Chapter 7 TIMERS, COUNTERS and T/C APPLICATIONS

Introduction

Timers and counters are discussed in the same chapter since most rules apply to both. Timers and counters have been in existence for as long as relays and provide an important component in the development of logic. Timers were constructed in the past as an add-on device to relays slowing down the transition of the plunger from immediately opening or closing. The time delay was accomplished with a pneumatic bladder that allowed the air to escape either quickly or slowly depending on the setting of the timer. Quick was usually less than a second and slow was usually between 30 and 60 seconds. Setting this kind of timer was an inexact science and today's traffic lights are an example of the fickle nature of timers that seldom respond in exactly the same from day to day and year to year.

For the first time, function blocks are introduced in the rung output position or coil position to provide timer and counter functions. Function blocks allow inputs from the left and pass power through to the right when the function is done or when various conditions are met. Either the timer has timed out or the counter has counted to the preset. Function block usage differs from manufacturer to manufacturer. Function blocks rely on a standard format to enter information about the counter or timer. All variables in the function block must be entered correctly before the device will operate.

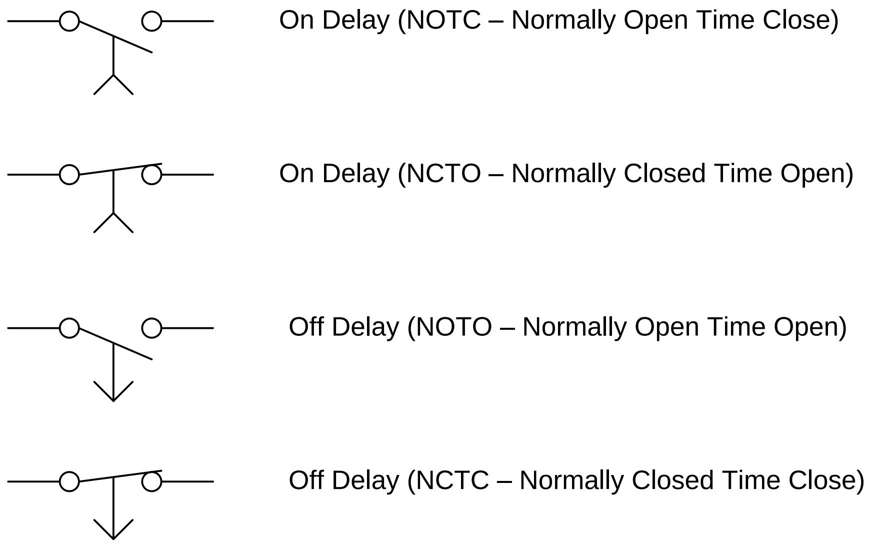
Some timers are referred to as retentive. Retentive refers to the device's ability to remember its exact status such that when the circuit is again activated, the timer continues from the previous point. Non-retentive timers reset to zero and start from zero each time the timer function block is energized. Retentive is similar to blowing up a balloon. One does not blow a balloon up with one blast of air. It takes quite a few. The retentive balloon has a finger along the neck of the balloon holding the air already blown in captive. When more air is blown in, the new air is added to the air already present. Many processes in the factory rely on logic needing this kind of physical property to control a machine.

Other terms used in the timer and counter blocks are "preset" and "accumulated". These words refer to the preset or target amount and the "accumulated" amount that the timer or counter has built to get to a preset. Times are really counts stored as integer numbers. Thus, counters and timers are very similar. Timers increment a number regularly each time period (usually in increments of 1 msec.).

Timers

Timers are used to provide logic when a circuit turns on or off. Traditional pneumatic timers were provided as either on-delay timers or off-delay timers. Contacts were provided both normally open and normally closed for each type of timer. The timer head was chosen as either the on-delay type or off-delay type. PLCs allow for a quick change from one type to the other with a few keystrokes on the programming panel.
Symbols for Timers:

The following symbols are used for pneumatic timer contacts:


On Delay (NOTC – Normally Open Time Close)
On Delay (NCTO – Normally Closed Time Open)
Off Delay (NOTO – Normally Open Time Open)
Off Delay (NCTC – Normally Closed Time Close)

Coils for pneumatic timers are drawn similar to relay coils except that TD is usually included in the label. TD refers to time delay.

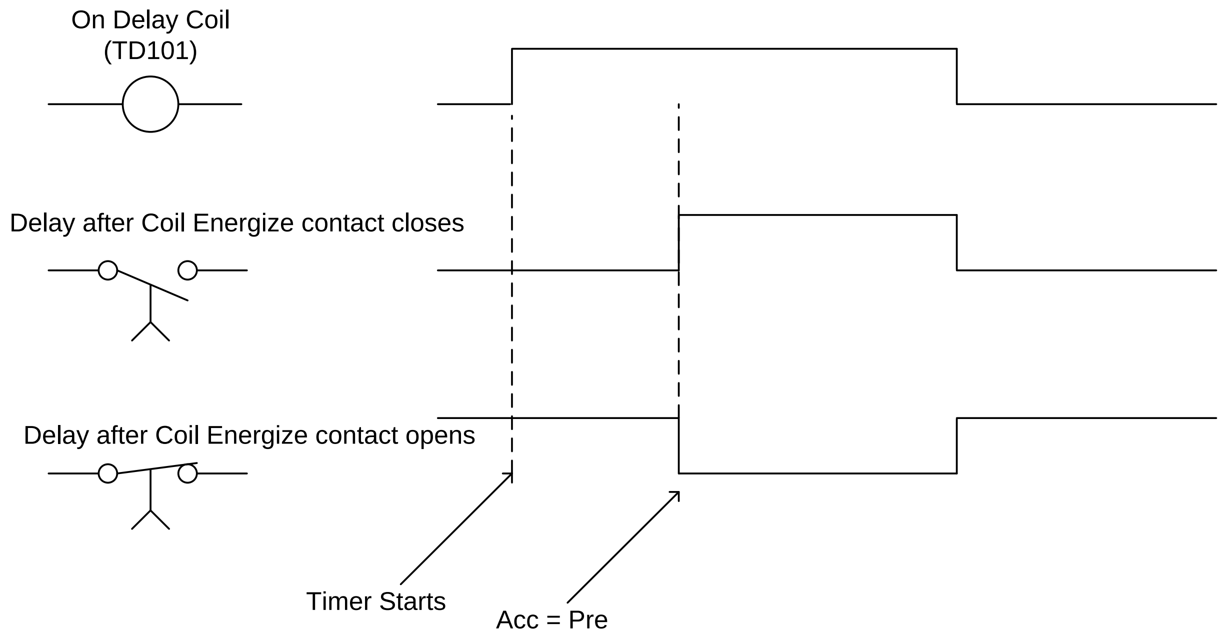

On Delay Coil
(TD101)
Delay after Coil Energize contact closes
Delay after Coil Energize contact opens
Timer Starts
Acc = Pre

Fig. 7-1 Pneumatic On Delay Timer Symbols and Timing Diagrams

Fig. 7-2 Pneumatic Delay Timer from Allen-Bradley

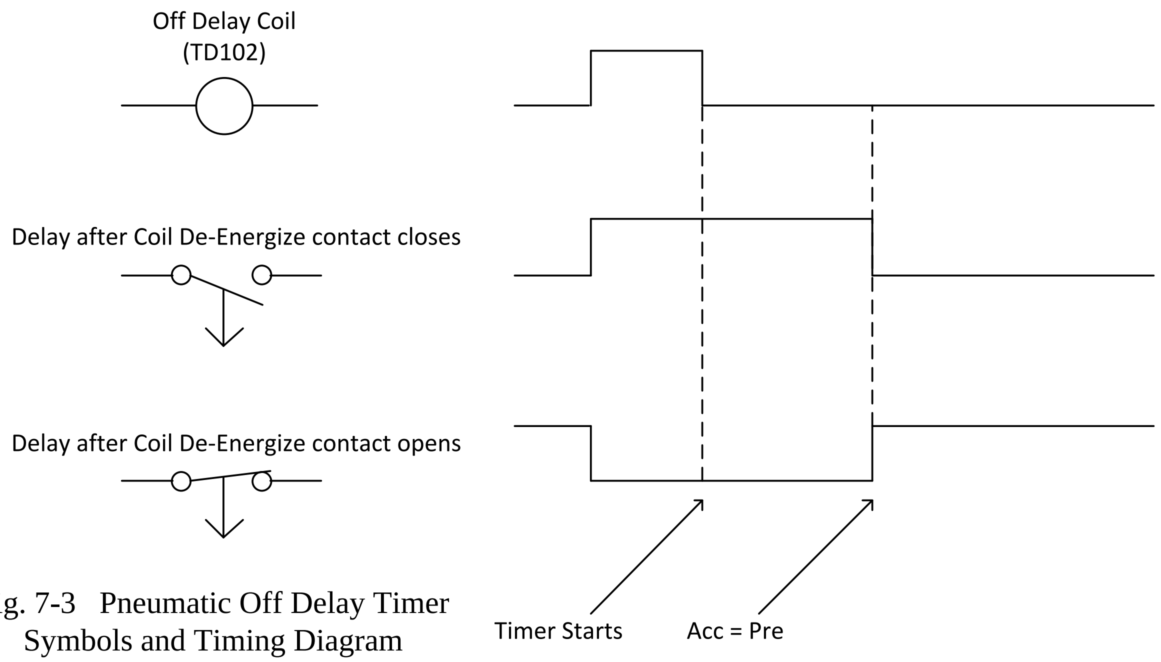

Off Delay Coil
(TD102)
Delay after Coil De-Energize contact closes
Delay after Coil De-Energize contact opens
Timer Starts
Acc = Pre

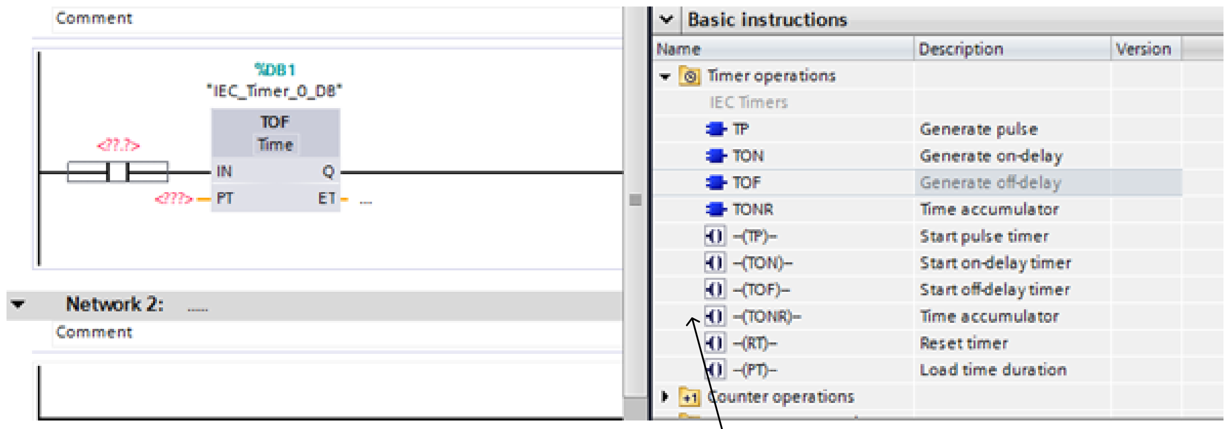
Fig. 7-3 Pneumatic Off Delay Timer Symbols and Timing Diagram

While these timers are only a sampling of the types of different timers, their function describes the main function of all timers, a time delay. While PLC vendors do not need to use the terms of on-delay or off-delay, normally closed, normally open, held closed, or held open, these terms are an important part of design of PLC circuits. Some vendors still use the terms to show linkage between the PLC and the original timer circuits.

Allen-Bradley provides three timers; TON, TOF, and RTO. All are block-type instructions and are located at the extreme right of each rung used. They are parallel to coils but may not be used in series with each other or in parallel with coils. Each has two coils extending from its right. These coils are not programmed separately. These coils appear when the timer function block is programmed.

Siemens Timers

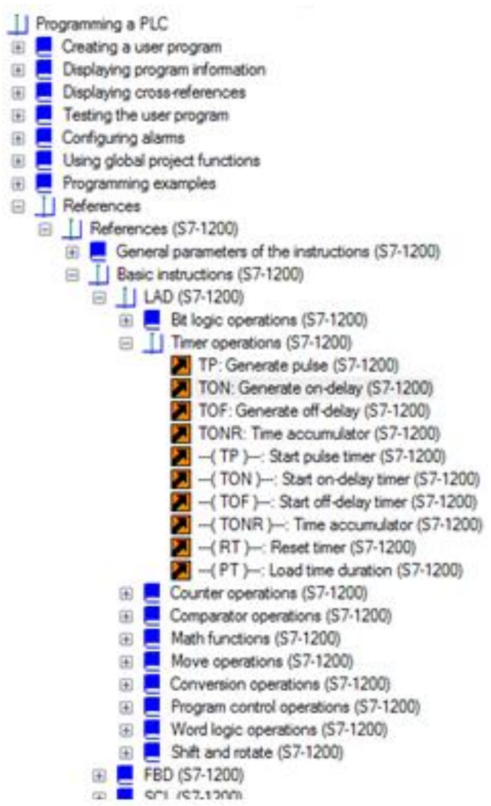
First, we look at the Siemens Timer block and discuss the various types of timers available.



Basic instructions

Name	Description	Version
Timer operations		
IEC Timers		
TP	Generate pulse	
TON	Generate on-delay	
TOF	Generate off-delay	
TONR	Time accumulator	
-(TP)-	Start pulse timer	
-(TON)-	Start on-delay timer	
-(TOF)-	Start off-delay timer	
-(TONR)-	Time accumulator	
-(RT)-	Reset timer	
-(PT)-	Load time duration	
Counter operations		

Timer Operations from Instruction List



```
Programming a PLC
  Creating a user program
  Displaying program information
  Displaying cross-references
  Testing the user program
  Configuring alarms
  Using global project functions
  Programming examples
  References
    References (S7-1200)
      General parameters of the instructions (S7-1200)
      Basic instructions (S7-1200)
        LAD (S7-1200)
          Bit logic operations (S7-1200)
          Timer operations (S7-1200)
            TP: Generate pulse (S7-1200)
            TON: Generate on-delay (S7-1200)
            TOF: Generate off-delay (S7-1200)
            TONR: Time accumulator (S7-1200)
            -( TP )-: Start pulse timer (S7-1200)
            -( TON )-: Start on-delay timer (S7-1200)
            -( TOF )-: Start off-delay timer (S7-1200)
            -( TONR )-: Time accumulator (S7-1200)
            -( RT )-: Reset timer (S7-1200)
            -( PT )-: Load time duration (S7-1200)
          Counter operations (S7-1200)
          Comparator operations (S7-1200)
          Math functions (S7-1200)
          Move operations (S7-1200)
          Conversion operations (S7-1200)
          Program control operations (S7-1200)
          Word logic operations (S7-1200)
          Shift and rotate (S7-1200)
        FBD (S7-1200)
        SCL (S7-1200)
```

Fig. 7-4a From Basic Instructions: Timer Operations

Timer Operations from Help List

TP: Generate pulse

The instruction **Generate pulse** sets output Q for duration PT. The instruction is started when the result of logic operation (RLO) at input IN changes from 0 to 1 (positive signal edge). The programmed time PT begins when the instruction starts. Output Q is set for the duration PT, regardless of the subsequent course of the input signal. Even if a new positive signal edge is detected, the signal state at the output Q is not affected as long as the PT time is running.

The current time value can be queried at the ET output (Elapsed Time). The time value starts at T#0s and ends when the value of duration PT (Preset Time) is reached. If duration PT is reached and the signal state at input IN is 0, the ET output is reset.

Each call of the **Generate pulse** instruction must be assigned an IEC timer in which the instruction data is stored. An IEC timer is a structure of the data type IEC_TIMER or TP that you can declare as follows:

- Declaration of a data block of system data type IEC_TIMER (for example, IEC_TIMER_0_DB)
- Declaration as a local tag of the type TP in the Input, InOut or Static section of a block (for example, #IEC_TIMER_0_DB)

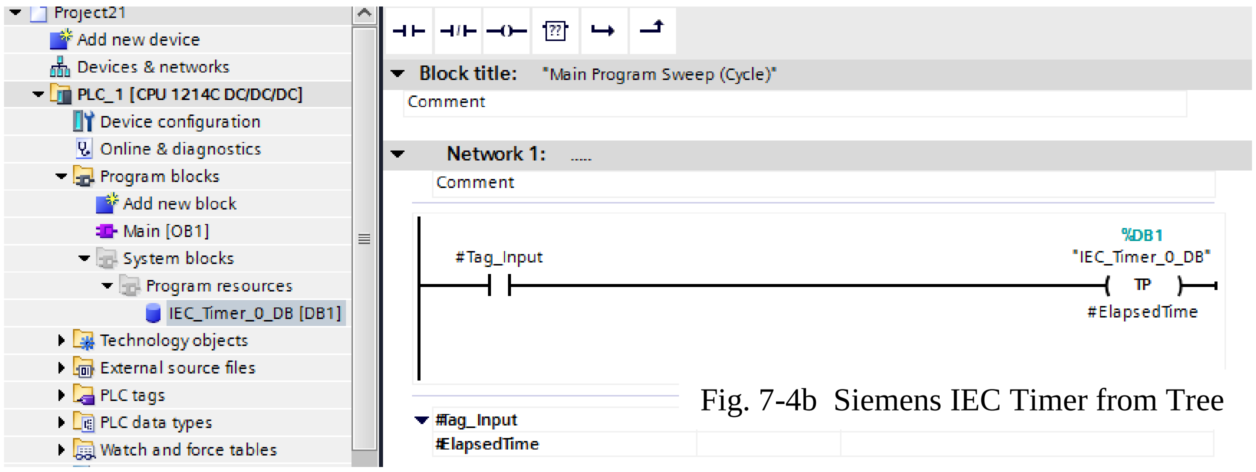


Fig. 7-4b Siemens IEC Timer from Tree

When you insert the instruction in the program, the Call options dialog opens in which you can specify whether the IEC timer is stored in its own data block (single instance) or as a local tag (multiple instance) in the block interface. If you create a separate data block, you will find this in the project tree in the Program resources folder under Program blocks > System blocks.

The execution of the **Generate pulse** instruction requires a preceding logic operation. It can be placed within or at the end of the network.

Parameters for the TP (**Generate pulse**) instruction:

Parameter	**Declaration**	**Data type**	**Memory Area**	**Description**
IN	Input	Bool	I,Q,M,D, L	Start input
PT	Input	TIME	I,Q,M,D,L or constant	Duration of the pulse. The value of PT parameter must be positive.
Q	Output	Bool	I,Q,M,D, L	Pulse output
ET	Output	Time	I,Q,M,D,L	Current time value

These figures show pulse diagrams of the Generate Pulse instruction:

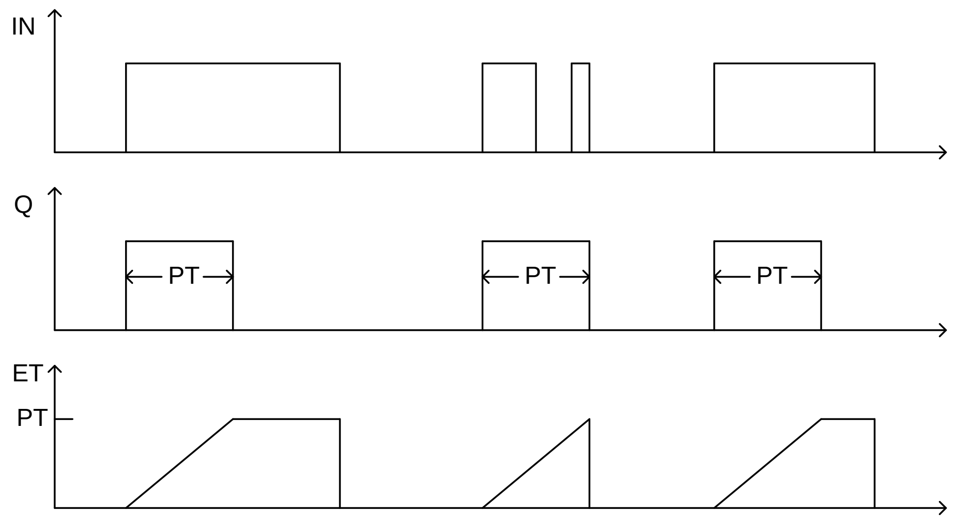


Fig. 7-5   Timing Diagrams for Generate Pulse Timer

TON: Generate on-delay

The instruction **Generate on-delay** delays setting of the output Q by the programmed duration PT. The instruction is started when the result of logic operation (RLO) at input IN changes from 0 to 1 (positive signal edge). The programmed time PT begins when the instruction starts. When the duration PT expires, the output Q has the signal state 1. Output Q remains set as long as the start input is still 1. When the signal state at the start input changes from 1 to 0, output Q is reset. The timer function is started again when a new positive signal edge is detected at the start input.

The current time value can be queried at the ET output. The time value starts at T#0s and ends when the value of duration PT is reached. The ET output is reset as soon as the signal state at the IN input changes to 0.

These figures show pulse diagrams of the Generate On-Delay instruction:

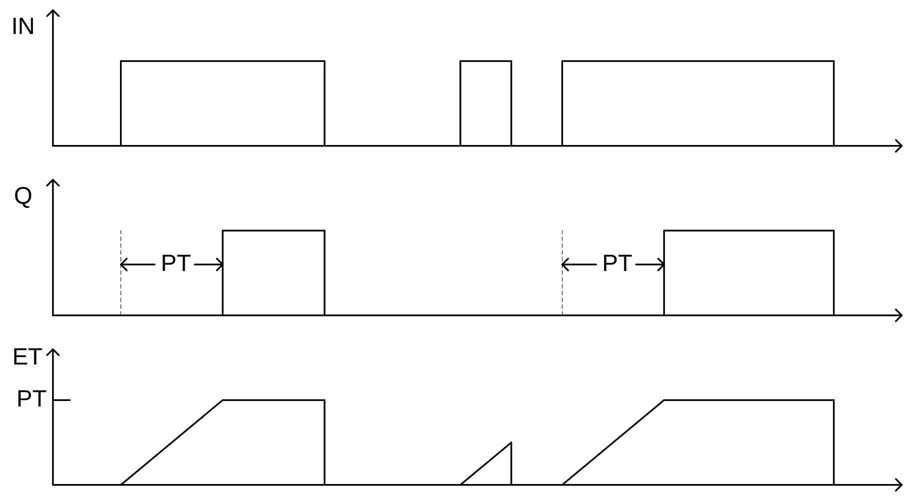


Fig. 7-6
On-Delay Timer
Timing Diagrams

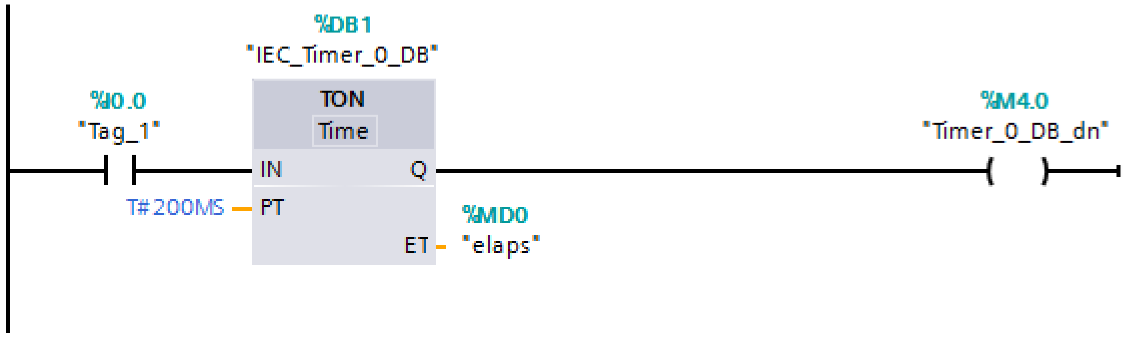


Fig. 7-7   On-Delay Timer Programmed in Ladder

TOF: Generate off-delay

The instruction **Generate off-delay** delays resetting of the output Q by the programmed duration PT. The Q output is set when the result of logic operation (RLO) at input IN changes from 0 to 1 (positive signal edge). When the signal state at input IN returns back to 0, programmed time (PT) starts. Output Q remains set as long as the duration PT is running. When duration PT expires, the Q output is reset. If the signal state at the IN input changes to 1 before the duration PT expires, the time is reset. The signal state at the output Q will continue to be 1.

The current time value can be queried at the ET output. The time value starts at T#0s and ends when the value of duration PT is reached. When the duration PT expires, the ET output remains set to the current value until input IN changes back to 1. If input IN switches to 1 before the duration PT has expired, the ET output is reset to the value T#0s.

These figures show pulse diagrams of the Generate Off-Delay instruction:

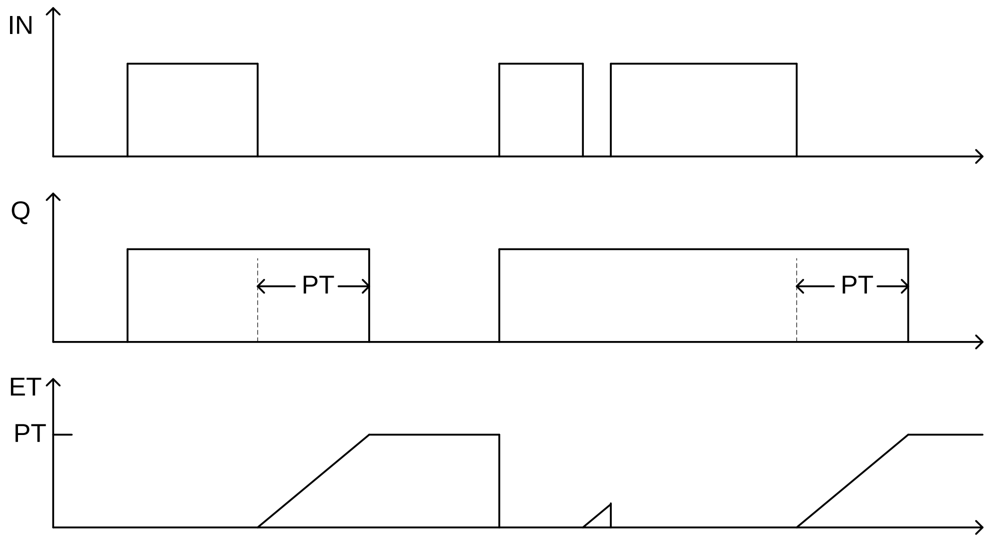


Fig. 7-8 Off-Delay Timer Timing Diagrams

ONR: Time accumulator

The **Time accumulator** instruction accumulates time values within a period set by parameter PT. When the signal state at input IN changes from 0 to 1 (positive signal edge), the instruction executes and the duration PT starts. While the duration PT is running, the timer values are accumulated that are recorded when the IN input has signal state 1. The accumulated time is written to output ET and can be queried there. When the duration PT expires, the output Q has the signal state 1. The Q parameter remains set to 1, even when the signal state at the IN parameter changes from 1 to 0 (negative signal edge).

The R input resets the outputs ET and Q regardless of the signal state at the start input.

These figures show pulse diagrams of the Time Accumulator instruction:

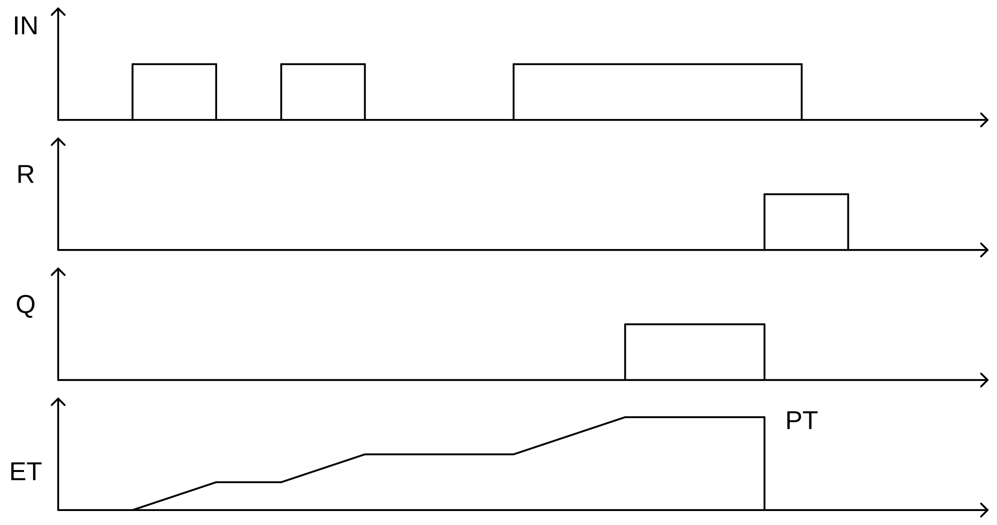


Fig. 7-9 Time Accumulate (or Retentive) Timer Timing Diagrams

---(TP)---: Start pulse timer

The Start pulse timer instruction starts an IEC timer with a specified duration as pulse. The IEC timer is started when the result of logic operation (RLO) changes from 0 to 1 (positive signal edge). The IEC timer runs for the specified time regardless of any subsequent changes in the RLO. The run of the IEC timer is also not affected by the detection of a new positive signal edge. As long as the IEC timer is running, the querying of the timer status for 1 returns the signal state 1. When the IEC timer has expired, the timer status returns the signal state 0.

The following example shows how the instruction works:

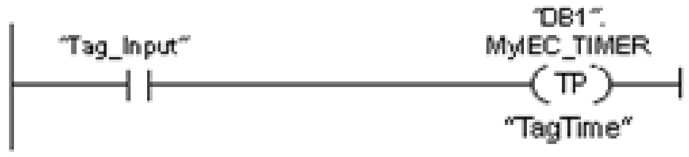


Fig. 7-10a Start Pulse Timer Input

The Start pulse timer instruction is executed when the signal state of the operand Tag_Input changes from 0 to 1. Timer DB1.MyIEC_TIMER starts running for the time duration that is stored in operand TagTime.

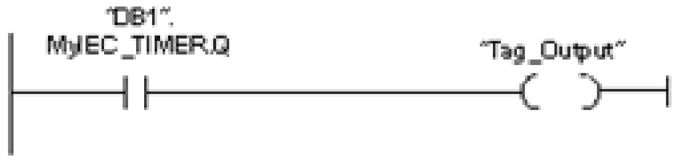


Fig. 7-10b Start Pulse Timer Output

As long as timer DB1.MyIEC_TIMER is running, the timer status (DB1.MyOEC_TIMER.Q) has the signal state 1 and the operand Tag_Output is set. When the IEC timer has expired, the signal state of the timer status changes back to 0 and the Tag_Output operand is reset.

---(TON)---: Start on-delay timer

The **Start on-delay timer** instruction starts an IEC timer with a specified duration as on-delay timer. The IEC timer is started when the result of logic operation (RLO) changes from 0 to 1 (positive signal edge). The IEC timer runs for the specified time. A query of the timer status for 1 returns signal state 1 if the time has expired and the RLO at the input of the instruction is 1. If the RLO changes to 0 before the time expires, the IEC timer is reset. In this case, querying the timer status for 1 returns signal state 0. The IEC timer restarts when the next positive signal edge is detected at the input of the instruction.

The following example shows how the instruction works:

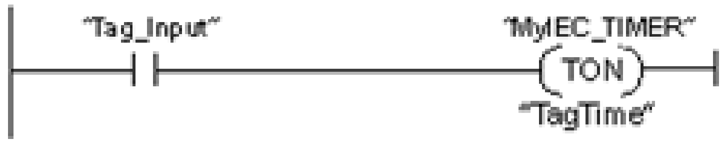


Fig. 7-11a Start On-Delay Timer Input

The **Start on-delay timer** instruction is executed when the signal state of the operand Tag_Input changes from 0 to 1. Timer MyIEC_TIMER starts running for the time duration that is stored in operand TagTime.

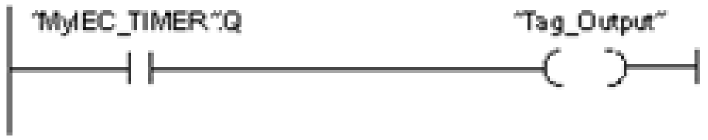


Fig. 7-11b Start On-Delay Timer Output

If the timer MyIEC_TIMER has expired and the operand Tag_Input has the signal state 1, querying the timer status (MyIEC_TIMER.Q) returns signal state 1 and the Tag_Output operand is set. When the signal state of the operand Tag_Input changes to 0, the querying of the timer status returns the signal state 0 and the operand Tag_Output is reset.

---(TOF)---: Start off-delay timer

The **Start off-delay timer** instruction starts an IEC timer with a specified duration as off-delay timer. The query of the timer status for 1 returns the signal state 1 if the result of the logic operation (RLO) at the input of the instruction has the signal state 1. When the RLO changes from 1 to 0 (negative signal edge), the IEC timer starts with the specified time. The timer status remains at signal state 1 as long as the IEC timer is running. When the timer has run out and the RLO at the input of the instruction has the signal state 0, the timer status is set to the signal state 0. If the RLO changes to 1 before the time expires, the IEC timer is reset and the timer status remains at signal state 1.

The following example shows how the instruction works:

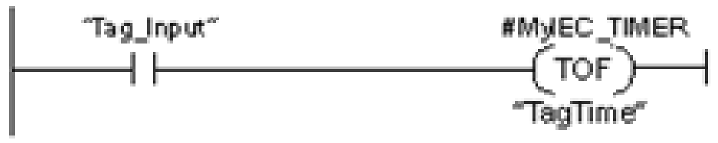


Fig. 7-12a Start Off-Delay Timer Input

The **Start off-delay timer** instruction is executed when the signal state of the operand Tag_Input changes from 1 to 0. Timer #MyIEC_TIMER starts running for the time duration that is stored in operand TagTime.

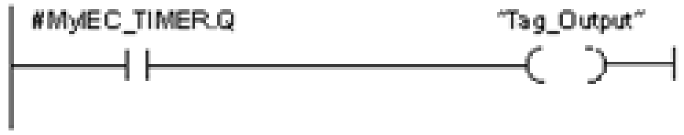


Fig. 7-12b Start Off-Delay Timer Output

As long as timer #MyIEC_TIMER is running, the query of the timer status (#MyIEC_TIMER.Q) returns the signal state 1 and operand Tag_Output is set. If the timer has expired and the operand Tag_Input has the signal state 0, the query of the timer status returns the signal state 0. If the signal state of the operand Tag_Input changes to 1 before timer #MyIEC_TIMER expires, the timer is reset. When the signal state of the operand Tag_Input is 1, the query of the timer status returns the signal state 1.

---(TONR)---: Time accumulator

The **Time accumulator** instruction records how long the signal is at the input of instruction 1. The instruction is started when the result of logic operation (RLO) changes from 0 to 1 (positive signal edge). The time is recorded as long at the RLO is 1. If the RLO changes to 0, the instruction is halted. If the RLO changes back to 1, the time recording is continued. The query of the timer status for 1 returns the signal state 1 if the recorded time exceeds the value of the specified duration and the RLO at the input of coil is 1.

The timer status and the currently expired timer can be reset to 0 using the **Reset timer** instruction.

The following example shows how the instruction works:

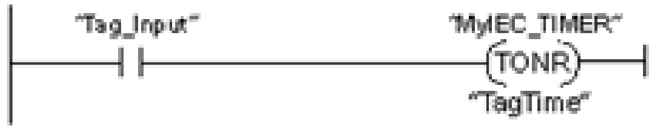


Fig. 7-13a Accumulator Timer Input

The **Time accumulator** instruction executes on a positive signal edge in the RLO. The time is recorded as long as the operand Tag_Input has the signal state 1.

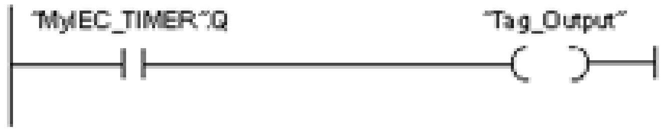


Fig. 7-13b Accumulator Timer Output

If the recorded time exceeds the value of the operand TagTime, the query of the timer status (#MyIEC_TIMER.Q) returns the signal state 1 and the operand Tag_Output is set.

---(RT)---: Reset timer

The **Reset timer** instruction resets an IEC timer to 0. The instruction is only executed if the result of logic operation (RLO) at the input of the coil is 1. If current is flowing to the coil (RLO is 1), the structure components of the timer in the specified data block are reset to 0. If the RLO at the input of the instruction is 0, the timer remains unchanged.

The instruction does not influence the RLO. The RLO at the input of the coil is sent directly to the output of the coil.

You assign the **Reset timer** instruction an IEC timer that has been declared in the program.

The instruction data is updated only when the instruction is called and not each time the assigned IEC timer is accessed. Querying the data is only identical from the call of the instruction to the next call of the instruction.

The following example shows how the instruction works:

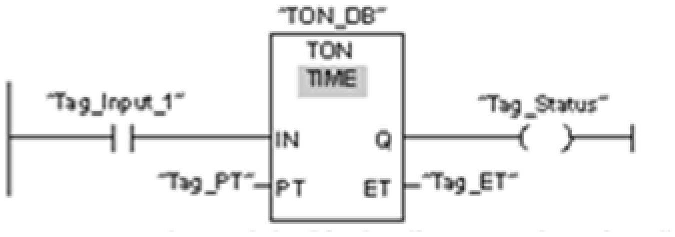


Fig. 7-14a Reset Timer Example

The **Generate on-delay** instruction executes when the signal state of the Tag_Input operand changes from 0 to 1. The timer stored in the TON_DB instance data block starts running for the time duration specified by operand Tag_PT.

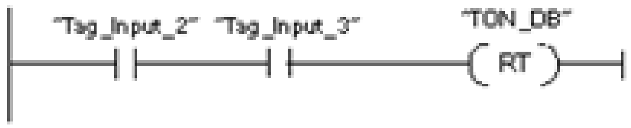


Fig. 7-14b Resetting the TON Timer

If operands Tag_Input_2 and Tag_Input_3 have the signal state 1, the Reset timer instruction is executed and the timer stored in the TON_DB data block is reset.

---(PT)---: Load time duration

The instruction **Load time duration** sets the duration of an IEC timer. The instruction is executed in every cycle when the result of logic operation (RLO) at the input of the instruction has the signal state 1. The instruction writes the specified time to the structure of the specified IEC timer.

You assign an IEC timer declared in the program to the **Load time duration** instruction.
The instruction data is updated only when the instruction is called and not each time the assigned IEC timer is accessed. Querying the data is only identical from the call of the instruction to the next call of the instruction.

The following example shows how the instruction works:

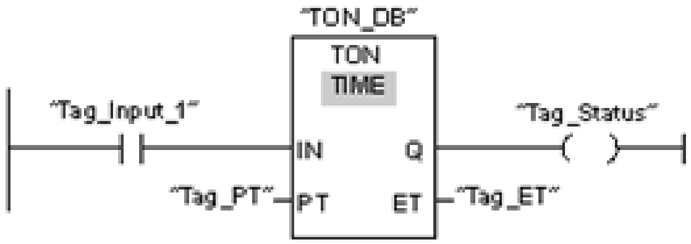


Fig. 7-15a Load Time Duration Example

The Generate on-delay instruction executes when the signal state of the Tag_Input_1 operand changes from 0 to 1. The IEC timer stored in the TON_DB instance data block starts running for the time duration specified by the operand Tag_PT.

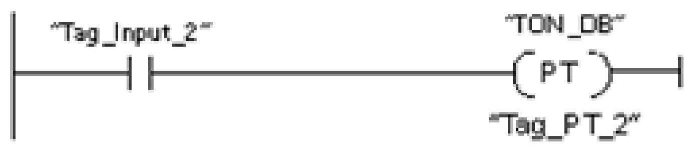
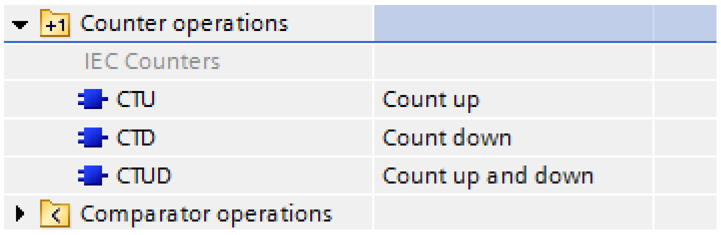


Fig. 7-15b Load Time Duration

When the Tag_Input_2 operand has signal state 1, the Load time duration instruction executes. The instruction writes the time duration Tag_PT_2 to the TON_DB instance data block, overwriting the value of the Tag_PT operand in the data block. The signal state of the timer status may therefore change at the next query.

Siemens Counters:

Next, we look at the Siemens Counter block and discuss the various types of counters available.



Counter operations	
IEC Counters	
CTU	Count up
CTD	Count down
CTUD	Count up and down
Comparator operations	

Fig. 7-16 Siemens Counter Operations

CTU: Count up

The instruction **Count up** counts up the value at output CV. When the signal state at the CU input changes from 0 to 1 (positive signal edge), the instruction executes and the current count value at the CV output is incremented by one. When the instruction executes for the first time, the current count value at the CV output is set to zero. The count value is incremented each time a positive signal edge is detected, until it reaches the high limit for the data type specified at the CV output. When the high limit is reached, the signal state at the CU input no longer has an effect on the instruction.

You can scan the counter status at the Q output. The signal state at the Q output is decided by the parameter PV. If the current count value is greater than or equal to the value of the PV parameter, the Q output is set to signal state 1. In all other cases, the Q output has signal state 0. You can also specify a constant for the PV parameter.

The value at the CV output is reset to zero when the signal state at the R input changes to 1. As long as the R input has signal state 1, the signal state at the CU input has no effect on the instruction.

Each call of the **Count up** instruction must be assigned an IEC counter in which the instruction data is stored. An IEC counter is a structure with one of the following data types:

- Data block of system data type IEC_counter (global DB):
    - IEC_SCOUNTER / IEC_USCOUNTER
    - IEC_COUNTER / IEC_UCOUNTER
    - IEC_DCOUNTER / IEC_UDCOUNTER
- Local tag:
    - CTU_SINT / CTU_USINT
    - CTU_INT / CTU_UINT
    - CTU_DINT / CTU_UDINT

You can declare an IEC counter as follows:

- Declaration of a data block of system data type IEC_COUNTER (for example, "MyIEC_COUNTER")
- Declaration as a local tag of the type CTU in the "Input", "InOut" or "Static" section of a block (for example, #MyIEC_COUNTER)

When you insert the instruction in the program, the Call options dialog opens in which you can specify whether the IEC counter is stored in its own data block (single instance) or as a local tag (multiple instance) in the block interface. If you create a separate data block, you will find this in the project tree in the Program resources folder under Program blocks > System blocks

The execution of the **Count up** instruction requires a preceding logic operation. It can be placed within or at the end of the network.

The following example shows how the instruction works:

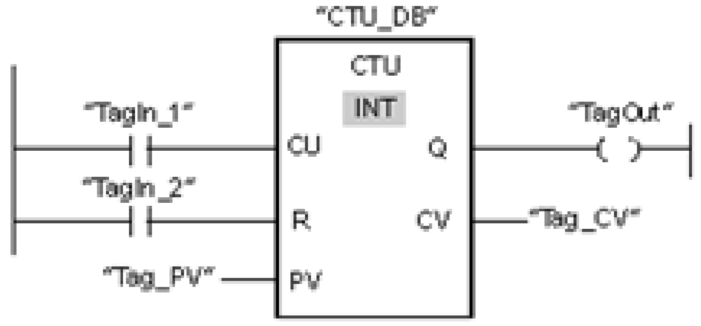


Fig. 7-17 Count Up Counter

When the signal state of the TagIn_1 operand changes from 0 to 1, the Count up instruction executes and the current value of the Tag_CV operand is incremented by one. With each additional positive signal edge, the count value is incremented until the high limit of the specified data type (32,767) is reached.

The value of the PV parameter is adopted as the limit for determining the TagOut output. The TagOut output has signal state 1 as long as the current value is greater than or equal to the value of the Tag_PV operand. In all other cases, the TagOut output has signal state 0.

CTD: Count down

You can use the **Count_down** instruction to decrement the value at output CV. When the signal state at the CD input changes from 0 to 1 (positive signal edge), the instruction executes and the current count value at the CV output is decremented by one. When the instruction executes the first time, the count value of the CV parameter is set to the value of the PV parameter. Each time a positive signal edge is detected, the count value is decremented until it reaches the low limit value of the specified data type. When the low limit is reached, the signal state at the CD input no longer has an effect on the instruction.

You can scan the counter status at the Q output. If the current count value is less than or equal to zero, the Q output is set to signal state 1. In all other cases, the Q output has signal state 0. You can also specify a constant for the PV parameter.

The value at the CV output is set to the value of the PV parameter when the signal state at the LD input changes to 1. As long as the LD input has signal state 1, the signal state at the CD input has no effect on the instruction.

The following example shows how the instruction works:

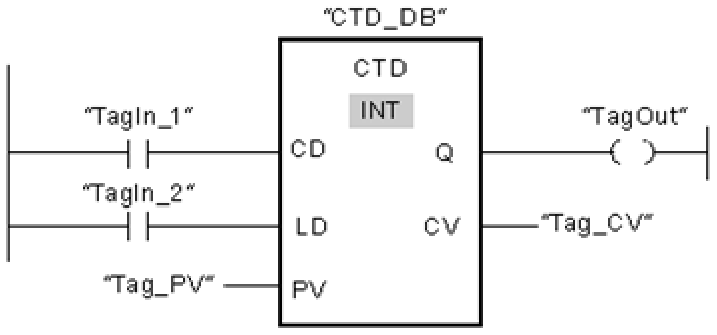


Fig. 7-18 Count Down Counter

When the signal state of the TagIn_1 operand changes from 0 to 1, the **Count down** instruction executes and the value at the Tag_CV output is decremented by one. With each additional positive edge signal, the count value is decremented until the low limit of the specified data type (-32,768) is reached.

The TagOut output has signal state 1 as long as the current count value is less than or equal to zero. In all other cases, the TagOut output has signal state 0.

CTUD: Count up and down

You can use the **Count up and down** instruction to increment and decrement the count value at the CV output. If the signal state at the CU input changes from 0 to 1 (positive signal edge), the current count value is incremented by one and stored at the CV output. If the signal state at the CD input changes from 0 to 1 (positive signal edge), the count value at the CV output is decremented by one. If there is a positive signal edge at the CU and CD inputs in one program cycle, the current count value at the CV output remains unchanged.

The count value can be incremented until it reaches the high limit of the data type specified at the CV output. When the high limit value is reached, the count value is no longer incremented on a positive signal edge. When the low limit of the specified data type is reached, the count value is not decremented any further.

When the signal state at the LD input changes to 1, the value (count value) at the CV output is set to the value of the PV parameter. As long as the LD input has the signal state 1, the signal state at the CU and CD inputs has no effect on the instruction.

The count value is set to zero when the signal state at the R input changes to 1. As long as the R input has signal state 1, a change in the signal state of the CU, CD and LD inputs has no effect on the **Count up and down** instruction.

You can scan the current status of the up counter at the QU output. If the current count value is greater than or equal to the value of the PV parameter, the QU output is set to signal state 1. In all other cases, the QU output has signal state 0. You can also specify a constant for the PV parameter.

You can scan the current status of the down counter at the QD output. If the current count value is less than or equal to zero, the QD output is set to signal state 1. In all other cases, the QD output has signal state 0.

The following example shows how the instruction works:

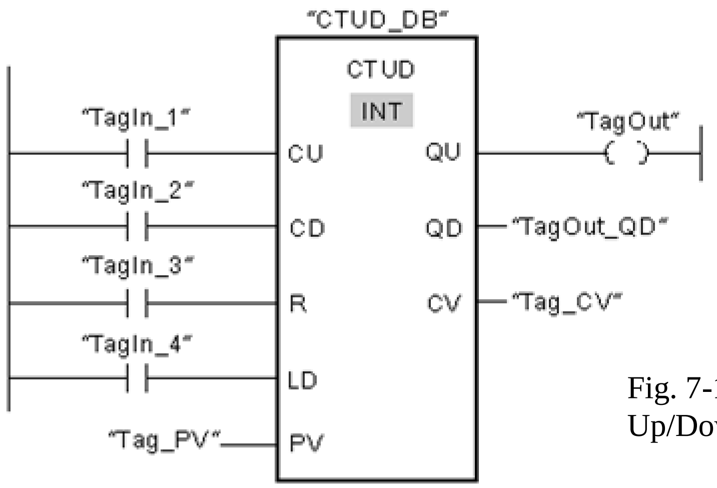


Fig. 7-19 Combined Count Up/Down Counter

If the signal state at the TagIn_1 or TagIn_2 input changes from 0 to 1 (positive signal edge), the Count up and down instruction is executed. When there is a positive signal edge at the TagIn_1 input, the current value is incremented by one and stored at the Tag_CV output. When there is a positive signal edge at the CU input, the count value is incremented until it reaches the high limit of 32,767. If input CD has a positive signal edge, the count value is decremented until it reaches the low limit of -32,768.

The TagOut output has signal state 1 as long as the current count value is greater than or equal to the value at the TagPV input. In all other cases, the TagOut output has signal state 0. The

TagOut_QD output has signal state 1 as long as the current count value is less than or equal to zero. In all other cases, the TagOut_QD output has signal state 0.

Allen-Bradley SLC Timers and Counters

SLC Timer Layout:

Timer On Delay TON is the non-retentive instruction for on-delay timers. It is used to provide signals that change state a time delay after the TON block is energized. TOF is the non-retentive instruction for off-delay timers. RTO is the retentive timer instruction. It does not reset to an initial value but rather stays at an accumulated value each time the input to the function block is energized. It 'retains' the count previously accumulated and continues on from that value. It must be reset with a RES (Reset) command. Reset commands are useful for not only the retentive timer instruction but also any timer or counter with a retentive nature.

Timers are addressed using the T4 file. Three words are used for each timer. These words are set up in a fixed block format as follows:

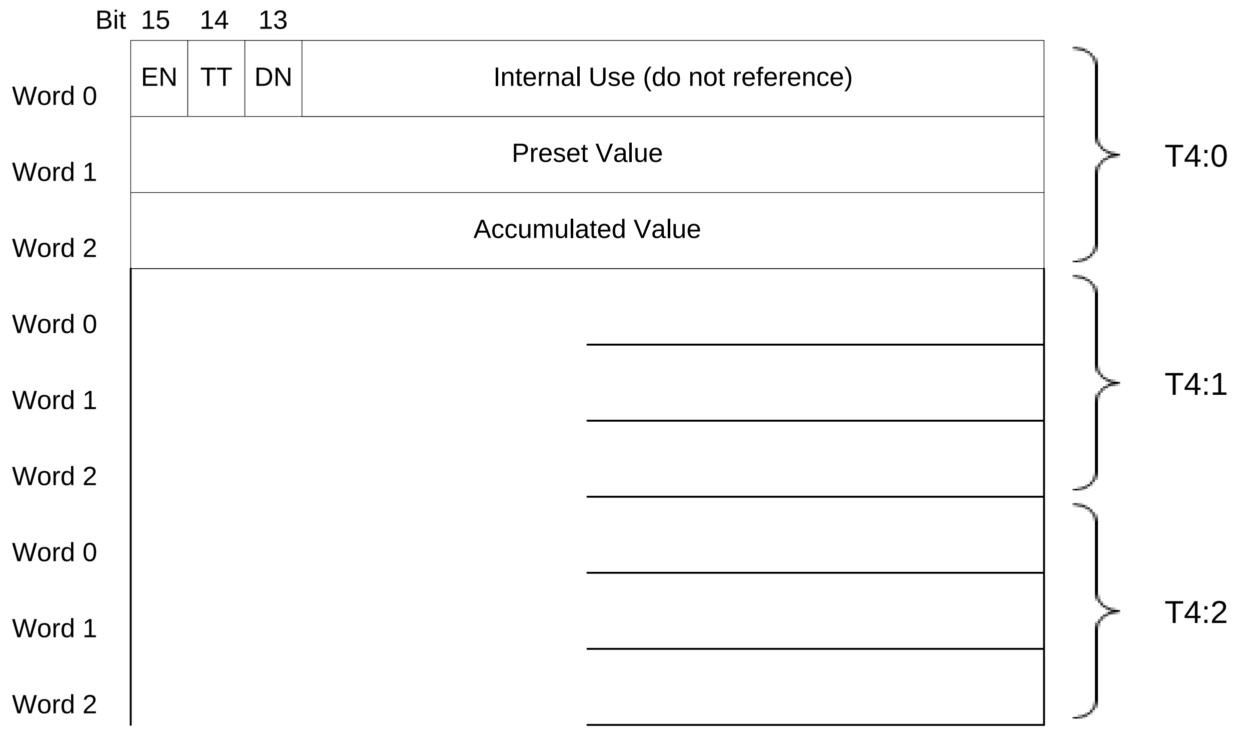
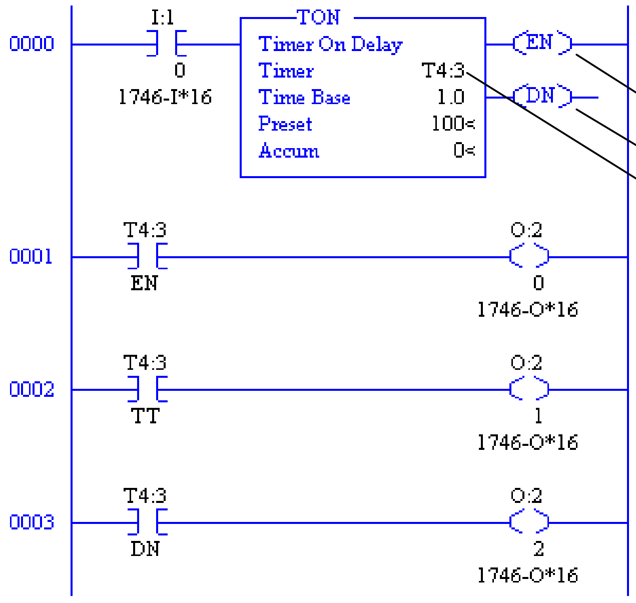


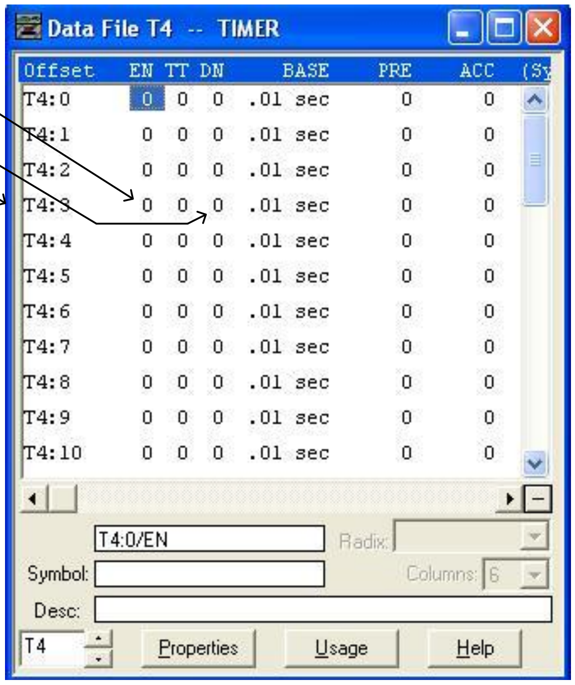
Fig. 7-20 SLC Timer T4 Table Layout

To program a timer and view its control contacts helps understand the functionality of the timer.

The example below shows the TON timer block with all types of contacts used referenced to output coils. Build this circuit to show delay contacts in action. Another approach would be to view the T4 data file from the project tree with the processor in On-Line Run Mode.

A major difference between A-B timers and counters and the Siemens' equivalent is that the control bits are built into the A-B devices. The control bits must be added per the Siemens' explanations. What is tied to the Q output becomes the control bit for a Siemens function.





Offset	EN	TT	DN	BASE	PRE	ACC	(Sy
T4:0	0	0	0	.01 sec	0	0	
T4:1	0	0	0	.01 sec	0	0	
T4:2	0	0	0	.01 sec	0	0	
T4:3	0	0	0	.01 sec	0	0	
T4:4	0	0	0	.01 sec	0	0	
T4:5	0	0	0	.01 sec	0	0	
T4:6	0	0	0	.01 sec	0	0	
T4:7	0	0	0	.01 sec	0	0	
T4:8	0	0	0	.01 sec	0	0	
T4:9	0	0	0	.01 sec	0	0	
T4:10	0	0	0	.01 sec	0	0	

T4:0/EN Radix:
Symbol: Columns: 6
Desc:
T4

Fig. 7-21 TON Timer with SLC Controller (and associated logic)

Preset (PRE) and Accumulated (ACC) values can be modified on-line as well as offline. These values are adjusted many times after the program is running in order to eliminate any wasted time in the cycle of a machine. To move a preset from .50 to .05 second can save hundreds or even thousands of dollars in added productivity of a machine over a year's time. As a rule, these timers are set to the lowest possible value to not damage the machine or cause a problem.

Addresses may use either the specific bit/word or the mnemonic label:

T4:0/15	equal to	T4:0/EN	(enable contact)
T4:0/14	equal to	T4:0/TT	(timer timing contact)
T4:0/13	equal to	T4:0/DN	(done contact)
T4:0.1	equal to	T4:0.PRE	(preset word)
T4:0.2	equal to	T4:0.ACC	(accumulated word)
T4:0.1/0	equal to	T4:0.PRE/0	(preset word, bit 0)
T4:0.2/2	equal to	T4:0.ACC/2	(accumulated word, bit 2)

Entering the instruction requires a time base option. Time bases are .01, 0.1 or 1.0 second. Presets are entered as multiple counts of the time base.

Time Base	Preset	Time Duration
.01	50	.5 sec
.01	9999	99.99 sec
.1	200	20.0 sec
1.0	30	30 sec

A flashing circuit with two timers:

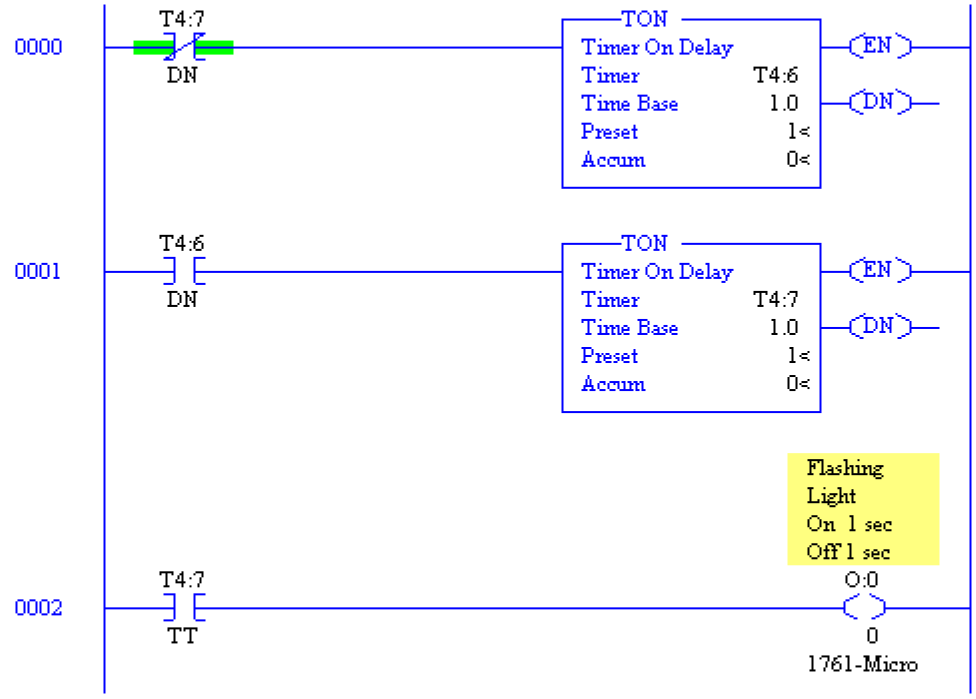


Fig. 7-22 Two TON Timers Used for Flashing Light Application

A Retentive Timer does not reset each time the Enable turns off. The ACC (accumulated value) remains unchanged. To turn off a Retentive Timer, a RES (Reset) coil must be energized.

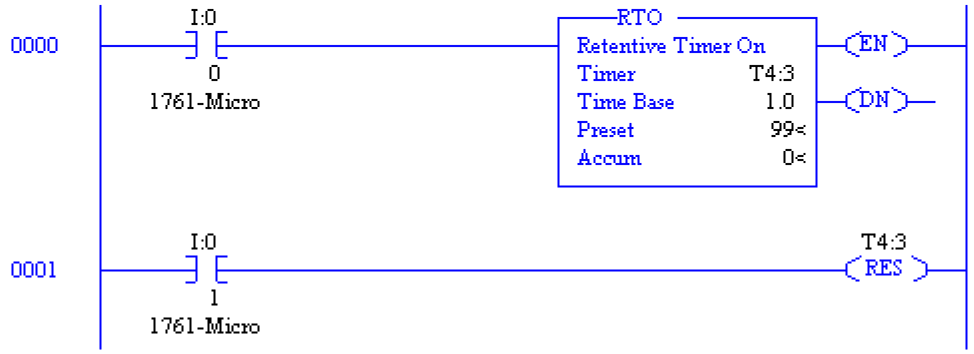


Fig. 7-23 Retentive Timer Application

Along with timers, counters are introduced to provide counting functions similar to mechanical counters. Function blocks allow inputs from the left and pass power through to the right when the counting function is done. When done, the counter has counted to the preset.

Counters can be either up or down counters. Up counters count from zero to a preset while down counters count down from preset to zero. When the target or preset is reached, the counter-done turns on. Some counters allow counts above the preset and below zero. It is always wise to try any counter using a push-button input before relying on it to control programs.
They require two numbers stored: a preset and an accumulated number. They need two inputs: an enable and reset. And, they have an output that turns on when the accumulated equals the preset.

Counters differ from timers in that the accumulated count moves by one for each new leading edge of the input. The count continues to move until the preset is reached. When the Preset equals Accumulated, the output turns on. If more new leading edges are received, the count continues to climb. The output stays on until the counter is reset. There is no need to have retentive or non-retentive counters since all counters are reset with the *RES* command. It does not matter how long a signal is on for a counter. The count will only increment one for each new leading edge. Counts are retained after a power outage.

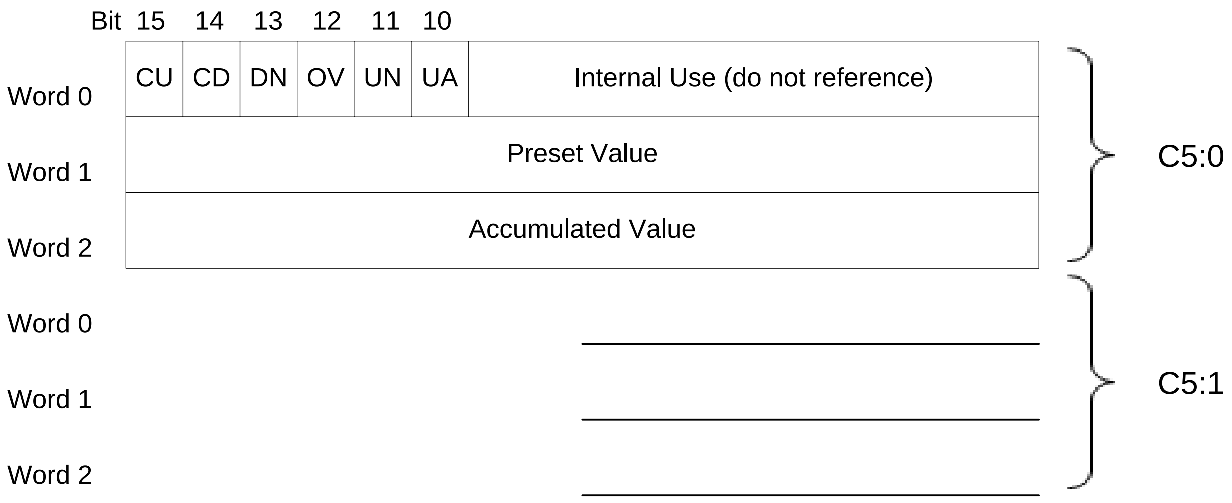
Counters may also be chained together to form very large counts. Counters can count to 32,767. The number 32,767 therefore is the largest preset. If counts climb into the thousands and millions, chaining of counters becomes a necessity.

Can a counter count both up and down? This is a necessity of some counters and all PLC vendors must provide a means to do it. This function will be described later. Several examples of counters will provide practical experiences of how counters are used.

As with all circuits in this book, it is encouraged that students construct the circuit and observe it in action. If each building block is constructed correctly, the whole program has a much greater chance of performing to expectation.

Allen-Bradley provides two counters: **CTU** and **CTD**. Both are block-type instructions and are located at the right of the rung similar to timers. Counters trigger or count on a false-to-true transition of the rung logic. Care must be taken to allow sufficient time for the signal to be read by the input card or input point on the fixed I/O processor. If the time of the pulse is less than the time of a scan, the signal will probably be missed. To accommodate higher pulse rates, PLC manufacturers provide high speed counter modules capable of counting pulses to much higher frequencies than by using the **CTU** and **CTD** function block.

Counters are addressed using the C5 file. Three words are used for each counter. These words are described as follows:



Bit 15:	CU	= Count Up Enable
Bit 14:	CD	= Count Down Enable
Bit 13:	DN	= Done Bit
Bit 12:	OV	= Overflow Bit
Bit 11:	UN	= Underflow Bit
Bit 10:	UA	= Update Accum Value (used in high speed counter only – HSC)

Fig. 7-24 SLC Counter C5 Layout

The Count Up Overflow bit is set when the accumulated value increments above 32767 to –32768 and stays set until the RES instruction resets the count or the count is decremented by 1 back to 32767. The Done bit turns on when the accumulated value becomes equal to or greater than the preset. Enable (either CU or CD) turn on when the rung condition is true.

The two counters below are independent. C5:0 is an Up Counter with an ACC starting at 0 and incrementing by 1 with each occurrence of I:0/0. When ACC = 99, the DN bit turns on.

C5:1 is a Down Counter with an ACC starting at 50 and decrementing by 1 with each occurrence of I:0/1. When ACC = 0, the DN bit turns on. Each counter has a reset rung with discrete input control.

To provide a counter that counts both up and down requires addressing the same C5: counter address in both the CTU and CTD block. The accumulated value keeps the characteristic of the CTU counter counting up from 0 to the preset value. This counter can count both positive and negative if accessed by both the CTU and CTD block.

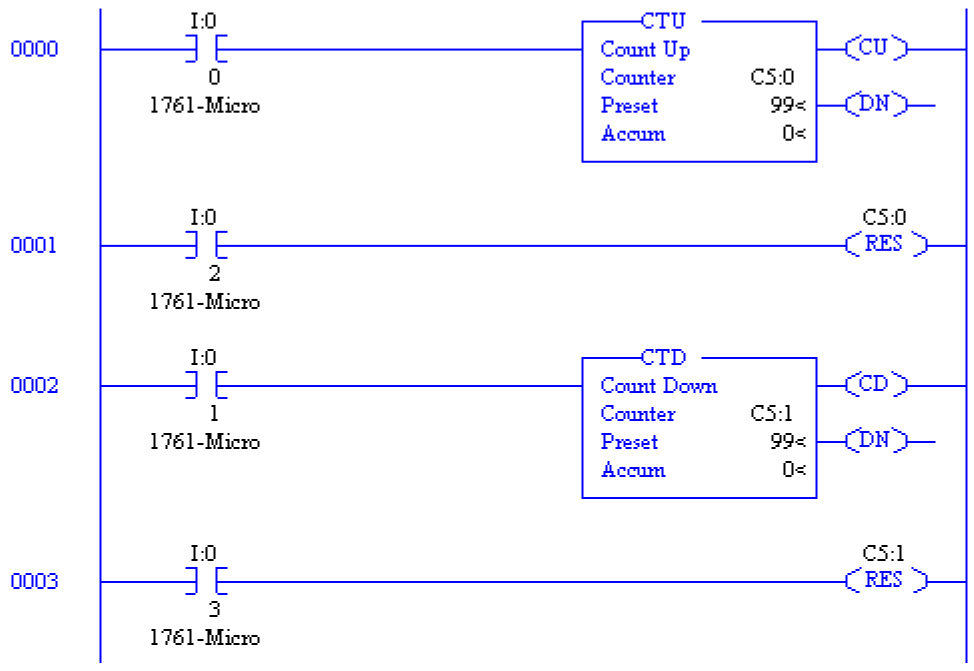


Fig. 7-25 Count Up Timer C5:0, Count Down Timer C5:1

CompactLogix Timers and Counters

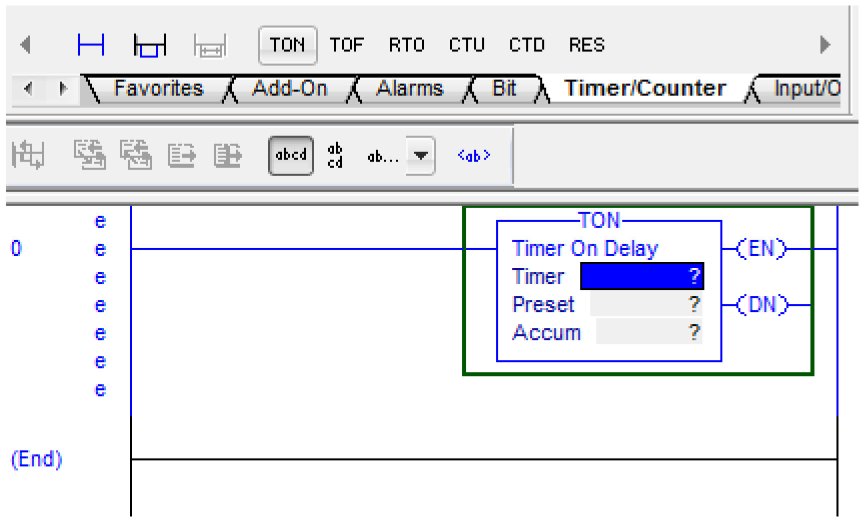


Fig. 7-26 Compact Logix On Delay Timer

Timers and counters are designed in the ControlLogix platform to perform in the same manner as with the SLC family. They must also be capable of functioning in alternate languages such as Function Block Diagram. One nice feature of the ControlLogix timer is the lack of a programmed time base. All timers have a .001 sec or 1 msec time base. Timer preset and accumulated values are stored in DINT (Double Integer) values.

Timer On Delay (TON)

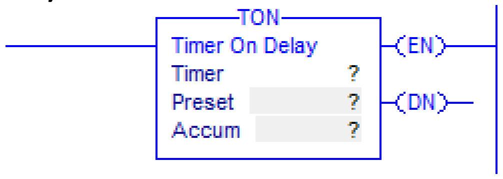


Fig. 7-27 On-Delay Timer Being Programmed

The TON timer is a non-retentive timer that accumulates time when the instruction is enabled. The instruction is available in FBD as TONR. It is also available in Structured Text as TONR. The Preset is stored as a DINT and represents the duration of the delay. The Accum is stored as a DINT and represents the total msec of accumulated time. The initial value is usually 0.

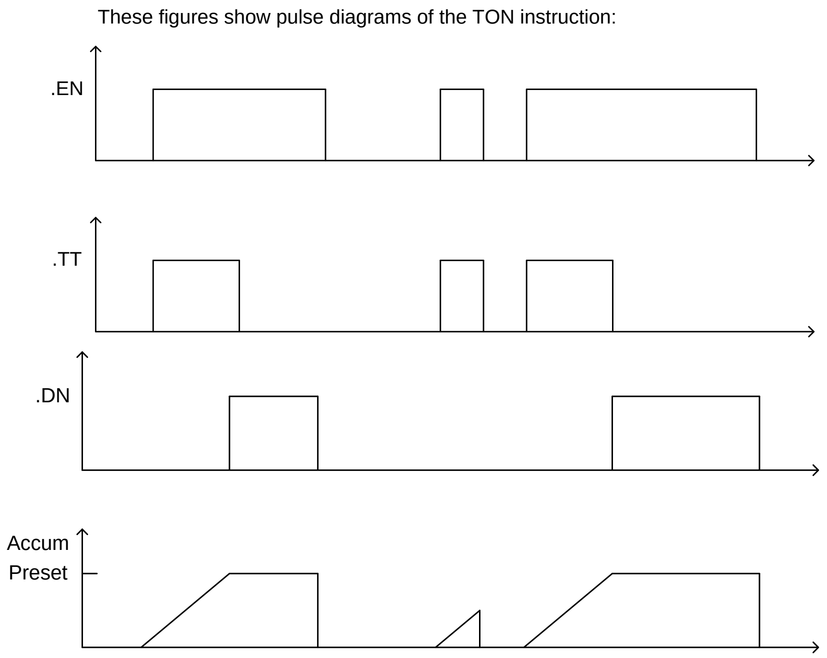


Fig. 7-28 Timing Diagram for On Delay Timer

Timer Off Delay (TOF)

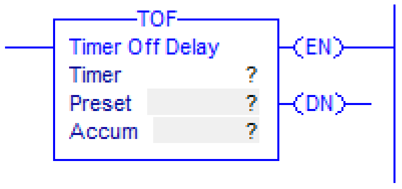


Fig. 7-29 Off-Delay Timer Being Programmed

The TOF instruction is a non-retentive timer that accumulates time when the instruction is enabled (rung-condition-in is false). The TOF instruction accumulates time until the TOF instruction is disabled or the .ACC ≥ .PRE. This instruction is available in function block and structured text as TOFR.

These figures show pulse diagrams of the TOF Off-Delay instruction:

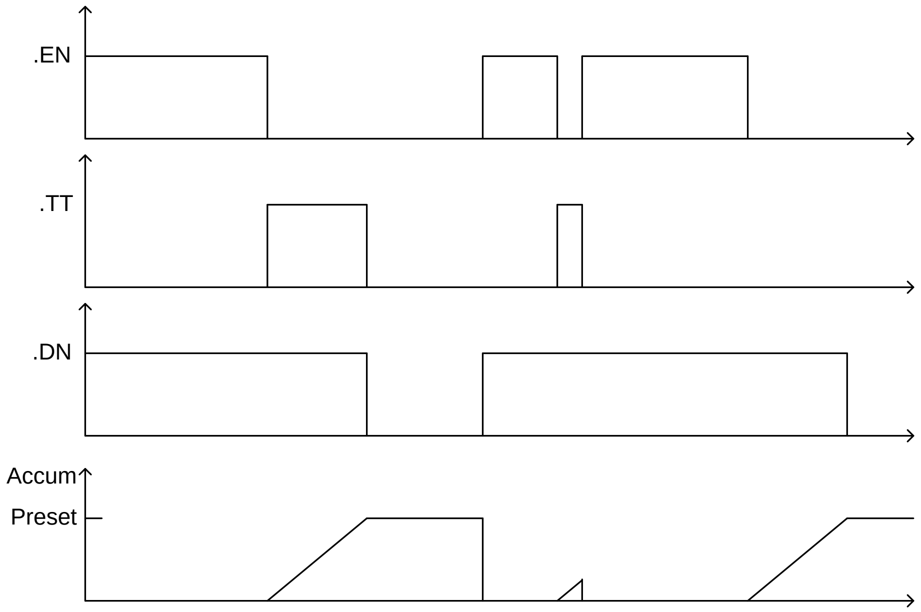


Fig. 7-30 Timing Diagram for Off-Delay Timer

Retentive Timer On Delay (RTO) and Reset (RES)

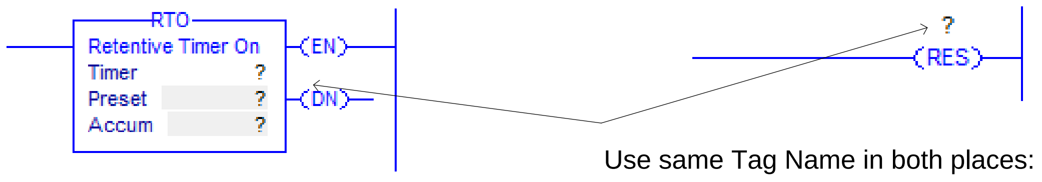


Fig. 7-31 Retentive Timer Being Programmed

The RTO instruction accumulates time until it is disabled. When the RTO instruction is disabled, it retains its .ACC value. You must clear the .ACC value, typically with a RES instruction referencing the same Timer.

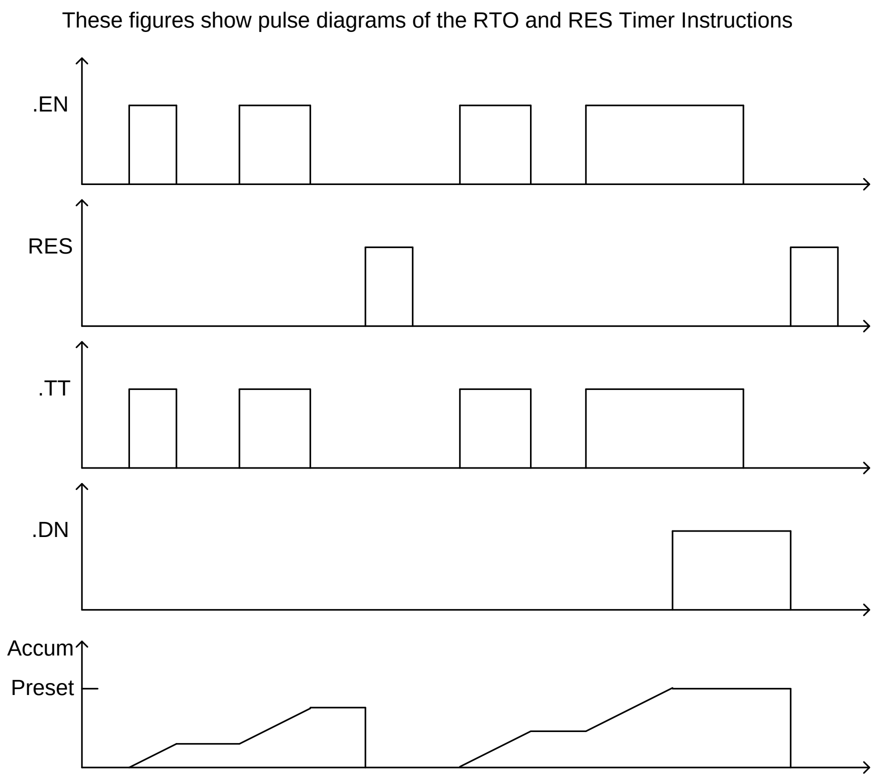


Fig. 7-32 Timing Diagram for Retentive Timer

Timers and Seal Circuit Combined

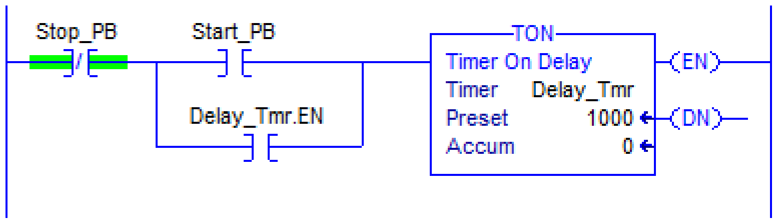


Fig. 7-33 Using Timers in Seal Circuits

The rung above may not resemble a seal circuit but look at it closely. When Start PB is energized, TON Delay_Tmr.EN turns on providing a parallel path around Start PB similar to other seal circuits. Stop PB energized turns off the circuit. While the circuit should show a NO with the Stop PB input, the use of the NC reminds us that the button was wired to NC contacts.

The seal circuit above is used quite often in sequential logic in which the machine moves from state to state with each new state established with the start contact of the seal circuit. The timer is useful to provide a time buffer between steps. Timers are necessary in this type of program since the machine should be set to move smoothly from action to action. If no time delay is provided between steps, the machine appears to travel in a jerky manner. Many machines will not function well for long without the timer to allow a slight delay between steps. These machines will appear to beat themselves to death.

Timers may be used in chained operations. Timer 1 turns on timer 2 which in turn activates timer 3, etc. Timers are useful in a number of applications using chained timer functions.

CompactLogix Counters

Counters are created in a fashion similar to the SLC counter. Counter types echo the SLC counters. As with timers, counter tags must be created before being programmed.

Count Up (CTU)

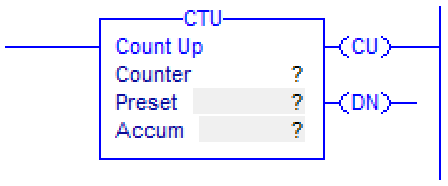


Fig. 7-34 Up Counter Being Programmed

When enabled and the .CU bit is cleared, the CTU instruction increments the counter by one. When enabled and the .CU bit is set, or when disabled, the CTU instruction retains its .ACC value. The instruction counts up with each new leading edge at the left of the block. Counts are stored in the .ACC value and range from 2^{-31} to 2^{31} (-2,147,483,647 to 2,147,483,647). The table of bit assignments for the CTU and CTD counter follow:

.CU	BOOL	The count up enable bit indicates the CTU instruction is enabled.
.DN	BOOL	The done bit indicates that .ACC $\geq$ .PRE.
.OV	BOOL	The overflow bit indicates the counter exceeded the upper limit of 2,147,483,647. The counter then rolls over to -2,147,483,648 and begins counting up again.
.UN	BOOL	The underflow bit indicates that the counter exceeded the lower limit of -2,147,483,647. The counter then rolls over to 2,147,483,647 and begins counting down again.
.PRE	DINT	The preset value specifies the value which the accumulated value must reach before the instruction sets the .DN bit.
.ACC	DINT	The accumulated value specifies the number of transitions the instruction has counted.

Count Down (CTD)

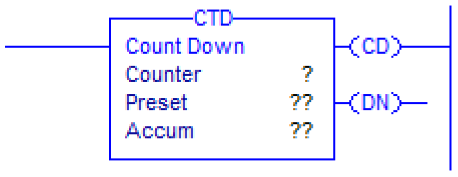


Fig. 7-35 Down Counter Being Programmed

The CTD instruction is typically used with a CTU instruction that references the same counter tag. When enabled and the .CD bit is cleared, the CTD instruction decrements the counter by one. When enabled and the .CU bit is set, or when disabled, the CTD instruction retains its .ACC value.

Both the CTU and CTD instruction use the RES instruction to reset the counter's .ACC value to zero. This is the same RES instruction used to reset the Retentive Timer instruction RTO.

Timer and Counter Examples

When Pump_Run is toggled on, Delay_Tmr begins to time and Delay_Tmr.DN turns on 500 msec after Pump_Run (Delay_Tmr.EN) turns on. With the counter, when Part_Present is toggled on, Part_Count increments by one. When Part_Present is toggled 1000 times, Part_Count.DN turns on.

These two examples are separate and not linked. The tables for each look very similar, however.

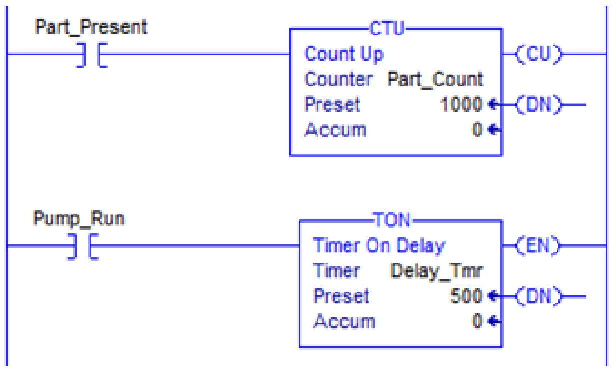
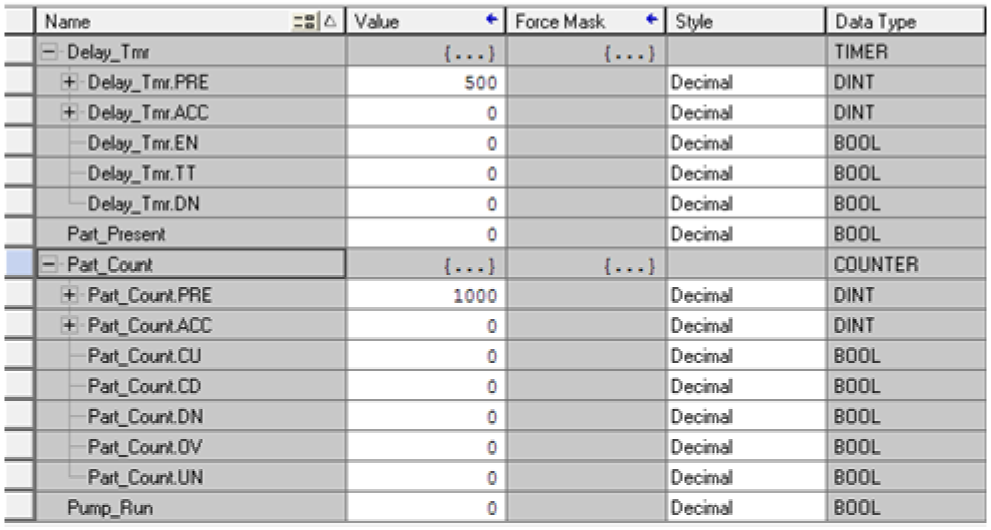


Fig. 7-36a Compact Logix TON-CTU Examples



Name	Value	Force Mask	Style	Data Type
Delay_Tmr	{...}	{...}		TIMER
Delay_Tmr.PRE	500		Decimal	DINT
Delay_Tmr.ACC	0		Decimal	DINT
Delay_Tmr.EN	0		Decimal	BOOL
Delay_Tmr.TT	0		Decimal	BOOL
Delay_Tmr.DN	0		Decimal	BOOL
Part_Present	0		Decimal	BOOL
Part_Count	{...}	{...}		COUNTER
Part_Count.PRE	1000		Decimal	DINT
Part_Count.ACC	0		Decimal	DINT
Part_Count.CU	0		Decimal	BOOL
Part_Count.CD	0		Decimal	BOOL
Part_Count.DN	0		Decimal	BOOL
Part_Count.OV	0		Decimal	BOOL
Part_Count.UN	0		Decimal	BOOL
Pump_Run	0		Decimal	BOOL

The same examples are next shown with Siemens' programming:

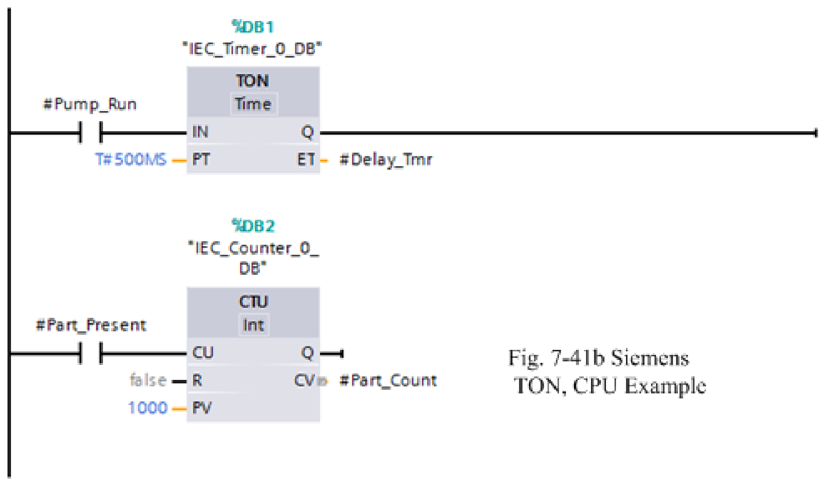
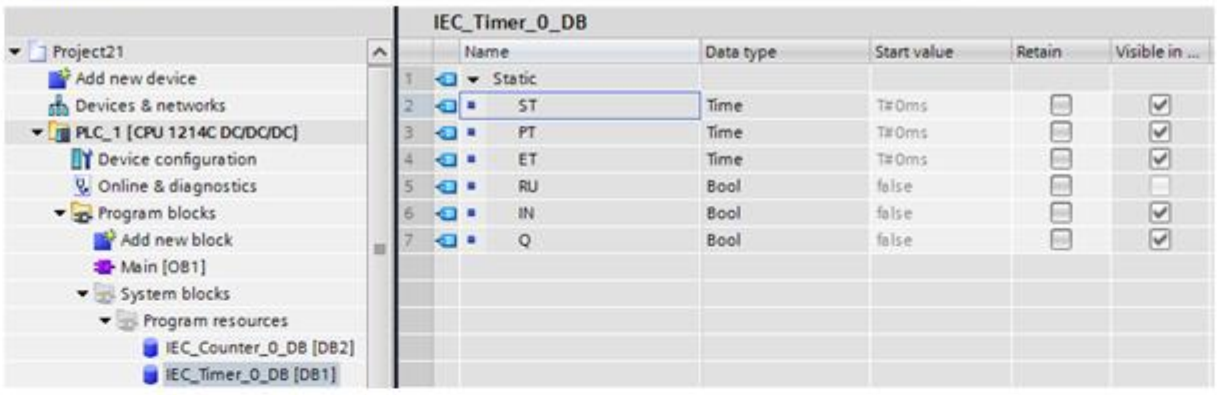


Fig. 7-41b Siemens TON, CPU Example

Fig. 7-36b Siemens S7-1200 TON-CTU Examples

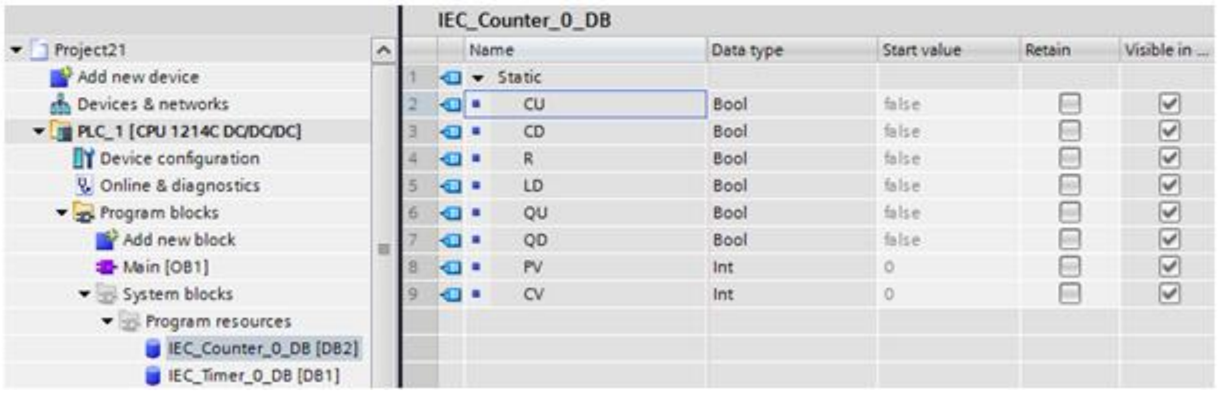
▼ #Pump_Run
#Delay_Tmr
#Part_Present
#Part_Count



Project21
- Add new device
- Devices & networks
- PLC_1 [CPU 1214C DC/DC/DC]
  - Device configuration
  - Online & diagnostics
  - Program blocks
    - Add new block
    - Main [OB1]
    - System blocks
      - Program resources
        - IEC_Counter_0_DB [DB2]
        - IEC_Timer_0_DB [DB1]

IEC_Timer_0_DB

	Name	Data type	Start value	Retain	Visible in ...
1	▼ Static				
2	ST	Time	T#0ms	☐	✓
3	PT	Time	T#0ms	☐	✓
4	ET	Time	T#0ms	☐	✓
5	RU	Bool	false	☐	☐
6	IN	Bool	false	☐	✓
7	Q	Bool	false	☐	✓



Project21
- Add new device
- Devices & networks
- PLC_1 [CPU 1214C DC/DC/DC]
  - Device configuration
  - Online & diagnostics
  - Program blocks
    - Add new block
    - Main [OB1]
    - System blocks
      - Program resources
        - IEC_Counter_0_DB [DB2]
        - IEC_Timer_0_DB [DB1]

IEC_Counter_0_DB

	Name	Data type	Start value	Retain	Visible in ...
1	▼ Static				
2	CU	Bool	false	☐	✓
3	CD	Bool	false	☐	✓
4	R	Bool	false	☐	✓
5	LD	Bool	false	☐	✓
6	QU	Bool	false	☐	✓
7	QD	Bool	false	☐	✓
8	PV	Int	0	☐	✓
9	CV	Int	0	☐	✓

Siemens builds a data block (DB1, DB2) that stores the values of the timer and the counter. The same values are found in the tags created in A-B's program tag database.

Use the Monitor function to change timer values when several timers need to be adjusted online. The use of the tag database to set timers requires planning to store timers in a common database area. Siemens uses a watch table to change these values.

Seal Circuits with Time Delays

Seal circuits used to start and stop pumps, fans and other equipment many times must interlock with switches to insure safe operation. For pumps, flow switches are used. For fans, pressure switches or flow switches are selected. For conveyor belts, zero-speed or plugging switches are used. If the switch fails or stops, the moving device is also to stop.

The problem is the starting operation. How does the device allow the circuit to start? A timer is used for each of the devices in the process. For instance, if a pump is allowed to start, a timer is also started which times out and then causes the flow switch to activate and stop the pump if low flow is detected. The following circuit shows the operation of this kind of seal logic.

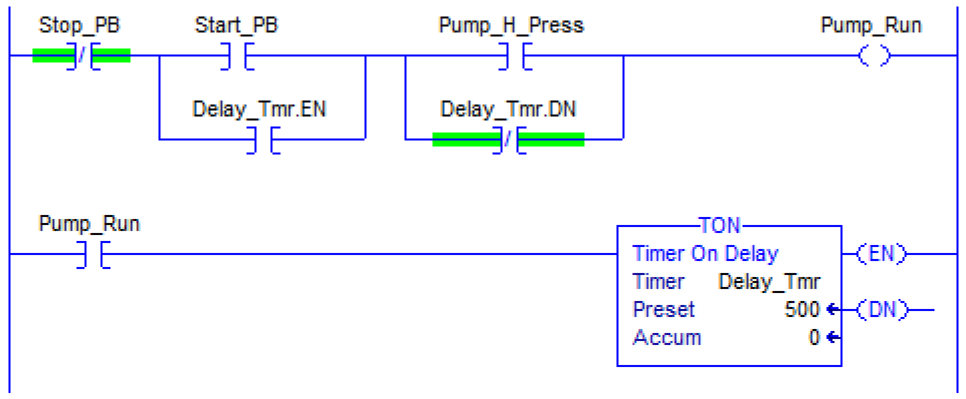


Fig. 7-37 Example of Timer in Stop Circuit

In rung 1 above, the pump starts with I:0/0 (Start_PB). Either the push button I:0/1(Stop_PB) or the combination of I:0/2 (Pump_H_Press) or T4:3/DN shut the pump off. If I:0/1 turns off, the pump shuts down. If I:0/2 shuts off (low pressure), the pump will shut down but only after the time delay of T4:3. The time delay allows time for the pump to build pressure enough to run under normal conditions. If the pressure then fails (turns off), the pump will turn off. Usually this circuit also includes a diagnostic rung which will turn on when the pressure switch fails and the pump is on to alert the operator of the failure of the pump.

From Chapter 6, we remember the following problem:

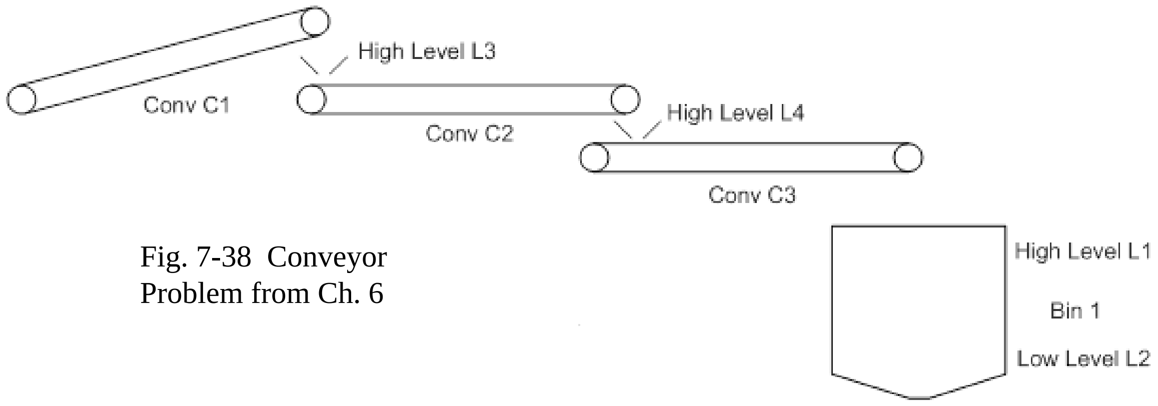


Fig. 7-38 Conveyor Problem from Ch. 6

We built Function/State and Signal Assignment Tables for it as follows:

Sensor	Function/State	Signal Assignment
L1	Bin 1 High Level	0
L2	Bin 1 Low Level	1
L3	Conv C2 Hopper High Level	0
L4	Conv C3 Hopper High Level	0

Actuator	Function/State	Signal Assignment
C1	Conv C1 Run	1
C2	Conv C2 Run	1
C3	Conv C3 Run	1

A zero speed switch is usually added to the conveyor to determine if it is running properly. A sensor is attached to the end of the conveyor opposite the motor shaft. This sensor must sense a motion or it will report that the conveyor is not running properly. The input is attached to the shaft. It may be as simple as a bolt that is being read by a prox switch as it passes the switch when the shaft is turning.

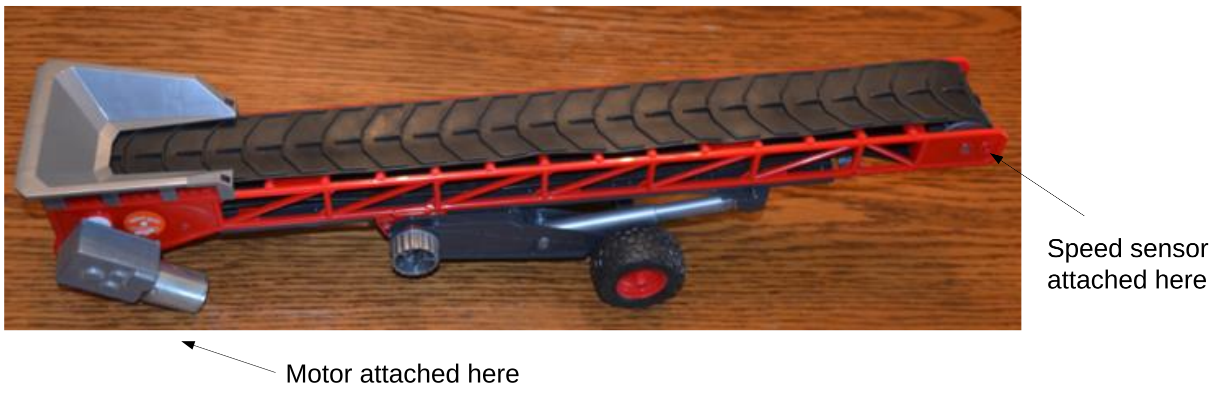


Fig. 7-39 Conveyor used for Moving Product

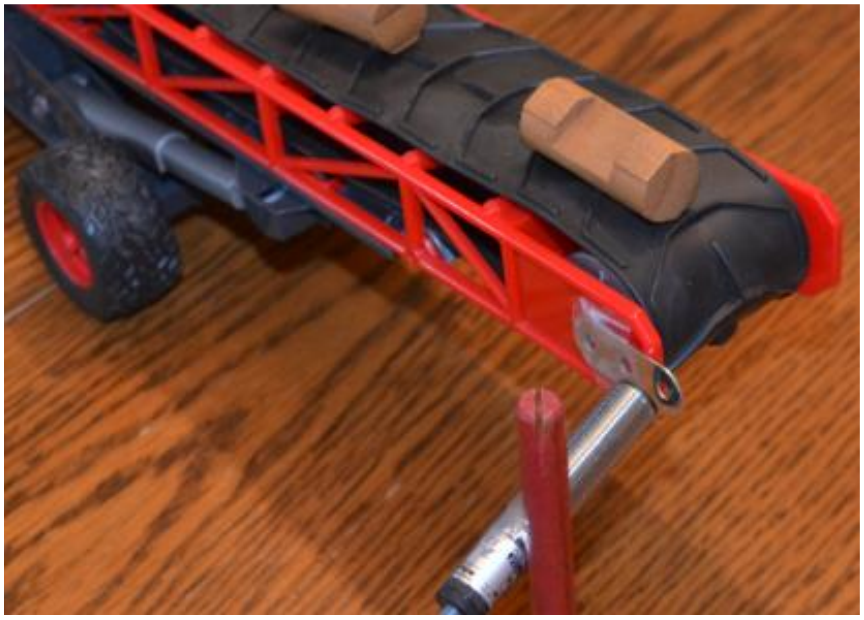


Fig. 7-40
A zero speed switch is built from a rotating metal piece attached to the shaft and a proximity switch

As can be seen, both the signal on too long or off too long is reason to report that the shaft is not turning. Both states must be checked. A reasonable time for the shaft to turn one revolution is calculated and a little extra time is added for start-up or stop conditions. Then the timer is added to the motor control program.

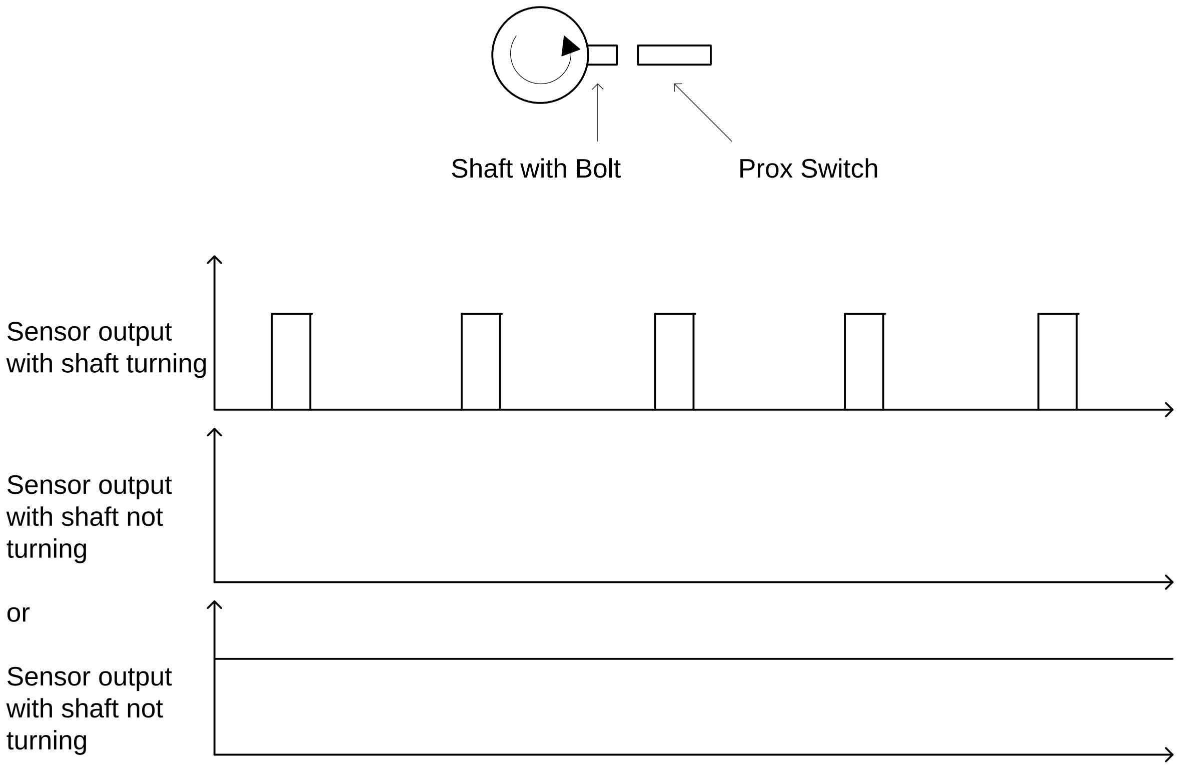

Shaft with Bolt
Prox Switch
Sensor output with shaft turning
Sensor output with shaft not turning
or
Sensor output with shaft not turning

Fig. 7-41 Conveyor Problem with Shaft Rotation Sensing

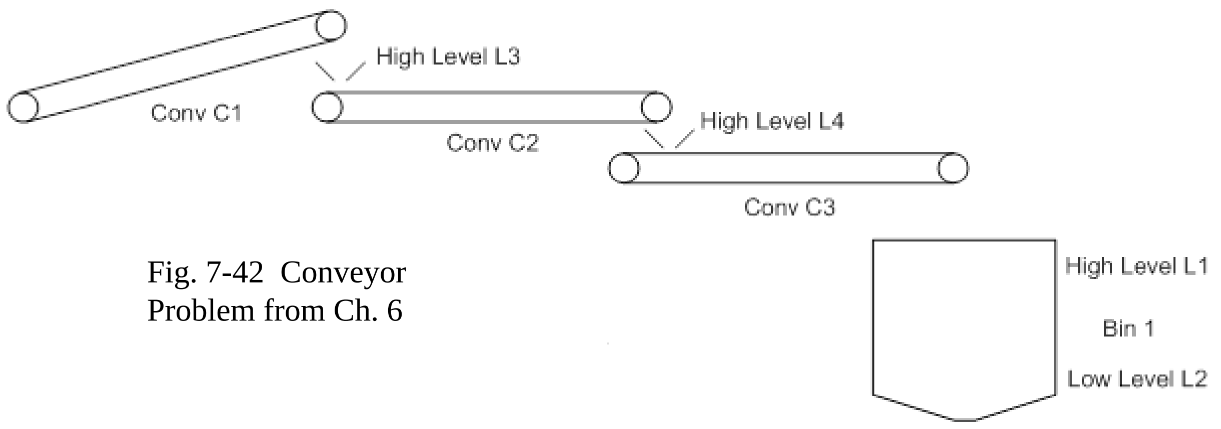
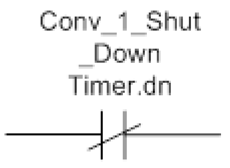


Fig. 7-42 Conveyor
Problem from Ch. 6



To be added to stop portion of
conveyor run circuit

What happens if a conveyor downstream stops working? One sensor is the zero speed sensor from the downstream conveyor. Another sensor is found at the hopper at the beginning of the downstream conveyor. This sensor will turn off the upstream conveyor.

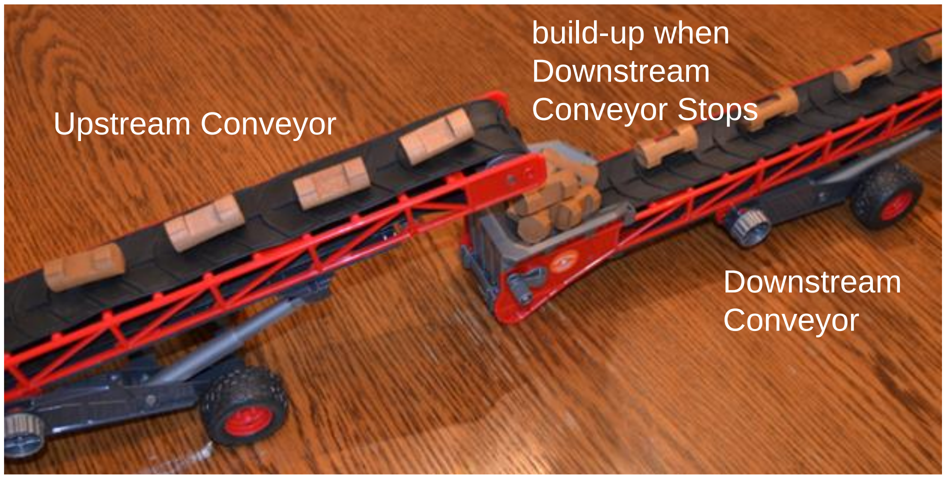


Fig. 7-43 Conveyor Problem with Build-up

The retro-reflective photo-eye is used here to sense a pile-up at the hopper in the conveyor.

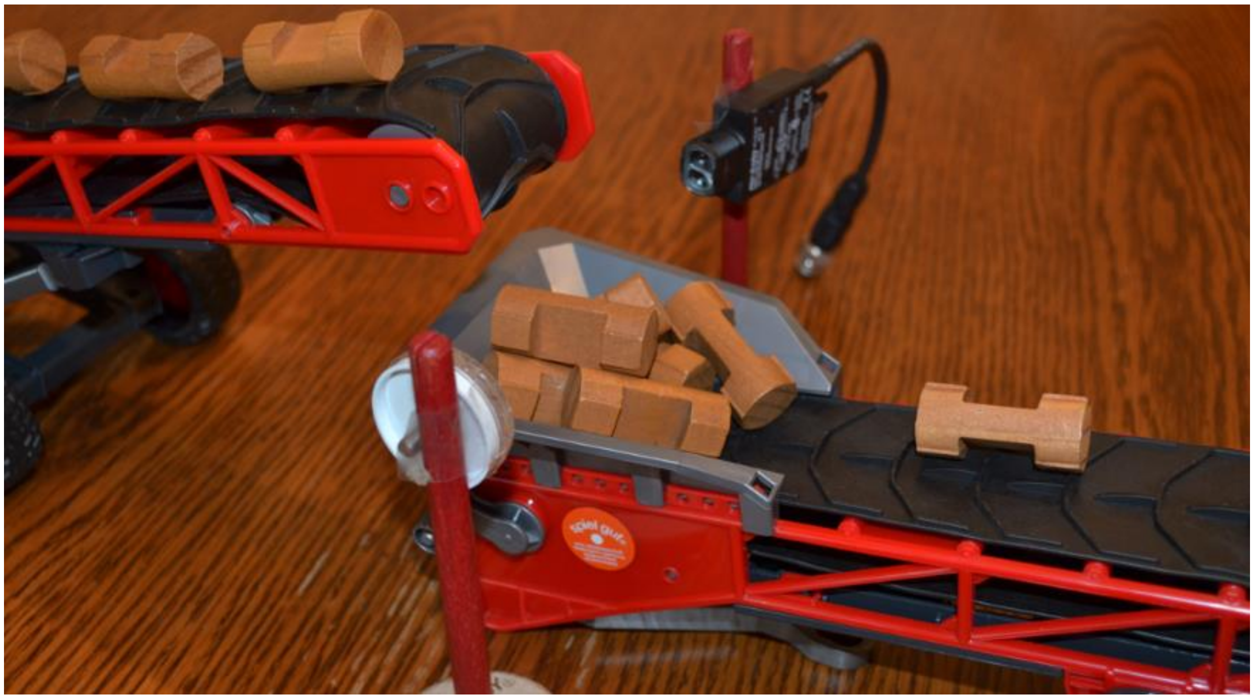


Fig. 7-44a Conveyor Problem with Build-up

Another timer is used to turn off the conveyor if the conveyor is to be turned off after all product has cycled off the conveyor. The retentive timer is used for this function.

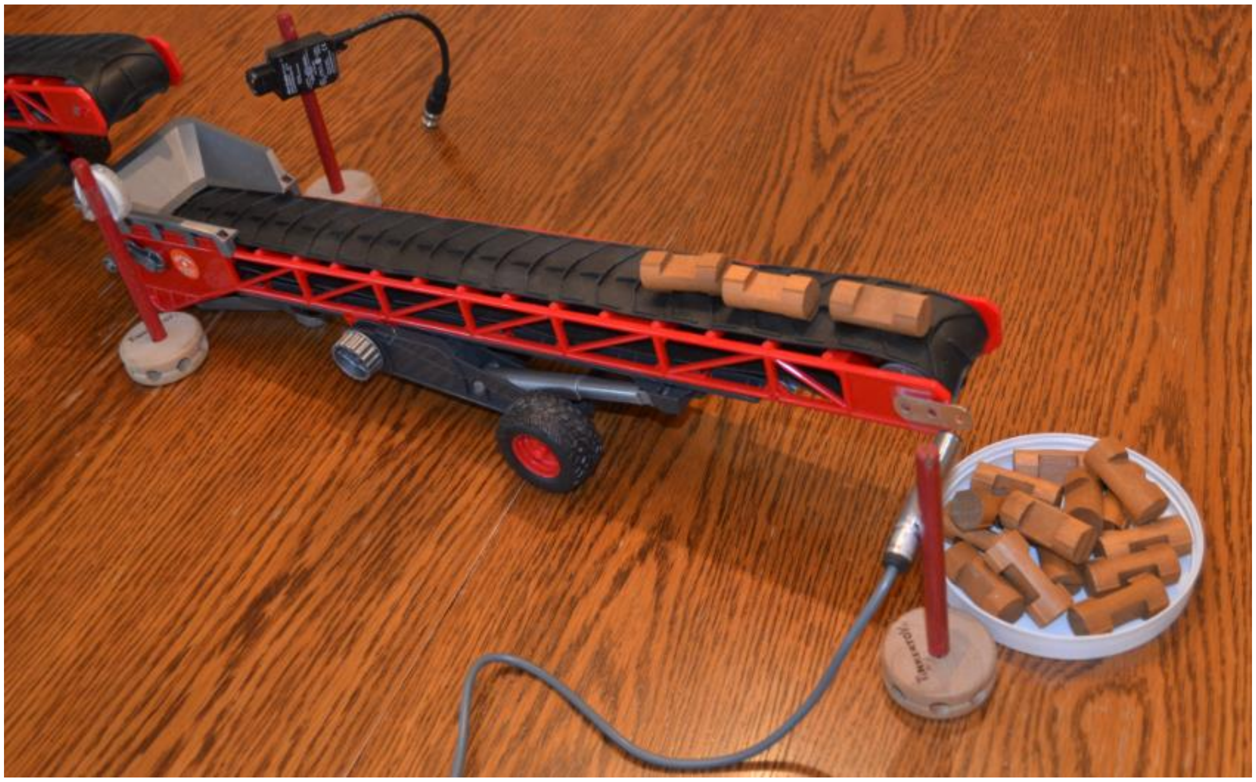


Fig. 7-44b Conveyor Problem with Material Tracking

Clocks and Timers

Timers can be used to design clocks. However, they tend to be inaccurate so real-time clocks have been designed into most PLCs to give very accurate times that time stamp events. Clocks programmed with timers lose accuracy when the timer is reset. Usually a scan or two is lost (added) to the time so when the timer is started a 2 to 20 msec delay occurred. Over a day or week, the timer may keep a fairly accurate clock but with time, the clock tends to get farther and farther from the true time. Of course, if the timer is reset once an hour or day with a master reset signal, the timer may mimic a very accurate clock.

More accurate clocks can be built if very long time delays are used. Avoiding the reset scan allows the highest accuracy in clocks.

Since PLC timer simulated clocks are not accurate for most applications, look to the PLC vendor to install a real-time clock. The real time clock is accurate to the same accuracy of the watch on your arm. It is operated with a crystal timer and set by the program and operated by the clock circuit separate from the scan time of the PLC. Most PLCs have built-in real time clocks.

While the A-B SLC-500 processors do not all contain the Real-Time Clock function, they are defined in the larger SLC 5/03, 5/04, and 5/05. Status register addresses S:37 to S:42 define the Real-Time Clock. These registers are defined as follows:

S:37	Clock/Calendar	Year	Range	0-65535
S:38	Clock/Calendar	Month	Range	1-12
S:39	Clock/Calendar	Day	Range	1-31
S:40	Clock/Calendar	Hours	Range	0-23
S:41	Clock/Calendar	Minutes	Range	0-59
S:42	Clock/Calendar	Seconds	Range	0-59

To disable the Clock/Calendar, write 0's to all clock or calendar words in the range S:37 to S:41.

Combining Counters and Timers

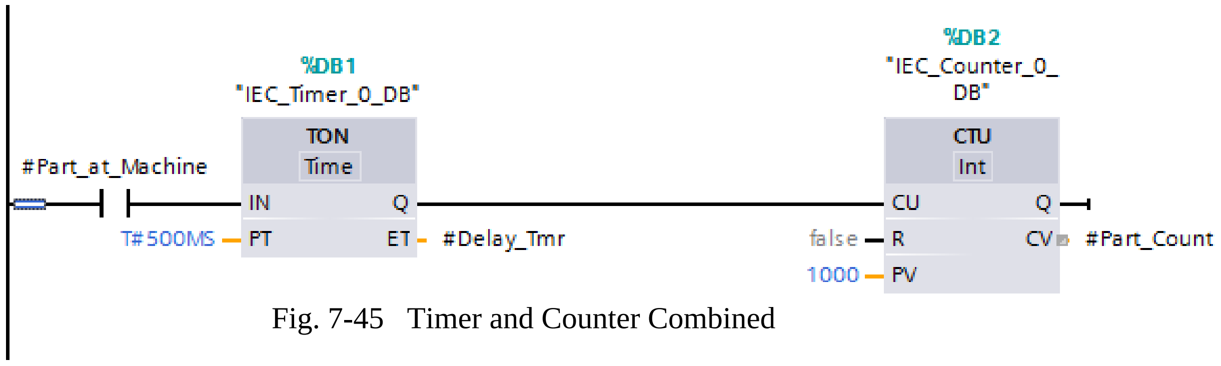
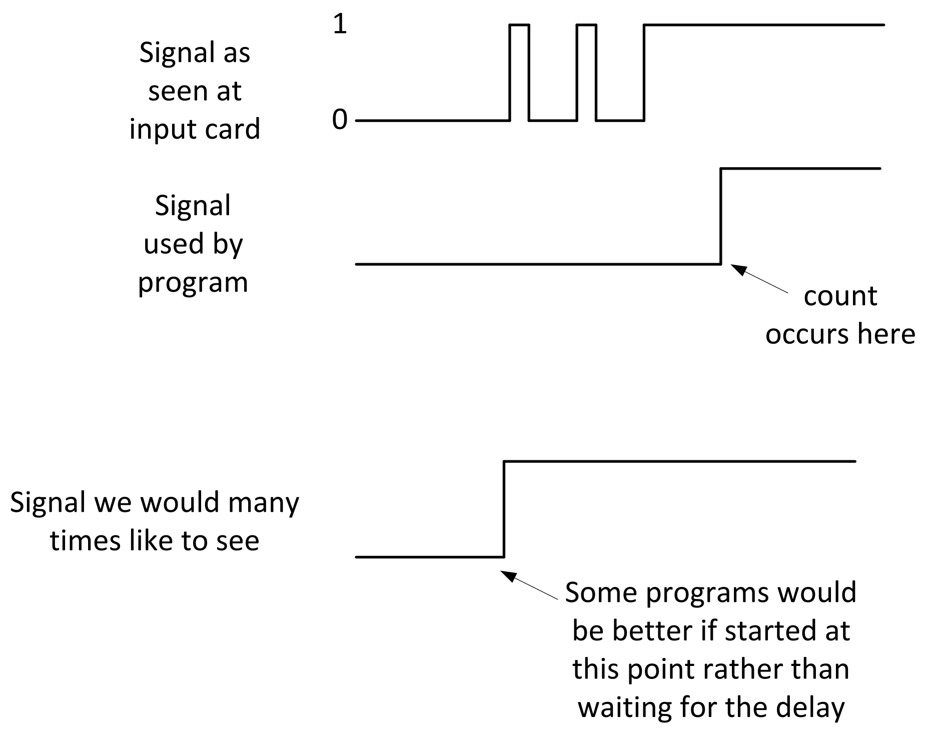


Fig. 7-45   Timer and Counter Combined

The circuit above uses a timer to provide signal conditioning for the counter IEC_Counter_0_DB. If the input Part_at_Machine does not stay on for at least 500*(.001) sec = .5 sec, the timer does not time out and the counter does not increment. Circuits similar to the one above are typical for counting of parts. Proximity switchs are especially noisy with multiple transitions on-off-on for each part sensed and need the type of circuit with timer ahead of the counter protects against false counts. Many automotive plants use timer circuits to buffer proximity switch logic to PLC programs.


Signal as seen at input card
1
0
Signal used by program
count occurs here
Signal we would many times like to see
Some programs would be better if started at this point rather than waiting for the delay

The signal seen at the card many times will be jumpy and not settle for a significant delay period. If the input signal is of this type and many times the prox switch or limit switch exhibits this type of behavior, then a delay such as seen above is needed. The problem with the delay of the TON timer is that the program is delayed from executing even though the switch 'sees' the device needed to start the program. The third signal is the one we would more like to see and may be worth constructing if the program needs to execute immediately with the detection of the switch.

For the counter circuit, the delay is acceptable. For other programs, there should be consideration given for the third signal which would start the user program.

Races using Timers and Counters

A common program to write for timers and counters includes a race condition to determine if a condition is met in a certain amount of time. For instance, the problem to determine if a button has been pushed two times in two seconds is an example of a race. In a race, two events start at the same time. The counter must start counting as the timer starts timing. If the counter reaches the preset prior to the timer reaches preset, the event is determined to be successful. Otherwise, the events are both reset and allowed to start again.

For instance, the programming of the push button to determine if it has been pushed two times in two seconds would resemble:

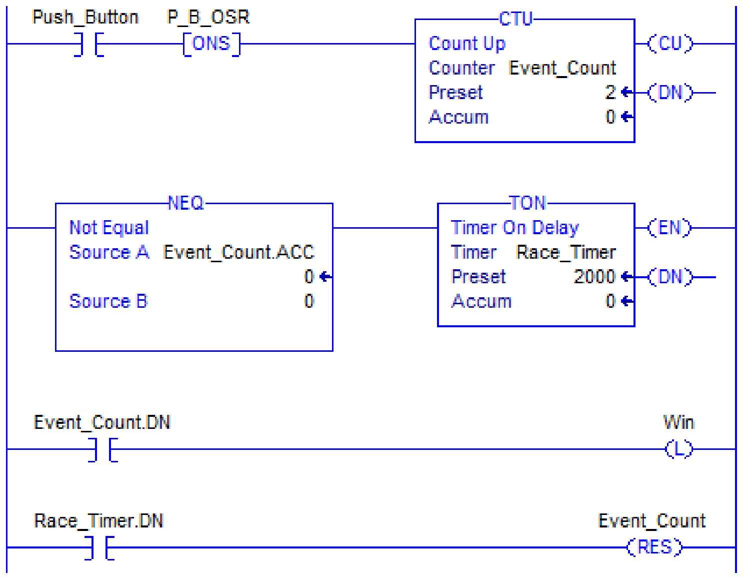


Fig. 7-46 Race Using Timer and Counter

In the circuit above, the counter must reach the preset count of 2 before the timer reaches 2 seconds. The timer starts as soon as the counter is determined to have a non-zero accumulated count. As soon as the counter increments to 2, the latch is set signifying the race was a success. If the timer reaches preset first, the counter is reset and the process begins again.

A Project for Stacking or Grouping Boxes

Example of Counters and Timers in Industrial Application

1. Forklift driver places a box on a conveyor:

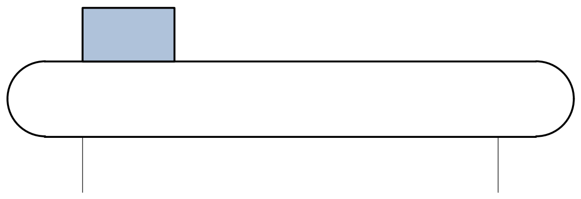
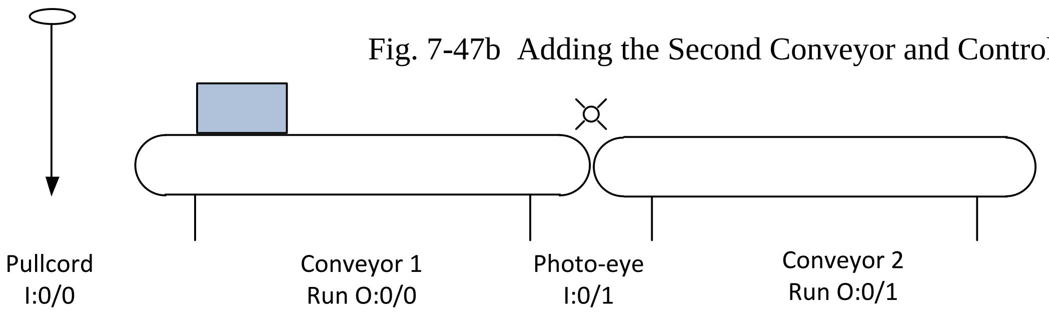


Fig. 7-47a Single Box on Conveyor

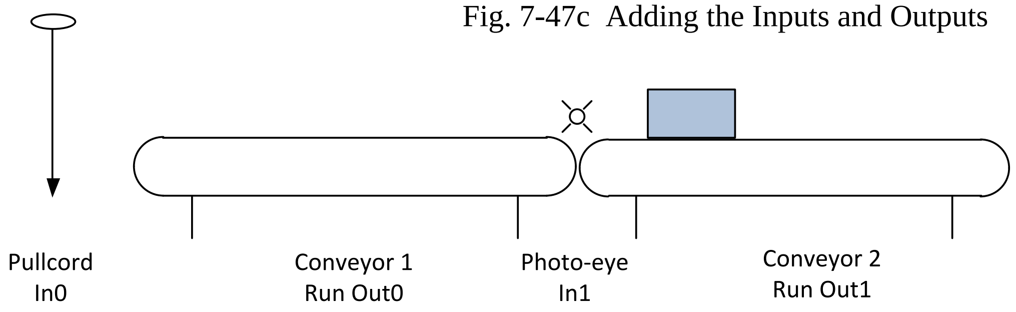
2. The forklift driver pulls a cord energizing an input and starts the box moving to a second conveyor that gathers/groups a set number of boxes:

Fig. 7-47b Adding the Second Conveyor and Controls



Conveyor 1 runs from when the pull cord is energized until the photo-eye sees the box in front of the eye. It then continues to move until the eye is no longer covered. Conveyor 2 moves from the instant the photo-eye sees the box until a delay after the box no longer is seen.

Fig. 7-47c Adding the Inputs and Outputs



The second conveyor doesn't stop until a delay after the photo-eye does not see box.
Program for Control of Boxes on Two Conveyors:

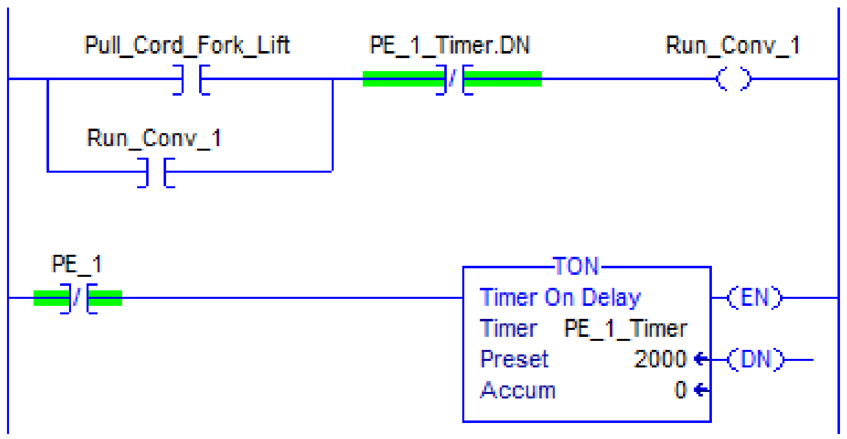


Fig. 7-47d Program for Moving Box to Second Conveyor (A-B)

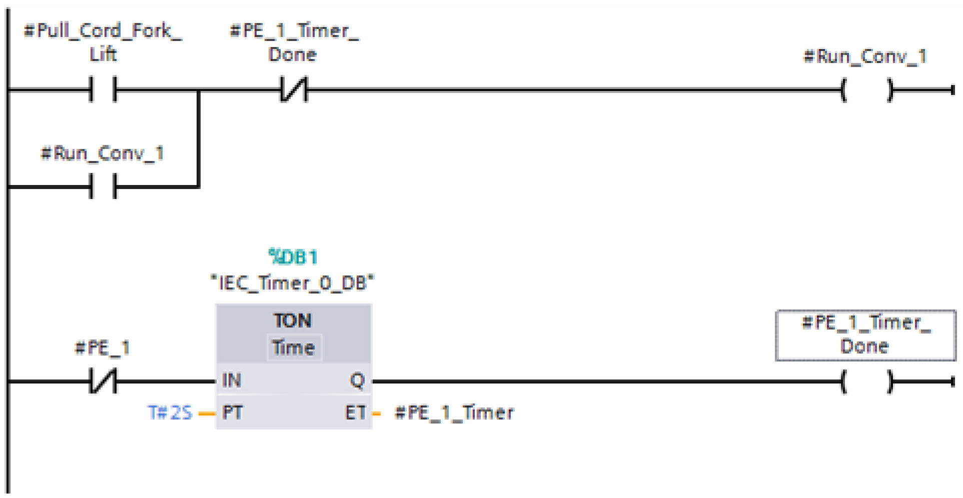
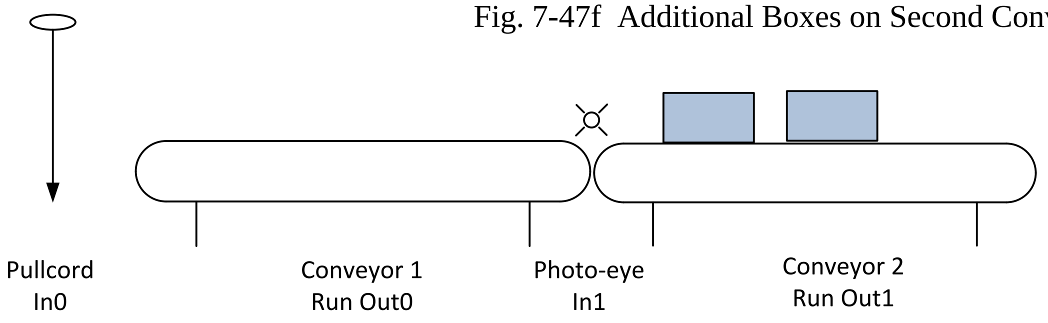


Fig. 7-47e Program for Moving Box to Second Conveyor (Siemens)

Use of the Off-Delay for movement of the box past the photo-eye by a set-time positions the box on conveyor 2. The second box placed on conveyor 1 would be positioned on conveyor 2 similar to the first box.

Fig. 7-47f Additional Boxes on Second Conveyor



Boxes are grouped on conveyor 2 until a preset count is reached. The preset in the Up Count Counter sets the limit of number of boxes to be collected on the second conveyor.

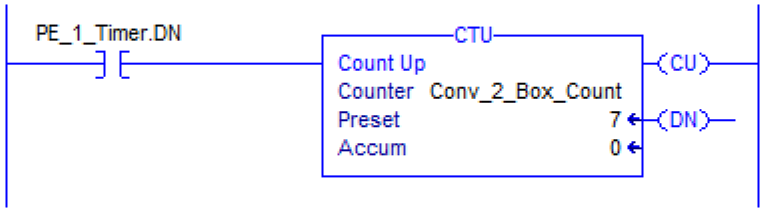


Fig. 7-47g Logic to Count Boxes until Grouping Complete (A-B)

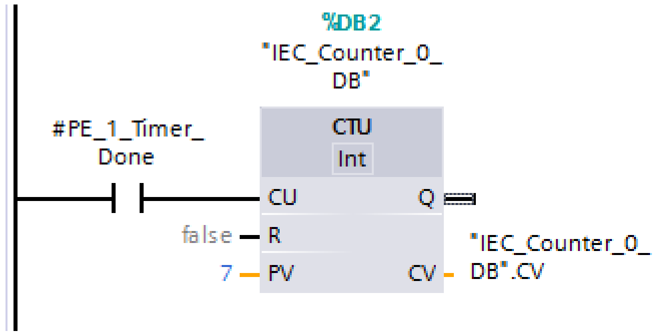


Fig. 7-47h Logic to Count Boxes until Grouping Complete (Siemens)

When this counter counts to preset, the number of boxes on Conveyor 2 has counted to 4 and the boxes are now ready to be moved as a group to a third conveyor.

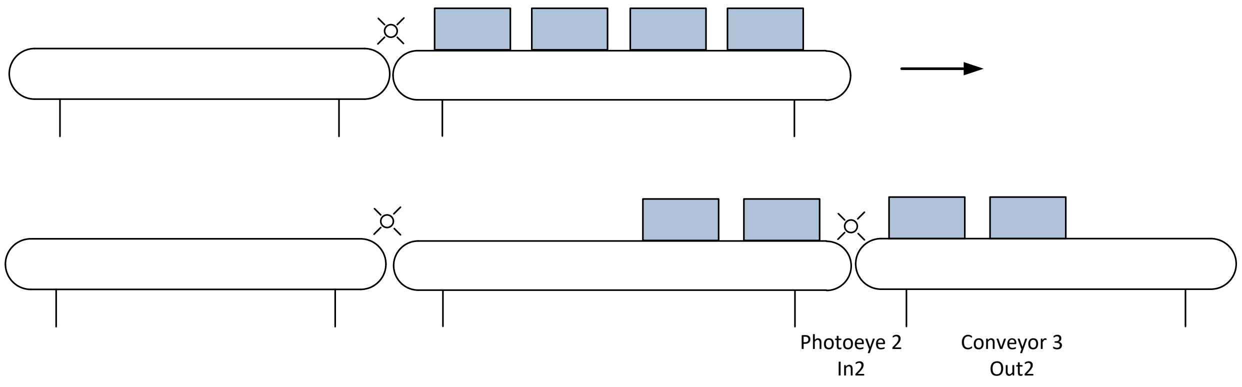


Fig. 7-47i Boxes Moving to Third Conveyor

The question about modes will be discussed in a later chapter. To count the boxes off Conveyor 2 requires using a second counter or a down counter with the same address as the up-counter for counting boxes onto conveyor 2.

Timers and counters are used together in a number of different kinds of programs. These show just a few of their many uses in automation programming

Stepping Program for Machine

The following machine is designed to move a part down a conveyor and back to home. It was originally introduced in Ch. 6. In this chapter, we add a timer at the end of travel to give the conveyor some time to stop before reversing.

In the forward movement, the part is to be sprayed after being sensed by the photo-eye in the middle of the conveyor. Once the part moves to the end of the conveyor, the conveyor reverses and the part is moved back to the home position to be removed. A start button begins the action.

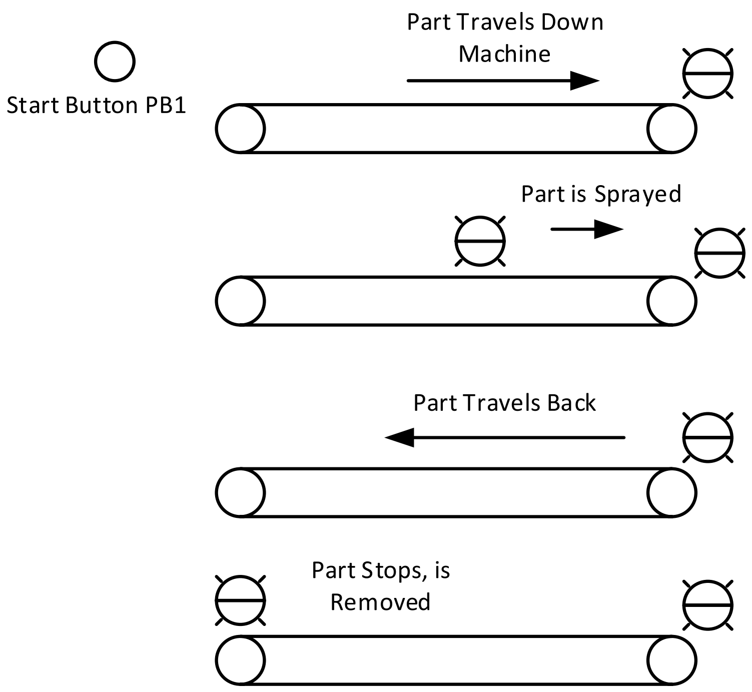
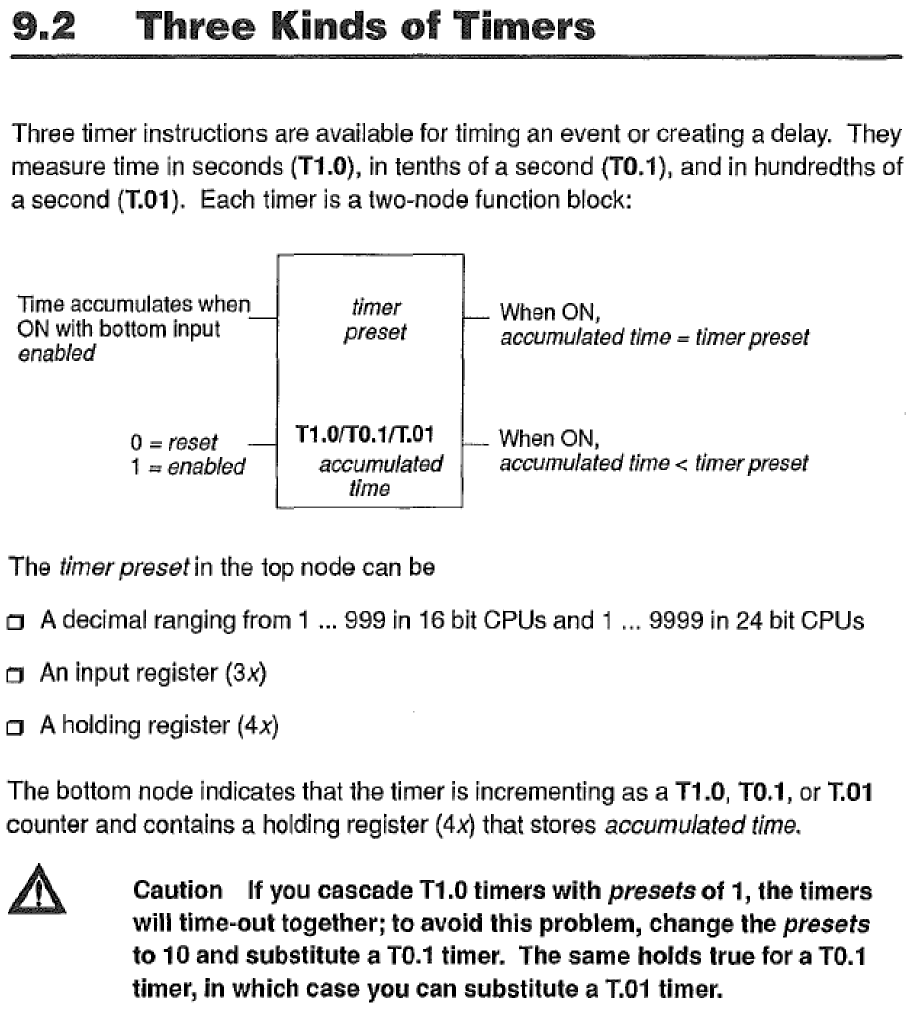


Fig. 7-48a

This program requires the operator start the movement by pushing a button. This action sets the machine in motion. Logic can be developed using seal circuits for forward motion, reverse motion and over-all motion. The spray action begins by the part passing the middle photo-eye. The timer is added to let motion come to a stop before the conveyor reverses.

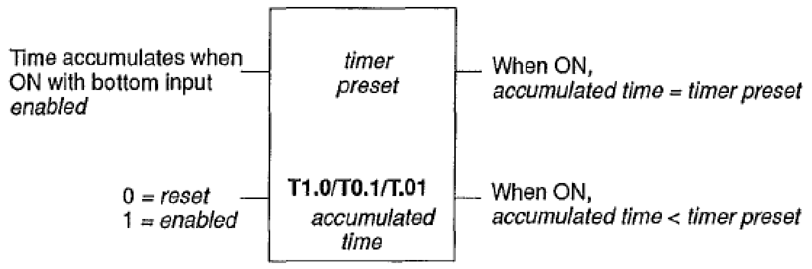
After All This

The original PLC manufacturer, Modicon, came to terms with timers in a much simpler manner than either Allen-Bradley or Siemens. They used a **single** block that could be programmed either as an **On Delay** or **Off Delay**, with either **Non-retentive** or **Retentive** capabilities:



9.2 Three Kinds of Timers

Three timer instructions are available for timing an event or creating a delay. They measure time in seconds (**T1.0**), in tenths of a second (**T0.1**), and in hundredths of a second (**T.01**). Each timer is a two-node function block:

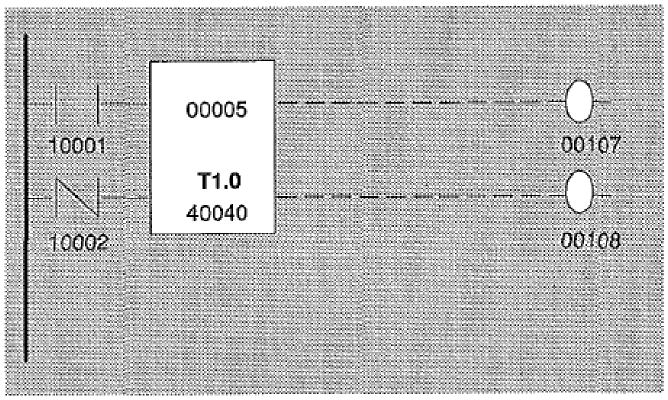

Time accumulates when ON with bottom input *enabled*
timer preset
When ON, *accumulated time = timer preset*
0 = *reset*
1 = *enabled*
T1.0/T0.1/T.01 *accumulated time*
When ON, *accumulated time < timer preset*

The *timer preset* in the top node can be

- A decimal ranging from 1 ... 999 in 16 bit CPUs and 1 ... 9999 in 24 bit CPUs
- An input register (3*x*)
- A holding register (4*x*)

The bottom node indicates that the timer is incrementing as a **T1.0**, **T0.1**, or **T.01** counter and contains a holding register (4*x*) that stores *accumulated time*.

Caution If you cascade T1.0 timers with *presets* of 1, the timers will time-out together; to avoid this problem, change the *presets* to 10 and substitute a T0.1 timer. The same holds true for a T0.1 timer, in which case you can substitute a T.01 timer.



The example above assumes that 10002 is closed (timer enabled) and that the value contained in register 40040 is 0. Because 40040 does not equal the *timer preset* (5), coil 00107 is OFF and coil 00108 is ON.

When 10001 is closed, 40040 begins to accumulate counts at 1 s intervals until it reaches 5. At that point, 00107 is ON and 00108 is OFF.

When 10002 is opened, 40040 resets to 0, coil 00107 goes OFF, and 00108 goes ON.

Note If the *accumulated time* value is less than the *timer preset* value, the bottom output will pass power even though no inputs to the block are present.

The above is from the Modicon 984 Systems Programming Manual and is the entire description of timers for the Modicon 984 PLC. Similar timers were available for the Modicon 484 and 884. How did they do it, provide all the timing functions necessary with just really one block? The block is very versatile and can be used in a number of different modes and can be cascaded left-to-right. This points out that the complete list of different timing functions is not really all that necessary.

Most of my programming life was spent only knowing how to program the TON timer for a specific PLC. With this one timer, one can get most of the functionality of the other timers with only a few modifications. Is it worth looking into all the various timer types? Usually only if the timers are used by someone else in a specific program, do we need to refer to the manual for those timers other than the TON timer.

Summary

The chapter on Timers and Counters introduces these devices and gives practical examples of their use as well as timing diagrams for applications utilizing their specific qualifications. Examples show a need to be able to use timers in seal or memory circuits. Use of timers and counters together is discussed.

Siemens and Allen-Bradley based timers and counters are shown and examples of each type are explained. While one might be interested in the specific attributes of each type, it is possible to walk through the chapter and understand the various timers without learning how to use each of the various types. To use the TON or on-delay timer in almost all occasions is possible with a rare need to use alternates. This approach has been used by the author and has worked in industry. This advice is to work extensively with the TON timer and learn its characteristics and use the timer for as many applications as possible. If another type is absolutely necessary, then read about it and use it as necessary. A search of the problems at the end of the chapter reveals that they may all be solved with the TON timer.

An example of a conveyor system counting boxes is discussed and the use of timers and counters to control the movement on the conveyors is shown.

One difference between the Siemens Up-Down Counter and the A-B Counters is that Siemens uses a different counter instruction for up, down and up-down. A-B uses different instructions for up and down but combines the two instructions to allow an up-down counter. The tag for the up-down counter is the same tag. This may confuse at first but should be tried in the labs to see the results. A troubleshooting note on the A-B up-down counter is that one may program one up-count and one down-count counter with the same tag but not three or four counters with the same tag. This will lead to unexpected results and a troubleshooting dilemma.

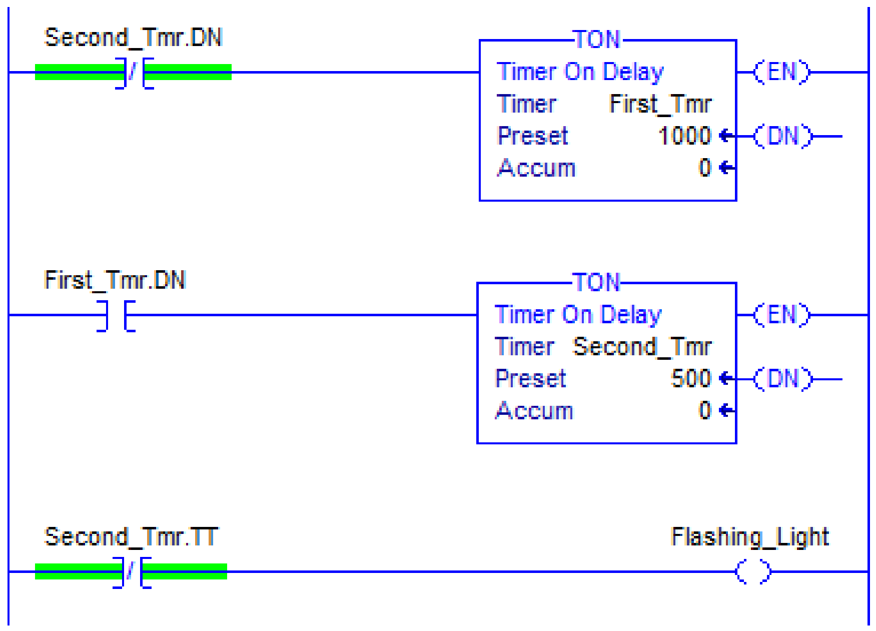
While Siemens uses multiple types of timer with the box structure and the IEC type, it is best to settle with the box type and stack them left to right on the screen. A-B allows some stacking of boxes from left-to-right as well. The student should be willing to try to stack instructions and see if the processor will allow this action.

Exercises

1. De-bounce Circuit Leading Edge:

   Design a de-bounce circuit which will override the bounce of a limit switch on the leading edge only. Allow no input that is not on for at least .5 second. Use Limit_switch_1 as input.

2. For the following circuit, answer the questions below:



   a) How long does the flashing light stay on?

   b) How long does the flashing light stay off?

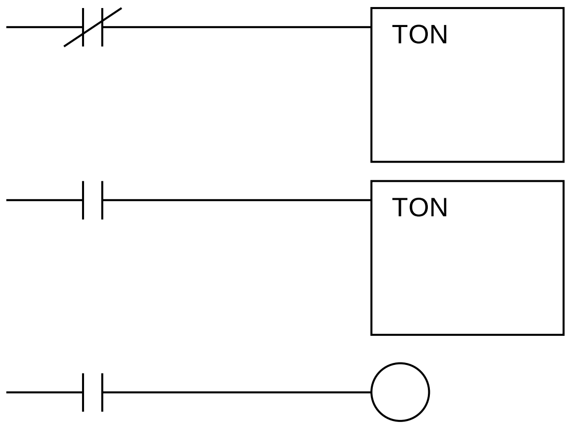
   c) Modify the circuit so that the output is on for 1.3 seconds and off for 1.2 seconds.

   d) Modify the circuit so that the output is on for 2.5 seconds and off for 1.7 seconds.

3. Design a program that turns on a latch when a button is pushed three times in less than 2 seconds.

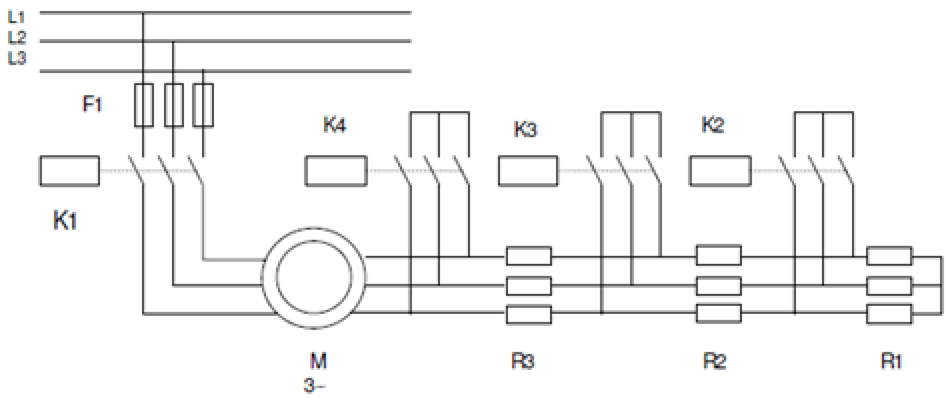
4. A low pressure switch is to be added to a fan starter circuit. Assign inputs for stop, start and low pressure. If the fan loses pressure, then it is to shut down. Design a start/stop seal circuit that stops on low pressure. Remember that when the fan is off, pressure is low.

5. Complete the following rungs to give a light that flashes on for 1.3 seconds and off for .2 seconds using Siemens' timers:



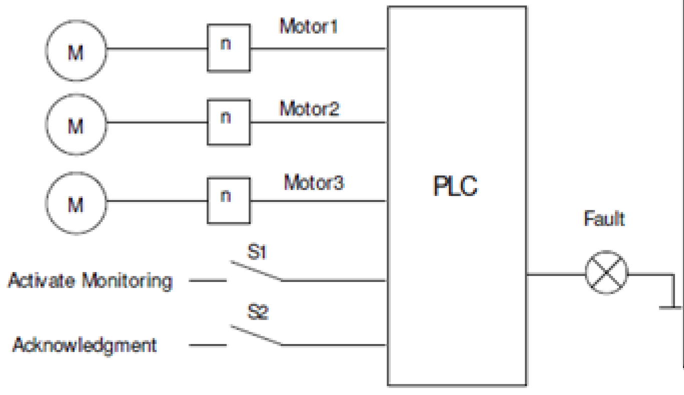
Rewrite the flashing circuit using Siemens FBD, using A-B's FBD.

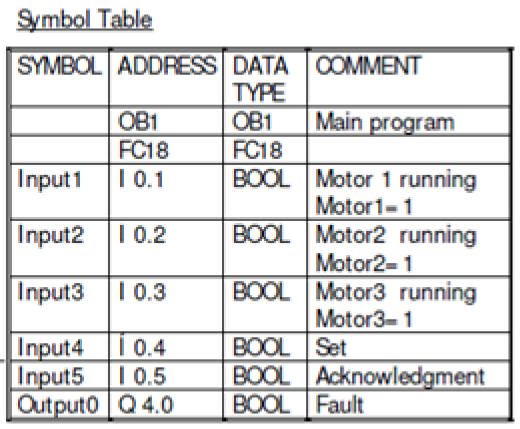
6. To start asynchronous wound motors, resistors are connected in the rotor circuit to avoid a high inrush current. After pushing the start button S1, the line relay is closed. Then relays K2, K3 and K4 are closed, each after a time delay of 5 seconds. Write the program to start M3.



7. A group of three motors should be controlled. Each motor is equipped with a revolution monitoring device. If the motor turns, the sensor indicates "1" (otherwise "0"). The switch S1 activates the monitoring circuit. The failure indication (fault) is to light in the following cases:
    a. If two of the three motors failed longer than 10 seconds
    b. If all three motors failed

The fault light is to stay lit until a reset switch is activated (acknowledged).




Symbol Table

SYMBOL	ADDRESS	DATA TYPE	COMMENT
	OB1	OB1	Main program
	FC18	FC18	
Input1	I 0.1	BOOL	Motor 1 running Motor1= 1
Input2	I 0.2	BOOL	Motor2 running Motor2= 1
Input3	I 0.3	BOOL	Motor3 running Motor3= 1
Input4	I 0.4	BOOL	Set
Input5	I 0.5	BOOL	Acknowledgment
Output0	Q 4.0	BOOL	Fault

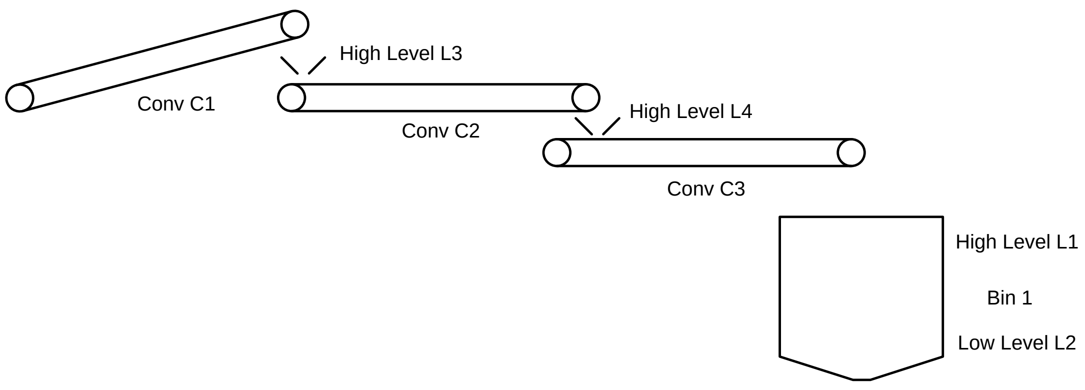
8. Count and Time Program:

   How many parts per minute are going past a certain process point? The counter is pulsed for each part going past the sensor, which is connected to Input 1. The counting begins and the timer starts timing through its 60 second time interval. At the end of 60 seconds, pulses continue but do not affect the counter. The part count for the minute will remain in the counter register until switch S is opened. Then the counter and timer are reset. When S is closed again, another 60 second interval occurs with another part count saved at the end of the 60 second period.

   How would you make the process continuous with the last 60 second part count the one read?

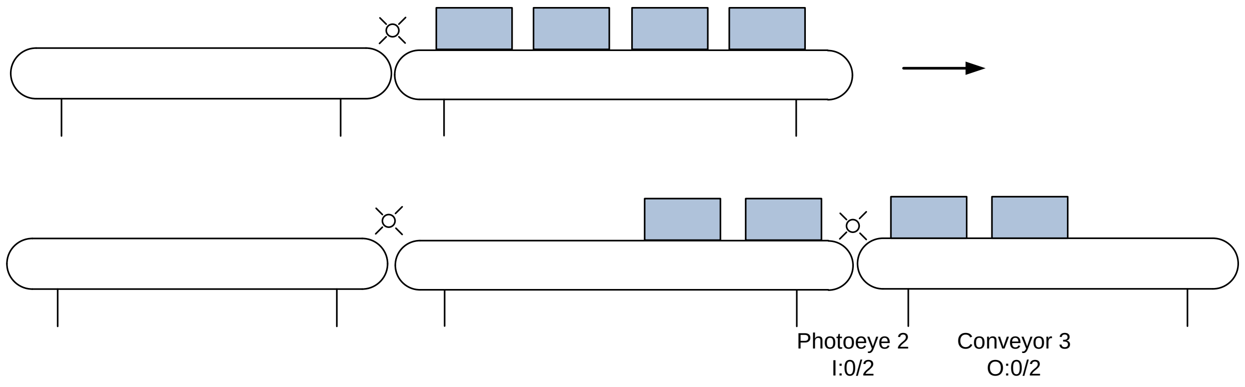
   How would you make the process show the average of the last 10 minutes' part-count, the last hour?

9. Rewrite the controls for the diagram below using the new actuator, function/state table:



Sensor	Function/State	Signal Assignment
L1	Bin 1 High Level	0
L2	Bin 1 Low Level	1
L3	Conv C2 Hopper High Level	0
L4	Conv C3 Hopper High Level	0
P1	Conv 1 Running	1
P2	Conv 2 Running	1
P3	Conv 3 Running	1
Actuator	**Function/State**	**Signal Assignment**
C1	Conv C1 Run	1
C2	Conv C2 Run	1
C3	Conv C3 Run	1

10. When this counter counts to preset, the number of boxes on Conveyor 2 has counted to 4 and the boxes are now ready to be moved as a group to a third conveyor.

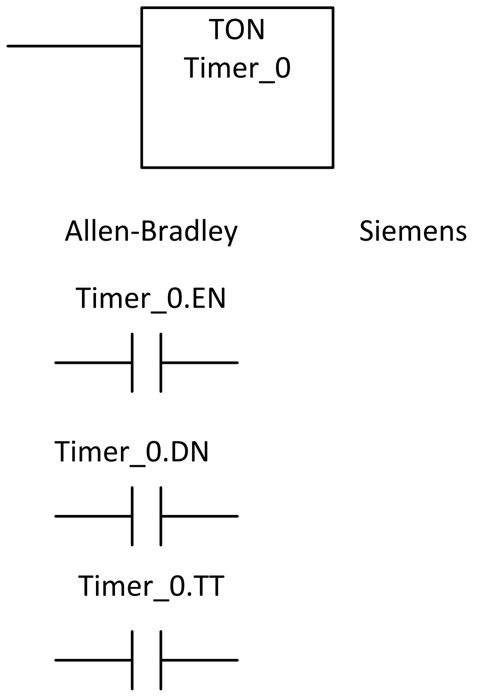


To count the boxes off Conveyor 2 requires using a second counter or a down counter with the same address as the up-counter for counting boxes onto conveyor 2. Write the code that moves the boxes off the conveyor.

11. The microwave is to be set for high for 1:30 (one minute, thirty seconds) to cook my oatmeal. The oatmeal sometimes boils and bubbles out of the bowl if left on continuously for 1:30 but the time is to be held at 1:30. Design a control circuit to guarantee 1:30 cooking time with no over-flow mess. Use sensors as needed and a start button and ready light to complete the control scheme. Design a function/state – signal assignment table to accompany your program. This is a retentive timer application. Use a retentive timer from either A-B or Siemens.

12. Design a de-bounce circuit which will override the bounce of a limit switch on the leading edge only. Allow no input that is not on for at least .5 second. Use Limit_switch_1 as input.

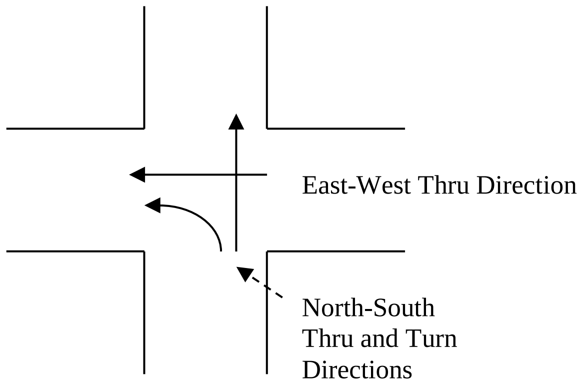
13. Siemens and A-B have different contact descriptors for the same functions. In the following, the A-B contacts are listed. Fill in the Siemens' equivalent:



Lab 7.1 The Traffic Intersection

A traffic intersection has the following three lane assignments:

- East-West Thru
- North-South Turn
- North-South Thru



Two sets of traffic lights are found for each turn direction although the lab uses only one set. Each turn direction has a set of three lights as follows:

🔴	Red
🟡	Yellow
🟢	Green

Although traffic intersection logic tends to be very complicated in order to provide fool-proof operation of the traffic intersection, a simplified chart of the operation of the lights can be used to program the lights and operate the intersection. Each interval is an interval of time and after the last interval, the process repeats from the top. The intersection's operational chart:

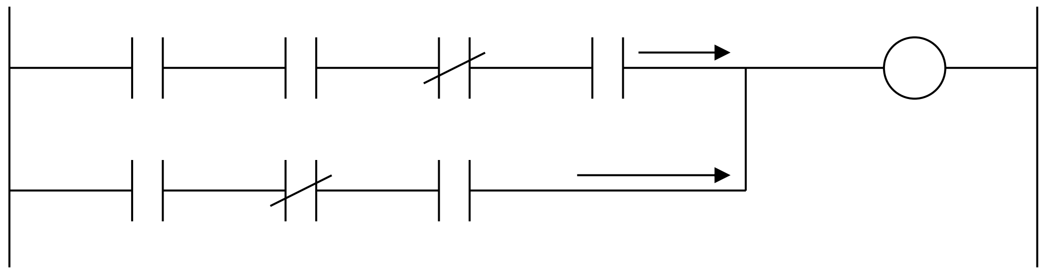
Interval	N-S Thru Lane	N-S Turn Lane	E-W Thru Lane
1	Green	Red	Red
2	Yellow	Red	Red
3	Red	Green	Red
4	Red	Yellow	Red
5	Red	Red	Green
6	Red	Red	Yellow

This lab consists of programming the nine lights to cycle through the proper sequence to control traffic flow at the intersection described above.

A helps program can be found accompanying this lab to start the process of setting up timers, especially to cycle and repeat a sequence. Notice that two timers can be set up the same as six or more timers to control such events as a flashing yellow or flashing red light. The same two timers can be used for all such flashing functions. Thus, two timers are all that are needed if a flashing sequence is needed.

Notice the EN, DN and TT contacts found with every timer. Timers start with T4:0 and proceed upward. Use the TON timer, or the On Delay Timer. Addressing for timer contacts is T4:0/DN, T4:0/TT, T4:0/EN. Refer to the Allen-Bradley Instruction Reference Manual for more detailed information about the use of timers. Also, look at the helps program to view a basic cycling program that works.

Also, the outputs must be programmed. Remember that only <u>one</u> output should be programmed for each light. Outputs are programmed to allow multiple branches to turn on the selected output. For instance, the top or bottom branch of the rung would allow the output to turn on. This is a parallel branching function so either of these branches would turn on the output.



Options for Lab 7.1

Lab 7.1 A Add a selector switch to delete intervals 3 and 4 during a rush hour.

Lab 7.1 B Add a selector switch to blink N-S lanes yellow, E-W lanes red for late-night.

Lab 7.1 C Add a short time delay between intervals 2-3, 4-5, and 6-1 while all lights are red.

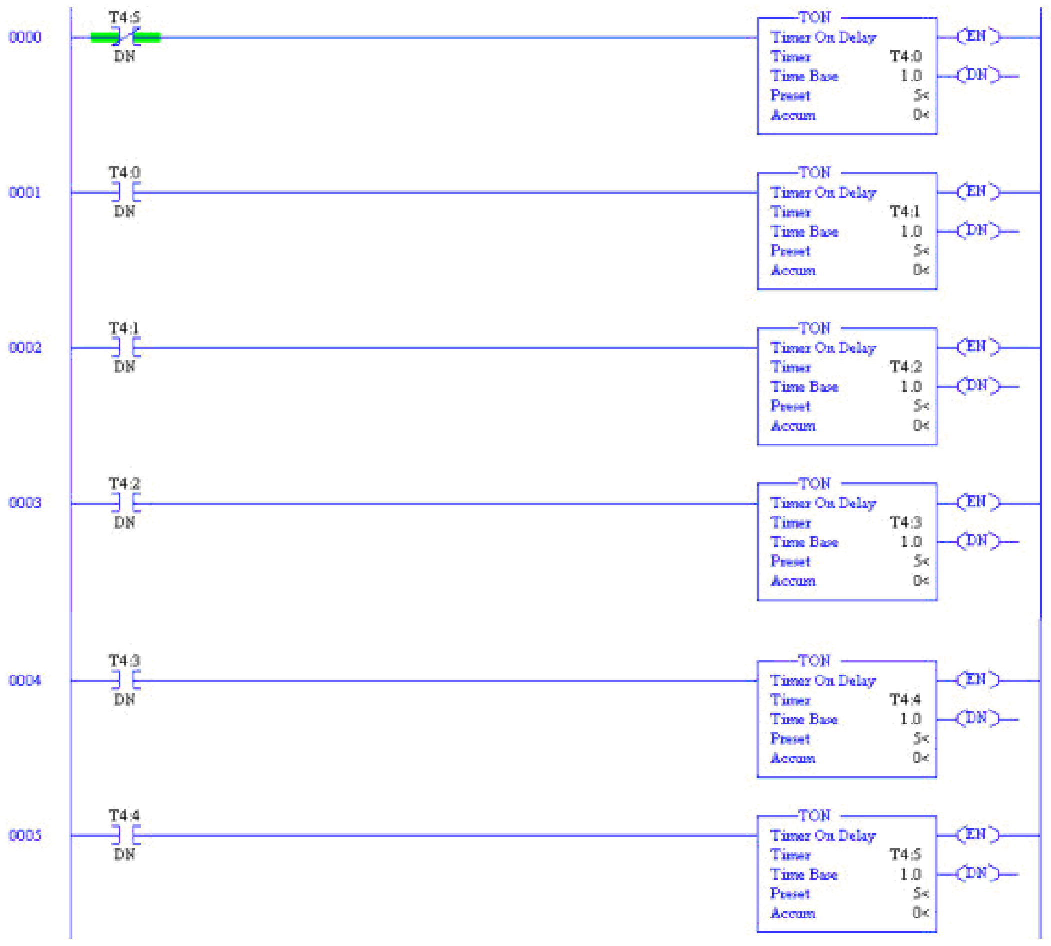
Lab 7.1 D Add a push button to allow pedestrians to walk in all directions for an interval of time while all lanes are red.

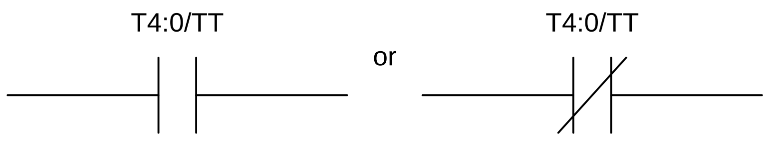
Lab 7.1E Add a push button that acts as a button in the pavement that will only allow a turn lane signal if there has been a car activate the turn signal in the time prior to the turn signal’s position in the cycle.

Lab 7.1F Add two switches imbedded in the road to sense when there is a back-up of cars wanting to use the turn lane. If there is a back-up, use a longer time preset for the turn lane. If the switches turn on in rapid succession, then there is no back-up. If the switches turn on but not in rapid succession, there is a back-up.

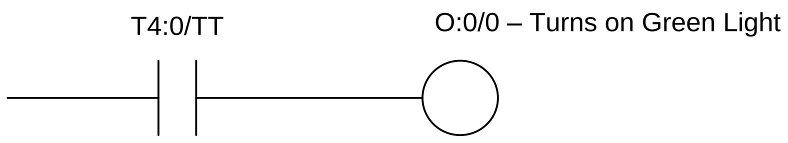
Hints for Lab 7.1

Program the timer circuit below to give the intervals needed. View the T4 Timer Table on-line with the processor running. Notice the TT and DN contacts. Use the TT (or EN or DN) contacts in logic to turn on the lights in order.

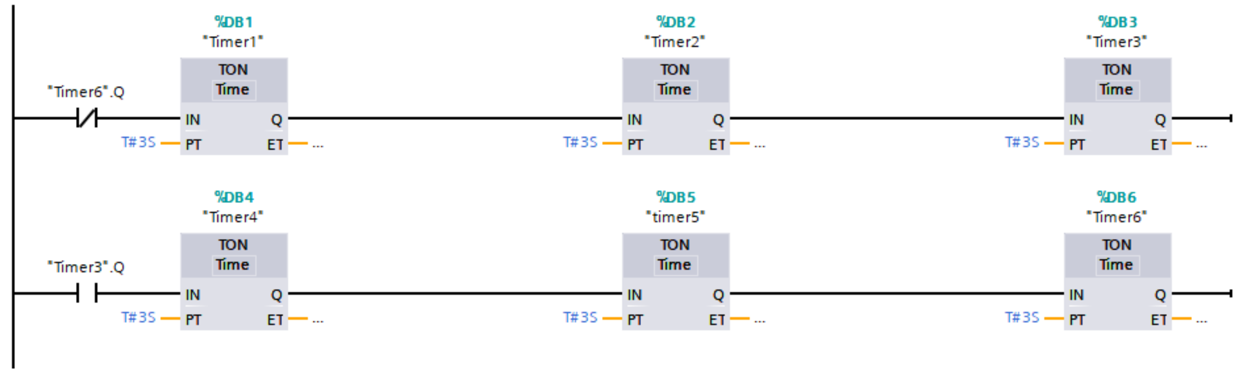




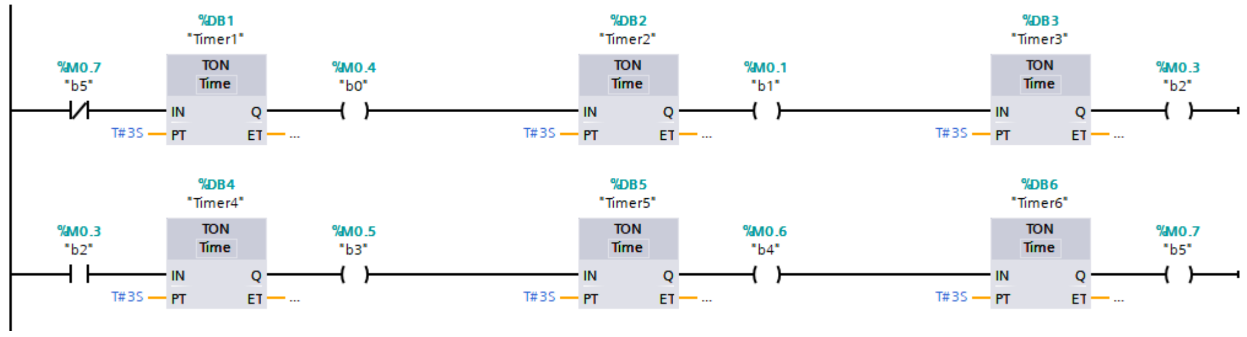
The logic to provide outputs and the outputs themselves may be combined. For instance, to turn on a specific traffic light from the program above, program the following:



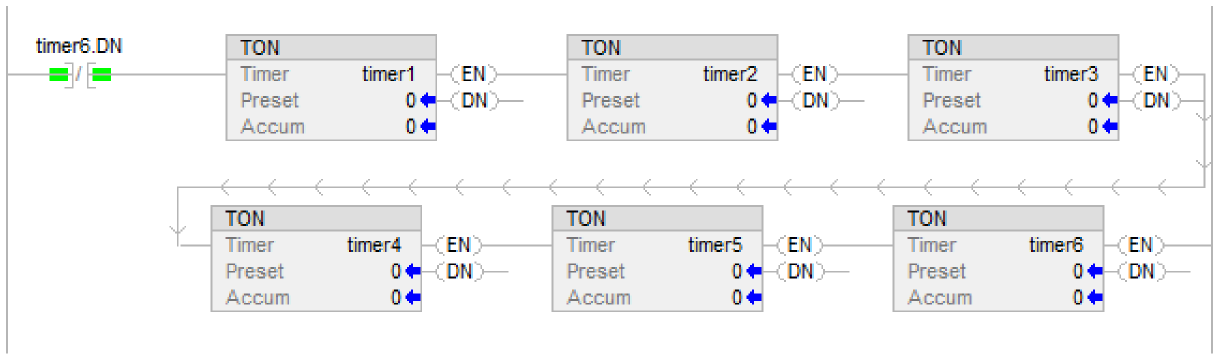
The following Siemens program will provide the same function as the A-B program above: You may use either the .Q bit or define timer output coils at each time interval.



The following Siemens uses the coils instead of .Q bit.

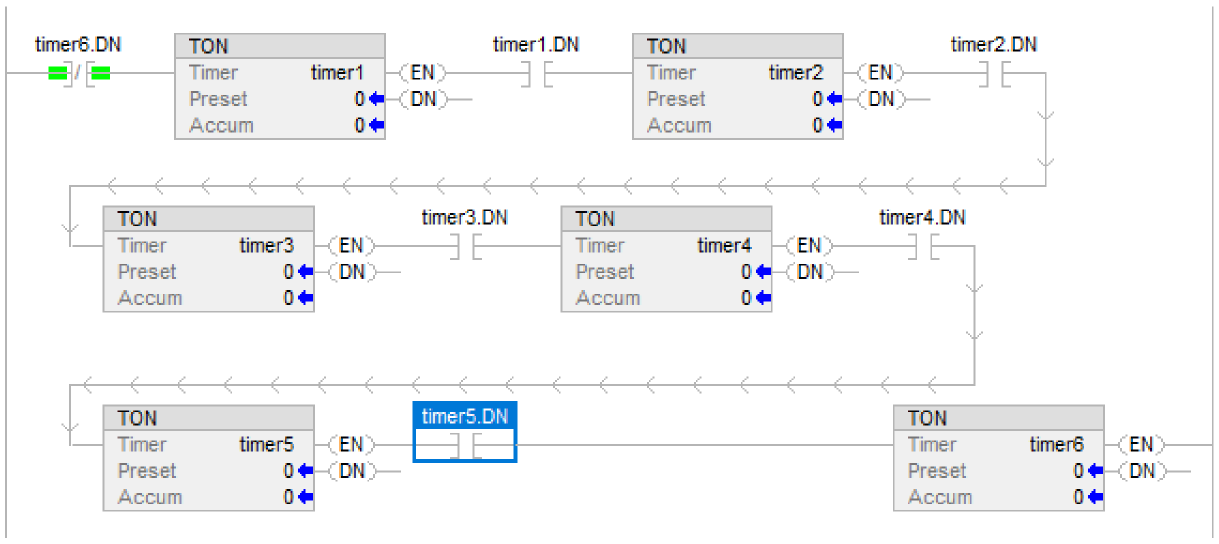


You can try the following but it will not work properly. Do you know why?



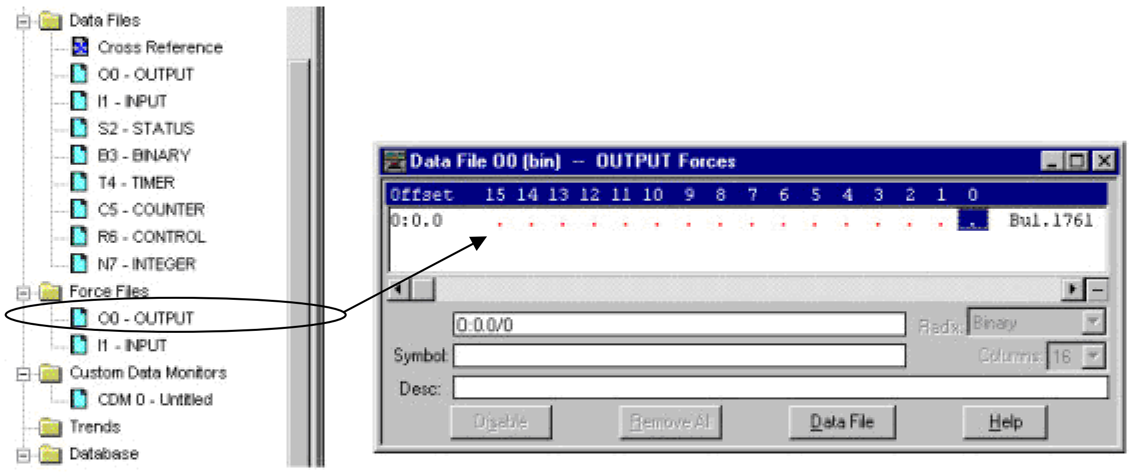
The above A-B rung gives what result? Why?

Instead, use the following. Why will this work and not the above?



Using the Force Table

With inputs, to verify a working input was relatively easy. Simply wire through the button to the input terminal. If the input is working, the LED on the input block lights when the pushbutton is energized. With outputs, life is not as simple. Both the wiring must be correct and the program work as well for the lights to function properly. To divide the task into two parts, RSLogix 500 provides a force function to overwrite the program and force the output bits on. The Force Table may be accessed through the System Tree.



Force on outputs as necessary to turn on lights. When done checking lights, do not forget to delete all forces.

Lab 7.2 The Cash Register

Design a simple cash register similar to one found at McDonald's or Burger King. To do this, determine a menu of five or six items from the restaurant. Also, include a Total button or a clear button or possibly both. Also, include a means for backing out of a mistake without starting over from zero. Display the cost of the total order in the PLC at an address in the data table. Use Floating Point Math with two decimal places.

For example:

Whopper Combo	Whopper	Cancel Last
Whopper Dbl Combo	Fries	New Order
Whopper Jr Combo	Drink	Total/Tax/Optional

Find the approximate prices from a McDonald's or Burger King for the items you choose. When an item is entered, its count is incremented automatically by one. If a button is entered multiple times, the count is incremented to display the total count. If a mistake is made, the attendant must be able to back up at least one entry and erase the last item or decrement that item by one.

Display the final total in the PLC (not on the display of the trainer).

Options to the lab:

Lab 7.2 A Add logic for "To Go" order so that 6.25% tax is added if not "To Go".

Lab 7.2 B Add lights to buttons so that when an entry is made, the light lights.

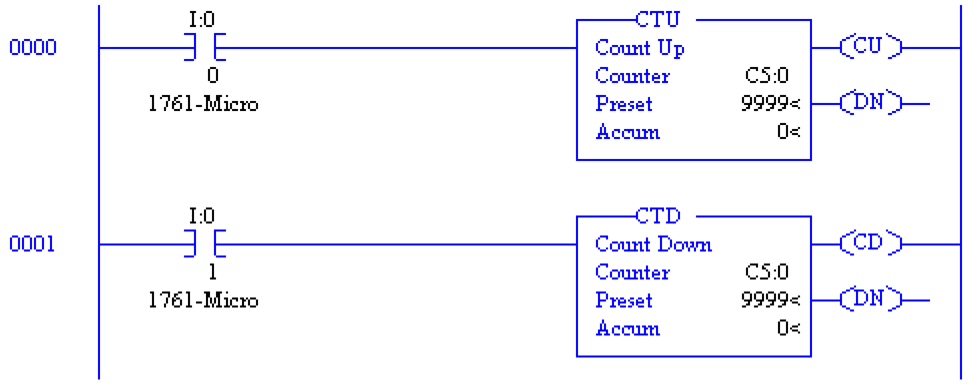
Lab 7.2 C Add logic to keep track of total number of each entry for the day.

Lab 7.2 D Calculate profit for the day using your own profit numbers for each entry.

Lab 7.2 E Automatically recognize that the entry of the individual items such as Whopper, Fries, and Drink will be given the price of the Whopper Combo instead of the individual prices.

Hints to the base lab:

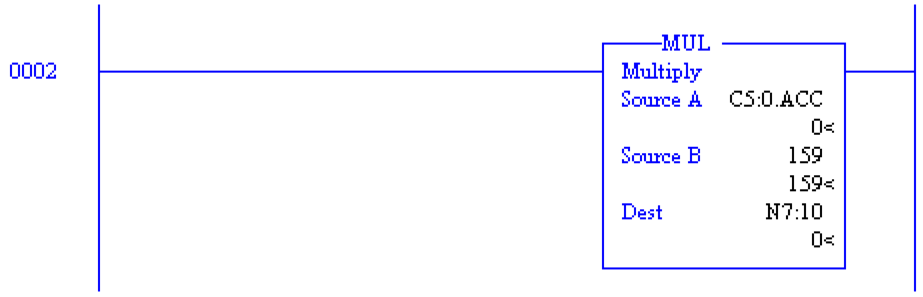
Notice that counters may be referenced as either Count Up or Count Down. If the count is counting up, the count is incremented in rung 0000. If the count is counted down, the count is decremented in rung 0001. Individual inputs are used to increment each product choice. However, to decrement the count, a separate button labeled “Cancel Last” is used. This button must remember the last product chosen and decrement that item. Use the logic in chapter 6 “Relay Instructions” to remember when a button was pushed.



The circuit above is for trial purposes only. Do not use it “as is” in the logic of programs.

The amount of each product is held in the counter Acc value. To access these values, use the addressing of C5:0.ACC (or C5.0.2).

Values of each product are multiplied by the amount of the item and the final total is summed together.



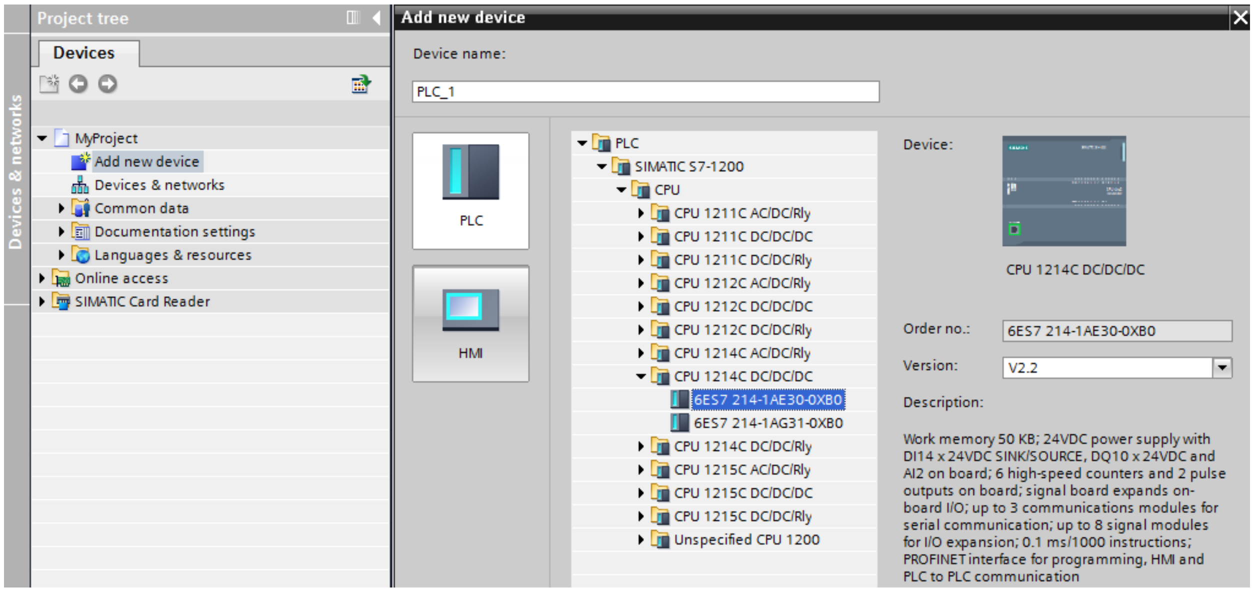
The number in Source B may be either a constant (as is here), or a value from a N7 location. If from an N7 location, the value that is to be used must be entered into that N7 location.

You may use the Siemens' HMI instead of the wired buttons. More information about configuring the HMI may be found at:

http://www.youtube.com/watch?v=Gh0s4TIDGEE

Siemens SIMATIC S7-1200 Part 3 - Adding an HMI to a controller project: See how easy it is to integrate HMI screens into the controller user program using the same Step 7 Basic Software for both SIMATIC Basic HMI panels and S7-1200 Controllers. This is part three of a four part series.

A Simple HMI Tutorial (Siemens)



Device name: PLC_1

Device: CPU 1214C DC/DC/DC

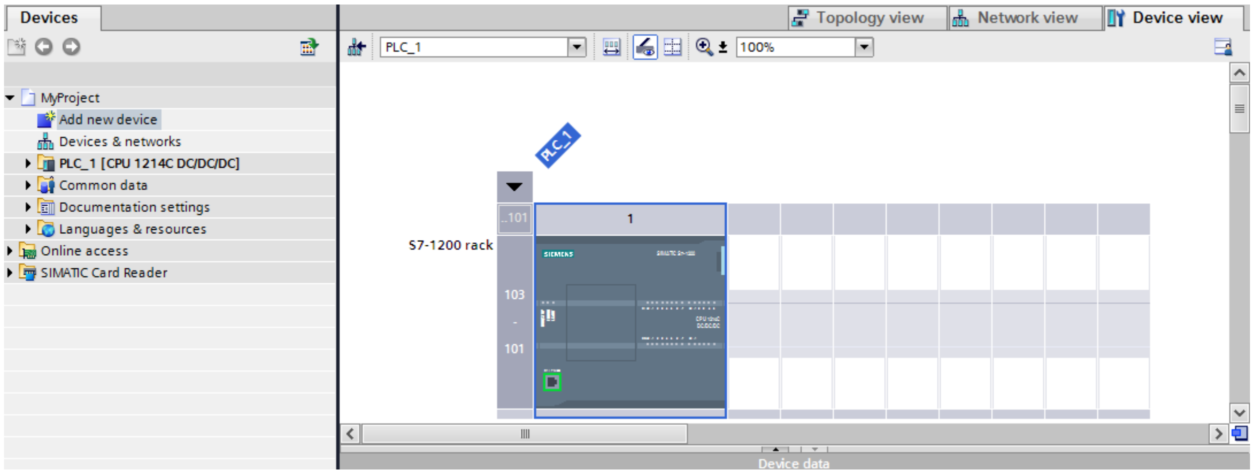
Order no.: 6ES7 214-1AE30-0XB0

Version: V2.2

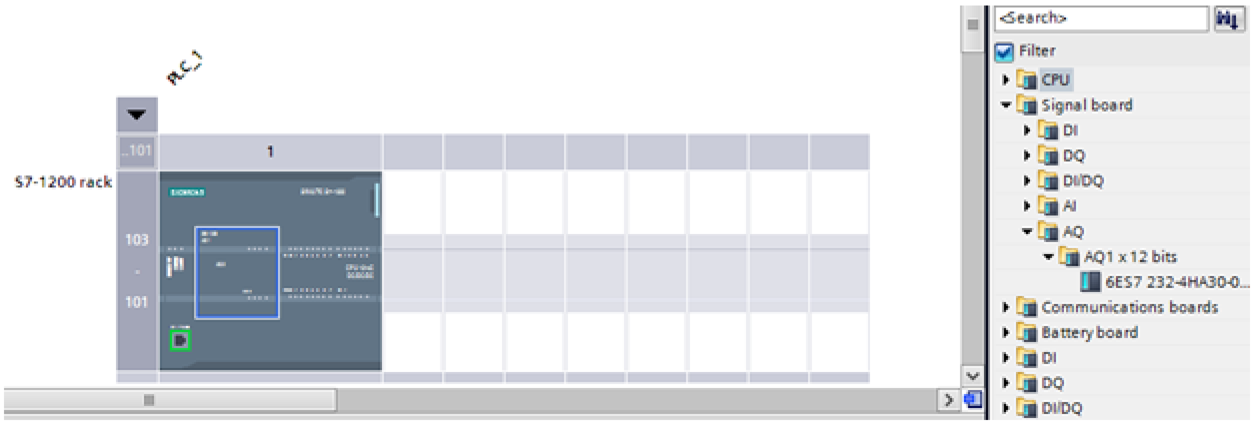
Description:

Work memory 50 KB; 24VDC power supply with DI14 x 24VDC SINK/SOURCE, DQ10 x 24VDC and AI2 on board; 6 high-speed counters and 2 pulse outputs on board; signal board expands on-board I/O; up to 3 communications modules for serial communication; up to 8 signal modules for I/O expansion; 0.1 ms/1000 instructions; PROFINET interface for programming, HMI and PLC to PLC communication

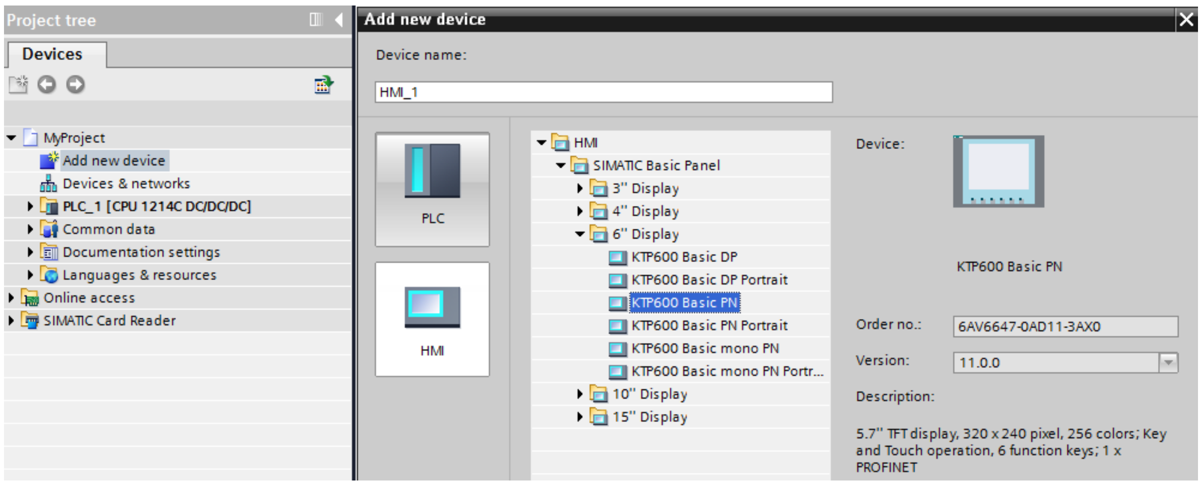
Again, the Siemens processor is to be configured similar to before:



Before and after addition of the signal board:



Addition of the HMI used in the labs:



Device name: HMI_1

Device: KTP600 Basic PN

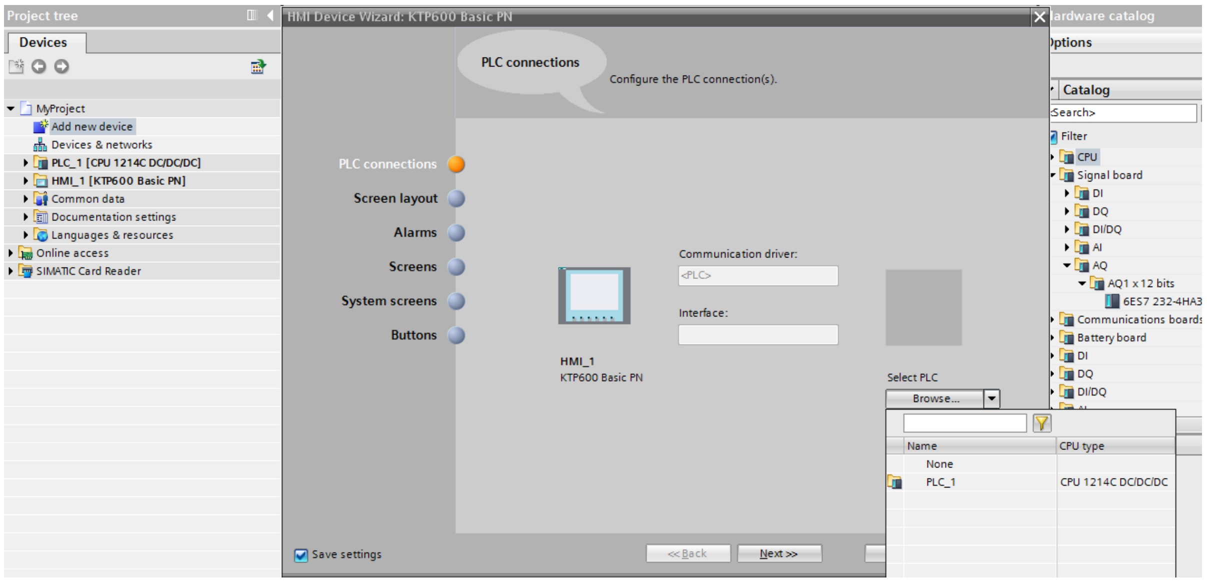
Order no.: 6AV6647-0AD11-3AX0

Version: 11.0.0

Description:

5.7" TFT display, 320 x 240 pixel, 256 colors; Key and Touch operation, 6 function keys; 1 x PROFINET

The HMI Device Wizard:



Configure the PLC connection(s).

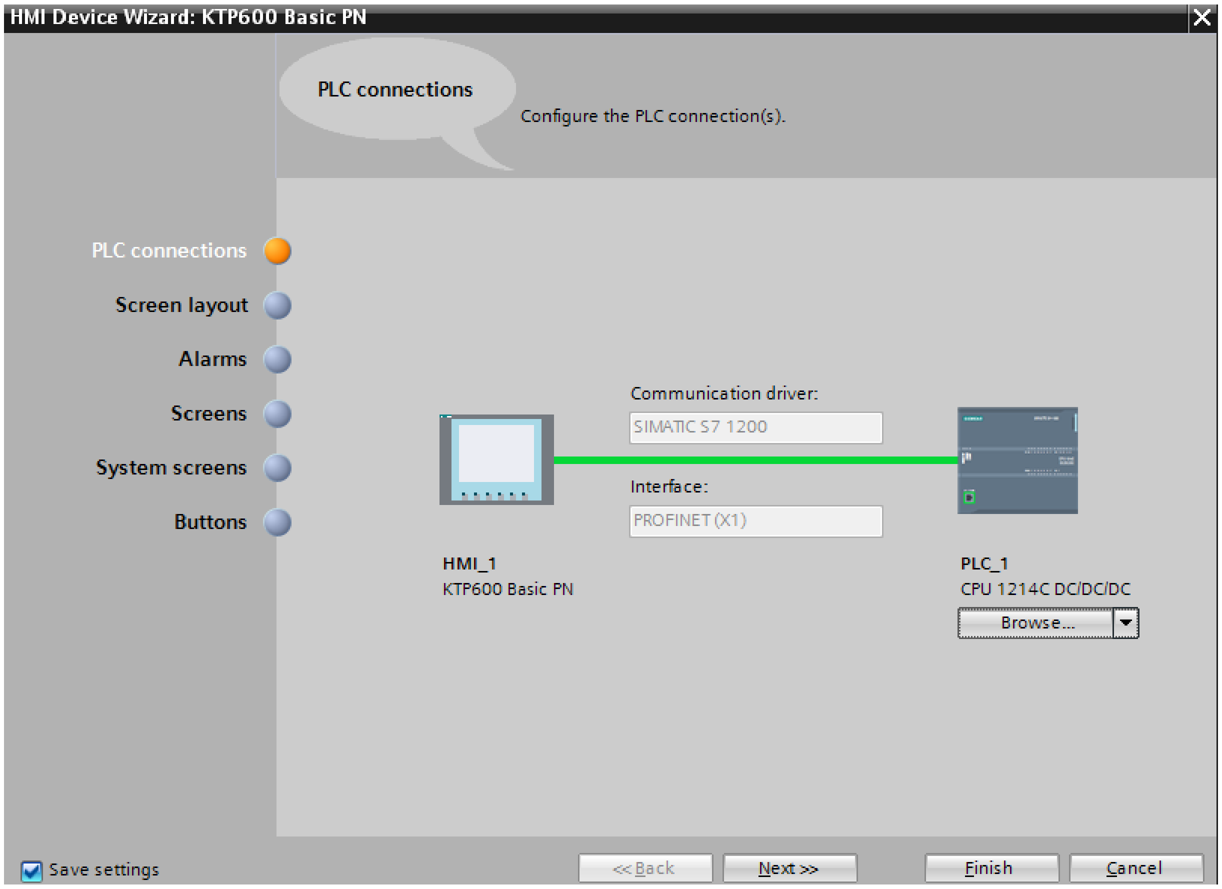
Communication driver: <PLC>

Interface:

Select PLC

Name	CPU type
None	
PLC_1	CPU 1214C DC/DC/DC

Keep answering Next>>

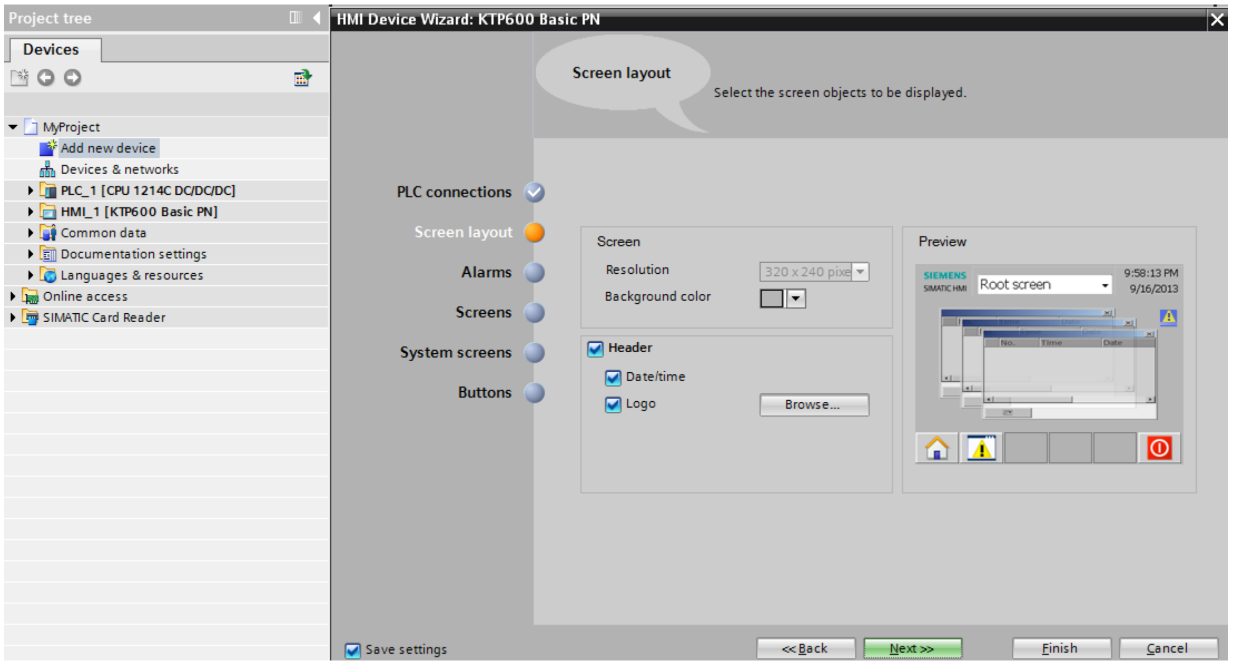


Configure the PLC connection(s).

Communication driver: SIMATIC S7 1200

Interface: PROFINET (X1)

Keep answering Next>>



Select the screen objects to be displayed.

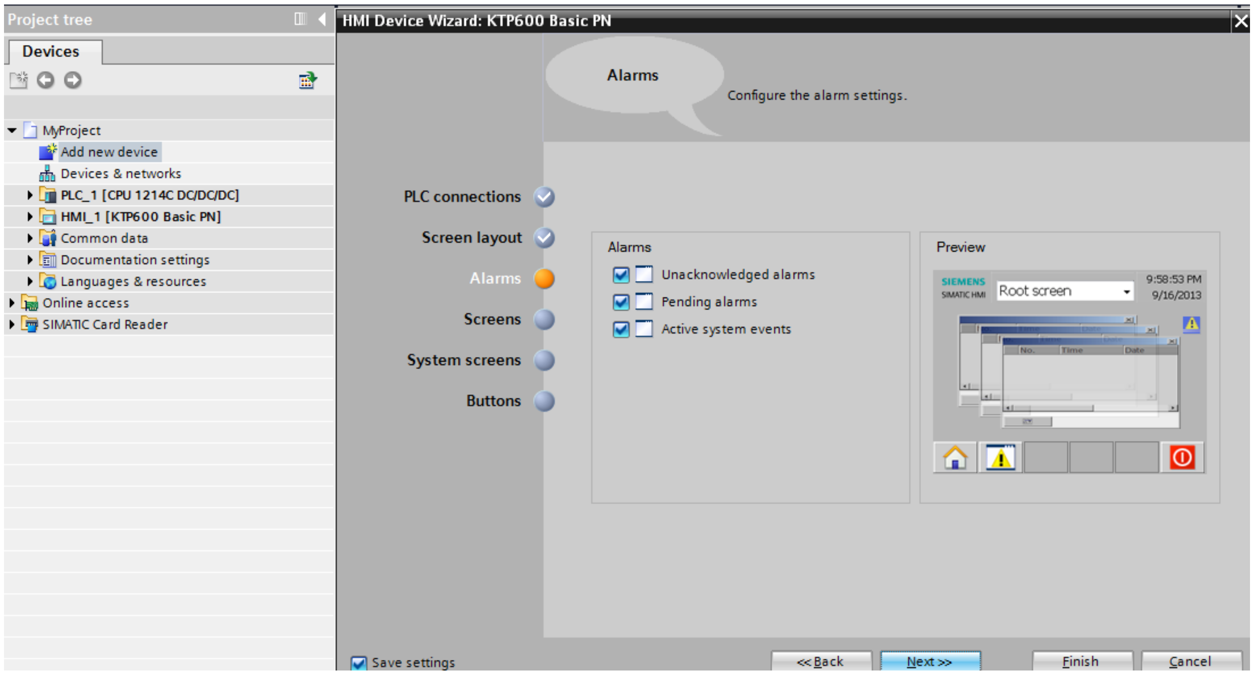
Resolution: 320 x 240 pixel

Background color

Header ✓

Date/time ✓

Logo ✓



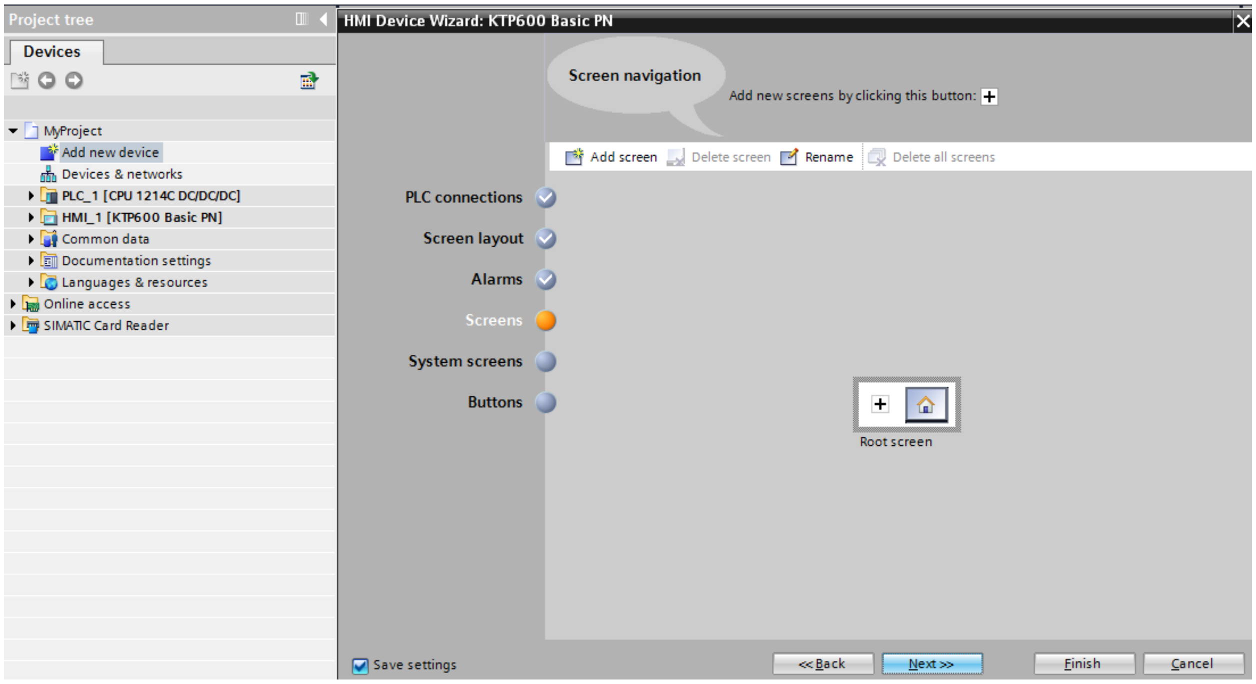
Project tree
Devices
- MyProject
  - Add new device
  - Devices & networks
  - PLC_1 [CPU 1214C DC/DC/DC]
  - HMI_1 [KTP600 Basic PN]
  - Common data
  - Documentation settings
  - Languages & resources
- Online access
- SIMATIC Card Reader

HMI Device Wizard: KTP600 Basic PN

Alarms
Configure the alarm settings.

Alarms
- [✓] Unacknowledged alarms
- [✓] Pending alarms
- [✓] Active system events

[✓] Save settings



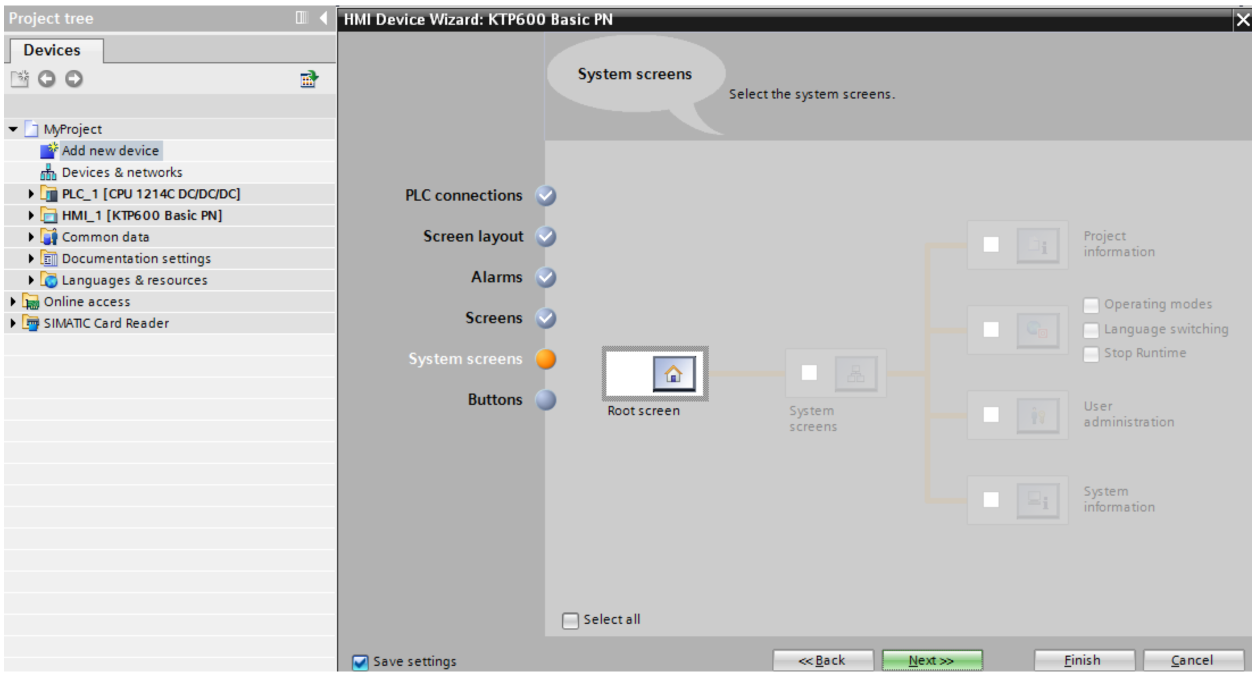
Project tree
Devices
- MyProject
  - Add new device
  - Devices & networks
  - PLC_1 [CPU 1214C DC/DC/DC]
  - HMI_1 [KTP600 Basic PN]
  - Common data
  - Documentation settings
  - Languages & resources
- Online access
- SIMATIC Card Reader

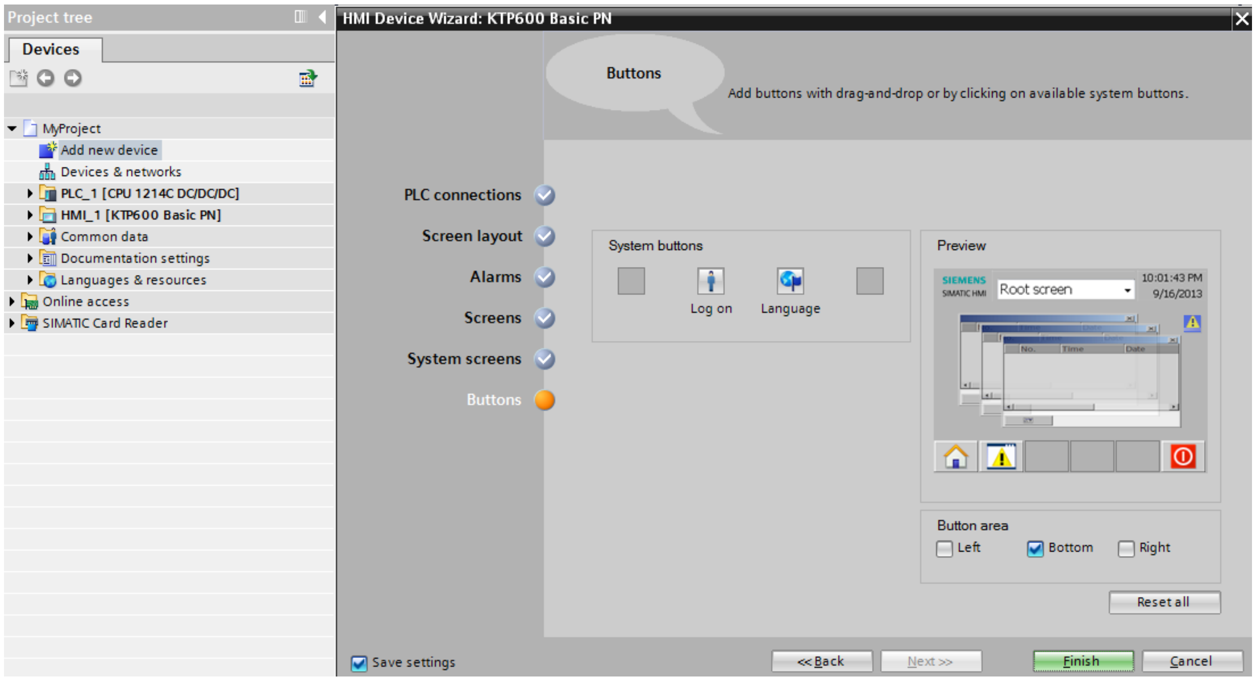
HMI Device Wizard: KTP600 Basic PN

Screen navigation
Add new screens by clicking this button: +

[✓] Save settings

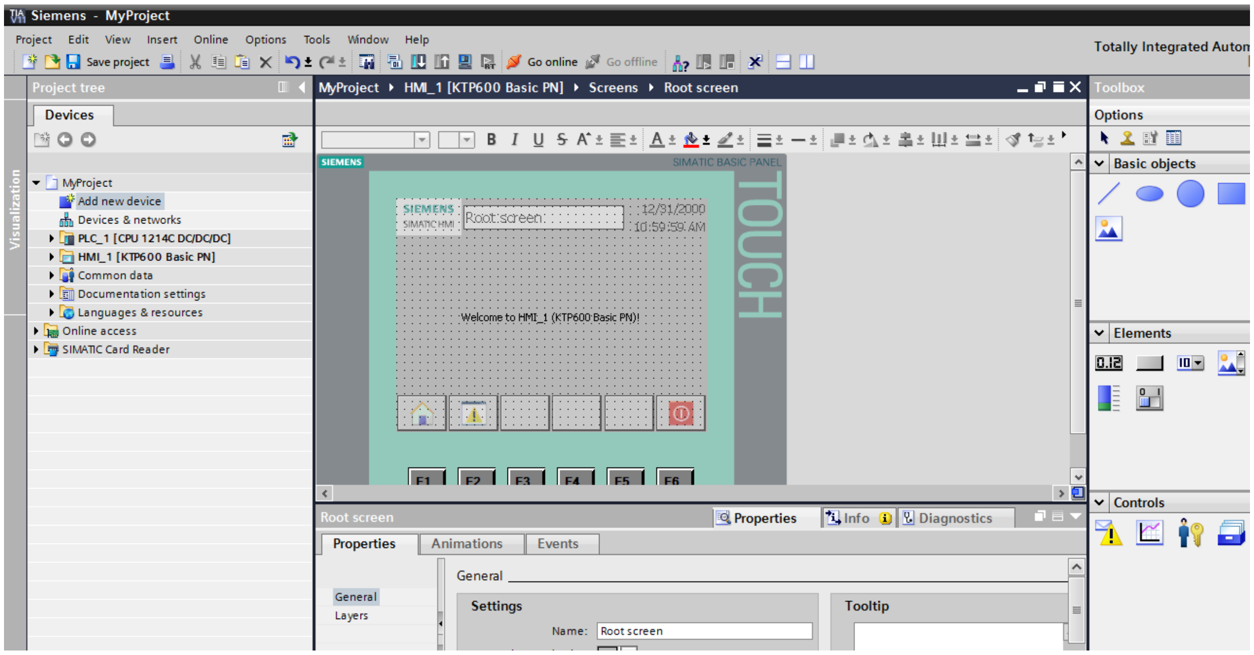
Keep answering Next>>





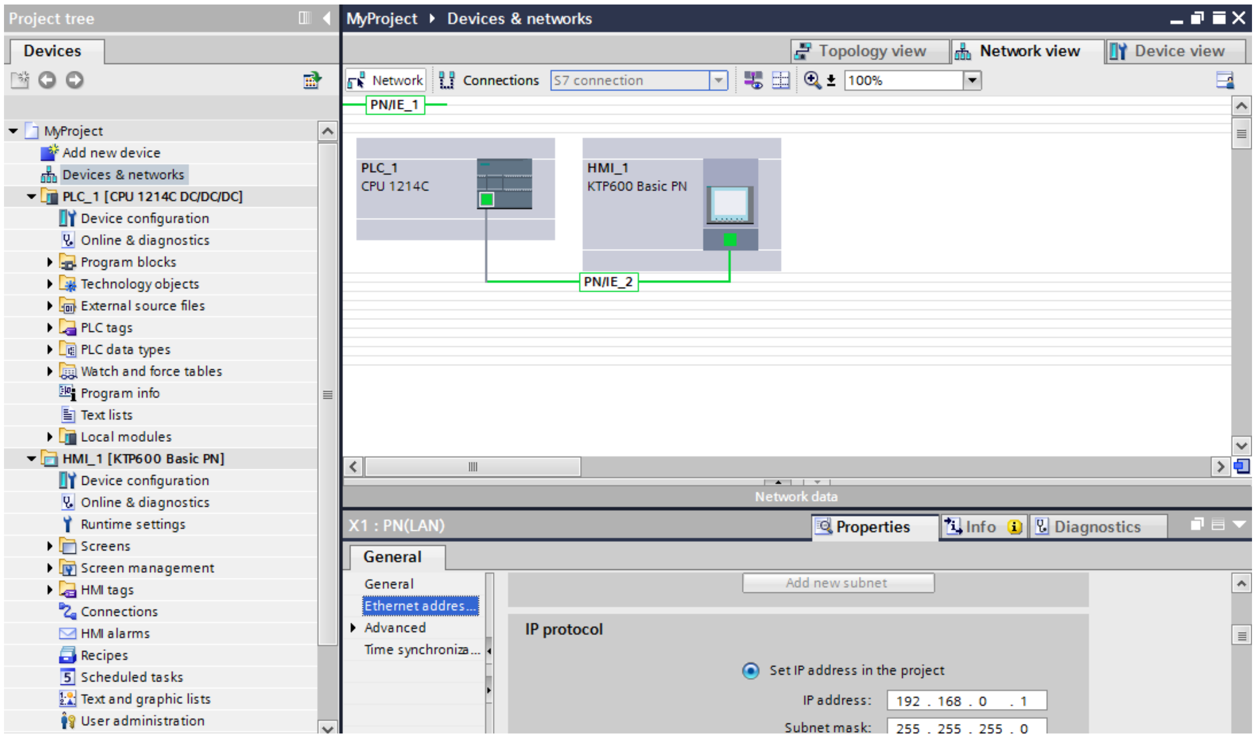
Then Finish

From the Devices and Networks choice in the Project Tree:



Name: Root screen

Choose Devices & networks

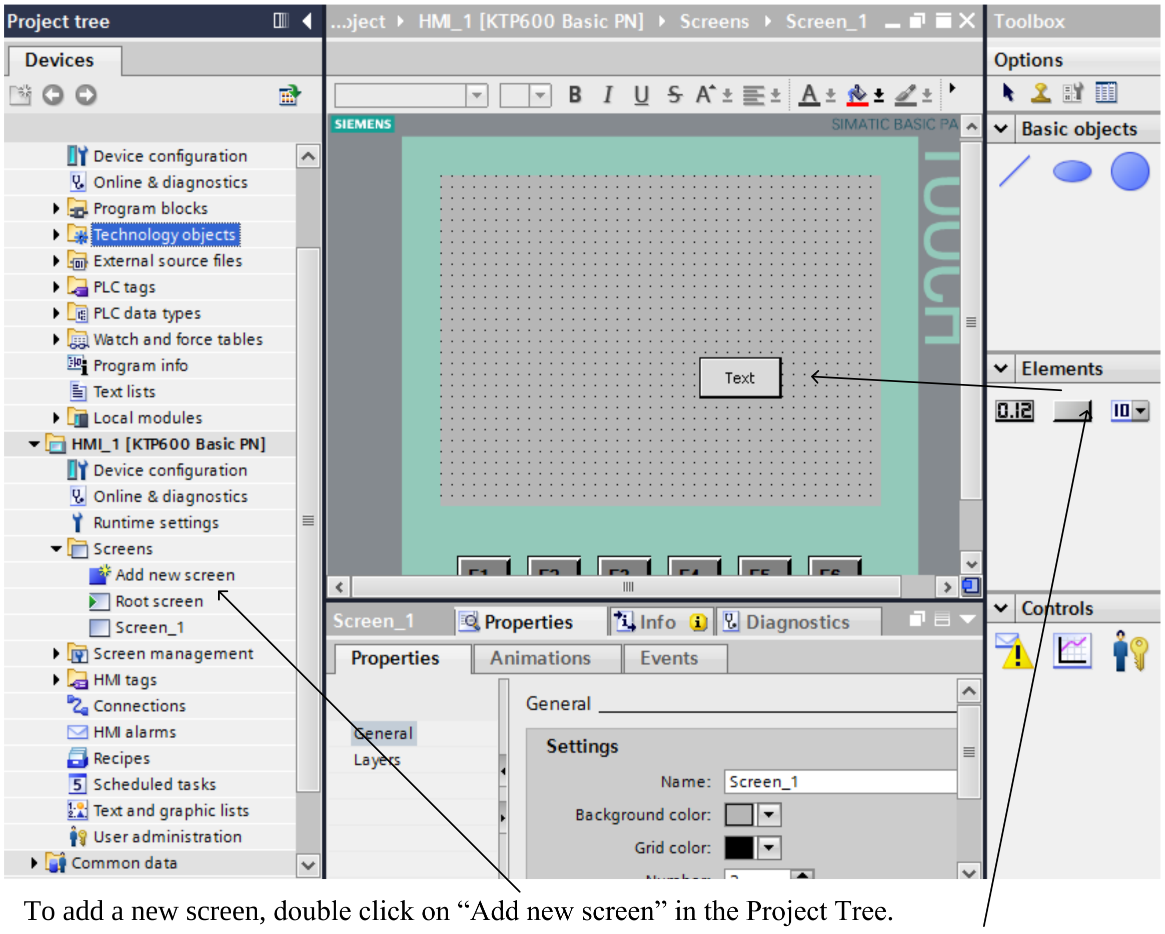


Set IP address in the project

IP address: 192 . 168 . 0 . 1

Subnet mask: 255 . 255 . 255 . 0

and set up both the IP address and Subnet mask for the PLC as well as the HMI. You may need to initialize the IP address of the HMI by setting the IP address up at power-up of the device. You have about 1 second to tap on the screen when power is first applied to get to the set-up screen. Set up the IP address of the HMI to 5 (192.168.0.5, 255.255.255.0). Read at the end of Ch. 15 about setting up a simulated HMI panel simulated on the computer screen.



Name: Screen_1
Background color:
Grid color:

To add a new screen, double click on "Add new screen" in the Project Tree.
To begin a design, select a button from the Elements Toolbox at right. Drag the button onto the screen.

Entering a Button on the Screen and configuring the button to turn on a bit in the PLC

Button

The Button object allows you to configure an object that the operator can use in runtime to execute any configurable function.

Button Layout

In the Inspector window, you customize the position, geometry, style, color and font types of the object. You can adapt the following properties in particular:

- Mode: Defines the graphic representation of the object.
- Text / Graphic: Defines whether the Graphic view is static or dynamic.
- Define hotkey: Defines a key, or shortcut that the operator can use to actuate the button.

You can only define a hotkey for HMI devices with keys.

Mode for Button

The button display is defined in Properties > Properties > General >Mode in the Inspector window.

Mode Description

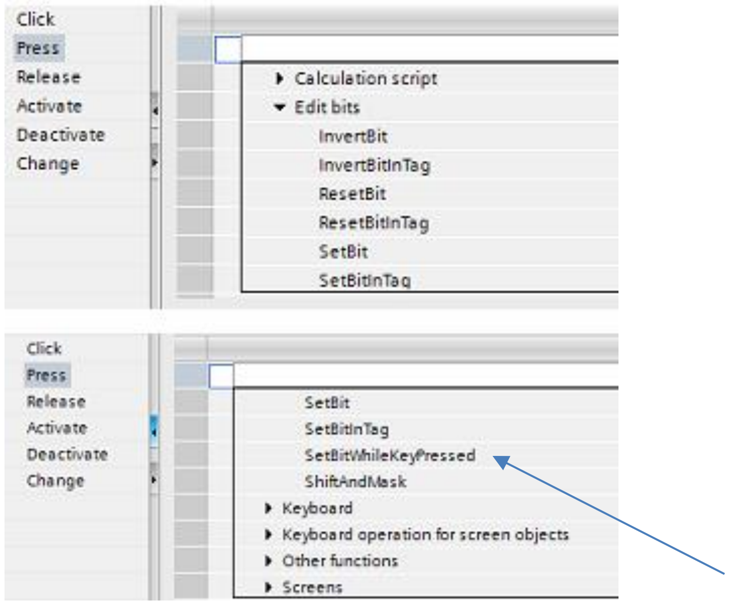
Invisible	The button is not visible in runtime.
Text	The button is displayed with text. This text explains the function of the button.
Graphic	The button is displayed with a graphic. This graphics represents the function of the button.

Depending on the device, Text /Graphic is also available.

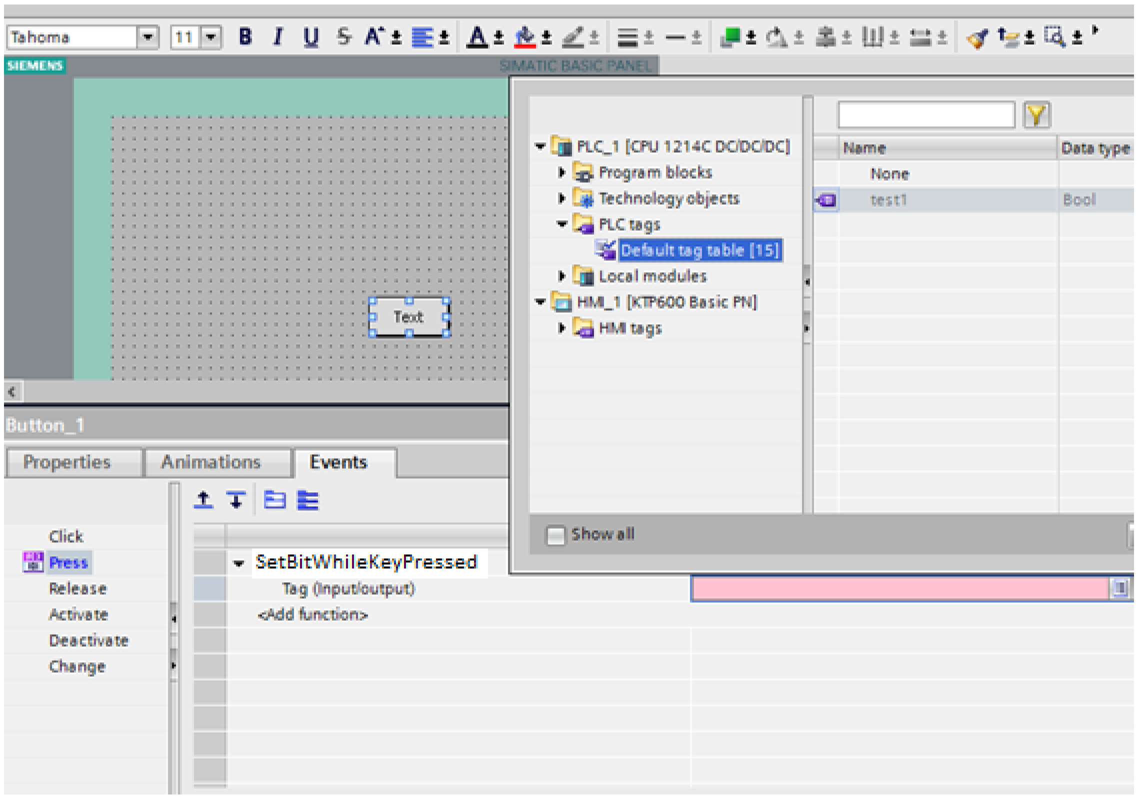
The Mode property settings are used to define whether the display is static or dynamic. The display is defined in Properties > Properties > General >Text or Graphic in the Inspector window. Your options for the type Graphic include the following.

Type	Option	Description	
Graphic	Graphic	Graphic OFF	is used to specify a graphic that is displayed in the button when the state is "OFF". If you enable Graphic ON, you can enter a graphic for the ON state.
	Graphics list		The graphic in the button depends on the state. The entry from the graphics list corresponding to the state is displayed.

To turn on the bit in the PLC, use Press:

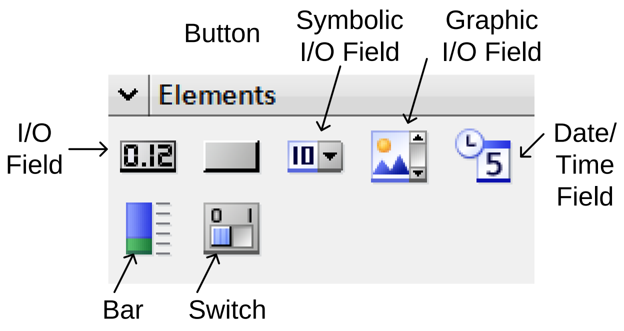


and then SetBitWhileKeyPressed:

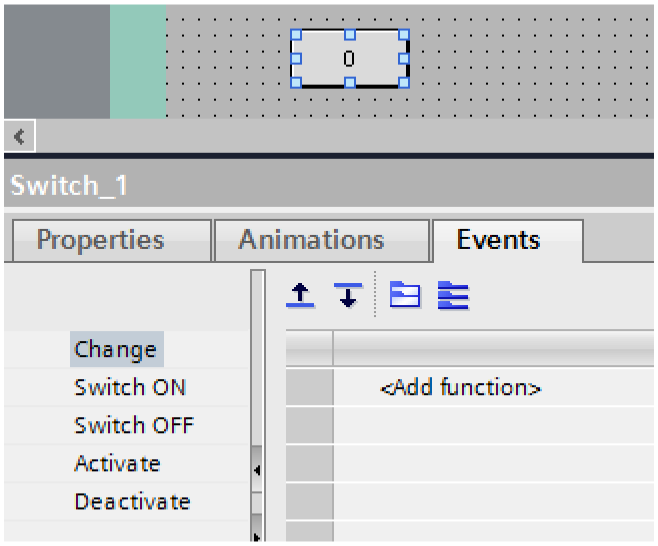


The tag is built in the PLC for an internal bit and referenced to the SetBitWhileKeyPressed function:

There are a number of input types for data entry from the HMI. They include:



For Switch, the following choices are available:



Screen Navigation

You will also need to consider configuring screen navigation. For a production process consisting of multiple sub-processes, you will configure multiple screens. You have the following options to enable the operator to switch from one screen to the next in Runtime:

- Assign buttons to screen changes
- Configuring screen changes at local function keys

The procedure for configuring screens follows:

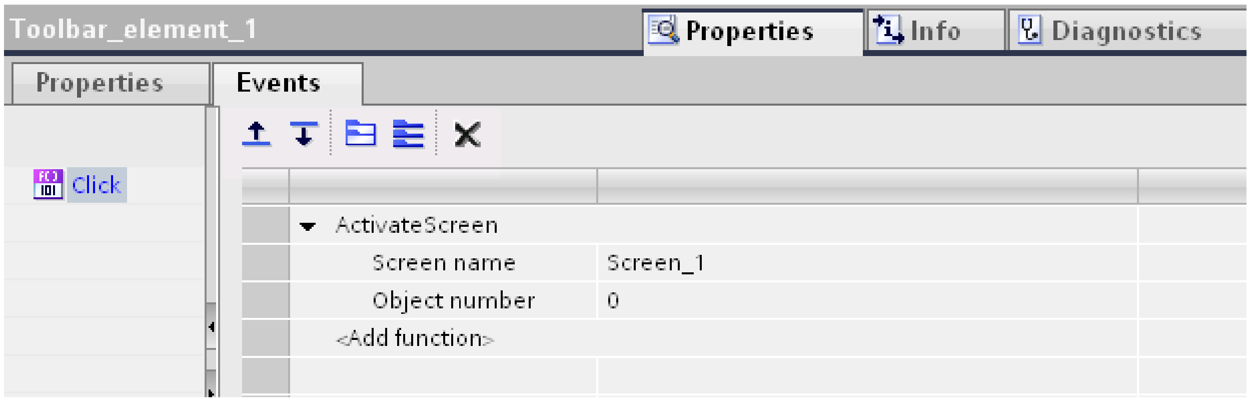
Before you create a screen change, define the plant structure and derive from it the screen changes that you want to configure.

Create the start screen under Runtime Settings > General > Start screen.

You will need to assign a button to change the screen. You will need to configure a button in the screen to switch between the screens on the HMI device during operation.

Procedure

1. Double-click Screen_1 in the project navigation.
2. Move Screen_2 from the project tree to the open screen by drag&drop. A button with the name Screen_1 is inserted.
3. In the Inspector window, select Properties > Events > Click. The ActivateScreen system function is displayed in the "Function list".



ActivateScreen		
Screen name	Screen_1	
Object number	0	
<Add function>		

4. At the Object number attribute, define, if required, the tab sequence number of the object on which the focus is to be set after a screen change. You can also specify a tag that contains the object number.

Overview of HMI tag tables

HMI tag tables contain the definitions of the HMI tags that apply across all devices. A tag table is created automatically for each HMI device created in the project. In the project tree there is an HMI tags folder for each HMI device. The following tables can be contained in this folder:

- Standard tag table
- User-defined tag tables
- All tags

The following tables are also available in an HMI tag table:

- Discrete alarms
- Analog alarms

With the help of these tables you configure alarms for the currently selected HMI tag.
In the project tree you can create additional tag tables in the HMI tags folder and use these to sort and group tags and constants. You can move tags to a different tag table using a drag&drop operation or with the help of the Tag table field. You activate the Tag table field using the shortcut menu of the column headings.
Standard tag table

There is one standard tag table for each HMI device of the project. It cannot be deleted, renamed or moved. The standard tag table contains HMI tags and, depending on the HMI device, also system tags. You can declare all HMI tags in the standard tag table, or create additional user-defined tag tables as you want.

User-defined tag tables

You can create multiple user-defined tag tables for each HMI device in order to group tags according to your requirements. You can rename, gather into groups, or delete user-defined tag tables. To group tag tables, create additional subfolders in the HMI tags folder.

All tags

The All tags table shows an overview of all HMI tags and system tags of the HMI device in question. This table cannot be deleted, renamed or moved.

Discrete alarms table

In the Discrete alarms table, you configure discrete alarms to the HMI tag selected in the HMI tag table. When you configure a discrete alarm, multiple selections in the HMI tag table is not possible. You configure the discrete alarms for each HMI tag separately.

Analog alarms table

In the Analog alarms table, you configure analog alarms to the HMI tag selected in the HMI tag table. When you configure an analog alarm, multiple selections in the HMI tag table is not possible. You configure the analog alarms for each HMI tag separately.

Defining Limits for a Tag

For numerical tags, you can specify a value range by defining a low and high limit. Additionally, you configure the system to process a function list whenever a tag value drops below or exceeds its configured value range.