Module 4

Programmable Logic Control Systems

Lesson 19

The Software Environment and Programming of PLCs

Instructional Objectives

After learning the lesson students should be able to

A. Describe the structure of a PLC Program
B. Describe the execution of a PLC Program
C. Describe the typical elements of an RLL Diagram
D. Design RLL Diagrams for simple industrial logic control problems

Structure of a PLC Program

There are several options in programming a PLC, as discussed earlier. In all the options the common control of them is that PLC programs are structured in their composition. i.e. they consist of individual, separately defined programs sections which are executed in sequence. These programs sections are called ‘blocks”. Each program section contains statements. The blocks are supposed to be functionally independent. Assigning a particular (technical) function to a specific block, which has clearly defined and simple interfaces with other blocks, yields a clear program structure. The testing of such programs in sections is substantially simplified.

Various types of blocks are available according to the function of the program section.

In general the major part of the program is contained in blocks that contain the program logic graphically represented. For improved modularity, these blocks can be called in a sequence or in nested configurations.

Special Function Blocks, which are similar to application library modules, are used to realize either frequently reoccurring or extremely complex functions. The function block can be “parameterized”.

The interface to the operating system of the PLC, which are similar to the system calls in application programming for Personal Computers, are defined in special blocks. They are only called upon by the system program for particular modes of execution and in the case of the faults.

Function blocks are also used where the realization of the logic control STEP 5 statements can’t be carried out graphically. Similarly, individual steps of a control sequence can be programmed into such a block and reused at various points in a program or by various programs. PLC manufacturers offer standard functions blocks for complex functions, already tested and documented. With adequate expertise the user can produce his own function blocks. Some very common function blocks (analog input put, interface function blocks for communication processors and others) may be integrated as standard function blocks and supported by the operating system of the PLC.

Users can also define separate data blocks for special purposes, such as monitoring, trending etc., and perform read/write on such areas.

Such facilities of structured programming result in programs, which are easier to read, write, debug and maintain.

Program Execution

There are different ways and means of executing a user program. Normally a cyclic execution program is preferred and this cyclic operators are given due priorities. Program processing in a PLC happens cyclically with the following execution:

1. After the PLC is initialised, the **processor** reads the individual inputs. This status of the input is stored in the process- image input table (**PII**).
2. This processor processes the program stored in the program memory. This consists of a list of logic functions and instructions, which are successively processed, so that the required input information will already be accessed before the read in PII and the matching results are written into a process-image output table (**PIQ**). Also other storage areas for counters, timers and memory bits will be accessed during program processing by the processor if necessary.
3. In the third step after the processing of the user program, the status from the PIQ will transfer to the outputs and then be switched on and/or off. Afterwards it begins the execution of the next cycle from step 1.

The same cyclic process also acts upon an RLL program.

The time required by the microprocessor to complete one cycle is known as the **scan time**. After all rungs have been tested, the PLC then starts over again with the first rung. Of course the scan time for a particular processor is a function of the processor speed, the number of rungs, and the complexity of each rung.

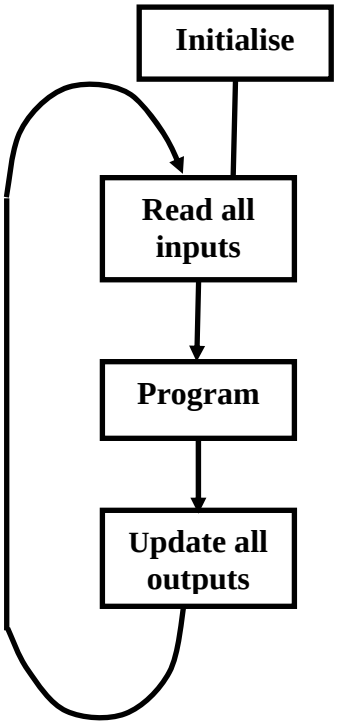


Fig. 19.1 The cyclic execution of PLC Programs

Interrupt Driven and Clock Driven Execution Modes

A cyclically executing program can however be interrupted by a suitably defined signal resulting in an interrupt driven mode of program execution (when fast reaction time is required). If the interrupting signal occurs at fixed intervals we can also realized time synchronous execution (i.e. with closed loop control function). The cyclic execution, synchronized by a real time clock is the most common program structure for a PLC.

Similarly, programmers can also define error-handling routines in their programs. Specific and defined error procedures are then invoked if the PLC operating system encounters fault of given types during execution.

Point to Ponder: 1

A. What are the different modes of execution?

B. Which is the most common?

C. State for each of the others, when these are to be used.

D. Give examples for your arguments if you can

Programming Languages

PLC programs can be constructed using various methods of representation. Some of the common ones are, described below.

The Relay Ladder Logic (RLL) Diagram

A Relay Ladder Logic (RLL) diagram, also referred to as a Ladder diagram is a visual and logical method of displaying the control logic which, based on the inputs determine the outputs of the program. The ladder is made up of a series of “rungs” of logical expressions expressed graphically as series and parallel circuits of relay logic elements such as contacts, timers etc. Each rung consist of a set of inputs on the left end of the rung and a single output at the right end of each rung. The structure of a rung is shown below in Fig. 19.1(a) & (b). Fig. 19.1 shows the internal structure of a simple rung in terms its element contacts connected in a series parallel circuit.

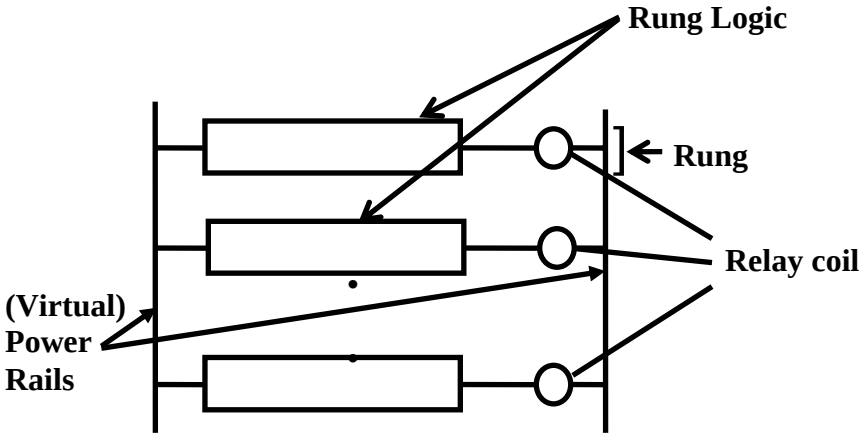


Fig. 19.1(a) The structure of Relay Ladder Logic Programs for PLCs

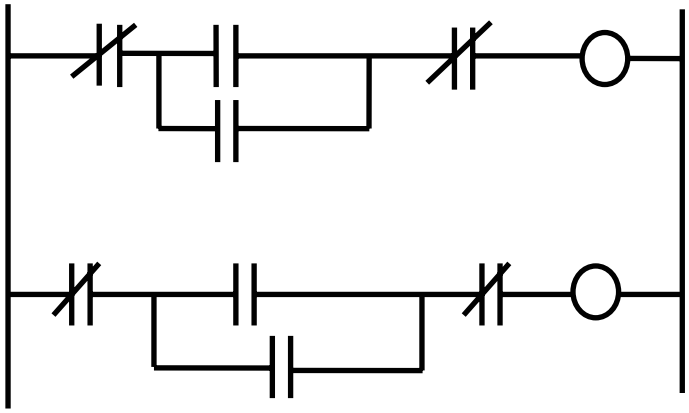


Fig. 19.1(b) The internal structure of a simple Rung

RLL Programming Paradigms: Merits and Demerits

For the programs of small PLC systems, RLL programming technique has been regarded as the best choice because a programmer can understand the relations of the contacts and coils intuitively. Additionally, a maintenance engineer can easily monitor the operation of the RLL program on its graphical representation because most PLC manufacturers provide an animated display that clearly identifies the states of the contacts and coils. Although RLL is still an important language of IEC 1131-3, as the memory size of today's PLC systems increases, a large-sized RLL program brings some significant problems because RLL is not particularly suitable for the well-structured programming: It is difficult to structure an RLL program hierarchically.

Example: Forward Reverse Control

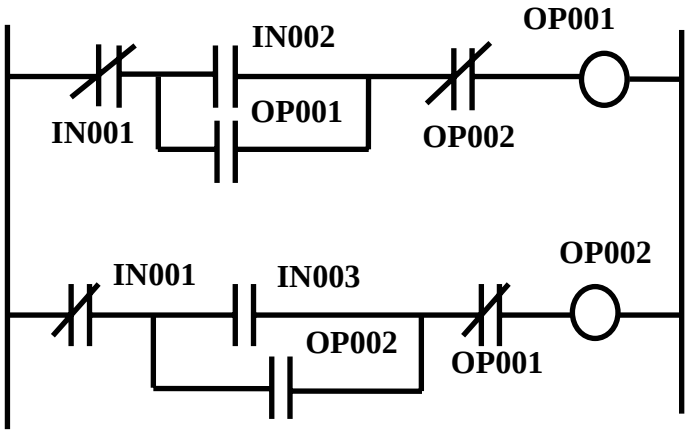


Fig. 19.2 RLL Diagram for the Forward Reverse Control Problem

This example explains the control process of moving a motor either in the forward direction or in the reverse direction. The direction of the motor depends on the polarity of the supply. So in order to control the motor, either in the forward direction or in the reverse direction, we have to provide the supply with the corresponding polarity. The Fig 19.2 depicts the procedure to achieve this using Relay Ladder Logic. Here, the Ladder consists of two rungs corresponding to forward and reverse motions.

The rung corresponding to forward motion consists of

1. A normally closed stop push-button (IN001),
2. A normally opened forward run push-button (IN002) in parallel with a normally opened auxillary contact(OP001),
3. A normally closed auxillary contact(OP002) and
4. The contacter for coil(OP001).

Similarly, the rung corresponding to reverse motion consists of

1. A normally closed stop push-button (IN001),
2. A normally opened forward run push-button (IN003) in parallel with a normally opened auxillary contact(OP002),
3. A normally closed auxillary contact(OP001) and
4. The contacter for coil(OP002).

Operation: The push-buttons(PB) represented by IN--- are real input push-buttons, which are to be manually operated. The auxillary contacts are operated through program. Initially the machine is at standstill, no voltage supply is present in the coils, and the PBs are as shown in the fig. The stop PB is intially closed, the motor will not move until the forward run PB/reverse run PB is closed. Suppose we want to run the motor in the forward direction from standstill, the outputs of the coils contacters have logic '0' and hence both the auxillary contacts are turned on. If we press and release the forward run PB, the positive voltage from the +ve voltage rail is passed to the

coil. Once the coil contacter gives the logic '1', the following consequences takes place simultaneously

A. The auxillary contact OP001 in the second rung becomes opened,which stops the voltage for reverse motion of the motor. At this stage, the second rung is not turned on even the reverse run PB is pressed by mistake.

B. The auxillary contact OP001 is the first rung is on, which provides the path for the positive voltage until the stop PB is pressed. Here the auxillary contact OP001 acts as a 'latch', which facilitates even to remove the PB IN002 once the coil OP001 is on.

If we want to rotate the motor in the reverse direction, the stop PB is to be pressed sothat no voltage in the coil is present, then we can turn on the PB corresponding to reverse run. This is a simple example of 'interlocking', where each rung locks the operation of the other rung.
There are several other programming paradigms for PLCs. Two of them are mentioned here for briefly.

The Function Chart (IEC)
Depicts the logic control task symbols in terms of functional blocks connected symbolically in a graphic format.

The Statement List (STL)
Is made up of series of assembly language like statements each one of which represents a logic control statement executable by the processor of the programmable controller. The statement list is the most unrestricted of all the methods of representation. Individual statements are made up of mnemonics, which represent the function to be executed. This method of representation is favoured by those who have already had experience in programming microprocessors or computers.

Point to Ponder

Can you think of a control logic which would be difficult to program in the RLL framework?

Typical Operands of PLC Programs

In the RLL Program that we have already seen we have already encountered contacts and output coils. However, there are some other elements of a RLL diagram which are commonly used in industrial applications. These are described in some detail below.

Inputs I, Output Q

Generally, the operands of a PLC program can be classified as inputs (I) and outputs (Q). The input operands refer to external signals of the controlled system, whose values are acquired from the input signal modules. The operating system of the PLC assigns the signal status of the input and output modules into the process image of the inputs and outputs at the beginning of the program. The operand area is located within the process image of the central controller RAM. Within the program, the signal status of the operand area is scanned and processed into logic functions in accordance with the user program; the individual bits within the process output image are fixed. When the program has been executed the operating system transfers the signal

status of the process image independently to the output modules. This method enables faster program execution because access to the process image is executed much faster than access to the I/O – modules.
In RLL Programs, inputs are represented as contacts. Two types of contacts are used, namely, normally open and normally closed contacts. The difference in the sense of interpretation between these contacts is shown below.
The switch shown here is a NO contact, i.e. it is closed when it is active.

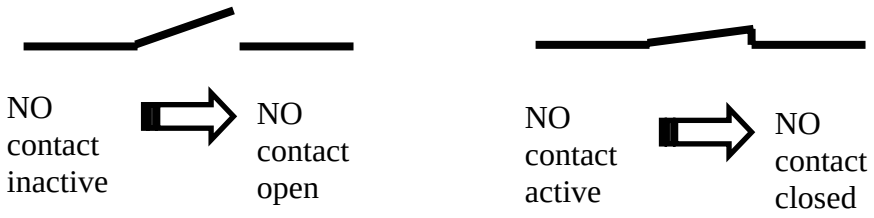


Fig. 19.3 (a) NO Contact interpretation

The switch shown here is a NC contact. i.e. it is closed when it is not active.

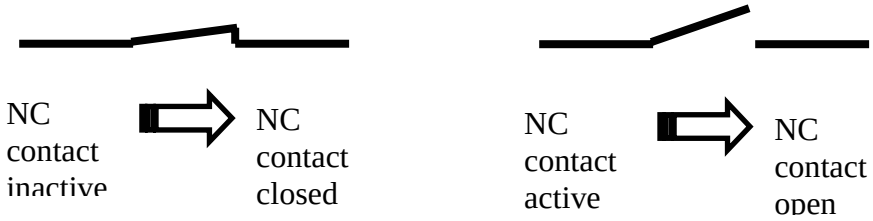


Fig. 19.3 (b) NC Contact interpretation

Internal Variable Operands or Flags

In addition to the inputs and outputs, which correspond to physical signals in the controlled systems internal variables are required to save the intermediate computational values of the program. These are referred to as Flags, or the Auxiliary Contacts in Relay Ladder Logic parlance. The number of such variables admissible in a program may be limited. Such auxiliary contacts correspond to output values and are assumed to be activated by the corresponding output values. They may be either of an NO or an NC type. Therefore, an NO auxiliary contact would be closed if the corresponding output is active i.e. has value “1”.

Point to Ponder: 2

A. *What is the basic difference between Input and Auxiliary Contacts?*

C. *Design an RLL Program for the following Industrial Problem*

In a controlled plant three fans are to be monitored. If at least two fans are running, the indicator light of the monitor is permanently switched on. The indicating lamp blinks slowly if only one fan is running (with 0.5 Hz) and rapidly (with 2 Hz) if no fan at all is on. the monitor is only active when the signal “plant in operation” signal status “1” is activated. Otherwise the indicating light is switched off. The function “lamp test” can be

carried out with the signal "plant in operation". At signal status "1" of this signal the indicating light is either permanently on or is flashing.

Timer

These are special operands of a PLC, which represent a time delay relay in a relay logic system. The time functions are a fixed component of the central processing unit. The number of these varies from manufacturer to manufacturer and from product to product. It is possible to achieve time delays in the range of few milliseconds to few hours.

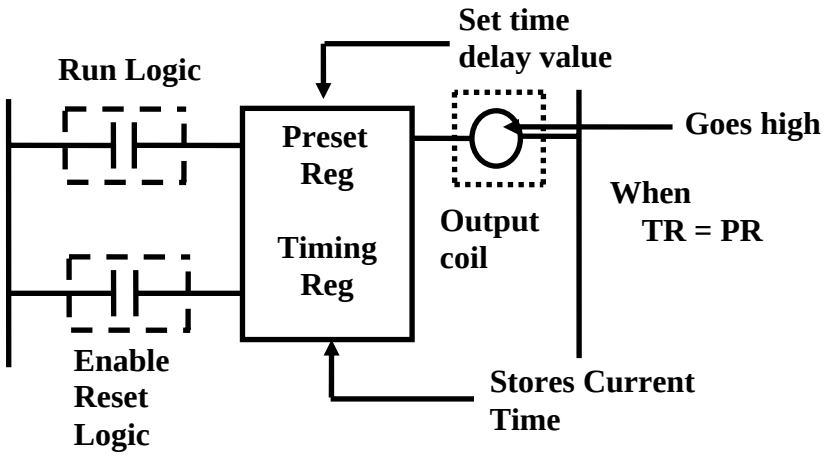


Fig. 19.4 Structure of a Typical Timer

Representation for timers is shown in Fig. 19.4. Timers have a preset register value, which represent the maximum count it can hold and can be set using software/program. The figure shown below has a 'enable reset logic' and 'run logic' in connection with the timer. The counter doesnot work and the register consists of 'zero' until the enable reset logic is 'on'. Once the 'enable reset logic' is 'on', the counter starts counting when the 'run logic' is 'on'. The output is 'on' only when the counter reaches the maximum count.

Various kinds of timers are explained as follows

On delay timer: The input and output signals of the on delay timer are as shown in the Fig. 19.6. When the input signal becomes on, the output signal becomes on with certain delay. But when the input signal becomes off, the output signal also becomes off at the same instant. If the input becomes on and off with the time which less than the delay time, there is no change in the output and remains in the 'off' condition even the input is turned on and off i.e., output is not observed until the input pulse width is greater than the delay time.

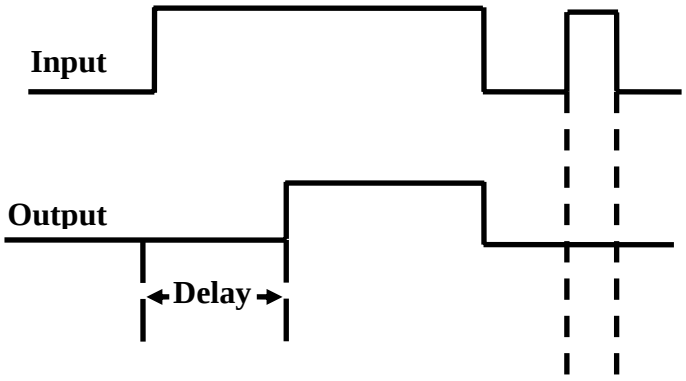


Fig. 19.5 A Typical Input output waveform for an On-Delay Timer

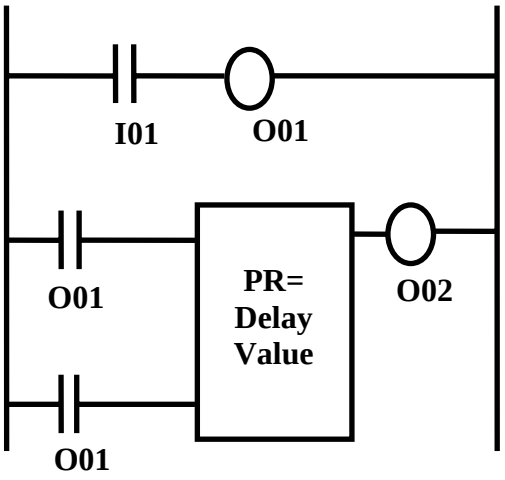


Fig. 19.6 The realization of an On-delay timer from a general timer.

Realization of on-delay timer: The realization of on-delay timer using the basic timer shown in the previous fig is explained here. The realization is as shown in the Fig. 19.6, which shows a real input switch(IN001), coil1(OP002), two normally opened auxillary contacts(OP002), coil2(OP002). When the real input switch is 'on' the coil(OP002) is 'on' and hence both the auxillary switches are 'on'. Now the counter value starts increasing and the output of the timer is 'on' only after it reaches the maximum preset count. The behaviour of this timer is shown in figure, which shows the on-delay timer. The value in the counter is 'reset' when the input switch(IN001) is off as the 'enable reset logic' is 'off'. This is a non-retentive timer.

Off delay timer: The input and output signals of the off delay timer are as shown in the Fig. 19.7. When the input signal becomes on, the output signal becomes on at the same time. But when the input signal becomes off, the output signal becomes 'off' with certain delay. If the input becomes on and off with the time which less than the delay time, there is no change in the output and remains in the 'on' condition even the ipnut is turned on and off i.e., the delay in the output is not observed until the input pulse width is greater than the delay time.

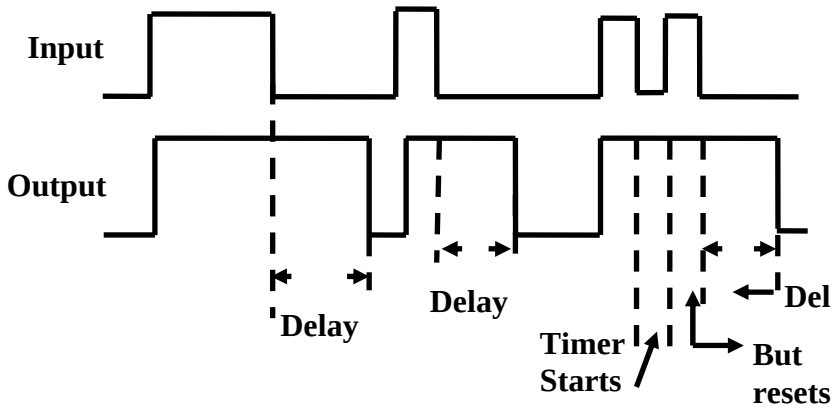


Fig. 19.7 A Typical Input output waveform for an Off-Delay Timer

Realization of off-delay timer: The realization of on-delay timer using the basic timer shown in the previous fig is explained here. The realization is as shown in the Fig. 19.8, which shows a real input switch(IN001), coil1(OP002), two normally closed input contacts(IN001), output contacts (OP002,OP003). When the real input switch is 'on', the coil(OP002) is 'on' and both the auxillary input switches are 'off'. Now the output contact(OP002) becomes 'off' which in turn makes the auxillary contact(OP002) in the third rung to become 'on' and hence the output contact(OP003) is 'on'. When the real input switch is 'off', the counter value starts increasing and the output of the contact becomes 'on' after the timer reaches the maximum preset count. At this time the auxillary contact in the third rung becomes 'off' and so is the output contact(OP003). The input and output signals are as shown in the figure, which explain the off-delay timer.

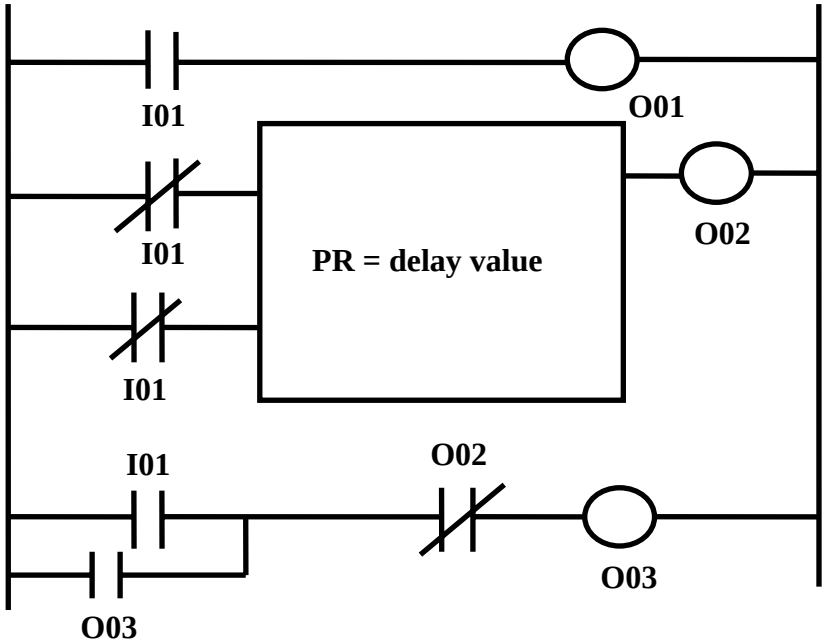


Fig. 19.8 The realization of an Off-delay timer from a general timer.

Fixed pulse width timer: The input and output signals of the fixed pulse width timer are as shown in the Fig 19.9. When the input signal becomes on, the output signal becomes on at the

same time and remains on for a fixed time then becomes ‘off’. The output pulse width is independent of input pulse width.

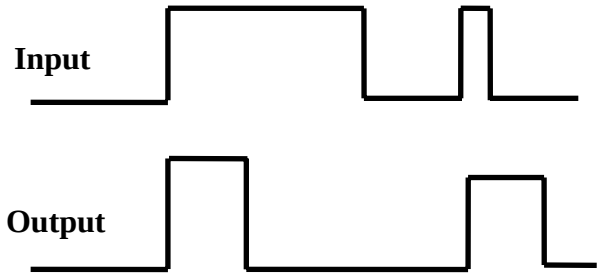


Fig. 19.9 A Typical Input output waveform for a Fixed Width Timer

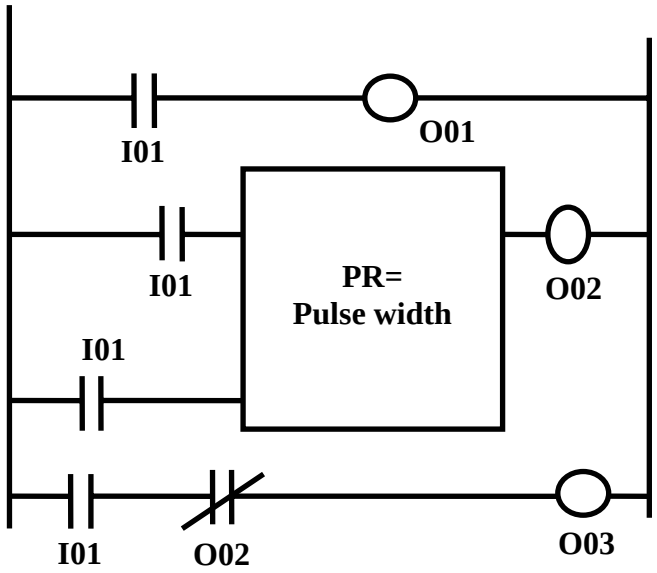


Fig. 19.10 The realization of a Fixed width timer from a general timer.

Retentive timer: The input and output signals of the retentive timer are as shown in the 19.11. This is also implemented internally in a register as in the previous case. When the input is ‘on’ , the internal counter starts counting until the input is ‘off’ and at this time, the counter holds the value till next input pulse is applied and then starts counting starting with the value existing in the register. Hence it is named as ‘retentive’ timer. The output is ‘on’ only when the counter reaches its ‘terminal count’.

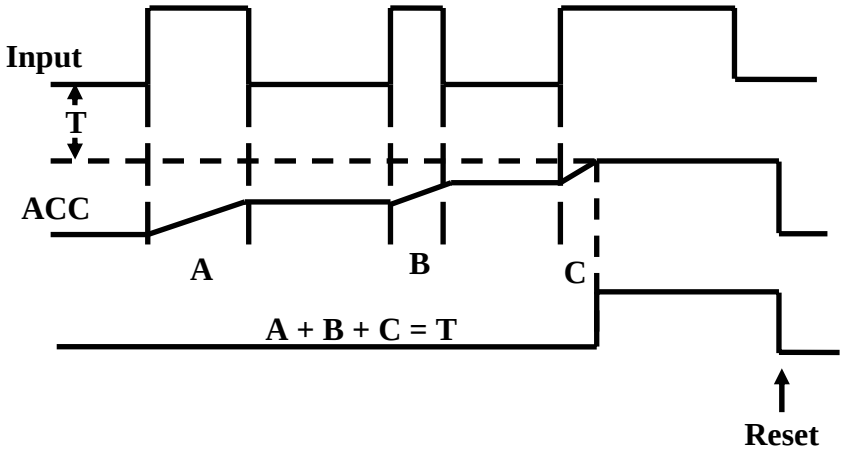


Fig. 19.11 A Typical Input output waveform for a Retentive Timer

Non-retentive timer: The input and output signals of the non-retentive timer are as shown in the Fig. 19.11. This is implemented internally in a register. When the input is 'on', the internal counter starts counting until the input is 'off' and at this time the value in the counter is reset to zero. Hence it is named as non-retentive timer. The output is 'on' only when the counter reaches its 'terminal count'.

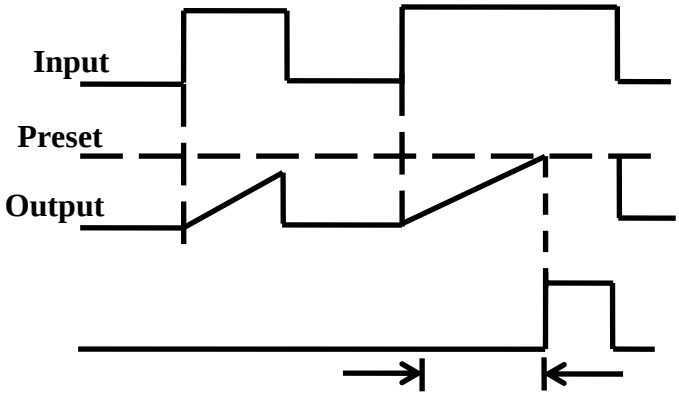


Fig. 19.12 A Typical Input output waveform for a Fixed Width Timer

Point to Ponder: 3

Draw the timing diagrams for the output coils in the RLL realizations of an on-delay, an off-delay and a fixed pulse width timer

Counter

The counting functions (C) operate as hardware counters, but are a fixed component of the central processing unit. The number of these varies for each of the programmable controllers. It is possible to count up as well as to count down. The counting range is from 0 to 999. The count is either dual or BCD coded for further processing.

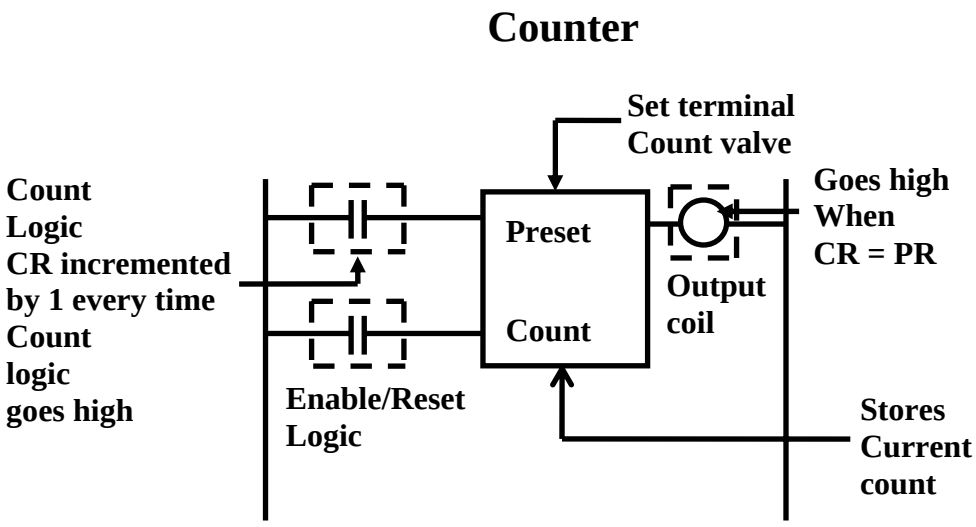


Fig. 19.13 Structure of a Typical Counter

User defined Data

If the memory capacity of the flag area is not sufficient to memorize the signal status and data, the operand area "data" (D) is applied. In general, in the flag area, primarily binary conditions apply, whereas in the data area digital values prevail and are committed to memory. The data is organized into data blocks (DB). 256 data words with 16 bit each can be addressed to each data block. The data is stored in the user memory sub module. The available capacity within the module has to be shared with the user program**.**

Addressing

The designation of a certain input or output within the program is referred to as addressing. Different PLC manufacturers adopt different conventions for specifying the address of a specific input or output signal. A typical addressing scheme adopted in PLCs manufacturers by Siemens is illustrated in the sequel.

The inputs and outputs of the PLCs are mostly defined in groups of eight on digital input and/or digital output devices. This eight unit is called a **byte**. Every such group receives a number as a **byte address.** Each in/output byte is divided into 8 individual **bits**, through which it can respond with. These bits are numbered from bit 0 to bit 7. Thus one receives a **bit address**. For example, in the address I0.4, **I** denotes that the address type is specified as Input, **0** is the byte address and **4** the bit address. Similarly in the address Q5.7, **Q** denotes that the address type is specified as Output, **5** is the byte address and **7** is the bit address.

Operation Set

The operation set of PLC programming languages can be divided into four major function groups:

- Binary or Logic functions

- Numeric or Arithmetic functions
- Program control functions and
- Other statements

Binary function combines primarily binary signal status with logic operations. The logic functions (binary logic operation) are the AND and the OR functions, according to the series and the parallel circuit arrangement on the ladder diagram. The result of the logic operation together with the memory function is then assigned to the appropriate operand. The majority of operands are inputs (I), output (O) and flags (F). With the result of logic operations of binary logic, timers can be enabled and started and counters can be initiated, incremented to count up or decremented to count down. Because the results of the time and count functions can be combined with logic functions, the associated operations are considered to be part of the group of binary functions.

Arithmetic functions are primarily used to perform arithmetic on numerical values. These can be combined with logic operations, such that the numeric operations can be enabled or disabled in a given scan based on logic conditions, much like "if then else" programming constructs. Similarly, logic conditions can be derived from numeric variables using operations like comparison.
Logic operations are established in the register of the processor (in the "accumulators"). The registers are loaded with a loading operation (they are supplied with a value). The result of the logic operation is then transferred back to the operand area via a transfer operation. Digital functions are:

Program Control operations include, Function block calls and Jump functions.

Answers, Remarks and Hints to Points to Ponder

Point to Ponder: 1

E. What are the different modes of execution?

Ans: Cyclic, Interrupt Driven and Clock Driven

F. Which is the most common?

Ans: The cyclic Mode is most common

G. State for each of the others, when these are to be used.

Ans: The interrupt driven mode is to be used for those tasks that need an immediate response during any time of normal execution, but as such are sporadic. Clock driven modes are used for those tasks which have to precisely synchronized with time relative to some defined clocks.

H. Give examples for your arguments if you can

Ans: Emergency Shutdown/Alarm Tasks are programmed using interrupt driven modes.

Communication tasks to supervisory computers for say trend updates can be implemented with clock-driven tasks.

Point to Ponder: 2

A. What is the basic difference between Input and Auxiliary Contacts?

Ans: Input contacts correspond to physical devices that can be asserted either by external agents or by the process itself for feedback. Auxiliary contacts are basically memory locations storing intermediate logical results and do not correspond to physical devices.

B. Design example of Fan monitor

Ans: In a controlled plant three fans are to be monitored. If at least two fans are running, the indicator light of the monitor is permanently switched on. The indicating lamp blinks slowly if only one fan is running (with 0.5 Hz) and rapidly (with 2 Hz) if no fan at all is on. the monitor is only active when the signal "plant in operation" signal status "1" is activated. Otherwise the indicating light is switched off. The function "lamp test" can be carried out with the signal "plant in operation". At signal status "1" of this signal the indicating light is either permanently on or is flashing.

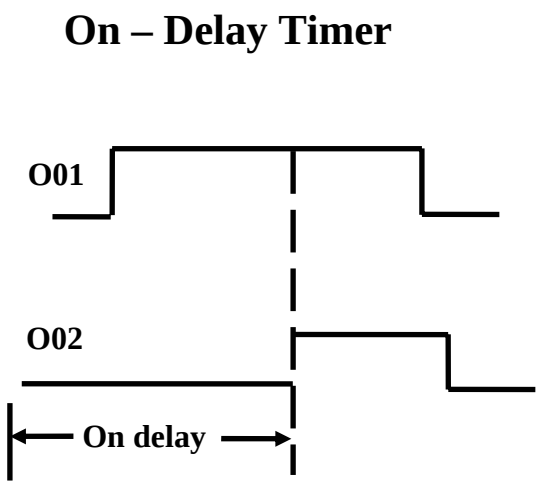
Hints: The total logic control is primarily made up of 4 elements

- Scan, if at least two fans are funning.
- Scan, if no fan is running
- Scan, if only one fan is running
- Summary of all three scans and logic control with the signal "plant in operation"

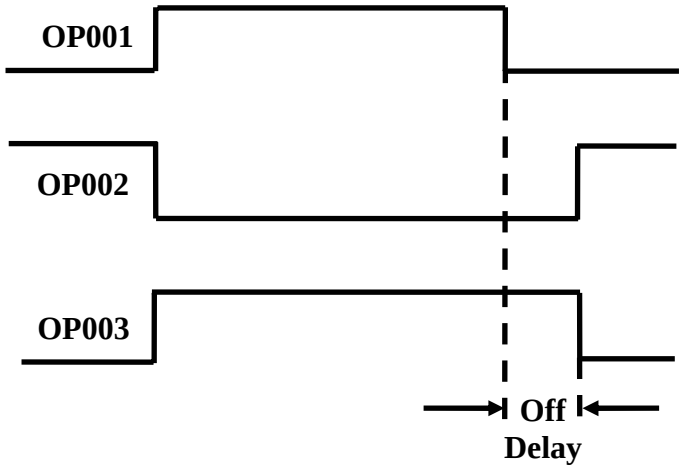
Point to Ponder: 3

Ans:

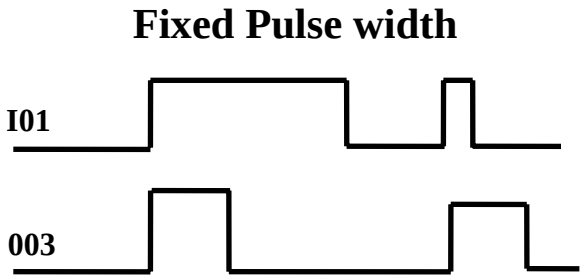
On delay timer



Off Delay timer



Fixed Pulse Width



Source:http://www.nptel.ac.in/courses/Webcourse-contents/IIT%20Kharagpur/Industrial%20Automation%20control/pdf/L-19(SM)%20(IA&C)%20((EE)NPTEL).pdf