation function blocks using the language objects %Ci.V and %Ci.P, respectively. Note that the current value of the counter is read-only.

The Compare function block

Non-logical operators (e.g. memory words or counter values) can be compared to each other using a compare function block.

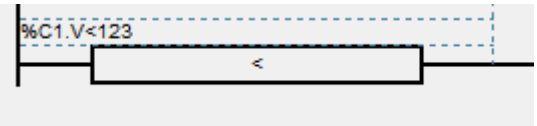


Figure 7 – The Compare function block

The block is evaluated only if energized, i.e. the condition to its left evaluates to true. The comparison operation can be <, <=, =, >, >= or <> while the operators can be word or double word objects, including immediate values (see Figure 7).

The Operate function block

Non-logical assignments and arithmetic operations can be carried out using an Operate function block. The block is evaluated only if energized, i.e. the condition to its left evaluates to true: in that case, the operation defined above the block is executed.

Assignment is represented by the operator :=. The left side of the operator can be any non-read-only language object, while any language object or immediate value can be present at the right side.

Right side might also include an arithmetic formula using addition (+), subtraction (−), multiplication (*), division (/), remainder (REM), square root (SQRT), increment (INC), decrement (DEC) or absolute value (ABS) operations, depending on the data types of the operands.

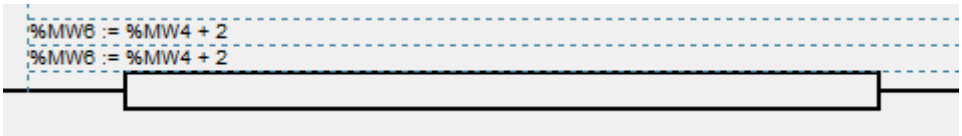


Figure 8 – The Operate function block

2.4 State machines

Operation of controlled processes can often be described by finite state machines, which are then implemented by a ladder diagram. Generally Moore-model automata are used, i.e. the outputs depend only on the current state.

As an illustrative example, consider the control of a simple heating device: the equipment can be turned on by pressing a pushbutton. In that case, only one of the two heaters is turned on (LOW mode). Upon a second press of the button, the other heater is also turned on (HIGH mode). The third press of the button turns off the equipment. The state transition diagram of the process is illustrated by Figure 9.

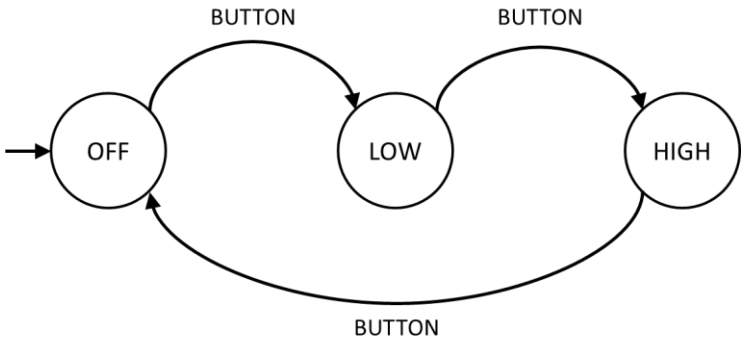


Figure 9 – State transition diagram of the heating device

Implementation of state machines can be carried out in three simple steps:

1. definition of the transitions
2. definition of the output mapping
3. initialization of the state machine

At first, a memory bit (the state register) should be assigned to each state. The value of a state register is 1 if and only if the corresponding state of the state machine is active, otherwise it is 0. In the sequel the registers assigned to the three states of the heating device will be referred to as S_OFF, S_LOW and S_HIGH.

Implementation of a state machine means the definition of the mapping which gives the next state as the function of the current state and the conditions of the transitions. Therefore, in order to execute a transition, it has to be evaluated whether the initial state of the transition is the current state and whether the condition of the transition is met. If both evaluate to true, the current state has to be replaced to the destination state of the given transition, i.e. the state register bit of the

current state has to be set to 0 and the state register bit of the destination state has to be set to 1. The rung for implementing a generalized transition is shown by Figure 10.

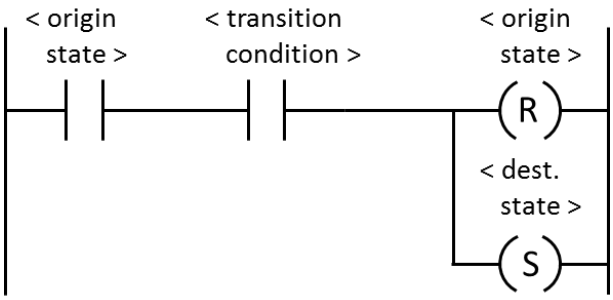


Figure 10 – General template for implementing state transitions

Here `<origin state>` and `<dest. state>` are the state registers associated to the origin and destination state of the transition, respectively. The condition of the transition is modelled here by a simple contact, but in general, any logical function can be plugged in there.

The ladder diagram describing the state transitions of the heating device is given by Figure 11.

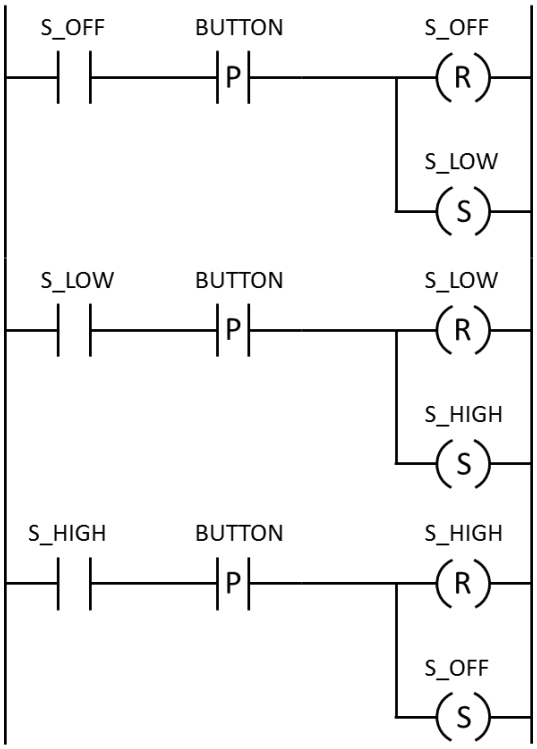


Figure 11 – Ladder diagram implementing the state transitions of the heating device

When implementing the output mapping, the rule of writing an output only in one single rung has to be respected. In case of state machines with multiple states, it is useful to make a table containing the state of each output in the different states. Let us denote the outputs of the heating device by the symbolic names HEATER1 and HEATER2 (corresponding the on/off state of the two heaters). Then the table of the output mapping is given by Table 3 – Output mapping of the .

State	S_OFF	S_LOW	S_HIGH
HEATER1	0	1	1
HEATER2	0	0	1

Table 3 – Output mapping of the heating device

Based on the table, the output mapping can easily be implemented as illustrated by Figure 12.

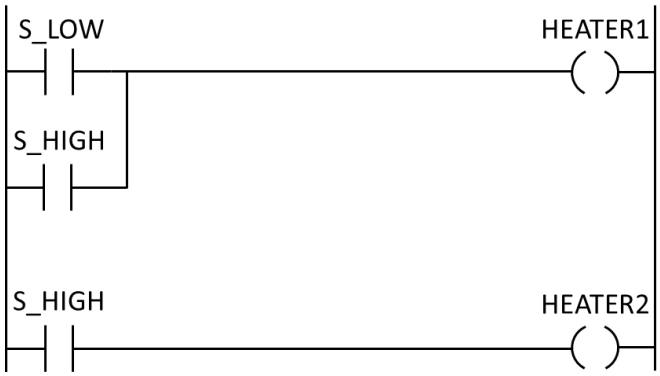


Figure 12 – Ladder diagram implementing the output mapping of the heating device

However the implementation of the transitions can assure that no more than one state can be active simultaneously, each state register bit is set to its default value 0 upon starting the PLC, i.e. there will be no active state. In order to start the state machine from its initial state, the corresponding state register bit has to be set to 1.

One solution, which is platform-independent, i.e. can be used in any environment, is to set the state register of the initial state to 1 if all state registers are of value 0 (see Figure 13).

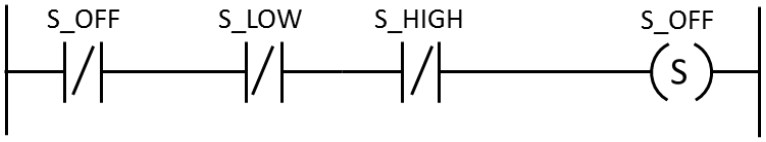


Figure 13 – General solution to initialize the state machine

A more sophisticated (and, especially in case of a high number of states, more convenient) solution is to use the current state of the PLC as condition for setting the initial state of the state machine. PLCs usually use a system bit or system word to notify the user program that the current scan cycle is the first one after startup, so this information can be used to initialize the state machine. TWIDO PLCs set the system bit `%S13` to 1 during the first scan, so the initialization of the state machine can be implemented as shown by Figure 14.

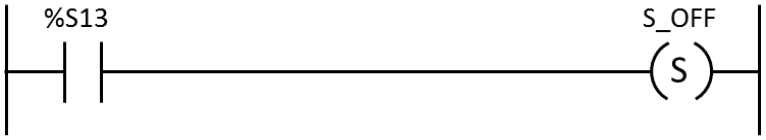


Figure 14 – Initialization of the state machine on a TWIDO PLC