**CHAPTER 2**

**INTRODUCTION TO EMBEDDED SYSTEMS**

**2.1 EMBEDDED SYSTEM:**

Embedded System is a combination of hardware and software used to achieve a single specific task. An embedded system is a microcontroller-based, software driven, reliable, real-time control system, autonomous, or human or network interactive, operating on diverse physical variables and in diverse environments and sold into a competitive and cost conscious market.

An embedded system is not a computer system that is used primarily for processing, not a software system on PC or UNIX, not a traditional business or scientific application. High-end embedded & lower end embedded systems. High-end embedded system - Generally 32, 64 Bit Controllers used with OS. Examples Personal Digital Assistant and Mobile phones etc .Lower end embedded systems - Generally 8,16 Bit Controllers used with an minimal operating systems and hardware layout designed for the specific purpose. Examples Small controllers and devices in our everyday life like Washing Machine, Microwave Ovens, where they are embedded in.

SYSTEM DESIGN CALLS:

**2.1.1 EMBEDDED SYSTEM DESIGN CYCLE**

“V Diagram”

In this place we need to discuss the role of simulation software, real-time systems and data acquisition in dynamic test applications. Traditional testing is referred to as “static” testing where functionality of components is tested by providing known inputs and measuring outputs. Today there is more pressure to get products to market faster and reduce design cycle times.   
This has led to a need for “dynamic” testing where components are tested while in use with the entire system – either real or simulated. Because of cost and safety concerns, simulating the rest of the system with real-time hardware is preferred to testing components in the actual real system.

The diagram shown on this slide is the “V Diagram” that is often used to describe the development cycle. Originally developed to encapsulate the design process of software applications, many different versions of this diagram can be found to describe different product design cycles. Here we have shown one example of such a diagram representing the design cycle of embedded control applications common to automotive, aerospace and defense applications.

In this diagram the general progression in time of the development stages is shown from left to right. Note however that this is often an iterative process and the actual development will not proceed linearly through these steps. The goal of rapid development is to make this cycle as efficient as possible by minimizing the iterations required for a design. If the x-axis of the diagram is thought of as time, the goal is to narrow the “V” as much as possible and thereby reduce development time.

The y-axis of this diagram can be thought of as the level at which the system components are considered. Early on in the development, the requirements of the overall system must be considered. As the system is divided into sub-systems and components, the process becomes very low-level down to the point of loading code onto individual processors. Afterwards components are integrated and tested together until such time that the entire system can enter final production testing. Therefore the top of the diagram represents the high-level system view and the bottom of the diagram represents a very low-level view.

Notes:

* V diagram describes lots of applications—derived from software development.
* Reason for shape, every phase of design requires a complimentary test phase. High-level to low-level view of application.
* This is a simplified version.
* Loop Back/Iterative process, X-axis is time (sum up).

**2.1.2 CHARACTERISTICS OF EMBEDDED SYSTEM**

* An embedded system is any computer system hidden inside a product other than a computer.
* There will encounter a number of difficulties when writing embedded system software in addition to those we encounter when we write applications
  + ***Throughput*** – Our system may need to handle a lot of data in a short period of time.
  + ***Response***–Our system may need to react to events quickly
  + ***Testability***–Setting up equipment to test embedded software can be difficult
  + ***Debugability***–Without a screen or a keyboard, finding out what the software is doing wrong (other than not working) is a troublesome problem
  + ***Reliability*** – embedded systems must be able to handle any situation without human intervention
  + ***Memory space*** – Memory is limited on embedded systems, and you must make the software and the data fit into whatever memory exists
  + ***Program installation*** – you will need special tools to get your software into embedded systems
  + ***Power consumption*** – Portable systems must run on battery power, and the software in these systems must conserve power
  + ***Processor hogs*** – computing that requires large amounts of CPU time can complicate the response problem
  + ***Cost*** – Reducing the cost of the hardware is a concern in many embedded system projects; software often operates on hardware that is barely adequate for the job.
* Embedded systems have a microprocessor/ microcontroller and a memory. Some have a serial port or a network connection. They usually do not have keyboards, screens or disk drives.

**2.2 APPLICATIONS**

1. Military and aerospace embedded software applications

2. Communication applications

3. Industrial automation and process control software

**2.3 CLASSIFICATION**

1. Real Time Systems.
2. RTS is one which has to respond to events within a specified deadline.
3. A right answer after the dead line is a wrong answer

**2.3.1 RTS CLASSIFICATION**

1. Hard Real Time Systems

1. Soft Real Time System

**2.3.1.1 HARD REAL TIME SYSTEM**

* + - "Hard" real-time systems have very narrow response time.
    - Example: Nuclear power system, Cardiac pacemaker.

**2.3.1.2 SOFT REAL TIME SYSTEM**

* "Soft" real-time systems have reduced constrains on "lateness" but still must operate very quickly and repeatable.
* Example: Railway reservation system – takes a few extra seconds the data remains valid.

**LANGUAGES USED**

* C
* C++
* Java
* Linux
* Ada
* Assembly

**CHAPTER 3**

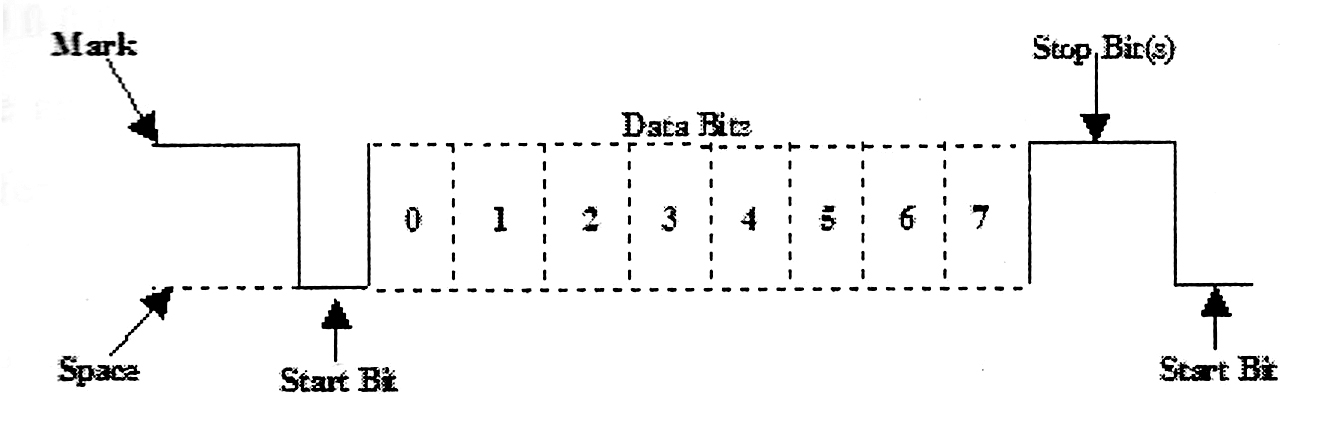
**Serial Communication**

**3.1 INTRODUCTION**

Serial communication is basically the transmission or reception of data one bit at a time. Today's computers generally address data in bytes or some multiple thereof. A byte contains 8 bits. A bit is basically either a logical 1 or zero. Every character on this page is actually expressed internally as one byte. The serial port is used to convert each byte to a stream of ones and zeroes as well as to convert a stream of ones and zeroes to bytes. The serial port contains a electronic chip called a Universal Asynchronous Receiver/Transmitter (UART) that actually does the conversion.

The serial port has many pins. We will discuss the transmit and receive pin first. Electrically speaking, whenever the serial port sends a logical one (1) a negative voltage is effected on the transmit pin. Whenever the serial port sends a logical zero (0) a positive voltage is affected. When no data is being sent, the serial port's transmit pin's voltage is negative (1) and is said to be in a MARK state. Note that the serial port can also be forced to keep the transmit pin at a positive voltage (0) and is said to be the SPACE or BREAK state. (The terms MARK and SPACE are also used to simply denote a negative voltage (1) or a positive voltage (0) at the transmit pin respectively).

When transmitting a byte, the UART (serial port) first sends a START BIT which is a positive voltage (0), followed by the data (general 8 bits, but could be 5, 6, 7, or 8 bits) followed by one or two STOP Bits which is a negative(l) voltage. The sequence is repeated for each byte sent. Figure shows a diagram of what a byte transmission would look like.

****

*Fig 3.1 Byte Transmission*

At this point you may want to know what the duration of a bit is. In other words, how long does the signal stay in a particular state to define a bit. The answer is simple. It is dependent on the baud rate. The baud rate is the number of times the signal can switch states in one second. Therefore, if the line is operating at 9600 baud, the line can switch states 9,600 times per second. This means each bit has the duration of 1 '9600 of a second or about 100µsec.

When transmitting a character there are other characteristics other than the baud rate that must be known or that must be setup. These characteristics define the entire interpretation of the data stream.

The first characteristic is the length of the byte that will be transmitted. This length in general can be anywhere from 5 to 8 bits.

The second characteristic is parity. The parity characteristic can be even, odd, mark, space, or none. If even parity, then the last data bit transmitted will be a logical 1 if the data transmitted had an even amount of 0 bits. If odd parity, then the last data bit transmitted will be a logical 1 if the data transmitted had an odd amount of 0 bits. If MARK parity, then the last transmitted data bit will always be a logical 1. If SPACE parity, then the last transmitted data bit will always be a logical 0. If no parity then there is no parity bit transmitted.

The third characteristic is the amount of stop bits. This value in general is 1 or 2. Assume we want to send the letter A' over the serial port. The binary representation of the letter 'A' is 01000001. Remembering that bits are transmitted from least significant bit (LSB) to most significant bit (MSB), the bit stream transmitted would be as follows for the line characteristics 8 bits, no parity, 1 stop bit and 9600 baud. LSB (0100009101) MSB.

The above represents (Start Bit) (Data Bits) (Stop Bit). To calculate the actual byte transfer rate simply divide the baud rate by the number of bits that must be transferred for each byte of data. In the case of the above example, each character requires 10 bits to be transmitted for each character. As such, at 9600 baud, up to 960 bytes can be transferred in one second.

The above discussion was concerned with the "electrical/logical" characteristics of the data stream. We will expand the discussion to line protocol.

Serial communication can be half duplex or full duplex. Full duplex communication means that a device can receive and transmit data at the same time. Half duplex means that the device cannot send and receive at the same time. It can do them both, but not at the same time. Half duplex communication is all but outdated except for a very small focused set of applications.

Half duplex serial communication needs at a minimum two wires, signal ground and the data line. Full duplex serial communication needs at a minimum three wires, signal ground, transmit data line, and receive data line. The RS232 specification governs the physical and electrical characteristics of serial communications. This specification defines several additional signals that are asserted (set to logical 1) for information and control beyond the data signal.

These signals are the Carrier Detect Signal (CD), asserted by modems to signal a successful connection to another modem, Ring Indicator (RI), asserted by modems to signal the phone ringing. Data Set Ready (DSR), asserted by modems to show their presence, Clear To Send (CTS), asserted by modems if they can receive data, Data Terminal Ready (DTR), asserted by terminals to show their presence, Request To Send (RTS), asserted by terminals if they can receive data. The section R.S232 Cabling describes these signals and how they are connected.

The above paragraph alluded to hardware flow control. Hardware flow control is  
a method that two connected devices use to tell each other electronically when to send or  
when not to send data. A modem in general drops (logical 0) its CTS line when it can no  
longer receive characters. It re-asserts it when it can receive again. A terminal does the  
same thing instead with the RTS signal. Another method of hardware flow control in  
practice is to perform the same procedure in the previous paragraph except that the DSR  
and DTR signals are used for the handshake.

Note that hardware flow control requires the use of additional wires. The benefit to this however is crisp and reliable flow control. Another method of flow control used is known as software flow control. This method requires a simple 3 wire serial communication link, transmit data, receive data, and signal ground. If using this method, when a device can no longer receive, it will transmit a character that the two devices agreed on. This character is known as the XOFF character.

**CHAPTER 3**

**HARDWARE REQUIREMENTS**

**3.1 MICROCONTROLLER**

**Introduction to microcontroller:**

A computer-on-a-chip is a variation of a microprocessor which combines the processor core (CPU), some memory, and I/O (input/output) lines, all on one chip. The computer-on-a-chip is called the microcomputer whose proper meaning is a computer using a (number of) microprocessor(s) as its CPUs, while the concept of the microcomputer is known to be a microcontroller. A microcontroller can be viewed as a set of digital logic circuits integrated on a single silicon chip. This chip is used for only specific applications.

Most microcontrollers do not require a substantial amount of time to learn how to efficiently program them, although many of them, which have quirks, which you will have to understand before you, attempt to develop your first application.

Along with microcontrollers getting faster, smaller and more power efficient they are also getting more and more features. Often, the first version of microcontroller will just have memory and digital I/O, but as the device family matures, more and more pat numbers with varying features will be available.

In this project we used PIC 16f877A microcontroller. For most applications, we will be able to find a device within the family that meets our specifications with a minimum of external devices, or an external but which will make attaching external devices easier, both in terms of wiring and programming.

For many microcontrollers, programmers can built very cheaply, or even built in to the final application circuit eliminating the need for a separate circuit. Also simplifying this requirement is the availability of micro-controllers wit SRAM and EEPROM for control store, which will allow program development without having to remove the micro controller fro the application circuit.

**3.2 PIC MICROCONTROLLER CORE FEATURES:**

* High-performance RISC CPU.
* Only 35 single word instructions to learn.
* All single cycle instructions except for program branches which are two cycle.
* Operating speed: DC - 20 MHz clock input DC - 200 ns instruction cycle.
* Up to 8K x 14 words of FLASH Program Memory, Up to 368 x 8 bytes of Data Memory (RAM) Up to 256 x 8 bytes of EEPROM data memory.
* Pin out compatible to the PIC16C73B/74B/76/77
* Interrupt capability (up to 14 sources)
* Eight level deep hardware stack
* Direct, indirect and relative addressing modes.
* Power-on Reset (POR).
* Power-up Timer (PWRT) and Oscillator Start-up Timer (OST).
* Watchdog Timer (WDT) with its own on-chip RC oscillator for reliable operation.
* Programmable code-protection.
* Power saving SLEEP mode.
* Selectable oscillator options.
* Low-power, high-speed CMOS FLASH/EEPROM technology.
* Fully static design.
* In-Circuit Serial Programming (ICSP) .
* Single 5V In-Circuit Serial Programming capability.
* In-Circuit Debugging via two pins.
* Processor read/write access to program memory.
* Wide operating voltage range: 2.0V to 5.5V.
* High Sink/Source Current: 25 mA.
* Commercial and Industrial temperature ranges.
* Low-power consumption.

In this project we used PIC 16f877A microcontroller. PIC means Peripheral Interface Controller. The PIC family having different series. The series are 12- Series, 14- Series, 16- Series, 18- Series, and 24- Series. We used 16 Series PIC microcontrollers.

**3.3 ADVANTAGES OF USING A MICROCONTROLLER OVER MICROPROCESSOR:**

A designer will use a Microcontroller to

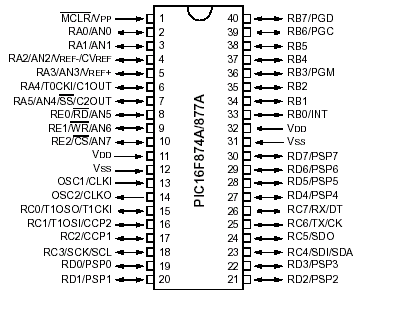
* Gather input from various sensors
* Process this input into a set of actions
* Use the output mechanisms on the Microcontroller to do something useful
* RAM and ROM are inbuilt in the MC.
* Cheap compared to MP.
* Multi machine control is possible simultaneously.

**Examples**  8051 (ATMEL), PIC (Microchip), Motorola (Motorola), ARM Processor.

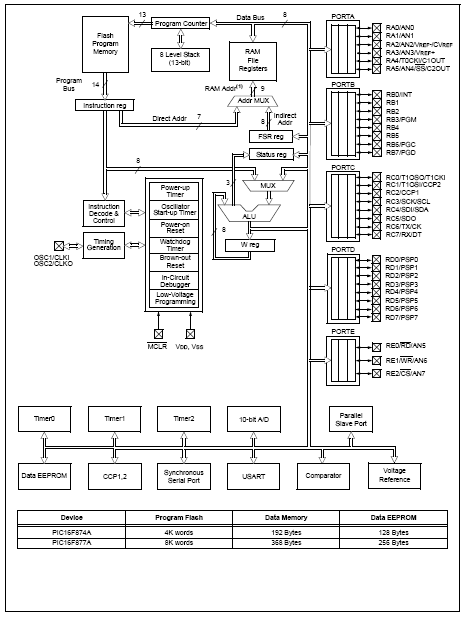
**APPLICATIONS:**

* Cell phones.
* Computers.
* Robots.
* Interfacing to two PC’s.

**3.4 pin diagram pic 16 f874a/877a:**



**3.5 FUNCTIONAL BLOCK DIAGRAM OF PIC 16F877A:**



**PIN DESCRIPTION:**

**OSC1/CLKI**:

Oscillator crystal or external clock input. Oscillator crystal input or external clock source input. ST buffer when configured in RC mode; otherwise CMOS. External clock source input. Always associated with pin function OSC1 (see OSC1/CLKI, OSC2/CLKO pins).

**OSC2/CLKO:**

Oscillator crystal or clock output. Oscillator crystal output. Connects to The crystal or resonator in Crystal Oscillator mode. In RC mode, OSC2 pin outputs CLKO, which has 1/4 the frequency of OSC1 and denotes the instruction cycle rate.

**MCLR/VPP:**

Master Clear (input) or programming voltage (output). Master Clear (Reset) input. This pin is an active low Reset to the device. Programming voltage input.

* RA0/AN0.
* RA1/AN1.
* RA2/AN2/VREF-/CVREF.
* VREFCVREF.
* RA3/AN3/VREF+.
* VREF+.
* RA4/T0CKI/C1OUT.
* T0CKI.
* C1OUT.
* RA5/AN4/SS/C2OUT/SS/C2OUT.

**I/O PORTS:**

Some pins for these I/O ports are multiplexed with an alternate function for the peripheral features on the device. In general, when a peripheral is enabled, that pin may not be used as a general purpose I/O pin.

**PORT A and TRIS A Register:**

PORT A is a 6-bit wide, bidirectional port. The corresponding data direction register is TRIS A. Setting a TRIS A bit (= 1) will make the corresponding PORT A pin an input (i.e., put the corresponding output driver in a High – Impedance mode). Clearing a TRIS A bit (= 0) will make the corresponding PORT A pin an output (i.e., put the contents of the output latch on the selected pin). Reading the PORT A register reads the status of the pins, whereas writing to it will write to the port latch. All write operations are read-modify-write operations. Therefore, a write to a port implies that the port pins are read; the value is modified and then written to the port data latch.

Pin RA4 is multiplexed with the Timer0 module clock input to become the RA4/T0CKI pin. The RA4/T0CKI pin is a Schmitt Trigger input and an open-drain output. All other PORT A pins have TTL input levels and full CMOS output drivers. Other PORT A pins are multiplexed with analog inputs and the analog VREF input for both the A/D converters and the comparators. The operation of each pin is selected by clearing/setting the appropriate control bits in the ADCON1 and/or CMCON registers. The TRIS A register controls the direction of the port pins even when they are being used as analog inputs. The user must ensure the bits in the TRIS A register are maintained set when using them as analog inputs.

**Note**: On a Power-on Reset, these pins are configured as analog inputs and read as ‘0’. The comparators are in the off (digital).

**PORT b and TRIS B Register:**

PORT B is an 8-bit wide, bidirectional port. The corresponding data direction register is TRIS B. Setting a TRIS B bit (= 1) will make the corresponding PORT B pin an input (i.e., put the corresponding output driver in a High-Impedance mode). Clearing a TRIS B bit (= 0) will make the corresponding PORT B pin an output (i.e., put the contents of the output latch on the selected pin). Three pins of PORT B are multiplexed with the In-Circuit Debugger and Low-Voltage Programming function: RB3/PGM, RB6/PGC and RB7/PGD.

Four of the PORT B pins, RB7:RB4, have an interruption- change feature. Only pins configured as inputs can cause this interrupt to occur (i.e., any RB7:RB4 pin configured as an output is excluded from the interruption- change comparison). The input pins (of RB7:RB4) are compared with the old value latched on the last read of PORTB. The “mismatch” outputs of RB7:RB4 are OR’ed together to generate the RB port change interrupt with flag bit RBIF (INTCON<0>).

This interrupt can wake the device from Sleep. The user, in the Interrupt Service Routine, can clear the interrupt in the following manner:

a) Any read or write of PORT B. This will end the mismatch condition.

b) Clear flag bit RBIF.

A mismatch condition will continue to set flag bit RBIF. Reading PORT B will end the mismatch condition and allow flag bit RBIF to be cleared. The interrupt-on-change feature is recommended for wake-up on key depression operation and operations where PORT B is only used for the interrupt-on-change feature. Polling of PORT B is not recommended while using the interrupt-on- change feature. This interrupt-on-mismatch feature, together with software configurable pull-ups on these four pins, allow easy interface to a keypad and make it possible for wake-up on key depression.

**PORT C and TRIS C Register:**

PORT C is an 8-bit wide, bidirectional port. The corresponding data direction register is TRIS C. Setting a TRIS C bit (= 1) will make the corresponding PORT C pin an input (i.e., put the corresponding output driver in a High- Impedance mode). Clearing a TRIS C bit (= 0) will make the corresponding PORT C pin an output (i.e., put the contents of the output latch on the selected pin). PORT C is multiplexed with several peripheral functions (Table 4-5). PORT C pins have Schmitt Trigger input buffers. When the I2C module is enabled, the PORT C<4:3> pins can be configured with normal I2C levels, or with SMBus levels, by using the CKE bit (SSPSTAT<6>). When enabling peripheral functions, care should be taken in defining TRIS bits for each PORT C pin. Some peripherals override the TRIS bit to make a pin an output, while other peripherals override the TRIS bit to make a pin an input. Since the TRIS bit override is in effect while the peripheral is enabled, read-modify write instructions (BSF, BCF, XORWF) with TRIS C as the destination, should be avoided. The user should refer to the corresponding peripheral section for the correct TRIS bit settings.

**PORT D and TRIS D Registers:**

PORT D is an 8-bit port with Schmitt Trigger input buffers. Each pin is individually configurable as an input or output. PORT D can be configured as an 8-bit wide microprocessor port (Parallel Slave Port) by setting control bit, PSP MODE (TRISE<4>). In this mode, the input buffers are TTL.

**PORT E and TRIS E Register:**

PORT E has three pins (RE0/RD/AN5, RE1/WR/AN6 and RE2/CS/AN7) which are individually configurable as inputs or outputs. These pins have Schmitt Trigger input buffers. The PORT E pins become the I/O control inputs for the microprocessor port when bit PSPMODE (TRISE<4>) is set. In this mode, the user must make certain that the TRIS E<2:0> bits are set and that the pins are configured as digital inputs. Also, ensure that ADCON1 is configured for digital I/O. In this mode, the input buffers are TTL. Register 4-1 shows the TRIS E register which also controls the Parallel Slave Port operation. PORT E pins are multiplexed with analog inputs.

When selected for analog input, these pins will read as ‘0’s. TRIS E controls. The direction of the RE pins, even when they are being used as analog inputs. The user must make sure to keep the pins configured as inputs when using them as analog inputs.

**MEMORY ORGANIZATION:**

There are three memory blocks in each of the PIC16F87XA devices. The program memory and data memory have separate buses so that concurrent access can occur and is detailed in this section. The EEPROM data memory block is detailed in.

**Program Memory Organization:**

The PIC16F87XA devices have a 13-bit program counter capable of addressing an 8K word x 14 bit program memory space. The PIC16F876A/877A devices have 8K words x 14 bits of Flash program memory, while PIC16F873A/874A devices have 4K words x 14 bits. Accessing a location above the physically implemented address will cause a wraparound. The Reset vector is at 0000h and the interrupt vector is at 0004h.

The data memory is partitioned into multiple banks which contain the General Purpose Registers and the Special Function Registers. Bits RP1 (Status<6>) and RP0 (Status<5>) are the bank select bits. Each bank extends up to 7Fh (128 bytes). The lower locations of each bank are reserved for the Special Function Registers. Above the Special Function Registers are General Purpose Registers, implemented as static RAM. All implemented banks contain Special Function Registers. Some frequently used Special Function Registers from one bank may be mirrored in another bank for code reduction and quicker access.

**TIMER0 MODULE:**

**The Timer0 module timer/counter has the following features**:

• 8-bit timer/counter

• Readable and writable

• 8-bit software programmable prescaler

• Internal or external clock select

• Interrupt on overflow from FFh to 00h

• Edge select for external clock

Timer mode is selected by clearing bit T0CS (OPTION\_REG<5>). In Timer mode, the Timer0 module will increment every instruction cycle (without prescaler). If the TMR0 register is written, the increment is inhibited for the following two instruction cycles. The user can work around this by writing an adjusted value to the TMR0 register.

**Timer0 Interrupt:**

The TMR0 interrupt is generated when the TMR0 register overflows from FFh to 00h. This overflow sets bit TMR0IF (INTCON<2>). The interrupt can be masked by clearing bit TMR0IE (INTCON<5>). Bit TMR0IF must be cleared in software by the Timer0 module Interrupt Service Routine before re-enabling this interrupt. The TMR0 interrupt cannot awaken the processor from Sleep since the timer is shut-off during Sleep.

**TIMER1 MODULE:**

The Timer1 module is a 16-bit timer/counter consisting of two 8-bit registers (TMR1H and TMR1L) which are readable and writable. The TMR1 register pair (TMR1H:TMR1L) increments from 0000h to FFFFh and rolls over to 0000h. The TMR1 interrupt, if enabled, is generated on overflow which is latched in interrupt flag bit, TMR1IF (PIR1<0>). This interrupt can be enabled/disabled by setting or clearing TMR1 interrupt enable bit, TMR1IE (PIE1<0>). Timer1 can operate in one of two modes:

• As a Timer

• As a Counter

The operating mode is determined by the clock select bit, TMR1CS (T1CON<1>).

In Timer mode, Timer1 increments every instruction cycle. In Counter mode, it increments on every rising edge of the external clock input. Timer1 can be enabled/disabled by setting/clearing control bit, TMR1ON (T1CON<0>).Timer1 also has an internal “Reset input”. This Reset can be generated by either of the two CCP modules. Shows the Timer1 Control register. When the

Timer1 oscillator is enabled (T1OSCEN is set), the RC1/T1OSI/CCP2 and RC0/T1OSO/T1CKI pins become inputs. That is, the TRISC<1:0> value is ignored and these pins read as ‘0’.

**TIMER2 MODULE:**

Timer2 is an 8-bit timer with a pre scaler and a post scaler. It can be used as the PWM time base for the PWM mode of the CCP module(s). The TMR2 register is readable and writable and is cleared on any device Reset. The input clock (FOSC/4) has a prescale option of 1:1, 1:4 or 1:16, selected by control bits T2CKPS1:T2CKPS0 (T2CON<1:0>). The Timer2 module has an 8-bit period register, PR2. Timer2 increments from 00h until it matches PR2 and then resets to 00h on the next increment cycle. PR2 is a readable and writable register. The PR2 register is initialized to FFh upon Reset. The match output of TMR2 goes through a 4-bit postscaler (which gives a 1:1 to 1:16 scaling inclusive) to generate a TMR2 interrupt (latched in flag bit, TMR2IF (PIR1<1>)). Timer2 can be shut-off by clearing control bit, TMR2ON (T2CON<2>), to minimize power consumption.

**In-Circuit Debugger:**

PIC16F87XA devices have a Watchdog Timer which can be shut-off only through configuration bits. It runs off its own RC oscillator for added reliability. There are two timers that offer necessary delays on power-up. One is the Oscillator Start-up Timer (OST), intended to keep the chip in Reset until the crystal oscillator is stable. The other is the Power-up Timer (PWRT), which provides a fixed delay of 72 ms (nominal) on power-up only. It is designed to keep the part in Reset while the power supply stabilizes. With these two timers on-chip, most applications need no external Reset circuitry. Sleep mode is designed to offer a very low current power-down mode. The user can wake-up from Sleep through external Reset, Watchdog Timer wake-up or through an interrupt. Several oscillator options are also made available to allow the part to fit the application. The RC oscillator option saves system cost while the LP crystal option saves power. A set of configuration bits is used to select various options.

**3.6 2X16 LCD:**

Most LCD programmed in 8 bit configuration. Moreover LCD put on equipment that show the value of measurement, i.e. temperature, voltage, current, etc. There are a lot of tutorial show steps how to configure out in order to LCD on. But each LCD has  own characteristic

**Basic Specifications**

|  |  |
| --- | --- |
| Power requirements | 4.8 to 5.5Vdc @ 3Ma |
| User connector | 5-pin header; 0.025" posts on 0.10" centers |
| Connector pinout | +5V GND SERIAL GND +5V |
| Serial Input | RS-232 or inverted TTL, 2400/9600, N81 |
| Operating Temperature | 0° to 50° C |
| Initialization | switches LCD power; performs soft init |
| Instruction prefix | ASCII 254 (0FE hex) |
| LCD type | Supertwist (STN), yellow-green |
| Optimum viewing direction | 6 o'clock |

**LCD Instructions by Function**

|  |  |
| --- | --- |
| **Function** | **ASCII Value** |
| Clear screen | 1 |
| Home cursor | 2 |
| Blank display (retaining data) | 8 |
| Hide cursor | 12 |
| Show underline cursor | 14 |
| Move cursor 1 character left | 16 |
| Move cursor 1 character right | 20 |
| Scroll 1 character left | 24 |
| Scroll 1 character right | 28 |
| Set display address (position the cursor) | 128 + location |
| Move to 1st character of 1st line | 128 |
| Move to *nth* character of 1st line | 128 + *n* |
| Move to 1st character of 2nd line | 192 |
| Move to *nth* character of 2nd line | 192 + *n* |
| Set character-generator address | 64 + address |

**3.7 ZIGBEE:**

ZigBee is the name of a specification for a suite of high level communication protocols using small, low-power digital radios based on the IEEE 802.15.4 standard for wireless personal area networks (WPANs), such as wireless headphones connecting with cell phones via short-range radio. The technology is intended to be simpler and cheaper than other WPANs, such as Bluetooth. ZigBee is targeted at radio-frequency (RF) applications which require a low data rate, long battery life, and secure networking.

**3.8 OVERVIEW:**

ZigBee builds upon the physical layer and medium access control defined in IEEE standard 802.15.4 (2003 version) for low-rate WPAN's. The specification goes on to complete the standard by adding four main components: network layer, application layer, ZigBee device objects (ZDO's) and manufacturer-defined application objects which allow for customization and favor total integration.

Besides adding two high-level network layers to the underlying structure, the most significant improvement is the introduction of ZDO's. These are responsible for a number of tasks, which include keeping of device roles, management of requests to join a network, device discovery and security. At its core, ZigBee is a mesh network architecture. Its network layer natively supports three types of topologies: both star and tree typical networks and generic mesh networks. Every network must have one coordinator device, tasked with its creation, the control of its parameters and basic maintenance. Within star networks, the coordinator must be the central node. Both trees and meshes allow the use of ZigBee routers to extend communication at the network level (they are not ZigBee coordinators, but may act as 802.15.4 coordinators within their personal operating space), but they differ in a few important details: communication within trees is hierarchical and optionally utilizes frame beacons, whereas meshes allow generic communication structures but no router beaconing. The relationship between IEEE 802.15.4-2003 and ZigBee is similar to that between IEEE 802.11 and the Wi-Fi Alliance. The ZigBee 1.0 specification was ratified on December 14, 2004 and is available to members of the ZigBee Alliance. An entry level membership called Adopter, in the ZigBee Alliance costs US$3500 annually and provides access to the specifications and permission to create products for market using the specifications. For non-commercial purposes, the ZigBee specification is available to the general public at the ZigBee Specification Download Request. Most recently, the ZigBee 2006 specification was posted in December 2006.3 ZigBee operates in the industrial, scientific and medical (ISM) radio bands; 868 MHz in Europe, 915 MHz in countries such as USA and Australia, and 2.4 GHz in most jurisdictions worldwide. The technology is intended to be simpler and cheaper than other WPANs such as Bluetooth. The most capable ZigBee node type is said to require only about 10% of the software of a typical Bluetooth or Wireless Internet node, while the simplest nodes are about 2% However, actual code sizes are much higher, closer to 50% of Bluetooth code size ZigBee chip vendors have announced 128-kilobyte devices. As of 2006, the retail price of a Zigbee-compliant transceiver is approaching $1, and the price for one radio, processor, and memory package is about $3. Comparatively, before Bluetooth was launched (1998) it had a projected price, in high volumes, of $4 - $6; the price of consumer-grade Bluetooth chips is now under $3. First stack release is now called "Zigbee 2004". The 2nd stack release is called 2006, and mainly replaces the MSG/KVP structure used in 2004 with a "cluster library". The 2004 stack is now more or less obsolete.

The ZigBee Alliance has started work on ZigBee 2007, looking to extend the ZigBee 2006 specification capabilities; the main enhancements are optimizing certain network level functionality (such as data aggregation).

There are also some new application profiles like Automatic Meter Reading, Commercial building automation and home automation based on the "cluster library principle". Zigbee 2007, now the current stack release, is sometimes called "Pro", but pro is a stack profile, which defines certain stack settings and mandatory features. ZigBee 2007 at the network level is not backwards-compatible with ZigBee 2004/2006, although a ZigBee 2004/2006 RFD node can join a 2007 network, and vice-versa. It's not possible to mix 2004/2006 routers with 2007 routers/coordinator.

**3.9 NETWORK LAYERS:**

The main functions of the network layer are to enable the correct use of the MAC sublayer and provide a suitable interface for use by the next upper layer, namely 4 the application layer. Its capabilities and structure are those typically associated to such network layers, including routing. On the one hand, the data entity creates and manages network layer data units from the payload of the application layer and performs routing according to the current topology. On the other hand, there is the layer control, which is used to handle configuration of new devices and establish new networks: it can determine whether a neighboring device belongs to the network and discovers new neighbors and routers. The control can also detect the presence of a receiver, which allows direct communication and MAC synchronization.

**3.10 APPLICATION LAYER:**

The application layer is the highest-level layer defined by the specification, and is the effective interface of the ZigBee system to its end users. It comprises the majority of components added by the ZigBee specification: both ZDO and its management procedures, together with application objects defined by the manufacturer, are considered part of this layer.

**3.10.1 Main Components:**

The ZDO is responsible for defining the role of a device as either coordinator or end device, as mentioned above, but also for the discovery of new (one-hop) devices on the network and the identification of their offered services. It may then go on to establish secure links with external devices and reply to binding requests accordingly. The application support sublayer (APS) is the other main standard component of the layer, and as such it offers a well-defined interface and control services. It works as a bridge between the network layer and the other components of the application layer: it keeps up-to-date binding tables in the form of a database, which can be used to find appropriate devices depending on the services that are needed and those the different devices offer. As the union between both specified layers, it also routes messages across the layers of the protocol stack.

**3.11 SECURITY SERVICES:**

As one of its defining features, ZigBee provides facilities for carrying out secure communications, protecting establishment and transport of cryptographic keys, cyphering frames and controlling devices. It builds on the basic security framework defined in IEEE 802.15.4. This part of the architecture relies on the correct management of symmetric keys and the correct implementation of methods and security policies.

**3.11.1 Basic security model:**

The basic mechanism to ensure confidentiality is the adequate protection of all keying material. Trust must be assumed in the initial installation of the keys, as well as in the processing of security information. In order for an implementation to globally work, its general correctness (e.g., conformance to specified behaviors) is assumed. Keys are the cornerstone of the security architecture; as such their protection is of paramount importance, and keys are never supposed to be transported through 8 an insecure channel. There is a momentary exception to this rule, which occurs during the initial phase of the addition to the network of a reviouslyunconfigured device. The ZigBee network model must take particular care of security considerations, as ad hoc networks may be physically accessible to xternal devices and the particular working environment cannot be foretold; likewise, different applications running concurrently and using the same ransceiver to communicate are supposed to be mutually trustworthy: for cost reasons the model does not assume a firewall exists between application-level ntities. Within the protocol stack, different network layers are not cryptographically parated, so access policies are needed and correct design assumed. The open trust model within a device allows for key sharing, which otably decreases potential cost. Nevertheless, the layer which creates a frame is responsible for its security. If malicious devices may exist, every network layer payload must be cyphered, so unauthorized traffic can be immediately cut off. The exception, again, is the transmission of the network key, which confers a unified security layer to the network, to a new connecting device. Point-to-point encryption is also supported.

**3.12 PROTOCOLS:**

The protocols build on recent algorithmic research (Ad-hoc On-demand Distance Vector, neuRFon) to automatically construct a low-speed ad-hoc network of nodes. In most large network instances, the network will be a cluster of clusters. It can also form a mesh or a single cluster. The current profiles derived from the ZigBee protocols support beacon and non-beacon enabled networks. In non-beacon-enabled networks (those whose beacon order is 15), an unslotted CSMA/CA channel access mechanism is used. In this type of network, ZigBee Routers typically have their receivers continuously active, requiring a more robust power supply. However, this allows for heterogeneous networks in which some devices receive continuously, while others only transmit when an external stimulus is detected. The typical example of a heterogeneous network is a wireless light switch: The ZigBee node at the lamp may receive constantly, since it is connected to the mains supply, while a battery-powered light switch would remain asleep until the switch is thrown. The switch then wakes up, sends a command to the lamp, receives an acknowledgment, and returns to sleep. In such a network the lamp node willbe at least a ZigBee Router, if not the ZigBee Coordinator; the switch node is typically a ZigBee End Device.

In beacon-enabled networks, the special network nodes called ZigBee Routers transmit periodic beacons to confirm their presence to other network nodes. Nodes may sleep between beacons, thus lowering their duty cycle and extending their battery life. Beacon intervals may range from 15.36 milliseconds to 15.36 ms \* 214 = 251.65824 seconds at 250 kbit/s, from 24 milliseconds to 24 ms \* 214 = 393.216 seconds at 40 kbit/s and from 48 milliseconds to 48 ms \* 214 = 786.432 seconds at 20 kbit/s. However, low duty cycle operation with long beacon intervals requires precise timing which can conflict with the need for low product cost.

In general, the ZigBee protocols minimize the time the radio is on so as to reduce power use. In beaconing networks, nodes only need to be active while a beacon is being transmitted. In non-beacon-enabled networks, power consumption is decidedly asymmetrical: some devices are always active, while others spend most of their time sleeping. ZigBee devices are required to conform to the IEEE 802.15.4-2003 Low-Rate Wireless Personal Area Network (WPAN) standard. The standard specifies the lower protocol layers—the physical layer (PHY), and the medium access control (MAC)

portion of the data link layer (DLL). This standard specifies operation in theunlicensed 2.4 GHz, 915 MHz and 868 MHz ISM bands. In the 2.4 GHz band there are 16 ZigBee channels, with each channel requiring 5 MHz of bandwidth. The center frequency for each channel can be calculated as, FC = (2405 + 5\*(k-11)) MHz, where k = 11, 12, ..., 26.

The radios use direct-sequence spread spectrum coding, which is managed by the digital stream into the modulator. BPSK is used in the 868 and 915 MHz bands, and orthogonal QPSK that transmits two bits per symbol is used in the 2.4 GHz band. The raw, over-the-air data rate is 250 kbit/s per channel in the 2.4 GHz band, 40 kbit/s per channel in the 915 MHz band, and 20 kbit/s in the 868 MHz band. Transmission range is between 10 and 75 meters (33 and 246 feet), although it is heavily dependent on the particular environment. The maximum output power of the radios is generally 0 dBm (1 mW).

The basic channel access mode specified by IEEE 802.15.4-2003 is "carrier sense, multiple access/collision avoidance" (CSMA/CA). That is, the nodes talk in the same way that people converse; they briefly check to see that no one is talking before they start. There are three notable exceptions to the use of CSMA. Beacons are sent on a fixed timing schedule, and do not use CSMA. Message acknowledgements also do not use CSMA. Finally, devices in Beacon Oriented networks that have low latency real-time requirements may also use Guaranteed Time Slots (GTS) which by definition does not use CSMA.

**3.13 NODE TYPES:**

This page describes the types of node that are used in a ZigBee network. Reference will be made to the toplogies introduced on the previous page (Star, Tree, Mesh), but these topologies will be described in more detail later in this module. The ZigBee standard has the capacity to address up to 65535 nodes in a single network. However, there are only three general types of node: These roles described below exist at the network level – a ZigBee node may also be performing tasks at the application level independent of the role it plays in the network. For instance, a network of ZigBee devices measuring temperature may have a temperature sensor application in each node, irrespective of whether they are End Devices, Routers or the Co-ordinator.These node types are described below.

**3.13.1 ZigBeecoordinator(ZC):**

The most capable device, the coordinator forms the root of the network tree and might bridge to other networks. There is exactly one ZigBee coordinator in each network since it is the device that started the network originally. It is able to store information about the network, including acting as the Trust Centre & repository for security keys. All ZigBee networks must have one (and only one) Co-ordinator, irrespective of the network topology.

In the Star topology, the Co-ordinator is the central node in the network. In the Tree and Mesh topologies, the Co-ordinator is the top (root) node in the network. · This is illustrated below, where the Co-ordinator is colour-coded in dark black. At the network level, the Co-ordinator is mainly needed at system initialisation. The tasks of the Co-ordinator at the network layer are:

* Selects the frequency channel to be used by the network (usually the one with the least detected activity)
* Starts the network
* Allows other devices to connect to it (that is, to join the network)

The Co-ordinator can also provide message routing (for example,

in a Star network), security management and other services.

In some circumstances, the network will be able to operate normally if the Coordinator fails or is switched off. This will not be the case if the Co-coordinator provides a routing path through the network (for instance, in a Star topology, where it is needed to relay messages). Similarly the Co-ordinator provides services at the Application layer and if these services are being used (for example, Co-coordinator binding), the Co-ordinator must be able to provide them at all times.

**3.13.2 ZigBee Router (ZR):** As well as running an application function a router can act as an intermediate router, passing data from other devices.

Networks with Tree or Mesh topologies need at least one Router. The main tasks of a Router are:

* Relays messages from one node to another
* Allows child nodes to connect to it

In a Star topology, these functions are handled by the Co-ordinator and, therefore, a Star network does not need Routers. In Tree and Mesh topologies, Routers are located as follows:

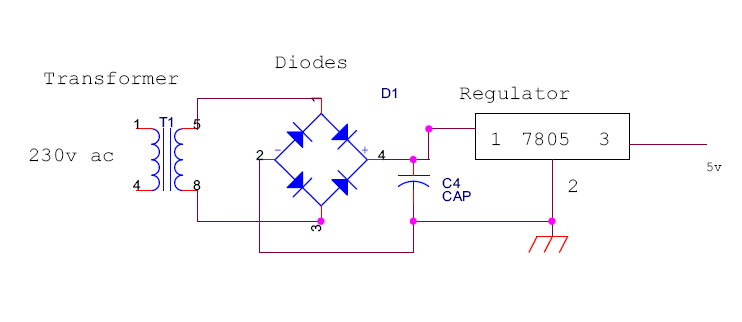
* In a Tree topology, Routers are normally located in network positions that allow messages to be passed up and down the tree.
* In a Mesh topology, a Router can be located anywhere that a message passing node is required.

However, in all topologies (Star, Tree and Mesh), Router devices can be located at the extremities of the network, if they run applications that are needed in these locations - in this case, the Router will not perform its message relay function, unless in a Mesh network (see above). The possible positions of Routers in the different network topologies are illustrated below, where the Routers are color-coded in red.

**3.14 USES:**

ZigBee protocols are intended for use in embedded applications requiring low data rates and low power consumption. ZigBee's current focus is to define a generalpurpose, inexpensive, self-organizing mesh network that can be used for industrial control, embedded sensing, medical data collection, smoke and intruder warning, building automation, home automation, etc. The resulting network will use very small amounts of power so individual devices might run for a year or two using the originally installed battery.

# 3.20 POWER SUPPLY UNIT



Power supply unit consists of following units**:**

1) Step down transformer

2) Rectifier unit

3) Input filter

4) Regulator unit

5) Output filter

3.20**.1 Stepdown transformer:**

The Step down Transformer is used to step down the main supply voltage from 230V AC to lower value. This 230 AC voltage cannot be used directly, thus it is stepped down. The Transformer consists of primary and secondary coils. To reduce or step down the voltage, the transformer is designed to contain less number of turns in its secondary core. The output from the secondary coil is also AC waveform. Thus the conversion from AC to DC is essential. This conversion is achieved by using the Rectifier Circuit/Unit.

Step down transformers can step down incoming voltage, which enables you to have the correct voltage input for your electrical needs.  For example, if our equipment has been specified for input voltage of 12 volts, and the main power supply is 230 volts, we will need a *step down transformer*, which decreases the incoming electrical voltage to be compatible with your 12 volt equipment.

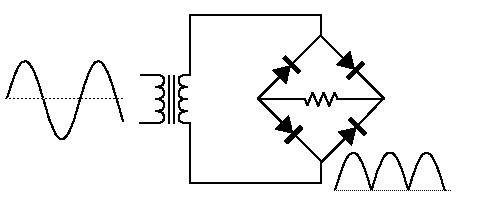
3.20**.2 Rectifier Unit:**

The Rectifier circuit is used to convert the AC voltage into its corresponding DC voltage. The most important and simple device used in Rectifier circuit is the diode. The simple function of the diode is to conduct when forward biased and not to conduct in reverse bias. Now we are using three types of rectifiers. They are

1. Half-wave rectifier
2. Full-wave rectifier
3. Bridge rectifier

|  |
| --- |
| **3.20.1 Half-wave rectifier:** In half wave rectification, either the positive or negative half of the AC wave is passed, while the other half is blocked. Because only one half of the input waveform reaches the output, it is very inefficient if used for power transfer. Half-wave rectification can be achieved with a single diode in a one phase supply, or with three diodes in a three-phase supply.  **3.20.2 Full-wave rectifier**: A full-wave rectifier converts the whole of the input waveform to one of constant polarity (positive or negative) at its output. Full-wave rectification converts both polarities of the input waveform to DC (direct current), and is more efficient. However, in a circuit with a non-center tapped transformer, four diodes are required instead of the one needed for half-wave rectification. A full-wave rectifier uses a diode bridge, made of four diodes, like this  diodeBridge  At first, this may look just as confusing as the one-way streets of Boston. The thing to realize is that the diodes work in pairs. As the voltage of the signal flips back and forth, the diodes shepard the current to always flow in the same direction for the output.  Here's what the circuit looks like to the signal as it alternates:  diodeBridge1  diodeBridge2  So, if we feed our AC signal into a full wave rectifier, we'll see both halves of the wave above 0 Volts. Since the signal passes through two diodes, the voltage out will be lower by two diode drops, or 1.2 Volts.  AC Wave In: ACin  AC Wave Out (Full-Wave Rectified): fullWaveOut  If we're interested in using the full-wave rectifier as a DC power supply, we'll add a smoothing capacitor to the output of the diode bridge. |

**3.20.3 Bridge rectifier:** A bridge rectifier makes use of four diodes in a bridge arrangement to achieve full-wave rectification. This is a widely used configuration, both with individual diodes wired as shown and with single component bridges where the diode bridge is wired internally.



A **diode bridge** or **bridge rectifier** is an arrangement of four [diodes](http://en.wikipedia.org/wiki/Diode) in a [bridge](http://en.wikipedia.org/wiki/Bridge_circuit) configuration that provides the same [polarity](http://en.wikipedia.org/wiki/Polarity_(physics)) of output [voltage](http://en.wikipedia.org/wiki/Volt) for either polarity of input voltage. When used in its most common application, for conversion of [alternating current](http://en.wikipedia.org/wiki/Alternating_current) (AC) input into [direct current](http://en.wikipedia.org/wiki/Direct_current) (DC) output, it is known as a bridge [rectifier](http://en.wikipedia.org/wiki/Rectifier). A bridge rectifier provides [full-wave rectification](http://en.wikipedia.org/wiki/Rectifier) from a two-wire AC input, resulting in lower cost and weight as compared to a [center-tapped](http://en.wikipedia.org/wiki/Center_tap) [transformer](http://en.wikipedia.org/wiki/Transformer) design.

The Forward Bias is achieved by connecting the diode’s positive with positive of the battery and negative with battery’s negative. The efficient circuit used is the Full wave Bridge rectifier circuit. The output voltage of the rectifier is in rippled form, the ripples from the obtained DC voltage are removed using other circuits available. The circuit used for removing the ripples is called Filter circuit.

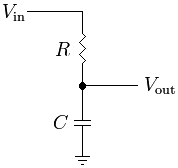
**3.21Input Filter**

Capacitors are used as filter. The ripples from the DC voltage are removed and pure DC voltage is obtained. And also these capacitors are used to reduce the harmonics of the input voltage. The primary action performed by capacitor is charging and discharging. It charges in positive half cycle of the AC voltage and it will discharge in negative half cycle. So it allows only AC voltage and does not allow the DC voltage. This filter is fixed before the regulator. Thus the output is free from ripples.

There are two types of filters. They are

1. Low pass filter
2. High pass filter

**3.21.1 Low pass filter:**



One simple [electrical circuit](http://en.wikipedia.org/wiki/Electrical_circuit) that will serve as a low-pass filter consists of a [resistor](http://en.wikipedia.org/wiki/Resistor) in series with a [load](http://en.wikipedia.org/wiki/External_electric_load), and a [capacitor](http://en.wikipedia.org/wiki/Capacitor) in parallel with the load. The capacitor exhibits [reactance](http://en.wikipedia.org/wiki/Reactance_(electronics)), and blocks low-frequency signals, causing them to go through the load instead. At higher frequencies the reactance drops, and the capacitor effectively functions as a short circuit. The combination of resistance and capacitance gives you the [time constant](http://en.wikipedia.org/wiki/Time_constant) of the filter τ = *RC* (represented by the Greek letter [tau](http://en.wikipedia.org/wiki/Tau)). The break frequency, also called the turnover frequency or [cutoff frequency](http://en.wikipedia.org/wiki/Cutoff_frequency) (in hertz), is determined by the time constant: or equivalently (in [radians](http://en.wikipedia.org/wiki/Radians) per second):

One way to understand this circuit is to focus on the time the capacitor takes to charge. It takes time to charge or discharge the capacitor through that resistor:

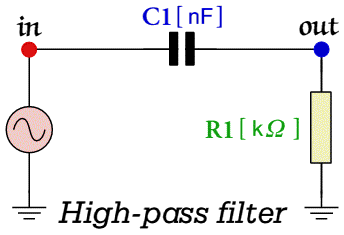
* At low frequencies, there is plenty of time for the capacitor to charge up to practically the same voltage as the input voltage.
* At high frequencies, the capacitor only has time to charge up a small amount before the input switches direction. The output goes up and down only a small fraction of the amount the input goes up and down. At double the frequency, there's only time for it to charge up half the amount.

Another way to understand this circuit is with the idea of [reactance](http://en.wikipedia.org/wiki/Reactance_(electronics)) at a particular frequency:

* Since [DC](http://en.wikipedia.org/wiki/Direct_current) cannot flow through the capacitor, DC input must "flow out" the path marked *V*out (analogous to removing the capacitor).
* Since [AC](http://en.wikipedia.org/wiki/Alternating_current) flows very well through the capacitor — almost as well as it flows through solid wire — AC input "flows out" through the capacitor, effectively [short circuiting](http://en.wikipedia.org/wiki/Short_circuit) to ground (analogous to replacing the capacitor with just a wire).

It should be noted that the capacitor is not an "on/off" object (like the block or pass fluidic explanation above). The capacitor will variably act between these two extremes. It is the [Bode plot](http://en.wikipedia.org/wiki/Bode_plot) and [frequency response](http://en.wikipedia.org/wiki/Frequency_response) that show this variability.

**3.21.2 High pass filter:**



The above circuit diagram illustrates a simple *'RC'* high-pass filter. we should find that the circuit passes 'high' frequencies fairly well, but attenuates 'low' frequencies. Hence it is useful as a filter to block any unwanted low frequency components of a complex signal whilst passing higher frequencies. Circuits like this are used quite a lot in electronics as a 'D.C. Block' - i.e. to pass a.c. signals but prevent any D.C. voltages from getting through.  
  
The basic quantities which describe this circuit are similar to those used for the [Low Pass Filter](http://www.st-andrews.ac.uk/~www_pa/Scots_Guide/experiment/lowpass/lpf.html). In effect, this circuit is just a simple low-pass filter with the components swapped over.

http://www.st-andrews.ac.uk/~www_pa/Scots_Guide/experiment/highpass/tau.gif

The action of the circuit can also be described in terms of a related quantity, the *Turn over Frequency*, *f0*, which has a value

http://www.st-andrews.ac.uk/~www_pa/Scots_Guide/experiment/highpass/f0.gif

As with the low-pass filter, the circuit's behavior we can be understood as arising due to the time taken to change the capacitor's charge when we alter the applied input voltage. It always takes a finite (i.e. non-zero) time to change the amount of charge stored by the capacitor. Hence it takes time to change the potential difference across the capacitor. As a result, any sudden change in the input voltage produces a similar sudden change on the other side of the capacitor. This produces a voltage across the resistor and causes a current to flow thorough it, charging the capacitor until all the voltage falls across it instead of the resistor. The result is that steady (or slowly varying) voltages appear mostly across the capacitor and quick changes appear mostly across the resistor. Since we're using the voltage across the resistor as out output the main properties of the circuit are

Therefore  
  
The *Voltage Gain*:

http://www.st-andrews.ac.uk/~www_pa/Scots_Guide/experiment/highpass/av.gif

The *Phase Delay*:

http://www.st-andrews.ac.uk/~www_pa/Scots_Guide/experiment/highpass/phi.gif

Try using the above experimental system to collect results and plot a graph of how the voltage gain, *Av*, (and the phase change) depend upon the input frequency and if we check result agrees with the above formulae. Compare this with a low-pass filter that uses the same component values and you should see that they give 'opposite' results. In the high-pass filter, the output waveform 'leads' the input waveform - i.e. it peaks before the input.

**3.22Regulator unit**

****

7805 Regulator

Regulator regulates the output voltage to be always constant. The output voltage is maintained irrespective of the fluctuations in the input AC voltage. As and then the AC voltage changes, the DC voltage also changes. Thus to avoid this Regulators are used. Also when the internal resistance of the power supply is greater than 30 ohms, the output gets affected. Thus this can be successfully reduced here. The regulators are mainly classified for low voltage and for high voltage. Further they can also be classified as:

i) Positive regulator

1---> input pin

2---> ground pin

3---> output pin

It regulates the positive voltage.

ii) Negative regulator

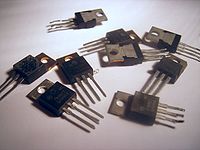
1---> ground pin

2---> input pin

3---> output pin

It regulates the negative voltage.

**Fixed regulators**

[](http://en.wikipedia.org/wiki/File:7800_IC_regulators.jpg)

An assortment of [78xx](http://en.wikipedia.org/wiki/78xx) series ICs

"Fixed" three-terminal linear regulators are commonly available to generate fixed voltages of plus 3 V, and plus or minus 5 V, 9 V, 12 V, or 15 V when the load is less than about 7 [amperes](http://en.wikipedia.org/wiki/Ampere).

## 7805 VOLTAGE REGULATOR:

## The **7805** provides circuit designers with an easy way to regulate DC voltages to 5v. Encapsulated in a single chip/package (IC), the **7805** is a positive voltage DC **regulator** that has only 3 terminals. They are: Input voltage, Ground, Output Voltage.

**General Features:**

* Output Current up to 1A
* Output Voltages of 5, 6, 8, 9, 10, 12, 15, 18, 24V
* Thermal Overload Protection
* Short Circuit Protection
* Output Transistor Safe Operating Area Protection

**7812 12V Integrated Circuit3-Terminal Positive Voltage Regulator:**

* The 7812 fixed voltage regulator is a monolithic integrated circuit in a TO220 type package designed for use in a wide variety of applications including local, onboard regulation. This regulator employs internal current limiting, thermal shutdown, and safe area compensation.
* With adequate heat-sinking it can deliver output currents in excess of 1.0 ampere. Although designed primarily as a fixed voltage regulator, this device can be used with external components to obtain adjustable voltages and currents.

**7.5 Output Filter**

The Filter circuit is often fixed after the Regulator circuit. Capacitor is most often used as filter. The principle of the capacitor is to charge and discharge. It charges during the positive half cycle of the AC voltage and discharges during the negative half cycle. So it allows only AC voltage and does not allow the DC voltage. This filter is fixed after the Regulator circuit to filter any of the possibly found ripples in the output received finally. Here we used 0.1µF capacitor. The output at this stage is 5V and is given to the Microcontroller. The output voltage overshoots when the load is removed or a short clears. When the load is removing from a switching mode power supply with a LC low-pass output filter, the only thing the control loop can do is stop the switching action so no more energy is taken from the source. The energy that is stored in the output filter inductor is dumped into the output capacitor causing a voltage overshoot.

The magnitude of the overshoot is the vector sum of two orthogonal voltages, the output voltage before the load is removed and the current through the inductor times the characteristic impedance of the output filter, Zo = (L/C)^1/2. This can be derived from conservation of energy considerations.

The initial energy, Ei, is:

Ei = 1/2\*(L\*Ii^2 + C\*Vi^2)

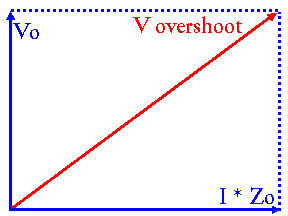
The final energy, Ef, is:

Ef = 1/2\*(L\*If^2 = C\*Vf^2)

The two energies are equal when the load is removed, since the load is no longer taking energy from the system. Equating the two energies, substituting zero current for the final inductor current, then the solution for the final voltage Vf is:

Vf = (Vi^2 + (Ii\*Zo)^2)^1/2

This is the orthogonal vector sum of the output voltage and the load current times the characteristic impedance and is illustrated in Figure 1.

  
Figure 1: Overshoot Voltage as Vector Sum

The problem becomes worse if the current in the inductor is established by a short circuit on the output and the short circuit clears. In this case, the initial voltage is zero (short circuit) and the overshoot is I\*Zo, where I can be very large, resulting in a ruinous overshooot

**CHAPTER 4**

**SOFTWARE REQUIREMENTS**

**8.1 Software Tools**

* Development tool – MPLAB IDE v7.42
* Hardware Compiler - HI-Tech PIC C
* Programmer - PIC Flash
* Hardware Simulation tool - Proteus v7.6Sp0

**8.2 INTRODUCTION TO EMBEDDED ‘C’:**

**Ex: Hitec – c, Keil – c**

HI-TECH Software makes industrial-strength software development tools and C compilers that help software developers write compact, efficient embedded processor code.

For over two decades HI-TECH Software has delivered the industry's most reliable embedded software development tools and compilers for writing efficient and compact code to run on the most popular embedded processors. Used by tens of thousands of customers including General Motors, Whirlpool, Qualcomm, John Deere and many others, HI-TECH's reliable development tools and C compilers, combined with world-class support have helped serious embedded software programmers to create hundreds of breakthrough new solutions.

Whichever embedded processor family you are targeting with your software, whether it is the ARM, PICC or 8051 series, HI-TECH tools and C compilers can help you write better code and bring it to market faster.

HI-TECH PICC is a high-performance C compiler for the Microchip PIC micro 10/12/14/16/17 series of microcontrollers. HI-TECH PICC is an industrial-strength ANSI C compiler - not a subset implementation like some other PIC compilers. The PICC compiler implements full ISO/ANSI C, with the exception of recursion. All data types are supported including 24 and 32 bit IEEE standard floating point. HI-TECH PICC makes full use of specific PIC features and using an intelligent optimizer, can generate high-quality code easily rivaling hand-written assembler. Automatic handling of page and bank selection frees the programmer from the trivial details of assembler code.

**8.3 Embedded “C” Compiler**

* ANSIC - full featured and portable
* Reliable - mature, field-proven technology
* Multiple C optimization levels
* An optimizing assembler
* Full linker, with overlaying of local variables to minimize RAM usage
* Comprehensive C library with all source code provided
* Includes support for 24-bit and 32-bit IEEE floating point and 32-bit long data types
* Mixed C and assembler programming
* Unlimited number of source files
* Listings showing generated assembler
* Compatible - integrates into the MPLAB IDE, MPLAB ICD and most 3rd-party development tools
* Runs on multiple platforms: Windows, Linux, UNIX, Mac OS X, Solaris

**8.4 MPLAB INTEGRATION**

MPLAB Integrated Development Environment (IDE) is a free, integrated toolset for the development of embedded applications employing Microchip's PIC micro and dsPIC microcontrollers. MPLAB IDE runs as a 32-bit application on MS Windows, is easy to use and includes a host of free software components for fast application development and super-charged debugging. MPLAB IDE also serves as a single, unified graphical user interface for additional Microchip and third party software and hardware development tools. Moving between tools is a snap, and upgrading from the free simulator to MPLAB ICD 2 or the MPLAB ICE emulator is done in a flash because MPLAB IDE has the same user interface for all tools.

Choose MPLAB C18, the highly optimized compiler for the PIC18 series microcontrollers, or try the newest Microchip's language tools compiler, MPLAB C30, targeted at the high performance PIC24 and dsPIC digital signal controllers. Or, use one of the many products from third party language tools vendors. They integrate into MPLAB IDE to function transparently from the MPLAB project manager, editor and compiler.

## Embedded Development Environment

This environment allows you to manage all of your PIC projects. You can compile, assemble and link your embedded application with a single step.

Optionally, the compiler may be run directly from the command line, allowing you to compile, assemble and link using one command. This enables the compiler to be integrated into third party development environments, such as Microchip's MPLAB IDE.

**8.5 Embedded system tools**

**8.5.1 ASSEMBLER**

An assembler is a [computer program](http://en.wikipedia.org/wiki/Computer_program) for translating [assembly language](http://en.wikipedia.org/wiki/Assembly_language) — essentially, a [mnemonic](http://en.wikipedia.org/wiki/Mnemonic) representation of [machine language](http://en.wikipedia.org/wiki/Machine_language) — into [object code](http://en.wikipedia.org/wiki/Object_code). A cross assembler (see [cross compiler](http://en.wikipedia.org/wiki/Cross_compiler)) produces code for one type of processor, but runs on another. The computational step where an assembler is run is known as assembly time. Translating assembly instruction mnemonics into [opcodes](http://en.wikipedia.org/wiki/Opcode), assemblers provide the ability to use symbolic names for memory locations (saving tedious calculations and manually updating addresses when a program is slightly modified), and [macro](http://en.wikipedia.org/wiki/Macro) facilities for performing textual substitution — typically used to encode common short sequences of instructions to run inline instead of in a [subroutine](http://en.wikipedia.org/wiki/Subroutine). Assemblers are far simpler to write than [compilers](http://en.wikipedia.org/wiki/Compiler) for [high-level languages](http://en.wikipedia.org/wiki/High-level_language).

**Assembly language has several benefits**

**Speed**: Assembly language programs are generally the fastest programs around.

**Space**: Assembly language programs are often the smallest.

**Capability:** You can do things in assembly which are difficult or impossible in High level languages.

**Knowledge**: Your knowledge of assembly language will help you write better programs, even when using High level languages. An example of an assembler we use in our project is RAD 51.

**8.5.2 SIMULATOR**

Simulator is a machine that simulates an environment for the purpose of training or research. We use a UMPS simulator for this purpose in our project.

**8.5.3 Compiler**

A compiler is a program that reads a program in one language, the source language and translates into an equivalent program in another language, the target language. The translation process should also report the presence of errors in the source program.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Source Program | → | Compiler | → | Target Program |
|  |  | ↓ |  |  |
|  |  | Error Messages |  |  |

There are two parts of compilation. The analysis part breaks up the source program into constant piece and creates an intermediate representation of the source program. The synthesis part constructs the desired target program from the intermediate representation.

**cousins of the compiler are**

1. Preprocessor.
2. Assembler.
3. Loader and Link-editor.

A naive approach to that front end might run the phases serially.

1. Lexical analyzer takes the source program as an input and produces a long string of tokens.
2. Syntax Analyzer takes an out of lexical analyzer and produces a large tree.
3. Semantic analyzer takes the output of syntax analyzer and produces another tree. Similarly, intermediate code generator takes a tree as an input produced by semantic analyzer and produces intermediate code

**Phases of compiler**

The compiler has a number of phases plus symbol table manager and an error handler.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Input Source Program |  |  |
|  |  | ↓ |  |  |
|  |  | Lexical Analyzer |  |  |
|  |  | ↓ |  |  |
|  |  | Syntax Analyzer |  |  |
|  |  | ↓ |  |  |
| Symbol Table Manager |  | Semantic Analyzer |  | Error Handler |
|  |  | ↓ |  |  |
|  |  | Intermediate Code Generator |  |  |
|  |  | ↓ |  |  |
|  |  | Code Optimizer |  |  |
|  |  | ↓ |  |  |
|  |  | Code Generator |  |  |
|  |  | ↓ |  |  |
|  |  | Out Target Program |  |  |

# FABRICATION DETAILS

The fabrication of one demonstration unit is carried out in the following sequence.

* Finalizing the total circuit diagram, listing out the components and sources of procurement.
* Procuring the components, testing the components and screening the components.
* Making layout, repairing the interconnection diagram as per the circuit diagram.
* Assembling the components as per the component layout and circuit diagram and soldering components.
* Integrating the total unit, interwiring the unit and final testing the unit.

**8.6 DESIGN OF EMBEDDED SYSTEM**

Like every other system development design cycle embedded system too have a design cycle. The flow of the system will be like as given below. For any design cycle these will be the implementation steps. From the initial state of the project to the final fabrication the design considerations will be taken like the software consideration and the hardware components, sensor, input and output. The electronics usually uses either a microprocessor or a microcontroller. Some large or old systems use general-purpose mainframe computers or minicomputers.

**8.6.1 USER INTERFACES**

User interfaces for embedded systems vary widely, and thus deserve some special comment. User interface is the ultimate aim for an embedded module as to the user to check the output with complete convenience. One standard interface, widely used in embedded systems, uses two buttons (the absolute minimum) to control a menu system (just to be clear, one button should be "next menu entry" the other button should be "select this menu entry").

Another basic trick is to minimize and simplify the type of output. Designs sometimes use a status light for each interface plug, or failure condition, to tell what failed. A cheap variation is to have two light bars with a printed matrix of errors that they select- the user can glue on the labels for the language that he speaks. For example, most small computer printers use lights labeled with stick-on labels that can be printed in any language. In some markets, these are delivered with several sets of labels, so customers can pick the most comfortable language.

In many organizations, one person approves the user interface. Often this is a customer, the major distributor or someone directly responsible for selling the system.

**8.6.2 PLATFORM**

There are many different [CPU architectures](http://en.wikipedia.org/wiki/CPU_architecture) used in embedded designs such as [ARM](http://en.wikipedia.org/wiki/ARM_architecture), [MIPS](http://en.wikipedia.org/wiki/MIPS), [Coldfire](http://en.wikipedia.org/wiki/Coldfire)/[68k](http://en.wikipedia.org/wiki/68k), [PowerPC](http://en.wikipedia.org/wiki/PowerPC), [X86](http://en.wikipedia.org/wiki/X86), [PIC](http://en.wikipedia.org/wiki/PIC_microcontroller), [8051](http://en.wikipedia.org/wiki/8051), [Atmel AVR](http://en.wikipedia.org/wiki/Atmel_AVR), [H8](http://en.wikipedia.org/wiki/H8), [SH](http://en.wikipedia.org/wiki/SuperH), [V850](http://en.wikipedia.org/wiki/V850), [FR-V](http://en.wikipedia.org/wiki/FR-V), [M32R](http://en.wikipedia.org/wiki/M32R) etc.

This in contrast to the desktop computer market, which [as of this writing (2003)](http://en.wikipedia.org/wiki/As_of_2003) is limited to just a few competing architectures, mainly the [Intel](http://en.wikipedia.org/wiki/Intel)/[AMD](http://en.wikipedia.org/wiki/AMD) [x86](http://en.wikipedia.org/wiki/X86), and the [Apple](http://en.wikipedia.org/wiki/Apple_Computer)/[Motorola](http://en.wikipedia.org/wiki/Motorola)/[IBM](http://en.wikipedia.org/wiki/IBM) [PowerPC](http://en.wikipedia.org/wiki/PowerPC), used in the [Apple Macintosh](http://en.wikipedia.org/wiki/Apple_Macintosh). With the growing acceptance of [Java](http://en.wikipedia.org/wiki/Java_programming_language) in this field, there is a tendency to even further eliminate the dependency on specific CPU/hardware (and OS) requirements.

Standard [PC/104](http://en.wikipedia.org/wiki/PC/104) is a typical base for small, low-volume embedded and ruggedized system design. These often use [DOS](http://en.wikipedia.org/wiki/DOS), [Linux](http://en.wikipedia.org/wiki/Linux) or an embedded real-time operating system such as [QNX](http://en.wikipedia.org/wiki/QNX) or [Inferno](http://en.wikipedia.org/wiki/Inferno).

A common configuration for very-high-volume embedded systems is the [system on a chip](http://en.wikipedia.org/wiki/System_on_a_chip), an [application-specific integrated circuit](http://en.wikipedia.org/wiki/Application-specific_integrated_circuit), for which the CPU was purchased as intellectual property to add to the IC's design. A related common scheme is to use a [field-programmable gate array](http://en.wikipedia.org/wiki/Field-programmable_gate_array), and program it with all the logic, including the CPU. Most modern [FPGAs](http://en.wikipedia.org/wiki/FPGA) are designed for this purpose.

**8.6.3 TOOLS**

Like typical computer programmers, embedded system designers use [compilers](http://en.wikipedia.org/wiki/Compiler), [assemblers](http://en.wikipedia.org/wiki/Assembler), and [debuggers](http://en.wikipedia.org/wiki/Debugger) to develop embedded system software. However, they also use a few tools that are unfamiliar to most programmers.

Software tools can come from several sources:

* Software companies that specialize in the embedded market.
* Ported from the [GNU](http://en.wikipedia.org/wiki/GNU) software development tools.

Sometimes, development tools for a personal computer can be used if the embedded processor is a close relative to a common PC processor. Embedded system designers also use a few software tools rarely used by typical computer programmers.

One common tool is an "in-circuit emulator" (ICE) or, in more modern designs, an embedded debugger. This debugging tool is the fundamental trick used to develop embedded code. It replaces or plugs into the microprocessor, and provides facilities to quickly load and debug experimental code in the system. A small pod usually provides the special electronics to plug into the system. Often a personal computer with special software attaches to the pod to provide the debugging interface.

Another common tool is a utility program (often home-grown) to add a checksum or [CRC](http://en.wikipedia.org/wiki/Cyclic_redundancy_check) to a program, so it can check its program data before executing it.

An embedded programmer that develops software for [digital signal processing](http://en.wikipedia.org/wiki/Digital_signal_processing) often has a math workbench such as [MathCad](http://en.wikipedia.org/wiki/MathCad) or [Mathematics](http://en.wikipedia.org/wiki/Mathematica) to simulate the mathematics.

Less common are utility programs to turn data files into code, so one can include any kind of data in a program. A few projects use [Synchronous programming languages](http://en.wikipedia.org/wiki/Synchronous_programming_language) for extra reliability or [digital signal processing](http://en.wikipedia.org/wiki/Digital_signal_processing).

**8.6.4 DEBUGGING**

[Debugging](http://en.wikipedia.org/wiki/Debugging) is usually performed with an [in-circuit emulator](http://en.wikipedia.org/wiki/In-circuit_emulator), or some type of debugger that can [interrupt](http://en.wikipedia.org/wiki/Interrupt) the microcontroller's internal microcode. The microcode interrupt lets the debugger operate in hardware in which only the CPU works. The CPU-based debugger can be used to test and debug the electronics of the computer from the viewpoint of the CPU. This feature was pioneered on the [PDP-11](http://en.wikipedia.org/wiki/PDP-11).

As the complexity of embedded systems grows, higher level tools and operating systems are migrating into machinery where it makes sense. For example, [cell phones](http://en.wikipedia.org/wiki/Cellphone), [personal digital assistants](http://en.wikipedia.org/wiki/Personal_digital_assistant) and other consumer computers often need significant software that is purchased or provided by a person other than the manufacturer of the electronics. In these systems, an open programming environment such as [Linux](http://en.wikipedia.org/wiki/Linux), [OSGi](http://en.wikipedia.org/wiki/OSGi) or [Embedded Java](http://en.wikipedia.org/wiki/Embedded_Java) is required so that the third-party software provider can sell to a large market.

**8.6.5 OPERATING SYSTEM**

Embedded systems often have no [operating system](http://en.wikipedia.org/wiki/Operating_system), or a specialized [embedded operating system](http://en.wikipedia.org/wiki/Embedded_operating_system) (often a [real-time operating system](http://en.wikipedia.org/wiki/Real-time_operating_system)), or the programmer is assigned to port one of these to the new system.

**BUILT- IN SELF- TEST**

Most embedded systems have some degree or amount of built-in self-test.

There are several basic types.

1. Testing the computer.

2. Test of peripherals.

3. Tests of power.

4. Communication tests.

5. Cabling tests.

6. Rigging tests.

7. Consumables test.

8. Operational test.

9. Safety test.

**START UP**

All embedded systems have start-up code. Usually it disables interrupts, sets up the electronics, tests the computer (RAM, CPU and software), and then starts the application code. Many embedded systems recover from short-term power failures by restarting (without recent self-tests). Restart times under a tenth of a second are common.

Many designers have found a few [LEDs](http://en.wikipedia.org/wiki/Light-emitting_diode) useful to indicate errors (they help [troubleshooting](http://en.wikipedia.org/wiki/Troubleshooting)). A common scheme is to have the electronics turn on all of the LED(s) at reset (thereby proving that power is applied and the LEDs themselves work), whereupon the software changes the LED pattern as the [Power-On Self Test](http://en.wikipedia.org/wiki/Power-On_Self_Test) executes. After that, the software may blink the LED(s) or set up light patterns during normal operation to indicate program execution progress or errors. This serves to reassure most technicians/engineers and some users. An interesting exception is that on electric power meters and other items on the street, blinking lights are known to attract attention and vandalism.

*APENDIX-A*

**INSTALLING CODING INTO PIC MICROCONTROLLER**

1. Write the program in MPLAB IDE.
2. Save the file as \*.c. and compile it.
3. After successful compilation of the coding close the MPLAB IDE.
4. Fix the Controller IC into PIC Flash kit.
5. Then click on Micro controller Micro Systems PIC Flash Software Icon on the desktop.
6. It displays on dialog box. Then select open and select the program which we already saved as \*.c.
7. Then it asks the Confirmation that The IC is empty, select ok.
8. Then it asks Fuses Settings, select YES
9. Then it displays Fuses Settings Dialog Box.
10. In that put WDT -- > Disabled, WRT-- > Enabled, Oscillator-- > XT then click on OK.
11. Then it displays the Program successfully installed into PIC.
12. Then Remove the IC from the PIC Flash and it is ready for used into the project or circuit operation.