



PDHengineer.com

Course No IC-1004

Introduction to Programming Design for Controllers

To receive credit for this course

This document is the course text. You may review this material at your leisure either before or after you purchase the course. To purchase this course, click on the course overview page:

<http://www.pdhenhineer.com/pages/IC-1004.htm>

or type the link into your browser. Next, click on the **Take Quiz** button at the bottom of the course overview page. If you already have an account, log in to purchase the course. If you do not have a PDHengineer.com account, click the **New User Sign Up** link to create your account.

After logging in and purchasing the course, you can take the online quiz immediately or you can wait until another day if you have not yet reviewed the course text. When you complete the online quiz, your score will automatically be calculated. If you receive a passing score, you may instantly download your certificate of completion. If you do not pass on your first try, you can retake the quiz as many times as needed by simply logging into your PDHengineer.com account and clicking on the link **Courses Purchased But Not Completed**.

If you have any questions, please call us toll-free at 877 500-7145.



PDHengineer.com
5870 Highway 6 North, Suite 310
Houston, TX 77084
Toll Free: 877 500-7145
administrator@PDHengineer.com

Introduction to Programming

The modern concept of the computer originated in the mid 1940's. In 1945, John Von Neumann proposed a computer architecture that included storing instructions and data in the computer. This is the computer architecture of most computers today, and is referred to as the "Von Neumann" computer architecture. The programming of early computers involved preparing a sequence of machine coded instructions. In 1954, John Backus invented the first popular high level programming language, Fortran (Formula Translator). Fortran is oriented to scientific programming and allowed one to write programs that could run on different computers. Fortran facilitated writing programs such as Finite Element Analysis programs that aided the engineering analysis of structures and mechanical components.

The number of computing devices has grown from several research computers to over a billion computers today. In addition to the computer and programmable logic controllers (PLC), embedded computers are very popular, especially in hand-held devices. The demand for programming has been driven by the need to program the many computing devices currently available. An international standard, IEC 61131-3 was published in 1993 for standardization of programmable controller programming. The purpose of the standard was to make programming more consistent among PLC systems.

Program Development

A **program** is defined as a set of coded instructions for insertion into a machine, which then performs the desired sequence of operations. Creating a program can be divided into seven steps:

1. Define the problem. Specify inputs, outputs and the processing task.
2. Analyze the requirements. Divide the processing task into subtasks. Create a hierarchical chart
3. Logic design of the program. Prepare flowcharts, pseudocode, and state diagrams as required.
4. Code the program. Convert the design into a formal programming language. Write source code.
5. Compile and run. Convert the source code to a machine executable code. Run the program.
6. Test the program. Run test scenarios to verify outputs are correct. Correct programming errors.
7. Document and maintain. Prepare user manuals and maintain the program.

Defining the Inputs and Outputs

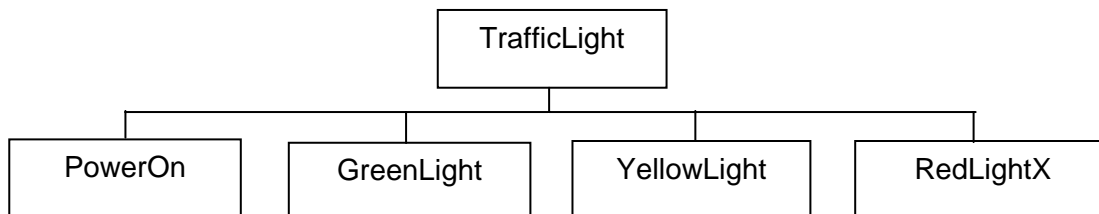
Specifying the program's inputs and outputs is an essential part of defining the programming problem. All inputs have to be converted into a data type before processing. The basic unit of data is the **bit**, a binary digit. The bit has a value of 0 or 1. This can represent the state of a switch, relay, or sensor. A bit can also represent an output control signal that can turn on lights, motors, valves and pumps. It is often referred to as a **Boolean (BOOL)** data type. This is a common data type for inputs and

outputs of programmable controllers. It is also common to group 16 bits in a word. A bit input can be identified as I:3/0 or PowerOn. Examples of outputs are O:4/2 and RedLight. Using locations to reference data is referred to as **addressing**. A symbolic address is a way to reference input and output data using descriptive words. For example, "PowerOn" is a symbolic address for a Power On switch input. Giving input and output addresses descriptive names makes the program more understandable.

There are a number of other useful data types. A sixteen bit signed number is often referred to as an **integer (INT)**. An integer can represent positive and negative numbers of up to 32,767. This data type is useful for counting. Decimal representation of numbers is useful to represent the output of sensors such as weight, pressure and temperature. Decimal numbers require 32 bits and are referred to as a **float or REAL** data type. Eight bits is called a byte and is sufficient to code a character. The ASCII code refers to a binary coding scheme for coding characters. Textual data is usually longer than one character and requires more than one byte. The string of characters is a **text or STRING** data type. This data type is useful for messages. There are also DATE and TIME data types. Strong data typing can prevent mismatching data which can have unpredictable results. Data typing is common to most high level programming languages.

Defining the Processing Tasks

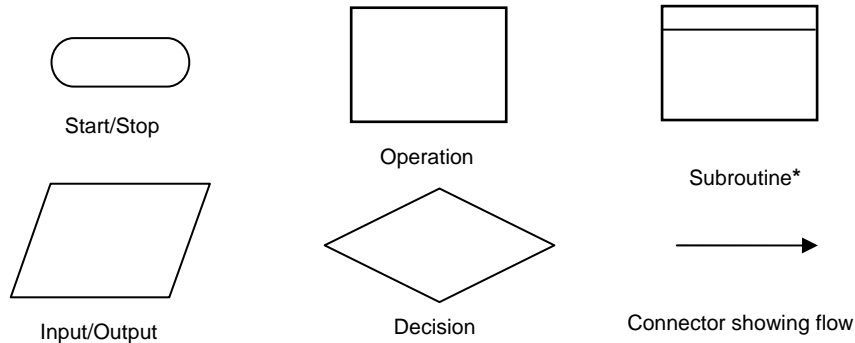
IEC 61131 International Standard provides methodologies for organizing programmable controller programs. Programmable controllers are used to automate a variety of systems: HVAC, alarm, traffic control, process control, water distribution, assembly lines, and robots. The programming components are organized in a hierarchical manner. Programs are built from **function blocks**. Function blocks are sequences of code or graphical components to accomplish a specific task. Most programming languages allow a program to be divided into functional parts; other names for these are subroutine, function, procedure and method. The following hierarchical chart shows the organization of a traffic light program showing the individual function blocks. PowerOn controls startup, and the others control the lights.



Logic Design

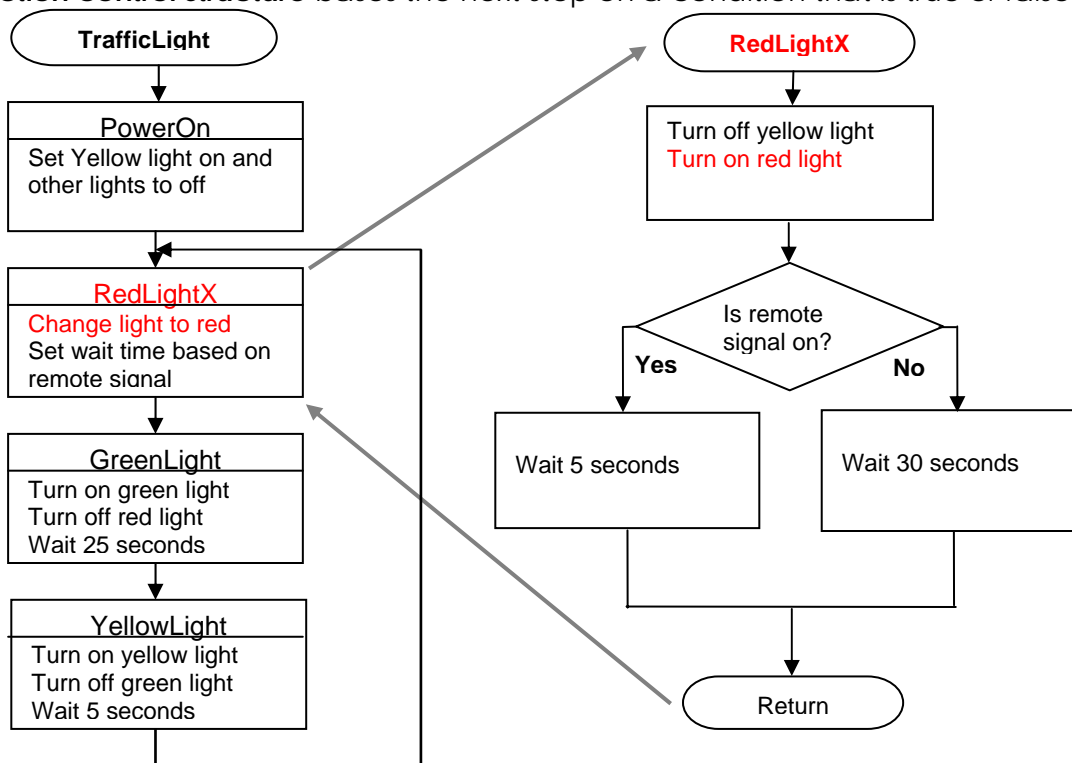
A flow chart is a common design tool to graphically organize the sequence of instructions. Flow charts have been used for designing programs since the development of the modern computer. The following shows the common flowchart symbols. Start/stop symbols indicate the beginning and ending of an instruction sequence. Most programs require inputs and outputs. These are shown using a parallelogram. An operation symbol commonly indicates actions, value assignments, and calculations.

The decision symbol indicates a test or condition that will result in two possible paths: one for when the condition is true, "yes"; and one for when the condition is false, "no".



Basic Flowchart Symbols

Using flow charts, programs can be divided into subroutines. Subroutines break a large program into smaller sections. In the following Traffic Light program, it is desired to sequence through red, green, and yellow lights. In this program, each subroutine turns on a different color light. The flowchart of the following Traffic Light program shows the fundamental flow control structures. All computer programs can be created using three fundamental control structures: sequence, selection, and iteration. In this program, the sequence of subroutines turns on each light. A consecutive series of steps is a **sequence control structure**. The connecting line returns to the start of the sequence to show repetition. A repetition of steps is a loop or **iteration control structure**. In the RedLightX subroutine, if the remote signal is on (the condition), the red light is on 5 seconds; otherwise, the red light is on 30 seconds. This is a selection or decision control structure. A **selection control structure** bases the next step on a condition that is true or false.



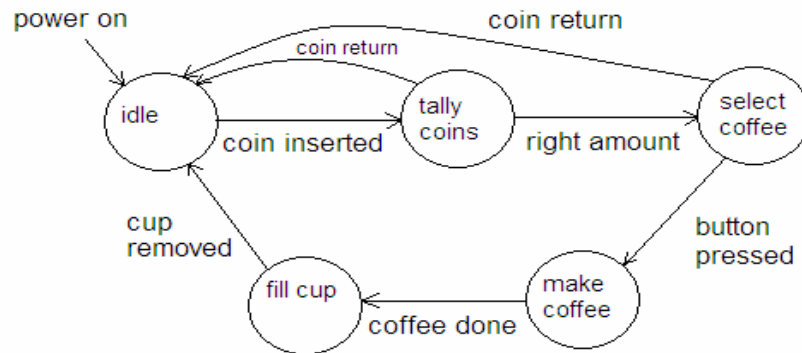
Pseudocode is a design tool that describes the instructions in an outline fashion. Pseudocode is similar to programming languages, so coding into a computer language can be easier. The following pseudocode is an example of a simple vending machine program design that dispenses coffee or coke and calculates the change. In the beginning of a pseudocode program, one defines the variables used in the program.

```
VARIABLES
price $0.50, $1.00
change $0.00, $0.25, $0.50
quarter = $0.25
selection = "coffee", "coke"
dollarReader = True, False
```

Pseudocode programs include words and symbols that describe computer operations such as input, output, AND, OR, add (+), subtract (-), multiply (*) and divide (/). The "=" can mean a comparison of two values, or an assignment of a value to a variable. Pseudocode can implement the fundamental control structures: series of instructions (sequence); "if...then...else" statements (selection); "do while" statements for repeating instructions based on a condition being true(iteration). These are shown below.

```
START:
output "Make A Selection"
input selection
if selection = "coffee" then
    price = $0.50
else
    price = $1.00
endif
if dollarReader = True
    change = $1.00 - price
    dowhile change is greater than or equal to $0.25
        dispense a quarter
        change = change - $0.25
    endwhile
    dispense selection
    output "Thank You"
endif
END
```

State diagrams define a system using states and transitions between states. States are written in the circles and directional lines show the transitions to the next state. Only one state is active at one time. A state diagram for a simple coffee machine is given below. Here we can see that when powered up, the machine will start in an idle state. The transitions are based on the inputs of sensors in the coffee machine. The states can indicate the operations that need to be further defined in detail.

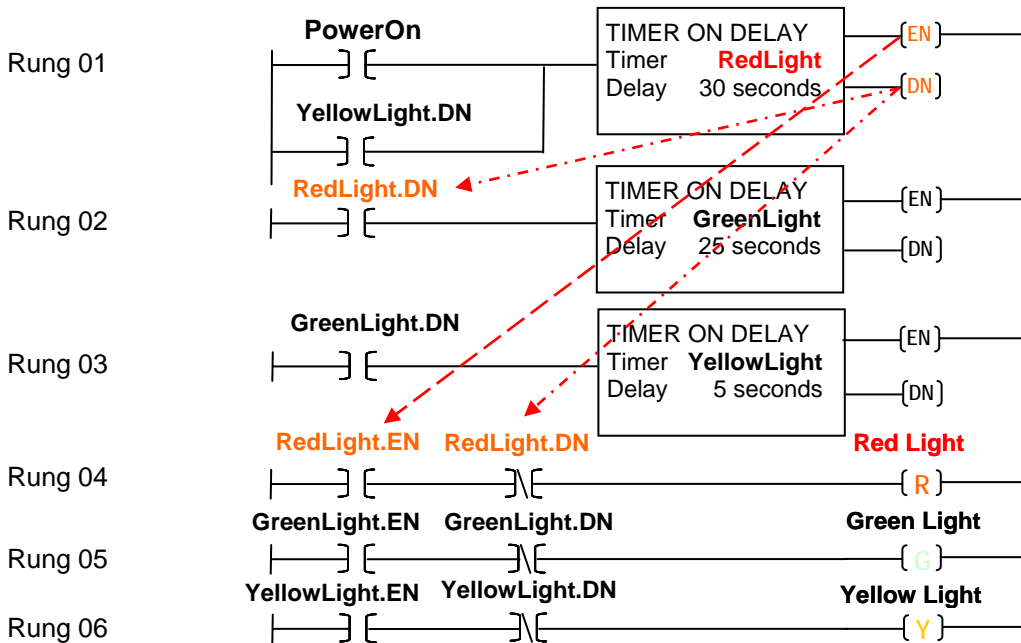


Coding the program

Coding is the conversion of the program design into a formal programming language. The programming language used can depend on whether the language is available for the controller or computing device. It can also depend on the skills of the personnel available for the programming task. The following includes examples of programs using the IEC standard controller programming languages. The end result of coding of the program is referred to as source code.

Ladder Diagram (LD) language is a graphical language based on relay ladder logic. This is the popular method of programming controllers. Ladder Diagrams are based on relay and switching logic. It is more intuitive and easy for electricians and electrical technicians to understand. Program instructions are entered on "rungs". Input components are inserted on the left side and the outputs on the right. Common input components are Normally Open and Normally Closed contacts. Common outputs are relay solenoids, motors, valves and lights. Ladder diagrams can also include advanced functions such as timers like the one shown below. A Traffic Light sequencing program is shown below:

The diagram looks complicated, but it is simple. Each light has a timer and the timer turns on the light. After a preset wait time, the timer turns off the light and starts the next timer in the sequence. The diagram shows the timer enable (EN) output turning on the light by closing normally open (EN) contacts. After the preset wait time, the timer done (DN) output turns off the light by opening normally closed (DN) contacts and starts the next timer by closing the normally open RedLight.DN contacts.



Ladder diagram programs can also include calculations, counting, control loops and subroutines. The ladder diagram strength is in performing switching and relay logic.

Instruction list (IL) language is more closely related to the instructions that the computer executes. Mnemonics designate computer operations. A computer programmer is more familiar with this approach. The following program reduces the voltage if a motor speed is greater than 1000. It also turns on a green light.

	LD	speed	(* Load speed into storage register *)
	GT	1000	(* Is speed greater than 1000 *)
	JMPCN	Volts_OK	(* If speed is NOT greater than 1000, go to Volts_OK *)
	LD	volts	(* Load voltage value into storage register *)
	SUB	10	(* Subtract 10 from voltage value*)
	ST	volts	(* Store new voltage output value *)
Volts_OK	LD	1	(* Load 1 in storage register *)
	ST	Green_Light	(* Store 1 at Green_Light to turn on green light *)

The IL language can perform logical operations (AND, OR, NOT) ; calculations using ADD, SUB, MUL, DIV instructions; selections and loops. This IL program uses instructions for a conditional test (GT) and a conditional jump (JMPCN) to code the selection structure. The IL language can also include subroutines and functions such as timers. Although this is not as intuitive as ladder logic, it is easier to have the instruction list language implemented on a programmable controller. Programming using the Instruction List language is similar to using assembler code and can result in fast executing code.

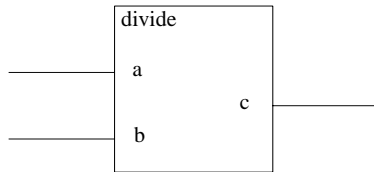
Structured Text (ST) is a structured high-level programming language that is similar to the Pascal programming language. The language is designed for easier coding of control structures. This programming is more suited for an engineer with a programming

background or a computer programmer. ST programs require identifying the data types used in the program. The variables used for inputs, outputs and calculations are “declared” at the beginning of the program and their data type are identified. For example, the Run_Motor function uses six variables: start, stop, current, volts, motor, and mpower. Each variable has an associated data type. This program is used to turn on a motor when the start button is pressed, and calculate the electrical power. The motor output variable represents the relay that applies power to the motor. The mpower output variable represents the value of electrical power. The “*” is the multiply operation to calculate the power from current and volts. Other mathematic operations used in structured text are add (+), subtract (-), and divide (/).

```
(* Switching Logic and Calculation Example *)  
FUNCTION Run_Motor  
  INPUT_VAR  
    start : BOOL; (* A Normally Open start input *)  
    stop  : BOOL; (* A Normally Closed stop input *)  
    current, volts: REAL;  
  END_VAR  
  OUTPUT_VAR  
    motor : BOOL;(* Motor control relay output *)  
    mpower: REAL;(* Motor power *)  
  END_VAR  
  motor := (motor OR start) AND stop;(* Run motor *)  
  mpower := current * volts;(* Power calculation *)  
END_FUNCTION
```

Structure Text also includes “IF... THEN... ELSE” statements for implementing the selection structure and “WHILE...DO” statements for iterations. The “WHILE...DO” statement is useful for checking whether a sensor is on, or a setting is reached, like a time setting, temperature setting or tank level. The “FOR...DO” statements are used for a fixed number of repetitions. This is useful for situations involving a fixed count. ST strength is in performing complex calculations and it is easy to read and understand.

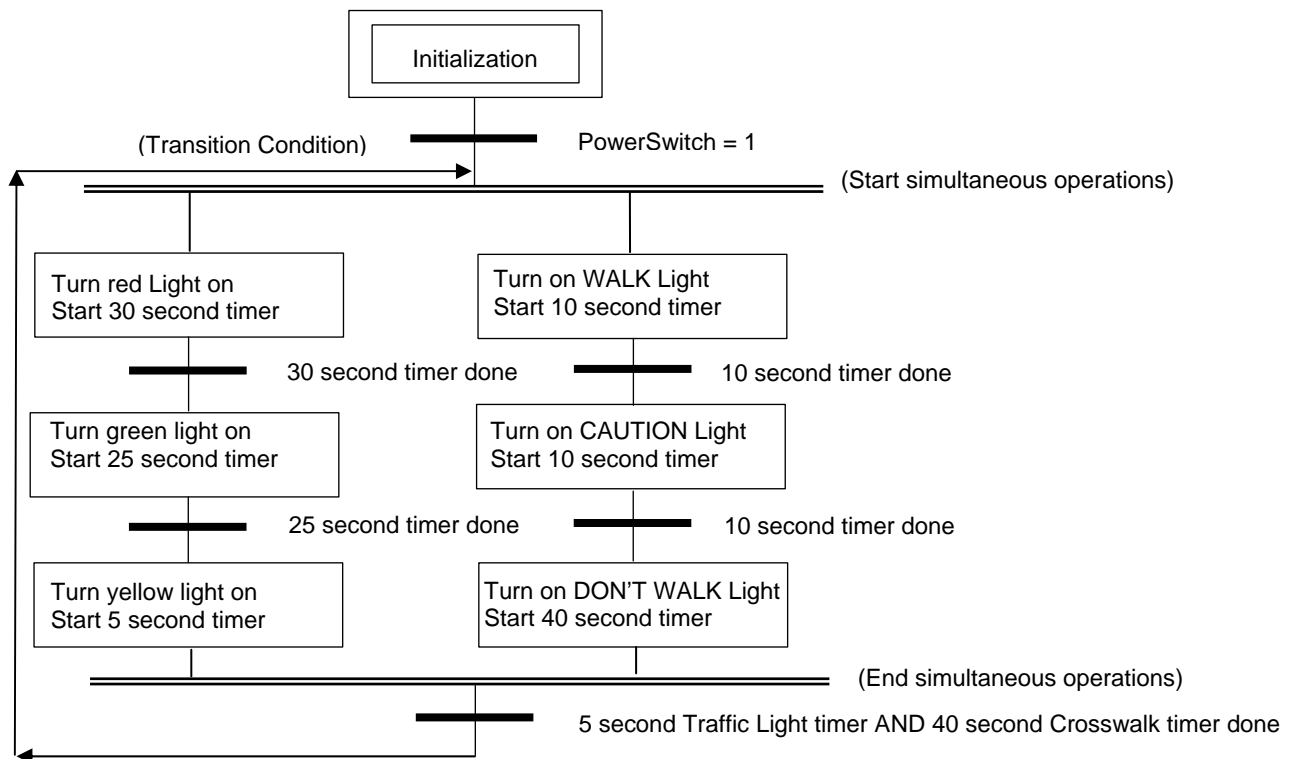
Function Block Diagram (FBD) is a graphical language that uses function blocks and connections that show data and signal flow. The FBD is more common in the process industries. The following compares a function block and structured text. This shows a divide function block. A divide by zero can cause serious errors for a computer and can halt operations. This function block checks to be sure “b” is not zero, before proceeding. If b is 0, it provides a 0 output.



```

FUNCTION_BLOCK divide
VAR_INPUT
    a: INT;
    b: INT;
END_VAR
VAR_OUTPUT
    c: INT;
END_VAR
    IF b <> 0 THEN
        c := a / b;
    ELSE
        c := 0;
    END_IF;
END_FUNCTION_BLOCK
    
```

Sequential Function Chart (SFC) is a charting methodology that is also a graphical programming language. The SFC is a design tool that is focused on describing the sequential behavior of a control program. The basic elements of an SFC diagram are shown in the Traffic Light program below:



In this example, the program sequences traffic lights through red, green and yellow and also sequences a crosswalk (WALK/DON'T WALK) sign, simultaneously. Each step shown has actions and a transition condition (bar). The transition condition needs to be met before proceeding to the next step. The transition condition for this case is that the timer must complete the set time. Unlike other languages, the SFC allows simultaneous execution of steps in parallel branches. SFC also allows steps to be programmed in other languages. The strength of SFC is the capability for synchronization of steps. It is also suited for complex control problems.

Other High Level Languages and Programs. In addition to the previously mentioned standard languages, controllers can be programmed in higher level languages like the C programming language. C is an ANSI standard language; however, it is not included in the controller programming language standard. Another language that is not a standard controller language is Visual Basic. The programming of visual display interfaces does not usually require the precision timing of controller programs. Visual Basic is a programming language that includes many features that can allow one to create visual displays of processes.

The previous examples show traffic light sequencing, a motor control, and calculations in the different programming languages. There are many systems that use computing devices that require programming: process control, alarm systems, HVAC, robots, assembly lines, motion control, gas monitoring, and power generation. The programs can involve many lines of code and can be in different languages. This has provided an overview of the different programming languages and code one could see.

Compiling, Testing and Maintenance

Compiling is the process of converting the instructions in the programming language to machine code instructions. The closer the language is to the machine instructions, the less complicated the compiler needs to be. Compilers check the source instructions for common errors. If the compiler can not convert the source code to machine code, it usually indicates that it can't, so corrections to the program can be made. Once the program is converted to machine code it can be executed or run. PLC's have programming software for creating, compiling and running programs. RSlogix software for the Allen Bradley PC-5 controllers is an example. Microsoft Visual Studio is an integrated program development system that allows one to create, compile and run programs in a number of languages. Testing is the process of running scenarios or test cases to verify that the program provides the intended outputs and operates as expected. When there is confidence that all errors are corrected, the program is made available for users. This is the last step in which final documentation is prepared and maintenance is performed. Even though testing can eliminate most errors, some errors will occur through program usage. Maintenance involves the correcting of the errors that users discover.

Conclusion

Programming design can involve a number of methodologies for designing and implementing a program. Hierarchical charts, flowcharts and pseudocode are good tools for designing programs. State diagrams are oriented on detailed digital logic and Sequential Function Charts are oriented for more complex control systems design. Similarly one can have a variety of programming languages for implementing the design: Ladder Diagrams, Instruction Lists, Structured Text, Function Blocks, and other programming languages like C and Visual Basic. The references below provide web sites that include more details in programming, demonstrations and evaluation software to explore programming further.

Bibliography

Farrell, Joyce, *Programming Logic and Design*, Course Technology, Boston, Mass 2002

Kernighan, Brian W. and Pike, Rob, *The Practice of Programming*, Addison-Wesley, Reading, Mass, 1999

Lewis, R. W., *Programming Industrial Control Systems using IEC 1131-3 Revised Edition*, Institute of Electrical Engineers, UK, 1998

Petruzella, Frank D., *Programmable Logic Controllers*, McGraw-Hill, N.Y.N.Y., 2005

Robertson, Lesley Anne, *Simple Program Design, Fourth Edition*, Course Technology, Boston, Mass 2004

Tucker, Allen B. et al, *Fundamentals of Computing I, Logic Problem Solving, Programs and Computers*, McGraw-Hill, N.Y.N.Y, 1995

Web Sites

Hugh Jack. Automated Manufacturing Systems; PLCs, GNU Free Document License 1996-2007 <http://claymore.engineer.gvsu.edu/~jackh/books/plcs/>

PLCopen for efficiency in automation. PLCopen.org. 2004. <http://plcopen.org>

Tim Young. Tools for PLC Programming. 2007. <http://www.plcdev.com/>

LogixPro 500. The Learning Pit Dot Com. 2007. <http://www.thelearningpit.com/lp/logixpro.html>

Hardware Development – Projects you can build. Microsoft MSDN. 2007. <http://msdn.microsoft.com/vstudio/express/hardware/>

Bjarne Stroustrup. The C++ Programming Language. AT&T labs, Inc and TAMU. 2007. <http://www.research.att.com/~bs/C++.html>