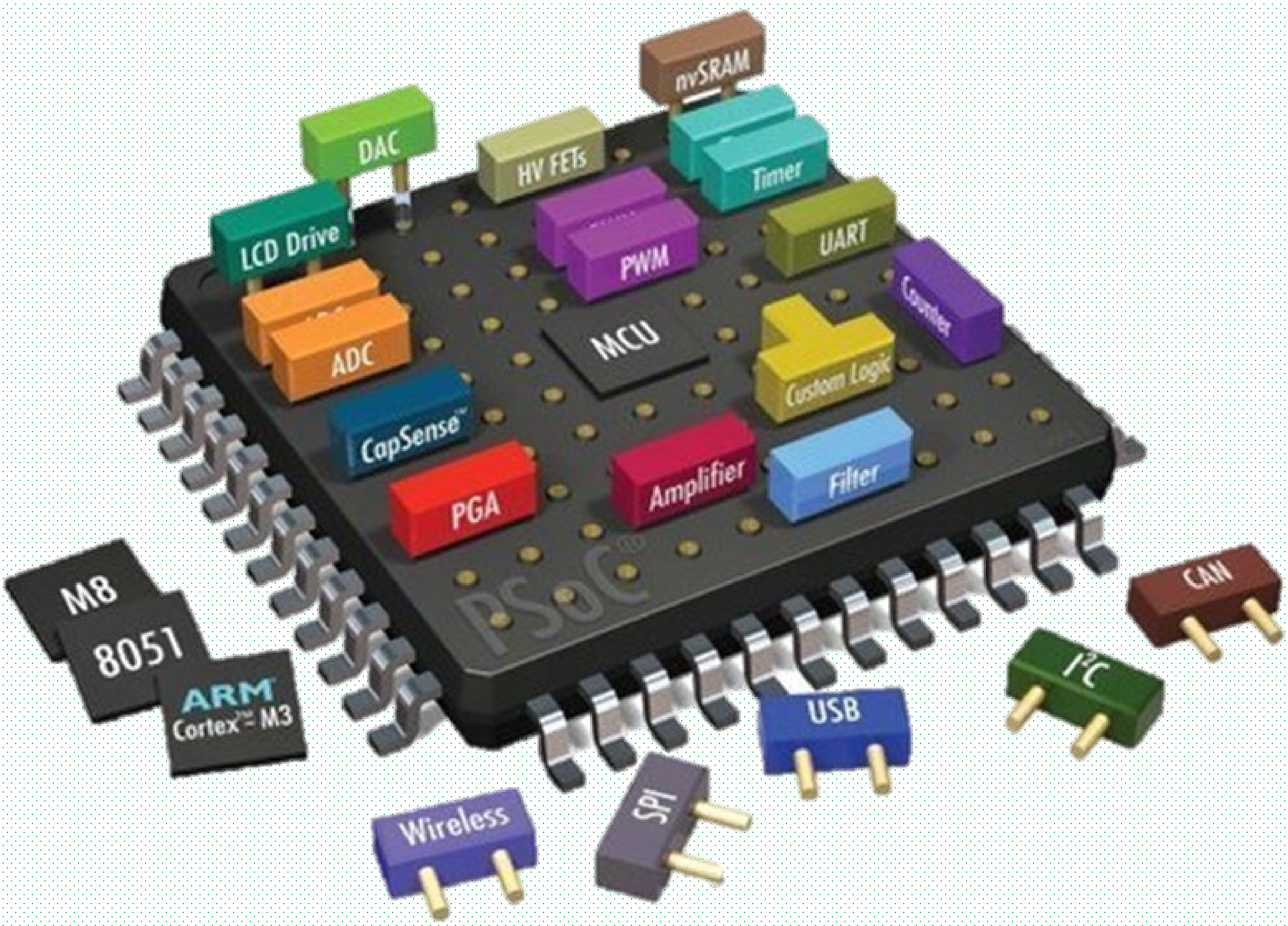
POLITEKNIK MALAYSIA

KEMENTERIAN PENGAJIAN TINGGI

KOLEJ KOMUNITI MALAYSIA

EMBEDDED SYSTEM APPLICATIONS DEC40053



DEC40053
EMBEDDED SYSTEM APPLICATIONS

Perpustakaan Negara Malaysia Cataloguing-in-Publication Data (after isbn is received)

Cataloguing Information (to be informed)

PUBLISHED BY:

Politeknik Port Dickson

KM14, Jalan Pantai, 71050 Si Rusa

Port Dickson, Negeri Sembilan

AUGUST 2021

PREFACE

The highest gratitude to Allah SWT because with His permission, this Embedded System Applications course e-book was successfully published.

This e-book is published as a guide or reference to all students and lecturers who are studying this course. This e-book has arranged its contents according to the latest syllabus of the course. There are five (5) main topics in this e-book and at the end of each topic are also included tutorial questions to test the level of understanding of students. As an author, we hope this e-book can help you to better understand the content of learning in the Embedded System Applications course.

We would also like to express our deepest appreciation to Department of Polytechnic and Community College Education particularly the E-learning and Instructional Division (BIPD) for funding our e-book project.

We hereby declare that this module is our original work. To the best of our knowledge it contains no materials previously written or published by another person. However, if there is any, due acknowledgment and credit are mentioned accordingly in the e-book.

Hopefully this e-book can be used to the best advantage by all students and lecturers.

Thank you

TABLE OF CONTENTS

CHAPTER 1 : INTRODUCTION TO EMBEDDED SYSTEM

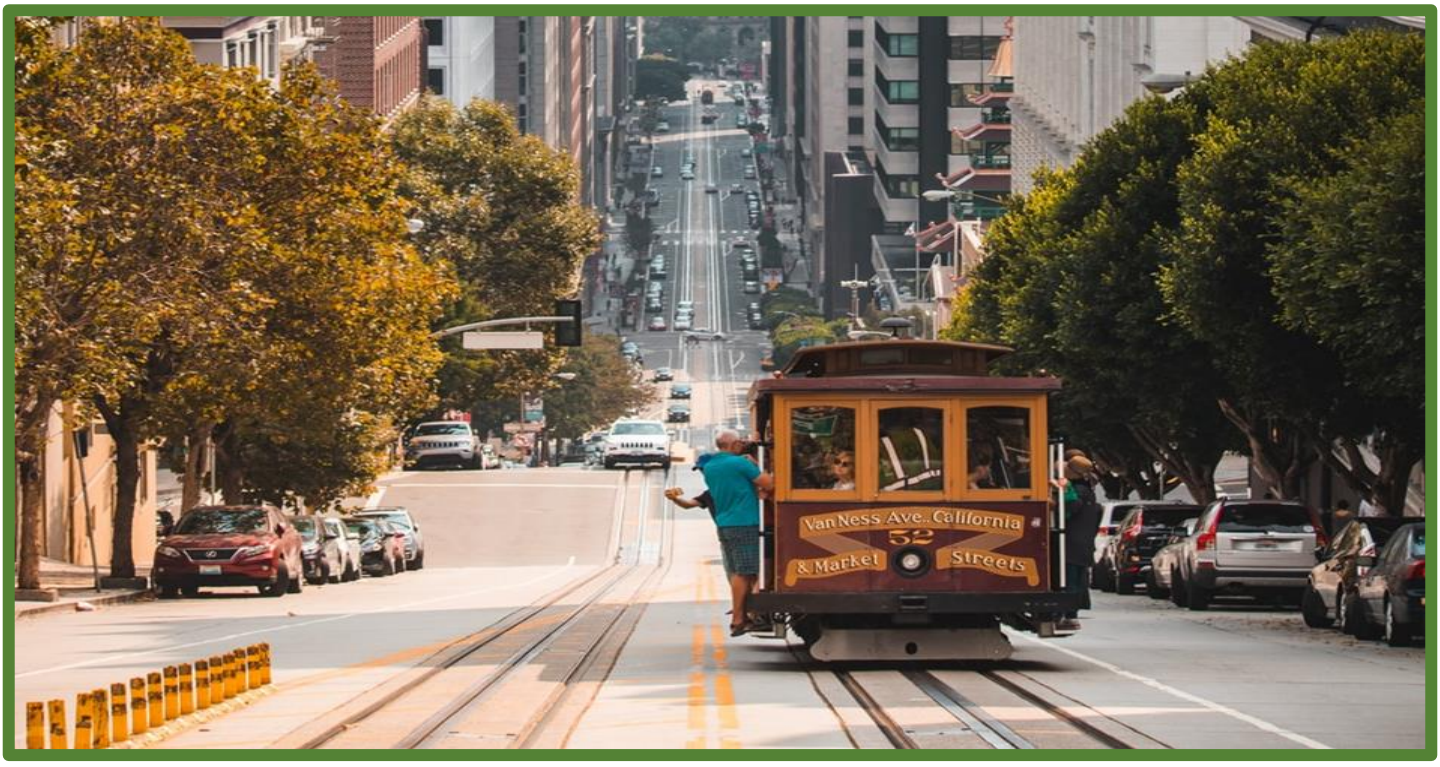
CHAPTER 4 : INTERRUPT PROGRAMMING IN C

CHAPTER 5 : HARDWARE INTERFACING

CHAPTER 1

INTRODUCTION

TO EMBEDDED SYSTEM



WHAT IS EMBEDDED SYSTEM?

- A system that performs a dedicated function which is a combination between hardware and software.
- It consists of input devices such as switches, sensors, keypads and output devices such as a buzzer, motors and LCD, processor and a program embedded on chip memory and a Real-Time system.
- Figure 1 below is a Block Diagram of Embedded System Applications .

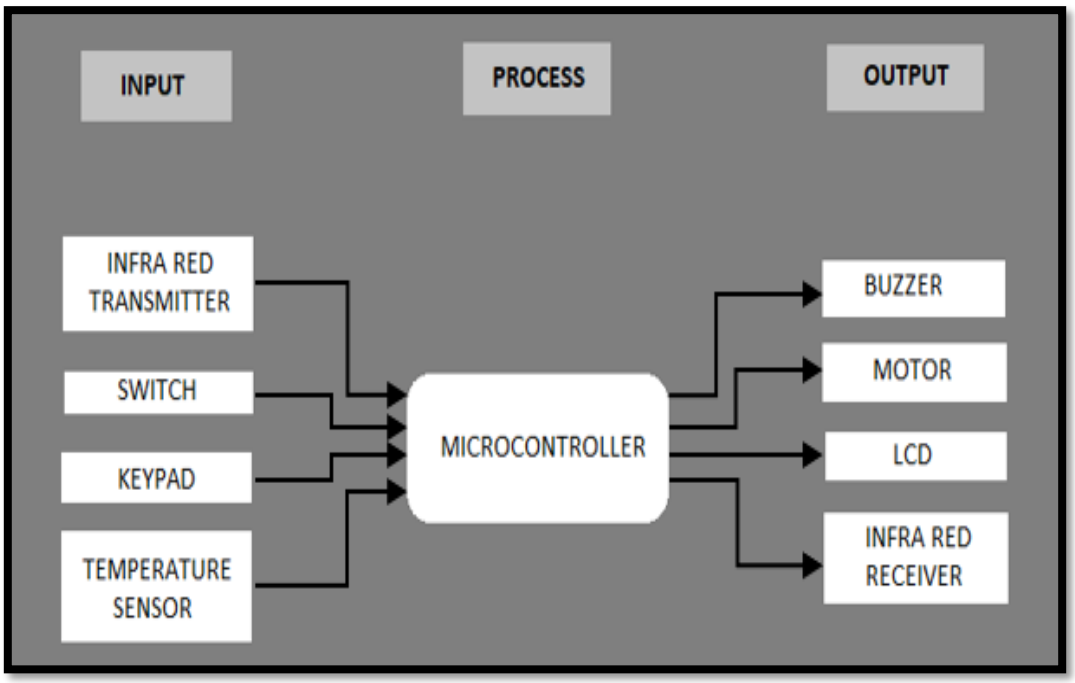


Figure 1.1 : Block Diagram of Embedded System Applications

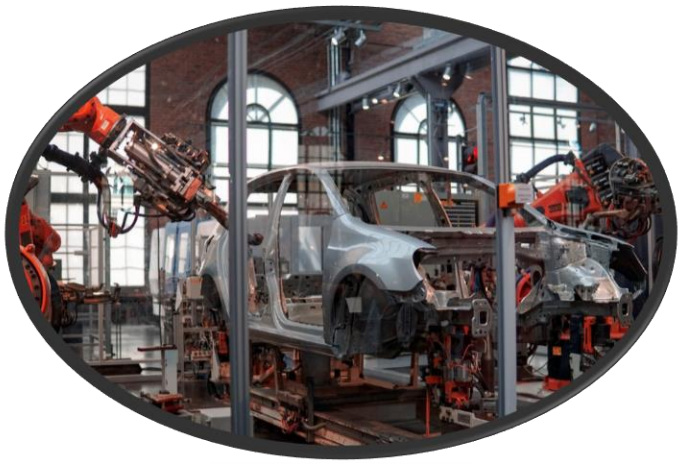
WHAT IS A MICROCONTROLLER?

- **Microcontroller** is a small computer on a single Integrated Circuit (IC) that contain a processor core, memory and programmable input/output peripherals.
- Processor Core is called the brain of the device. It processes the various instructions like input/output operations, arithmetic logic instructions and data transfer instruction that direct the function of the microcontroller. A Microcontrollers memory is used to store the data process by the processor while the input/output devices are used to communicate with the outside world.
- Microcontrollers are design for embedded system applications. It gives life to the system. That is why the Microcontroller is called the heart of the Embedded System.
- Microcontrollers are purchased blank and they will be programmed by a programmer with a specific program.

APPLICATIONS OF EMBEDDED SYSTEMS IN REAL-LIFE

AUTOMOTIVE INDUSTRY

Engine Control, Ignition Systems, Anti-Lock Breaking Systems (ABS)



HOME AUTOMATION AND SECURITY SYSTEMS

Water Sprinkler, Air conditioners, Intruder Detection Alarm

MEDICAL/HEALTHCARE

ECG (Electrocardiogram) Machines, Patient Monitoring Machine

HOUSEHOLD APPLIANCES

Washing Machine, Microwave Oven, Fridge, Television

COMPUTER NETWORKING SYSTEMS

Firewall, Network Router, Hubs, Switches

MEASUREMENT AND INSTRUMENTATIONS

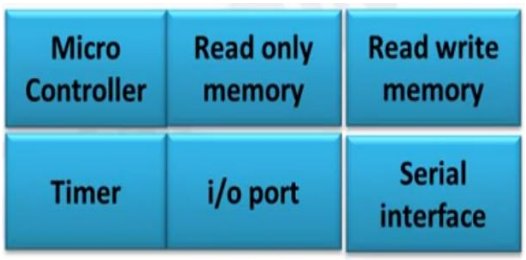
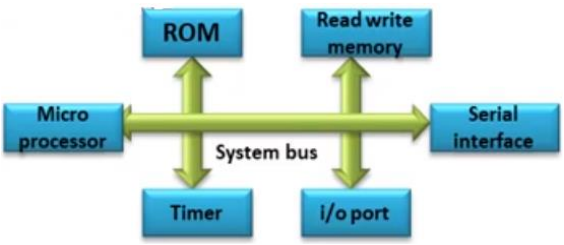
Logic Analyzer, Spectrum Analyzer

TELECOMMUNICATIONS

Satellite, Smartphone, Telephone Switches

DIFFERENCES BETWEEN MICROCONTROLLER AND MICROPROCESSOR

Table 1.1 : Microcontroller vs Microprocessor

MICROCONTROLLER	MICROPROCESSOR
	
Inbuilt CPU, Memory and Input/ Output Devices in one chip	Memory and Input/ Output Devices connected externally
Low power consumption	High power consumption
Cheaper	Expensive
Multifunction pins on the IC	Less multifunction pin on IC
Separate memory to store data and program	Data and program are stored in the same memory

ADVANTAGES AND DISADVANTAGES OF MICROCONTROLLER APPLICATION

ADVANTAGES

- Increase Efficiency
- Portable and more reliable
- Increase safety
- Allow greater flexibility- can be reprogrammed
- Reduced power consumption

DISADVANTAGES

- Limited processing power
- Having relatively complex architecture
- A longer time is needed due to the complexity of the circuit

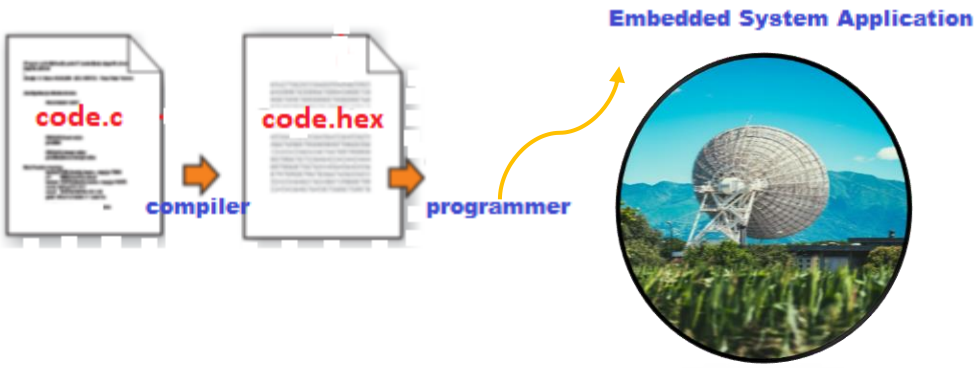
MICROCONTROLLERS MANUFACTURERS

There are many microcontrollers available in our market from different manufacturers. The microcontrollers are characterized based on their instruction set, memory structure and bus width. Below is the example of 8-bit Microcontrollers Manufacturers.

C PROGRAMMING FOR PIC MICROCONTROLLER

- The PIC-Peripheral Interface Controller is a family of the microcontroller manufactured by Microchip.
- PIC Series offer a wide range of low-cost devices, ranging from 8 pin to a feature 40 pin device.
- The PIC microcontrollers can be programmed by the C or Assembly Language using dedicated software. However the most popular language used for Embedded System Applications is a C Language. Some of the advantages of using C Language are:
    - Easier to write and less time consuming
    - Reduce costs through traditional programming techniques
    - Easier to update and do a modification
    - Easily to import code available in function libraries i.e. delays.
    - It is portable to other microcontrollers with less modification
    - Suitable with Microchip C18 Compiler

- Compiler is a specialized program that compiles our sourcecode and generates a few files related to the program.
- The .hex file produced by the compiler will be download into microcontroller ROM.
- To integrate with MPLAB IDE, Microchip's C18 Compiler can be used .



STRUCTURE OF C PROGRAM FOR PIC

- Documentation, Link Section, Definition Section, Global Declaration, Main Function, and Subprograms are the six sections that make up the C programming structure.
- **Documentation** : A section in which the programmer can give comments to make the program more interactive like details of the author as shown in the figure below.

```
/*

Date : 01/01/2021
Author : Siti Zaria Ali
Decsription: A program to Toggle four LED in 10s.

*/
```

- **Link Section** : This section includes an instruction to the compiler that links the header files or functions from the library system.

  Example :

```
#include <p18f4550.h>
```

- ***Definition Section*** : *The definition section defines all symbolic constants such by using the #define directive.*

  Example :

  ```
  #define max 100
  ```

- ***Global Declaration*** *: Some variables that are used in more than one function, such variables are called global variables.*

  Example :

  ```
  void add();
  int a=10;
  ```

- ***Main Function*** : *Every C-program should have one main() function*.

  Example:

  ```
  void main (void)
  ```

- ***Subprograms.*** *If the program is a multi-function program, then the subprogram section contains all user-defined functions that are called in the main() function.*

  Example:

  ```
  float area (float r)
  {
   Return PI*r*r;
  }
  ```

PIC18 C DATA TYPE

- A suitable selection of C Data Type will help programmers to create a smaller .hex files.
- Table 1 below is a type of PIC18 Data Type that is commonly used for PIC Microcontroller.

Table 1.2 : PIC18 Data Type

Data Type	Size (No of bits)	Data Range
Char	8-bit	-128 to +127
Unsigned char	8-bit	0 to 255
int	16-bit	-32 768 to +32767
Unsighed int	16-bit	0 to 65 535
Short	16-bit	-32 768 to +32 767
Unsigned short	16-bit	0 to 65 535
long	32-bit	-2 147 483 648 to +2 147 483 648
Unsigned long	32-bit	0 to 4 294 967 295

- Below is an example of a simple program that is used to send values of 00-FF to PortD

```
#include <p18f4550.h>

void main (void)                    //main progress
{
    unsigned char z;                //declare z as unsigned character
    TRISD=0;                        //set ALL Port D as Output

    while (1)                       //continuously repeated
    {

        for(z=0;z<=255;z++)         //loop z from 0 to 255
        PORTD=z;

    }

}
```

LOGIC, ARITHMETIC AND BIT MANIPULATION OPERATION

- **Logic Operators** will return to logic 1 if the expression evaluates to non zero and false (logic 0) if the expression evaluates to zero. (Table 2)
- Logic Operators are commonly used upon expressions in the program.
- **Arithmetic Operators** are used in Arithmetic operations and always return positive results. (Table 3).
- To execute a binary operation, two numbers are required because it performed upon two operands.

Table 1.3 : Logic Operators

	Operand 1	Operand 2	Result
Logic AND (&)	0	0	0
	0	1	0
	1	0	0
	1	1	1

	Operand 1	Operand 2	Result
OR (I)	0	0	0
	0	1	1
	1	0	1
	1	1	1

	Operand 1	Operand 2	Result
EX- OR (^)	0	0	0
	0	1	1
	1	0	1
	1	1	0

	Operand 1	Result
NOT (~)	0	1
	1	0

Table 1.4 : Arithmetic Operators

	OPERATION	OPERATOR
ARITHMETIC OPERATION	Addition	+
	Subtraction	-
	Multiplication	*
	Division	/
	Reminder	%

- **Bit Manipulation operators** are used to modify the bits of a variable.
- It is performed upon single bits within operands.

Table 1.5 : Bit Manipulation Operators

OPERAND	MEANING	EXAMPLE
<<	Shift Left	a = b << 3
>>	Shift Right	a = b >> 3
&	Bitwise AND	c = a & b
I	Bitwise OR	c = a I b
^	Bitwise EX OR	c = a ^ b
~	Bitwise Complement	a = ~ b

DIGITAL INPUT AND OUTPUT (I/O) PROGRAMMING IN C

- It is easy to program a PIC microcontroller. The microcontroller program memory satisfies most applications. Along with 33 programmable input/output pins, it can interface with many peripherals easily.
- With Watchdog timer to reset under error it can be used on systems with no human interference. By having a USB interface feature, communication with a controller from any PC without any hassle can be done. Many features added together further promote the use of PIC18F4550 microcontroller.

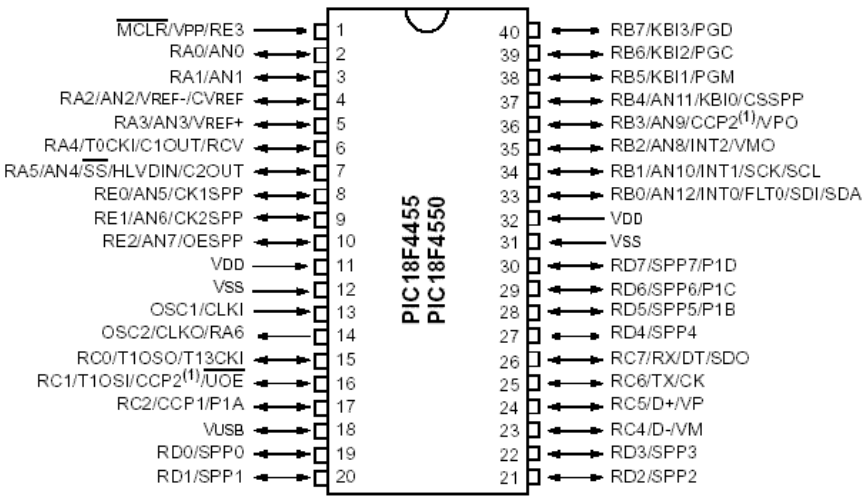


Figure 1.2 : PIC18F4550 Pin Diagram

- A number of input/output pins on the PIC18F4550 microcontroller are used to connect to external devices. It has 40 pins in total. Thirty-three of the forty pins can be used as input-output pins. These pins are divided into five PORTS, which are denoted by the letters A, B, C, D, and E as follows:

  a) Port A has seven pins labelled RA0-RA6;

  b) Port B has eight pins labelled RB0-RB7;

  c) Port C has seven pins labelled RC0-RC2, RC4-RC7;

  d) Port D has eight pins labelled RD0-RD7;

  e) Port E has three pins labelled RE0-RE2.

TRISx , PORTx and LATx REGISTER

- A data direction register is **TRIS**. The data direction (whether Read or Write) to the microcontroller is determined by setting the TRIS bit for the relevant port.

```
TRISB = 1;          // make Port B as an Input Port
TRISB = 0;          // make Port B as an Output Port
```

- Specific bits of the port can also be set as input or output. This type of writing is called **bit addressability**.

```
TRISBbits. RB3 = 1;          // make Port B pin RB3  as an Input Port
```

- The **PORT** register reads the device's pin levels (low/high) and assigns logic values (0/1) to the ports. The PORT register's job is to accept data from an external source, such as a temperature sensor, or to send data to external parts, such as a buzzer.

```
TRISB= 1;          // make Port B as an Input Port
PORTB=0xFF;        // assigning High logic to the to ALL pins of Port B
```

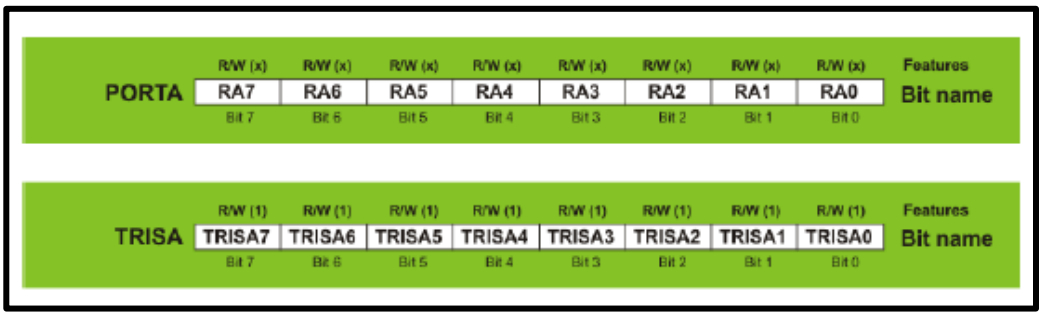
- Now, for PORT B's output mode, we'll use the TRIS Register to set the data direction. The output value is sent to the Port. The data value is written to the port latch by a Port Register.

```
TRISB = 0;          // make Port B as an Output Port
PORTB = 0x05;       // Assigning High logic to pin RB0 and RB2
```

- Memory is also mapped to the Data Latch register. An I/O pin is linked to the LAT register. It eliminates the issues that read-modify-write instructions can cause.
- Instead of the values on the I/O pins, a read of the LAT register returns the values held in the port output latches. A read-modify-write operation on the LAT register associated with an I/O port prevents the input pin values from being written into the port latches.
- The impact of writing to the LAT register is the same as writing to the PORT register. The data value is written to the port latch when you write to the PORT register. A write to the LAT register, on the other hand, writes the data value to the port latch.

Details on Port A

- Port A is a bidirectional, 8-bit wide port. The Port A pins are controlled by bits in the TRISA and ANSEL registers. All of the pins on Port A function as digital inputs and outputs. Five of them can also be used as analogue inputs (denoted by the letter AN):

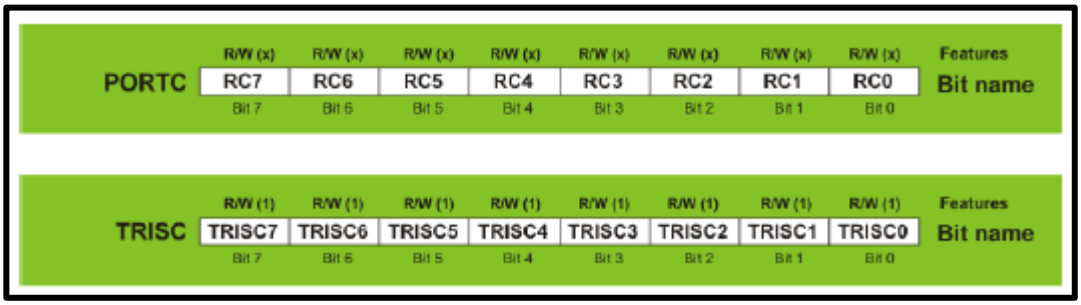


Details on Port B

- Port B is a bidirectional, 8-bit wide port. The purpose of the TRISB register's pins is determined by its bits.

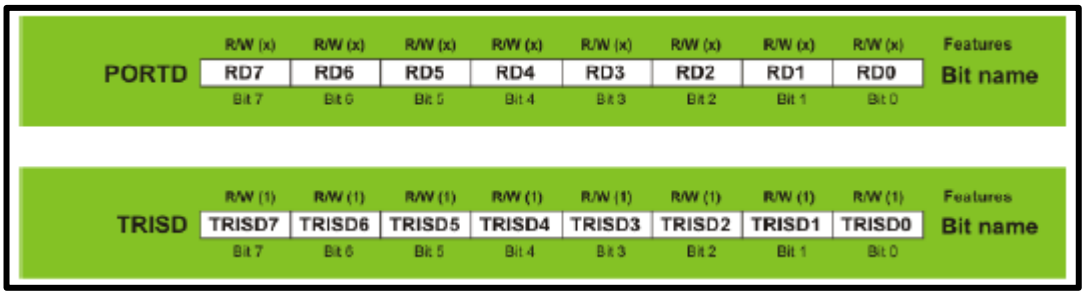
Details on Port C

- Port C is a bidirectional, 8-bit wide port. The purpose of the TRISC register's pins is determined by its bits. A logic one (1) in the TRISC register, like other ports, configures the relevant port C pin as an input.



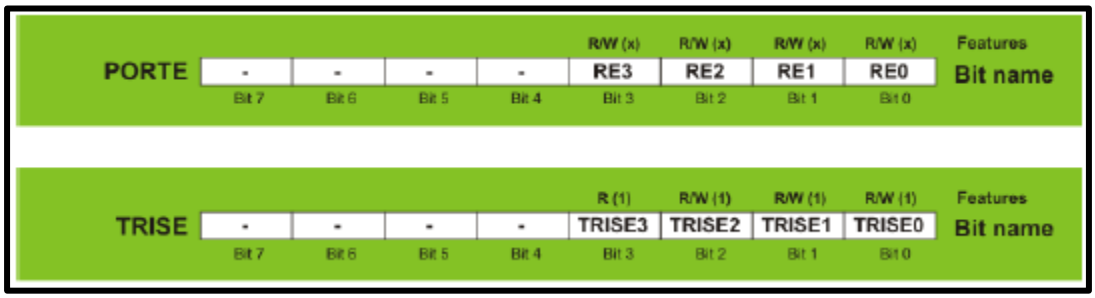
Details on Port D

- Port D is a bidirectional, 8-bit wide port. The purpose of the TRISD register's pins is determined by its bits. The appropriate Port D pin is configured as an input by a logic one (1) in the TRISD register.



Details on Port E

- Port E is a bidirectional, 4-bit wide port. The function of the TRISE register's pins is determined by its bits.A logic one (1) in the TRISE register, like other ports, configures the corresponding portE pin as an input. The RE3 pin, on the other hand, is always configured as an input.



Dual Alternate Function in PIC18F4550

- Some I/O port pins are multiplexed with an alternate function from the device's peripheral features. A pin may not be utilised as a general purpose I/O pin if it is used for any other purpose.

SIMPLE DIGITAL I/O PROGRAM

Example 1 : Write a C18 programme that toggles bit RB2 constantly while leaving the rest of Port B's bits alone.

Answer :

```
#include <p18f4550.h>
#define bit PORTBbits.RB2

void main (void)
{

    TRISBbits.RB2=0;

while (1)

{
   bit=1;
   bit=0;
}

}
```

Example 2 : Write a C18 sourcecode to monitor bit RB7. If it is HIGH, send 00H to PORTD otherwise send FFH to PORTC

Answer

```
#include <p18f4550.h>
#define bit PORTBbits.RB7

void main (void)
{

    TRISBbits.RB7=1;
    TRISC =0;
    TRISD =0;

while (1)

  {
   if (bit==1)
        PORTD=0X00;
   else
        PORTC=0XFF;
   }

}
```

PROGRAM FOR TIME DELAY

- In PIC18F4550, the delay is implemented with general counting loops or with timers. A 'for' loop is a repetition control structure that allows you to design a loop that needs to run a certain number of times quickly.

```
void MSDelay (unsigned int itime)
{

    unsigned int i;
    unsigned char j;
    for (i=0;i<itime;i++)
        for (j=0;j<165;j++);

}
```

- For example; 'for (i=0;i<itime;i++)', the 'i=0' initializes a loop count variable once at start of loop. The expression is the test condition, where the loop will continue while it is true (i <= itime). While 'i++' is executed at the end of each iteration, usually to modify the loop count variable.

Example 1 : Write a C18 sourcecode that toggles all bits of PortB and PortD continuously with a 250ms delay. Assume XTAL Frequency is 10MHz.

Answer :

```
#include <pic.h>
void MSDelay (unsigned int);
void main (void)
{

    TRISB=0;
    TRISD=0;

while (1)

  {

   PORTB=0xAA;
   PORTD=0x55;
   MSDelay(250);
   PORTB=0x55;
   PORTD=0xAA;
   MSDelay(250);
    }

}


void MSDelay (unsigned int itime)
{

    unsigned int i;
    unsigned char j;
    for (i=0;i<itime;i++)
        for (j=0;j<165;j++);

}
```

EXERCISE CHAPTER 1

1. List two (2) advantages and two (2) disadvantages of a microcontroller.

Answer:

Advantage	Disadvantage
The price is cheap.	Limited amount of timers, memory and Input Output ports.
Less power consumed.	Cannot add any timers, external memory and Input Output ports
Use less space	Can only do a specific and limited task
Build in ROM, RAM, I/O port and Timer etc.	Different manufacturer using different instruction set.

2. Write the function of TRISx, PORTx and LATx registers

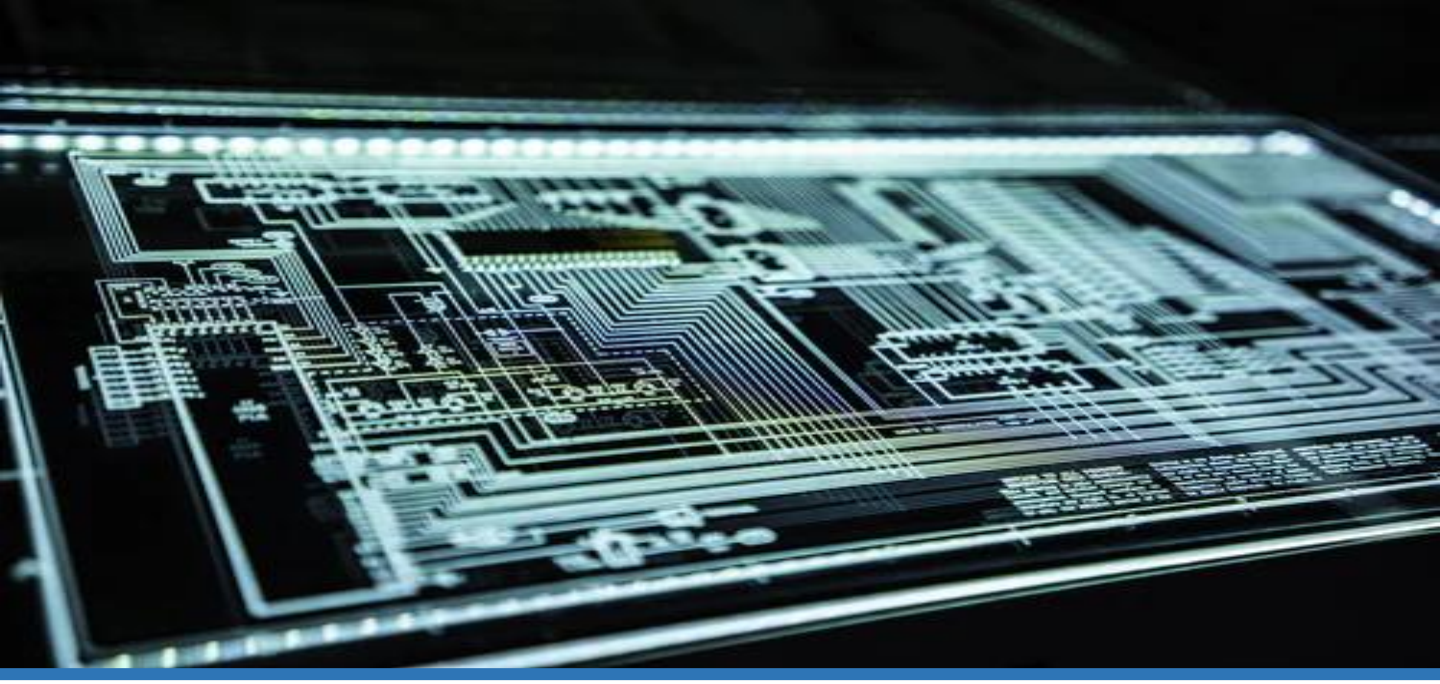
Answer:

TRISx	To make the port as input or output port
PORTx	Bidirectionally reads/writes the levels on the device's pins.
LATx	On the I/O port, this is useful for read-modify-write operations.

3. A programmer decides to make a home security using a touch card at pin RB3, a smoke sensor at pin RC5, temperature sensor at pin RC6 while magnetic solenoid, buzzer and LED at PORT D. Complete the pin's input and output declaration for the program using bit addressable format in C language

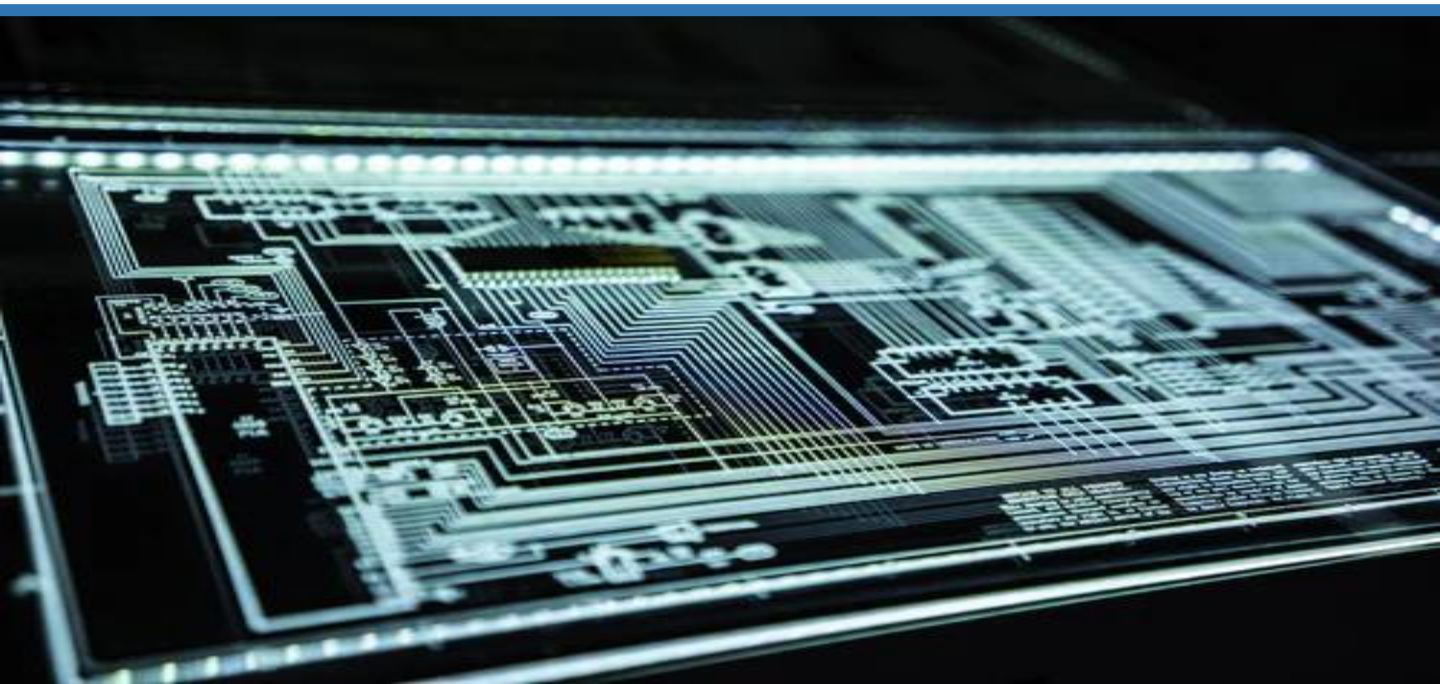
Answer:

```
TRISBbit.RB3=1
TRISCbit.RC5=1
TRISCbit.RC6=1
TRISDbit.RDx =0
TRISDbit.RDx =0
TRISDbit.RDx =0
```



CHAPTER 2

CIRCUIT SIMULATION



INTRODUCTION TO PROTEUS PROFESSIONAL

- **A simulation environment** can be defined as a computer programming dedicated for simulation for a specific experiment with a flexible and intelligent interface between a user and the system to be studied. For this course purpose, a few software is eligible in the market such as Proteus, MPlab and some other available softwares.

- The Proteus Design Set is a proprietary software tool suite that is primarily used to automate electronic design. Electronic design experts and technicians use the software to develop schematics and electronic prints for printed circuit board manufacture. Labcenter Electronics Ltd. in Yorkshire, England, created it.

- The Proteus Design Suite is a Windows tool that allows you to record schematics, simulate them, and design PCB layouts. The Proteus Design Suite's schematic capture feature is used for design simulation during the design phase of a PCB layout project. As a result, it is a critical component that comes standard with all product configurations. In Proteus, the microcontroller simulation is accomplished by adding a hex or debug file to the microcontroller portion on the schematic. It is then co-simulated with any analogue and digital electronics that are linked to it.

DESIGNING CIRCUIT

Following are the steps to be followed to start designing a circuit:

1. To begin using the software, go to Start, Programs, Proteus 8 Professional, and then the Proteus 8 application. After that, the main application will load and execute, and you will be sent to the Proteus home page, which looks like this:

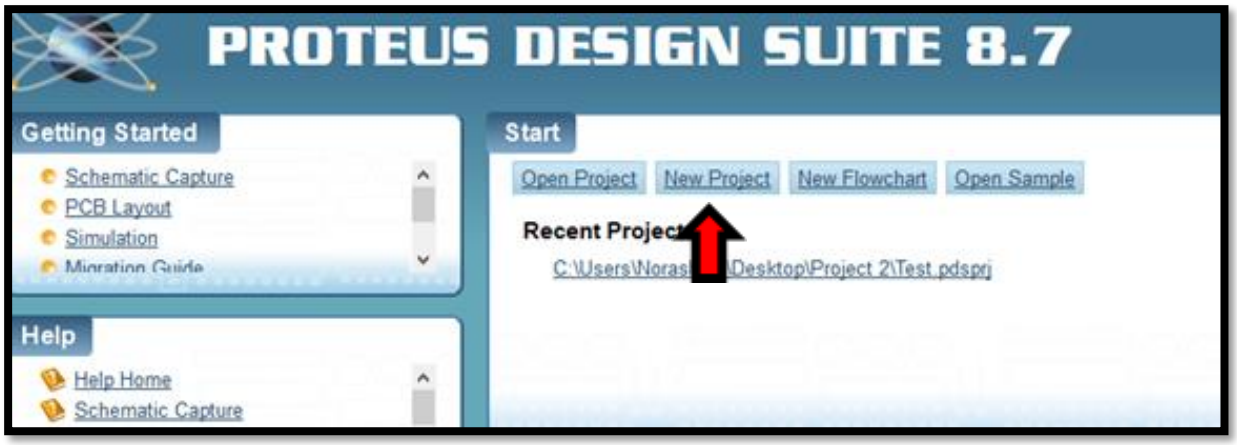


Figure 2.1: Proteus Software

2. We must first create a project before we can design a schematic.

3. In Proteus, begin by pressing the new project button towards the top of the home page.

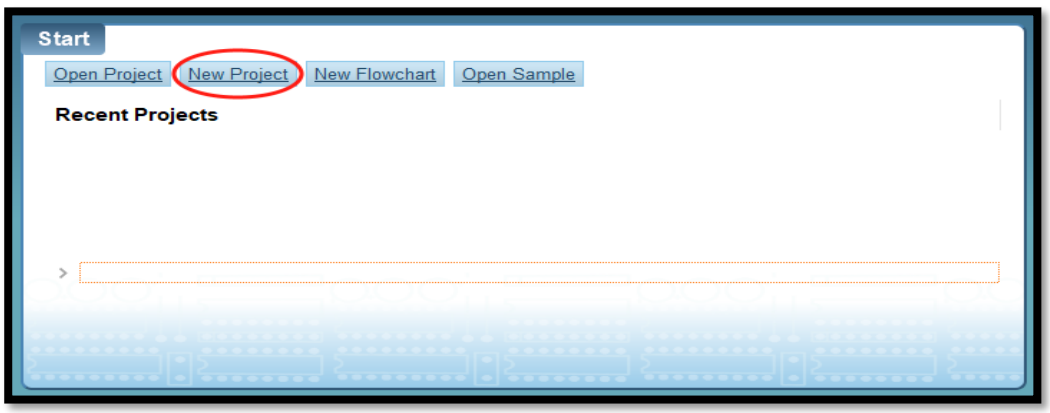


Figure 2.2 : Start a new project

4. On the wizard's first page, give the project a name and a path.

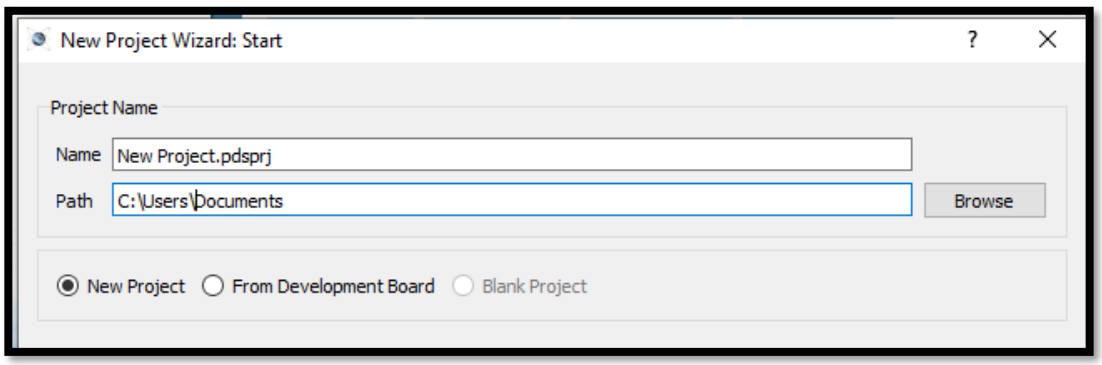


Figure 2.3 : Project Name

5. We'll need a schematic, so select the default template and tick the box at the top of the next step.

Figure 2.4 : New Project Wizard

6. We also require a layout, so click the box at the top of the layout page and select the default template once more.

7. Design a printed circuit board (PCB). We won't be designing any PCBs, so leave the firmware page blank and move on to the next option.

Figure 2.5 : New Project Wizard (Schematic Design)

8. Next, we will be choose Firmware to be used in the simulation. Create Firmware Project and choose desired tools needed before continuing to New Project Wizard: Summary as **Figure 2.6**.

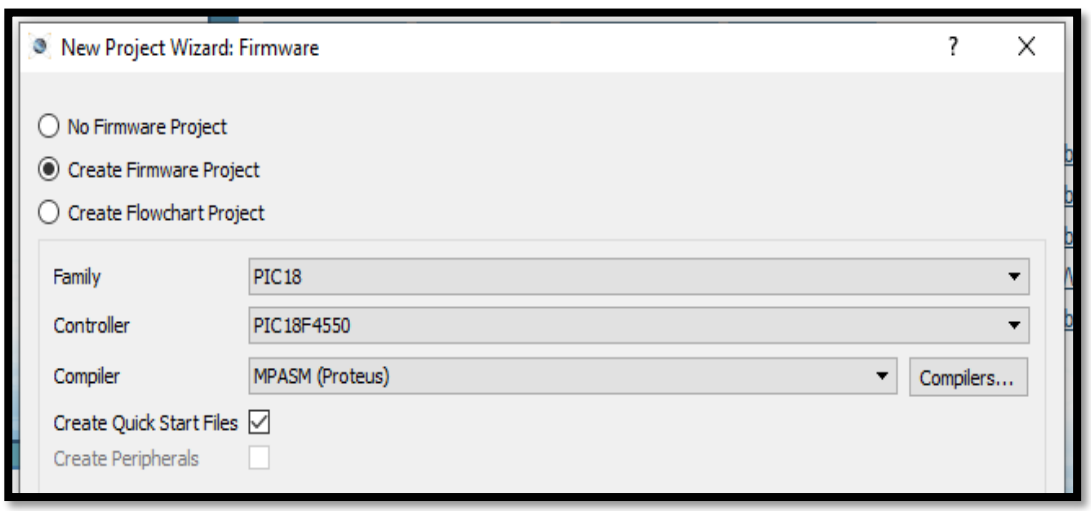


Figure 2.6 : Create Firmware Project

9. To create the project, click on the finish button .

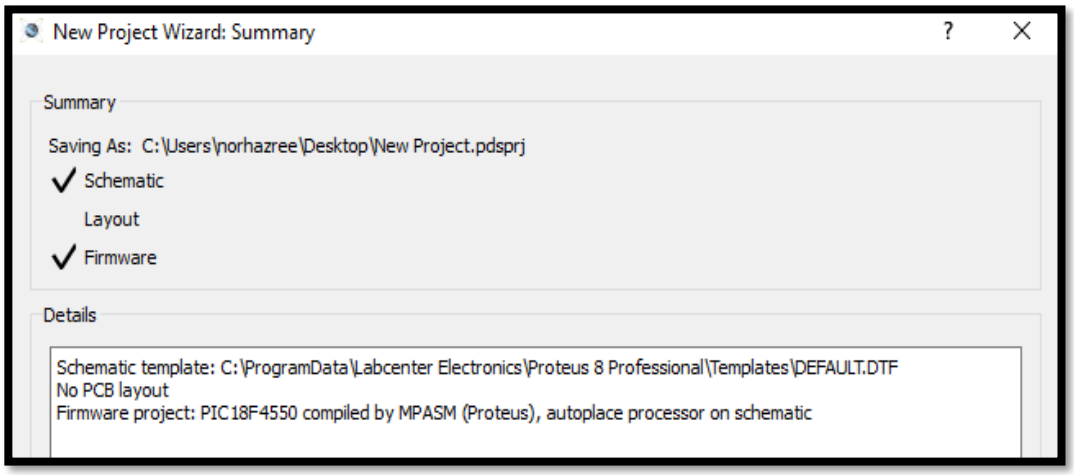


Figure 2.7 : New Project Summary

- Sheet size, colour scheme, corporate logo, header block, and other aesthetic presets can all be found in a schematic template. The Templates chapter of the reference handbook contains more information.

- Two tabs will appear in the project, one for schematic capture and the other for source code. To bring the Schematic module to the foreground, click the schematic tab.

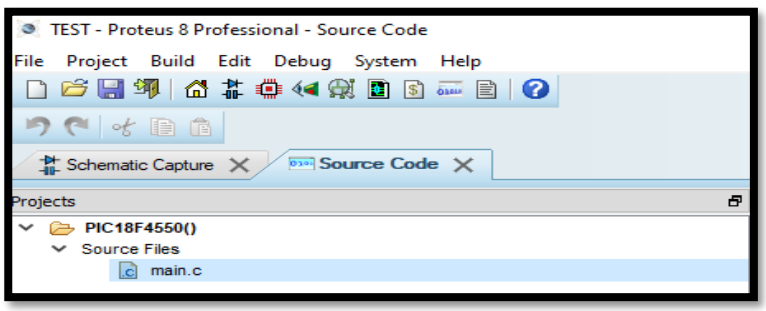


Figure 2.7 : Schematic Tab

SIMULATION ENVIRONMENT

- The Editing Window is the largest section of the screen, and it works as a window on the drawing, where you will position and wire-up components.
- The Overview Window is the smaller section at the top left of the screen. In normal use, the Overview Window offers an overview of the complete drawing, with the blue box representing the current sheet's edge and the green box representing the portion of the sheet currently showing in the Editing Window.
- The Overview Window is used to preview the selected object when a new object is selected using the Object Selector.

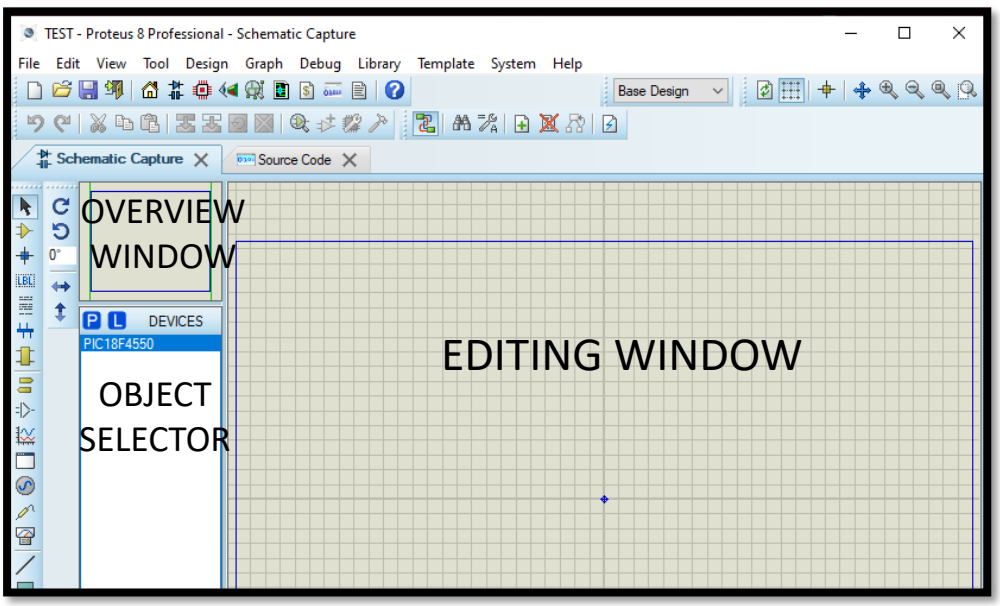


Figure 2.8 : Schematic Capture Window

- You can pick up and dock the toolbars on any of the four sides of the application if you don't like the default layout. You can also move the Object Selector and Overview.
- By dragging the end of the window pane all the way across to the other side of the application, you can move it to the right-hand side.Depending on whatever tab is active, the toolbars and menu items will change (at the front).
- When we talk about an icon or a menu command in this course, we're presuming the schematic tab is open.A context menu appears when you right-click the mouse in the Object Selector or the Overview Window, and it includes the option to 'auto hide' the left-hand pane.
- If you wish to maximise the editing area of the application, this is really handy. When this option is selected, the Object Selector and Overview Window will be minimised to a 'flyout bar' on the left (or right) side of the application. They appear when the mouse is moved over the bar or when the operating mode is changed by picking a different icon.

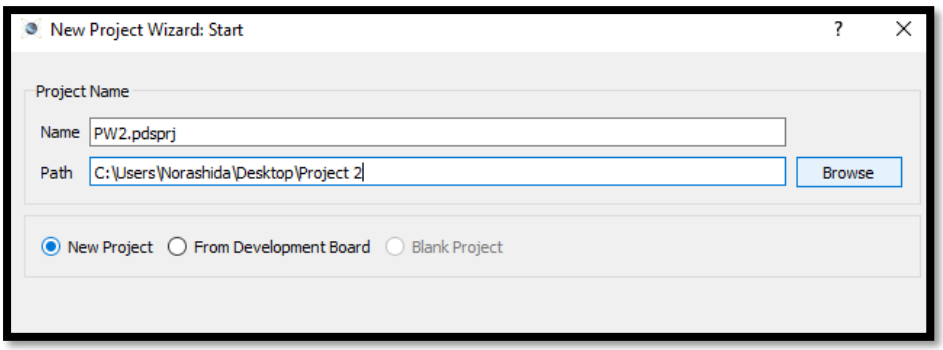
BASIC SCHEMATIC ENTRY

- We'll begin by learning the fundamentals of schematic design, such as selecting components from libraries, placing them on the schematic, and connecting them.
- The first step is to get the parts we'll need for our schematic from the libraries.

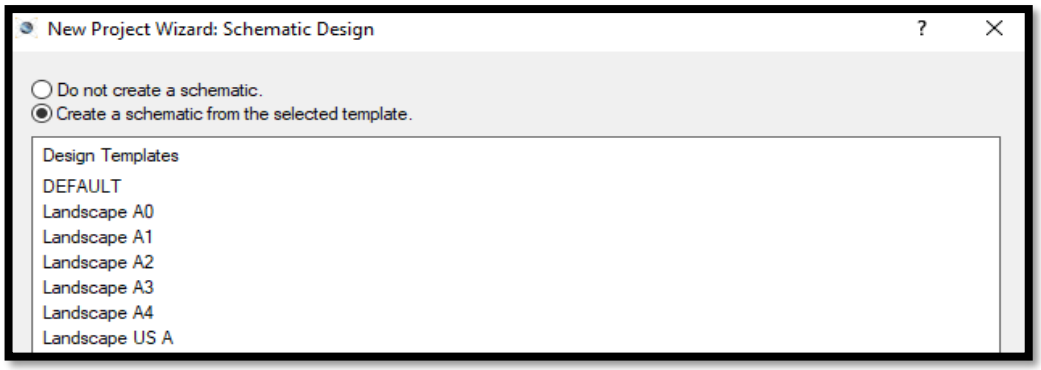
Let's Start Design

STEP1 : Click the *New Project* Button.

STEP 2 : Name your project and the path where you want to save your project.

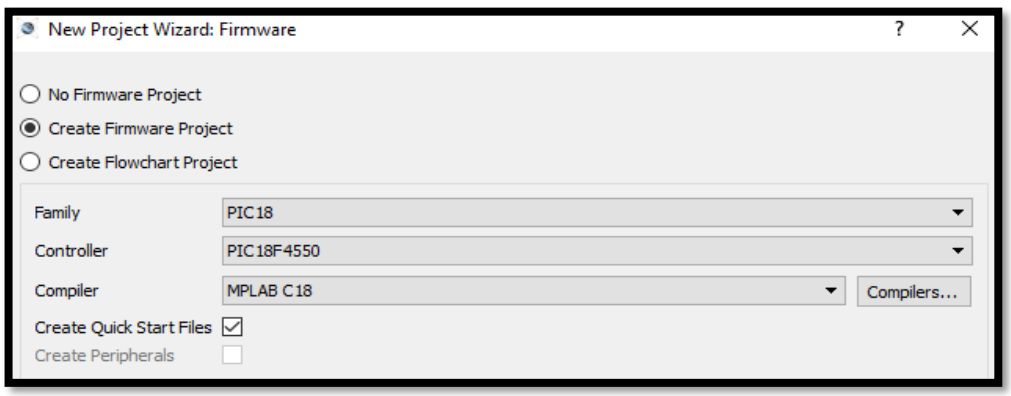


STEP 3 : Create a schematic Design

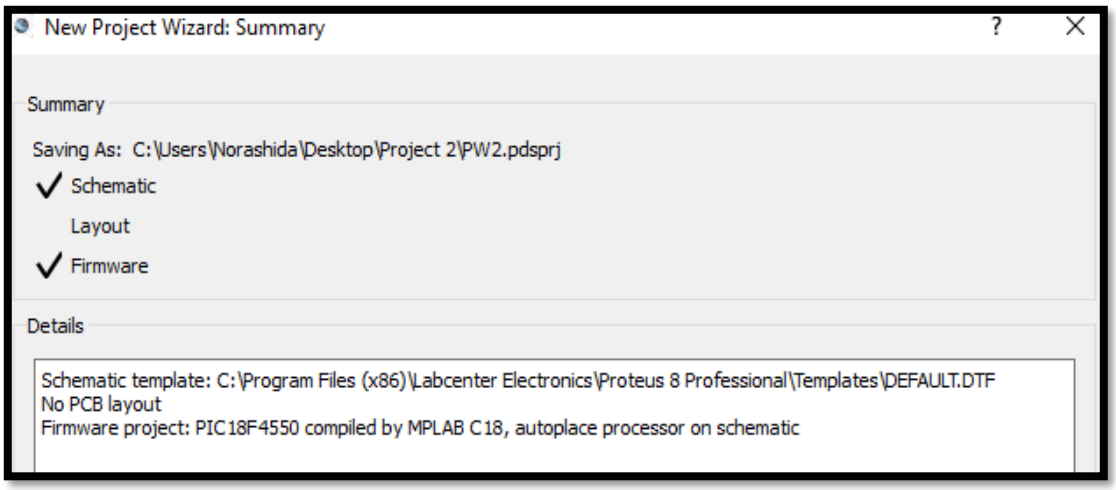


STEP 4 : Select either create a PCB Layout or not.

STEP 5 : Create Firmware Project and fill up the details needed as below. A firmware programme or set of instructions is a software programme or set of instructions that is installed on a hardware device. It contains the instructions for how the device interacts with the rest of the computer hardware.

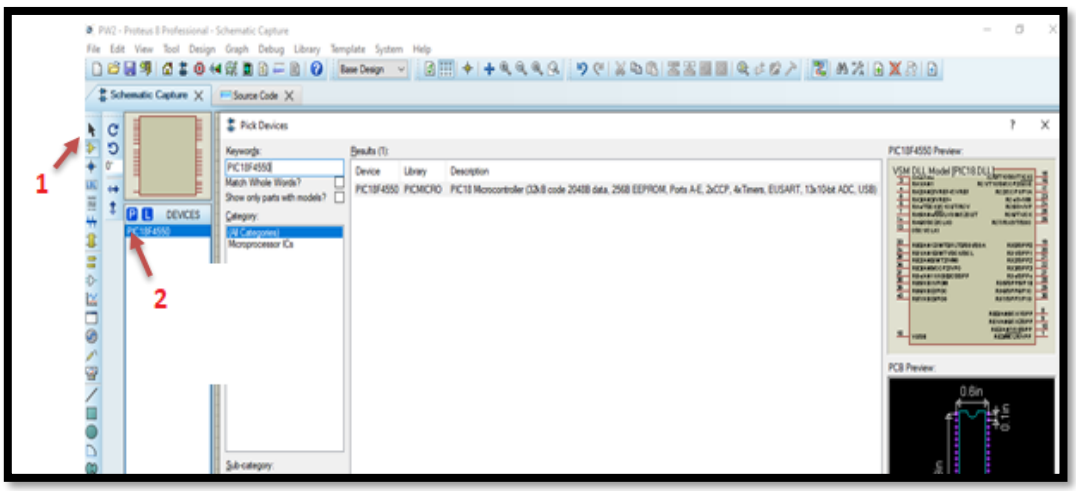


STEP 6 : Finish. Your schematic template is ready to use



STEP 7 : Select a component needed by clicking the *Component Mode* (1) and *P, Pick Devices* (2) icon button. Type your component detail at *keywords*.

.



STEP 8 : To place the components,click on a selected component list in device's box, and click on the drawing area.The image of the component will appear.Just place the component on any part of the drawing area.

STEP 9 : To move a placed component, place the cursor on the component until MOVE icon appears, click HOLD and move the component. To change the component's orientation , right-clicking the component and choose the orientation type you want.

STEP 10 : Place all the components needed in the drawing area.

STEP 11 : To make a wire connection, place the cursor on the components pin until a red square appears and drag it to the pin of another components. To remove the wrong connection, right-clicking on the wire connection and delete the wire.

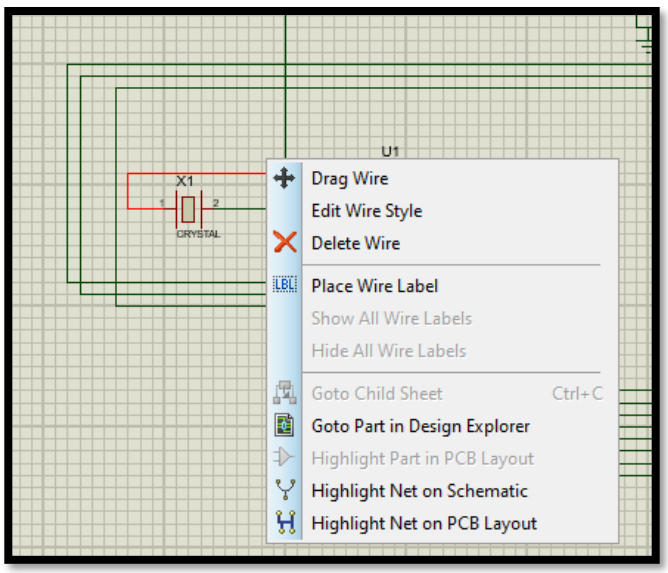


Figure 2.8 : Placing and deleting the wire

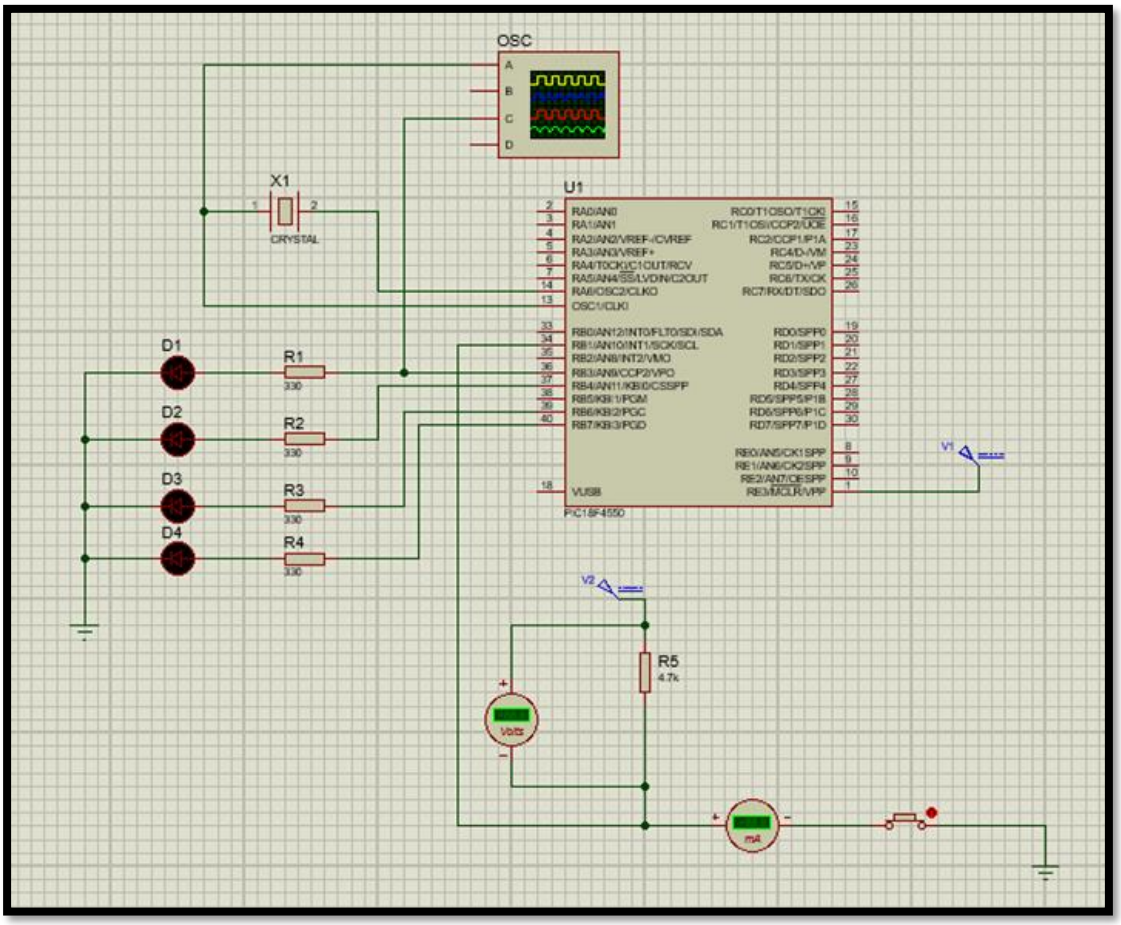


Figure 2.9 : A Complete Schematic Circuit Design

Source Code and Simulation

- Proteus Professional 8.7 from Labcenter Electronics is a professional application for designers to develop and test circuit boards.
- It has an interactive environment that allows you to see and alter simulations. It also gives you access to the source code so you can customise each piece independently.

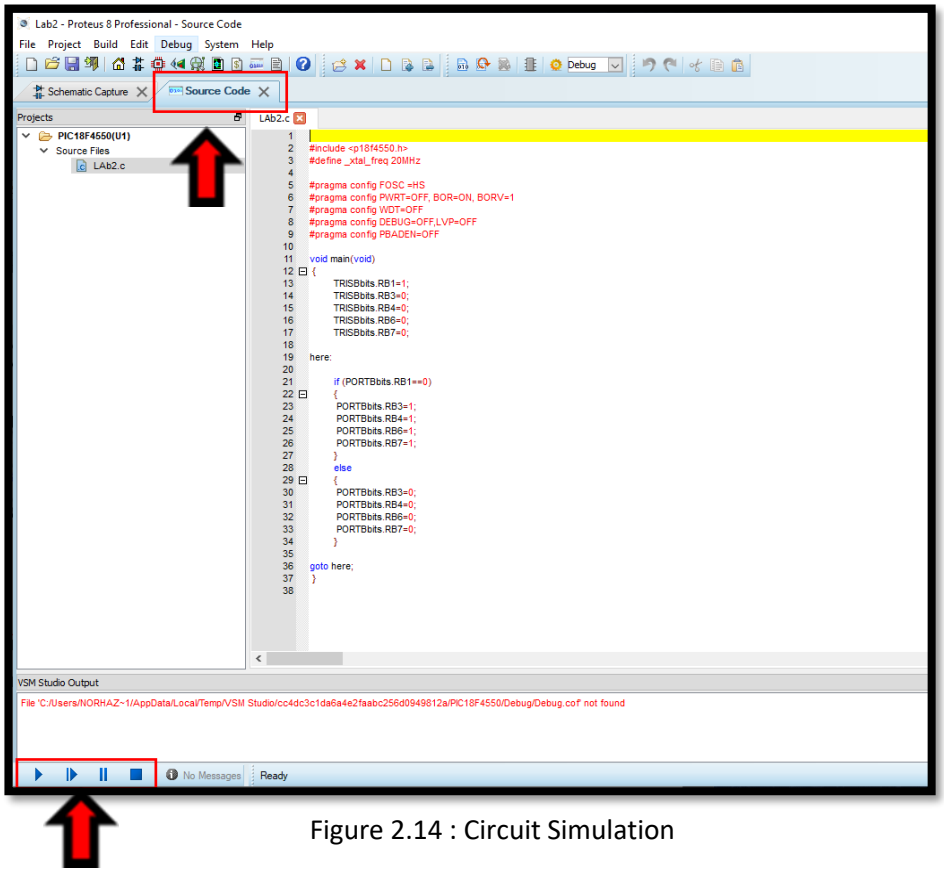


Figure 2.14 : Circuit Simulation

Steps to do a Circuit Simulation

- Type your source code and click Build ---→Build Project to compile your source code.
- Use a RUN button to START and STOP button tu END the simulation.
- By selecting the Virtual Instruments Mode icon in Proteus VSM, you can see a comprehensive list of measurement instruments.

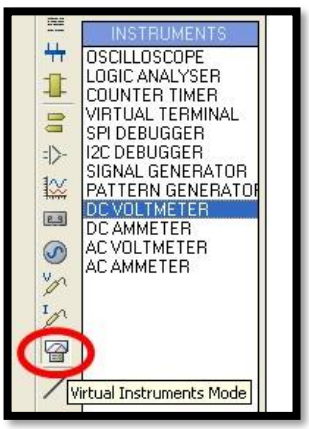


Figure 2.15 : Virtual Instruments

CHAPTER 3

PIC TIMER

PROGRAMMING IN C

TIMER REGISTER

Timers and Counters in Microcontroller

A timer is a clock that governs the sequence of events over a set period time. Timers are used to create time delays and as a counter to count events that occur outside of the microcontroller.

The counter increments from 0 to (2n)-1, where n denotes the number of bits.

In PIC18F4550, there are 8-bit and 16-bit timers

As a reference clock, the timer uses the internal clock.

External clocks or pulses applied through port pins are counted by the counter.

- The fundamental benefit of timers and counters is

  they run independently of the microcontroller CPU

  - ❑ the timer values can be accessed at any moment.

- Microcontroller consists of 1 or more hardware timer modules of different bit lengths
  - ❑ 8-bit timers- capable of counting 0-255
  - ❑ 16-bit timer – capable of counting 0-65535

- The initial values of the timer register can be set by the user and can be used to generate required counts.

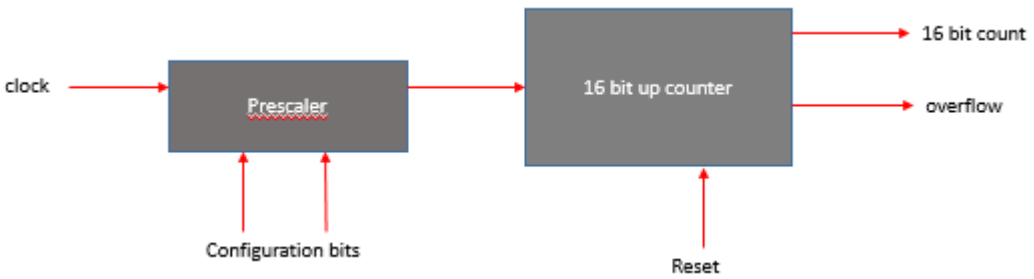
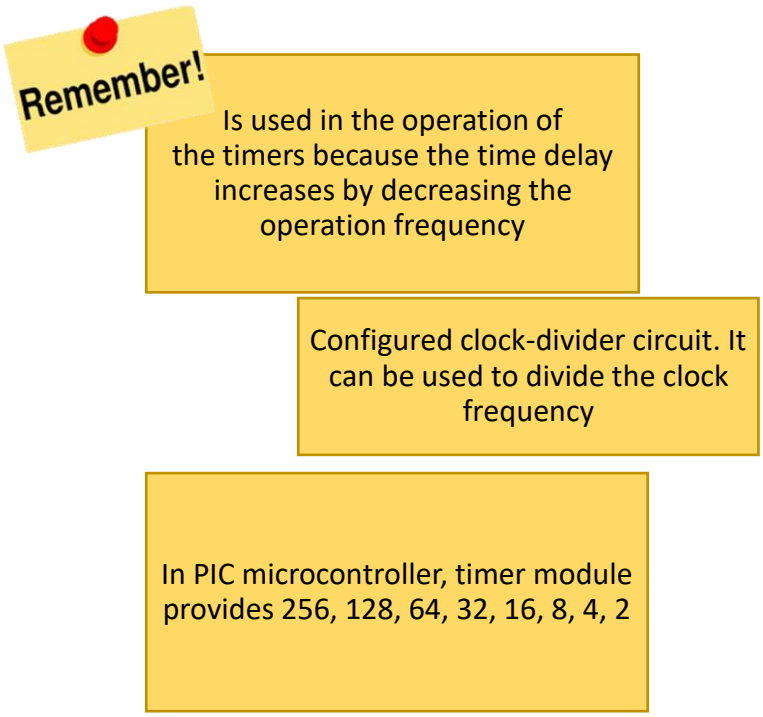


Figure 3.1 : Timer Block

- Example: The sensor is set on a students counter to detect a student who enters a room, and the sensor goes high when someone crosses the door. The sensor's output is connected to the microcontroller's Timer Clock Input Pin. Each time someone walks through the door, the timer goes up.

Prescaler



- **Overflow**

This indicates that the counter has reached its maximum output and has reset to zero.

Overflow interrupts are generated when the counter overflows and the overflow flag becomes 1.

Overflowed can provide an extra option to count the number of times and extend the range of the counter.

PIC18F4550 TIMER

- There are four hardware timers on the PIC18F4550: Timer 0, Timer 1, Timer 2, and Timer 3. Timer 2 is an 8-bit timer, whereas the rest are 16-bit timers..

(i) Timer 0

The Timer0 module has the following features:

- Timer0 can work as both 8-bit and 16-bit modes timer/counter
- Select able clock source (internal or external)
- Software programmable Prescaler
- Interrupt on overflow from 0XFF to 0X00

(ii) Timer 1

The Timer1 module has the following features:

- Timer1 can work as 16-bit timer or counter
- 8-bit registers that can be read and written (TMR1H and TMR1L)
- Source of clock selection (internal or external)
- Interrupt on overflow
- Timer1 can be multiplexed with other peripherals such as an ADC to generate CCP (Capture, Compare, and PWM) event triggering.

(iii) Timer 2

The Timer2 module has the following features:

- 8-bit Timer and Period registers (TMR2 and PR2, respectively)
- Software programmable prescaler (1:1, 1:4 and 1:16)
- Interrupt on TMR2 to PR2 match
- Use as the MSSP (Master Synchronous Serial Port) modul's shift clock if desired.

(iv) Timer 3

The Timer3 module timer/counter incorporates these features:

- Timer3 can work as 16-bit timer or counter
- 8-bit registers that can be read and written (TMR3H and TMR3L)
- Interrupt on overflow
- Timer3 can be multiplexed with other peripherals such as an ADC to generate CCP (Capture, Compare, and PWM) event triggering.
- or external Selectable clock source (internal al)

REGISTER USED FOR TIMERS IN PIC

- PIC18F4550 Timer0 Implementation The Timer 0's operation can be deduced from this block diagram.

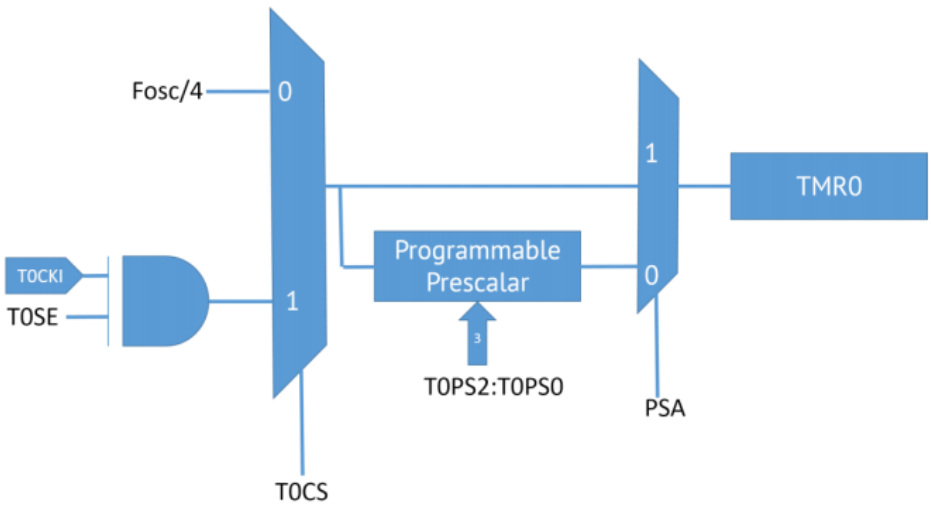


Figure 3.2: Timer 0 block diagram

1. The Timer0 stores the value TMR0 registers, TMR0H is the high byte (when using 16-bit mode) and TMR0L is the lower byte (when using 8-bit mode)
2. Timer0 can be used as both a timer and a counter. In timer mode, the module will increment every clock unless a prescaler value is placed into the TMR0 register.
3. To select the timer mode, clear the T0CS bit (5th bit) of the T0CON register.

REGISTER CONFIGURATION

These settings are usually applied via other special function registers inside the microcontroller. With these timer configuration, we can use to measure execute time.

```
T0CONbits.T08bit = 0;          //16-bit mode selector
T0CONbits.T0CS = 0;            // internal clock selected (timer Mode ON)
T0CONbits.PSA =1;              // prescaler NOT Assigned
T0CONbits.TMR0ON =1;           // Turn ON the timer
```

Figure 3.3: Register Configuration

USING PRESCALER IN TIMER

- Clear the PSA bit with Prescaler and assign the matching value to the T0PS2, T0PS1, and T0PS0 registers.

Table 3.1: Prescaler TIMER0

T0PS2	T0PS1	T0PS0	PRESCALER VALUE
1	1	1	1:256
1	1	0	1:128
1	0	1	1:64
1	0	0	1:32
0	1	1	1:16
0	1	0	1:8
0	0	1	1:4
0	0	0	1:2

IMPLEMENTING COUNTER USING TIMER0 MODULE IN PIC18F4550

- To select the Counter mode, set the T0CS bit in the T0CON register. The pulse on pin RA4/T0CKI causes Timer0 to increment. The rising edge or falling edge is determined by the T0SE bit in the T0CON register. If set this bit to true, it will select a rising edge, and if set it to false, it will select a falling edge. In the chapter Timer Delay Implementation, the process for calculating the delay is presented.

REGISTER CONFIGURATION

```
T0CONbits.T0SE= 0;          //falling edge selected
T0CONbits.T0CS = 0;         // internal clock selected (timer Mode ON)
T0CONbits.PSA =1;           // prescaler NOT Assigned
T0CONbits.TMR0ON =1;        // Turn ON the timer
```

Figure 3.4: Register Configuration

TIMERS REGISTERS CONFIGURATION

- Every Timer has a set of registers that must be set up for the necessary operations. The bit length of the Timer register can be as follows:

- 08 bit timers – These can count between 0-255
- 16 bit timers – These can count between 0-65536
- 32 bit timers – These can count between 0-4294967296

TIMER 0

- The registers of Timer0 have been explained below :

T0CON – Timer0 Control Register

R/W-I	R/W-I	R/W-I	R/W-I	R/W-I	R/W-I	R/W-I	R/W-I
TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0
bit 7							bit 0

TMR0ON : Timer0 ON/OFF bit
This bit is SET to HIGH to Enable the Timer0
1= Enable the Timer0
0= To Stop Timer0

T08BIT : 8/16 bit mode selection bit
This bit selects the Timer Mode
1= Timer0 is configured as an 8-bit timer/counter
0= Timer0 is configured as a 16-bit timer/counter

T0CS : Timer0 Clock Source Set
Timer Mode is selected by clearing the TMR0CS bit of the Control Register.
1= Transition on T0CKI pin
0=Internal Instruction cycle clock (CLK0)

T0SE : Timer0 Source Edge select
The raising or falling transition of the incrementing edge for either input source is determined by the TMR0SE bit in the Control Register
1= Increment TMR0 on HIGH to LOW Transition
0= Increment TMR0 on LOW to HIGH Transition

PSA : Prescaler Assignment
The prescaler is enable by clearing the PSA bit of the Control Register.
1= Prescaler not Assigned to TMR0
0= Prescaler Assigned to TMR0

There are eight prescaler options for the Timer0
111 = 1:256 Prescaler Value
110 = 1:128 Prescaler Value
101 = 1:64 Prescaler Value
100 = 1:32 Prescaler Value
011 = 1:16 Prescaler Value
010 = 1:8 Prescaler Value
001 = 1:4 Prescaler Value
000 = 1:2 Prescaler Value

TMR0H and TMR0L

- Timers have a 16-bit resolution. Because the PIC18 has an 8-bit architecture, each 16-bit timer is accessed as two distinct low byte (TMRxL) and high byte (TMRxH) registers (TMRxH).

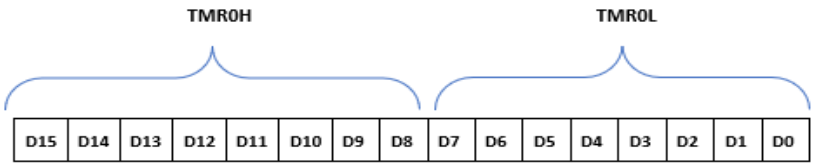
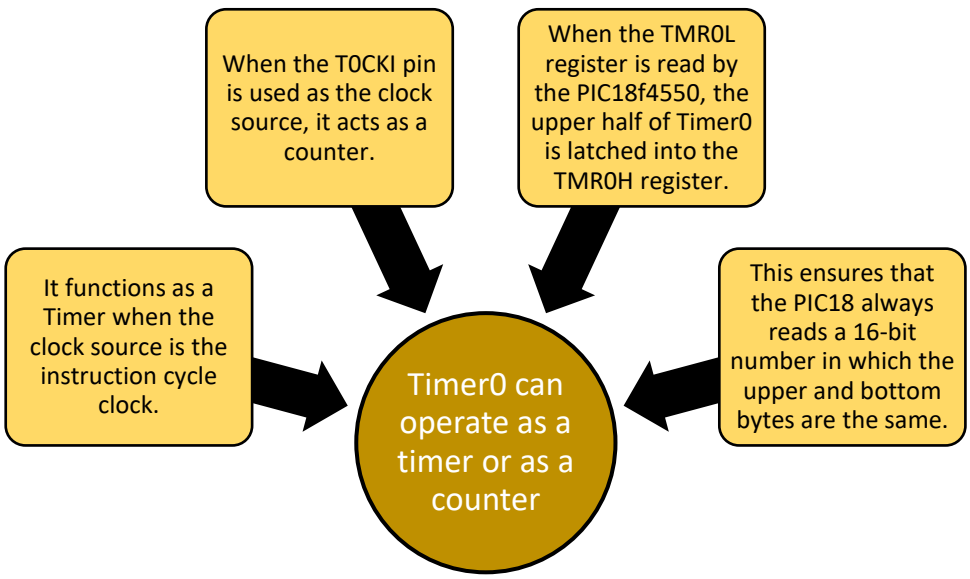


Figure 3.5: TMR0H and TMR0L Register

C PROGRAM FOR TIMERS IN PIC

Working of PIC Microcontroller Timer



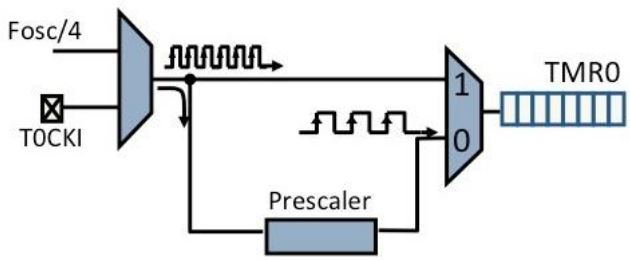
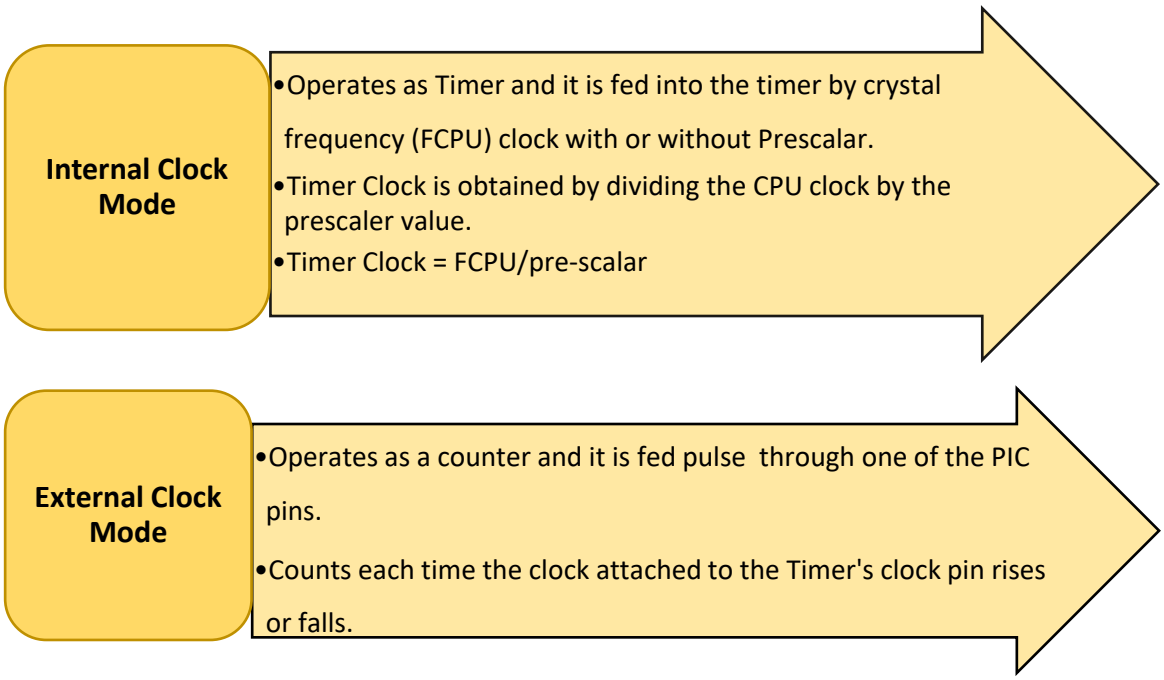


Figure 3.6: Timer 0 block diagram

CLOCK SOURCE OF PIC MICROCONTROLLER TIMERS

- The TIMER0 is the most basic. It's crucial to understand how the timer calculates time delay while configuring a timer. The Timer clock source is first set.



DELAY CALCULATION OF TIMERS

- PIC 18F has the ability for external with internal clock sources but we are using Timer0 with an internal clock (Timer mode).

STEP 1 : Instruction time ,TCY = $\frac{4}{Fosc}$

STEP 2 : Find the number of ticks

FFFF - register value + 1 = X ticks (Hex)

@

take the negative value of the tick counts

STEP 3 : Time Delay = TCY x number of ticks (in decimal) x prescaler

EXAMPLE 1

Calculate the time delay generated by Timer0 if T0CON, TMR0H and TMR0L which are configured as below. Given, Frequency Oscillator= 20MHz.

T0CON = 0X01
TMR0H = 0XBC
TMR0L = 0X83

ANSWER :

1

TCY = 4 / f_{osc}
= 4 / 20x10^6
= 0.2 µs

2

No of ticks = (FFFF-BC83) + 1
= 437C + 1
= 437DH
= 17277 d#

@

No of ticks = - (TMR0H:TMR0L)
= - BC83
= 437DH
= 17277 d#

3

Time delay = No of ticks x TCY x PSA
= 17277 x 0.2 µ x 4
= 13.8 ms

EXAMPLE 2 ...IER0 of PIC18 is used to generate 5ms time delay. Exhibit the value ...T0CON, TMR0H and TMR0L. No pre-scaler options to generate the delay. XTAL = 20MHz.

ANSWER:

1. TCY = 4 / fosc
   = 4 / 20 M
   = 0.2 µs

2. No of ticks = Time delay / (TCY x PSA)
   = 5ms / (0.2 µs x 1)
   = 25000
   = 61A8 H

3. Register Value = - (No of ticks)
   = - (61A8)
   = 9E58

@

Register Value = FFFF- Register Value +1

= FFFF - 61A8 +1

= 9E58

- ❑T0CON = 0X08
- ❑TMR0H = 0X9E
- ❑TMR0L = 0X58

EXAMPLE 3

Find the value of T0CON if a programmer wants to program Timer0 in16-bit mode with a Prescaler of 64 and use an internal clock (Fosc/4) for the clock source at the positive edge.

ANSWER:

T0CON = 0b00000101

R/W-I	R/W-I	R/W-I	R/W-I	R/W-I	R/W-I	R/W-I	R/W-I
TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0
bit 7							bit 0

PROTEUS SIMULATIONS OF TIMERS

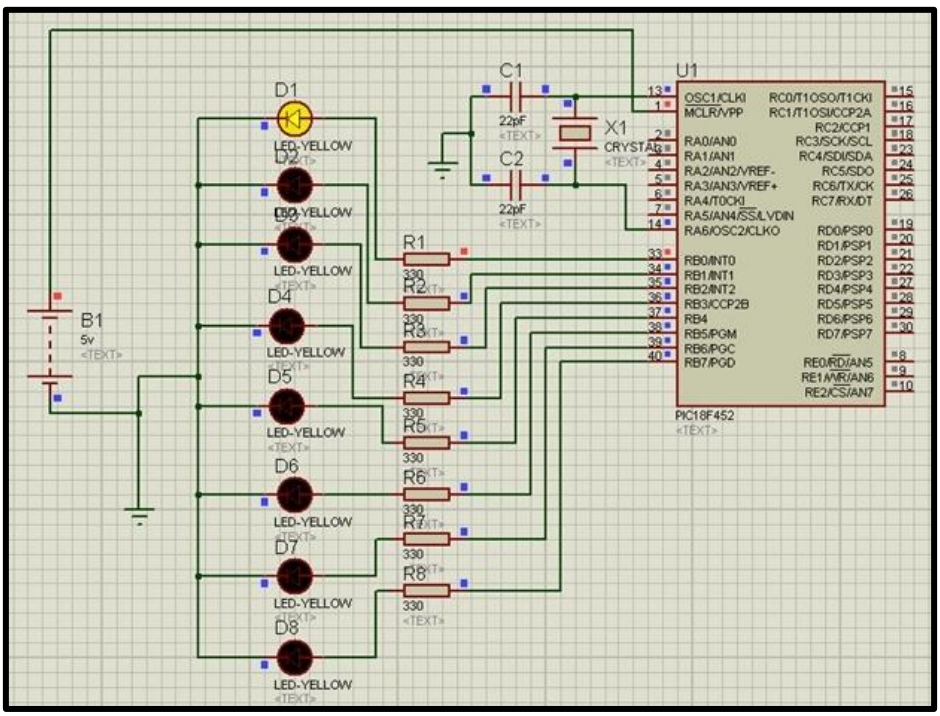


Figure 3.7: Schematic Circuit of Timer

PROGRAMMING

- To begin, we will select the Prescaler, Clock choice, and Mode of Timer0 using the T0CON register..
- Then we fill TMR0H with the higher byte of the Timer value, and TMR0L with the lower byte.
- To start the Timer, set the TMR0ON bit. Wait until the TMR0IF flag reaches its maximum value.
- We set TMR0IF to zero and stop the Timer by clearing the TMR0ON bit as TMR0IF rises. To restart Timer0, use the while loop to continue the operation of inserting higher and lower bytes.
- The time delay was determined by blinking a set of 8 LEDs one by one with a 1 second delay. The code can also be found below.

```
void main()
{
TRISB=0;
PORTB=0X01;
T0CON=0X07;
while(1)
        {
        TMR0H=0XB3;
        TMR0L=0XB4;
        T0CON.TMR0ON=11;

        while(INTCON.TMR0IF==0);
        T0CON.TMR0ON=0;
        INTCON.TMR0IF=0;
        PORTB=(PORTB<<1)|(PORTB>>7);
        }
}
```

Figure 3.8: Source code for Timer0

EXERCISE CHAPTER 3

1. Every timer needs a clock pulse to tick. Give two (2) examples of clock sources and differentiate their functions.

Answer:

Internal clock – The source is fed into the Timer by crystal frequency and used for time delay generation. It's called the Timer.

External clock – The Source is fed into the pulse through one of the PIC pins and this used for counter.

2. Explain the operation modes of Timer0.

Answer:

Timer0 can be used as a counter or a Timer to generate time delay. It can be used to count in 8 bit or 16 bit mode. For 8 bit mode, it can count for maximum of 256 times (0-255) before overflow occurs. For 16 bit mode, it can count for maximum of 65536 times (0-65535) before overflow occurs. When the Timer reaches its maximum value of FFFFH (16 bit) and FFH (8 bit), it rolls over to 0000 and TMR0IF is set to 1.

3. Identify the bit value to initiate the T0CON register to implement a counter using the input clock at pin T0CKI, 8 bit mode, clock increment on low to high and no Prescaler.

Answer:

T0CON = 1**1101**111

T08B = 1

T0CS = 1

T0SE = 0

PSA = 1

CHAPTER 4

INTERRUPT

PROGRAMMING IN C

PIC INTERRUPTS

- One of the most powerful features of any microcontroller is the use of its interrupts.
- An interrupt is simply a signal that tells the microcontroller to stop its execution or whatever it is doing, and interrupt is to save execution time and make certain aspect of the hardware programchange its priority to execute the sub-program assigned by that particular interrupt.
- These sub-programs are technically referred to as the Interrupt Service Routine (or ISR).
- The Interrupts are implemented either as internally initiated signals such as the one generated by a Timer or by externally initiated hardware devices like those used in external hardware interfaces.
- The primary reason for using an ming simpler.

THE INTERRUPT AND POLLING METHODS IN A MICROCONTROLLER

- In order to better understand the need to apply for interrupt programs, remember the example of how to implement a program to switch “ON” an LED if a push button switch is pressed. The example is reproduced below for clarity. To simplify the explanation, only 1 LED and switch is used.

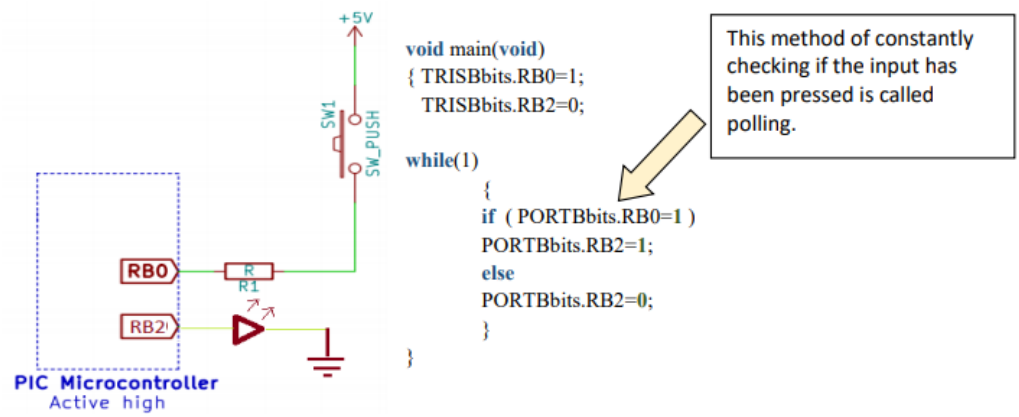


Figure 4.1 : Polling an input switch SW1

- Another Interrupt method feature is its ability to choose and prioritize all inputs it is supposed to process.
- This method is called the Round Robin method. The name itself is popularly used in networking classes to explain how data packet in a Star configuration network travels.
- Another added advantage of the Interrupt method is that if one of the sensors is damaged, under maintenance or is simply disregarded, the Interrupt program can be changed to exclude that sensor so that it does not have to be serviced. Also, the program for the Interrupt method can be modified to service all sensors. However, the processes can be assigned certain priority so that if any sensors are triggered at the same time, a higher priority sensor can be serviced first before the lower priority sensors are dealt with.

Table 4.1: Differences between the Polling Method and the Interrupt Method

The Polling Method	The Interrupt Method
The state of a gadget is constantly monitored by the microcontroller. This is done by a programme that regularly checks to see if the device has communicated the microcontroller with a different state.	To notify the microcontroller of a status change, the device will issue an interrupt signal. The gadget is not regularly checked by the microcontroller.
If the software detects a status change, it executes the service-running application (not ISR). It then moves on to the next device to monitor until all of them have been serviced.	The microcontroller halts the execution of the programme it is running. The software is then run to serve the device. ISR (interrupt service routine) or interrupt handler is the programme that is associated with the interrupt.
It is unable to set a priority because it monitors all devices in a round-robin manner.	The microcontroller can pay attention to each device based on the priority assigned to it.
You can't ignore an equipment that has to be serviced.	Can choose to ignore a service request from a device.

SOURCES OF INTERRUPTS

- The following are the sources of Interrupts available for the PIC series microcontroller. The ones listed here are specifically for the PIC18F4550 microcontrollers.

Port B Change Interrupt	Timers Interrupts
External Hardware Interrupt	The Serial Communication Interrupt
The Analogue to Digital Converter Interrupts	The Pulse Width Modulation Interrupt.

TIMERS INTERRUPTS

- There are 4 timers available in the PIC184550. These are Timer0 (TMR0), Timer1 (TMR1), Timer2 (TMR2) and Timer3 (TMR3)..

Table 4.2: The PIC18F4550 timer interrupts and its associated signals

TIMER INTERRUPT	FLAG BIT	REGISTER	ENABLE BIT	REGISTER
Timer0	TMR0IF	INTCON	TMR0IE	INTCON
Timer1	TMR1IF	PIR1	TMR1IE	PIE1
Timer2	TMR2IF	PIR1	TMR2IE	PIE1
Timer3	TMR3IF	PIR2	TMR3IE	PIE2

- First row, TIMER0 has the flag bit TMR0IF.
  - The signal TIMER0 will use in initiating an interrupt service routine.
  - It also has its enable bit, TMR0IE . The enable bit simply enables or disables the function of the timer.
  - TMR0IF and TMR0IE are located at the INTCON register.
  - The TIMER1, TIMER2 and TIMER3, its flag signal and enable bit can be located on different registers. This arrangement is normal for microcontrollers to save memory space.
- Figure 4.2 below shows the full content of the 8-bit INTCON register. The circled section shows the location of the enable bit (bit 5) and the flag of the TIMER0 (bit 2).

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF(1)
bit 7							bit 0

Figure 4.2: The full content of the Interrupt Control Register (INTCON)

- Figure 4.3 shows the content of all registers involved in TIMER operations.
- The blue circles show the important bit involved in the operation of TIMER1, TIMER2 and TIMER3 .
- It includes the flag bit and enable bit. The Flag bit is used to signal the microcontroller if the interrupt has been activated. The enable bit will enable or disable the flag bit. It means we must enable this bit first if we want to use timer interrupt.

PIR1: PERIPHERAL INTERRUPT REQUEST (FLAG) REGISTER 1

R/W-0	R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
SPPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
bit 7							bit 0

PIR2: PERIPHERAL INTERRUPT REQUEST (FLAG) REGISTER 2

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
OSCFIF	CMIF	USBIF	EEIF	BCLIF	HLVDIF	TMR3IF	CCP2IF
bit 7							bit 0

PIE1: PERIPHERAL INTERRUPT ENABLE REGISTER 1

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SPPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
bit 7							bit 0

PIE2: PERIPHERAL INTERRUPT ENABLE REGISTER 2

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
OSCFIE	CMIE	USBIE	EEIE	BCLIE	HLVDIE	TMR3IE	CCP2IE
bit 7							bit 0

Figure 4.3: The full content of the PIR1 , PIE1 , PIR2and PIE2 register

REMEMBER

TIMER0 INSTRUCTIONS	INTCONbits.TMR0IF INTCONbits.TMR0IE
TIMER1 INSTRUCTIONS	PIR1bits.TMR1IF PIE1bits.TMR1IE
TIMER2 INSTRUCTIONS	PIR1bits.TMR2IF PE1bits.TMR2IE
TIMER3 INSTRUCTIONS	PIR2bits.TMR3IF PIE2bits.TMR3IE

EXTERNAL HARDWARE INTERRUPTS

- This interrupt is specially provided so that a designer can fix a maximum of three external devices such as a switch, sensor or a signal to the microcontroller and process them via interrupts.
- The microcontroller pins provided are pin RB0 (INT0), RB1 (INT1) and RB2 (INT2). Any sensors can fix here, such as a proximity alert sensor, heat sensor and so on.
- Figure 4.4 below is how the setup looks for fixing the devices to the external hardware interrupts. There is a total of three INTx. All of its flag signals are positive edge triggered by default. The convention to name its signals is using INT0IF and INT0IE for external hardware interrupt 0, where you use the number 0 in the middle to denote which interrupt you are using.

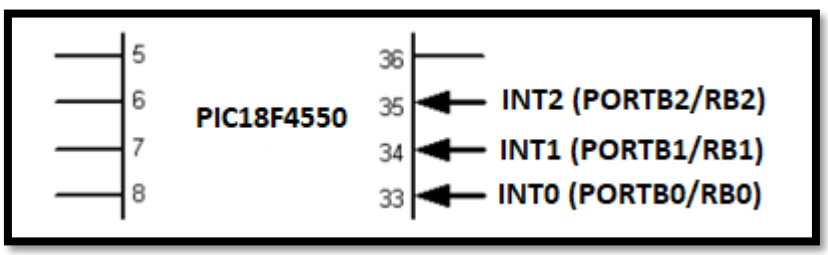


Figure 4.4: The Setup for External Hardware Interrupts

- The relevant signals for all external hardware interrupts are listed in Table 4.3. All the signals are contained in the INTCON register for INT0 only, while for INT1 and INT2, all signals are contained in INTCON3 register. Scroll to the next page to find out where INTCON 2 is used.

Table 4.3: The PIC18F4550 external hardware interrupts and its associated signals

EXTERNAL HARDWARE INTERRUPT	FLAG BIT	REGISTER	ENABLE BIT	REGISTER
INT0 (RB0)	INT0IF	INTCON	INT0IE	INTCON
INT1 (RB1)	INT1IF	INTCON3	INT1IE	INTCON3
INT2 (RB2)	INT2IF	INTCON3	INT2IE	INTCON3

INT0 INSTRUCTIONS	INTCONbits.INT0IE INTCONbits.INT0IF
INT1 INSTRUCTIONS	INTCON3bits.INT1IE INTCON3bits.INT1IF
INT2 INSTRUCTIONS	INTCON3bits.INT2IE INTCON3bits.INT2IF

- Figure 4.5 below shows, the INTCON register also contains the external hardware interrupt flag and enable bit for External Hardware Interrupt 0 (INT0).

INTCON : INTERRUPT CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF(1)
bit 7							bit 0

INTCON2 : INTERRUPT CONTROL REGISTER 2

R/W-1	R/W-1	R/W-1	R/W-1	U-0	R/W-1	U-0	R/W-1
$\overline{\text{RBPU}}$	INTEDG0	INTEDG1	INTEDG2	-	TMR0IP	-	RBP
bit 7							bit 0

INTCON3 : INTERRUPT CONTROL REGISTER 3

R/W-1	R/W-1	U-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0
INT2IP	INT1IP	-	INT2IE	INT1IE	-	INT2IF	INT1IE
bit 7							bit 0

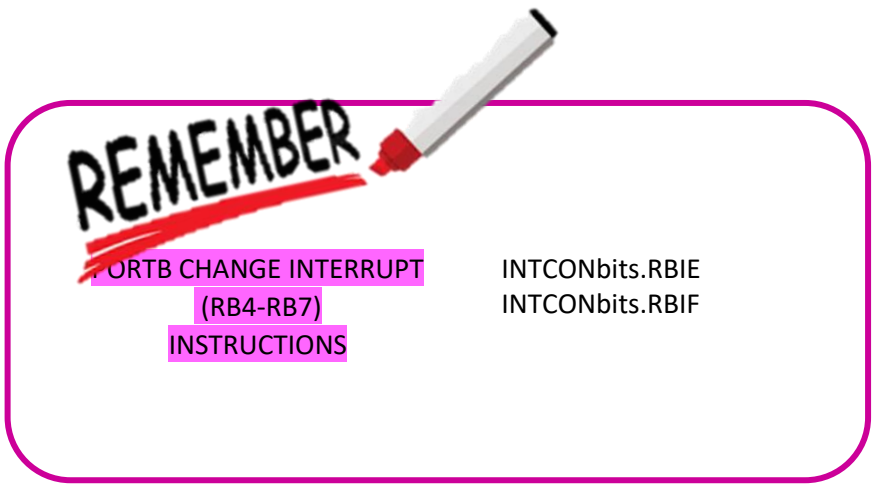
Figure 4.5: The full content of the INTCON0, INTCON1 and INTCON2 register

PORT B CHANGE INTERRUPT

- This interrupt only monitors PORTB4, PORTB5, PORTB6 and PORTB7. The same register is used for these four pins.

PORTB CHANGE INTERRUPT	FLAG BIT	REGISTER	ENABLE BIT	REGISTER
RB4-RB7	RBIF	INTCON	RBIE	INTCON

Figure 4.6 : The PIC18F4550 Port-B Change Interrupts



DIFFERENCES BETWEEN EXTERNAL HARDWARE INTERRUPT AND PORTB-CHANGE INTERRUPT

Table 4.4: External hardware interrupt vs Port B change interrupt

EXTERNAL HARDWARE (INT0-INT2) INTERRUPT	PORTB-CHANGE INTERRUPT
PORTB : RB0 (INT0), RB1 (INT1), RB2(INT2)	PORT B : RB4, RB5. RB6 and RB7
Each of the INT0-INT2 interrupts has its own pin and is self-contained.	Even though it can use up to four pins, it uses all four of the PORTB pins RB4-RB7 and is considered a single interrupt.
Each of the INT0-INT2 interrupts has its own flag and is self-contained.	Only single flag.
It's possible to set it up to activate on the positive or negative edge.	If the condition of any pin changes from LOW to HIGH or HIGH to LOW, the interrupt is triggered.

SERIAL COMMUNICATION INTERRUPTS (RCIF and TXIF)

- The PIC18F4550 microcontroller is also capable of serial communication using the RS232 interface.
- This is also called Universal Synchronous Asynchronous Receiver Transceiver (USART) communication. This means 8-bit data can be transmitted or received by the microcontroller.
- The two associated interrupts used are the Transmit Interrupt Flag (TXIF) and the Receive Interrupt Flag (RCIF).
- Figure 4.7 shows where the two interrupt pins are located in the PIR1 registers while Figure 4.8 shows the setup for using the PIC18F4550 microcontroller for RS232 serial communication transmission. Notice that the two pins used are TxD and RxD.

R/W-0	R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-x
SPPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
bit 7							bit 0

Figure 4.7: The Interrupt pins located in the PIR1 Registers

TTL	
Logic 0	0 V
Logic 1	5 V

RS232	
Logic 0	+3V to +25V
Logic 1	-3V to -25V

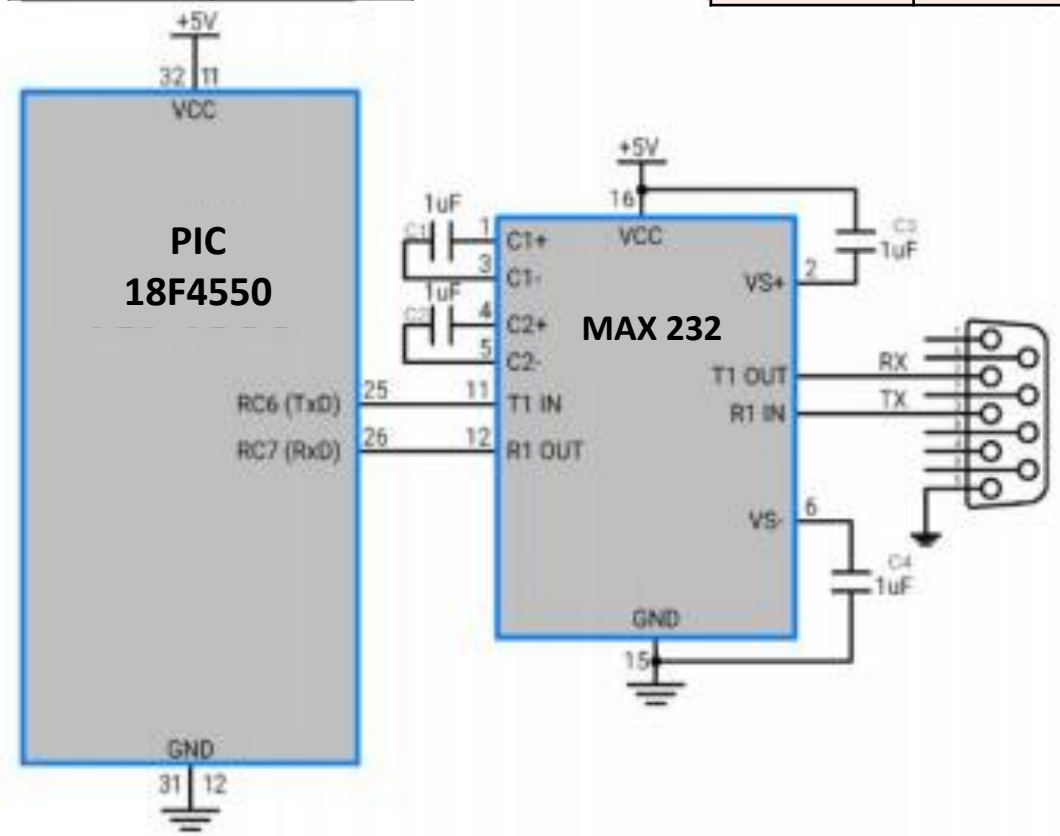


Figure 4.8: PIC18F4550 microcontroller RS232 USART communication setup

THE ANALOGUE TO DIGITAL CONVERTER INTERRUPTS (ADIF)

- The PIC18F4550 microcontroller allows for the operation of a 10 bit analogue to digital conversion.
- This is done via the input bit pin RA0 – RA12 of the microcontroller as shown in Figure 4.9.
- It means, each pin can be set up to monitor and convert 13 different input signals to its equivalent digital output. The interrupt associated with the ADC module is the ADIF or ADC Interrupt Flag

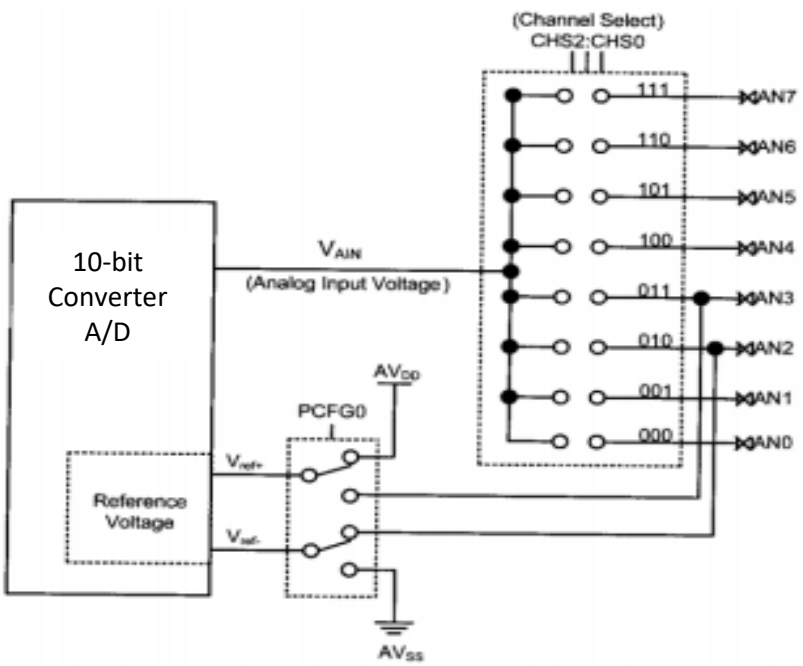


Figure 4.9: The 13 input channels available to the PIC18F4550

- The PIR1 register holds the location of the ADIF Interrupt, while the PIE1 register holds the enable signal.

PIR1 : PERIPHERAL INTERRUPT REQUEST (FLAG) REGISTER 1

R/W-0	R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-x
SPPIF(1)	ADIF	RCIF	TXIF	SSPIE	CCP1IF	TMR2IF	TMRIF
bit 7							bit 0

PIE1 : PERIPHERAL INTERRUPT ENABLE REGISTER 1

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SPPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMRIE
bit 7							bit 0

Figure 4.10: The ADIF and ADIE signals for ADC operation

THE PULSE WIDTH MODULATION (PWM) INTERRUPT

- DC motors can be driven by using a PIC18F4550 microcontroller.
- To provide variable power to the motors, the microcontroller provides a PWM signal as output. Its means the average power to the DC load can be changed. The motor speed will change if the power is changing
- Other applications can also be used where the PWM signal is needed to drive a load that needs variable pulse width.
- The pins used for the PWM module are shown in Figure 4.11 which is CCP1 and CCP2. The pins' name means compare and capture.

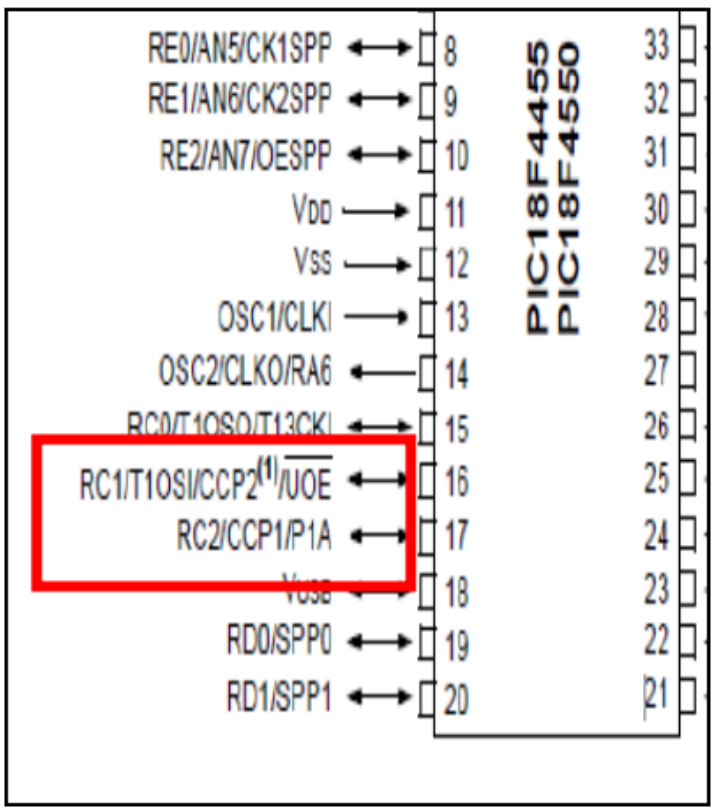


Figure 4.11: Common setup for motor load using PWM modules

- **PWM (Pulse Width Modulation)** is a signal that gives a load a variable pulse width. Figure 4.12 below is a typical example of how the PWM module is used for motor control. The L293D chip here is a motor driver.

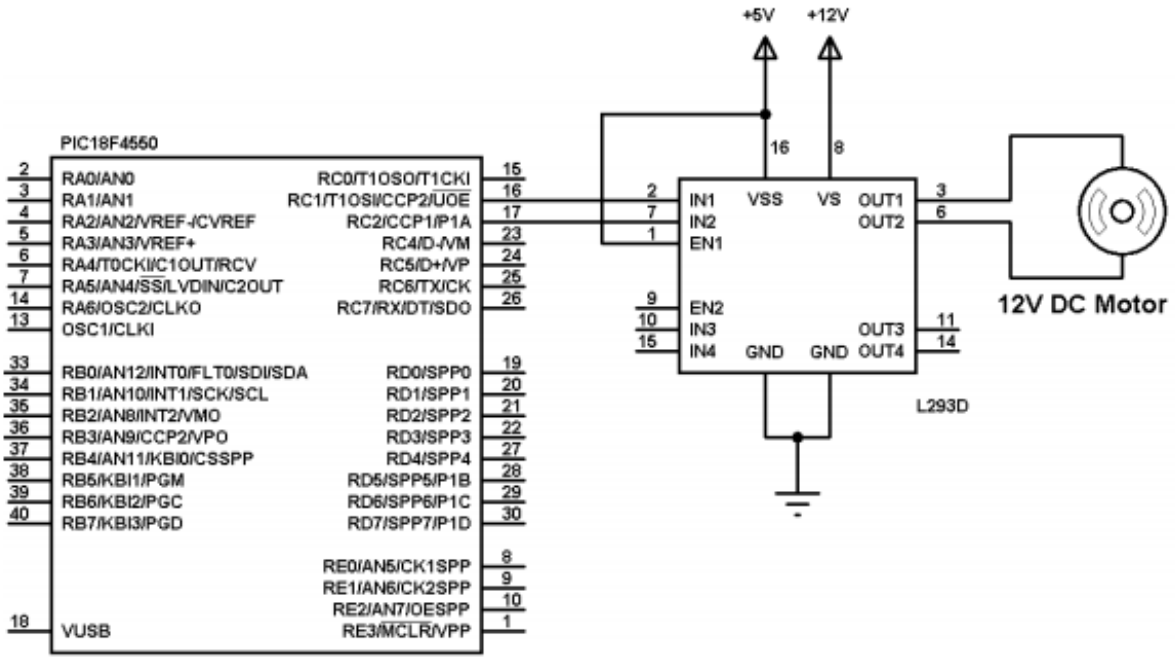


Figure 4.12: Common setup for motor load using PWM modules

- The associated signal pin here is the CCP1F and CCP2F for flag operation and the CCP1E and CCP2E for its enable bit. The Figure 4.13 shows its location in PIR1 and PIE1 registers.

PIR1 : PERIPHERAL INTERRUPT REQUEST (FLAG) REGISTER 1

R/W-0	R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
SPPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
bit 7							bit 0

PIR2 : PERIPHERAL INTERRUPT REQUEST (FLAG) REGISTER 2

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
OSCFIF	CMIF	USBIF	EEIF	BCLIF	HLVDIF	TMR3IF	CCP2IF
bit 7							bit 0

PIE1 : PERIPHERAL INTERRUPT ENABLE REGISTER 1

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SPPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
bit 7							bit 0

PIE2 : PERIPHERAL INTERRUPT ENABLE REGISTER 2

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
OSCFIE	CMIE	USBIE	EEIE	BCLIE	HLVDIE	TMR3IE	CCP2IE
bit 7							bit 0

Figure 4.13: The CCP1IF, CCP2IF,CCP1IE and CCP2IE location

ENABLE AND DISABLE AN INTERRUPT

- We have to Enable first each Interrupt we use.
- For example, if we want to use the Timer0, we will have to enable it through its enable bit TMR0IE. Only then can use its flag TMR0IF.
- In order to use any one of the PIC18F4550 interrupts, the one important enable signal that must be activated is the INTCON register's Global Interrupt Enable (GIE) bit. Take a look again at the INTCON register below.

INTCON : INTERRUPT CONTROL REGISTER

R/W-0	R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF(1)
bit 7							bit 0

Figure 4.14: Enabling the GIE and TMR0IE signals

- Suppose we want to use TIMER0. The INTCON register also contains TMR0IE bit, which must be enabled to use TIMER0. To enable this, we declare the following code :

```
void main (void)
{     INTCONbits.GIE = 1;              //Enable All the interrupts to be used
      INTCONbits.TMR0IE = 1;           //Enable the Timer0 Interrupt
      INTCONbits.TMR0IF = 0;           //Clear the Flag for the program to use
}
```

- From the above code we set GIE as “1” to enable it. It will become disabled if we set it to “0”. The second line also we set TMR0IE as “1” to enable it. The last code is typically applied to any interrupt where a “0” is set to clear the flag so that when the program runs, it could detect a “1”. This same concept applies to all interrupts for the PIC18F4550 microcontrollers

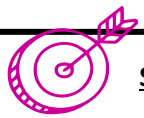
THE PIC INTERRUPT PROGRAMMING IN C

Table 4.5: The Memory Location of The Interrupt Service Routine Program in the ROM

INTERRUPT	ROM LOCATION (hex)
Power-on-Reset	0000
High Priority Interrupt	0008 (Default upon power-on-reset)
Low Priority Interrupt	0018

- Table 4.5 displays the precedence of Interrupts in two locations: 0008h and 0018h.
- The High Priority Interrupt's beginning Memory Address is 0008h. It means that when an interrupt occurs, the microcontroller will stop working and run the interrupt programme at address 0008h.
- This program that deals with a particular interrupt and resides at this address, is called the Interrupt Service Routine or ISR. It is a small code or program, but it has to be called an ISR since it is executing because of the interrupt signal.
- Each ISR can correspond to an interrupt in the microcontroller. However, if the IPEN bit in the RCON register is set to “1”, some interrupts can be set to low priority and, as a result, the ISR program execution could go to address 0018h.
- The term Interrupt Vector Table comes from Table 4.5 above, where all ISR locations are set in the table.

Below are the steps used in executing an interrupt. Refer to figure 4.15 when reading the explanations below to understand the interrupt processes better. Figure 4.15 shows what happens when a normal instruction is under execution beginning from the address 0200h.



STEP 1

- Suppose an interrupt signal is generated, which could come from any source such as a timer or external hardware.
- The microcontroller will stop the instruction it is executing in the memory midway. In Figure 4.15, we could see that the microprocessor stops its execution at address 0206h.
- The microcontroller comes to a halt at address 0206h, then proceeds to PUSH the memory location of the next instruction onto the Stack Pointer, which is at address 0208h.
- We can see this as the thin blue line in figure 4.15. Since the 16-bit Program Counter is the memory which keeps track of which memory location the microcontroller is executing, this PUSH operation will save the address 0208h into another Stack Pointer register and stay there until this whole process is over.
- As we can see in Figure 4.15, the PUSH operation will occasionally save the next two or three addresses as well in the Stack Pointer. It will do this to resume back the normal instruction execution at step 4 quickly.

STEP 2

- By default, if the program is not set to a low priority execution, the microcontroller will jump the execution address to 0008h which comes from the Interrupt Vector Table (IVT).
- This is done by automatically loading the address 0008h to the Program Counter. (look at the thin blue line in Figure 4.15). Thus allowing it to begin execution of any ISR program from that address.
- This is known as the IVT, and it directs the microcontroller to the Interrupt Service Routine's location (ISR).

STEP 3

- The interrupt service function is started after the microcontroller receives the address of the ISR from the IVT and jumps to it (ISR).
- There can be other ISR beginning from the address 0008h. In this example in figure 4.15, the ISR happen to be located at address 0300h.
- It will keep running at that location until it reaches the last instruction of the subroutine, which is RETFIE (Return from Interrupt Exit).

STEP 4

- The microcontroller returns to where it was paused after executing the RETFIE instruction.
- This is done by the reverse of the PUSH instruction, which is called POP. Here the address initially saved to the Stack Pointer is returned to the Program Counter.
- That means the microcontroller will sequentially "POP" the address 0208h back into the Program Counter together with the two other addresses it initially saved.
- Thus, the microcontroller will now continue to execute the original program it was executing before the interrupt came in

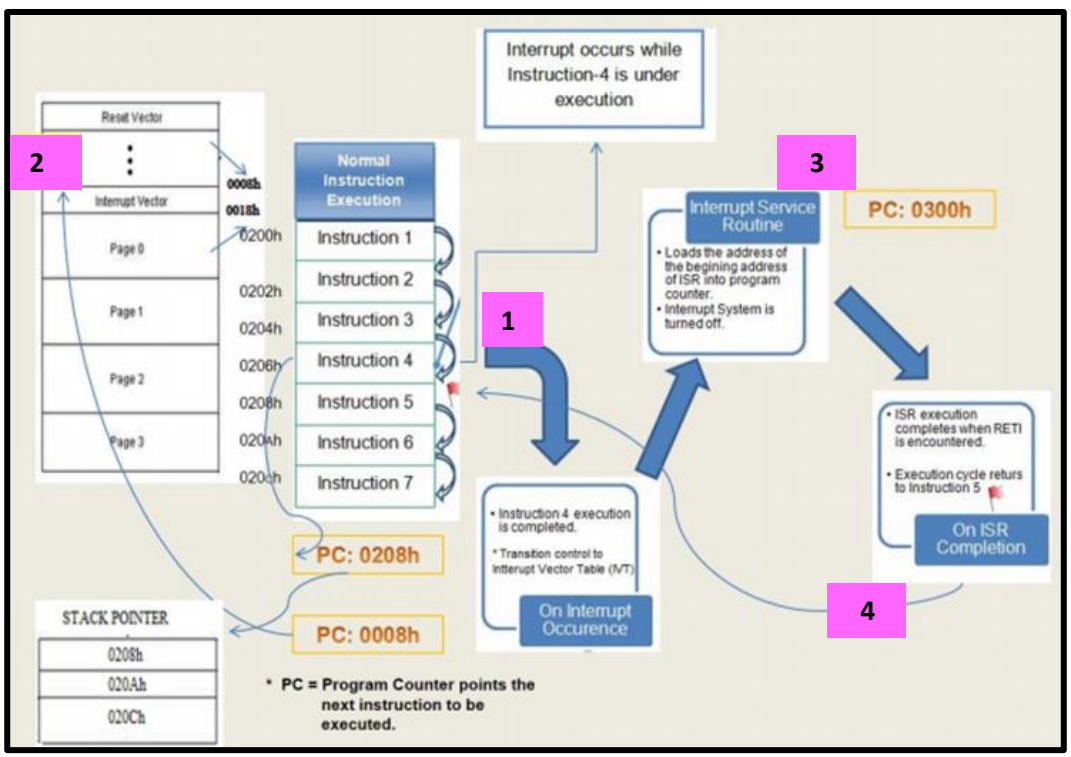


Figure 4.15: The Interrupt Processes in the Microcontroller

INTERRUPT PROGRAMMING APPLICATION IN PIC

- A simple code in the C language below is used to transfer some 8-bit data continuously from PORTC to PORTD.
- By using Timer0, after every 5000 clock cycles of transferring data, the program will toggle pin RB5 in a continuous operation. This code uses the polling method.

```
1  #include<p18f4550.h>
2
3  #pragma config WDT=OFF
4  #pragma config PBADEN=OFF
5  #pragma config LVP=OFF
6
7  //.....

8
9  #define myPB5bit PortBbits.RB5
10
11 void main(void)
12 {
13 TRISBbits.RB5=0;
14 TRISD=0;
15 TRISC=0xFF;
17 T0CON=0X08,
18 TMR0H-OXEC;
19 TMROL-0X78;
20
21 while(1)
22 {
23 INTCONbits.TMR0IF=0;
24 INTCONbits.TMR0ON=1;
25 INTCONbits.TMR0IE=1;
26 INTCONbits.GIE=1;
27
28 if(INTCONbits.TMR0IF==0)
28   PORTD=PORTC;
29 else
30   myPB5bit=~myPB5bit;
31   INTCONbits.TMR0IF=0;
32   INTCONbits.TMR0ON=0;

32 }
33 }
```

- This is a typical code using the polling method. The timer code introduced in Chapter 3 will be used here to fill the TOCON and TMR0 registers and data direction registers used here.
- We sometimes get confused and call this an interrupt method because it uses the bit TMR0IF flag at lines number 22 and 27. However, at line number 27, since the code checks if the flag is equal to “0” all the time, this is a polling method. Take a look at the example code on the following page (Figure 4.16).

- As shown in the code on the next page, this uses the interrupt programming method for the same task as above.
- It looks daunting at first, but the interrupt method simplifies things once you understand its basic. The left column of the code is the initial setup for interrupt programming.
- Here, there are two functions. The **My_HiPrio_Int() function** code beginning from line 11 sets the high priority interrupt location. The **chk_ivt() function** code beginning from line 24 is the vector table that is supposed to redirect the program execution to the **T0_ISR() function** at line 30 (second column) when an interrupt occurs.

```
1  #include<p18f4550.h>
2
3  #pragma config WDT=OFF
4  #pragma config PBADEN=OFF
5  #pragma config LVP=OFF
6
7  //.....
8  void chk_ivt();
9  void INT0_ISR(void);
10 #define myPB5bit PortBbits.RB5
11
12 //.....sets the high vector interrupts.....
13 #pragma code My_HiPrio_Int=0x08
14 void My_HiPrio_Int(void)
15 {
16 _asm
17 goto chk_ivt
18 _endasm
19 }
20 #pragma code
21
22 //....used for high priority interrupt only....
23
24 #pragma interrupt chk_ivt
25 void chk_ivt(void)
26 {
27 if(INTCONbits.TMR0IF==1)
28 INT0_ISR();
29 }
```

```
//.....main program.....

void main(void)
{
TRISBbits.RB5=0;
TRISC=0XFF;
TRISD=0X00;

T0CON=0X00,
TMR0H=0XEC,
TMR0L=0X78,

INTCONbits.TMR0IF=0;
INTCONbits.TMR0IE=1;
INTCONbits.GIE=1;
INTCON2bits.PEIE=1;


TOCONBbits.TMR0ON=1;

//.....

while(1)
{
PORTD=PORTC;    send data from PORTC to
PORTD
}

//.....

void INT0_ISR(void)
{
myPB5bit=~myPB5bit;
TMR0H=0XEC;
TMR0L=0X78;
INTCONbits.TMR0IF=0;
}
```

Figure 4.16 : Source code using Timer 0 Interrupt

- For the codes at the right side column, there are only two functions that you will need to remember.
- One is the **main() function** that runs before the interrupt occurs, and the other is the **ISR function** that runs if the interrupt occurs.
- The main() function is clearly shown on the right column at line number 3. This will be the same task that was done in the previous code which was implemented using the polling method. Most of the code is almost the same.
- However, at the while() loop, the code only deals with transferring data from PORTC to PORTD. This is done through the code “PORTD = PORTC;” .
- The program does not need to check the flag continuously. All this is done by the T0_ISR() function at line number 30.
- The T0_ISR() is the Interrupt Service Routine for the TIMER 0 interrupt flag. This function will complete the task of toggling the LED at bit R5 for every 5000 cycles. At this point, it should be clear that the only thing that should be coded here is whatever that was supposed to happen during an interrupt.

INTERFACING APPLICATION OF THE EXTERNAL HARDWARE INTERRUPT, INTX TO AN EXTERNAL DEVICE

- The code for this interrupt is not very different from the program using the Timer0 Interrupt.
- An example task in figure 4.17 below involving INT0 which is a switch tied at pin RB0. Two LEDs connected at pin RB2 and RB7 are output devices.
- The instruction #include at line number 2 in the left coloumn is a delay function library provided to code delay functions in a program easily (Figure 4.18).
- For example, to blink an LED with a specified delay. The instruction that uses this library is on the right column at lines number 19 and 21.
- For example, Delay10KTCYx(255); means the program introduces 10k x 255 x TCY (one cycle pulse duration) seconds worth of delay.

- During normal program execution, the LED D1 will blink at that rate. However, when the switch is pressed, that LED D1 will stop blinking.
- Instead, the ISR at line 27 will execute to blink the LED D2 at the same pulse duration as the LED D1

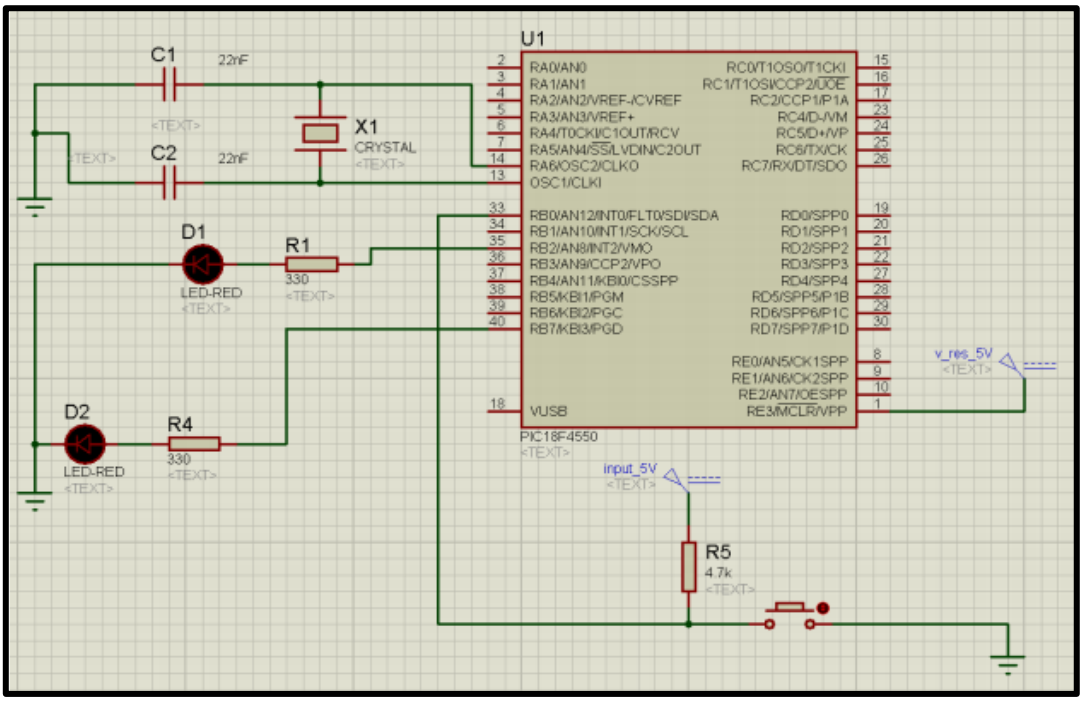


Figure 4.17: Test Setup for External Hardware Interrupt

```
1  #include<p18f4550.h>
2  #include<delays.h>
3  #pragma config WDT=OFF
4  #pragma config PBADEN=OFF
5  #pragma config LVP=OFF
6
7  //.....
8  void chk_ivt();
9  void INT0_ISR(void);
10 #define mybit PORTBbits.RB7
11
12 //.....sets the high vector interrupts.....
13 #pragma code My_HiPrio_Int=0x08
14 void My_HiPrio_Int(void)
15 {
16  _asm
17  goto chk_ivt
18  _endasm
19 }
20 #pragma code
21
22 //....used for high priority interrupt
only....
23
24 #pragma interrupt chk_ivt
25 void chk_ivt(void)
26 {
27 if(INTCONbits.INT0IF==1)
28 INT0_ISR();
29 }
```

```
//.....main program.....

void main(void)
{
TRISBbits.RB7=0;
TRISBbits.RB2=0;
TRISBbits.RB0=1;

INTCONbits.INT0IF=0;
INTCONbits.INT0IE=1;
INTCONbits.GIE=1;
INTCON2bits.INTEDG0=0;

//.....

while(1)
{
PORTBbits.RB2=1;
Delay10KTCYx(255);
PORTBbits.RB2=0;
Delay10KTCYx(255);
}

//.....

void INT0_ISR(void)
{
mybit=~mybit;
Delay10KTCYx(255)
INTCONbits.INT0IF=0;
}
```

Figure 4.18 : Source code using external hardware interrupt

- Looking at the code above, we notice that there is not much difference from the previous code used for the TIMER0 Interrupt.
- Here is the list of differences:
  - Since we are using INT0, the term INTCONbits.TMR0IF is replaced with INTCONbits.INT0IF.
  - The function T0_ISR() is replaced with INT0_ISR(). The name itself is just to reflect the change of interrupt usage.
  - The content of main() has been changed to reflect the new task.
  - Since different pins are used to configure the interrupt, this has been changed in the code, such as INTCONbits.INT0IE=1 is used instead of INTCONbits.TMR0IE=1;

o With this, it is easy to see how this code template can be modified easily to use other interrupt schemes.

EXERCISE CHAPTER 4

1. List three (3) external hardware interrupts in PIC

Answer:

INT0/RB0

INT1/RB1

INT2/RB2

2. An interrupt process is a signal from hardware or software to the processor (microcontroller) indicating an occurrence that requires rapid attention. Describe the interrupt response of the CPU in order to perform the event.

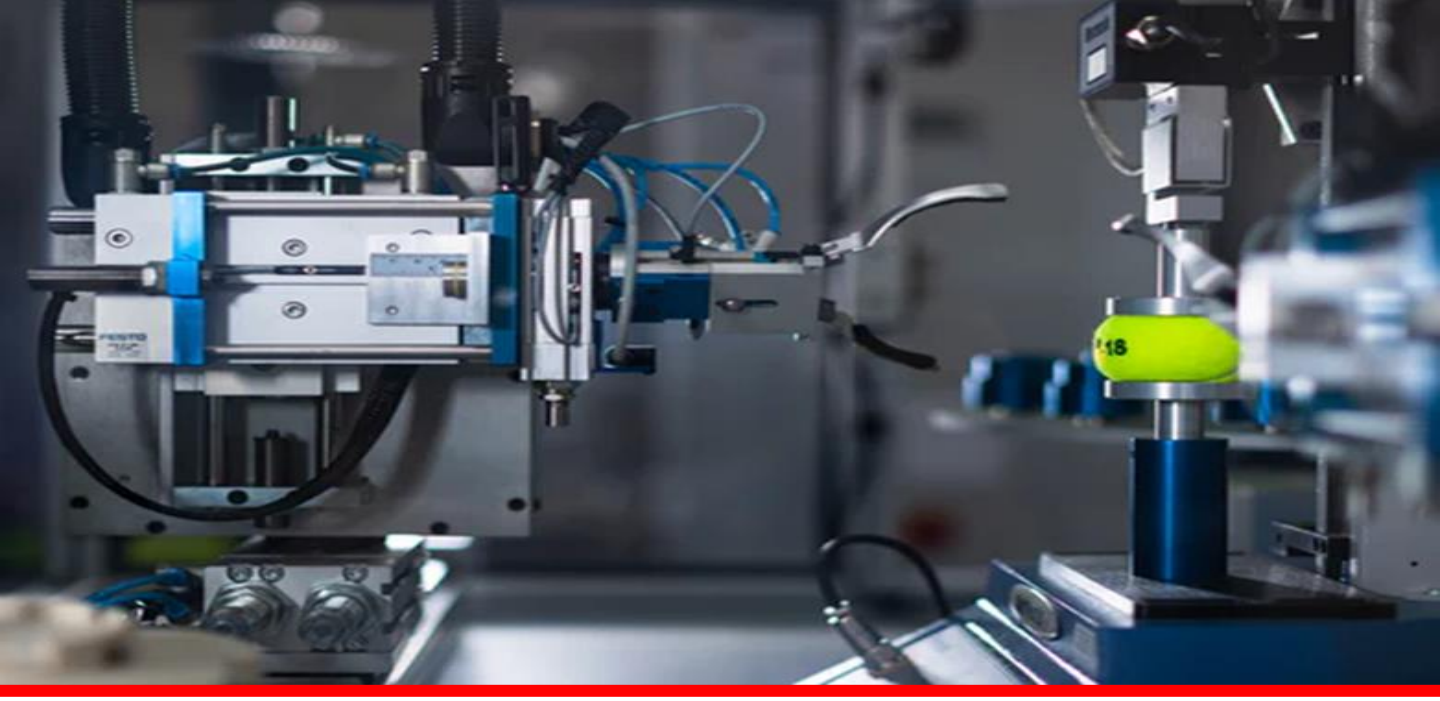
Answer:

To deal with the event, the CPU suspends its current activities, saves its state, and executes a short programme known as an interrupt handler (or interrupt service routine, ISR). The CPU resumes the preceding thread's execution after the interrupt handler completes.

3. Construct a C language program to interface with external hardware interrupt at pin RB1/INT1. An LED that is connected to pin RD0 will blink twice every time the INT1 is activated

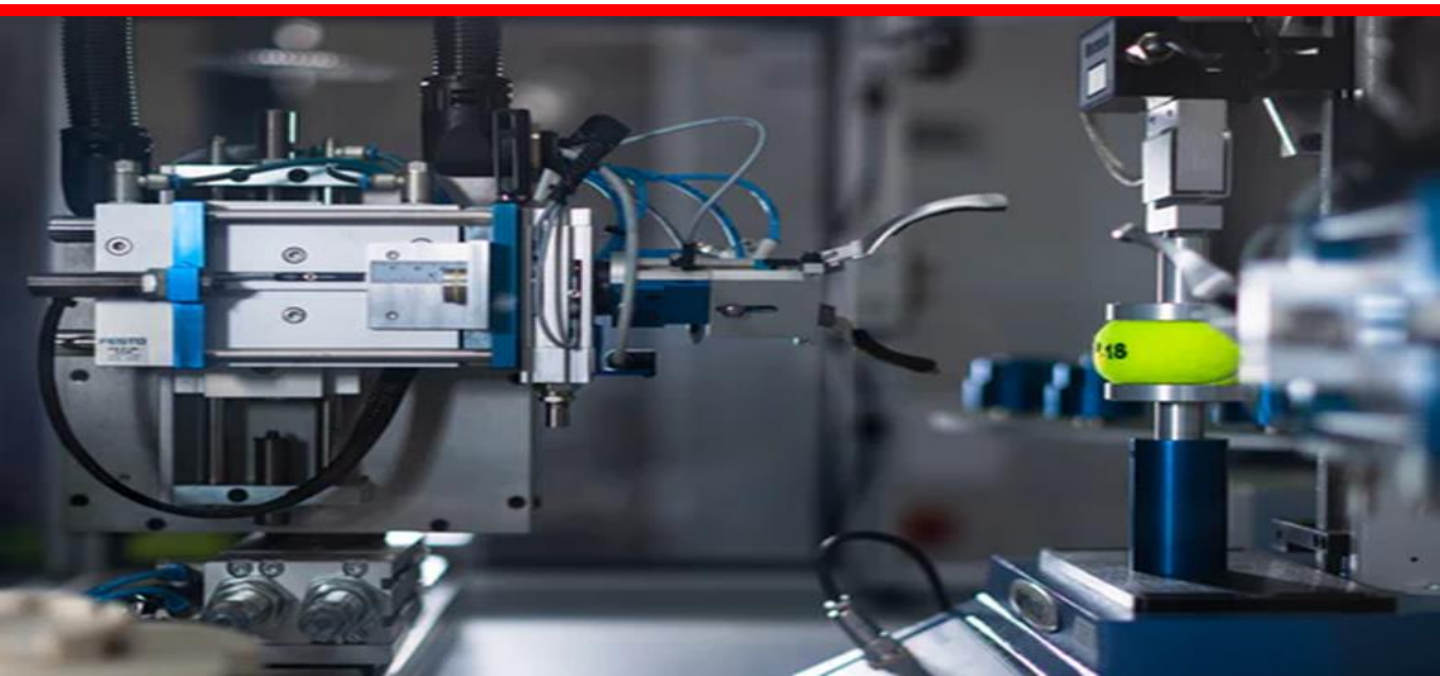
Answer:

```
#include <p18f4550.h>
void chk_isr (void)
void INT1_ISR (void)
#pragma interrupt chk_isr

void chk_isr (void)
{
If (INTCON3bits.INT1IF==1)
INT1_ISR();
}

#pragma code My_HiPrio = 0x08
void My_HiPrio_INT (void)
{
 _asm
GOTO chk_isr
 _endasm
#pragma code

void main (void)
{

TRISBbits.RB1=1;
TRISDbits.RD0=0;
INTCON3bits.INT1IF=0;
INTCON3bits.INT1IE=1;
INTCONbits.GIE=1;
while(1);
{}
}

void INT1_ISR(void)
 {
 PORTDbits.RD0=~ PORTDbits.RD0;
 PORTDbits.RD0=~ PORTDbits.RD0;
 INTCON3bits.INT0IF=0;
}
```



CHAPTER 5

HARDWARE INTERFACING



EXTERNAL DEVICES THAT CAN BE INTERFACED WITH PIC MICROCONTROLLER

- The PIC-Peripheral Interface Controller which was a family of MCUs manufactured by Microchip Technology that available in 8-bit, 16-bit and 32-bit.
- PIC microcontrollers are used widely in our real-life such as in advanced medical devices, video game peripherals, audio accessories, mobile phones and industrial robots.
- The PIC18F4550 is an 8-bit RISC microcontroller with distinct programme and data memory and independent buses.
- Port A, Port B, Port C, Port D, and Port E are the five I/O ports of the PIC18F4550, each with the following registers:

TRISx Register – Used as Data Direct Register

PORTx Register – Used to read the levels on the pins of Port

LATx Register – Used as Data Output Register.

PIC18F4550 CONFIGURATION BITS

- Configuration Bits are a group of special bits used to regulate various characteristics of the microcontroller.
- During reset, configuration bits are read and the values of the bits are used to enable or disable certain hardware functionalities. We can basically control the watchdog timer, clock source, brown-out detect, and memory read protection with these bits.
- Configuration bits aren't code that can be executed. It can only be changed during the programming process, and it can be programmed to select different device configurations.
- CONFIG1L, CONFIG1H, CONFIG2L, CONFIG2H, CONFIG3H, CONFIG4L, CONFIG5L, CONFIG5H, CONFIG6L, CONFIG6H, CONFIG7L, CONFIG7H, DEVID1, and DEVID2 are the 14 configuration registers of the PIC18F4550. The remaining two registers are read-only device ID registers. The remaining 12 registers are utilised to set the PIC18F4550's configuration bits.
- A compiler directive #pragma config can be used to assign configuration bits. The following is an example of how to use CONFIG in the C language:

```
#pragma config OSC = HS, OSCS = OFF
#pragma config BORV = 45, PWRT = ON, BORV = ON
#pragma config WDT = OFF
#pragma config DEBUG = OFF, LVP =OFF, STVR = OFF
```

Table 5.1: Configuration bits for PIC

ADDRESS (Hex)	NAME	GENERAL DESCRIPTION
300000	CONFIG1L	No Prescaler
300001	CONFIG1H	Oscillator Selection
300002	CONFIG2L	Brown Out
300003	CONFIG2H	Watchdog Enable
300005	CONFIG3H	CCP2 input/output
300006	CONFIG4L	ISCP and the Background Debugger
300008	CONFIG5L	Code Security
300009	CONFIG5H	EEPROM and boot block protection
30000A	CONFIG6L	Write protection
30000B	CONFIG6H	Write protection
30000C	CONFIG7L	Read protection
30000D	CONFIG7H	Boot Block read protection
3FFFFE	DEVID1	Device ID and Revision
3FFFFF	DEVID2	Device ID

BASIC MICROCONTROLLER INTERFACING

- Data is transferred between microcontrollers and various interfacing peripherals such as dc motors, relays, keypads, sensors, GSM Modules, and LCD displays using interfacing. Interfacing devices are devices that interface with the PIC18F4550 and are used to execute specific functions.
- PIC18F4550 has a 33 input/output pins. The number of devices connected to that pins depends on the features of the hardware devices.

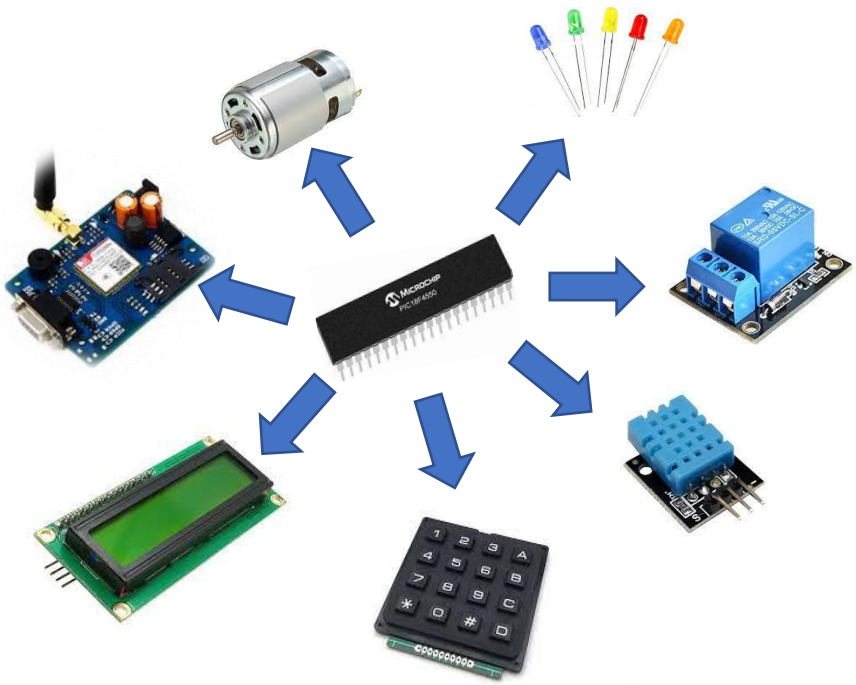


Figure 5.1: Example of input and output devices attached to PIC18F4550

FEATURES OF EXTERNAL DEVICES

KEYPAD

- When the circuit requires it, keypads are connected to the microcontroller for entering quantities or names.
- Matrix keypads are rows and columns that make up this system.

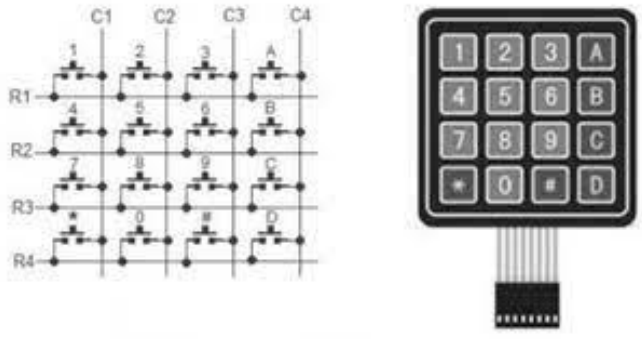


Figure 5.2: 4x4 Matrix Keypad

SWITCH

- Switches are a type of digital inputs and are widely used in electronic projects. Connecting a switch to a microcontroller is straightforward. What we need is a pull-up or pull-down resistor. If there is no resistor, it will be difficult to determine the state of the pin. We call that floating.

Figure 5.3: Switches

LCD

- The LCD (Liquid Crystal Display) is a user-friendly interface that can be extremely valuable for debugging.
- Liquid crystal displays were used to create viewable images.
- LCD displays are super-thin display screens that are found in smart phones, portable video games, laptops, and computer monitors.

Figure 5.4: 16x2 LCD Character Module

RELAY

- A relay is an electromagnetic switch that uses a low-power circuit to switch high voltage or current. Low-power circuits are separated from high-power circuits by this device. It works by electrifying a wrapped coil on a soft iron core. Higher-power devices, such as light bulbs, motors, and solenoids, can be switched with it.

Figure 5.5: 5V DC power relay

I/O PIN OF PIC USED FOR INTERFACING

- PIC18F4550 microcontroller pins for interfacing are 33 pins : Port A,B,C,D,E and its alternate functions.
- Others : 2 pins for Vdd , 2 pins for Vss, 2 pins for OSC, 1 pin for MCLR and 1 pin for USB.

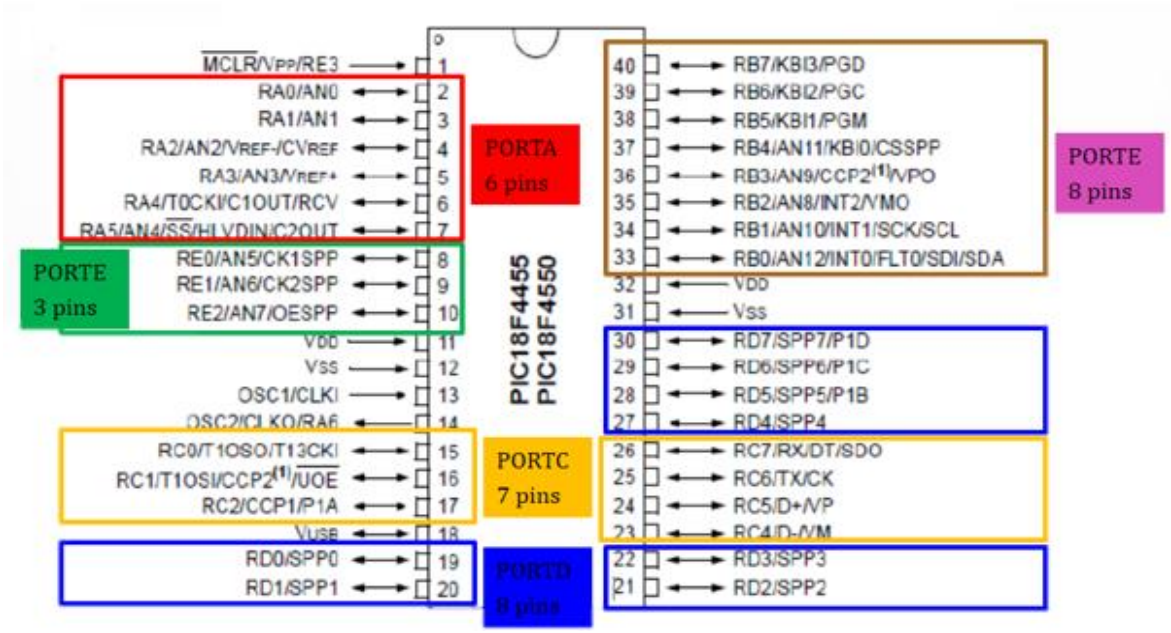


Figure 5.6: PIC18F4550 pin diagram

SIMPLE DIGITAL I/O CIRCUIT : EXAMPLE 1

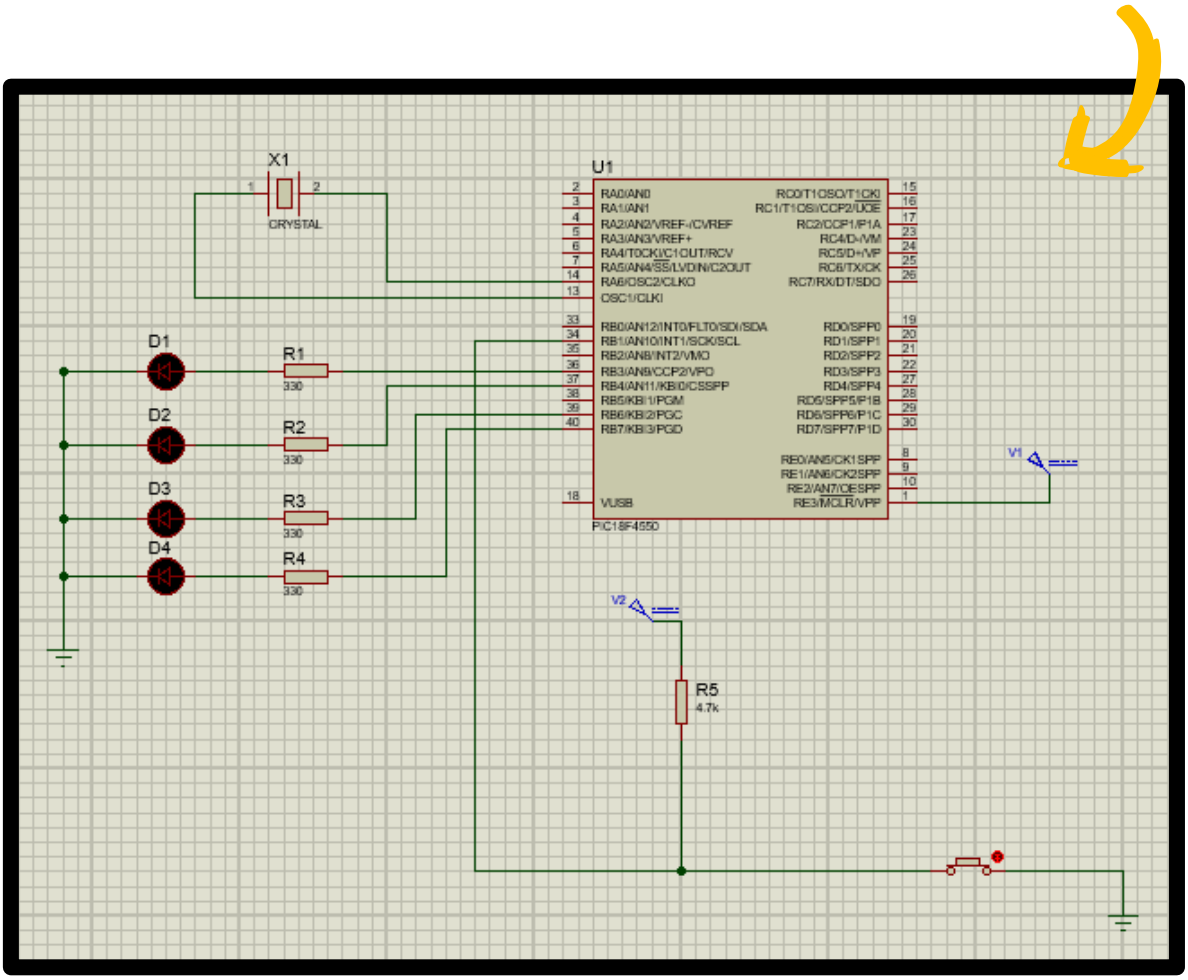


Figure 5.7 : Switch & LED Circuit

```
#include <p18f4550.h>
#define _xtal_freq 20MHz

#pragma config FOSC =HS
#pragma config PWRT=OFF,BOR=ON,BORV=1
#pragma config WDT=OFF
#pragma config DEBUG=OFF,LVP=OFF
#pragma config PBADEN=OFF

void main (void)

{

        TRISBbits.RB1=1;
        TRISBbits.RB3=0;
        TRISBbits.RB4=0;
        TRISBbits.RB6=0;
        TRISBbits.RB7=0;

here:

        if(PORTBbits.RB1==0)

        {
                PORTBbits.RB3=1;
                PORTBbits.RB4=1;
                PORTBbits.RB6=1;
                PORTBbits.RB7=1;

                }

                else
                {
                PORTBbits.RB3=0;
                PORTBbits.RB4=0;
                PORTBbits.RB6=0;
                PORTBbits.RB7=0;

                }
        goto here;

                }
```

Figure 5.8.: Switch and LED C Program

SIMPLE DIGITAL I/O CIRCUIT : EXAMPLE 2

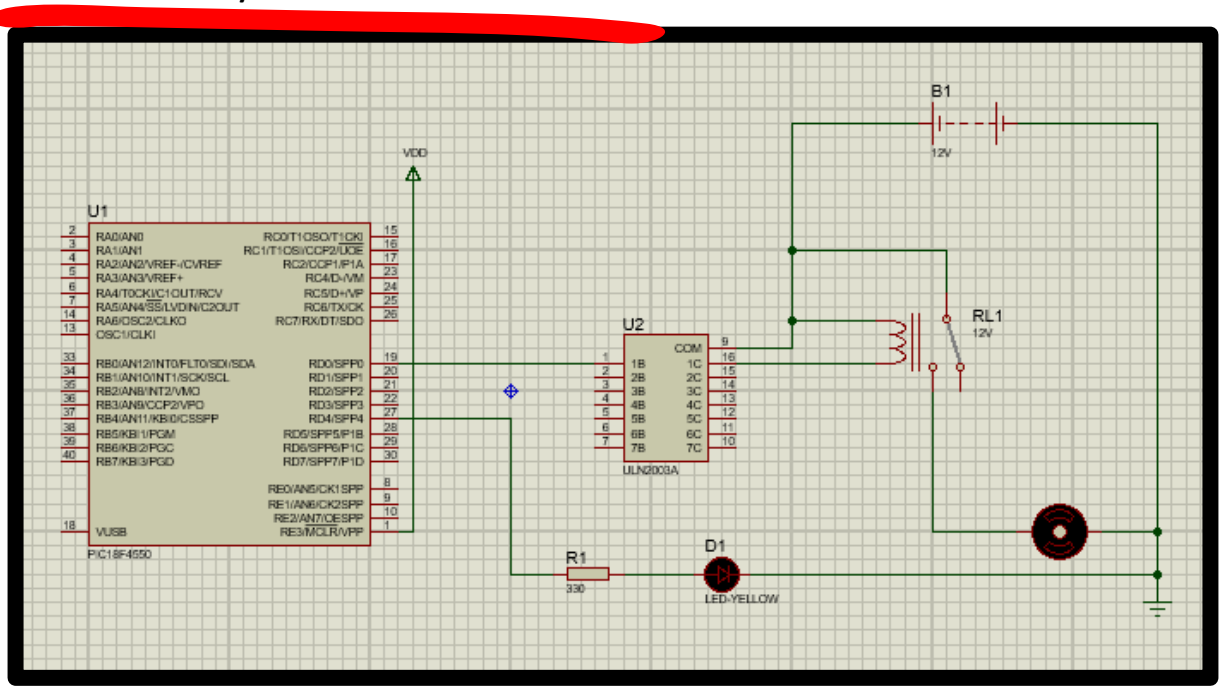


Figure 5.9 : LED and DC Motor Circuit

```
#include <p18f4550.h>
#include<delays.h>
#define RL1         LATDbits.LATD0
#define LED         LATDbits.LATD4



void main(void)
 {
        ADCON1=0X0F;
        CMCON=0X07;
        TRISDbits.RD0=0;
     TRISDbits.RD4=0;

   while (1)

      {
         RL1=1;
                 LED=1;
         Delay10KTCYx(250);
         RL1=0;
         LED=0;
         Delay10KTCYx(250);
        }

}
```

Figure 5.10: LED and DC Motor C Program

ANALOGUE-TO-DIGITAL CONVERTER (ADC) MODULE IN PIC

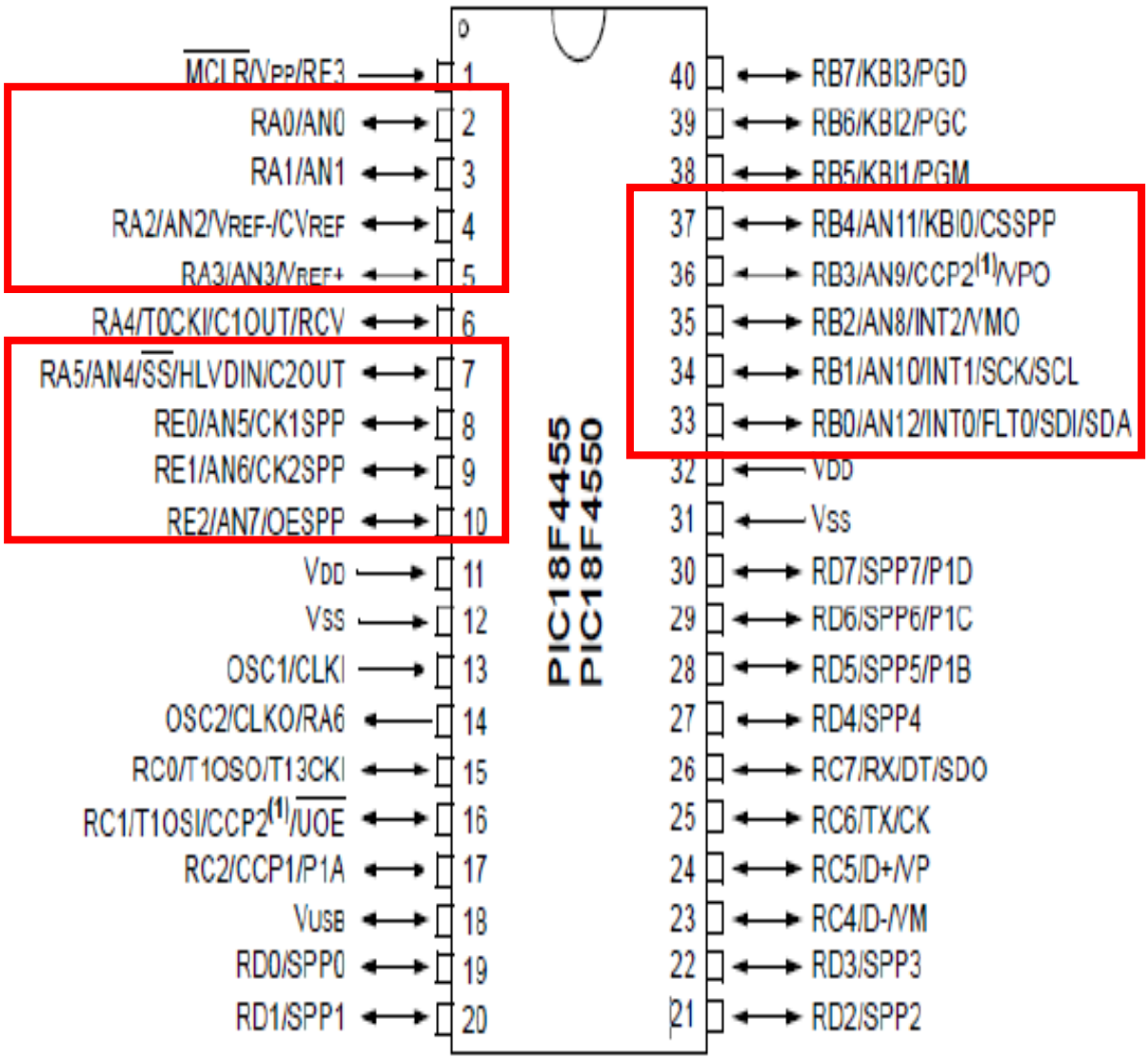


Figure 5.11: ADC pin in PIC18 diagram

ANALOGUE-TO-DIGITAL CONVERTER (ADC) MODULE IN PIC

- Microprocessor-controlled circuits, Arduinos, Raspberry Pis, and other digital logic circuits need Analogue-to-Digital Converters (ADCs) to connect with the outside world.
- Analogue signals in the actual world have constantly changing values from a variety of sources including sensors that monitor sound, light, temperature, and movement. The analogue signals from such transducers are used by many digital systems to interact with their surroundings.
- While analogue signals can be continuous and provide an endless number of voltage values, digital circuits function with binary signals that have only two discrete states: logic "1" (HIGH) or logic "0" (LOW) (LOW). As a result, an electronic circuit capable of converting between the two domains of continuously changing analogue signals and discrete digital signals is required. Analogue-to-Digital Converters (A/D) are used in this situation.
- An analogue to digital converter, in simple terms, takes a snapshot of an analogue voltage at a specific point in time and generates a digital output code that represents that voltage. The amount of binary digits (bits) utilised to represent this analogue voltage value is determined by the A/D converter's resolution.
- A 4-bit ADC, for example, has a resolution of one part in 15, (24 – 1), while an 8-bit ADC has a resolution of one part in 255, (28 – 1). An analogue to digital converter, for example, takes an unknown continuous analogue stream and converts it to a binary number of "n" bits with 2n bits.
- Resolution, Conversion Time, Vref, Digital Data Output, and Analog Input Channel are all important features of an ADC.

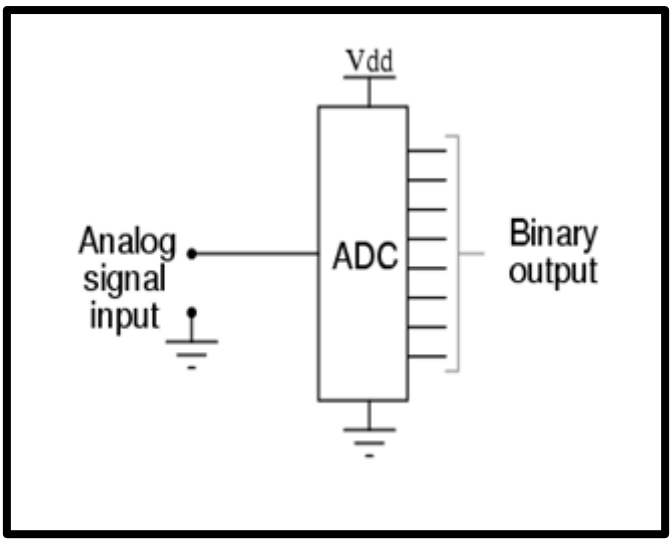


Figure 5.12: Analogue Signal to Binary Output

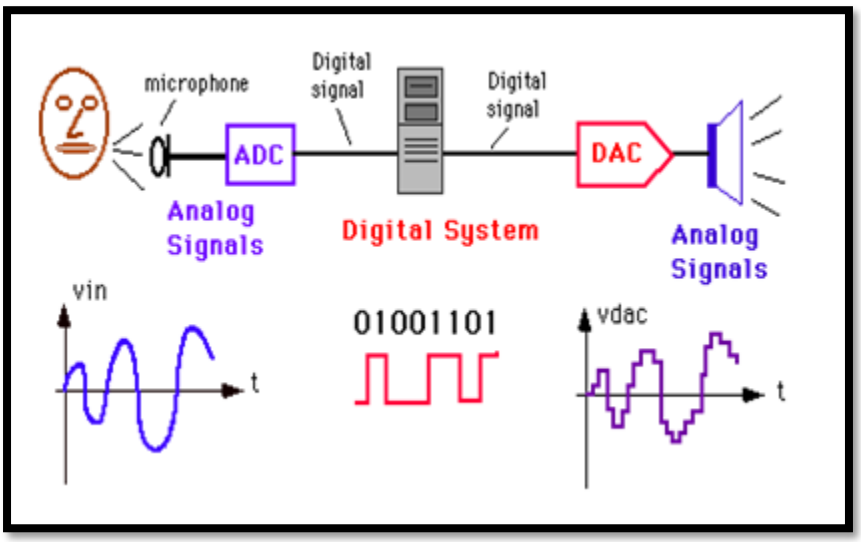
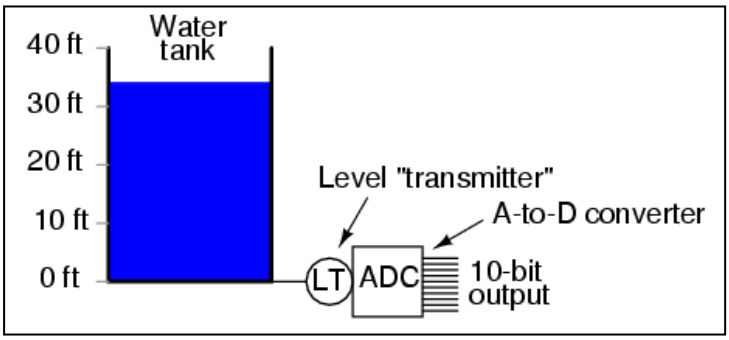


Figure 5.13: ADC Diagram

EXAMPLE 1:

From the information in the figure below, calculate Digital Data, Dout in binary number.



ANSWER :

Number of Step = 2^n

= 2^{10}

= 1024 (0 to 1023)

Step Size = Vref / (number of step – 1)

= 5/1023

= 4.887 mV

D out = Vin / step size

= 5 V / 4.887 mV

= 1023

= $11\ 1111\ 1111_2$

EXAMPLE 2 :

An 8 bit ADC has reference voltage, Vref=4V. If the analogue input is 3.5V, compute the digital value produced by the ADC in hexadecimal.

ANSWER :

Number of Step = 2^n
= 2^8
= 256

Step 1

Step Size = Vref / (number of step – 1)
= 4/255
= 15.69 mA

Step 2

D out = Vin / step size
= 3.5 V / step size
= 3.5 V / 15.69 mV
= 223
= DF_{16}

Step 3

REGISTER IN ADC

- An ADC module in PIC has **five** registers for its operation:
  - A/D Result High Register (ADRESH)
  - A/D Result Low Register (ADRESL)
  - A/D Control Register 0 (ADCON0) → Controls the operation of the A/D module
  - A/D Control Register 1 (ADCON1) → Configures the functions of the port pins
  - A/D Control Register 2 (ADCON2) → Configures the A/D clock source, programmed acquisition time and justification

A/D CONTROL REGISTER ADCON0 AND ADCON1

ADCON0:

Conversion clock		Channel select			Status		Active
BIT7	**BIT6**	**BIT5**	**BIT4**	**BIT3**	**BIT2**	~~**BIT1**~~	**BIT0**
00 - F/2 01 - F/8 10 - F/32 11 - FRC		000 - RA0/AN0 001 - RA1/AN1 010 - RA2/AN2 011 - RA3/AN3 100 - RA4/AN4 101 - RA5/AN5 110 - RA6/AN6 111 - RA7/AN7			0 - End 1 - Go A/D	not used	0 - A/D off 1 - A/D on

ADCON1:

BIT7	~~BIT6~~	~~BIT5~~	~~BIT4~~	BIT3	BIT2	BIT1	BIT0
RESULT formats	not used			A/D Port Configuration bits			

ADCON 0 - A/D CONTROL 0

Table5 .2: ADCON0 Register

To set conversion time (f_{osc} – crystal oscll. Frequency)

To select analogue input channel

To indicate the time to start /complete conversion

ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DONE	--	ADON

ADCS2 (from ADCON1)	ADCS1	ADCS0	Conversion Clock Source
0	0	0	Fosc/2
0	0	1	Fosc/8
0	1	0	Fosc/32
0	1	1	Internal RC used for clock source
1	0	0	Fosc/4
1	0	1	Fosc/16
1	1	0	Fosc/64
1	1	1	Internal RC used for clock source

CHS2	CHS1	CHS0	CHANNEL SELECTION	
0	0	0	CHAN0 (AN0)	
0	0	1	CHAN1 (AN1)	
0	1	0	CHAN2 (AN2)	
0	1	1	CHAN3 (AN3)	
1	0	0	CHAN4 (AN4)	
1	0	1	CHAN5 (AN5)	not implemented on 28-pin PIC18
1	1	0	CHAN6 (AN6)	not implemented on 28-pin PIC18
1	1	1	CHAN7 (AN7)	not implemented on 28-pin PIC18

GO/DONE A/D conversion status bit.
1 = A/D conversion is in progress. This is used as start conversion, which means after the conversion is complete, it will go LOW to indicate the end-of-conversion.
0 = A/D conversion is complete and digital data is available in registers ADRESH and ADRESL.

ADON A/D on bit
0 = A/D part of the PIC18 is off and consumes no power. This is the default and we should leave it off for applications in which ADC is not used.
1 = A/D feature is powered up.

- The second register, ADCON1, must be set for the following reasons: to pick the result value format (bit 7), to select the reference voltage (bit0...bit3), and to set the port configuration control bits according to the following table.

Table 5.2 : Port Configuration Control Bits for ADCON1

PCFG3: PCFG0	AN7(1) RE2	AN6(1) RE1	AN5(1) RE0	AN4 RA5	AN3 RA3	AN2 RA2	AN1 RA1	AN0 RA0	VREF+	VREF-	CHAN/ Refs(2)
0000	A	A	A	A	A	A	A	A	VDD	VSS	8/0
0001	A	A	A	A	VREF+	A	A	A	RA3	VSS	7/1
0010	D	D	D	A	A	A	A	A	VDD	VSS	5/0
0011	D	D	D	A	VREF+	A	A	A	RA3	VSS	4/1
0100	D	D	D	D	A	D	A	A	VDD	VSS	3/0
0101	D	D	D	D	VREF+	D	A	A	RA3	VSS	2/1
011x	D	D	D	D	D	D	D	D	VDD	VSS	0/0
1000	A	A	A	A	VREF+	VREF-	A	A	RA3	RA2	6/2
1001	D	D	A	A	A	A	A	A	VDD	VSS	6/0
1010	D	D	A	A	VREF+	A	A	A	RA3	VSS	5/1
1011	D	D	A	A	VREF+	VREF-	A	A	RA3	RA2	4/2
1100	D	D	D	A	VREF+	VREF-	A	A	RA3	RA2	3/2
1101	D	D	D	D	VREF+	VREF-	A	A	RA3	RA2	2/2
1110	D	D	D	D	D	D	D	A	VDD	VSS	1/0
1111	D	D	D	D	VREF+	VREF-	D	A	RA3	RA2	1/2

Example: Programme in C

This application retrieves data from ADC's channel 0 (RA0) and displays it on PORTC and PORTD. Every quarter of a second, this is done.

```
void main (void)
 {
    TRISC=0;
    TRISD=0;
    TRISAbits.TRISA0=0;       //RA0 =INPUT for analog input
    ADCON0 =0x81;             //Fosc/64, channel 0, A/D is on
    ADCON1 =0xCE;             //right justified, Fosc/64,
                              //AN0 = analog
    while(1)
    {
     DELAY(1) ;               //give A/D channel time to sample
     ADCON0bits.GO = 1;       //start converting
     while (ADCON0bits.DONE == 1);
     PORTC=ADRESL;            //display low byte on PORTC
     PORTD=ADRESH;            //display high byte on PORTD
     DELAY (250);             //wait for one quarter of a
                              //second before trying again
    }
}
```

PULSE WIDTH MODULATION (PWM) IN PIC

- PWM waves can be easily generated using CCP (*CCP stands for Capture / Compare / PWM)* module (pin 16 & pin 17 of PIC18F4550).
- Although Timers can be used to create PWM, the CCP module makes the programming easier .
- PWM features allow us to create pulse with variable width.
- Two factors in creating variable widths:
    - Period of pulse
    - Pulse duty cycle

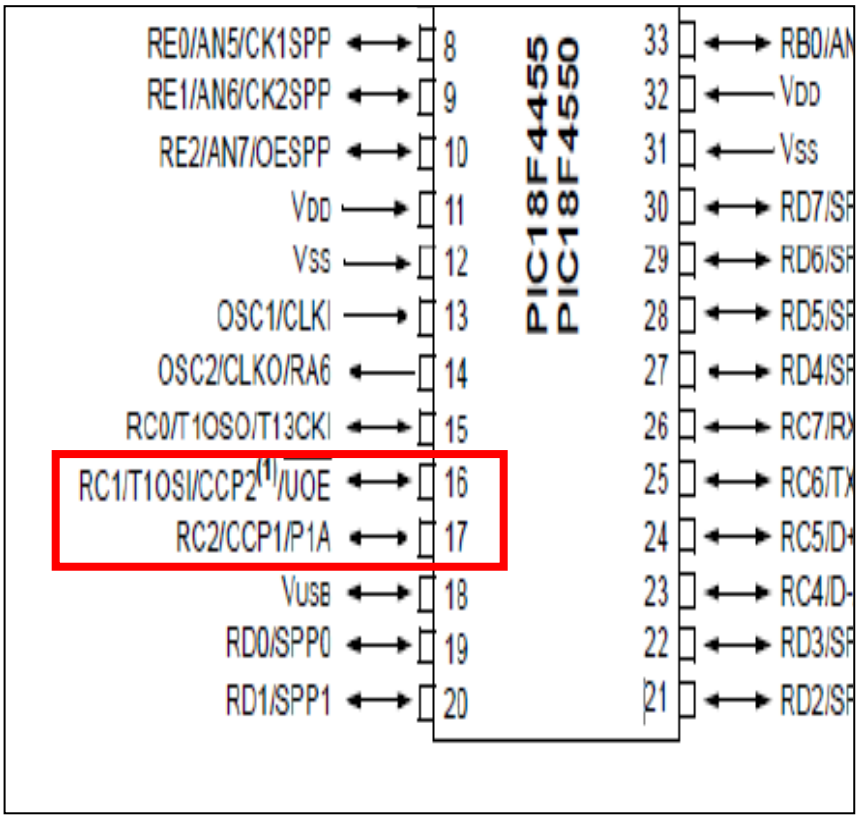
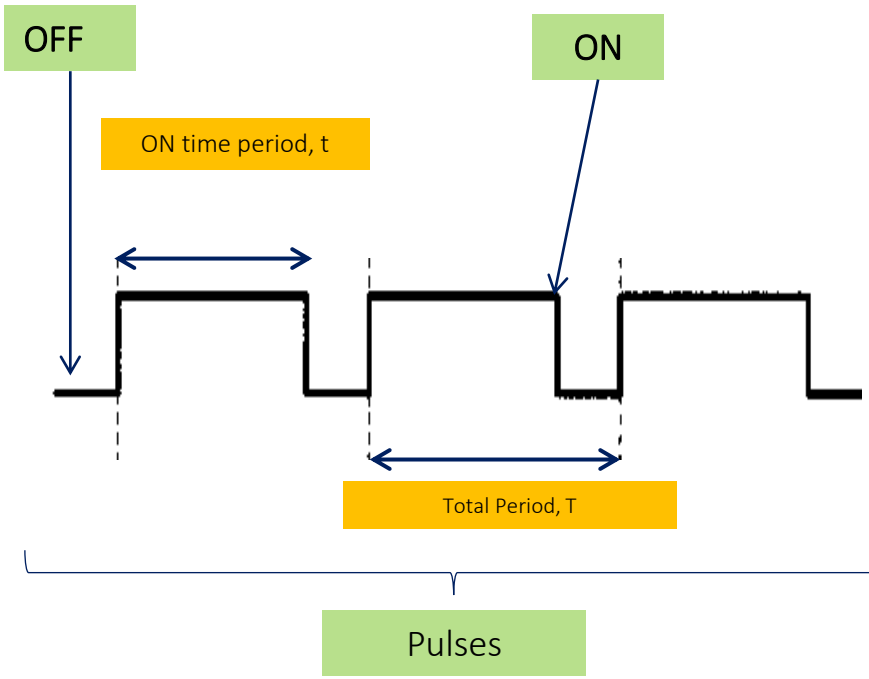


Figure 5.14: PWM pin in PIC18 diagram

PULSE WIDTH MODULATION (PWM) - PULSE

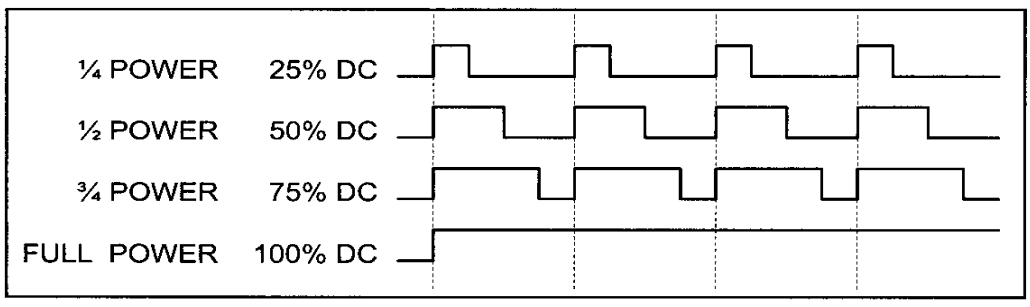
- A method for generating analogue output signals from digital signals.



PWM signals are **ON – OFF signals** (HIGH or LOW) (Pulse)

HIGH or **ON duration** is changed (Width) in accordance with our requirements.

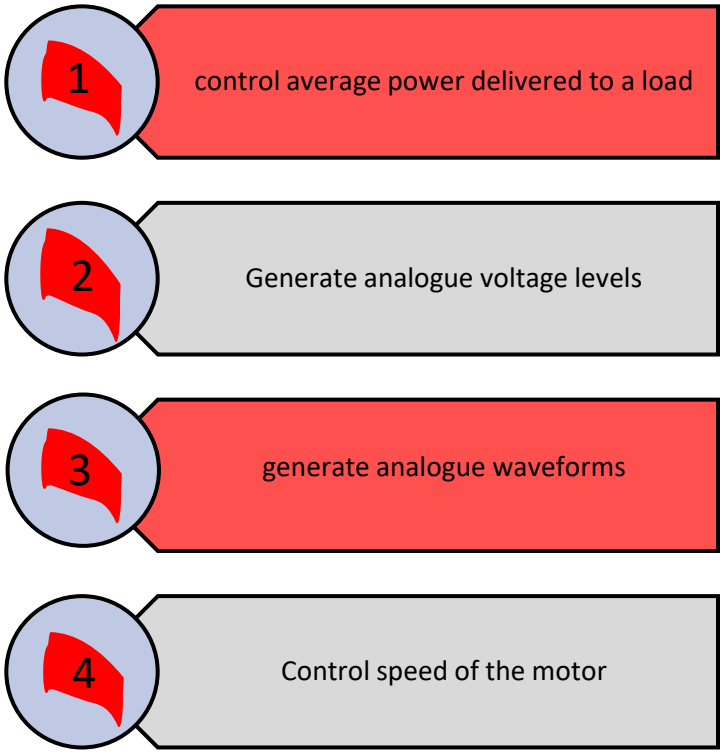
Time period for which the signal is ON to the total time period is termed as Duty Cycle



Pulses with different duty cycle

PWM : Applications

- Common applications that use PWM technology, including :



The quantity of power delivered to a load can be increased or decreased by changing (modulating) the pulse width.

In the case of DC motor, the changing will increase or decrease the motor speed. The wider the pulse, the higher the speed

PWM pulse has a fixed amplitude but it has a variable duty cycle

STEPS IN PROGRAMMING PWM

1. •Set the PWM period by writing to PR2
2. •Set the PWM duty cycle by writing to CCPRxL for higher bits
3. •Set the CCP pin as an output
4. •Using the T2CON register, set the prescale value
5. •Clear the TMR2 register
6. •Set the DCxB1:DCxB0 bits for the decimal component of the duty cycle in the CCPxCON register for PWM.

GENERATING PWM

Period of PWM, T_{PWM}

Uses:

- Timer2 register
- PR2 register

$$Tpwm = [(PR2) + 1\ ]\ 4 \times N \times Tosc$$

$$PR2 = [Fosc / (Fpwm \times\ 4 \times N)] - 1$$

Notes:

- PR2 (period register) – register to set PWM period
- $T_{OSC} = 1 / F_{OSC}$
- N - prescaler (1,4,16); set by Timer2 Control Register (T2CON)

Duty Cycle of PWM, D.C

Uses 10-bit register of DC1B9:DC1B0 from:

- CCPRxL register (upper 8 bits) – main duty cycle
- CCPxCON <5:4> register (lower 2 bits) – decimal point

DC1B2	DC1B1	Decimal Points
0	0	0
0	1	0.25
1	0	0.5
1	1	0.75

EXAMPLE 3:

- Find the PR2 value and the Prescaler needed to get the following PWM frequencies.
- Assume XTAL = 20MHz
- a) 1.22KHz
- b) 4.88KHz
- c) 78.125KHz

ANSWER :

a) PR2 = [20MHz / (4 X 1.22KHz)]-1
= 4097 (larger than 255 which is maximum value for PR2).

Now choosing the prescaler of 16 :
PR2 = [20MHz / (4 X 1.22KHz X 16)]-1
= 255

b) PR2 = [20MHz / (4 X 4.88KHz)]-1
= 1023 (larger than 255 which is maximum value for PR2).

Now choosing the prescaler of 4
PR2 = [20MHz / (4 X 4.88KHz X 4)]-1
= 255

c) PR2 = [20MHz / (4 X 78.125KHz)]-1
= 63

EXAMPLE 4:

- Find the value of registers PR2, CCP1RL and DC1B2:DC1B1 for the PWM frequencies value below if we want a 75% duty cycle. Assume XTAL = 10MHz
- a) 1KHz b) 2.5KHz

- ANSWER:

a) Using the equation

PR2 = [FOSC / (4 X FPWM X N)]-1

N=16 (prescaler)

PR2 = [10MHz / (4 X 1KHz X 16)]-1

= 156 – 1

= 155

CCPR1L = 155 X 75%

= 116.25

~ 116

DC1B2:DC1B1 = 01 for the 0.25 portion

b) Using the equation

PR2 = [FOSC / (4 X FPWM X N)]-1

N=4 (prescaler)

PR2 = [10MHz / (4 X 2.5KHz X 4)]-1

= 250 – 1

= 249

CCPR1L = 249 X 75%

= 186.75

~ 186

DC1B2:DC1B1 = 11 for the 0.75 portion

Example: Programme in C

```
CCP1CON=0;                    //clear CCP1CON reg
PR2=249;
CCPR1L=186;                   //75% duty cycle
TRISCbits.TRISC2=0;           //make PWM pin an output
T2CON=0x01                    //Timer2, 4 prescale, no postscaler
CCP1CON=0x3C;                 //PWM mode, 11 for DC1B1:BO
TMR2=0;                       //clear Timer2
T2CONbits.TMR2ON=1;           //turn on Timer2
while(1)
{
  PIR1bits.TMR2IF=0;          //clear Timer2 flag
  while(PIR1bits.TMR2IF==0);  //wait for end of period
}
```

UNIVERSAL SYNCHRONOUS ASYNCHRONOUS RECEIVER TRANSMITTER (USART) IN PIC

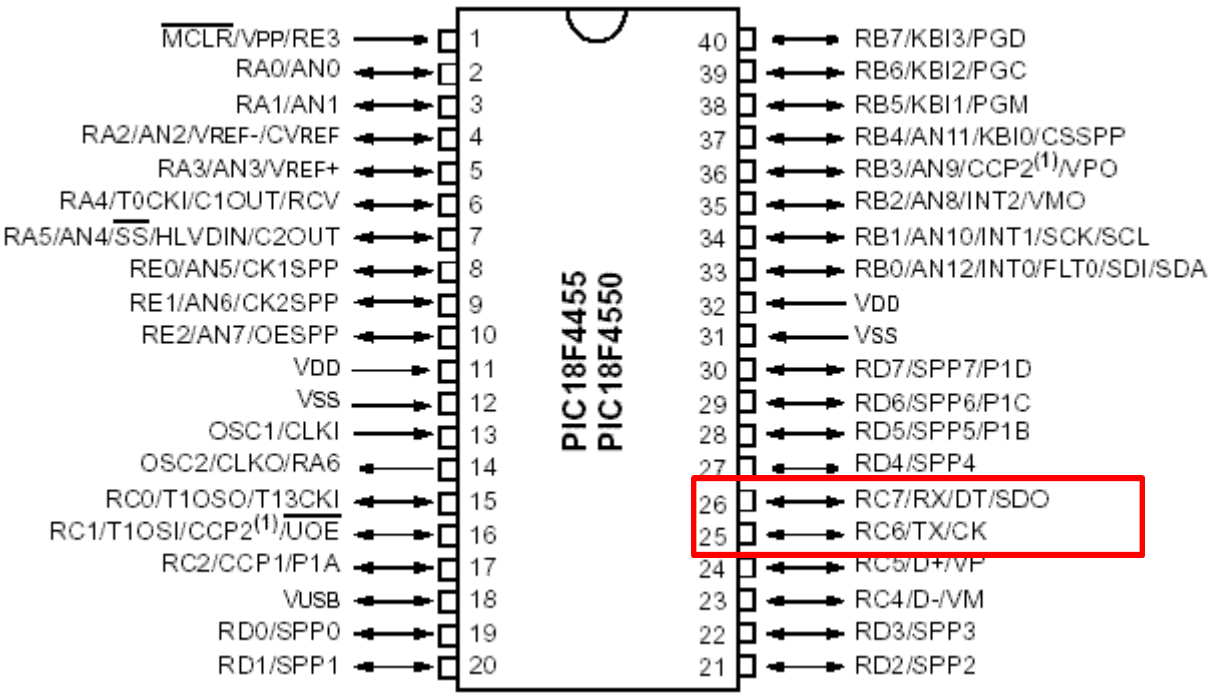


Figure 5.15: USART pin in PIC18 diagram

USART IN PIC

- PIC18 has USART (universal synchronous asynchronous receiver transmitter) features.
- It allows receiving (RX) and transfering (TX) data using synchronous and asynchronous modes.

Six Major Register associated with UART	
1	Serial Port Baud Rate Generator (SPBRG)
2	Transfer Register (TxREG)
3	Receive Register (RCREG)
4	Transmit Status and Control Register (TXSTA)
5	Receive Status and Control Register (RCSTA)
6	Peripheral Interrupt Request Register 1 (PIR1)

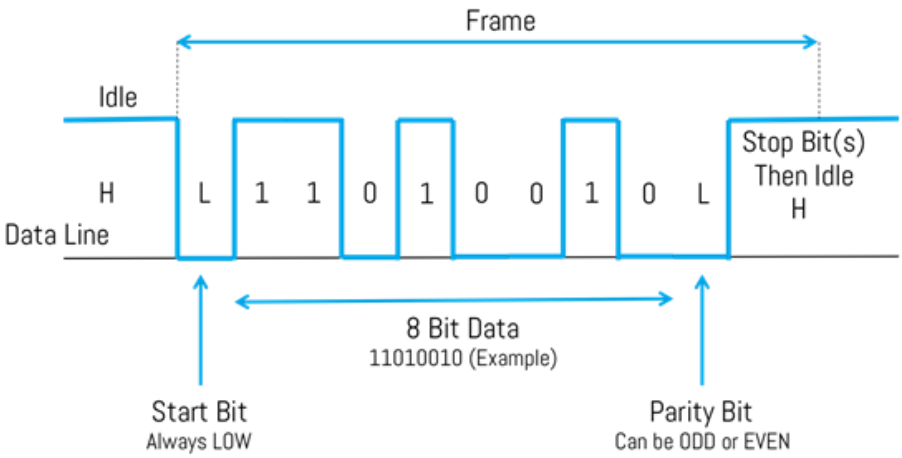


Figure 5.16 : Asynchronous Serial Communication : Data Framing

EXAMPLE OF USART CIRCUIT USING PIC

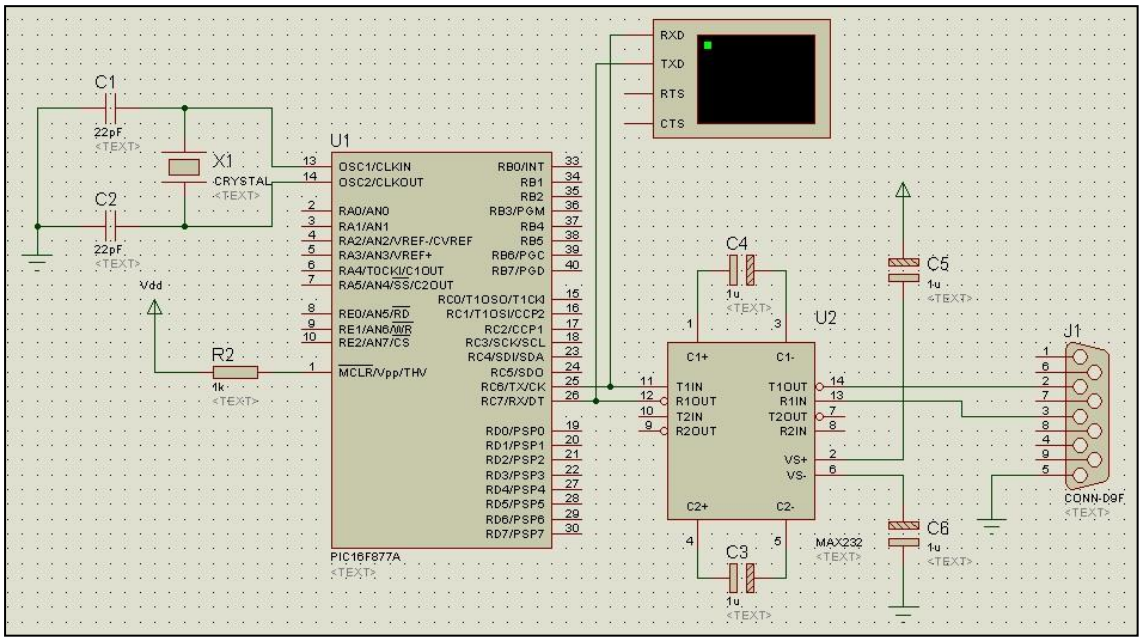


Figure 5.17 : USART CIRCUIT

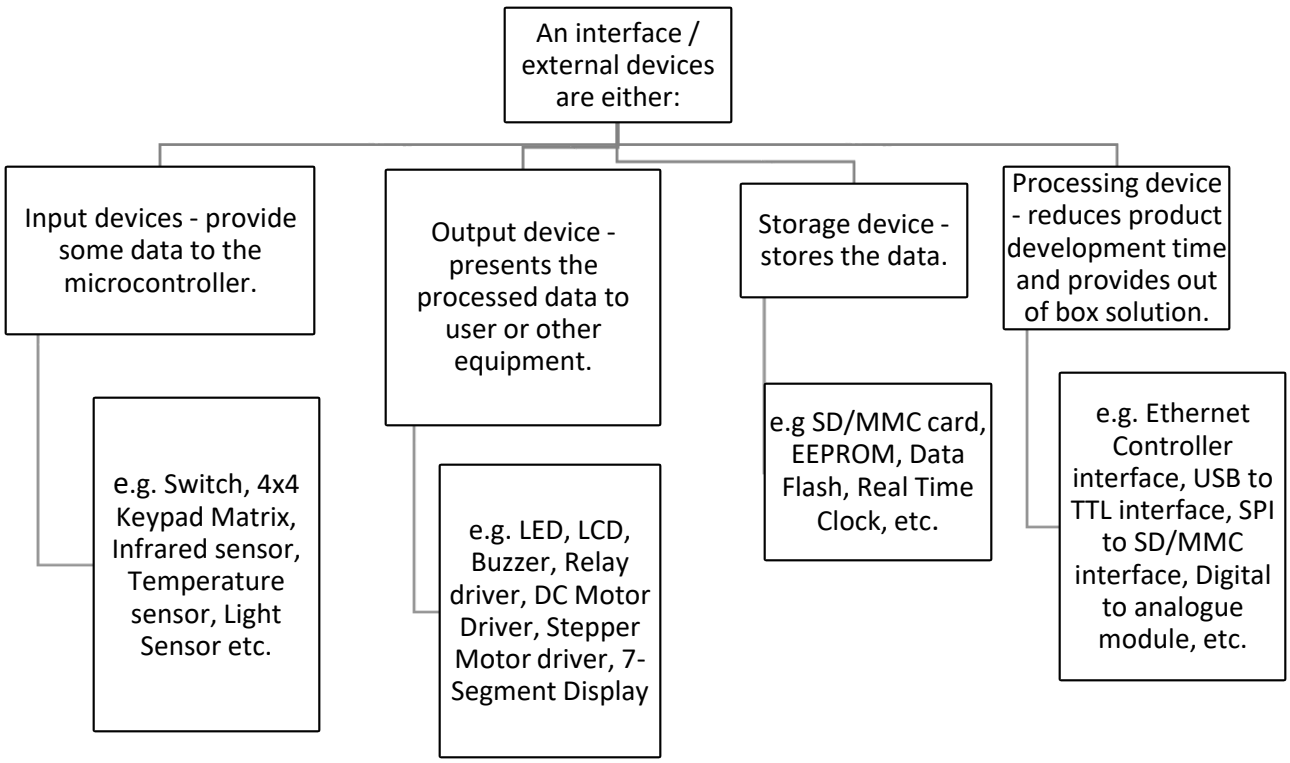
EXAMPLE: PROGRAMME IN C

Write a C program to receive bytes of data serially and put them on PORTB. Set the baud rate at 9600, 8 bit data 1 stop bit.

```
#include <p18f4580.h>          //Select PIC18F4580

void main (void)               //Main program
{
    TRISB = 0;                 //PortB as output
    RCSTAbits.SPEN = 1;        //Enable serial port
    RCSTAbits.CREN = 1;        //Enable continuous receive
    SPBRG = 0x0F;              //9600bps, XTA = 10MHz

    while (1)                  //Repeat forever
    {

wait:
    if(PIR1bits.RCIF == 0)     //Wait to receive data
    {   goto wait;}

    PORTB = RCREG;             //save value in PORTB
    }
}
```

HARDWARE INTERFACING

- The term "interfacing" refers to the process of connecting a microcontroller to an interface or external device.



- Several hardware devices that are able to be interfaced with PIC microcontroller by using available internal features, are listed in the table below:

PIC Internal Features	Interface External Devices
I/O Pin	Keypad
ADC	Analogue Sensor
PWM	Voltage & Current Control
USART	LCD

REAL TIME EMBEDDED SYSTEM APPLICATIONS

- Real-time systems are computer systems that monitor, respond to, or control an external environment.
- This environment is connected to the computer system through **sensors**, **actuators**, and other input-output interfaces.
- The computer system must meet various timing and other constraints that are imposed on it by the real-time behavior of the external world to which it is interfaced.
- Often humans are part of the connected external world, but a wide range of other natural and artificial objects and animals are also possible.

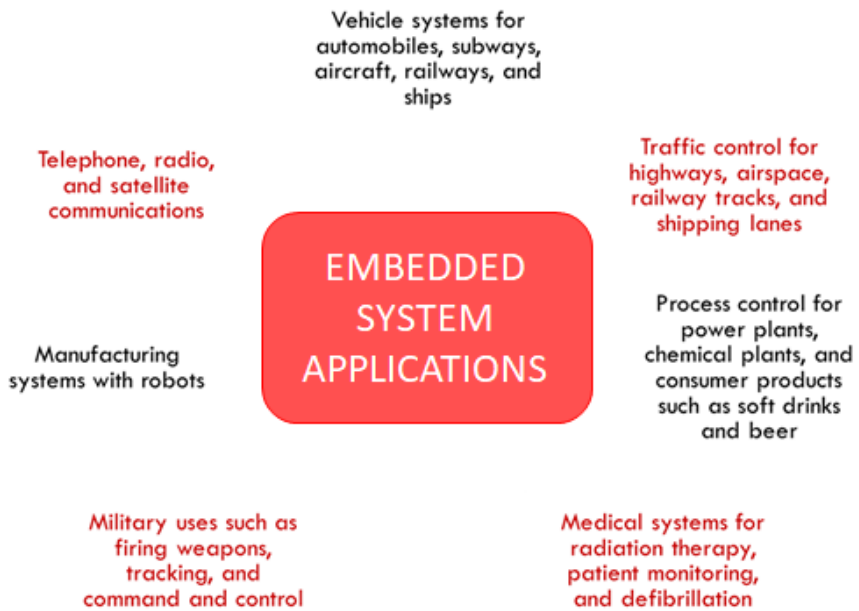


Figure 5.18 : Real Time Embedded System Applications

EXERCISE CHAPTER 5

1. An 8bit ADC has a reference voltage of, Vref =5 V. If the analogue input is 3.0 V, calculate the digital value produced by the ADC in hexadecimal.

Answer:

Number of steps = 2^8 = 256

Step size = Vref / number of steps

= 5V / 256

= 19.53mV per steps

Dout = Vin / Step Size = 3 / 19.53mV

Dout = $154_{10} = 99_{16} = 10011001_2$

2. Calculate the values of registers PR2, CCP1RL and DC1B2:DC1B1 bits for the PWM frequency of 1.2 KHz with duty cycle of 25%. Assume XTAL= 10MHz.

Answer:

Using PR2=Fosc/ (4 x Fpwm x N) - 1

PR2= [(10MHz / 4 x 1.2KHz x 4)] – 1

= 520.8 -1

= 519 and because 25% duty cycle

= 520 x 25%

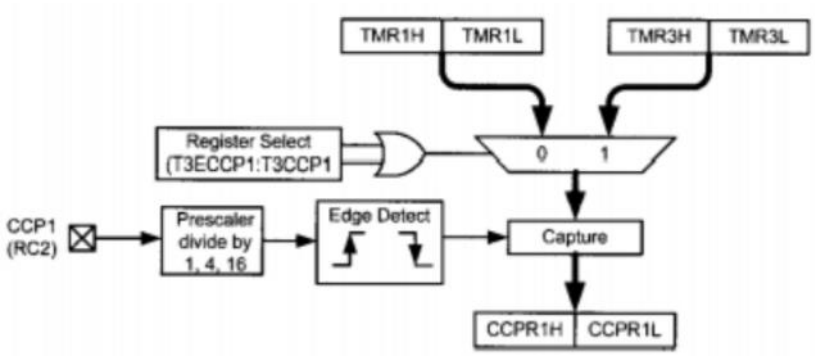
= 129.75

We have CCP1RL=130 and

DC1B2:DC1B1 = 11 for the 0.75 portion

EXERCISE CHAPTER 5

3. PIC18F4550 has an in-built CCP (Capture/Compare/PWM) module with input Capture Mode. Analyze a Capture Mode block diagram in Figure A3(b) below:



Answer:

An event at the CCP pin causes the contents of the Timer1 or Timer 3 register to be loaded into the 16-bit CCPR1H:CCPR1L register, as shown in figure A3(b).For the capture mode to work, the CCP pin must be specified as an input pin.A HIGH to LOW (falling edge) pulse or a LOW to HIGH (rising edge) pulse can cause the contents of Timer1 or Timer3 to be captured in the CCPR1H:CCPR1L register.The PIC18 event for the edge triggering pulse can be one of the following:

- Every falling edge pulse
- Every rising edge pulse
- Every 4th rising edge pulse
- Every 16th rising edge pulse

REFERENCES

Alokshiya .(2014). *PIC18F4550 Interrupts*. http://site.iugaza.edu.ps/mokshiya/files/2014/12/Embedded_Lab8_Interrupts.pdf

Bhat & Ghosh .(2011). *PIC18F4550- A General Overview.* https://www.ee.iitb.ac.in/~wel_iitb/qip2019-uC-docs/Day-1_June17-Contents/pic18_overview.pdf

Farahmand (2019).*Microprocessors and System Design-Using PIC Microcontroller.* https://www.sonoma.edu/users/f/farahman/sonoma/courses/es310/lectures/chapter10_interrupt.pdf

Khan (2013). *Using PIC Timer Module for Delay-C Code with Proteus Simulation*. http://embeddede-lab.blogspot.com/2013/09/using-pic18-timer-module-for-delay-c.html

Labcenter Electronics (2021). https://www.labcenter.com/simulation/

Mazidi, Mckinlay & Causay.(2008). *PIC Microcontroller and Embedded Systems Using Assembly and C for PIC18.* New Jersey *:* Pearson Education

Open Lab pro (2012-2019). Analog to Digital Converter Using PIC18F4550. https://openlabpro.com/guide/analog-to-digital-converter-using-pic18f4550/

PIC Microcontroller programming in C using Mikroc Pro for PIC. https://microcontrollerslab.com/pic-microcontroller-programming-c/

Raj (2017). Understanding Timers in PIC Microcontroller with LED Blinking Sequence. https://circuitdigest.com/microcontroller-projects/pic-microcontroller-timer-tutorial