MAJOR PROJECT REPORT

on

AMBULANT FIRE TERMINATOR

Submitted by

ASWATHY A(13143062)

CHITRA SHERINE JOHN (13143067)

DEVIPRIYA P J (13143068)

LAYA HARWIN(13143079)

In partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

In

ELECTRONICS & COMMUNICATION ENGINEERING

MARCH 2017

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

Toc H INSTITUTE OF SCIENCE & TECHNOLOGY

Arakkunnam P.O, Ernakulam District, KERALA – 682 313

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

CERTIFICATE

This is to certify that major project entitled “**AMBULANT FIRE TERMINATOR”** is a bonafide work carried out in the seventh semester by “**ASWATHY.A, CHITRA SHERINE JOHN, DEVIPRIYA P.J, LAYA HARWIN”** in partial fulfillment for the award of Bachelor of Technology in “Electronics & Communication Engineering” from Cochin University of Science and Technology during the academic year 2016-2017

Asst.Prof. Anu Jose
Project Guide

Assoc. Prof. Deepa Elizabeth George
Head of the Department

Prof. Dr. Preethi Thekkath
Head of the Institution

ACKNOWLEDGEMENT

To the grace and generous blessing of **God Almighty,** we attribute the successful completion of our project. It is our duty to respectfully offer our sincere gratitude to all the people who have kindly offer their valuable support, guidance and support.

We would like to extend our heartiest thanks to the **Managemen**t of our college, who provided us with necessities for the completion of the project.

We feel highly privileged in making a mention of**Prof. Dr. Preethi Thekkath (Principal, TIST) and Prof. Dr. Ravindran Nair (DEAN, Academics & Students Affairs)**for the inspiration inculcated in us and for the apt guidance.

We deeply and wholeheartedly thank **Assoc. Prof. Deepa Elizabeth George (HOD, ECE)** for her extreme valuable advice and encouragement.

It would be a grave error if we forget to take a mention of our project co-ordinator**Asst. Prof. Neethu George and Asst. Prof. Deepthi Thomas(ECE)** and our project guide **Asst. Prof. AnuJose** whose constant persistence and support helped us in the completion of our project.

Before we culminate we would like to extend our heartfelt gratitude to all the teachers and staff of Department of Electronics and Communication Engineering, TIST for their co-operation and support.

Last but not the least; we thank all others and especially our family members and our classmates who in some way or other helped us in successful completion of this work.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

GSM	Global System for mobiles
HTML	Hyper Text Markup Language
ICSP	In-Circuit Serial Programming
IDE	Integrated Development Environment
IoT	Internet of Things
LCD	Liquid Crystal Display
LDR	Light Dependent Resistor
LED	Light emitting diode
MEMS	Micro Electro Mechanical Systems
PIC	Peripheral Interface Controller

1. INTRODUCTION

In today’s age, robotic has the fundamental key for new invention. The development of human-machine communications on an everyday basis has made the people to utilize the technology. This project, which is our endeavor to design a Fire Fighting Robot, comprises of a machine which not only has the basic features of a robot, but also has the ability to detect fire and extinguish it. This robot processes information from its various sensors and key hardware elements through microcontroller. It uses temperature sensor and LDR to detect fire accident. A robot capable of extinguishing a simulated tunnel fire, industry fire and military applications are designed and built. Temperature sensor and LDR will be used for the initial detection of the fire. Once the fire is detected, a message will be sent to the registered mobile number and the robot can be controlled using the web application.

The project helps to generate interests as well as innovations in the field of robotics while working towards a practical and obtainable solution to save lives and mitigate the risk of property damage. Fire fighters face risky situations when extinguishing fire and rescuing victims, it is an inevitable part of being a fire fighter. In contrast, a robot can function by itself or be controlled from a distance, which means that firefighting and rescue activities could be executed without putting fire fighters at risk by using robot technology instead. In other words, robots decrease the need for fire fighters to get into dangerous situations.

This robot provides fire protection when there is a fire in a tunnel or in an industry or in schools, homes, and hospitals by the use of a microcontroller in order to reduce the loss of life and property damage. This robot uses dc motors, castor wheel, microcontroller, sensors and foam bottle. Microcontroller is the heart of the project. It controls all the parts if the robot by the help of programming. In this robot as the fire is detected, it sends the signal to the microcontroller. The microcontroller in turn uploads the data to the net via the dot net software and a warning message of fire being detected is sent to the registered mobile number. The user can control the movement of the robot from anywhere using the web application that has been developed.

The microcontroller actuates the driver circuit and it drives the robot towards the fire place, as the robot reaches near the fire, the microcontroller actuates the motor and the foam is allowed to fall on the fire and extinguish the fire. The Internet of Things (IoT) , the technologies, architecture and service that allow massive number if sensor enabled, uniquely addressable things to communicate with each other and transfer data over pervasive networks using Internet protocols, is expected to be the next great technological innovation and business opportunity. In this project, IoT serves the purpose of communication between transmitter and receiver module.

2. LITERATURE REVIEW

Robots are playing a vital role in today's industrial automation and monitoring system. As technology developed these robots have increased their applications and functionality. Working robots will cooperate to the people makes the work more Effortless and uncomplicated.

The fire detection system using four flame sensors in the Fire Sensing and Extinguishing Robot, and program forfire detection and extinguishing procedure using sensor based method. The fire fighting robot is equipped with four thermistors / flamesensors that continuously monitor the temperature.[1] If the temperature increases beyond the predetermined threshold value, buzzersounds to intimate the occurrence of fire accident A warning message will be sent to the respective personnel in the industry and tonearby fire station with the GSM module provided to it. Fire Sensing and Extinguishing Robot continuously monitors the temperature by the sensors and if fire accident is true, the robot moves to the direction to which the temperature is recorded to be the relativelymaximum from the sensors and extinguishes the fire with water pump provided to it.[2] After extinguishing the fire Robot comesback to its initial position.

A fast and simple algorithm proposed for hand gesture recognition for controlling robot using IOT and wireless network. The system of gesture controlled robots, only a limited number of gestures were considered . [3] It provides 4 different gestures for controlling the robots, i.e., forward, backward, left, right. For cutting weeds a gripper concept using buttons is anticipated. These movements are given by the user using MEMS Sensor. The MEMS Sensor will be set to the hand. Whenever the hand moves in some direction, the mechanical movement of the hand will be recognized by MEMS. MEMS translate this mechanical hand movement into equivalent electrical signals and send it to the Raspberry Pi.

The Raspberry Pi at the transmitter side sends control signals to the receiver side through IOT (Internet of Things).[4] The controller (ARM7) at the receiver area receives these signals and gives direction to the robot through IOT i.e. through cloud.

An internet based system entitled Eye Blink and head movement Monitoring System‘ which will help drivers to alert in drowsiness. [5]This system is based on principle of monitoring eye movements of driver continuously using an IR sensor and head movement using accelerometer. If he/she falls asleep, then an alarm will ring to wake him/her up.[6] EBM (Eye Blink Monitoring Technique) to detect drowsiness of night drivers and preventing accidents.

3.PROBLEM DEFINITION

One of the most important problems in the world today is the fire extinguishing problem. This problem results from the increase of demand for human energy and extinguishing a big fire. Nowadays, fire cases are increasing year by year. There are a lot of ways to fight against fire, but it still does not ensure the safety of people. Fire fighting is the attempt to minimize injury or loss of life, or damage to property, caused by a fire, either by extinguishing the fire or by reducing the speed of its containment. The aim of this project is to develop a microcontroller based fire fighting robot which uses the IoT platform to monitor the occurrence of fire and extinguish it. It uses LM35 and LDR sensor for detection. The robot has a fan and a CO2 bottle to extinguish fire; it is controlled through wireless communication. For the desired operation, PIC microcontroller is used.

4. PROJECT ANALYSIS AND DESIGN

4.1 BLOCK DIAGRAM

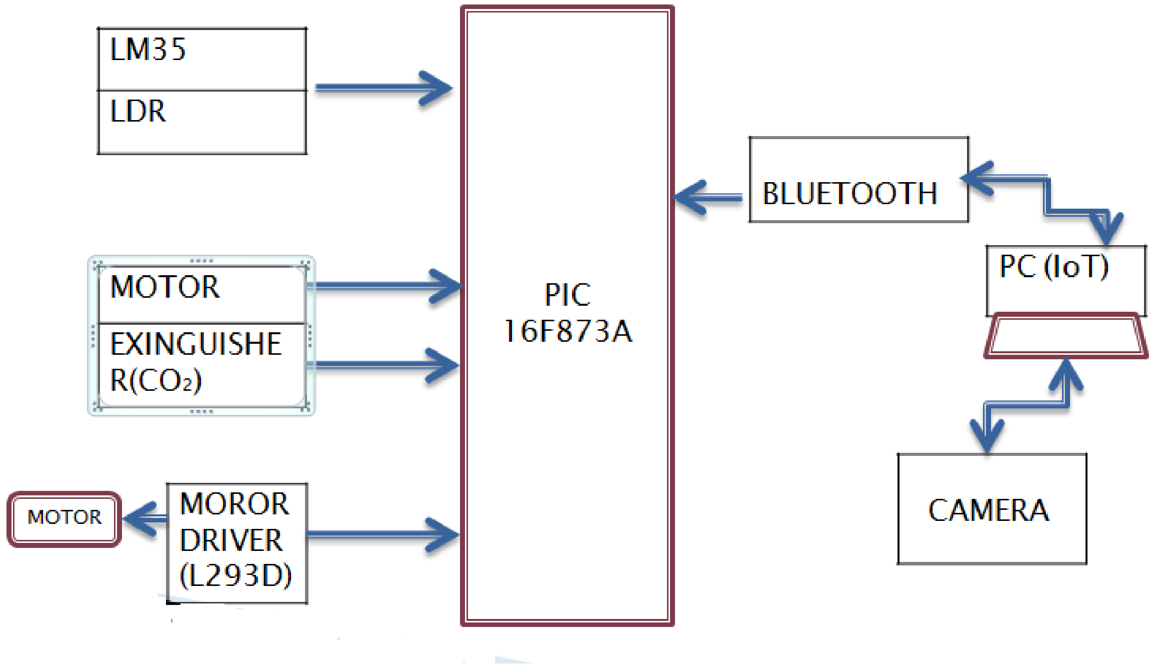


Fig 4.1 Block Diagram

4.1.1 EXPLANATION

The heart of this project is PIC16F873A.All other components are connected to the microcontroller. As the main aim of the project is to detect the fire, alert and then extinguish fire ,each block defines each of these function.LM35 and LDR represents the temperature sensor and the light dependant resistor respectively. Both these sensors detect the temperature change and the increase in intensity of light. They are connected to the analog pins of the microcontroller. The PIC is programmed so as to alert user upon increase in threshold value. This part is done through Bluetooth block that is present within the vehicle body which sends signal to the PC present in its environment .This PC have internet facility and thus upload these values to internet using IOT platform. User is alerted through an alert message. User can have live video streaming with the help of camera block.

In order to move the vehicle L293D motor driver IC is used. That gives the interfacing between the motor and microcontroller so as to provide the required voltage to run the motors. A motor driven extinguisher is provided that is controlled by the user and thus extinguish fire.

4.2 CIRCUIT DIAGRAM AND WORKING

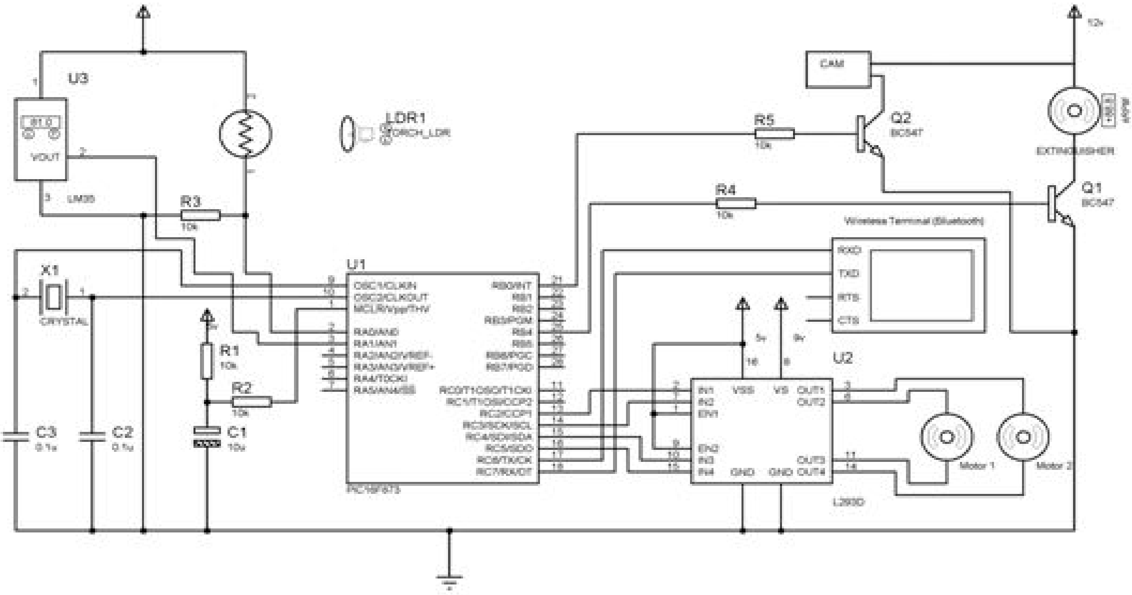


Fig 4.2 Circuit Diagram

4.2.1 EXPLANATION

The basic connections is as shown in the fig 4.2. primary connections include,MCLR or the master clear pin is an active low pin which is used to reset the circuit,this pin is made to be high as long as the circuit is in operation mode. The PIC16F873A can be operated in four different
oscillator modes. The user can program two configuration bits (FOSC1 and FOSC0) to select one of these four modes:

- LP Low-Power Crystal
- XT Crystal/Resonator
- HS High-Speed Crystal/Resonator
- RC Resistor/Capacitor

Here it is operated in XT mode. In XT, LP or HS modes, a crystal or ceramic resonator is connected to the OSC1/CLKI and OSC2/CLKO pins to establish oscillation . The PIC16F873Aoscillator design requires the use of a parallel cut crystal. The crystal frequency is set at 4MHz.The internal oscillation frequency is 20MHz.

When a fire is sensed both temperature sensor (LM35) and LDR produces corresponding voltage.This voltage is given to the analog pins of pic,as output of each of the sensors would be analog pins.A reference value for both sensor outputs are set in the program that is loaded into the microcontroller as limits. On exceeding this threshold voltage an alert message is send to the user.This happens through the transmitter and reciever pins of the IC(pin no:17&18) .Actually this signal is send to the server computer which uploads the values into internet ,programmed such a way that it can send alert message to the specified mobile number.This communication is enabled through Bluetooth module that is connected to the transmitter and receiver pins of the microcontroller. This Bluetooth module is connected to the Bluetooth module interfaced with the server computer in its environment .This system having internet connectivity uploads values to the webpage created using visual basic studio software so as to send alert message to the mobile number provided in the program .The user can control the robot by using the same web page. There are control buttons provided so as to control each operation on the robot. Initially the camera is switched off .the user can switch it on to view the current happenings. When the camera on button is pressed the microcontroller is programmed so as to provide a high on pin 21, here the transistor act as a switch ,as a high on the same pin makes transistor on and passes the required voltage to make camera on.

When the arrow buttons are given by user ,the vehicle starts to move in the desired direction towards the fire. The output voltage from microcontroller is low hence to drive two motors, a motor driver IC such as L293D is used. Four digital I/O pins of port C are programmed as output pins. These pins are connected to the four input pins of L293D .L293D can drive two motors at the same time. As the vehicle is moved towards the fire source, the user can use the extinguisher on button on the web page so as to trigger the extinguisher provided on the vehicle ,PIC provides a high on pin 25 on receiving signal to switch on extinguisher and thus the transistor acts as a closed switch and required voltage reaches the motor to trigger the extinguisher, after extinguishing fire the extinguisher button can be used to switch it off .

4.3 FLOWCHART

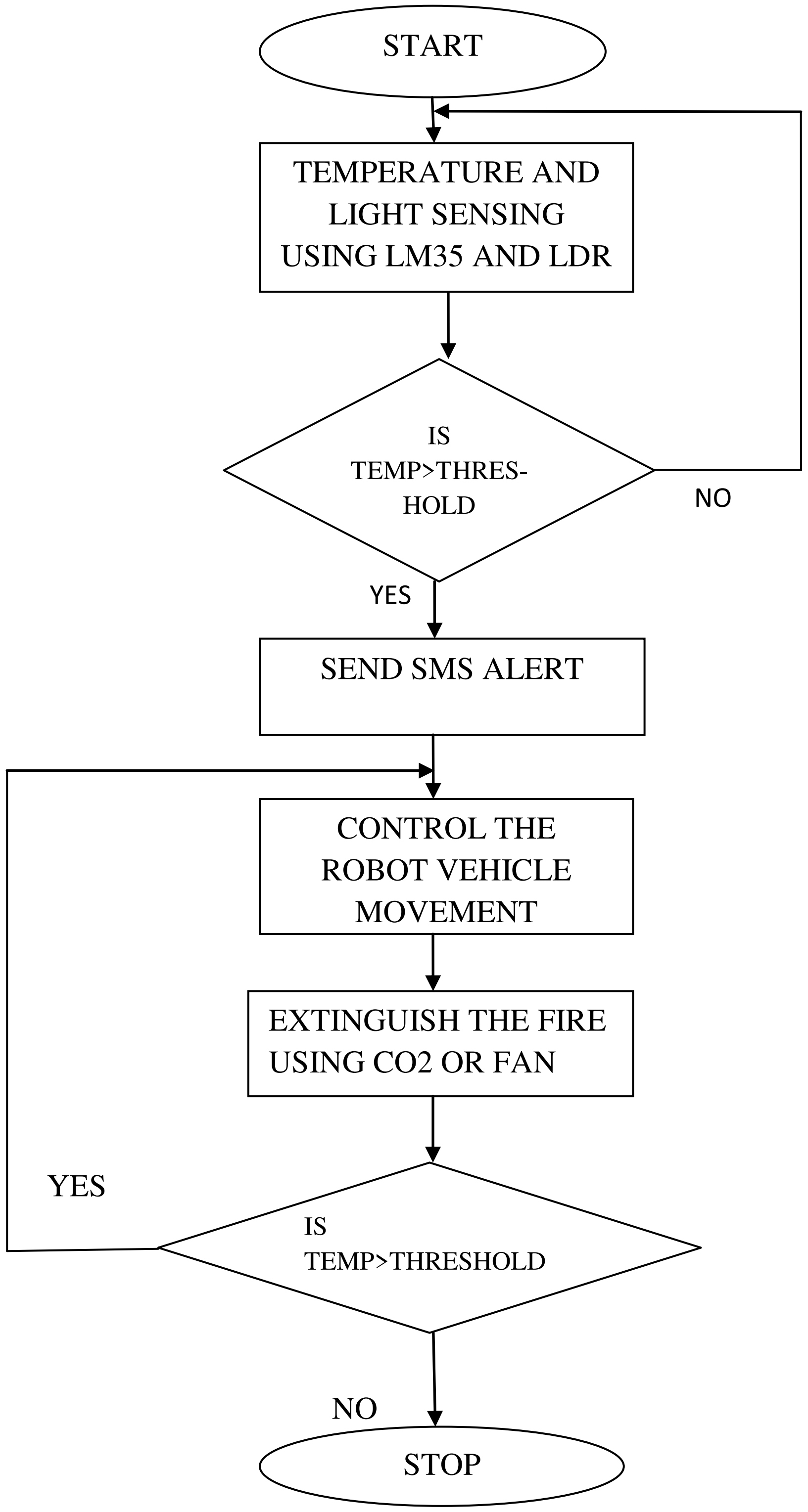

START
TEMPERATURE AND LIGHT SENSING USING LM35 AND LDR
IS TEMP>THRES-HOLD
NO
YES
SEND SMS ALERT
CONTROL THE ROBOT VEHICLE MOVEMENT
EXTINGUISH THE FIRE USING CO2 OR FAN
YES
IS TEMP>THRESHOLD
NO
STOP

Fig 4.3 Flow Chart

4.4 HARDWARE IMPLEMENTATION

4.4.1 PIC16F873A MICROCONTROLLER

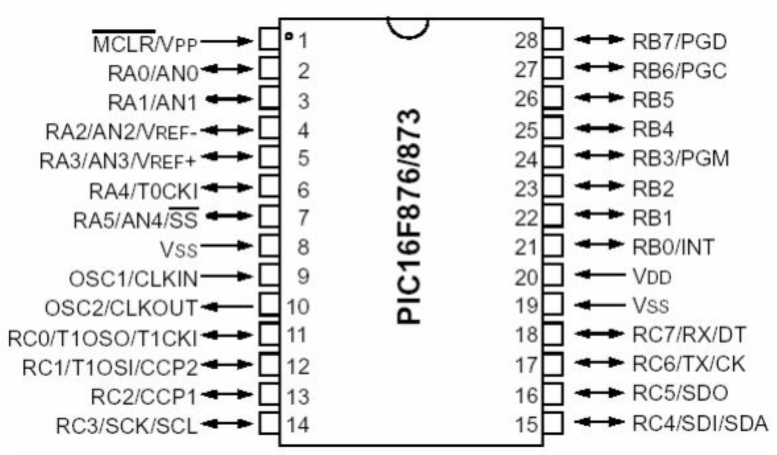
PIC 16F873 Pinout	Pic Board Pin #	Function
		
	1	V_{cc} – +5 V
	2	No Connection
	3	RB2
	4	RB1
	5	RB0/INT
	6	No Connection
	7	RA0/AN0
	8	RA1/AN1
	9	RA2/AN2/V_{REF-}
	10	No Connection
	11	RC7/RX/DT
	12	RC6/TX/CK
	13	RC5/SDO
	14	RC4/SDI/SDA
	15	RC3/SCK/SCL
	16	RC2/CCP1
	17	RC1/T1OSI/CCP2
	18	RC0/T1OSO/T1CKI
	19	No Connection
	20	Ground – 0 V

Fig .4.4 Pin out of PIC16F873A

Features:

-High-Performance RISC CPU

- Only 35 single word instructions to learn
- All instructions are single cycle (1µs) except for program branches
- Operating speed: DC - 20MHz clock input
- 4 KBytes Flash Program Memory
- 192 Byte RAM Data Memory
- 128 Byte EEPROM Data Memory
- In-circuit Serial Programming

- Interrupt Capability (up to 10 sources)

Peripheral Features

- Two 8-bit timer/counter(TMR0, TMR2) with 8-bit programmable prescalar
- One 16 bit timer/counter (TMR1)
- High current source/sink for direct LED drive
- Watchdog Timer (WDT) with Separate RC Oscillator
- Two Capture, Compare, PWM Modules
- Synchronous Serial Port with SPI and I²C
- Five Channel, 10-bit Analog to Digital Converter
- Universal Synchronous Asynchronous Receiver Transmitter (USART)

Special Microcontroller Features

- Power-On Reset
- Power-up Timer (PWRT) and Oscillator Start-Up Timer (OST)
- 1,000 erase/write cycls Enhanced Flash Program Memory
- 1,000,000 typical erase/write cycles EEPROM Data Memory
- Selectable Oscillator Options

CMOS Technology

- Low power, high speed CMOS FLASH technology
- Fully Static Design
- Low Power Consumption

I/O and Packages

- 22 I/O pins with individual direction control
- 28-pin DIP

USART and UART

UART is a commonly used hardware module for serial communication based on communication protocols like RS 232. Serial communication can be of synchronous or asynchronous type. Common clock will guide both the receiver and the transmitter in synchronous communication which is absent in asynchronous communication. UART is the most common method of asynchronous serial communication between two devices. UART communication is carried out using two lines Tx (transmission line) and Rx (reception line).

It is possible to transmit and receive data simultaneously (full duplex) using UART interface. PIC microcontrollers possess (most of them) an in-built USART hardware which can be configured in three different modes.

USART asynchronous (full duplex)

USART synchronous master (half duplex)

USART synchronous slave (half duplex)

Rs-232 Level Converter

RS-232 is a serial communication protocol for the transmission of data between two devices. The protocol defines a logic 0 with voltage in the range of +3V to +15V and a logic 1 with voltage in between -3V to -15V. This voltage level is incompatible with a CMOS or TTL logic family device, since the voltage levels are 0V for a logic 0 and 5V for logic 1. This mismatch in the voltage levels demand the requirement of a voltage level shifter to convert RS-232 logic levels into TTL logic levels and vice versa. Most commonly used voltage level shifter IC for this purpose is MAX232.

Analog-To-Digital Converter (A/D) Module

The Analog-to-Digital (A/D) Converter module has five inputs for the 28-pin devices and eight for the 40/44-pin devices. The conversion of an analog input signal results in a corresponding 10-bit digital number. The A/D module has high and low-voltage reference input that is software selectable to some combination of VDD, VSS, RA2 or RA3. The A/D converter has a unique feature of being able to operate while the device is in Sleep mode. To operate in Sleep, the A/D clock must be derived from the A/D’s internal RC oscillator.

The A/D module has four registers. These registers are:

- A/D Result High Register (ADRESH)
- A/D Result Low Register (ADRESL)
- A/D Control Register 0 (ADCON0)
- A/D Control Register 1 (ADCON1)

The ADCON0 register controls the operation of the A/D module. The ADCON1 registers, configures the functions of the port pins. The port pins can be configured as analog inputs (RA3 can also be the voltage reference) or as digital I/O.

Special Features Of The CPU

All PIC16F87XA devices have a host of features intended to maximize system reliability, minimize cost through elimination of external components, provide power saving operating modes and offer code protection. These are:

• Oscillator Selection
• Reset
- Power-on Reset (POR)
- Power-up Timer (PWRT)
- Oscillator Start-up Timer (OST)
- Brown-out Reset (BOR)
• Interrupts
• Watchdog Timer (WDT)
• Sleep
• Code Protection
• ID Locations
• In-Circuit Serial Programming
• Low-Voltage In-Circuit Serial Programming
• In-Circuit Debugger

PIC16F87XA devices have a Watchdog Timer which can be shut-off only through configuration bits. It runs off its own RC oscillator for added reliability. There are two timers that offer necessary delays on power-up. One is the Oscillator Start-up Timer (OST), intended to keep the chip in Reset until the crystal oscillator is stable. The other is the Power-up Timer (PWRT), which provides a fixed delay of 72 ms (nominal) on power-up only. It is designed to keep the part in Reset while the power supply stabilizes. With these two timers on-chip, most applications need no external Reset circuitry.

Sleep mode is designed to offer a very low current power-down mode. The user can wake-up from Sleep through external Reset, Watchdog Timer wake-up or through an interrupt. Several oscillator options are also made available to allow the part to fit the application. The RC oscillator option saves system cost while the LP crystal option saves power. A set of configuration bits is used to select various options.

I/O PORTS

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

PORTA and the TRISA Register

PORTA is a 6-bit wide, bidirectional port. The corresponding data direction register is TRISA. Setting a TRISA bit (= 1) will make the corresponding PORTA pin an input (i.e., put the corresponding output driver in a High-Impedance mode). Clearing a TRISA bit (= 0) will make the corresponding PORTA pin an output (i.e., put the contents of the output latch on the selected pin). Reading the PORTA register reads the status of the pins, whereas writing to it will write to the port latch. All write operations are read-modify-write operations. Therefore, a write to a port implies that the port pins are read, the value is modified and then written to the port data latch. Pin RA4 is multiplexed with the Timer0 module clock input to become the RA4/T0CKI pin. The RA4/T0CKI pin is a Schmitt Trigger input and an open-drain output. All other PORTA pins have TTL input levels and full CMOS output drivers. Other PORTA pins are multiplexed with analog inputs and the analog VREF input for both the A/D converters and the comparators. The operation of each pin is selected by clearing/setting the appropriate control bits in the ADCON1 and/or CMCON registers.

PORTB and the TRISB Register

PORTB is an 8-bit wide, bidirectional port. The corresponding data direction register is TRISB. Setting a TRISB bit (= 1) will make the corresponding PORTB pin an input (i.e., put the corresponding output driver in a High-Impedance mode). Clearing a TRISB bit (= 0)will make the corresponding PORTB pin an output (i.e., put the contents of the output latch on the selected pin). Three pins of PORTB are multiplexed with the In-Circuit Debugger and Low-Voltage Programming function: RB3/PGM, RB6/PGC and RB7/PGD. Each of the PORTB pins has a weak internal pull-up. A single control bit can turn on all the pull-ups. This is performed by clearing bit RBPU (OPTION_REG<7>). The weak pull-up is automatically turned off when the port pin is configured as an output. The pull-ups are disabled on a Power-on Reset.

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

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

b) Clear flag bit RBIF.

A mismatch condition will continue to set flag bit RBIF. Reading PORTB will end the mismatch condition and allow flag bit RBIF to be cleared. The interrupt-on-change feature is recommended for wake-up on key depression operation and operations where PORTB is only used for the interrupt-on-change feature. Polling of PORTB is not recommended while using the interrupt-on-change feature.

PORTC and the TRISC Register

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

4.4.2 SENSOR INTERFACING

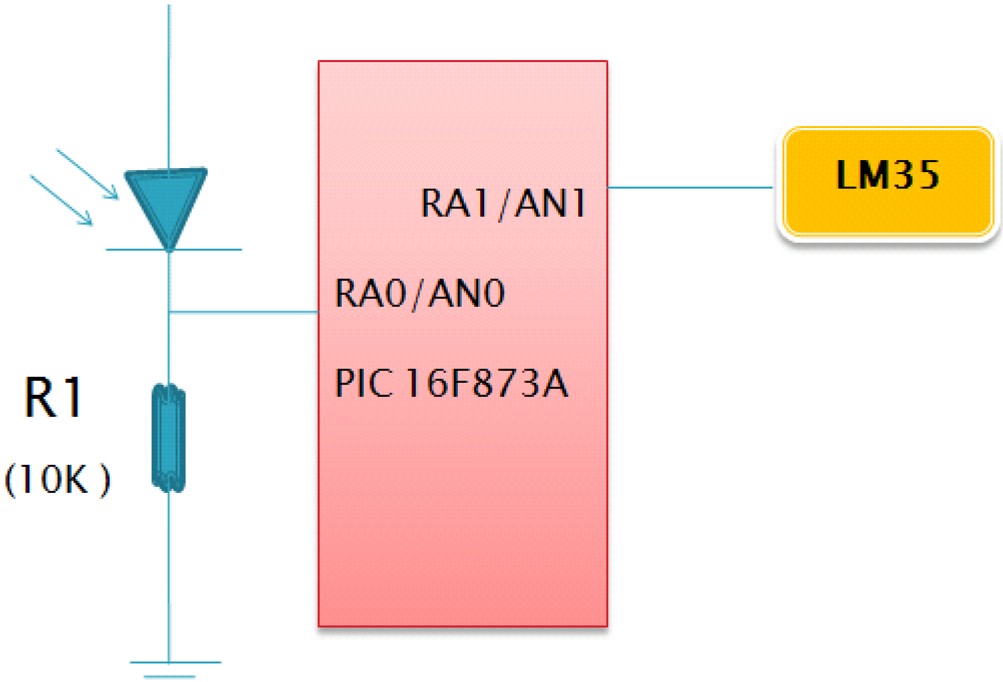


Fig 4.5 Sensor Interfacing

Here we have used two sensors for fire detection. They are LDR and LM35.The LDR is connected to the pin no 2 of the pic 16f873a and LM35 is connected to the pin no 3 of the pic 16f873a.

4.4.2.1 LDR

A Light Dependent Resistor (LDR) or a photo resistor is a device whose resistivity is a function of the incident electromagnetic radiation. Hence, they are light sensitive devices. They are also called as photo conductors, photo conductive cells or simply photocells. They are made up of semiconductor materials having high resistance. There are many different symbols used to indicate a LDR, one of the most commonly used symbol is shown in the figure below. The arrow indicates light falling on it.

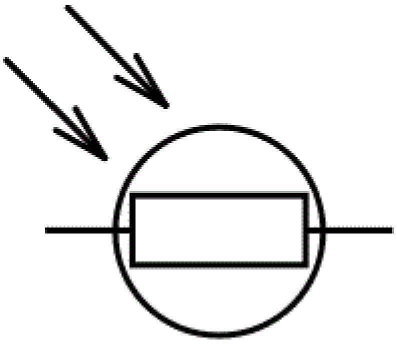


Fig 4.6 Symbolic Representation

Working Principle of LDR

A light dependent resistor works on the principle of photo conductivity. Photo conductivity is an optical phenomenon in which the materials conductivity is increased when light is absorbed by the material. When light falls i.e. when the photons fall on the device, the electrons in the valence band of the semiconductor material are excited to the conduction band. These photons in the incident light should have energy greater than the band gap of the semiconductor material to make the electrons jump from the valence band to the conduction band. Hence when light having enough energy strikes on the device, more and more electrons are excited to the conduction band which results in large number of charge carriers. The result of this process is more and more current starts flowing through the device when the circuit is closed and hence it is said that the resistance of the device has been decreased. This is the most common working principle of LDR.

Characteristics of LDR

LDR’s are light dependent devices whose resistance is decreased when light falls on them and that is increased in the dark. When a light dependent resistor is kept in dark, its resistance is very high. This resistance is called as dark resistance. It can be as high as 10^{12} Ω and if the device is allowed to absorb light its resistance will be decreased drastically. If a constant voltage is applied to it and intensity of light is increased the current starts increasing. Figure below shows resistence vs. illumination curve for a particular LDR.

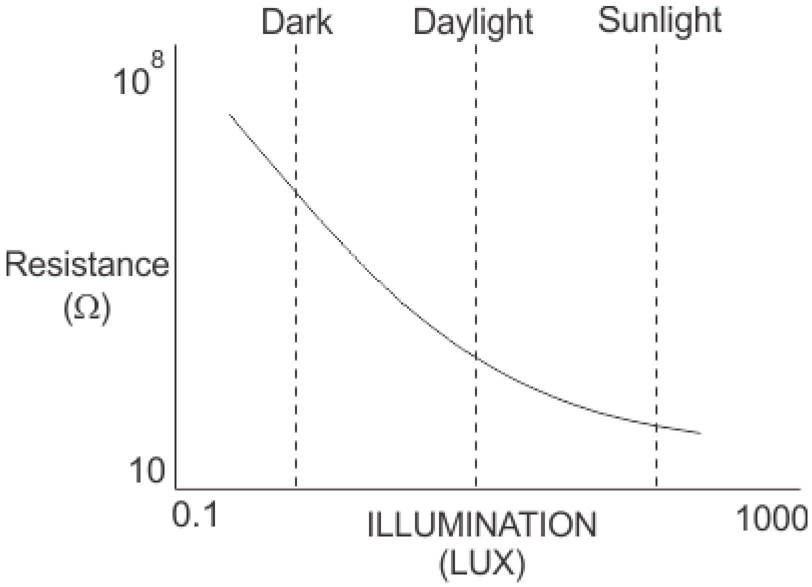


Fig 4.7 Resistence Vs.Illumination Curve

Photocells or LDR’s are non linear devices. There sensitivity varies with the wavelength of light incident on them. Some photocells might not at all response to a certain range of wavelengths. Based on the material used different cells have different spectral response curves.

When light is incident on a photocell it usually takes about 8 to 12 ms for the change in resistance to take place, while it takes one or more seconds for the resistance to rise back again to its initial value after removal of light. This phenomenon is called as resistance recovery rate. This property is used in audio compressors. Also, LDR’s are less sensitive than photo diodes and photo transistor. (A photo diode and a photocell (LDR) are not the same, a photo-diode is a p-n junction semiconductor device that converts light to electricity, whereas a photocell is a passive device, there is no p-n junction in this nor it “converts” light to electricity). Types of Light Dependent Resistors: Based on the materials used they are classified as:

1. Intrinsic photo resistors (Undoped semiconductor): These are made of pure semiconductor materials such as silicon or germanium. Electrons get excited from valance band to conduction band when photons of enough energy fall on it and number charge carriers is increased.
2. Extrinsic photo resistors: These are semiconductor materials doped with impurities which are called as dopants. Theses dopants create new energy bands above the valence band which are filled with electrons. Hence this reduces the band gap and less energy is required in exciting them. Extrinsic photo resistors are generally used for long wavelengths.

4.4.2.2 LM35

The LM35 is an integrated circuit sensor that can be used to measure temperature with an electrical output proportional to the temperature (in °C).It can measure temperature more accurately than a using a thermistor. The sensor circuitry is sealed and not subject to oxidation. The LM35 generates a higher output voltage than thermocouples and may not require that the output voltage be amplified. The LM35 has an output voltage that is proportional to the Celsius temperature. The scale factor is .01V/°C.

The LM35 does not require any external calibration or trimming and maintains an accuracy of +/-0.4°C at room temperature and +/-0.8°C over a range of 0°C to +100°C.Another important characteristic of the LM35 is that it draws only 60 micro amps from its supply and possesses a low self-heating capability.The LM35 comes in many different packages such as TO-92 plastic transistor-like package,T0-46 metal can transistor-like package,8-lead surface mount SO-8 small outline package.

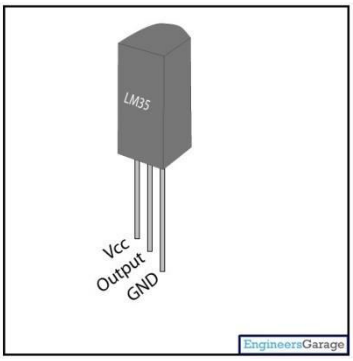


Fig 4.8 Pin Diagram

Pin No	Function	Name
1	Supply voltage; 5V (+35V to -2V)	Vcc
2	Output voltage (+6V to -1V)	Output
3	Ground (0V)	Ground

Table 4.1 Pin Description

Working Principle Of LM35

There are two transistors in the center of the drawing. One has ten times the emitter area of the other. This means it has one tenth of the current density, since the same current is going through both transistors. This causes a voltage across the resistor R1 that is proportional to the absolute temperature, and is almost linear across the range.The "almost" part is taken care of by a special circuit that straightens out the slightly curved graph of voltage versus temperature.

The amplifier at the top ensures that the voltage at the base of the left transistor (Q1) is proportional to absolute temperature (PTAT) by comparing the output of the two transistors.

The amplifier at the right converts absolute temperature (measured in Kelvin) into either Fahrenheit or Celsius, depending on the part (LM34 or LM35).The little circle with the "i" in it is a constant current source circuit.

The two resistors are calibrated in the factory to produce a highly accurate temperature sensor.

The integrated circuit has many transistors in it -- two in the middle, some in each amplifier, some in the constant current source, and some in the curvature compensation circuit. All of that is fit into the tiny package with three leads.

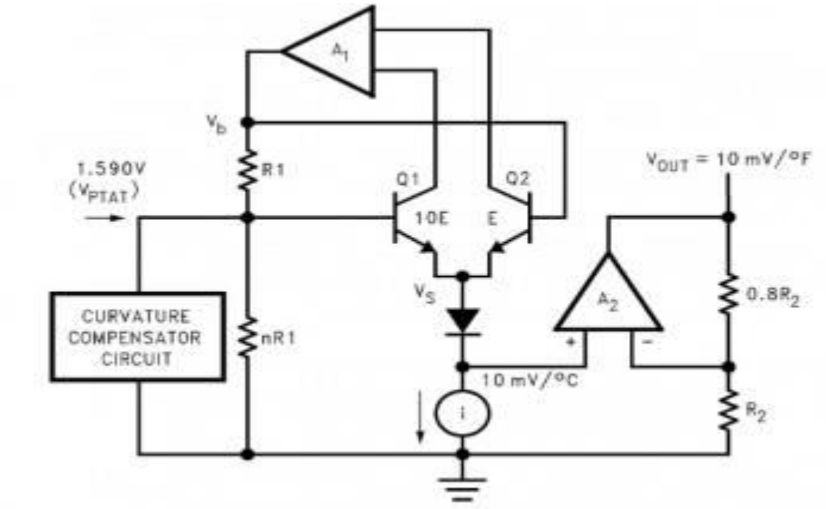


Fig 4.9 Internal Architecture Of Lm35

4.4.3 BLUETOOTH INTERFACING

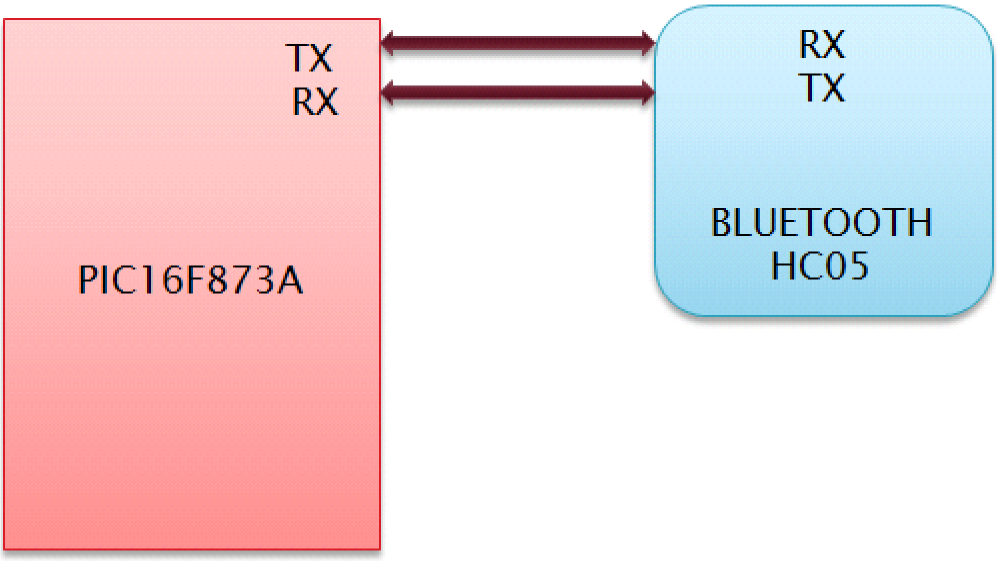


Fig 4.10 Bluetooth Interfacing

Here the pin no 17 of the pic 16f873a which is the TX(transmitter) is connected to the RX (receiver) of Bluetooth HC05 and the pin no 18 of the pic 16f873a which is the RX(receiver) is connected to the TX(transmitter) of Bluetooth HC05 .

4.4.3.1 BLUETOOTH HC05

HC-05 module is an easy to use Bluetooth SPP (Serial Port Protocol) module, designed for transparent wireless serial connection setup.

Serial port Bluetooth module is fully qualified Bluetooth V2.0+EDR (Enhanced Data Rate) 3Mbps Modulation with complete 2.4GHz radio transceiver and baseband. It uses CSR Bluecore 04-External single chip Bluetooth system with CMOS technology and with AFH(Adaptive Frequency Hopping Feature). It has the footprint as small as 12.7mmx27mm. Hope it will simplify your overall design/development cycle.

Fig:4.11 Bluetooth Module

Software Features

- Default Baud rate: 38400, Data bits:8, Stop bit:1,Parity:No parity, Data control: has.
- Supported baud rate: 9600,19200,38400,57600,115200,230400,460800.
- Given a rising pulse in PIO0, device will be disconnected.
- Status instruction port PIO1: low-disconnected, high-connected;
- PIO10 and PIO11 can be connected to red and blue led separately. When master and slave
- are paired, red and blue led blinks 1time/2s in interval, while disconnected only blue led blinks 2 time/(s).
- Auto-connect to the last device on power as default.
- Permit pairing device to connect as default.
- Auto-pairing PINCODE:"0000" as default
- Auto-reconnect in 30 min when disconnected as a result of beyond the range of Connection.

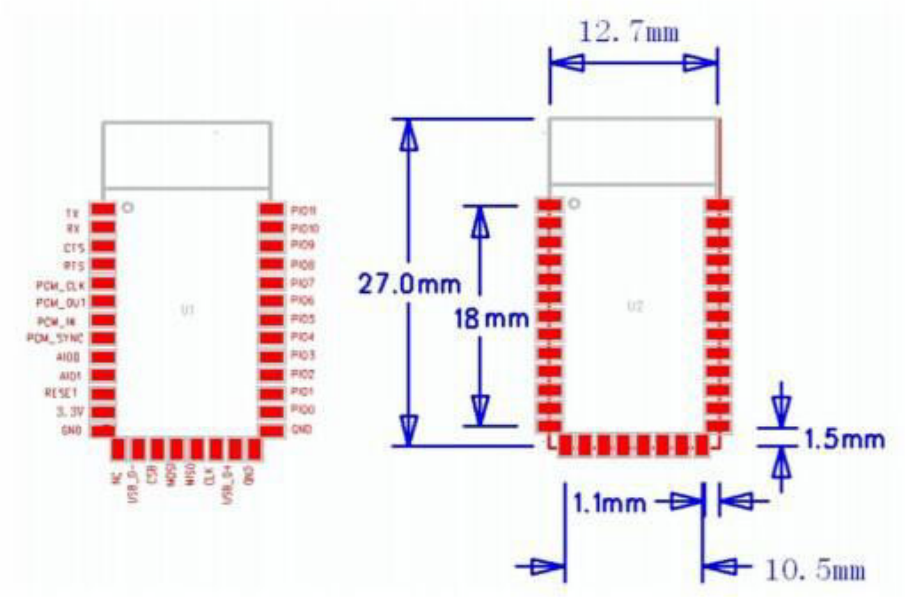


Fig4.12 Bluetooth Hardware

PIN Name	PIN #	PAD Type	Description
GND	13,21,22	VSS	Ground Pot
3.3 VCC	12	3.3V	Integrated 3.3V(+) supply with On-chip linear regulator output within 3.15-3.3V
AIO0	9	Bi-directional	Programmable input/output line
AIO1	10	Bi-directional	Programmable input/output line
AIO0	23	Bi-directional RX EN	Programmable input/output line, control output for LNA (if fitted)
AIO1	24	Bi-directional TX EN	Programmable input/output line, control output for PA (if fitted)

TABLE 4.2 Pin description

Hardware Features

- Typical -80dBm sensitivity
- Up to +4dBm RF transmit power
- Low Power 1.8V Operation ,1.8 to 3.6V I/O
- PIO control
- UART interface with programmable baud rate
- With integrated antenna
- With edge connector

Applications:

- Computer and peripheral devices
- GPS receiver
- Industrial control
- MCU projects

4.4.4 MOTOR DRIVER INTERFACING

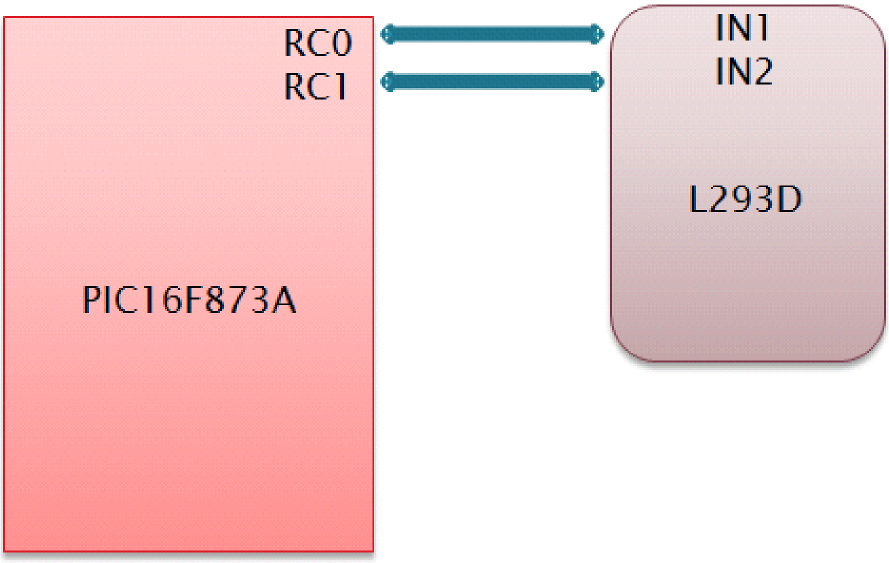


Fig 4.13 Motor Driver Interfacing

Here the pin no 13 of pic 16f873a which is the RC2 is connected to pin no 2 of L293D which is the IN1, the pin no 14 of pic 16f873a which is the RC3 is connected to pin no 7 of L293D which is the IN2, the pin no 15 of pic 16f873a which is the RC4 is connected to pin no 10 of L293D which is the IN3 and the pin no 16 of pic 16f873a which is the RC5 is connected to pin no 15 of L293D which is the IN4.

4.4.4.1 L293D

L293D is a typical Motor driver or Motor Driver IC which allows DC motor to drive on either direction. L293D is a 16-pin IC which can control a set of two DC motors simultaneously in any direction. It means that you can control two DC motor with a single L293D IC. Dual H-bridge Motor Driver integrated circuit (IC*)*.

It works on the concept of H-bridge. H-bridge is a circuit which allows the voltage to be flown in either direction. As you know voltage need to change its direction for being able to rotate the motor in clockwise or anticlockwise direction, Hence H-bridge IC are ideal for driving a DC motor.

In a single L293D chip there are two h-Bridge circuit inside the IC which can rotate two dc motor independently. Due its size it is very much used in robotic application for controlling DC motors. Given below is the pin diagram of a L293D motor controller.

There are two Enable pins on l293d. Pin 1 and pin 9, for being able to drive the motor, the pin 1 and 9 need to be high. For driving the motor with left H-bridge you need to enable pin 1 to high. And for right H-Bridge you need to make the pin 9 to high. If anyone of the either pin1 or pin9 goes low then the motor in the corresponding section will suspend working. It’s like a switch.

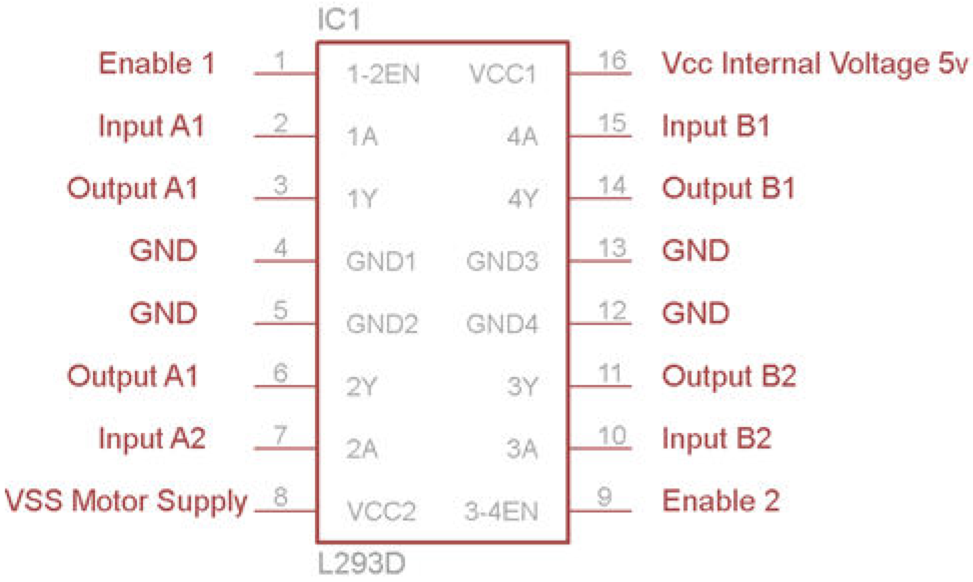


Fig:4.14 L293D Pin Diagram

Working of L293D

There are 4 input pins for l293d, pin 2,7 on the left and pin 15 ,10 on the right as shown on the pin diagram. Left input pins will regulate the rotation of motor connected across left side and right input for motor on the right hand side. The motors are rotated on the basis of the inputs provided across the input pins as LOGIC 0 or LOGIC 1.

4.4.4.2 MOTOR INTERFACING

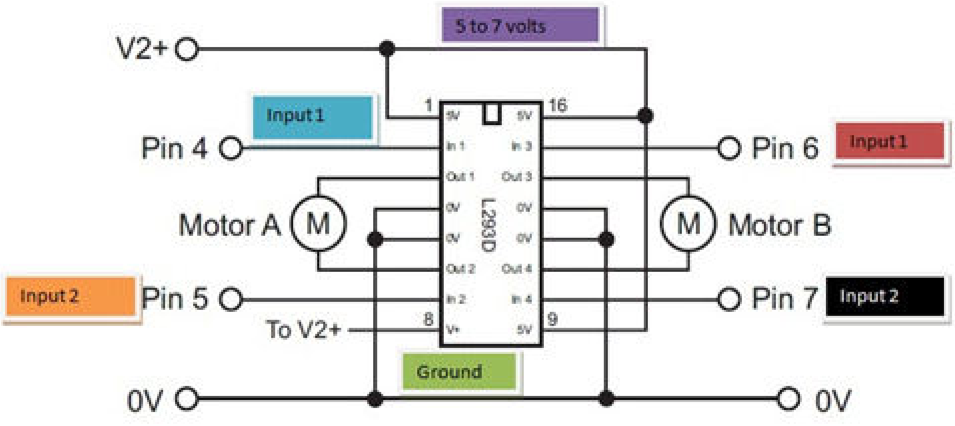


Fig4.15 Motor Interfacing

Motor A is connected between pin no 3 and 6 of L293D and Motor B is connected between pin no 11 and 14 of L293D.

DC Gear Motor

Miniature DC gear motors using spur gears offer advantages in applications with a maximum current limit, where the lowest input friction and high efficiency are essential. The broad range of Portescap spur gearboxes is well adapted to our motor lines, spanning the range of motor diameters. A combination of our spur gearheads with our brush DC coreless motors provides a single, factory-integrated solution for demanding, high torque applications. Due to its spur compound geartrain arrangement, the micro dc gear motor provides the output along the same axis as the motor. The inherent design of our brush DC gear motors results in overall better efficiency and lower noise, benefiting a wide range of applications including medical pumps, security and access and watch winders. Alternately, the main advantages of Portescap planetary small dc gear motors are their high rated torque and high reduction ratio per gear train, utilizing high-quality composite materials. Portescap's high-speed planetary brushless dc gear motors product line was designed for use on brushless motors with iron core windings. The gearboxes tolerate input speeds in the range of 10,000 to 70,000 rpm and output speeds of several thousand rpm. This facilitates a motor-gearbox unit of small dimensions that can provide extremely high values of speed and torque.

Geared DC motors can be defined as an extension of DC motor which already had its Insight details demystified here. A geared DC Motor has a gear assembly attached to the motor. The speed of motor is counted in terms of rotations of the shaft per minute and is termed as RPM .The gear assembly helps in increasing the torque and reducing the speed. Using the correct combination of gears in a gear motor, its speed can be reduced to any desirable figure. This concept where gears reduce the speed of the vehicle but increase its torque is known as gear reduction. This Insight will explore all the minor and major details that make the gear head and hence the working of geared DC motor.

External Structure

At the first sight, the external structure of a DC geared motor looks as a straight expansion over the simple DC ones.

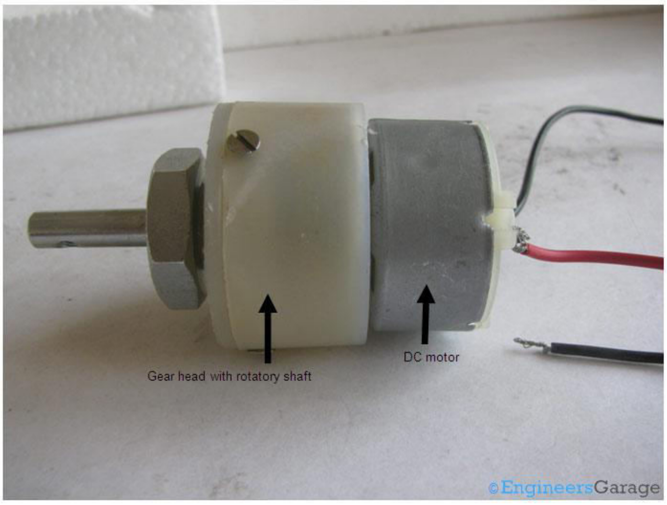


Fig:4.16 DC Gear Motor

Features

- Wide range of selection from 0.3 watt to 3.4 watt
- Metal gear
- General application
- Low and ultra low power solution

4.4.5 20MHz CRYSTAL OSCILLATOR

A crystal oscillator is an electronic oscillator circuit that uses the mechanical resonance of a vibrating crystal of piezoelectric material to create an electrical signal with a precise frequency. This frequency is commonly used to keep track of time, as in quartz wristwatches, to provide a stable clock signal for digital integrated circuits, and to stabilize frequencies for radio transmitters and receivers. The most common type of piezoelectric resonator used is the quartz crystal, so oscillator circuits incorporating them became known as crystal oscillators, but other piezoelectric materials including polycrystalline ceramics are used in similar circuits.

Quartz crystals are manufactured for frequencies from a few tens of kilohertz to hundreds of megahertz. More than two billion crystals are manufactured annually. Most are used for consumer devices such as wristwatches, clocks, radios, computers, and cell phones. Quartz crystals are also found inside test and measurement equipment, such as counters, signal generators, and oscilloscopes.

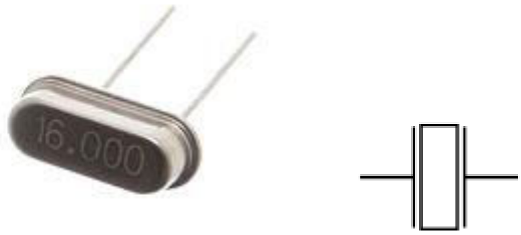


Fig 4.17 Electronic Symbol

A crystal oscillator is an electronic oscillator circuit that uses a piezoelectric resonator, a crystal, as its frequency-determining element.Crystal is the common term used in electronics for the frequency-determining component, a wafer of quartz crystal or ceramic with electrodes connected to it. A more accurate term for it is piezoelectric resonator. Crystals are also used in other types of electronic circuits, such as crystal filters.

Piezoelectric resonators are sold as separate components for use in crystal oscillator circuits. An example is shown in the picture. They are also often incorporated in a single package with the crystal oscillator circuit, shown on the righthand side.

Resonance modes

A quartz crystal provides both series and parallel resonance. The series resonance is a few kilohertz lower than the parallel one. Crystals below 30 MHz are generally operated between series and parallel resonance, which means that the crystal appears as an inductive reactance in operation, this inductance forming a parallel resonant circuit with externally connected parallel capacitance. Any small additional capacitance in parallel with the crystal pulls the frequency lower. Moreover, the effective inductive reactance of the crystal can be reduced by adding a capacitor in series with the crystal. This latter technique can provide a useful method of trimming the oscillatory frequency within a narrow range; in this case inserting a capacitor in series with the crystal raises the frequency of oscillation. For a crystal to operate at its specified frequency, the electronic circuit has to be exactly that specified by the crystal manufacturer. Note that these points imply a subtlety concerning crystal oscillators in this frequency range: the crystal does not usually oscillate at precisely either of its resonant frequencies.

Crystals above 30 MHz (up to >200 MHz) are generally operated at series resonance where the impedance appears at its minimum and equal to the series resistance. For these crystals the series resistance is specified (<100 Ω) instead of the parallel capacitance. To reach higher frequencies, a crystal can be made to vibrate at one of its overtone modes, which occur near multiples of the fundamental resonant frequency. Only odd numbered overtones are used. Such a crystal is referred to as a 3rd, 5th, or even 7th overtone crystal. To accomplish this, the oscillator circuit usually includes additional LC circuits to select the desired overtone.

Temperature effects

A crystal's frequency characteristic depends on the shape or "cut" of the crystal. A tuning-fork crystal is usually cut such that its frequency dependence on temperature is quadratic with the maximum around 25 °C. This means that a tuning-fork crystal oscillator resonates close to its target frequency at room temperature, but slows when the temperature either increases or decreases from room temperature. A common parabolic coefficient for a 32 kHz tuning-fork crystal is −0.04 ppm/°C^2:

4.4.6 CAMERA INTERFACING

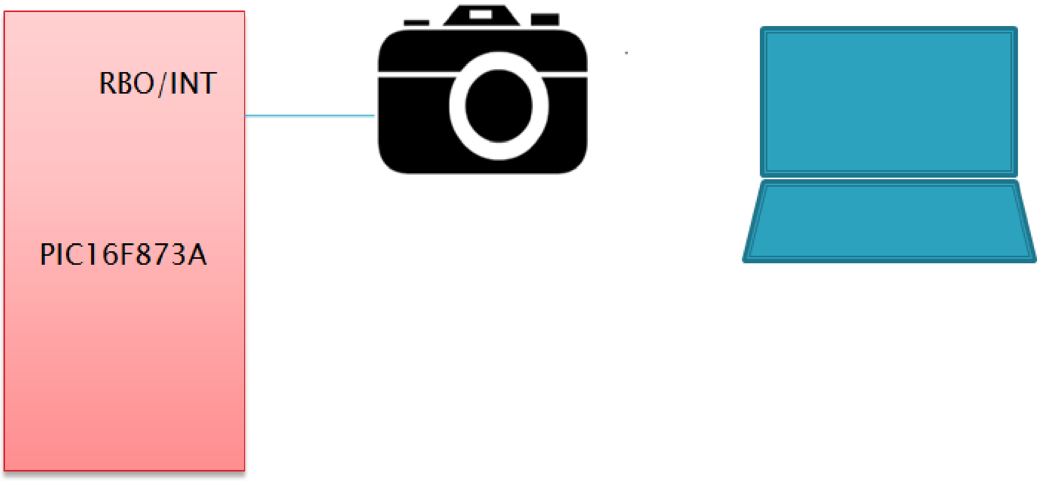


Fig 4.18 Camera Interfacing

Here the camera is connected to pin no.21 of PIC 16F873a through a transistor. The camera gets turned on only when the transistor is active.

4.4.6.1 Wireless A/V Camera

Wireless security cameras are closed-circuit television (CCTV) cameras that transmit a video and audio signal to a wireless receiver through a radio band. Many wireless security cameras require at least one cable or wire for power; "wireless" refers to the transmission of video/audio. However, some wireless security cameras are battery-powered, making the cameras truly wireless from top to bottom.

Wireless cameras are proving very popular among modern security consumers due to their low installation costs (there is no need to run expensive video extension cables) and flexible mounting options; wireless cameras can be mounted/installed in locations previously unavailable to standard wired cameras. In addition to the ease of use and convenience of access, wireless security camera allows users to leverage broadband wireless internet to provide seamless video streaming over-internet.

Features and specifications of Wireless A/V Camera:

- 380TV lines picture display.
- Low radiation, safe and healthy.
- Built-in microphone for audio monitoring.
- Including adaptive bracket, easy installation.
- Suitable for monitoring children, elders and widely used for theft prevention, after hours surveillance, home security,etc.
- Camera :Night vision enables no light or low light usage.
- High-quality picture transmitting and receiving.
- Transmit distance upto 200 Ft.

Fig:4.19 A/V Camera And Receiver

4.4.7 IOT(INTERNET OF THINGS)

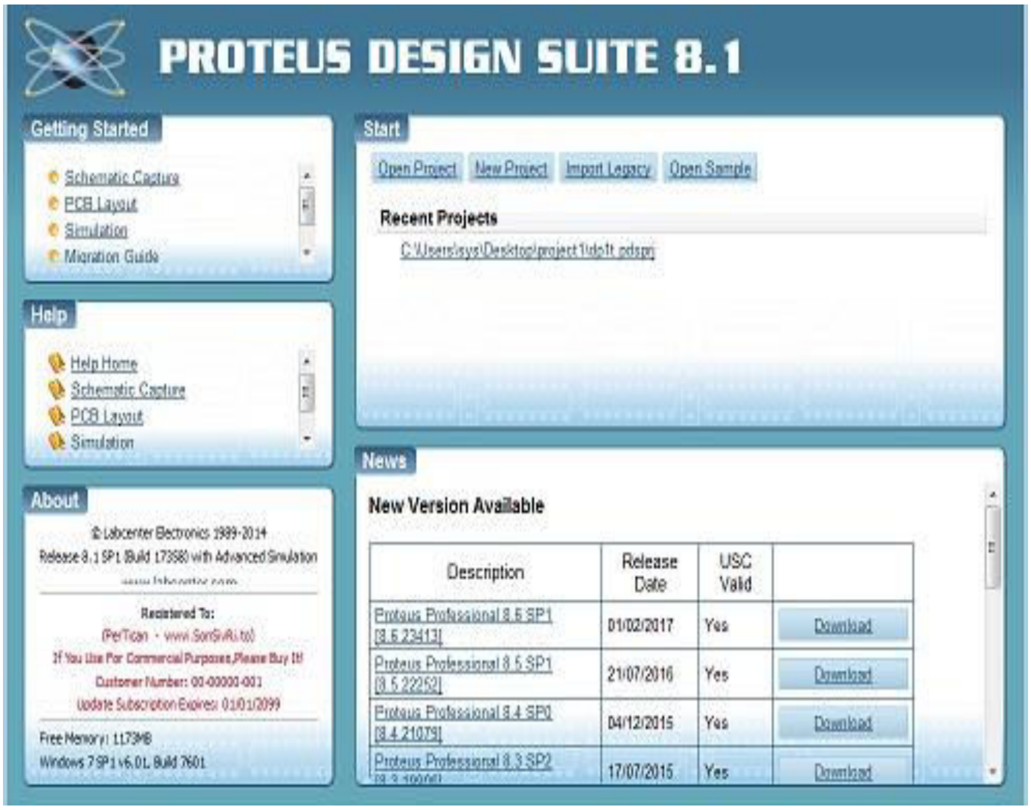
The Internet of things (stylised Internet of Things or IoT) is the internetworking of physical devices, vehicles (also referred to as "connected devices" and "smart devices"), buildings, and other items—embedded with electronics, software, sensors, actuators, and network connectivity that enable these objects to collect and exchange data. In 2013 the Global Standards Initiative on Internet of Things (IoT-GSI) defined the IoT as "the infrastructure of the information society." The IoT allows objects to be sensed or controlled remotely across existing network infrastructure, creating opportunities for more direct integration of the physical world into computer-based systems, and resulting in improved efficiency, accuracy and economic benefit in addition to reduced human intervention. When IoT is augmented with sensors and actuators, the technology becomes an instance of the more general class of cyber-physical systems, which also encompasses technologies such as smart grids, smart homes, intelligent transportation and smart cities. Each thing is uniquely identifiable through its embedded computing system but is able to interoperate within the existing Internet infrastructure. Experts estimate that the IoT will consist of almost 50 billion objects by 2020.

Typically, IoT is expected to offer advanced connectivity of devices, systems, and services that goes beyond machine-to-machine (M2M) communications and covers a variety of protocols, domains, and applications.The interconnection of these embedded devices (including smart objects), is expected to usher in automation in nearly all fields, while also enabling advanced applications like a smart grid, and expanding to areas such as smart cities.

4.5 SOFTWARE IMPLEMENTATION

4.5.1 PROTEUS 8 PROFESSIONAL

The Proteus Design Suite is an Electronic Design Automation (EDA) tool including schematic capture, simulation and PCB Layout modules. It is developed in Yorkshire, England by Labcenter Electronics Ltd.



C:\Users\sys\Desktop\project1\do1t.pdsprj

© Labcenter Electronics 1989-2014
Release 8.1 SP1 (Build 17358) with Advanced Simulation
www.labcenter.com

Registered To:
(PerTican - www.SonSivRi.to)
If You Use For Commercial Purposes,Please Buy It!
Customer Number: 00-00000-001
Update Subscription Expires: 01/01/2099

Free Memory: 1173MB
Windows 7 SP1 v6.01, Build 7601

New Version Available

Description	Release Date	USC Valid	
Proteus Professional 8.6 SP1 [8.6.23413]	01/02/2017	Yes	Download
Proteus Professional 8.5 SP1 [8.5.22252]	21/07/2016	Yes	Download
Proteus Professional 8.4 SP0 [8.4.21079]	04/12/2015	Yes	Download
Proteus Professional 8.3 SP2 [8.3.19906]	17/07/2015	Yes	Download

Fig:4.20 Proteus Design Suite

The Proteus Design Suite is a Windows application for schematic capture, simulation, and PCB layout design. It can be purchased in many configurations, depending on the size of designs being produced and the requirements for microcontroller simulation. All PCB Design products include an auto router and basic mixed mode SPICE simulation capabilities.

Schematic capture in the Proteus Design Suite is used for both the simulation of designs and as the design phase of a PCB layout project. It is therefore a core component and is included with all product configurations. The micro-controller simulation in Proteus works by applying either a hex file or a debug file to the microcontroller part on the schematic. It is then co-simulated along with any analog and digital electronics connected to it. This enables it's use in a broad spectrum of project prototyping in areas such as motor control, temperature control and user interface design. Since no hardware is required, it is a convenient training or teaching tool.

The PCB Layout module is automatically given connectivity information in the form of a netlist from the schematic capture module. It applies this information, together with the user specified design rules and various design automation tools, to assist with error free board design. Design Rule Checking does not include high speed design constraints. PCB's of up to 16 copper layers can be produced with design size limited by product configuration.

In this project this software is used for circuit simulation and for designing the PCB layout.

4.5.2 MPLAB IDE

MPLAB is a proprietary freeware integrated development environment for the development of embedded applications on PIC and dsPIC microcontrollers. It is developed by Microchip Technology. MPLAB 8.X is the last version of the legacy MPLAB IDE technology, custom built by Microchip Technology in Microsoft Visual C++. MPLAB supports project management, editing, debugging and programming of Microchip 8-bit, 16-bit and 32-bit PICmicrocontrollers. MPLAB only works on Microsoft Windows.

Compilers are special programs that read the statements in a high-level language (source code), analyze them and convert them into machine language that the computer processor can understand (object code). Compilers also improve execution time by converting the program into an executable object code, which is more compact and runs much faster, such as ".exe" files in Windows environments. This conversion is what improves security and keeps the source code safe from being retrieved by other parties.

MPLAB supports the following compilers:

- MPLAB MPASM Assembler
- MPLAB ASM30 Assembler
- MPLAB C Compiler for PIC18
- MPLAB C Compiler for PIC24 and dsPIC DSCs

- MPLAB C Compiler for PIC32
- HI-TECH C

The compiler used depends on the microcontroller in use. Those compilers which won't get updated will not, won't support. In this project the compiler used is the HI-TECH C.

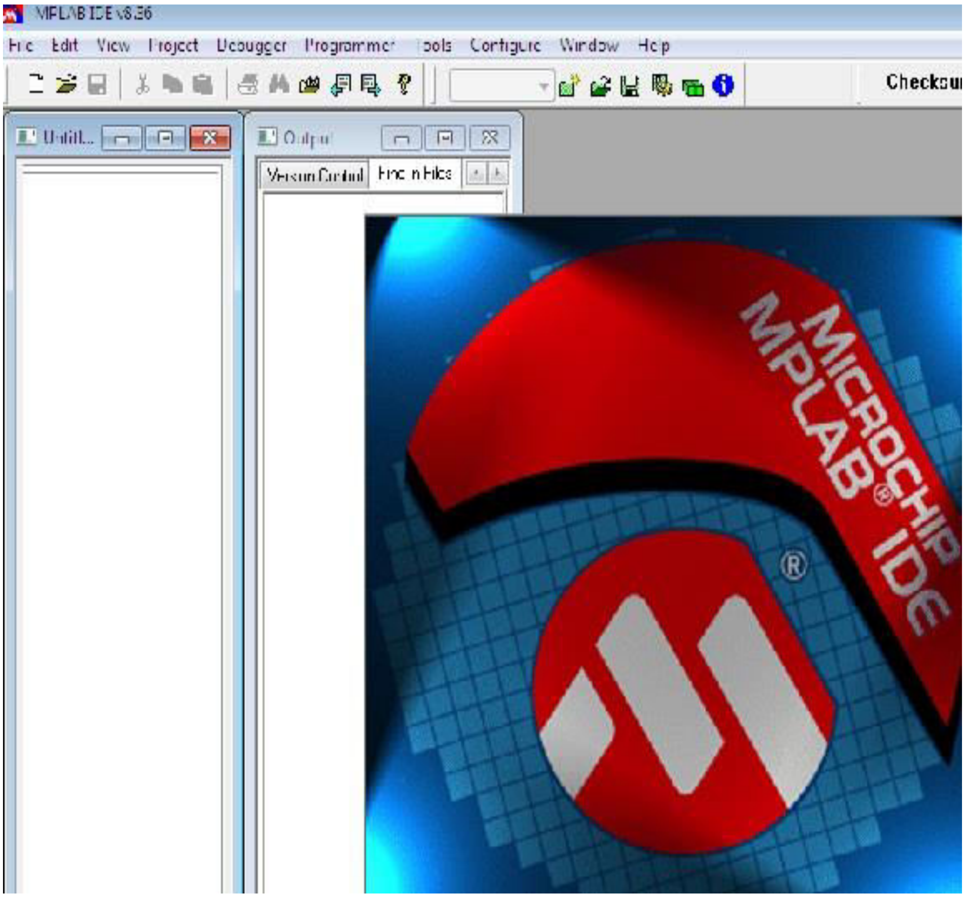


Fig 4.21 MPLAB IDE

4.5.3 PICkit 2 PROGRAMMER

PICkit is a family of programmers for PIC microcontrollers made by Microchip Technology. They are used to program and debug microcontrollers, as well as program EEPROM. Some models also feature logic analyzer and serial communications (UART) tool.

The PICkit 2 Programmer/Debugger is a low-cost development tool with an easy to use interface for programming and debugging Microchip's Flash families of microcontrollers. The PICkit 2 was introduced in May 2005 replacing the PICkit 1. The most notable difference between the two is that the PICkit 2 has a separate programmer/debugger unit which plugs into the board carrying the chip to be programmed, whereas the PICkit 1 was a single unit. This makes it possible to use the programmer with a custom circuit board via an In Circuit Serial Programming (ICSP) header.

The PICkit 2 uses an internal PIC18F2550 with FullSpeed USB. The latest PICkit 2 firmware allows the user to program and debug most of the 8 and 16 bit PICmicro and dsPIC members of the Microchip product line.It is open to the public, including its hardware schematic, firmware source code (in C language) and application programs (in C# language). End users and third parties can easily modify both the hardware and software for enhanced features. e.g. Linux version of PICkit 2 application software, DOS style CMD support, etc.

The PICkit 2 has a programmer-to-go (PTG) feature, which can download the hex file and programming instructions into on-board memory (128 KB I²C EEPROM or 256 KB I2C EEPROM), so that no PC is required at the end application.The Microchip version of PICkit 2 has a standard 128 . 256 KB memory can be achieved by modifying the hardware or from third party.Additionally, a 500 kHz three-channel logic analyser and a UART tool are built into the PICkit 2.

In this project this software is used to load the programs to the Pic16F873a.

Features & Specification of PICkit 2

- User friendly & High performance.
- Low cost ,Small size & Simple design.
- External power supply not required.
- Support only 5V PIC (5V output voltage with 120mA cureent at VDD pin)
- 5 pin ICSP (MCLR, VDD, GND, PGC, PGD).
- Operating system support: Windows XP, Vista, 7, 8.
- Compatible MPLAB IDE, MPLAB X & Mcrochip PICkit 2.
- Micro USB for easy transter.

- Flexible with both PC DESKTOP & LAPTOP.

ICSP Pin out

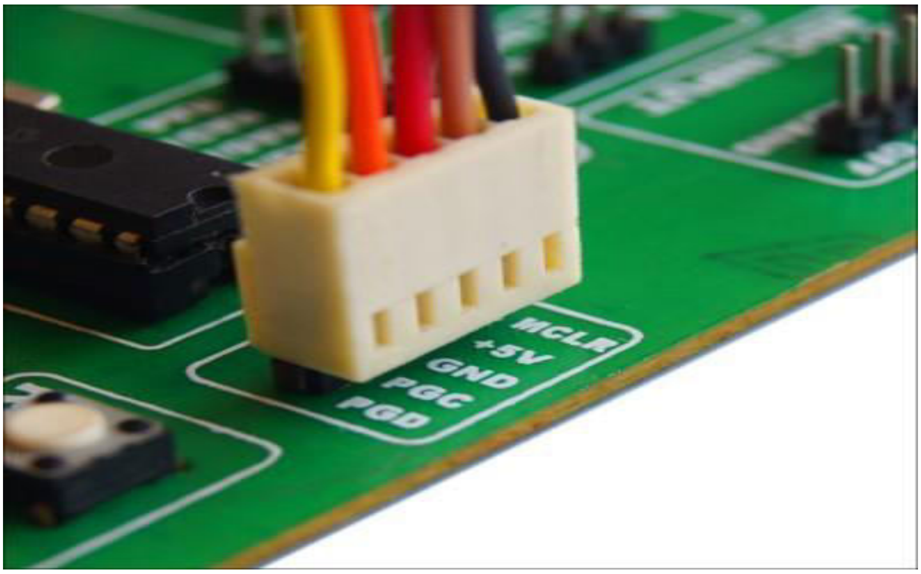
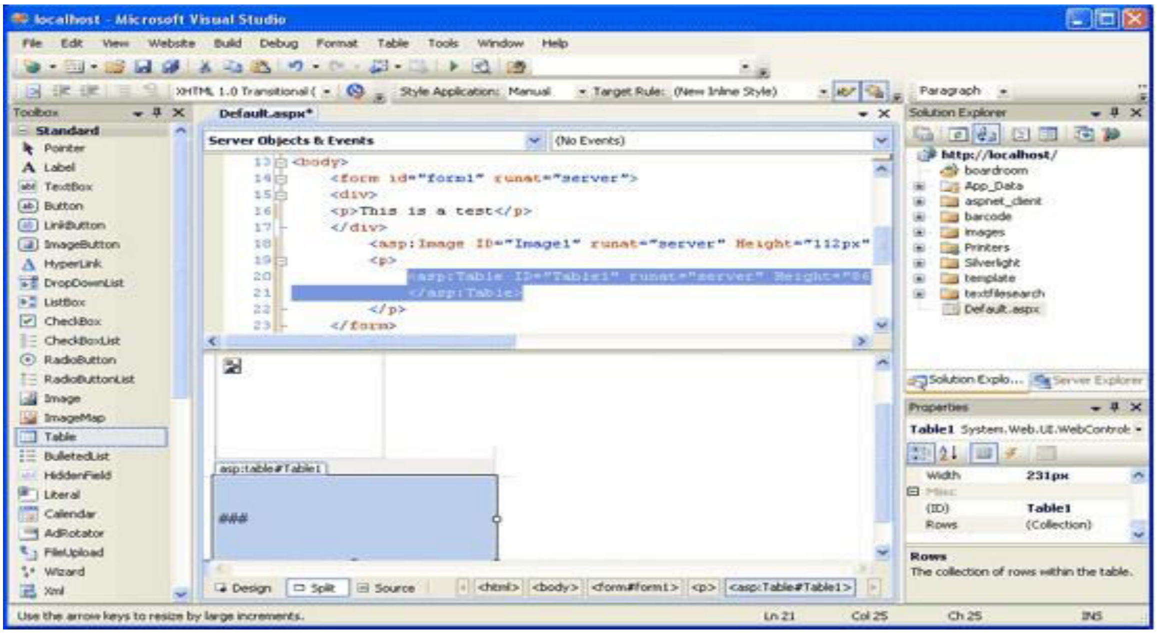


Fig 4.22. ICSP Pin out

4.5.4 MICROSOFT VISUAL STUDIO 2008

Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop computer programs for Microsoft Windows, as well as web sites, web apps, web services and mobile apps. Visual Studio uses Microsoft software development platforms such as Windows API, Windows Forms, Windows Presentation Foundation, Windows Store and Microsoft Silverlight. It can produce both native code and managed code

Visual Studio includes a code editor supporting IntelliSense(the code completion component) as well as code refactoring. The integrated debugger works both as a source-level debugger and a machine-level debugger. Other built-in tools include a forms designer for building GUI applications, web designer, class designer, and database schema designer. It accepts plug-ins that enhance the functionality at almost every level—including adding support for source control systems (like Subversion) and adding new toolsets like editors and visual designers for domain-specific languages or toolsets for other aspects of the software development lifecycle (like the Team Foundation Server client: Team Explorer).



```
<body>
    <form id="form1" runat="server">
    <div>
    <p>This is a test</p>
    </div>
        <asp:Image ID="Image1" runat="server" Height="112px"
        <p>
            <asp:Table ID="Table1" runat="server" Height="86
            </asp:Table>
        </p>
    </form>
```

Table1 System.Web.UI.WebControls

Width 231px

(ID) Table1

Rows (Collection)

Rows

The collection of rows within the table.

Use the arrow keys to resize by large increments. Ln 21 Col 25 Ch 25 INS

Fig 4.23 Visual Studio Platform

Visual Studio supports different programming languages and allows the code editor and debugger to support (to varying degrees) nearly any programming language, provided a language-specific service exists. Built-in languages include C,C++ and C++/CLI (via Visual C++), VB.NET (via Visual Basic .NET), C# (via Visual C#), and F# (as of Visual Studio 2010). Support for other languages such as Python, Ruby, Node.js, and M among others is available via language services installed separately .It also supports XML/XSLT, HTML/XHTML, JavaScript and CSS.

In this project Visual Studio is used to program the robotic vehicle movement and the fire detection.

5. SIMULATION

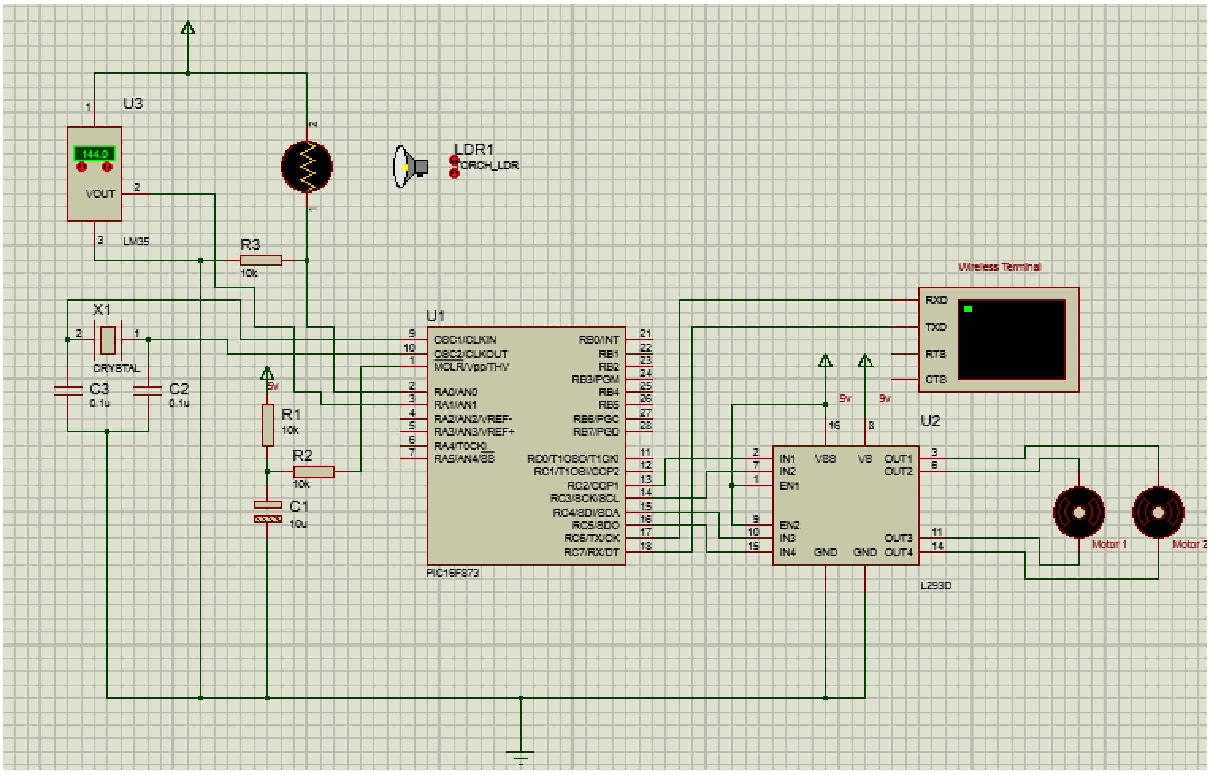


Fig 5.1 Simulation Results

6. ADVANTAGES

1. Continuous monitoring of temperature so that the fire can be detected at the earliest.
2. SMS alert so that the concerned person gets alerted at the right time.
3. The robot can move in all directions and is easy to control.
4. Live video streaming.
5. Reliable and economical.
6. Reduces risk to life.
7. Reduces human work.
8. Minimization of financial loss.

7. FUTURE SCOPE

The project is a prototype version of a fire fighting robot. So a lot of changes can be made to the current model so that the fire detection and extinguishing can be made more accurate and efficient. In the present condition it can extinguish fire only in the room in which it is placed. It can be extended to a real fire extinguisher by making it to extinguish fires of all the room using microprogramming.

Also the robot could not be run through the batteries because at some conditions the current requirement for the circuit rises to about .8A which is very high and cannot be obtained using batteries.

- The robot can be made autonomous.
- Zigbee or wifi module can be placed instead of Bluetooth for better operating range.
- GPRS module can be added.
- Smoke and colour sensor can be added to increase the accuracy of fire detection.
- A more efficient fire extinguisher system can be employed.
- Robot can be controlled through voice recognition system.
- Improve weight capacity of the robot.

8.CONCLUSION

The project explains how we can extinguish fire and how the damages caused by fire accidents can be avoided easily. The robot we have designed has many advantages and features. Minimum human intervention is needed for the operation of the robot. It stops the spreading of fire effectively. It can be reprogrammed easily to add modifications. It can be used in thermal stations, chemical industries, industrial plants, petrochemical industries to detect leakage of combustible gases during fire accidents to extinguish fire.. It can be programmed to operate in hazardous environments and underground mines where the entry of humans is dangerous to them. However, the robot designed has certain constraints. It cannot function completely autonomously without any human intervention. It can extinguish or fight fire for a small amount of time until human fire fighters arrive. It can detect fire only in certain locations. We can add further modifications in the robot for future purposes. The robot can be designed to avoid obstacles in its path by using IR or Ultrasonic transmitters and receivers. It can also be used to detect bombs by fitting bomb sensors in the robot. It will be a safest mode of operation by which many disasters can be prevented without damage. These robots will be vital for our day today life. Thus this robot can be used in place of a human therby reducing the threat ti life. The robot can be used to detect the flame more efficiently.

9.REFERENCES

[1]Jagan.M, Vinitha.N, Ananth.R, ”Voice operated intelligent elevator”ijetr.org,Volume 1, Issue 4, May-June, 2014

[2] AboliGatane,AratiDalvi,PoojaArkal,Prof.S.M.Jagdale ,” Using Speech Recognition Create Smart Elevator Controlling”,irjet,Volume 3,Issue 3,March,2016

[3]Sangole.K.Mosam, Bhamare.H.Yogita,Gangurde.B.Kanchan,Shejwal.S.Pankaj, “Voice Operated Elevator with Emergency Indicator”,ijarcsse.org,Volume 5,Issue 3,March 2015

[4] KashyapPatel,R.K.Prasad, “Speech Recognition and Verification Using MFCC & VQ”ijarcsse.org, Volume 3,Issue 5,May 2013

[5] Preeti Saini, Parneet Kaur, “Automatic Speech Recognition”,ijett,Volume 4, Issue 2, 2013

APPENDIX

```
////////////////////////////////////////////////////////////////////////////////////////////////////
               /* HEADER FILES, CRYSTAL FREQUENCY AND IC CONFIGURATION */
////////////////////////////////////////////////////////////////////////////////////////////////////

#include <htc.h>

#define _XTAL_FREQ 4000000
#define BAUDRATE 9600

////////////////////////////////////////////////////////////////////////////////////////////////////
                              /* FUNCTION PROTOTYPES */
////////////////////////////////////////////////////////////////////////////////////////////////////

void ADC_Init();
unsigned int ADC_Read(unsigned char channel);

void InitUART(void);
void SendByteSerially(unsigned char Byte);
void SendStringSerially(const unsigned char *st);
unsigned char ReceiveByteSerially(void);

void left(void);
void right(void);
void front(void);
void back(void);
void stop(void);

void camera_on(void);
void camera_off(void);
void extinguisher_on(void);
void extinguisher_off(void);

////////////////////////////////////////////////////////////////////////////////////////////////////
                                  /* DECLARATION */
////////////////////////////////////////////////////////////////////////////////////////////////////

int x=0, y=0, xref=0, yref=0;
char mem[6]="000000", u;
int j=0;

////////////////////////////////////////////////////////////////////////////////////////////////////
                                 /* MAIN FUNCTION */
////////////////////////////////////////////////////////////////////////////////////////////////////

void interrupt isr()
{
        mem[1]=ReceiveByteSerially();        //The recieved byte is saved on the first memory location
of                                           the array.
        if(mem[1]=='*')                      //checks if the recieved byte was *.
        mem[2]=ReceiveByteSerially();        //The recieved byte is saved on the second memory location
of                                           the array.
        mem[3]=ReceiveByteSerially();        //The recieved byte is saved on the third memory location
of                                           the array.
        mem[4]=ReceiveByteSerially();        //The recieved byte is saved on the fourth memory location
of                                           the array.
        mem[5]=ReceiveByteSerially();        //The recieved byte is saved on the fifth memory location
of                                           the array.
        mem[6]=ReceiveByteSerially();        //The recieved byte is saved on the sixth memory location
of                                           the array.

        while(!TXIF);                        //When the USART transmit buffer is empty
        TXREG=mem[2];                        //The byte saved on 2nd memory location is sent back to
the                                          bluetooth module.
        __delay_ms(15);                      //15 millisecond delay for stabilizing the UART module.
        while(!TXIF);                        //When the USART transmit buffer is empty
        TXREG=mem[3];                        //The byte saved on 3rd memory location is sent back to
the                                          bluetooth module.
        __delay_ms(15);                      //15 millisecond delay for stabilizing the UART module.
        while(!TXIF);                        //When the USART transmit buffer is empty
        TXREG=mem[4];                        //The byte saved on 4th memory location is sent back to
the                                          bluetooth module.
```

```
        __delay_ms(15);                      //15 millisecond delay for stabilizing the UART module.
        while(!TXIF);                        //When the USART transmit buffer is empty
        TXREG=mem[5];                        //The byte saved on 5th memory location is sent back to
the                                          bluetooth module.
        __delay_ms(15);                      //15 millisecond delay for stabilizing the UART module.

        u=mem[6];                            //The byte saved on 6th memory location is transfered to
                                                    variable u.
        while(!TXIF);                        //When the USART transmit buffer is empty
        TXREG=mem[1];                        //The byte saved on 1st memory location is sent back to
the                                          bluetooth module.
        __delay_ms(15);                      //15 millisecond delay for stabilizing the UART module.
}

//////////////////////////////////////////////////////////////////////////////////////////////////

void main(void)
{
        TRISA=0xff;                          //Sets all PORTA pins as input
        TRISB=0x00;                          //Sets all PORTB pins as output
        TRISC=0x00;                          //Sets all PORTC pins as output

        PORTB=0x00;                          //Clear all PORTB pins
        PORTC=0x00;                          //Clear all PORTC pins

        ADC_Init();                          //Initializethe       ADC module
        InitUART();                          //Initializethe       UART module

        xref = ADC_Read(1);                  //Sets reference value for temparature
        yref = ADC_Read(0);                  //Sets reference value for light

        INTCON=0xC0;                         //Enables the interrupt functionality of the
microcontroller
        CREN=1;                              //Enables Continuous Reception of data.
        RCIE=1;                              //Enables USART Receive Interrupt

        while(1)
        {
               __delay_ms(100);              //100 millisecond delay.
               x = ADC_Read(1);              //Reads data from ADC pin RA1
               y = ADC_Read(0);              //Reads data from ADC pin RA2

               if((x>(xref+5))&&(y>(yref+5)))//Confition to check if both temparature and intensity of
light is raising
               {
                      j++;                   //increments a variable for verifying the if statement
again.
                      if(j==10)              //when the if statement beocomes true for 10 times.
                      {
                              SendStringSerially("F*");            //Fire alert (F) is sent to
bluetooth.
                              j=0;                                 //clears the variable.
                      }
               }

               if((x<(xref+65))&&(y<(yref+125)))//Confition to check if both temparature and intensity
of light is normal
               {
                      j++;                   //increments a variable for verifying the if statement
again.
                      if(j==10)              //when the if statement beocomes true for 10 times.
                      {
                              SendStringSerially("N*");            //No fire alert (N) is sent to
bluetooth.
                              j=0;                                 //clears the variable.
                      }
               }


               if(u=='W'||u=='w')            //Checks if recieved data is W
               {
```

```
                        front();                //function for moving forward
                }
                if(u=='S'||u=='s')              //Checks if recieved data is W
                {
                        back();                 //function for moving backward
                }
                if(u=='A'||u=='a')              //Checks if recieved data is W
                {
                        left();                 //function for moving left
                }
                if(u=='D'||u=='d')              //Checks if recieved data is W
                {
                        right();                //function for moving right
                }
                if(u=='Q'||u=='q')              //Checks if recieved data is Q
                {
                        stop();                 //function for stopping the vehicle
                }
                if(u=='E'||u=='e')              //Checks if recieved data is E
                {
                        camera_on();            //function for turning the camera on
                }
                if(u=='R'||u=='r')              //Checks if recieved data is R
                {
                        camera_off();           //function for turning the camera off
                }
                if(u=='H'||u=='h')              //Checks if recieved data is F
                {
                        extinguisher_on();      //function for turning the extinguisher on
                }
                if(u=='G'||u=='g')              //Checks if recieved data is G
                {
                        extinguisher_off();     //function for turning the extinguisher on
                }

        }
}

////////////////////////////////////////////////////////////////////////////////////////////////////
                              /* FUNCTION DEFINITIONS */
////////////////////////////////////////////////////////////////////////////////////////////////////

void ADC_Init(void)`
{
        ADCON0 = 0x41;                          //ADC module configuration
        ADCON1 = 0xC0;                          //ADC port configuration
}

unsigned int ADC_Read(unsigned char channel)
{
        if(channel > 7)                         //we have only 8 (0-7) ADC. So, hannel number >7 is
ignored.
        return 0;                               //returns 0 if channel number recieved is > 7.

        ADCON0 &= 0xC5;                         //Bitwise AND Operation Clearing the Channel Selection
Bits
        ADCON0 |= channel<<3;                   //Bitwise OR Operation for Setting the required Bits
        __delay_ms(2);                          //2 millisecond delay for stabilizing the ADC module.
        GO_nDONE = 1;                           //Initializes A/D Conversion
        while(GO_nDONE);                        //Wait for A/D Conversion to complete
        return ((ADRESH<<8)+ADRESL);            //Returns Result
}

void InitUART(void)
{
        TRISC6 = 1;                             // TX Pin
        TRISC7 = 1;                             // RX Pin

        SPBRG = ((_XTAL_FREQ/16)/BAUDRATE) - 1;
        BRGH  = 1;                              // Fast baudrate
        SYNC  = 0;                              // Asynchronous
        SPEN  = 1;                           // Enable serial port pins
        CREN  = 1;                           // Enable reception
        SREN  = 0;                           // No effect
        TXIE  = 0;                           // Disable tx interrupts
        RCIE  = 1;                           // Enable rx interrupts
        TX9   = 0;                           // 8-bit transmission
        RX9   = 0;                           // 8-bit reception
        TXEN  = 0;                           // Reset transmitter
        TXEN  = 1;                           // Enable the transmitter
}

void SendByteSerially(unsigned char Byte)    // Writes a character to the serial port
{
        while(!TXIF);                        // wait for previous transmission to finish
        TXREG = Byte;
}

void SendStringSerially(const unsigned char *st)
{
        while(*st)
        SendByteSerially(*st++);
}

unsigned char ReceiveByteSerially(void)      // Reads a character from the serial port
{
        if(OERR)                             // If over run error, then reset the receiver
        {
                CREN = 0;
                CREN = 1;
        }

        while(!RCIF);                        // Wait for transmission to receive

        return RCREG;
}

void left(void)
{
        PORTC=0x14;                          //0001 0100
}

void back(void)
{
        PORTC=0x30;                          //0011 0000
}
void right(void)
{
        PORTC=0x28;                          //0010 1000
}

void front(void)
{
        PORTC=0x0C;                          //0000 1100
}

void stop(void)
{
        PORTC=0x00;                          //0000 0000
}

void camera_on(void)
{
        RB5=1;                               //Control PIN for camera is turned on.
}

void camera_off(void)
{
        RB5=0;                               //Control PIN for camera is turned off.
}

void extinguisher_on(void)
```

```
{
        RB4=1;                                //Control PIN for extinguisher is turned on.
}

void extinguisher_off(void)
{
        RB4=0;                                //Control PIN for extinguisher is turned on.
}

////////////////////////////////////////////////////////////////////////////////////////////////////
                                      /* THE END */
////////////////////////////////////////////////////////////////////////////////////////////////////
```

DATAMONITOR.CS

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace FireMon
{
    public partial class DataMonitor : Form
    {
        ServerDataAccess oServerDataAccess = new ServerDataAccess();
        CommDataWrapper _CommDataWrapper = new CommDataWrapper();
        Listener l;
        public string prevParam = "";
        public DataMonitor()
        {
            InitializeComponent();
            LoadAvailablePortNames();
            _CommDataWrapper.DataStreamSeparatorString = "*";
            l = new Listener(this);
            l.Subscribe(_CommDataWrapper);
        }

        private void sportFire_DataReceived(object sender,
System.IO.Ports.SerialDataReceivedEventArgs e)
        {
            _CommDataWrapper.ReadData(sportFire.ReadExisting());
        }

        public string AppendedString(string rawString, int totalDigits)
        {
            int stringLength = rawString.Length;
            int charCounter = 0;
            string outputString = "";
            string apnd = "";
            if (stringLength == totalDigits)
            {
                outputString = rawString;
            }
            else if (stringLength < totalDigits)
            {
                while (charCounter < totalDigits - stringLength)
                {
                    apnd = apnd + "0";
                    charCounter++;
                }
                outputString = apnd + rawString;

            }
            else
            {
                outputString = rawString.Substring(0, totalDigits);
            }
            return outputString;
        }

        private void timerPollParam_Tick(object sender, EventArgs e)
        {
            timerPollParam.Enabled = false;
            string paramID="", sParam="";
            string sqlQuery = "";
            string sentValue = "";
            DataTable dtTemp=new DataTable();
            sqlQuery = "select ParamID,SParam from ServerParam where Status=0
order by ParamID asc";
            oServerDataAccess.OpenConnection();
            dtTemp = oServerDataAccess.SelectMultiData(sqlQuery);
            oServerDataAccess.CloseConnection();
            if (dtTemp.Rows.Count > 0)
            {
                sParam = dtTemp.Rows[0]["SParam"].ToString();
                paramID = dtTemp.Rows[0]["ParamID"].ToString();
                dtTemp.Dispose();
                paramID = AppendedString(paramID, 4);
                sentValue = "*" + paramID + sParam;
                if (prevParam.Trim().Length > 0)
                {
                    if (prevParam == "W" && sParam == "S")
                    {
                        sportFire.Write(sentValue);
                    }
                    if (prevParam == "S" && sParam == "W")
                    {
                        sportFire.Write(sentValue);
                    }
                    if (prevParam == "A" && sParam == "D")
                    {
                        sportFire.Write(sentValue);
                    }
                    if (prevParam == "D" && sParam == "A")
                    {
                        sportFire.Write(sentValue);
                    }
                    txtSentLog.Text = txtSentLog.Text + sentValue +
Environment.NewLine;
                    Application.DoEvents();
                    System.Threading.Thread.Sleep(300);
                }
                prevParam = sParam;
                sportFire.Write(sentValue);
                Application.DoEvents();
                txtSentLog.Text = txtSentLog.Text + sentValue +
Environment.NewLine;
            }
            timerPollParam.Enabled = true;

        }
        private void LoadAvailablePortNames()
        {
            foreach (string portName in
System.IO.Ports.SerialPort.GetPortNames())
            {
                cboPortName.Items.Add(portName);
            }
            if (cboPortName.Items.Count > 0)
            {
                cboPortName.SelectedIndex = cboPortName.Items.Count - 1;
                //btnConnect.Enabled = true;
            }
        }
        private void btnConnectSerialPort_Click(object sender, EventArgs e)
        {
            if (sportFire.IsOpen)
            {

                sportFire.Close();

            }
            sportFire.PortName = cboPortName.Text;
            sportFire.Open();
            timerPollParam.Enabled = true;
            label1.Text = "Connected";
            //WritePort("c");
        }

        private void btnDisconnectSerialPort_Click(object sender, EventArgs
e)
        {
            sportFire.Close();
            timerPollParam.Enabled = false;
            label1.Text = "Ready to connect";
        }

        private void DataMonitor_Load(object sender, EventArgs e)
        {
            label1.Text = "Ready to connect";
        }
    }
}
```

OUTPUTSTRING.CS

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FireMon
{
    class OutputString : EventArgs
    {
        private string OutString;
        public string Content
        {
            get { return OutString; }
            set { OutString = value; }
        }
    }
}
```

RECEIVEDDATAHANDLER.CS

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.SqlClient;
using System.Data;

namespace FireMon
{
    class ReceivedDataHandler
    {
        public enum OperationMode
        {
            Fire,
            NoFire,
            AcknowledgementReceived

        }

        ServerDataAccess oServerDataAccess = new ServerDataAccess();
        public static OperationMode _OperationMode=OperationMode.NoFire;
        private static string _LastReceivedFile;
        private static string _CurrentFile;
        private static string _LastDeletedFile;
        DataMonitor _DataMonitor;

        public ReceivedDataHandler(DataMonitor fm)
        {
            _DataMonitor = fm;
        }

        public bool UpdateParamStatus(string paramID)
        {
            int transactionStatus = 0;
            int pID = 0;

            string sqlQuery = "";
            bool isParamInteger = int.TryParse(paramID,out pID);
            if (paramID.Length != 4)
            {
                return false;
            }
            else if (!isParamInteger)
            {
                return false;
            }
            else
            {
                pID = Convert.ToInt32(paramID);
                sqlQuery = "Update ServerParam set Status=1 where ParamID=" +
pID.ToString();
                oServerDataAccess.OpenConnection();
                oServerDataAccess.ExecuteSQL(sqlQuery);
                oServerDataAccess.CloseConnection();
                return true;
            }

        }

        public void ProcessReceivedData(string receivedData)
        {
            receivedData = receivedData.Replace("\r\n", string.Empty);
            switch (receivedData)
            {
                case "F":
                    if (_OperationMode == OperationMode.NoFire)
                    {
                        oServerDataAccess.SendSMS("7559961980");
                    }
                    _OperationMode = OperationMode.Fire;

                    break;

                case "N":
                    _OperationMode = OperationMode.NoFire;
                    break;
                default:
                    //_OperationMode = OperationMode.AcknowledgementReceived;
                    UpdateParamStatus(receivedData);
                    break;
            }
        }

         }


}
```

SERVERDATAACCESS.CS

```
using System;
using System.Data;
using System.Configuration;
using System.Linq;
using System.Web;
using System.Xml.Linq;
using System.Data.SqlClient;
using System.IO;
using System.Net.Mail;
using System.Net;
using System.Drawing;
using System.Text;
using System.Security.Cryptography;
```

```
public class ServerDataAccess
{

    SqlConnection _SqlConnection;
    SqlCommand _SqlCommand = new SqlCommand();
    SqlDataReader _SqlDataReader;
    SqlDataAdapter _SqlDataAdapter = new SqlDataAdapter();
    SqlTransaction _SqlTransaction;
    public enum SearchType
    {
        Exact,
        StartsWith,
        EndsWith,
        Anywhere
    }

    public ServerDataAccess(string connectionString)
    {
        _SqlConnection = new SqlConnection(connectionString);

    }
    public void RollbackTransation()
    {
        _SqlTransaction.Rollback();
    }

    public DataTable SelectMultiDataSP(string StoredProcedure)
    {
        DataTable dt = new DataTable();
        dt.Clear();
        _SqlCommand.CommandType = CommandType.StoredProcedure;
        _SqlCommand.CommandText = StoredProcedure;
        _SqlCommand.Connection = _SqlConnection;
        _SqlDataAdapter.SelectCommand = _SqlCommand;
        _SqlDataAdapter.Fill(dt);
        return dt;
    }

    public ServerDataAccess()
    {
        _SqlConnection = new
SqlConnection("Server=182.18.157.67;user=bluea_topkhd;password=asUdy7*U78di;D
atabase=blueapple_topkeralahd;");
        //_SqlConnection = new SqlConnection("Initial Catalog=FireMon;Data
Source=.;Integrated Security=SSPI;");
    }


    public void OpenConnection()
    {
        if (_SqlConnection.State != ConnectionState.Closed)
        {
            _SqlConnection.Close();
        }
        _SqlConnection.Open();
       //  _SqlConnection.Open();
    }
    public void CloseConnection()
    {
        _SqlConnection.Close();
    }

    public void BeginTransaction()
    {
        _SqlTransaction = _SqlConnection.BeginTransaction();
        _SqlCommand.Transaction = _SqlTransaction;
    }

    public void CommitTransaction()
    {
        _SqlTransaction.Commit();
    }

    public void RollbackTransaction()
    {
        _SqlTransaction.Rollback();
    }

    public bool ExecuteSQL(string sqlQuery)
    {
        try
        {

            _SqlCommand.CommandType = CommandType.Text;
            _SqlCommand.Connection = _SqlConnection;
            _SqlCommand.CommandText = sqlQuery;
            _SqlCommand.ExecuteNonQuery();
            return true;
        }
        catch (Exception)
        {
            return false;
        }

    }


    public bool ExecuteSP(string storedProcedure, SqlParameter[]
_SqlParameters)
    {
        try
         {
             _SqlCommand.Parameters.Clear();
            _SqlCommand.CommandType = CommandType.StoredProcedure;
            _SqlCommand.CommandText = storedProcedure;
            _SqlCommand.Connection = _SqlConnection;
            _SqlCommand.Parameters.AddRange(_SqlParameters);
            _SqlCommand.ExecuteNonQuery();
            _SqlCommand.Parameters.Clear();
            return true;
        }
         catch (Exception e)
        {
            //WebMsgBox1.Show(e.Message);
            return false;
        }
    }
    public bool ExecuteSP(string storedProcedure)
    {
        try
        {
            _SqlCommand.CommandType = CommandType.StoredProcedure;
            _SqlCommand.CommandText = storedProcedure;
            _SqlCommand.Connection = _SqlConnection;
            _SqlCommand.ExecuteNonQuery();
            _SqlCommand.Parameters.Clear();
            return true;
        }
        catch (Exception)
        {
            return false;
        }
    }

     public object SelectSingleData(string sqlQuery)
    {
        _SqlConnection.Close();
        object oSelectedData;
        _SqlCommand.CommandType = CommandType.Text;
        _SqlCommand.Connection = _SqlConnection;
        _SqlCommand.CommandText = sqlQuery;
        _SqlConnection.Open();
        _SqlDataReader = _SqlCommand.ExecuteReader();

        oSelectedData = "";
        while (_SqlDataReader.Read())
        {
            oSelectedData = _SqlDataReader[0];
            return (oSelectedData);
        }
        return (oSelectedData);
        _SqlConnection.Close();
    }
     public DataTable SelectMultiData(string sqlQuery)
     {
         DataTable objSelectedRows = new DataTable();
         _SqlCommand.CommandType = CommandType.Text;
         _SqlCommand.Connection = _SqlConnection;
         _SqlCommand.CommandText = sqlQuery;
         _SqlDataAdapter.SelectCommand = _SqlCommand;
         _SqlDataAdapter.Fill(objSelectedRows);
         return objSelectedRows;
     }
    public DataTable SelectMultiData(string storedProcedure, SqlParameter[]
_SqlParameters)
    {
        DataTable objSelectedRows = new DataTable();
        _SqlCommand.CommandType = CommandType.StoredProcedure;
        _SqlCommand.Connection = _SqlConnection;
        _SqlCommand.Parameters.AddRange(_SqlParameters);
        _SqlCommand.CommandText = storedProcedure;
        _SqlDataAdapter.SelectCommand = _SqlCommand;
        _SqlDataAdapter.Fill(objSelectedRows);
        _SqlCommand.Parameters.Clear();
        return objSelectedRows;
    }

    public bool DataExistsInDB(string sqlQuery)
    {
        _SqlCommand.CommandType = CommandType.Text;
        _SqlCommand.Connection = _SqlConnection;
        _SqlCommand.CommandText = sqlQuery;
        _SqlDataReader = _SqlCommand.ExecuteReader();
        bool vStatus = _SqlDataReader.Read();
        _SqlDataReader.Close();
        return vStatus;
    }
    public bool DataExistsInDBsp(string storedProcedure, SqlParameter[]
_SqlParameters)
    {
        _SqlCommand.CommandType = CommandType.StoredProcedure;
        _SqlCommand.Connection = _SqlConnection;
        _SqlCommand.Parameters.AddRange(_SqlParameters);
        _SqlCommand.CommandText = storedProcedure;
        _SqlDataReader = _SqlCommand.ExecuteReader();
        bool vStatus = _SqlDataReader.Read();
        return vStatus;
    }
    public string GetNextID(string fieldName, string tableName)
    {
        long vID;
        _SqlCommand.CommandType = CommandType.Text;
        _SqlCommand.Connection = _SqlConnection;
        _SqlCommand.CommandText = "SELECT MAX(" + fieldName + ") FROM " +
tableName;
        _SqlDataReader = _SqlCommand.ExecuteReader();
        if (_SqlDataReader.Read())
        {
            //Convert.ToUInt16(_SqlDataReader[0].ToString());
            string vTempData;
            vTempData = _SqlDataReader[0].ToString();
            if (vTempData.Length == 0)
            {
                return ("1");
            }
            else
            {
                vID = long.Parse(vTempData);
                vID = vID + 1;
                return (vID.ToString());
            }
        }
        else
        {
            return ("1");
        }
    }
    public bool IsReferenceExist(string valueToCompare, string
fieldToCompare, string tableName, string whereClause)
    {
        string sql;
        sql = "select " + fieldToCompare + " from " + tableName + " where " +
fieldToCompare + "=" + valueToCompare + " where " + whereClause;
        return DataExistsInDB(sql);
    }


    public bool IsReferenceExist(string valueToCompare, string
fieldToCompare, string commaSeperatedTables)
    {
        int counter = 0;
        string sql;
        string[] checkTables;
        checkTables = commaSeperatedTables.Split(',');
        while (counter < checkTables.Length)
        {
            sql = "select " + fieldToCompare + " from " +
checkTables[counter] + " where " + fieldToCompare + "=" + valueToCompare;
            counter++;
            if (DataExistsInDB(sql))
            {
                return true;
            }
        }
        return false;
    }

    public DataTable SearchMultiField(string selectFieldsCommaSeperated,
string searchValue, string searchTable, string searchFieldsCommaSeperated,
SearchType searchType)
    {
        int counter = 0;
        string sqlQuery = "";
        string[] searchFields, selectFields;
        DataTable dtTemp = new DataTable();
        selectFields = selectFieldsCommaSeperated.Split(',');
        searchFields = searchFieldsCommaSeperated.Split(',');
        sqlQuery = "select ";
        while (counter < selectFields.Length)
        {
            sqlQuery = sqlQuery + selectFields[counter] + ",";
            counter++;
        }
        counter = 0;
        sqlQuery = sqlQuery.TrimEnd(',') + "  from " + searchTable + " where
";

        while (counter < searchFields.Length)
        {
            if (searchType == SearchType.Exact)
            {
                sqlQuery = sqlQuery + searchFields[counter] + " = '" +
searchValue + "' or ";
            }
            else if (searchType == SearchType.StartsWith)
            {
                sqlQuery = sqlQuery + searchFields[counter] + " like '" +
searchValue + "%' or ";
            }
            else if (searchType == SearchType.EndsWith)
            {
                sqlQuery = sqlQuery + searchFields[counter] + " like '%" +
searchValue + "' or ";
            }
            else if (searchType == SearchType.Anywhere)
            {
                sqlQuery = sqlQuery + searchFields[counter] + " like '%" +
searchValue + "%' or ";
            }
            counter++;
        }
        sqlQuery = sqlQuery.Substring(0, sqlQuery.Length - 4);
        dtTemp = SelectMultiData(sqlQuery);
        return dtTemp;
    }

    public DataTable SearchMultiField(string selectFieldsCommaSeperated,
string searchValue, string searchTable, string searchFieldsCommaSeperated,
string whereClause, SearchType searchType)
    {
        int counter = 0;
        string sqlQuery = "";
        string[] searchFields, selectFields;
        DataTable dtTemp = new DataTable();
        selectFields = selectFieldsCommaSeperated.Split(',');
        searchFields = searchFieldsCommaSeperated.Split(',');
        sqlQuery = "select ";
        while (counter < selectFields.Length)
        {
            sqlQuery = sqlQuery + selectFields[counter] + ",";
            counter++;
        }
        counter = 0;
        sqlQuery = sqlQuery.TrimEnd(',') + "  from " + searchTable + " where
(";

        while (counter < searchFields.Length)
        {
            if (searchType == SearchType.Exact)
            {
                sqlQuery = sqlQuery + searchFields[counter] + " = '" +
searchValue + "' or ";
            }
            else if (searchType == SearchType.StartsWith)
            {
                sqlQuery = sqlQuery + searchFields[counter] + " like '" +
searchValue + "%' or ";
            }
            else if (searchType == SearchType.EndsWith)
            {
                sqlQuery = sqlQuery + searchFields[counter] + " like '%" +
searchValue + "' or ";
            }
            else if (searchType == SearchType.Anywhere)
            {
                sqlQuery = sqlQuery + searchFields[counter] + " like '%" +
searchValue + "%' or ";
            }
            counter++;
        }
        sqlQuery = sqlQuery.Substring(0, sqlQuery.Length - 4);
        sqlQuery = sqlQuery + ") and " + whereClause;
        dtTemp = SelectMultiData(sqlQuery);
        return dtTemp;
    }

    public void SendSMS(string toNumber)
    {
        HttpWebRequest objHTTPWReq =
(HttpWebRequest)WebRequest.Create("http://sms.blueapplelabs.com/api/sendmsg.p
hp?user=abc321&pass=demo1&sender=BUSSMS&phone="+ toNumber +"&text=Alert!!!!
Fire Detected&priority=ndnd&stype=normal");
        HttpWebResponse objHTTPWRes =
(HttpWebResponse)objHTTPWReq.GetResponse();
        objHTTPWRes.Close();
    }
    public string ValidData(string vData)
    {
        string vTemp = "";
        string vValidData = "";
        //char[] tmp;
        //char ct;
        string vValid =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXY0123456789";
        for (int i = 0; i < vData.Length; i++)
        {
            vTemp = vData.Substring(i, 1);
            //tmp = vTemp.ToCharArray();
            if (vValid.Contains(vTemp))
            {

                vValidData = vValidData + vTemp;
            }


        }
        return vValidData;
    }
    public byte[] StreamToByte(Stream oInputStream)
    {
        byte[] buffer = new byte[16 * 1024];
        using (MemoryStream ms = new MemoryStream())
        {
            int read;
            while ((read = oInputStream.Read(buffer, 0, buffer.Length)) > 0)
            {
                ms.Write(buffer, 0, read);
            }
            return ms.ToArray();
        }
    }

    public void send_email2(string smtp_host, string smtp_username, string 
smtp_password, string to, string to_name, string from, string from_name, 
string subject, string body, string myflg,string path)
    {
        try
        {
            SmtpClient SMTPClient = new SmtpClient(smtp_host);

            SMTPClient.EnableSsl = true;
            SMTPClient.Host = smtp_host;
            SMTPClient.Port = 587;
            MailAddress to_ = new MailAddress(to, to_name);
            MailAddress from_ = new MailAddress(from, from_name);


            MailMessage M = new MailMessage(from_, to_);
            M.Subject = subject;
            M.SubjectEncoding = System.Text.Encoding.UTF8;
            M.Body = body;
            M.IsBodyHtml = true;
            NetworkCredential NTLMAuthentication = new 
NetworkCredential(smtp_username, smtp_password);
            //M.BodyEncoding = System.Text.Encoding.UTF8;
            //M.IsBodyHtml = true;
            //SMTPClient.UseDefaultCredentials = false;
            SMTPClient.Credentials = NTLMAuthentication;
            if (myflg == "1")
            {
                Attachment attach = new Attachment(path);
                string fileName = System.IO.Path.GetFileName(path);
                attach.Name = fileName;
                M.Attachments.Add(attach);
            }
            SMTPClient.Send(M);

        }
        catch (Exception ex)
        {
            throw ex;
        }
    }
    public void send_email_html(string smtp_host, string smtp_username, 
string smtp_password, string to, string to_name, string from, string 
from_name, string subject, string body, string myflg)
    {
        try
        {
            SmtpClient SMTPClient = new SmtpClient(smtp_host);

            SMTPClient.EnableSsl = true;
            SMTPClient.Host = smtp_host;
            SMTPClient.Port = 587;
            MailAddress to_ = new MailAddress(to, to_name);
            MailAddress from_ = new MailAddress(from, from_name);


            MailMessage M = new MailMessage(from_, to_);
            M.Subject = subject;
            M.SubjectEncoding = System.Text.Encoding.UTF8;
            M.Body = body;
            NetworkCredential NTLMAuthentication = new
NetworkCredential(smtp_username, smtp_password);
            //M.BodyEncoding = System.Text.Encoding.UTF8;
            M.IsBodyHtml = true;
            //SMTPClient.UseDefaultCredentials = false;
            SMTPClient.Credentials = NTLMAuthentication;
            //if (myflg == "1")
            //{
            //    Attachment attach = new Attachment(Server.MapPath("") +
"//Admin//file//" + imgname1);
            //    attach.Name = imgname1;
            //    M.Attachments.Add(attach);
            //}
            SMTPClient.Send(M);
        }
        catch (Exception e)
        {

        }

    }
    public void Send_SMS(string AppleogUserName, string AppleogPassword,
string SenderID, string ToNumber, string SMSBody, int SMSPriority)
    {
        HttpWebRequest objHTTPWReq =
(HttpWebRequest)WebRequest.Create("http://www.appleog.com/messenger.aspx?user
name=" + AppleogUserName + "&password=" + AppleogPassword + "&sender=" +
SenderID + "&to=" + ToNumber + "&message=" + SMSBody + "&priority=" +
SMSPriority.ToString());
        HttpWebResponse objHTTPWRes =
(HttpWebResponse)objHTTPWReq.GetResponse();
        objHTTPWRes.Close();
    }

    //public string IpAddress(HttpRequest _Request)
    //{
    //    string strIpAddress;
    //    strIpAddress = _Request.ServerVariables["HTTP_X_FORWARDED_FOR"];
    //    if (strIpAddress == null)
    //    {
    //        strIpAddress = _Request.ServerVariables["REMOTE_ADDR"];
    //    }
    //    return strIpAddress;
    //}
    //public int FileSaveEx(HttpPostedFile objPostedFile, string
vUploadFolder, string vUploadedFileName, string vValidExtentions, bool
vOverWrite)
```

```
    //{
    //    /// Returns:
    //    ///1 -File Upload succeeded
    //    ///2 -Unexpected extension
    //    ///3 -Invalid argument(no valid alphabets)
    //    ///4 -File with same name exists
    //    ///5 -File uploaded, but not confirmed

    //    string vFilePath = objPostedFile.ToString();
    //    string vTempExt = "";
    //    string vValidExt = "";
    //    int vTempStartIndex;
    //    int vTempEndIndex;
    //    int vStatus = 0;
    //    vValidExtentions = vValidExtentions + ",";
    //    //vFilePath = objPostedFile.ToString();
    //    vTempExt =
Path.GetExtension(objPostedFile.FileName.ToString().ToLower());
    //    vTempExt = vTempExt.Replace(".", "");
    //    vTempStartIndex = 0;
    //    if (vFilePath.Trim().Length == 0 || vUploadFolder.Trim().Length ==
0 || vUploadedFileName.Trim().Length == 0 || vValidExtentions.Trim().Length
== 0)
    //    {
    //        return (3);

    //    }
    //    while (vTempStartIndex <= vValidExtentions.Length)
    //    {
    //        vTempEndIndex = vValidExtentions.IndexOf(",", vTempStartIndex);

    //        //vTemp = vData.Substring(i, 1);
    //        //tmp = vTemp.ToCharArray();
    //        if (vTempEndIndex == -1 || vTempStartIndex == -1)
    //        {
    //            break;
    //        }
    //        vValidExt = vValidExtentions.Substring(vTempStartIndex,
vTempEndIndex - vTempStartIndex);
    //        vValidExt = vValidExt.ToLower();
    //        if (vTempExt.Equals(vValidExt))
    //        {
    //            vStatus = 0;
    //            break;
    //        }
    //        else
    //        {
    //            vStatus = 2;

    //        }
    //        vTempStartIndex = vTempEndIndex + 1;
    //    }
    //    if (vStatus == 2)
    //    {
    //        return (2);
    //    }
    //    else if (File.Exists(vUploadFolder + "//" + vUploadedFileName + "."
+ vTempExt))
    //    {
    //        if (vOverWrite)
    //        {
    //            File.Delete(vUploadFolder + "//" + vUploadedFileName + "."
+ vTempExt);
    //            while (File.Exists(vUploadFolder + "//" + vUploadedFileName
+ "." + vTempExt))
    //            {
    //            }
    //            objPostedFile.SaveAs(vUploadFolder + "//" +
vUploadedFileName + "." + vTempExt);
    //            if (File.Exists(vUploadFolder + "//" + vUploadedFileName +
"." + vTempExt))
    //            {
    //                return (1);
    //            }
    //            else
    //            {
    //                return (5);
    //            }

    //        }
    //        else
    //        {
    //            return (4);
    //        }
    //    }
    //    else
    //    {
    //        objPostedFile.SaveAs(vUploadFolder + "//" + vUploadedFileName +
"." + vTempExt);
    //        if (File.Exists(vUploadFolder + "//" + vUploadedFileName + "."
+ vTempExt))
    //        {
    //            return (1);
    //        }
    //        else
    //        {
    //            return (5);
    //        }

    //    }

    //}
    //public string GetHash(string textToHash)
    //{
    //    string hash = "";
    //    SHA512 oSHA512 = SHA512.Create();
    //    byte[] result =
oSHA512.ComputeHash(Encoding.UTF8.GetBytes(textToHash));
    //    hash = Encoding.UTF8.GetString(result);
    //    return hash;
    //}}
```

LISTNER.CS

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace FireMon
{
    class Listener
    {
        DataMonitor _DataMonitor;
        ReceivedDataHandler oReceivedDataHandler;
        public Listener()
        {

        }
        public Listener(DataMonitor fm)
        {
            _DataMonitor = fm;
            oReceivedDataHandler = new ReceivedDataHandler(_DataMonitor);
        }

        public void Subscribe(CommDataWrapper commDataWrapper)
        {
            commDataWrapper.ReadComplete += new
CommDataWrapper.ReadCompleteHandler(DataOutputReceived);
        }
        private void DataOutputReceived(CommDataWrapper commDataWrapper,
OutputString e)
        {
            oReceivedDataHandler.ProcessReceivedData(e.Content);
            //_frmDataLogger.SaveData(e.Content);
            //_frmDataLogger.SetText(e.Content);
             }
    }
}
```

PROGRAM.CS

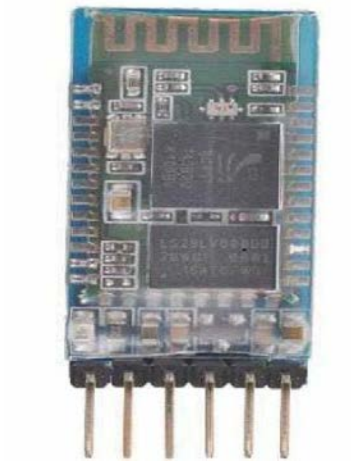
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

namespace FireMon
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new DataMonitor());
        }
    } }
```

DATASHEET

BLUETOOTH TO SERIAL PORT

MODULE HC05



Overview

HC-05 module is an easy to use Bluetooth SPP (Serial Port Protocol) module, designed for transparent wireless serial connection setup.

Serial port Bluetooth module is fully qualified Bluetooth V2.0+EDR (Enhanced Data Rate) 3Mbps Modulation with complete 2.4GHz radio transceiver and baseband. It uses CSR Bluecore 04-External single chip Bluetooth system with CMOS technology and with AFH (Adaptive Frequency Hopping Feature). It has the footprint as small as 12.7mmx27mm. Hope it will simplify your overall design/development cycle.

Specifications

Hardware features

Typical -80dBm sensitivity.

Up to +4dBm RF transmit power.

Low Power 1.8V Operation, 3.3

to 5 V I/O. PIO control.

UART interface with programmable

baud rate. With integrated antenna.

With edge connector.

Software features

Slave default Baud rate: 9600, Data bits:8, Stop bit:1,Parity:No parity. PIO9 and PIO8 can be connected to red and blue led separately. When

master and slave are paired, red and blue led blinks 1time/2s in interval, while disconnected only blue led blinks 2times/s.

Auto-connect to the last device on power as default.

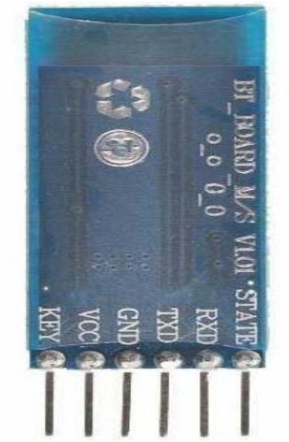
Permit pairing device to connect as

default. Auto-pairing

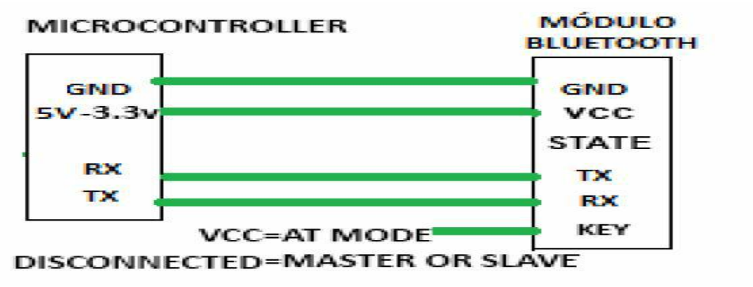
PINCODE:”1234” as default.

Auto-reconnect in 30 min when disconnected as a result of beyond the range of connection.

Pin out configuration



Typical Application Circuit



After connect the Bluetooth module, scan for new devices from the PC and you will find the module with the device name “HC-05”, after that, click to connect, if some message appears asking about “Pairing code” just put **“1234”** as default code.

National Semiconductor

November 2000

LM35
Precision Centigrade Temperature Sensors

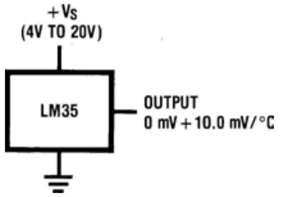
General Description

The LM35 series are precision integrated-circuit temperature sensors, whose output voltage is linearly proportional to the Celsius (Centigrade) temperature. The LM35 thus has an advantage over linear temperature sensors calibrated in Ê Kelvin, as the user is not required to subtract a large constant voltage from its output to obtain convenient Centi-grade scaling. The LM35 does not require any external calibration or trimming to provide typical accuracies of ±^{1}¤$_{4}$ÊC at room temperature and ±^{3}¤$_{4}$ÊC over a full −55 to +150ÊC temperature range. Low cost is assured by trimming and calibration at the wafer level. The LM35's low output impedance, linear output, and precise inherent calibration make interfacing to readout or control circuitry especially easy. It can be used with single power supplies, or with plus and minus supplies. As it draws only 60 µA from its supply, it has very low self-heating, less than 0.1ÊC in still air. The LM35 is rated to operate over a −55Ê to +150ÊC temperature range, while the LM35C is rated for a −40Ê to +110ÊC range (−10Ê with improved accuracy). The LM35 series is available packaged in hermetic TO-46 transistor packages, while the LM35C, LM35CA, and LM35D are also available in the plastic TO-92 transistor package. The LM35D is also available in an 8-lead surface mount small outline package and a plastic TO-220 package.

Features

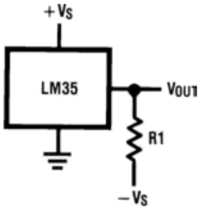
n Calibrated directly in Ê Celsius (Centigrade)
n Linear + 10.0 mV/ÊC scale factor
n 0.5ÊC accuracy guaranteeable (at +25ÊC)
n Rated for full −55Ê to +150ÊC range
n Suitable for remote applications
n Low cost due to wafer-level trimming
n Operates from 4 to 30 volts
n Less than 60 µA current drain
n Low self-heating, 0.08ÊC in still air
n Nonlinearity only ±^{1}¤$_{4}$ÊC typical
n Low impedance output, 0.1 Ω for 1 mA load

Typical Applications



DS005516-3

FIGURE 1. Basic Centigrade Temperature Sensor (+2ÊC to +150ÊC)



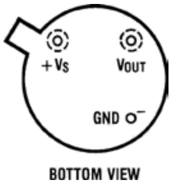
DS005516-4

Choose $R_1 = -V_S/50$ µA
V_{OUT}=+1,500 mV at +150ÊC
= +250 mV at +25ÊC
= −550 mV at −55ÊC

FIGURE 2. Full-Range Centigrade Temperature Sensor

Connection Diagrams

TO-46
Metal Can Package*

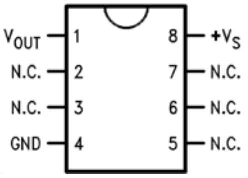


DS005516-1

*Case is connected to negative pin (GND)

Order Number LM35H, LM35AH, LM35CH, LM35CAH or LM35DH
See NS Package Number H03H

SO-8
Small Outline Molded Package

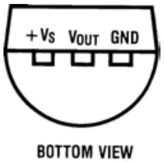


DS005516-21

N.C. = No Connection

Top View
Order Number LM35DM
See NS Package Number M08A

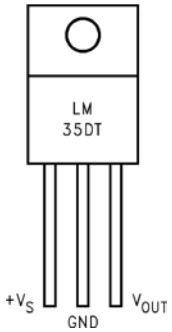
TO-92
Plastic Package



DS005516-2

Order Number LM35CZ, LM35CAZ or LM35DZ
See NS Package Number Z03A

TO-220
Plastic Package*



DS005516-24

*Tab is connected to the negative pin (GND).
Note: The LM35DT pinout is different than the discontinued LM35DP.

Order Number LM35DT
See NS Package Number TA03F

Absolute Maximum Ratings (Note 10)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/ Distributors for availability and specifications.

Supply Voltage	+35V to −0.2V
Output Voltage	+6V to −1.0V
Output Current	10 mA
Storage Temp.;	
TO-46 Package,	−60ÊC to +180ÊC
TO-92 Package,	−60ÊC to +150ÊC
SO-8 Package,	−65ÊC to +150ÊC
TO-220 Package,	−65ÊC to +150ÊC
Lead Temp.:	
TO-46 Package, (Soldering, 10 seconds)	300ÊC
TO-92 and TO-220 Package, (Soldering, 10 seconds)	260ÊC
SO Package (Note 12)	
Vapor Phase (60 seconds)	215ÊC
Infrared (15 seconds)	220ÊC
ESD Susceptibility (Note 11)	2500V
Specified Operating Temperature Range: T_{MIN} to T_{MAX} (Note 2)	
LM35, LM35A	−55ÊC to +150ÊC
LM35C, LM35CA	−40ÊC to +110ÊC
LM35D	0ÊC to +100ÊC

LM35

Electrical Characteristics

(Notes 1, 6)

Parameter	Conditions	LM35A			LM35CA			Units
		Typical	Tested Limit (Note 4)	Design Limit (Note 5)	Typical	Tested Limit (Note 4)	Design Limit (Note 5)	(Max.)
Accuracy (Note 7)	T_A=+25ÊC	±0.2	±0.5		±0.2	±0.5		ÊC
	T_A=−10ÊC	±0.3			±0.3		±1.0	ÊC
	$T_A=T_{MAX}$	±0.4	±1.0		±0.4	±1.0		ÊC
	$T_A=T_{MIN}$	±0.4	±1.0		±0.4		±1.5	ÊC
Nonlinearity (Note 8)	$T_{MIN}\leq T_A\leq T_{MAX}$	**± 0.18**		**± 0.35**	**± 0.15**		**± 0.3**	ÊC
Sensor Gain (Average Slope)	$T_{MIN}\leq T_A\leq T_{MAX}$	**+10.0**	**+9.9, +10.1**		**+10.0**		**+9.9, +10.1**	mV/ÊC
Load Regulation (Note 3) 0≤I_L≤1 mA	T_A=+25ÊC	±0.4	±1.0		±0.4	±1.0		mV/mA
	$T_{MIN}\leq T_A\leq T_{MAX}$	**± 0.5**		**± 3.0**	**± 0.5**		**± 3.0**	mV/mA
Line Regulation (Note 3)	T_A=+25ÊC	±0.01	±0.05		±0.01	±0.05		mV/V
	4V≤V_S≤30V	**± 0.02**		**± 0.1**	**± 0.02**		**± 0.1**	mV/V
Quiescent Current (Note 9)	V_S=+5V, +25ÊC	56	67		56	67		µA
	V_S=+5V	**105**		**131**	**91**		**114**	µA
	V_S=+30V, +25ÊC	56.2	68		56.2	68		µA
	V_S=+30V	**105.5**		**133**	**91.5**		**116**	µA
Change of Quiescent Current (Note 3)	4V≤V_S≤30V, +25ÊC	0.2	1.0		0.2	1.0		µA
	4V≤V_S≤30V	**0.5**		**2.0**	**0.5**		**2.0**	µA
Temperature Coefficient of Quiescent Current		**+0.39**		**+0.5**	**+0.39**		**+0.5**	µA/ÊC
Minimum Temperature for Rated Accuracy	In circuit of *Figure 1*, I_L=0	+1.5		+2.0	+1.5		+2.0	ÊC
Long Term Stability	$T_J=T_{MAX}$, for 1000 hours	±0.08			±0.08			ÊC

D **Featuring Unitrode L293 and L293D Products Now From Texas Instruments**

D **Wide Supply-Voltage Range: 4.5 V to 36 V**

D **Separate Input-Logic Supply**

D **Internal ESD Protection**

D **Thermal Shutdown**

D **High-Noise-Immunity Inputs**

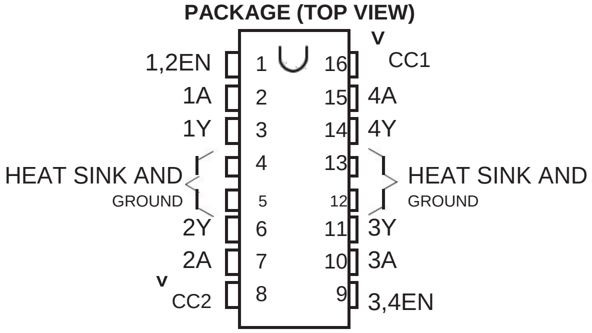
D **Functionally Similar to SGS L293 and SGS L293D**

D **Output Current 1 A Per Channel (600 mA for L293D)**

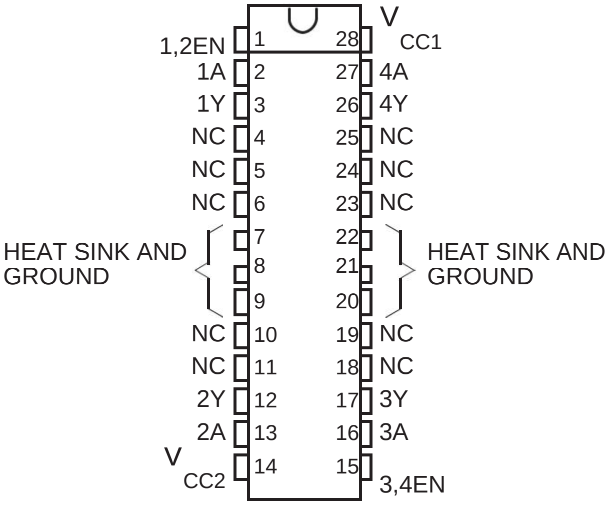
D **Peak Output Current 2 A Per Channel (1.2 A for L293D)**

D **Output Clamp Diodes for Inductive Transient Suppression (L293D)**

L293 . . . N OR NE PACKAGE L293D . . . NE PACKAGE (TOP VIEW)



L293 . . . DWP PACKAGE (TOP VIEW)



description/ordering information

The L293 and L293D are quadruple high-current half-H drivers. The L293 is designed to provide bidirectional drive currents of up to 1 A at voltages from 4.5 V to 36 V. The L293D is designed to provide bidirectional drive currents of up to 600-mA at voltages from 4.5 V to 36 V. Both devices are designed to drive inductive loads such as relays, solenoids, dc and bipolar stepping motors, as well as other high-current/high-voltage loads in positive-supply applications.

All inputs are TTL compatible. Each output is a complete totem-pole drive circuit, with a Darlington transistor sink and a pseudo-Darlington source. Drivers are enabled in pairs, with drivers 1 and 2 enabled by 1,2EN and drivers 3 and 4 enabled by 3,4EN. When an enable input is high, the associated drivers are enabled, and their outputs are active and in phase with their inputs. When the enable input is low, those drivers are disabled, and their outputs are off and in the high-impedance state. With the proper data inputs, each pair of drivers forms a full-H (or bridge) reversible drive suitable for solenoid or motor applications.

ORDERING INFORMATION

T_A	PACKAGE†		ORDERABLE PART NUMBER	TOP-SIDE MARKING
0°C to 70°C	HSOP (DWP)	Tube of 20	L293DWP	L293DWP
	PDIP (N)	Tube of 25	L293N	L293N
	PDIP (NE)	Tube of 25	L293NE	L293NE
		Tube of 25	L293DNE	L293DNE

† Package drawings, standard packing quantities, thermal data, symbolization, and PCB design guidelines are available at www.ti.com/sc/package.

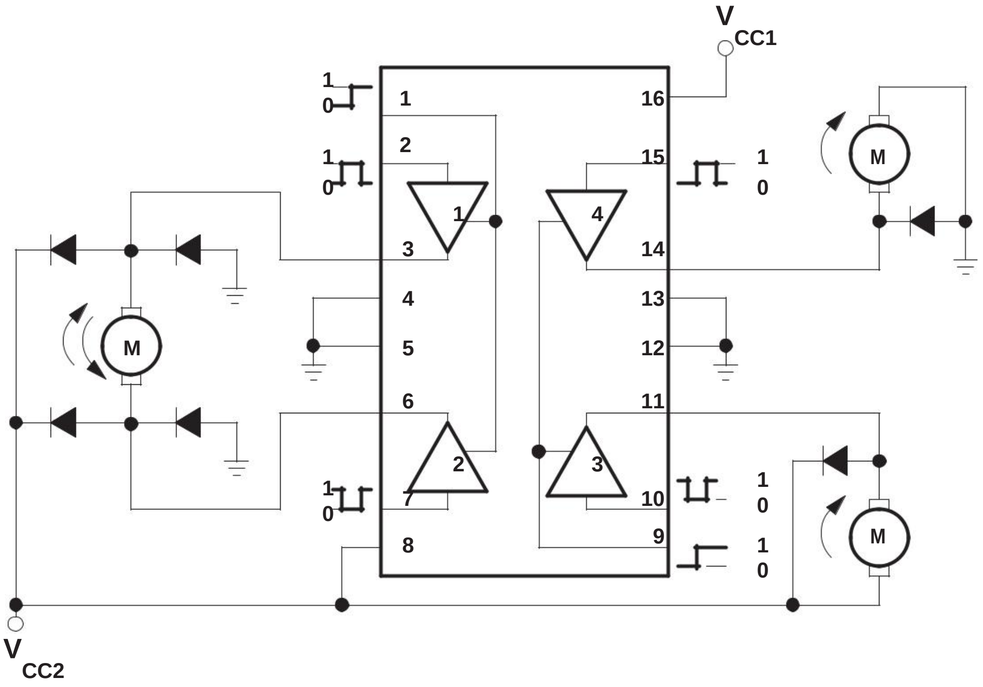
Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers thereto appears at the end of this data sheet.

description/ordering information (continued)

On the L293, external high-speed output clamp diodes should be used for inductive transient suppression.

A V_{CC1} terminal, separate from V_{CC2}, is provided for the logic inputs to minimize device power dissipation. The L293and L293D are characterized for operation from 0°C to 70°C.

block diagram



NOTE: Output diodes are internal in L293D.

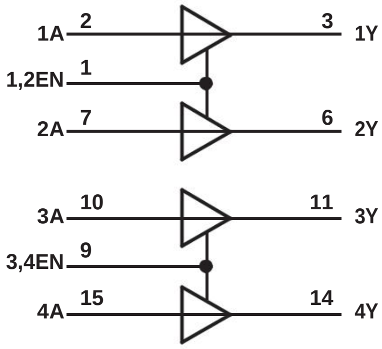
FUNCTION TABLE
(each driver)

INPUTS†		OUTPUT
A	EN	Y
H	H	H
L	H	L
X	L	Z

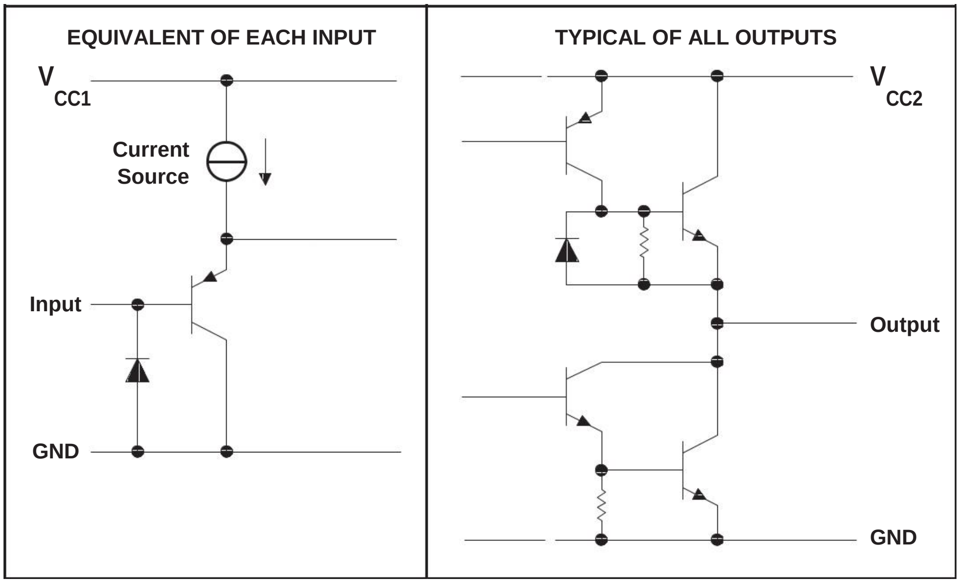
H = high level, L = low level, X = irrelevant,
Z = high impedance (off)

† In the thermal shutdown mode, the output is in the high-impedance state, regardless of the input levels.

logic diagram



schematics of inputs and outputs (L293)



PIC16F87XA
Data Sheet

28/40/44-Pin Enhanced Flash
Microcontrollers

IC16F87XA

28/40/44-Pin Enhanced Flash Microcontrollers

Devices Included in this Data Sheet:

- PIC16F873A
- PIC16F874A
- PIC16F876A
- PIC16F877A

High-Performance RISC CPU:

Only 35 single-word instructions to learn

All single-cycle instructions except for program branches, which are two-cycle

Operating speed: DC – 20 MHz clock input
DC – 200 ns instruction cycle

Up to 8K x 14 words of Flash Program Memory, Up to 368 x 8 bytes of Data Memory (RAM), Up to 256 x 8 bytes of EEPROM Data Memory

Pinout compatible to other 28-pin or 40/44-pin PIC16CXXX and PIC16FXXX microcontrollers

Peripheral Features:

Timer0: 8-bit timer/counter with 8-bit prescaler

Timer1: 16-bit timer/counter with prescaler, can be incremented during Sleep via external crystal/clock

Timer2: 8-bit timer/counter with 8-bit period register, prescaler and postscaler

Two Capture, Compare, PWM modules

Capture is 16-bit, max. resolution is 12.5 ns

Compare is 16-bit, max. resolution is 200 ns

PWM max. resolution is 10-bit

Synchronous Serial Port (SSP) with SPI™ (Master mode) and I^{2}C™ (Master/Slave)

Universal Synchronous Asynchronous Receiver Transmitter (USART/SCI) with 9-bit address detection

Parallel Slave Port (PSP) – 8 bits wide with external RD, WR and CS controls (40/44-pin only)

Brown-out detection circuitry for Brown-out Reset (BOR)

Analog Features:

10-bit, up to 8-channel Analog-to-Digital Converter (A/D)

Brown-out Reset (BOR)

Analog Comparator module with:

Two analog comparators

Programmable on-chip voltage reference (VREF) module

Programmable input multiplexing from device inputs and internal voltage reference

Comparator outputs are externally accessible

Special Microcontroller Features:

100,000 erase/write cycle Enhanced Flash program memory typical

1,000,000 erase/write cycle Data EEPROM memory typical

Data EEPROM Retention > 40 years

Self-reprogrammable under software control

In-Circuit Serial Programming™ (ICSP™) via two pins

Single-supply 5V In-Circuit Serial Programming

Watchdog Timer (WDT) with its own on-chip RC oscillator for reliable operation

Programmable code protection

Power saving Sleep mode

Selectable oscillator options

In-Circuit Debug (ICD) via two pins

CMOS Technology:

Low-power, high-speed Flash/EEPROM technology

Fully static design

Wide operating voltage range (2.0V to 5.5V)

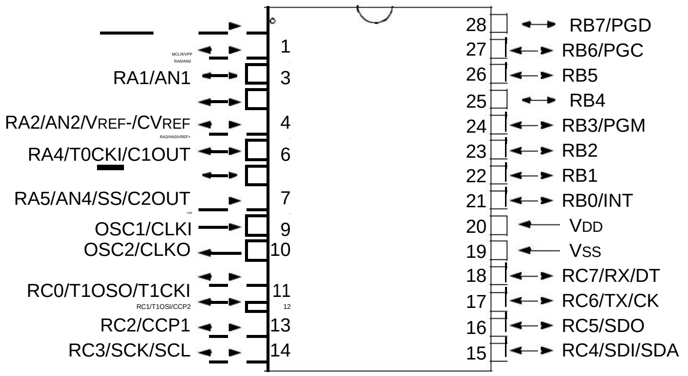
Commercial and Industrial temperature ranges

Low-power consumption

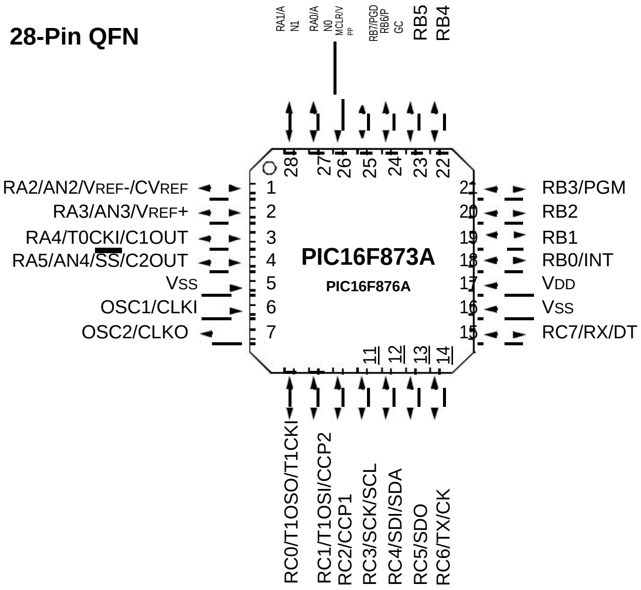
Device	Program Memory		Data SRAM (Bytes)	EEPROM (Bytes)	I/O	10-bit A/D (ch)	CCP (PWM)	MSSP		USART	Timers 8/16-bit	Comparators
	Bytes	# Single Word Instructions						SPI	Master I^{2}C			
PIC16F873A	7.2K	4096	192	128	22	5	2	Yes	Yes	Yes	2/1	2
PIC16F874A	7.2K	4096	192	128	33	8	2	Yes	Yes	Yes	2/1	2
PIC16F876A	14.3K	8192	368	256	22	5	2	Yes	Yes	Yes	2/1	2
PIC16F877A	14.3K	8192	368	256	33	8	2	Yes	Yes	Yes	2/1	2

Pin Diagrams

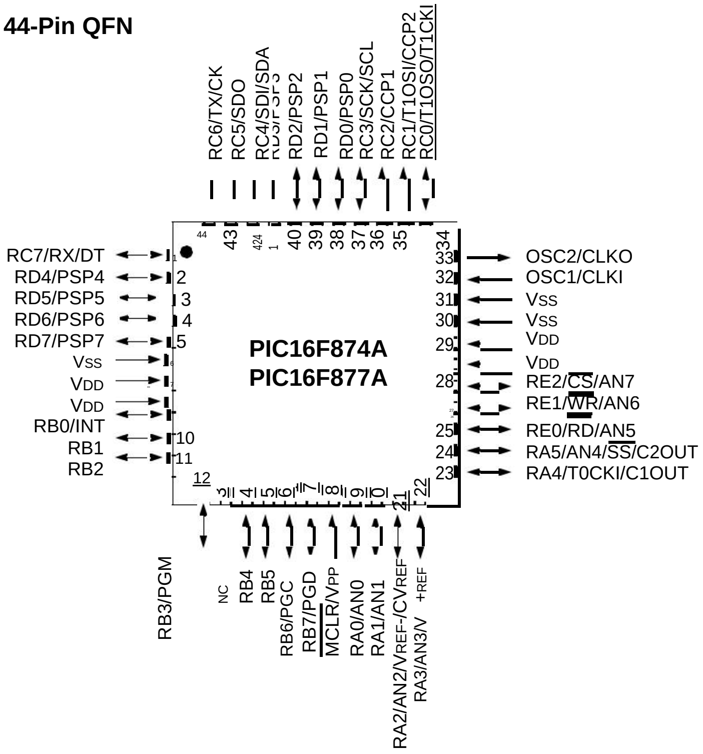
28-Pin PDIP, SOIC, SSOP



28-Pin QFN



44-Pin QFN



DEVICE OVERVIEW

This document contains device specific information about the following devices:

PIC16F873A

PIC16F874A

PIC16F876A

PIC16F877A

PIC16F873A/876A devices are available only in 28-pin packages, while PIC16F874A/877A devices are avail-able in 40-pin and 44-pin packages. All devices in the PIC16F87XA family share common architecture with the following differences:

The PIC16F873A and PIC16F874A have one-half of the total on-chip memory of the PIC16F876A and PIC16F877A

The 28-pin devices have three I/O ports, while the 40/44-pin devices have five

The 28-pin devices have fourteen interrupts, while the 40/44-pin devices have fifteen

The 28-pin devices have five A/D input channels, while the 40/44-pin devices have eight

The Parallel Slave Port is implemented only on the 40/44-pin devices

The available features are summarized in Table 1-1. Block diagrams of the PIC16F873A/876A and PIC16F874A/877A devices are provided in Figure 1-1 and Figure 1-2, respectively. The pinouts for these device families are listed in Table 1-2 and Table 1-3.

Additional information may be found in the PICmicro® Mid-Range Reference Manual (DS33023), which may be obtained from your local Microchip Sales Represen-tative or downloaded from the Microchip web site. The Reference Manual should be considered a complementary document to this data sheet and is highly recommended reading for a better understanding of the device architecture and operation of the peripheral modules.

TABLE 1-1: PIC16F87XA DEVICE FEATURES

Key Features	PIC16F873A	PIC16F874A	PIC16F876A	PIC16F877A
Operating Frequency	DC – 20 MHz	DC – 20 MHz	DC – 20 MHz	DC – 20 MHz
Resets (and Delays)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)
Flash Program Memory (14-bit words)	4K	4K	8K	8K
Data Memory (bytes)	192	192	368	368
EEPROM Data Memory (bytes)	128	128	256	256
Interrupts	14	15	14	15
I/O Ports	Ports A, B, C	Ports A, B, C, D, E	Ports A, B, C	Ports A, B, C, D, E
Timers	3	3	3	3
Capture/Compare/PWM modules	2	2	2	2
Serial Communications	MSSP, USART	MSSP, USART	MSSP, USART	MSSP, USART
Parallel Communications	—	PSP	—	PSP
10-bit Analog-to-Digital Module	5 input channels	8 input channels	5 input channels	8 input channels
Analog Comparators	2	2	2	2
Instruction Set	35 Instructions	35 Instructions	35 Instructions	35 Instructions
Packages	28-pin PDIP 28-pin SOIC 28-pin SSOP 28-pin QFN	40-pin PDIP 44-pin PLCC 44-pin TQFP 44-pin QFN	28-pin PDIP 28-pin SOIC 28-pin SSOP 28-pin QFN	40-pin PDIP 44-pin PLCC 44-pin TQFP 44-pin QFN

MEMORY ORGANIZATION

There are three memory blocks in each of the PIC16F87XA devices. The program memory and data memory have separate buses so that concurrent access can occur and is detailed in this section. The EEPROM data memory block is detailed in **Section 3.0 "Data EEPROM and Flash Program Memory"**.

Additional information on device memory may be found in the PICmicro® Mid-Range MCU Family Reference Manual (DS33023).

Program Memory Organization

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

The Reset vector is at 0000h and the interrupt vector is at 0004h.

FIGURE 2-1: PIC16F876A/877A PROGRAM MEMORY MAP AND STACK

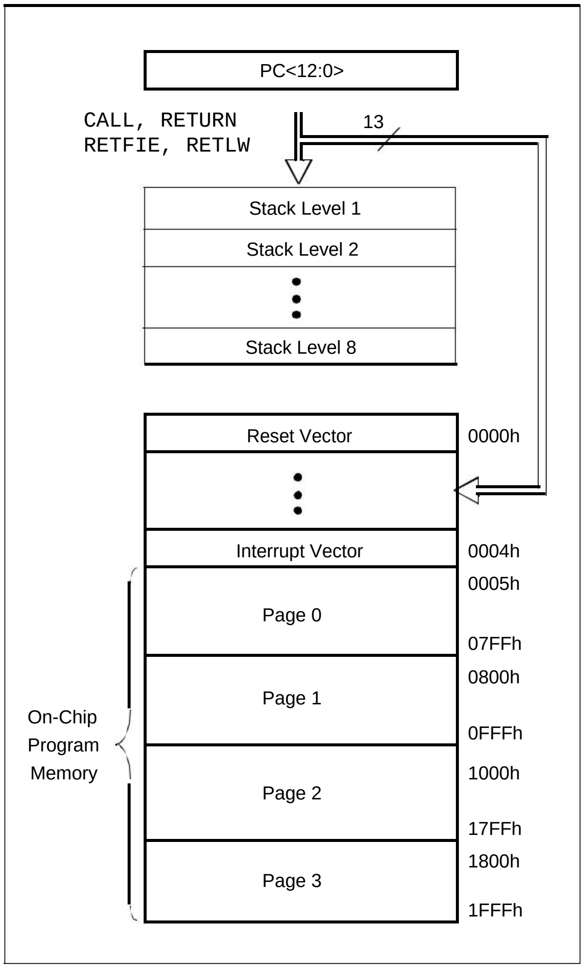
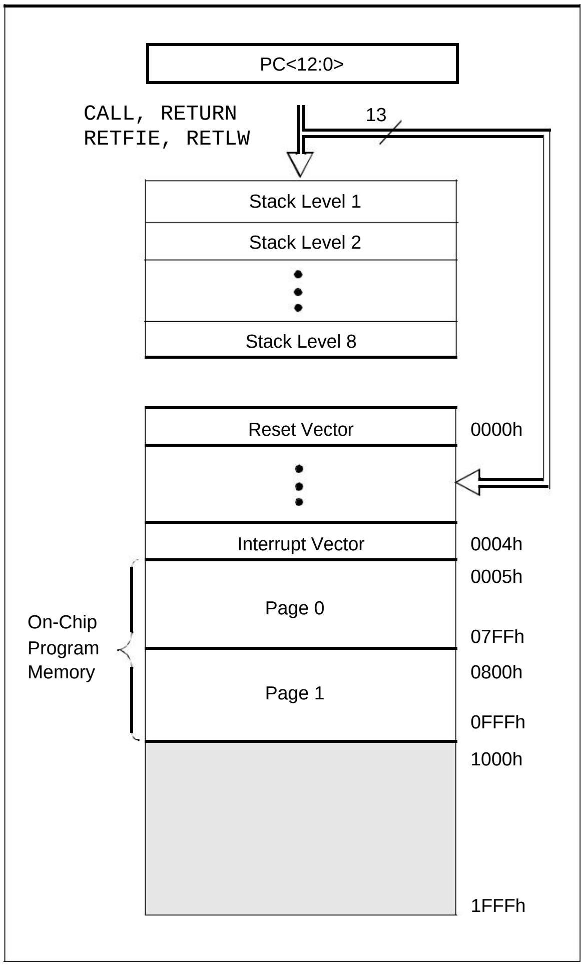


FIGURE 2-2: PIC16F873A/874A PROGRAM MEMORY MAP AND STACK



I/O PORTS

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

Additional information on I/O ports may be found in the PICmicro™ Mid-Range Reference Manual (DS33023).

PORTA and the TRISA Register

PORTA is a 6-bit wide, bidirectional port. The corresponding data direction register is TRISA. Setting a TRISA bit (= 1) will make the corresponding PORTA pin an input (i.e., put the corresponding output driver in a High-Impedance mode). Clearing a TRISA bit (= 0) will make the corresponding PORTA pin an output (i.e., put the contents of the output latch on the selected pin).

Reading the PORTA register reads the status of the pins, whereas writing to it will write to the port latch. All write operations are read-modify-write operations. Therefore, a write to a port implies that the port pins are read, the value is modified and then written to the port data latch.

Pin RA4 is multiplexed with the Timer0 module clock input to become the RA4/T0CKI pin. The RA4/T0CKI pin is a Schmitt Trigger input and an open-drain output. All other PORTA pins have TTL input levels and full CMOS output drivers.

Other PORTA pins are multiplexed with analog inputs and the analog VREF input for both the A/D converters and the comparators. The operation of each pin is selected by clearing/setting the appropriate control bits in the ADCON1 and/or CMCON registers.

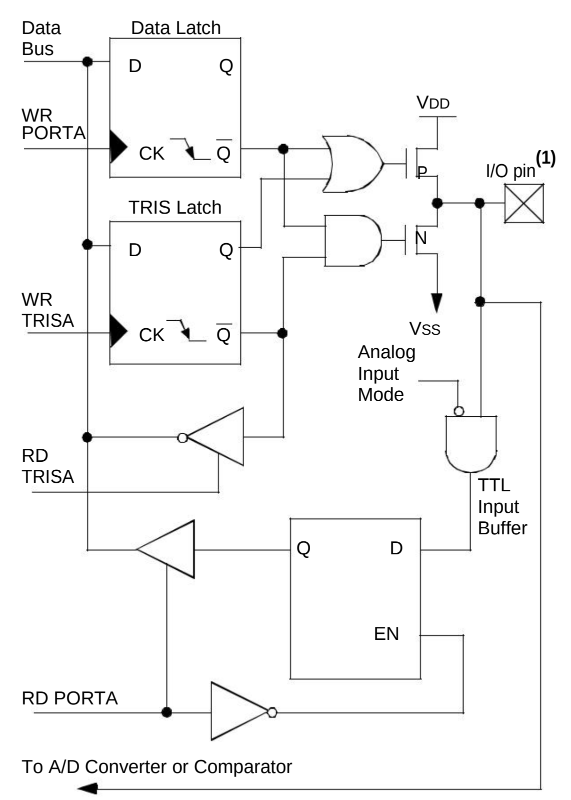
Note: On a Power-on Reset, these pins are con-figured as analog inputs and read as '0'. The comparators are in the off (digital) state.

The TRISA register controls the direction of the port pins even when they are being used as analog inputs. The user must ensure the bits in the TRISA register are maintained set when using them as analog inputs.

EXAMPLE 4-1: INITIALIZING PORTA

```
BCF     STATUS, RP0  ;
BCF     STATUS, RP1  ; Bank0
CLRF    PORTA        ; Initialize PORTA by
                     ; clearing output
                     ; data latches
BSF     STATUS, RP0  ; Select Bank 1
MOVLW   0x06         ; Configure all pins
MOVWF   ADCON1       ; as digital inputs
MOVLW   0xCF         ; Value used to
                     ; initialize data
                     ; direction
MOVWF   TRISA        ; Set RA<3:0> as inputs
                     ; RA<5:4> as outputs
                     ; TRISA<7:6>are always
                     ; read as '0'.
```

FIGURE 4-1: BLOCK DIAGRAM OF RA3:RA0 PINS



Note 1: I/O pins have protection diodes to VDD and VSS.

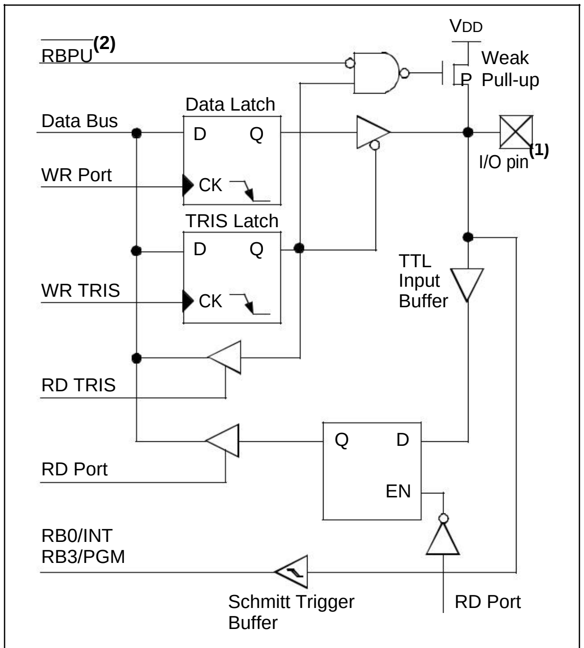
PORTB and the TRISB Register

PORTB is an 8-bit wide, bidirectional port. The corresponding data direction register is TRISB. Setting a TRISB bit (= 1) will make the corresponding PORTB pin an input (i.e., put the corresponding output driver in a High-Impedance mode). Clearing a TRISB bit (= 0) will make the corresponding PORTB pin an output (i.e., put the contents of the output latch on the selected pin).

Three pins of PORTB are multiplexed with the In-Circuit Debugger and Low-Voltage Programming function: RB3/PGM, RB6/PGC and RB7/PGD. The alternate functions of these pins are described in **Section 14.0 "Special Features of the CPU"**.

Each of the PORTB pins has a weak internal pull-up. A single control bit can turn on all the pull-ups. This is per-formed by clearing bit RBPU (OPTION_REG<7>). The weak pull-up is automatically turned off when the port pin is configured as an output. The pull-ups are disabled on a Power-on Reset.

FIGURE 4-4: BLOCK DIAGRAM OF RB3:RB0 PINS



Note 1: I/O pins have diode protection to VDD and VSS.

To enable weak pull-ups, set the appropriate TRIS bit(s) and clear the RBPU bit (OPTION_REG<7>).

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

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

Any read or write of PORTB. This will end the mismatch condition.

Clear flag bit RBIF.

A mismatch condition will continue to set flag bit RBIF. Reading PORTB will end the mismatch condition and allow flag bit RBIF to be cleared.

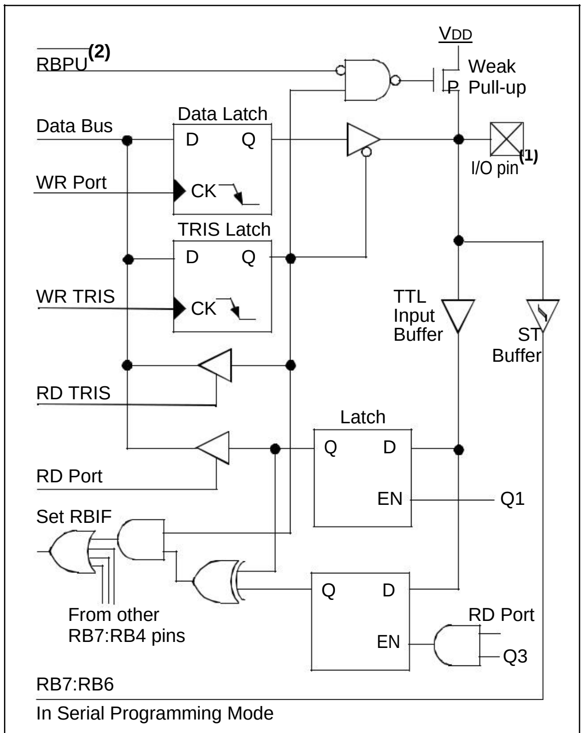
The interrupt-on-change feature is recommended for wake-up on key depression operation and operations where PORTB is only used for the interrupt-on-change feature. Polling of PORTB is not recommended while using the interrupt-on-change feature.

This interrupt-on-mismatch feature, together with software configurable pull-ups on these four pins, allow easy interface to a keypad and make it possible for wake-up on key depression. Refer to the application note, *AN552, "Implementing Wake-up on Key Stroke"* (DS00552).

RB0/INT is an external interrupt input pin and is configured using the INTEDG bit (OPTION_REG<6>).

RB0/INT is discussed in detail in **Section 14.11.1 "INT Interrupt"**.

FIGURE 4-5: BLOCK DIAGRAM OF RB7:RB4 PINS



Note 1: I/O pins have diode protection to VDD and VSS.

To enable weak pull-ups, set the appropriate TRIS bit(s) and clear the RBPU bit (OPTION_REG<7>).

PORTC and the TRISC Register

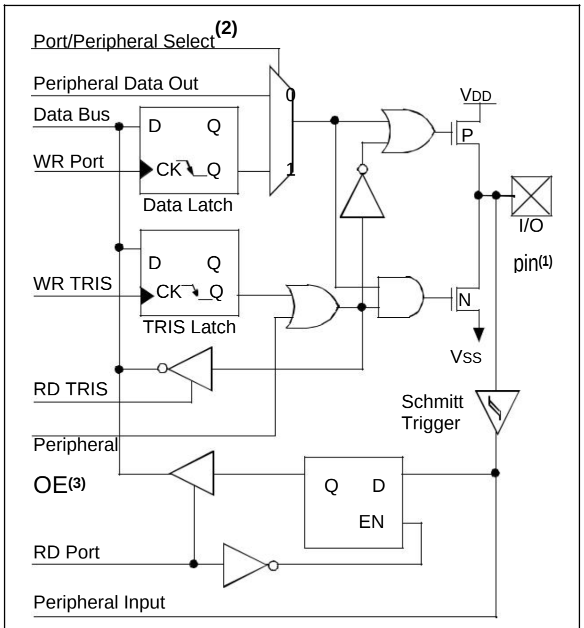
PORTC is an 8-bit wide, bidirectional port. The corresponding data direction register is TRISC. Setting a TRISC bit (= `1`) will make the corresponding PORTC pin an input (i.e., put the corresponding output driver in a High-Impedance mode). Clearing a TRISC bit (= `0`) will make the corresponding PORTC pin an output (i.e., put the contents of the output latch on the selected pin).

PORTC is multiplexed with several peripheral functions (Table 4-5). PORTC pins have Schmitt Trigger input buffers.

When the I^2C module is enabled, the PORTC<4:3> pins can be configured with normal I^2C levels, or with SMBus levels, by using the CKE bit (SSPSTAT<6>).

When enabling peripheral functions, care should be taken in defining TRIS bits for each PORTC pin. Some peripherals override the TRIS bit to make a pin an output, while other peripherals override the TRIS bit to make a pin an input. Since the TRIS bit override is in effect while the peripheral is enabled, read-modify-write instructions (`BSF`, `BCF`, `XORWF`) with TRISC as the destination, should be avoided. The user should refer to the corresponding peripheral section for the correct TRIS bit settings.

FIGURE 4-6: PORTC BLOCK DIAGRAM (PERIPHERAL OUTPUT OVERRIDE) RC<2:0>, RC<7:5>

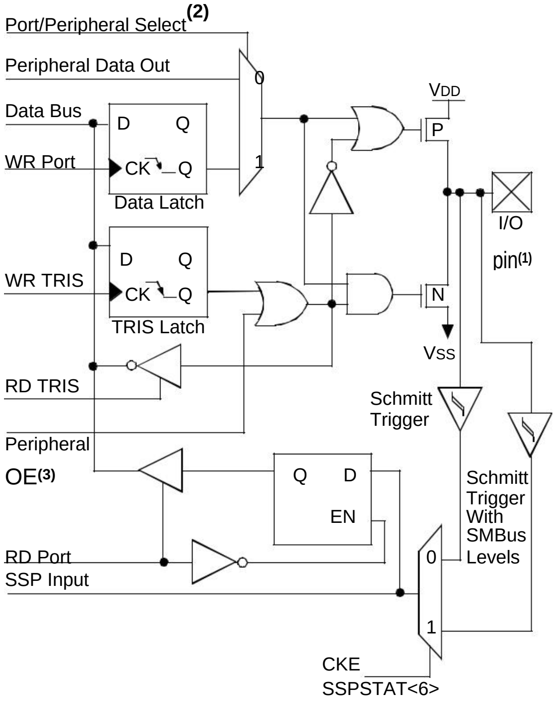


Note 1: I/O pins have diode protection to VDD and VSS.

Port/Peripheral Select signal selects between port data and peripheral output.

Peripheral OE (Output Enable) is only activated if Peripheral Select is active.

FIGURE 4-7: PORTC BLOCK DIAGRAM (PERIPHERAL OUTPUT OVERRIDE) RC<4:3>



Note 1: I/O pins have diode protection to VDD and VSS.
Port/Peripheral Select signal selects between port data and peripheral output.
Peripheral OE (Output Enable) is only activated if Peripheral Select is active.

TIMER2 MODULE

Timer2 is an 8-bit timer with a prescaler and a postscaler. It can be used as the PWM time base for the PWM mode of the CCP module(s). The TMR2 register is readable and writable and is cleared on any device Reset.

The input clock (FOSC/4) has a prescale option of 1:1, 1:4 or 1:16, selected by control bits T2CKPS1:T2CKPS0 (T2CON<1:0>).

The Timer2 module has an 8-bit period register, PR2. Timer2 increments from 00h until it matches PR2 and then resets to 00h on the next increment cycle. PR2 is a readable and writable register. The PR2 register is initialized to FFh upon Reset.

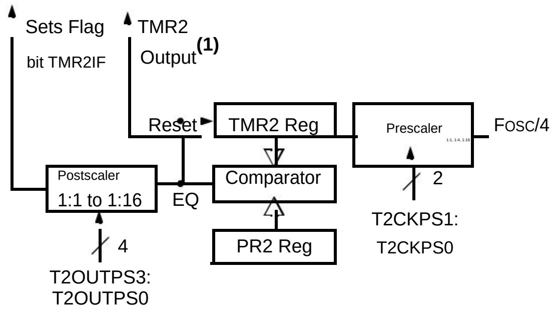
The match output of TMR2 goes through a 4-bit postscaler (which gives a 1:1 to 1:16 scaling inclusive) to generate a TMR2 interrupt (latched in flag bit, TMR2IF (PIR1<1>)).

Timer2 can be shut-off by clearing control bit, TMR2ON (T2CON<2>), to minimize power consumption.

Register 7-1 shows the Timer2 Control register.

Additional information on timer modules is available in the PICmicro® Mid-Range MCU Family Reference Manual (DS33023).

FIGURE 7-1: TIMER2 BLOCK DIAGRAM



Note 1: TMR2 register output can be software selected by the SSP module as a baud clock.

REGISTER 7-1: T2CON: TIMER2 CONTROL REGISTER (ADDRESS 12h)

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7							bit 0

bit 7 **Unimplemented:** Read as '0'

bit 6-3 **TOUTPS3:TOUTPS0**: Timer2 Output Postscale Select bits

0000 = 1:1 postscale
0001 = 1:2 postscale
0010 = 1:3 postscale
•
•
•
1111 = 1:16 postscale

bit 2 **TMR2ON**: Timer2 On bit

1 = Timer2 is on
0 = Timer2 is off

bit 1-0 **T2CKPS1:T2CKPS0**: Timer2 Clock Prescale Select bits

00 = Prescaler is 1
01 = Prescaler is 4
1x = Prescaler is 16

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Timer2 Prescaler and Postscaler

The prescaler and postscaler counters are cleared when any of the following occurs:

a write to the TMR2 register

a write to the T2CON register

any device Reset (POR, $\overline{\text{MCLR}}$ Reset, WDT Reset or BOR)

TMR2 is not cleared when T2CON is written.

Output of TMR2

The output of TMR2 (before the postscaler) is fed to the SSP module, which optionally uses it to generate the shift clock.

TABLE 7-1: REGISTERS ASSOCIATED WITH TIMER2 AS A TIMER/COUNTER

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
0Bh, 8Bh, 10Bh, 18Bh	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	`0000 000x`	`0000 000u`
0Ch	PIR1	PSPIF[(1)]	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	`0000 0000`	`0000 0000`
8Ch	PIE1	PSPIE[(1)]	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	`0000 0000`	`0000 0000`
11h	TMR2	Timer2 Module's Register								`0000 0000`	`0000 0000`
12h	T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	`-000 0000`	`-000 0000`
92h	PR2	Timer2 Period Register								`1111 1111`	`1111 1111`

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by the Timer2 module.

Note 1: Bits PSPIE and PSPIF are reserved on 28-pin devices; always maintain these bits clear.

ADDRESSABLE UNIVERSAL SYNCHRONOUS ASYNCHRONOUS RECEIVER TRANSMITTER (USART)

The Universal Synchronous Asynchronous Receiver Transmitter (USART) module is one of the two serial I/O modules. (USART is also known as a Serial Communications Interface or SCI.) The USART can be configured as a full-duplex asynchronous system that can communicate with peripheral devices, such as CRT terminals and personal computers, or it can be configured as a half-duplex synchronous system that can communicate with peripheral devices, such as A/D or D/A integrated circuits, serial EEPROMs, etc.

The USART can be configured in the following modes:

Asynchronous (full-duplex)
Synchronous – Master (half-duplex)
Synchronous – Slave (half-duplex)

Bit SPEN (RCSTA<7>) and bits TRISC<7:6> have to be set in order to configure pins RC6/TX/CK and RC7/RX/DT as the Universal Synchronous Asynchronous Receiver Transmitter.

The USART module also has a multi-processor communication capability using 9-bit address detection.

REGISTER 10-1: TXSTA: TRANSMIT STATUS AND CONTROL REGISTER (ADDRESS 98h)

R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D
bit 7							bit 0

bit 7 **CSRC:** Clock Source Select bit

Asynchronous mode:
Don't care.

Synchronous mode:
1 = Master mode (clock generated internally from BRG)
0 = Slave mode (clock from external source)

bit 6 **TX9**: 9-bit Transmit Enable bit
1 = Selects 9-bit transmission
0 = Selects 8-bit transmission

bit 5 **TXEN**: Transmit Enable bit
1 = Transmit enabled
0 = Transmit disabled

Note: SREN/CREN overrides TXEN in Sync mode.

bit 4 **SYNC**: USART Mode Select bit
1 = Synchronous mode
0 = Asynchronous mode

bit 3 **Unimplemented:** Read as '0'

bit 2 **BRGH**: High Baud Rate Select bit

Asynchronous mode:
1 = High speed
0 = Low speed

Synchronous mode:
Unused in this mode.

bit 1 **TRMT**: Transmit Shift Register Status bit
1 = TSR empty
0 = TSR full

bit 0 **TX9D:** 9th bit of Transmit Data, can be Parity bit

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

REGISTER 10-2: RCSTA: RECEIVE STATUS AND CONTROL REGISTER (ADDRESS 18h)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-x
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
bit 7							bit 0

bit 7 **SPEN:** Serial Port Enable bit

1 = Serial port enabled (configures RC7/RX/DT and RC6/TX/CK pins as serial port
pins) 0 = Serial port disabled

bit 6 **RX9**: 9-bit Receive Enable bit

1 = Selects 9-bit reception
0 = Selects 8-bit reception

bit 5 **SREN**: Single Receive Enable bit

Asynchronous mode:
Don’t care.

Synchronous mode – Master:
1 = Enables single receive
0 = Disables single receive
This bit is cleared after reception is complete.

Synchronous mode – Slave:
Don’t care.

bit 4 **CREN**: Continuous Receive Enable bit

Asynchronous mode:
1 = Enables continuous receive
0 = Disables continuous receive

Synchronous mode:
1 = Enables continuous receive until enable bit CREN is cleared (CREN overrides
SREN) 0 = Disables continuous receive

bit 3 **ADDEN:** Address Detect Enable bit

Asynchronous mode 9-bit (RX9 = 1):
1 = Enables address detection, enables interrupt and load of the receive buffer when RSR<8> is set
0 = Disables address detection, all bytes are received and ninth bit can be used as parity bit

bit 2 **FERR**: Framing Error bit

1 = Framing error (can be updated by reading RCREG register and receive next valid byte)
0 = No framing error

bit 1 **OERR**: Overrun Error bit

1 = Overrun error (can be cleared by clearing bit CREN)
0 = No overrun error

bit 0 **RX9D:** 9th bit of Received Data (can be parity bit but must be calculated by user firmware)

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
- n = Value at POR	‘1’ = Bit is set	‘0’ = Bit is cleared	x = Bit is unknown

USART Asynchronous Mode

In this mode, the USART uses standard Non-Return-to-Zero (NRZ) format (one Start bit, eight or nine data bits and one Stop bit). The most common data format is 8 bits. An on-chip, dedicated, 8-bit Baud Rate Generator can be used to derive standard baud rate frequencies from the oscillator. The USART transmits and receives the LSb first. The transmitter and receiver are functionally independent but use the same data format and baud rate. The baud rate generator produces a clock, either x16 or x64 of the bit shift rate, depending on bit BRGH (TXSTA<2>). Parity is not supported by the hardware but can be implemented in software (and stored as the ninth data bit). Asynchronous mode is stopped during Sleep.

Asynchronous mode is selected by clearing bit SYNC (TXSTA<4>).

The USART Asynchronous module consists of the following important elements:

Baud Rate Generator

Sampling Circuit

Asynchronous Transmitter

Asynchronous Receiver

USART ASYNCHRONOUS TRANSMITTER

The USART transmitter block diagram is shown in Figure 10-1. The heart of the transmitter is the Transmit (Serial) Shift Register (TSR). The shift register obtains its data from the Read/Write Transmit Buffer, TXREG. The TXREG register is loaded with data in software. The TSR register is not loaded until the Stop bit has been transmitted from the previous load. As soon as the Stop bit is transmitted, the TSR is loaded with new data from the TXREG register (if available). Once the TXREG register transfers the data to the TSR register (occurs in one TCY), the TXREG register is empty and flag bit, TXIF (PIR1<4>), is set. This interrupt can be enabled/disabled by setting/clearing enable bit, TXIE (PIE1<4>). Flag bit TXIF will be set regardless of the state of enable bit TXIE and cannot be cleared in software. It will reset only when new data is loaded into the TXREG register. While flag bit TXIF indicates the status of the TXREG register, another bit, TRMT (TXSTA<1>), shows the status of the TSR register. Status bit TRMT is a read-only bit which is set when the TSR register is empty. No interrupt logic is tied to this bit so the user has to poll this bit in order to determine if the TSR register is empty.

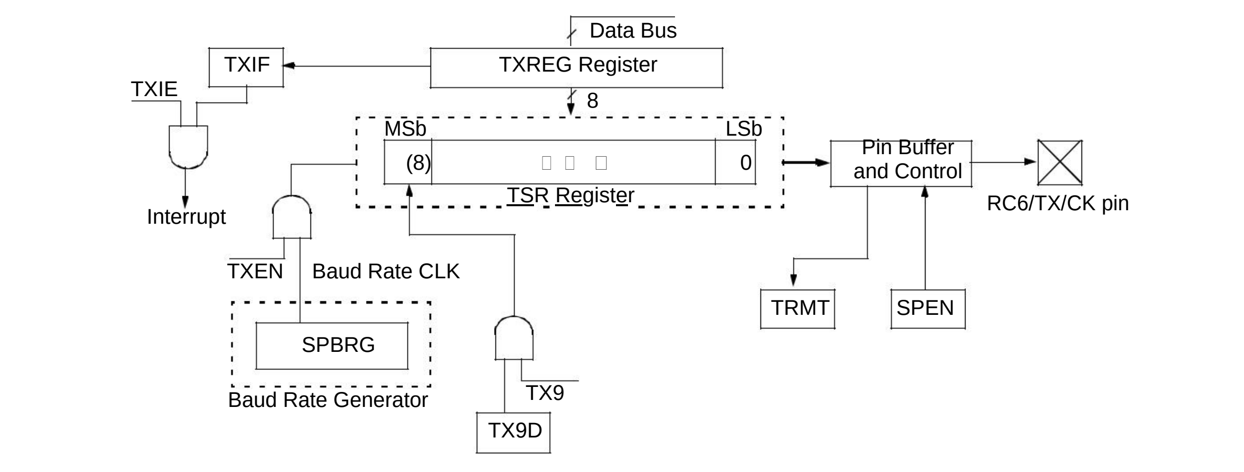
Note 1: The TSR register is not mapped in data memory so it is not available to the user.

Flag bit TXIF is set when enable bit TXEN is set. TXIF is cleared by loading TXREG.

Transmission is enabled by setting enable bit, TXEN (TXSTA<5>). The actual transmission will not occur until the TXREG register has been loaded with data and the Baud Rate Generator (BRG) has produced a shift clock (Figure 10-2). The transmission can also be started by first loading the TXREG register and then setting enable bit TXEN. Normally, when transmission is first started, the TSR register is empty. At that point, transfer to the TXREG register will result in an immedi-ate transfer to TSR, resulting in an empty TXREG. A back-to-back transfer is thus possible (Figure 10-3). Clearing enable bit TXEN during a transmission will cause the transmission to be aborted and will reset the transmitter. As a result, the RC6/TX/CK pin will revert to high-impedance.

In order to select 9-bit transmission, transmit bit TX9 (TXSTA<6>) should be set and the ninth bit should be written to TX9D (TXSTA<0>). The ninth bit must be written before writing the 8-bit data to the TXREG register. This is because a data write to the TXREG regis-ter can result in an immediate transfer of the data to the TSR register (if the TSR is empty). In such a case, an incorrect ninth data bit may be loaded in the TSR register.

FIGURE 10-1: USART TRANSMIT BLOCK DIAGRAM



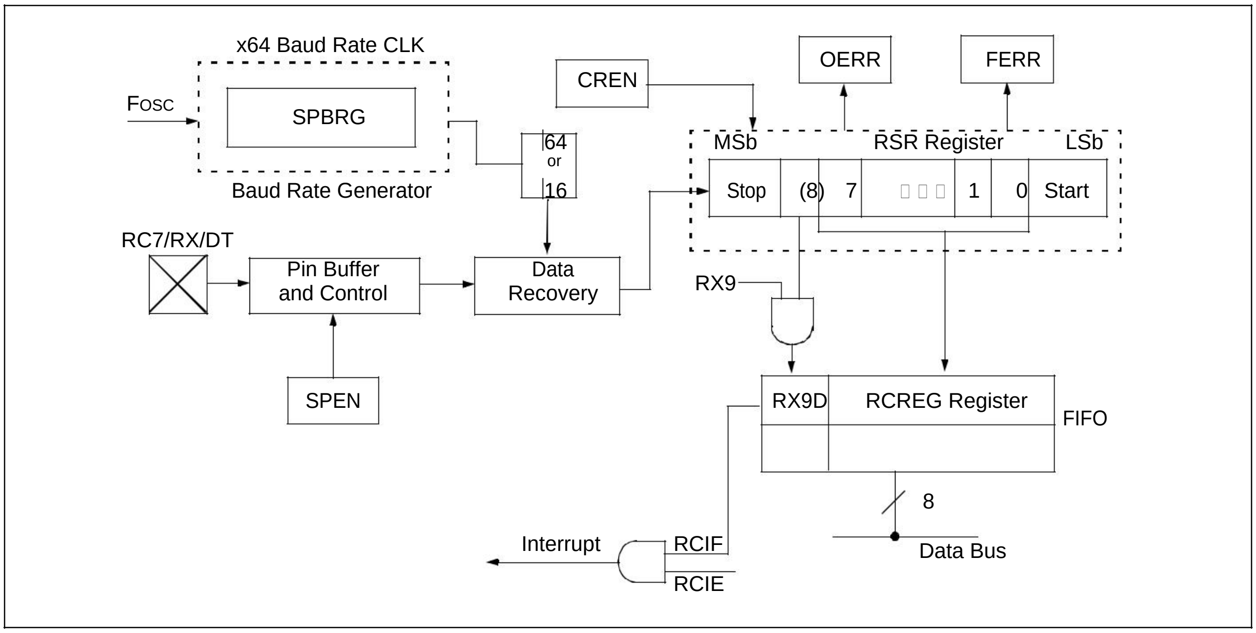
USART ASYNCHRONOUS RECEIVER

The receiver block diagram is shown in Figure 10-4. The data is received on the RC7/RX/DT pin and drives the data recovery block. The data recovery block is actually a high-speed shifter, operating at x16 times the baud rate; whereas the main receive serial shifter operates at the bit rate or at FOSC.

Once Asynchronous mode is selected, reception is enabled by setting bit CREN (RCSTA<4>).

The heart of the receiver is the Receive (Serial) Shift Register (RSR). After sampling the Stop bit, the received data in the RSR is transferred to the RCREG register (if it is empty). If the transfer is complete, flag bit, RCIF (PIR1<5>), is set. The actual interrupt can be enabled/disabled by setting/clearing enable bit, RCIE (PIE1<5>). Flag bit RCIF is a read-only bit which is cleared by the hardware. It is cleared when the RCREG register has been read and is empty. The RCREG is a double-buffered register (i.e., it is a two-deep FIFO). It is possible for two bytes of data to be received and transferred to the RCREG FIFO and a third byte to begin shifting to the RSR register. On the detection of the Stop bit of the third byte, if the RCREG register is still full, the Overrun Error bit, OERR (RCSTA<1>), will be set. The word in the RSR will be lost. The RCREG register can be read twice to retrieve the two bytes in the FIFO. Overrun bit OERR has to be cleared in soft-ware. This is done by resetting the receive logic (CREN is cleared and then set). If bit OERR is set, transfers from the RSR register to the RCREG register are inhib-ited and no further data will be received. It is, therefore, essential to clear error bit OERR if it is set. Framing error bit, FERR (RCSTA<2>), is set if a Stop bit is detected as clear. Bit FERR and the 9th receive bit are buffered the same way as the receive data. Reading the RCREG will load bits RX9D and FERR with new values, therefore, it is essential for the user to read the RCSTA register before reading the RCREG register in order not to lose the old FERR and RX9D information.

FIGURE 10-4: USART RECEIVE BLOCK DIAGRAM



ANALOG-TO-DIGITAL CONVERTER (A/D) MODULE

The Analog-to-Digital (A/D) Converter module has five inputs for the 28-pin devices and eight for the 40/44-pin devices.

The conversion of an analog input signal results in a corresponding 10-bit digital number. The A/D module has high and low-voltage reference input that is software selectable to some combination of VDD, VSS, RA2 or RA3.

The A/D converter has a unique feature of being able to operate while the device is in Sleep mode. To operate in Sleep, the A/D clock must be derived from the A/D's internal RC oscillator.

The A/D module has four registers. These registers are:

A/D Result High Register (ADRESH)
A/D Result Low Register (ADRESL)
A/D Control Register 0 (ADCON0)
A/D Control Register 1 (ADCON1)

The ADCON0 register, shown in Register 11-1, controls the operation of the A/D module. The ADCON1 register, shown in Register 11-2, configures the functions of the port pins. The port pins can be configured as analog inputs (RA3 can also be the voltage reference) or as digital I/O.

Additional information on using the A/D module can be found in the PICmicro® Mid-Range MCU Family Reference Manual (DS33023).

REGISTER 11-1: ADCON0 REGISTER (ADDRESS 1Fh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	—	ADON
bit 7							bit 0

bit 7-6 **ADCS1:ADCS0:** A/D Conversion Clock Select bits (ADCON0 bits in **bold**)

ADCON1 <ADCS2>	ADCON0 <ADCS1:ADCS0>	Clock Conversion
0	**00**	FOSC/2
0	**01**	FOSC/8
0	**10**	FOSC/32
0	**11**	FRC (clock derived from the internal A/D RC oscillator)
1	**00**	FOSC/4
1	**01**	FOSC/16
1	**10**	FOSC/64
1	**11**	FRC (clock derived from the internal A/D RC oscillator)

bit 5-3 **CHS2:CHS0:** Analog Channel Select bits

000 = Channel 0 (AN0)
001 = Channel 1 (AN1)
010 = Channel 2 (AN2)
011 = Channel 3 (AN3)
100 = Channel 4 (AN4)
101 = Channel 5 (AN5)
110 = Channel 6 (AN6)
111 = Channel 7 (AN7)

Note: The PIC16F873A/876A devices only implement A/D channels 0 through 4; the unimplemented selections are reserved. Do not select any unimplemented channels with these devices.

bit 2 **GO/$\overline{\text{DONE}}$:** A/D Conversion Status bit

When ADON = 1:
1 = A/D conversion in progress (setting this bit starts the A/D conversion which is automatically cleared by hardware when the A/D conversion is complete)
0 = A/D conversion not in progress

bit 1 **Unimplemented:** Read as '0'

bit 0 **ADON:** A/D On bit

1 = A/D converter module is powered up
0 = A/D converter module is shut-off and consumes no operating current

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

REGISTER 11-2: ADCON1 REGISTER (ADDRESS 9Fh)

R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

bit 7 **ADFM:** A/D Result Format Select bit

1 = Right justified. Six (6) Most Significant bits of ADRESH are read as ‘0’. 0
= Left justified. Six (6) Least Significant bits of ADRESL are read as ‘0’.

bit 6 **ADCS2:** A/D Conversion Clock Select bit (ADCON1 bits in shaded area and in **bold**)

ADCON1 <ADCS2>	ADCON0 <ADCS1:ADCS0>	Clock Conversion
0	00	FOSC/2
0	01	FOSC/8
0	10	FOSC/32
0	11	FRC (clock derived from the internal A/D RC oscillator)
1	00	FOSC/4
1	01	FOSC/16
1	10	FOSC/64
1	11	FRC (clock derived from the internal A/D RC oscillator)

bit 5-4 **Unimplemented:** Read as ‘0’

bit 3-0 **PCFG3:PCFG0:** A/D Port Configuration Control bits

PCFG <3:0>	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0	VREF+	VREF-	C/R
0000	A	A	A	A	A	A	A	A	VDD	VSS	8/0
0001	A	A	A	A	VREF+	A	A	A	AN3	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	AN3	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	AN3	VSS	2/1
011x	D	D	D	D	D	D	D	D	—	—	0/0
1000	A	A	A	A	VREF+	VREF-	A	A	AN3	AN2	6/2
1001	D	D	A	A	A	A	A	A	VDD	VSS	6/0
1010	D	D	A	A	VREF+	A	A	A	AN3	VSS	5/1
1011	D	D	A	A	VREF+	VREF-	A	A	AN3	AN2	4/2
1100	D	D	D	A	VREF+	VREF-	A	A	AN3	AN2	3/2
1101	D	D	D	D	VREF+	VREF-	A	A	AN3	AN2	2/2
1110	D	D	D	D	D	D	D	A	VDD	VSS	1/0
1111	D	D	D	D	VREF+	VREF-	D	A	AN3	AN2	1/2

A = Analog input D = Digital I/O
C/R = # of analog input channels/# of A/D voltage references

Legend:

R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
- n = Value at POR	‘1’ = Bit is set	‘0’ = Bit is cleared	x = Bit is unknown

Note: On any device Reset, the port pins that are multiplexed with analog functions (ANx) are forced to be an analog input.