HEXAPOD ROBOT

Sarath.M

May 3, 2016

ACKNOWLEDGMENTS

This major project has been possible only due to support and help from various people. This report would not complete unless their contributions are acknowledged.

First of all, we would like to thank **Dr.Suresh Kumar, Principal, College Of Engineering,Cherthala** and to the management of College of Engineering,Cherthala for giving us an opportunity and encouragement to do this work. We would like to express our sincere thanks to **Mr.Pradeep M, Head of the Department of Electronics and Communication,College of Engineering, Cherthala** for giving timely support he gave whenever required. We express our profound gratitude to **Mrs.Ajay Nath S.A, Assistant professor** for his help, inspiration, guidance, suggestions, cooperation and innovative ideas.

We wish to express our sincere gratitude to our major project coordinators **Mr.Sreekumar.K, Assistant professor** for his guidance and timely help rendered. Also, we would like to express our thanks to **Mrs Subeena Subair** and **Mr Vishnu Pradeep K**; Assistant Professors, EC Department, for their valuable help and guidance. Last but certainly not the least we would like to express our gratitude to our family and friends, for their unending support, love and encouragement.

ABSTRACT

Agriculture is big business. But the future of big farming isnt in massive machinery, but swarms of smart, cheap 'bots seeding, tending and harvesting fields one plant at a time. Whether conducted by an industrial farming outfit or a small, independent farmer, agriculture is all about yield. Per-acre production makes or break the year, and taken at the macro level it impacts global markets and can lead to humanitarian crises. And while agriculture already happens at the field-by-field level,we can improve the profit margin & the net productivity if we make agriculture even more precise. Think: plant-by-plant farming, optimized on a seed-by-seed basis. But to manage such a precise, immense workload we need autonomous robots.

Keeping this in mind we have tried to develop hexapod walking robots; a six-legged walking robot that is capable of basic mobility tasks such as walking forward, backward, rotating in place which can be used in the field of agriculture. The legs will be of a modular design. This robot will serve as a platform onto which additional sensory components could be added, or which could be programmed to perform increasingly complex motions.

The design presented in this report makes use of two simple revolute joints per leg and the required motion is provided by DC Servo motors. These 2 degrees of freedom will propel the robot in the direction of motion.

List of Figures

Contents

CHAPTER 1
PROBLEM STATEMENT

1.1 Introduction

1.1.1 General Aspects

For many years robotic systems have been widely used for industrial production and in warehoues, where a controlled environment has been a vision initiated in the early 1960's with basic research on projects on projects on automated steered systems and autonomous tracters. Recently , the developement of robotic systems in agriculture has experienced an increased interest , which has led many experts to explore the possibilities to develop more rational and adaptabe vehicles based on a behavioral approach.
In the open and outdoor environment, which will be the focus here , robotic and autonomous systems are more complex to develop - mainly because of safety issues . The robots safety system would have to be reliable enough for it to operate autonompously and unattended . It is relatively costly to develop safety systems if the vehicle has to be completely autonomous. In principle, they can work 24 hours a day but if a robot has to be attended then the time is limited by the person. In this matter, different scenarios and degrees for autonomy have been investigated depending on the task to be carried out.
The vehicles should be able to carry out useful tasks all year around, unattended and able to behave sensibly in a semi-natural environment over long periods of time. The small vehicles may also have less environmental imapct replacing the over- application of chemicals,and fertilisers , requires lower usage of energy with better control matched to requirements, as well as causing less soil compaction due to lighter weight.

1.1.2 Economics

Goense (2003) compared autonomous with conventional vehicles, equipped with implements having working widths from 50 to 120 cm. He showed that if an autonomous vehicles can be utilised 23 hours a day, it would be economic feasible with slight reductions in prices of navigation systems or with slight increases in labour costs. Goense also discussed a number of other changes that will effect the final result, such as the fraction of labour time needed out of the total machine time and the machine tracking system, which provides better utilisation of machine working width and there is no need for operators rest allowance. On

the other hand, there may be negative effects in the form of higher costs in travelling distances for service personal.

1.2 Areas Of Application

1.2.1 Crop establishment

For traditional crop establishment and seed bed preparation it has been common that the whole topsoil of a

eld is inverted with a plough to create a suitable seed bed. This is a well known operation that suits many circumstances but it also uses a lot of energy. If the same seed environment can be achieved by only mixing the soil within a few centimetres of the actual seed then the rest of the soil does not need to be disturbed as it can be well conditioned by natural soil ora and fauna.
Another traditional concept is to grow crops in rows. It would seem that the only explanation as to why this is done is that it requires the simplest type of machines. Seeds are placed relatively densely along each row. The problem is that in principle, each plant requires equal access to light, air, water and nutrients, which are often spatially related. Intra crop competition can be reduced by giving a more even or equal spacing and seed distribution with accurate placement of seeds in a more uniform pattern.
If the location of each seed is known and the position of each emerged crop plant is estimated, it will be possible to identify each plant by its spatial location. Improved information about plant characteristics allows improved management and decision making and allows a numbe of improved, more targeted operations that can improve the overall crop efficiency.
As only a small volume of soil is needed to be cultivated there are a number of different methods that could be used. Rotary mechanical tillage in two dimensions on a vertical or horizontal axis could be used instead of just one dimension or water-jetting or the use of a hygroscopic polymer gel could be used to create a micro climate for the seed.

1.3 Crop scouting

An important part of good management is the ability to collect timely and accurate information about the crop development. Quantified data has tended to be expensive and sampling costs can quickly out weigh the benefits of spatially variable management. (Godwin et al., 2003).
Data collection would be less expensive and timelier if an automated system could remain within the crop canopy for continual monitoring that can be used for assessing crop status. This could be achieved by either embedding cheap wireless sensors at strategic positions within the crop, or placing more expensive sensors onto a moving platform.
With the advent of biosensors, a whole new set of opportunities will become available to monitor growing crops for pest and disease attack (Tothill, 2001).

As the robotic/autonomous vehicle could patrol the fields continually looking for weeds and other threats, real-time alerts could be sent to the manager whenever certain conditions were encountered. These could take the form of noting actual pest or disease attack or by monitoring environmental conditions where they are likely to occur or that the risk of attack is significant. Differing growth rates could also be used to identify potential problems.

1.3.1 Selective harvesting

At present, crops are usually harvested when the average of the whole field is ready as this simplifies the harvest process. Selective harvesting involves the concept of only harvesting those parts of the crop that meet certain quantity or quality thresholds. It can be considered to be a type of pre sorting based on sensory perception. Selective harvesting has been well known in forestry for many years where certain trees are harvested according to quality and size or to improve the quality of the remaining trees.
In agriculture, examples could be to only harvest barley below a fixed protein content or combine grain that is dry enough (and leave the rest to dry out) or to select and harvest fruits and vegetables that meet a size criteria. As these criteria often attract quality premiums, increased economic returns could justify the additional sensing.
Most agricultural vehicles are getting bigger and hence not suited for this approach. Therefore, smaller and more versatile selective harvesting equipment is needed for this purpose. Either the crop can be surveyed before harvest so that the information needed about where the crop of interest is located, or that the harvester may have sensors mounted that can ascertain the crop condition. The selective harvester can then harvest that crop that is ready, while leaving the rest to mature, dry, or ripen etc.
As the products are already graded or sorted, it also adds value to the products before they leave the farm.

1.4 Target specifications of feasible solution

We need a robot configuration that is suitable for farm environment. Our next step was to find such a system.
We found out that for an outdoor environment legged robots where more promising compared to wheeled robots due to reasons as explained in next section

1.4.1 Reasons to choose legged configuration

Wheeled vehicles excel when used on prepared surfaces , such as roads and railways. They are cost-effective, efficient and capable of achieving higher velocities than any legged vehicle. However as the terrain becomes increasingly difficult, a point comes where wheeled vehicles prove inferior to legged vehicles.
Paddy fiels, sandy soils, and undulating terrain are troublesome for wheeled robots to cross; and specifically, that travelling over undulating natural terrains

involves difficult tasks for wheeled vehicles, such as step climbing , gap crossing gradients, side slopes and water crossing
Similarly traversing many man-made terrains is also difficult for wheeled robots because of narrow doorways, narrow hallways, sharp turns, floor irreglarities, ramps,raps, steps, staircase and ladders
When traversing soft soils, wheels sink down and must effectively be continuously climbing out of a hole and/or expending energy to compact the soil (See fig. 1.1). Foots most fundamental advantage over the wheel is that the compaction resistance of the soil can contribute to its forward thrust, whereas in the case of a wheel, compaction resistance reduces the forward thrust.

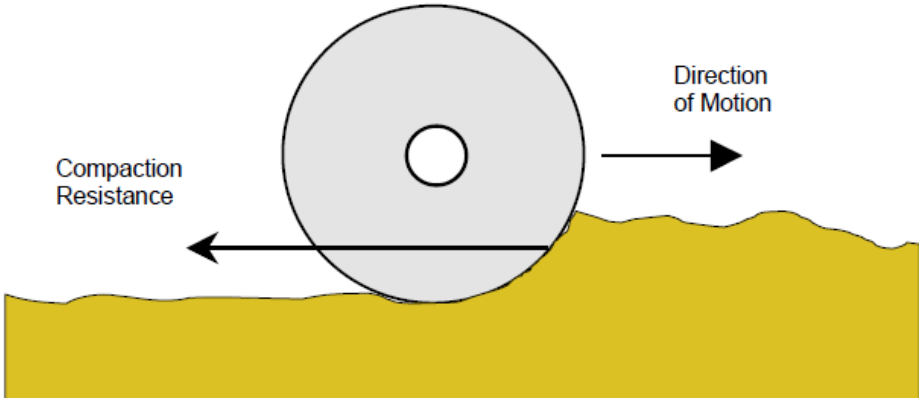


Figure 1.1: Compaction Resistance of a wheel in soft soil

Other advantages of legged configuration

1. Higher speed
2. Better fuel economy
3. Greater mobility (This includes the ability to step over obstacles and travel across terrain that is impossible for wheeled and tracked vehicles)
4. Better isolation from terrain irregularities. (This enables a smoother ride)
5. Less environmental damage. (Wheeled and tracked vehicles leave continuous ruts in soft soils, whereas legged vehicles leave only distinct footprints)

In general, legged vehicles become attractive for applications requiring traversal of terrain that is too difficult for wheeled and tracked vehicles.

1.4.2 Legged robots

Legged robots can be classified based on the number of legs used. They are :

- One legged: *Hopper robot*

Figure 1.2: Toyota Monoped

- Two legged robots: *Biped robot*

Figure 1.3: Honda ASIMO robot

- Three or more legged robot

Figure 1.4: Boston Dynamic BigDog

- Hexapod

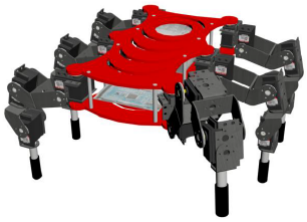


Figure 1.5: model of the Hexapod

Out of these configurations we choose the Hexapod configuration due to reasons explained in the next section

Reasons for choosing six-legged Hexapod configuration

Hexapod robots have a large number of real life applications, from crossing potentially dangerous terrain to carrying out search and rescue operations in hazardous and unpredictable disaster zones. They have a number of advantages over wheeled, quadruped or bipedal robots:

- While wheeled robots are faster on level ground than legged robots, hexapods are the fastest of the legged robots, as they have the optimum number of legs for walking speed - studies have shown that a larger number of legs does not increase walking speed (Alexadre et al, 1991).
- Having maneuverable legs allows hexapods to turn around on the spot
- In comparison to other multi-legged robots, hexapods have a higher degree of stability as there are can be up to 5 legs in contact with the ground during walking. Also, the robots center of mass stays consistently within the tripod created by the leg movements, which also gives great stability
- Hexapods also show robustness, because leg faults or loss can be managed by changing the walking mechanism
- This redundancy of legs also makes it possible to use one or more legs as hands to perform dexterous tasks

CHAPTER 2
CIRCUIT LEVEL DESCRIPTION

2.1 Components list

The various components used in the project are:

- DC Servo Motors
- PIC16F877a
- ZIGBEE module
- Wireless Camera
- DAGU Hexapod Chasis
- Sensors
    1. Temperature Sensor
    2. Humidity Sensor
    3. IR Senso

DC Servo motor

It is a Hitec HS-645MG High torque. The running current is 450mA and the idle current is 9.1mA with an operating voltage 4.8 volt or 6 volt.

Figure 2.1: Hitec HS-645MG servo motor

PIC 16f877a

PIC 16F877 is one of the most advanced microcontroller from Microchip. This controller is widely used for experimental and modern applications because of its low price, wide range of applications, high quality, and ease of availability. It is ideal for applications such as machine control applications, measurement devices, study purpose, and so on. The PIC 16F877 features all the components which modern microcontrollers normally have. Fig 2.2 show the DIP package of PIC 16f877a

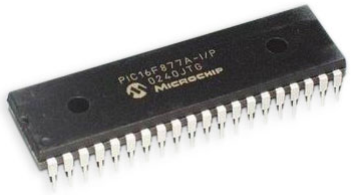


Figure 2.2: PIC 16f877a

Wireless Camera kit

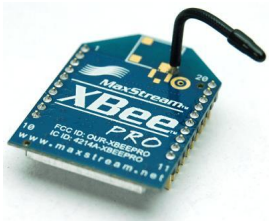
This is a 2.4GHz 4CH wireless mini camera kit including a wireless CMOS camera with MIC function and a wireless receiver. Easy installation and operation is suitable for home security. The frequency of the camera is fixed, by default is 2414 Mhz.

Figure 2.3: Wireless Camera kit

ZIGBEE modules

ZigBee is a specification for a suite of high level communication protocols used to create personal area networks built from small, low-power digital radios. ZigBee

is based on an IEEE 802.15 standard. Though low-powered, ZigBee devices can transmit data over long distances.



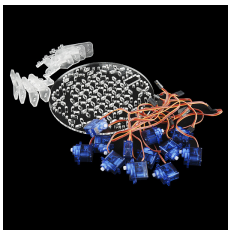
(a) ZIGBEE Pro

(b) ZIGBEE usb dongle

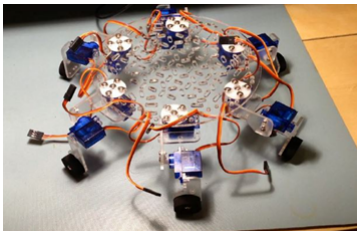
Figure 2.4: ZIGBEE transmitter and receiver

Hexapod robot chassis

This is a hexapod robot chassis that comes with 12 servos. It is made of acrylic with rubberized foam feet. This chassis was purchased online from Rhydolabz and was used for building the first prototype.



(a) Chassis parts



(b) after setting up

Figure 2.5: Hexapod chassis

Sensors:

Temperature sensor

This is the latest DS18B20 1-Wire digital temperature sensor from Maxim IC. Reports degrees C with 9 to 12-bit precision, -55C to 125C (+/-0.5C)

Figure 2.6: DS 1820 temperature sensor

Humidity sensor

This sensor module converts relative humidity(30-90% RH) to voltage and can be used in weather monitoring application.

Figure 2.7: SY-HS-220 Humidity sensor

Infrared proximity sensor

Infrared proximity sensor made by Sharp. Part # GP2Y0A21YK has an analog output that varies from 3.1V at 10cm to 0.4V at 80cm. The sensor has a Japanese Solderless Terminal (JST) Connector.

Figure 2.8: Proximity sensor - Sharp GPY2Y0A21YK

2.2 Circuit Diagram

The main circuit is split into various blocks and explained in the following sections.

2.2.1 PIC 16f877a and it's interface circuit

The above circuit shows PIC16F877A MC and its corresponding interface circuit. The X1 crystal and the capacitors C1 and C2 are used to generate the 20 MHz clock signal to the microcontroller.

The push button which is pulled down to ground by a 10k resistor R1 acts as the reset button for the MC

The IO pins RB1-RB7 and RD1-RD7 are used as control pins for 13 servo motors(12 legs+head servo).

RA3 is connected to the signal pin of IR sensor(pin 2). The signal value of pin2.

The zigbee is connected to pins 25 and 26 of PIC. Wherte 25 is the Tx pin and pin 26 is the Rx pin of the PIC IC

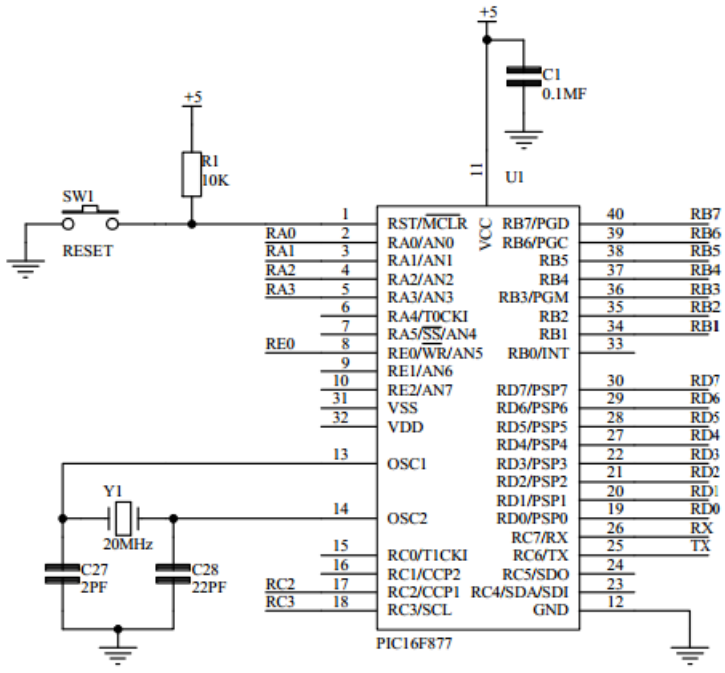


Figure 2.9: PIC 16f877a

2.2.2 ZIGBEE module and it's interface circuit

The circuit diagram shown below is the interface circuit of zigbee using darlington pair.

The Darlington transistor (often called a Darlington pair) is a compound structure consisting of two bipolar transistors connected in such a way that the current amplified by the first transistor is amplified further by the second one.This configuration gives a much higher common/emitter current gain than each transistor taken separately

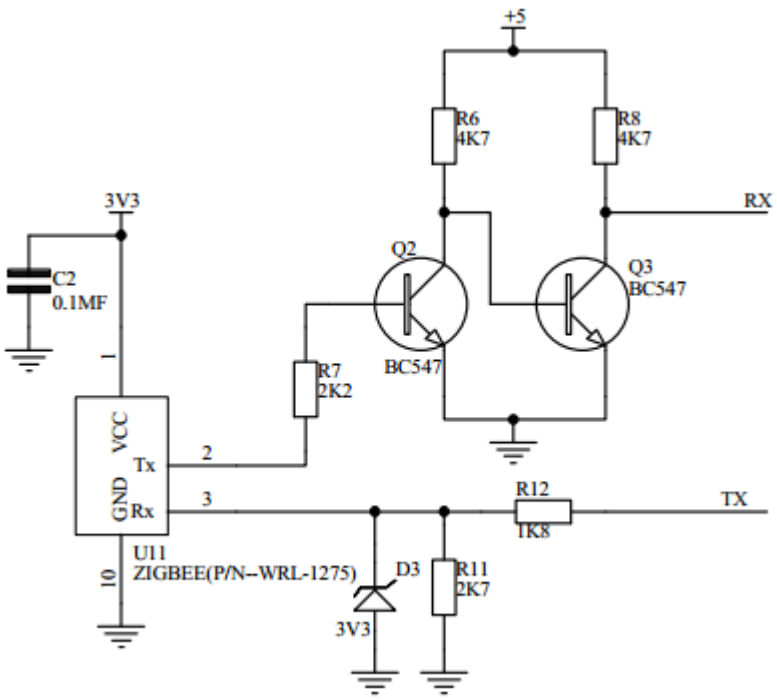


Figure 2.10: Zigbee interface circuit

2.2.3 Power supply-1

The power supply required for the PIC IC, the Camera and the IR sensor is generated using the voltage regulators shown below

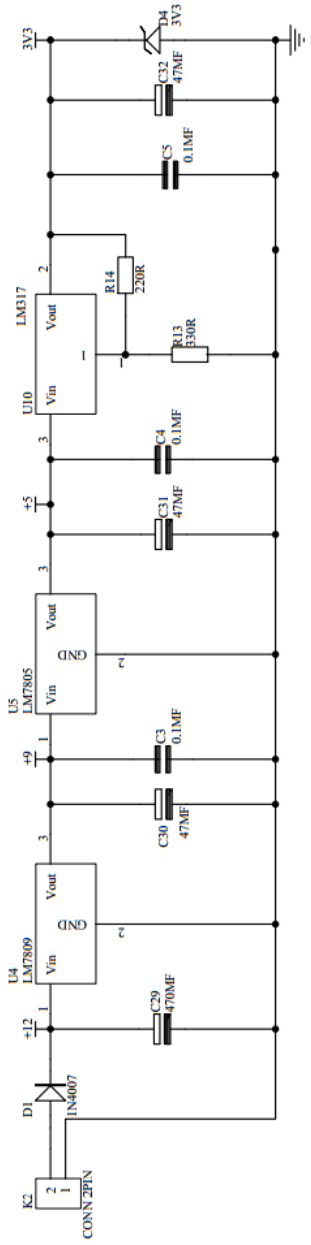
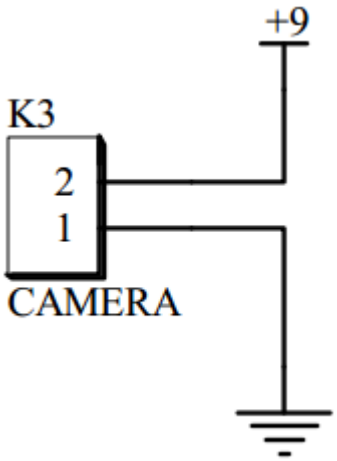


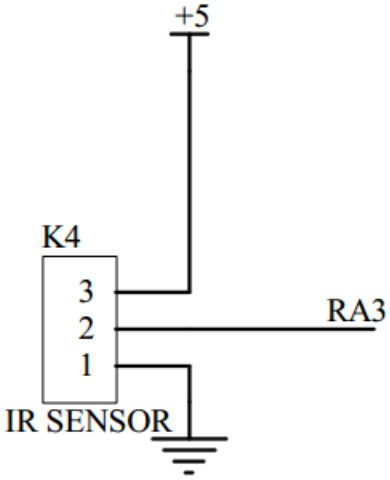
Figure 2.11: Power supply 1

2.2.4 Wireless camera and IR Proximity sensor

Wireless camera and the IR proximity sensors are supplied +9v and +5v respectively from the voltage regulators LM7809 and LM7805



(a) wireless camera



(b) IR proximity sensor

Figure 2.12: Camera and IR sensor

2.2.5 Servo motors

Fig. 2.13 shows all the servo motors used in the robot system. The servos are grouped as four and each group is powered by a power supply generated by the voltage regulator shown below. Each voltage value is labelled SERVO1, 2, 3 and 4.

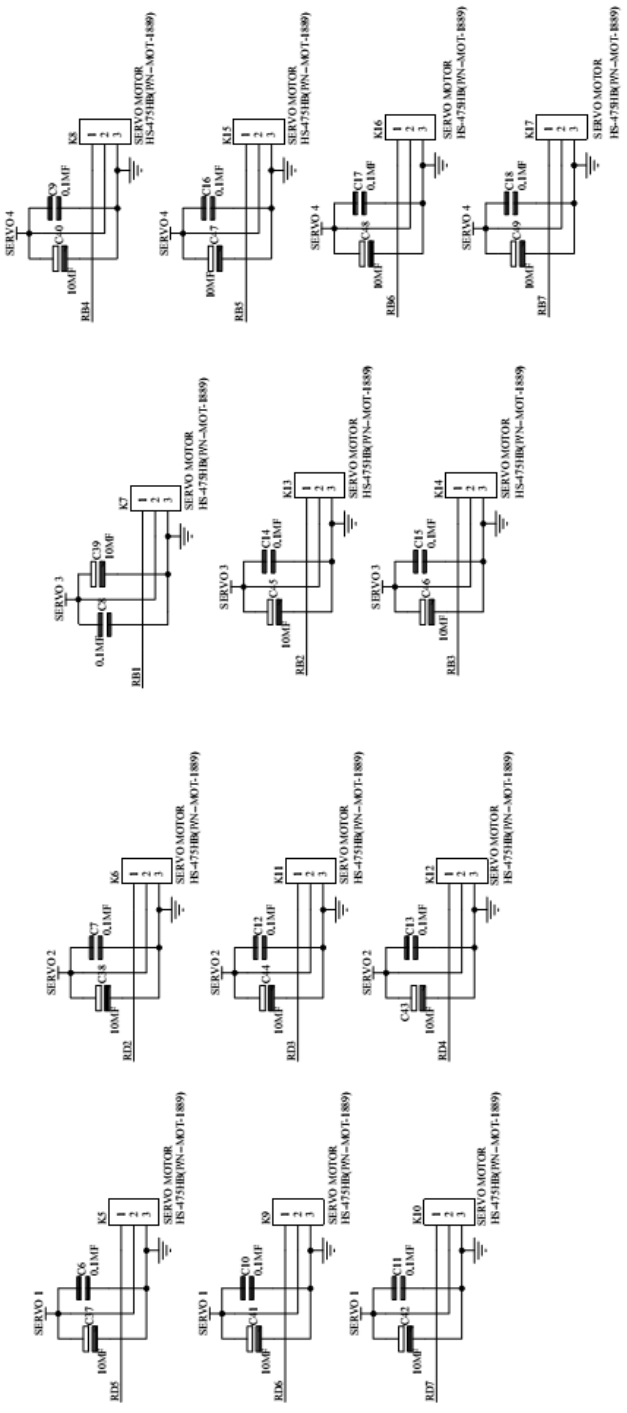


Figure 2.13: Servo motors

2.2.6 Power supply 2

The power required for each group of servos is generated by the voltage regulator circuit shown above.
The different voltage values generated are labelled SERVO 1, 2, 3 and 4. The power supply is shown in Fig. 2.14.

2.3 Tools used

The following software tools where used for this project:

1. Proteus for circuit simulation
2. Arduino UNO microcontroller for prototyping
3. MPLAB with HI-TECH C compiler for programming the PIC IC
4. MATLAB for digital image processing

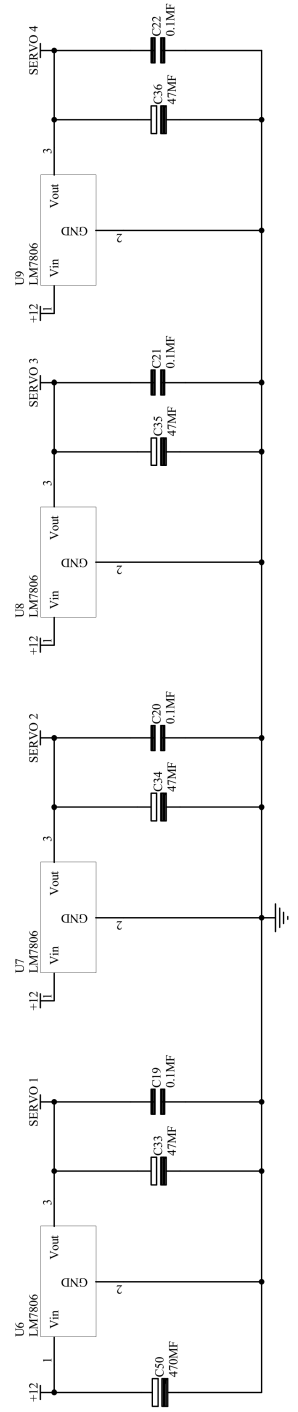


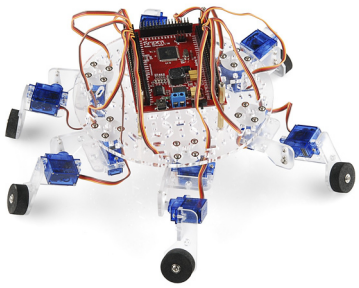
Figure 2.14: Power supply 2

CHAPTER 3
PROJECT IMPLEMENTATION

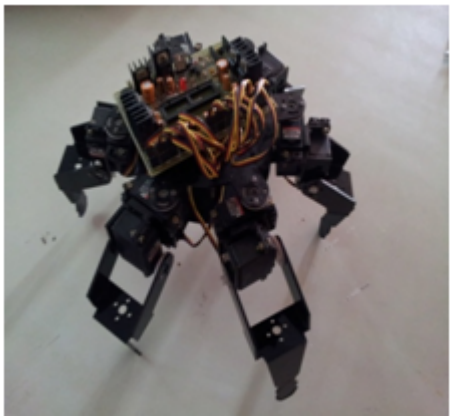
As the very first step in project implementation we learned the different methods to control servo motors. The next step was to choose a micro-controller to generate the PWM signals required to control all the 13 servo motors as well as other sensors that are used in the project.
Initially we started out with the plan to use Arduino Uno microcontroller to control the whole hexapod.
The arduino is an open-source platform and thus it's well documented. The coding was also fairly easy to learn. Soon we developed the code to control multiple servo motor's simultaneously.
But there is a problem with the Uno board. It had only 6 PWM pins and thus we could only control a maximum of 6 motors simultaneously with it.

So we switched to much more powerful PIC 16f877a micro-controller. The code for controlling all servos was written using the software MPLab and was tested using Proteus.

As a first prototype we used the Hexapod chassis kit from Dagu.The first prototype and the final design are shown below:



(a) First prototype



(b) Final Design

Figure 3.1: Prototype and final design

3.1 System Design

First the overall system is described and then each subsystem in detail. The system for controlling the robot relies on the interaction of three main elements.

- The operator, who gives directions to the robot
- The physical robot
- The software developed in this project, responsible for moving the robot in the direction provided by the operator.

The interaction between the elements are shown in the figure 3.2

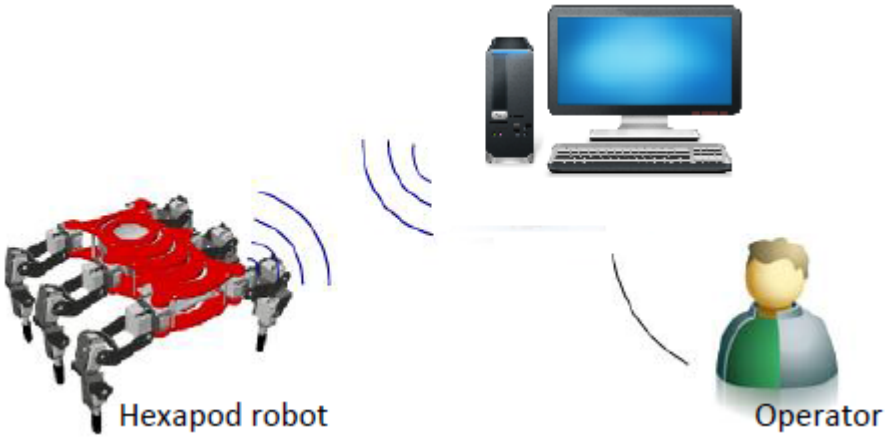


Figure 3.2: The system is based on interaction of three elements

3.1.1 System Structure

In this section we describe the different functional units and the working of the project. The block diagram in fig 3.3 shows the structure of the project
The operator gives the command to the robot via the Matlab GUI running on the PC. These commands are send to the robot using the zigbee usb dongle. The zigbee at the robot section receives these commands and sends them to the master processor. The processor contains all the necessary algorithms such as servo control, collision detection, sensor value collection and transmission. The code written in the master processor controls the walking pattern of the robot in accordance with the command received.
Simultaneously the sensors measure ambient temperature and humidity and these values are send to the PC when operator requests it via the GUI code.
The IR proximity sensor is used for collision detection algorithm. When an obstacle is detected the robot turns to avoid it.

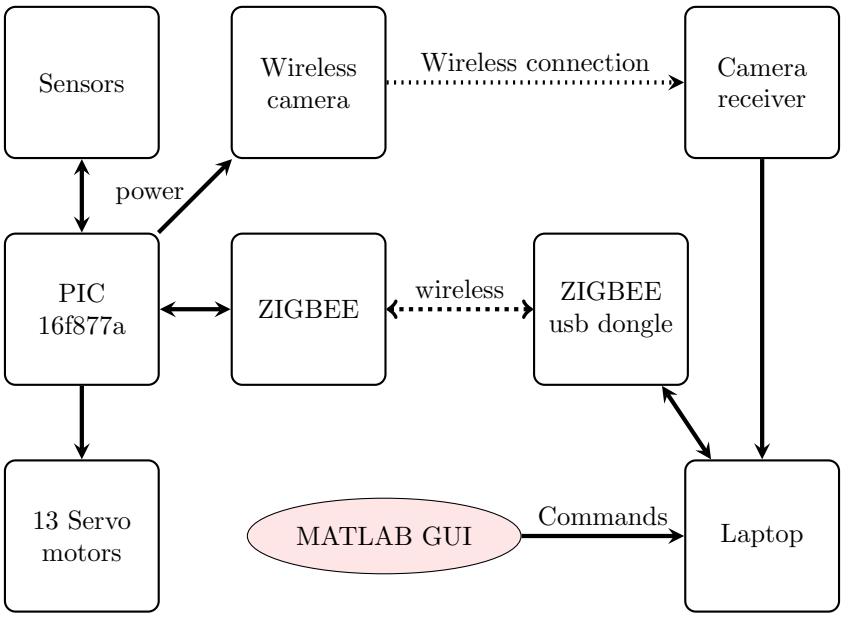


Figure 3.3: System structure

The wireless camera constantly transmits the video signals wirelessly to the camera RF receiver. The video is processed and displayed on the screen of the PC.
From this video signal individual pictures are taken and using image processing algorithms weeds are detected and the follow-up actions like application of pesticides are done. The structure is illustrated in the figure 3.3

3.2 Wlaking pattern

The Hexapod robot in this project uses the famous tripod gait for locomotion. In the following drawings a dark circle means the foot is firmly planted on the ground and is supporting the weight of the robot. A light circle means the foot is not supporting any weight and is movable.
This gait is a fast gait for the hexapod; it completes a cycle in two beats. In this gait the robot lift three legs simultaneously while leaving three legs on the ground, which keeps the robot stable.
This is called a tripod gait, because a tripod positioning of legs always supports the weight of the walker

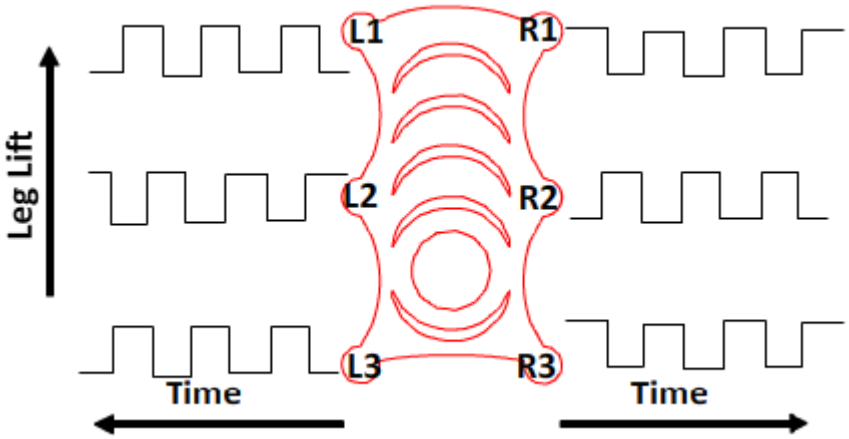


Figure 3.4: PWM pattern for each leg

3.2.1 Walking forward

We start in the rest position (see Fig. 3.5). As before, each circle represents a foot, and the dark circles show the weightbearing feet. Notice in the rest position, the center legs do not support any weight. These center legs are made to be 1/8 in shorter than the front and back legs.
In position A the center legs are rotated CW by about 25 from center position. This causes the robot to tilt to the right. The weight distribution is now on the front and back right legs and the center left leg. This is the standard tripod position as described earlier. Since there is no weight on the front and back left legs, they are free to move forward as shown in the B position of Fig. 3.5
In the C position the center legs are rotated CCW by about 25 from center position. This causes the robot to tilt to the left. The weight distribution is now on the front and back left legs and the center right leg. Since there is no weight on the front and back right legs, they are free to move forward, as shown in the D position
In position E the center legs are rotated back to their center position. The robot is not in a tilted position so its weight is distributed on the front and back legs. In the F position, the front and back legs are moved backward simultaneously, causing the robot to move forward. The walking cycle can then repeat.

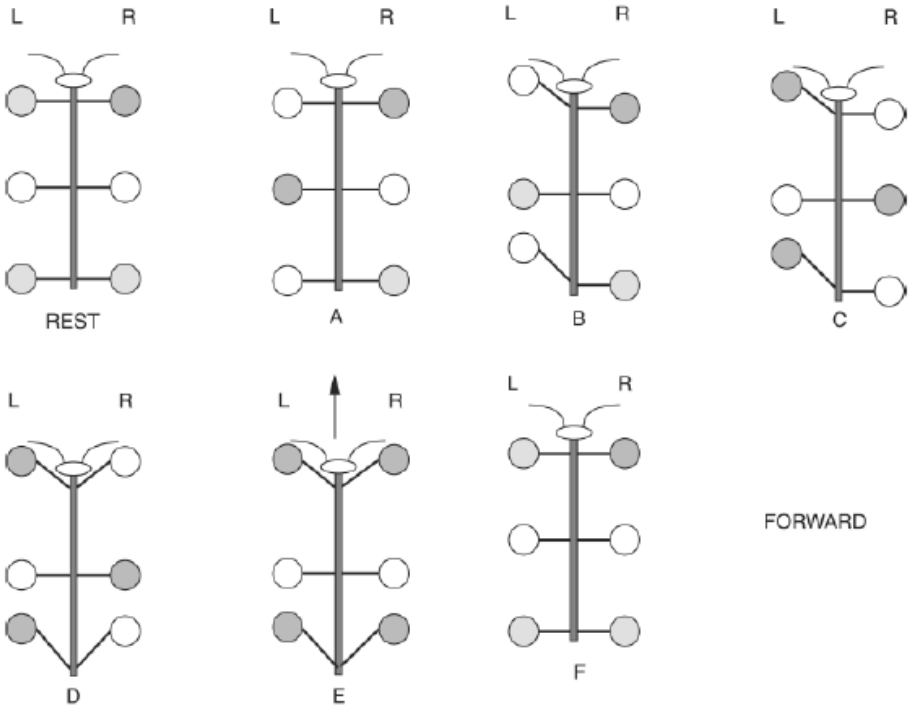


Figure 3.5: Forward gait for hexapod robot

3.2.2 Moving backward

We start in the rest position (see Fig. 3.6), as before. In position A the center legs are rotated CW by about 25 from center position. The robot tilts to the right. The weight distribution is now on the front and back right legs and the center left leg. Since there is no weight on the front and back left legs, they are free to move backward, as shown in the B position of Fig. 3.6
In the C position the center legs are rotated CCW by about 25 from center position. The robot tilts to the left. Since there is no weight on the front and back right legs, they are free to move backward, as shown in the D position.
In position E the center legs are rotated back to their center position. The robot is not in a tilted position, so its weight is distributed on the front and back legs. In the F position, the front and back legs are moved forward simultaneously, causing the robot to move backward. The walking cycle can then repeat

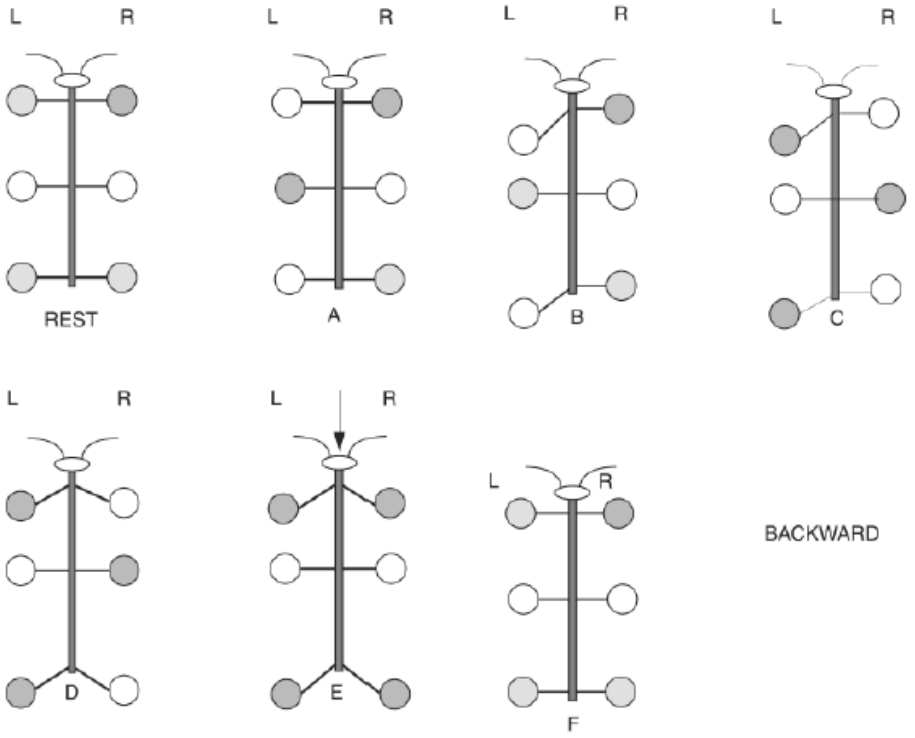


Figure 3.6: Backward gait for hexapod robot

3.2.3 Turning left

The leg motion sequence to turn left is shown in Fig. 3.7. In position A the center legs are rotated CW by about 25 from center position. The robot tilts to the right. The weight distribution is now on the front and back right legs and the center left leg. Since there is no weight on the front and back left legs, they are free to move forward, as shown in Fig. 3.7
In the B position, the center legs are rotated CCW by about 25 from center position. The robot tilts to the left. Since there is no weight on the front and back right legs, they are free to move backward, as shown in the C position
In position D, the center legs are rotated back to their center position. The robot is not in a tilted position, so its weight is distributed on the front and back legs. In position, the left legs moved backward while the right legs moved forward, simultaneously causing the robot to turn left. It typically takes three turning cycles to turn the robot 90.

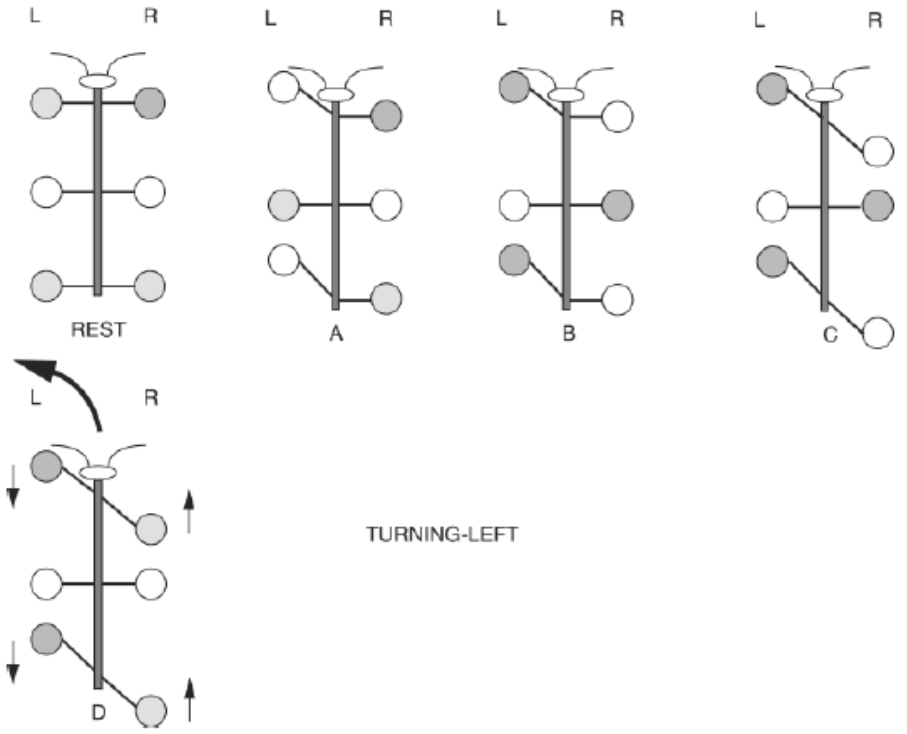


Figure 3.7: Turning-left

3.2.4 Turning right

Turning right follows the same sequence as turning left, with the leg positions reversed.

3.3 Servo control

The motion of the robot is controlled by controlling the servos attached to each leg in a specific pattern.

Introduction

Servo motors are electro-mechanical devices which provide good torque . If you open a servo motor you will see a simple DC motor connected to a potentiometer and as motor rotates output value of potentiometer also changes . Now a servo motor has three wires

rst one for voltage ,second for data and third for ground . Generally value for voltage is 5V but always check your motor ratings. Now see servo motor

expects a pulse every 20ms and width of the pulse varies from 1ms to 2ms. 1ms for maximum rotation in one direction lets say -90 and 2ms for +90
There are many methods for moving servo motors i.e generating the required pulses we used timer0 for generating an interrupt every 20ms and then send a pulse using delay function.
The 12 servo motors are controlled by the pwm signals generated by the PIC microcontroller. The code for motor control is written in C language using MPLAB with HI-TECH C compiler.

In the circuit, PIC16F877A is running on external crystal of 20MHz value. RA0 pin is toggled every time timer0 expires and executes it's ISR(Interrupt Service Routine) code.

PWM generation

The code used to initialize timer0 is shown below.

```
void InitTimer0(void)
{
//Timer0 is 8 bit timer, select TOCS and PSA to be zero
OPTION_REG = 0xC0;  //Make prescalar 1:2

T0IE = 1;       // Enable Timer0 interrupt
GIE = 1;        // Enable global interrupts
}
```

Listing 3.1: Timer0 Initialisation

In this function, OPTION_ REG is initialized to make timer0 prescalar to be 1:2. Timer0 is an 8bit timer, so it expires after reaching a value of 255. When timer0 prescalar is made 1:2 then it means that timer0 value will increment after every two clock cycles. T0IE bit enables timer0 interrupts and GIE bit enables global interrupts.

Timer0 interrupt service routine code is shown below.

```
void interrupt ISR(void)
{
  if(T0IF)  //If Timer0 interrupt is set
  {
    RA0=~RA0; // Toggle RA0 pin
    T0IF=0 ;  // Clear the interrupt
  }
}
```

Listing 3.2: Timer0 Interrupt service routine

Whenever timer0 expires, then an interrupt is generated which executes the function shown above in the Figure 3.2. In this function, RA0 pin is toggled every time and timer0 interrupt flag is cleared. This means that whenever RA0 pin toggles then timer0 interrupt has occurred. In the main function, firstly TRISA0 is made zero to make RA0 pin an output pin, also RA0 pin is made zero. After that, InitTimer0() function is called which initializes timer0. Rest of the work is done in timer0 interrupt service routine as explained below. Every time timer0 expires RA0 pin is toggled.
The following section shows how to interrupt subroutine to control the duty cycle of each PWM channel. In the interrupt subroutine, an interrupt counter is incremented once for every interrupt happened and can be used for the duration calculation of each pins high level output. Assuming the duty cycle of each PWM channel is calculated in the beginning of every interrupt cycle. Comparing the preset duty cycle of each channel and the time of duration, the corresponding pin will be reset as zero if the equal condition is satisfied. Therefore, the duty cycle of each channel is under control. Also, based on the precise time base, a preset interrupt mcounter bound is used to tune the PWM period. Once the upper bound of interrupt counter is achieved, the interrupt counter will be reset as zero and all the pins outputs are reset at high level.

Program 3.3 is the partial code of interrupt subroutine to illustrate above operation. In program 1, interrupt_ counttime_ base denotes the duration time of each PWM cycle. Comparing the duration time with the preset duty cycle of each channel set_ PWM_ duty[i], if the equal condition is satisfied, the corresponding IO pin will be reset as LOW. The final if instruction is used to tune the PWM period (frequency). Once the equal condition is satisfied, the interrupt_ count will be reset as 0 and all the PWM IO pins reset as LOW. The PWM period is count_ upper_ bound time_ base.

```
void interrupt_h_function(void) //interrupt function
{
.....;
.....;
interrupt_count=interrupt_count+1;//interrupt_count
//incremented
if(interrupt_ count ==set_PWM_duty[0])//Is the duty cycle
//of channel 1 achieved?
PWM_1=0;//reset the corresponding IO pin 1 as /LOW
if(interrupt_ count ==set_PWM_duty[1]) //Is the duty
   cycle
//of channel 2 achieved?
PWM_2=0;//reset the corresponding IO pin 2 as //LOW
if(interrupt_ count ==set_PWM_duty[2]) //Is the duty
   cycle
//of channel 3 achieved?
PWM_3=0;//reset the corresponding IO pin 3 as LOW
```

```
.....;
if(interrupt_count==set_PWM_duty[n]) //Is the duty cycle
//of channel n achieved?
PWMn=0;//reset the corresponding IO pin 3 as LOW
if(interrupt_count==count_upper_bound) //Does the
//interrupt_ count upper bound achieved?
{ interrupt_count=0;//reset the interrupt_count value
PORTD=0XFF; //reset the corresponding IO pins
PORTJ=0XFF;
PORTF=0XFF;
PORTB=0XFF;
}
.....;
.....;
}
```

Listing 3.3: ISR

3.3.1 Flowcharts

The flow charts for the main program (Fig. 3.8) and the Interrupt Service Routine (Fig. 3.9) are given below

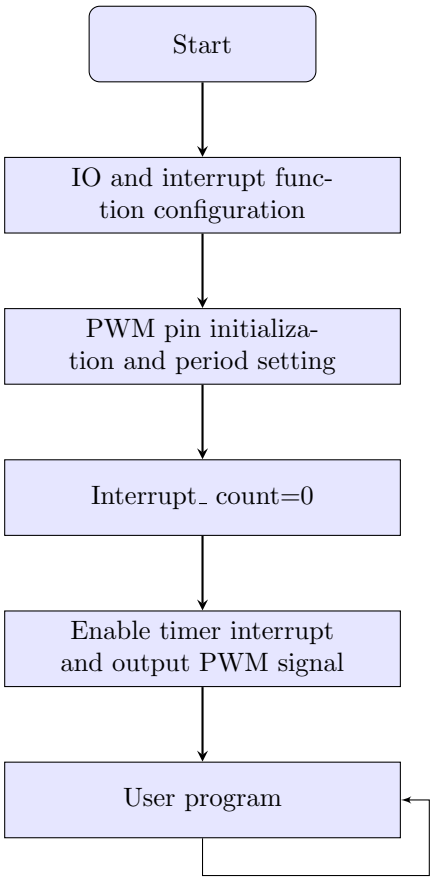


Figure 3.8: Main Program Flowchart

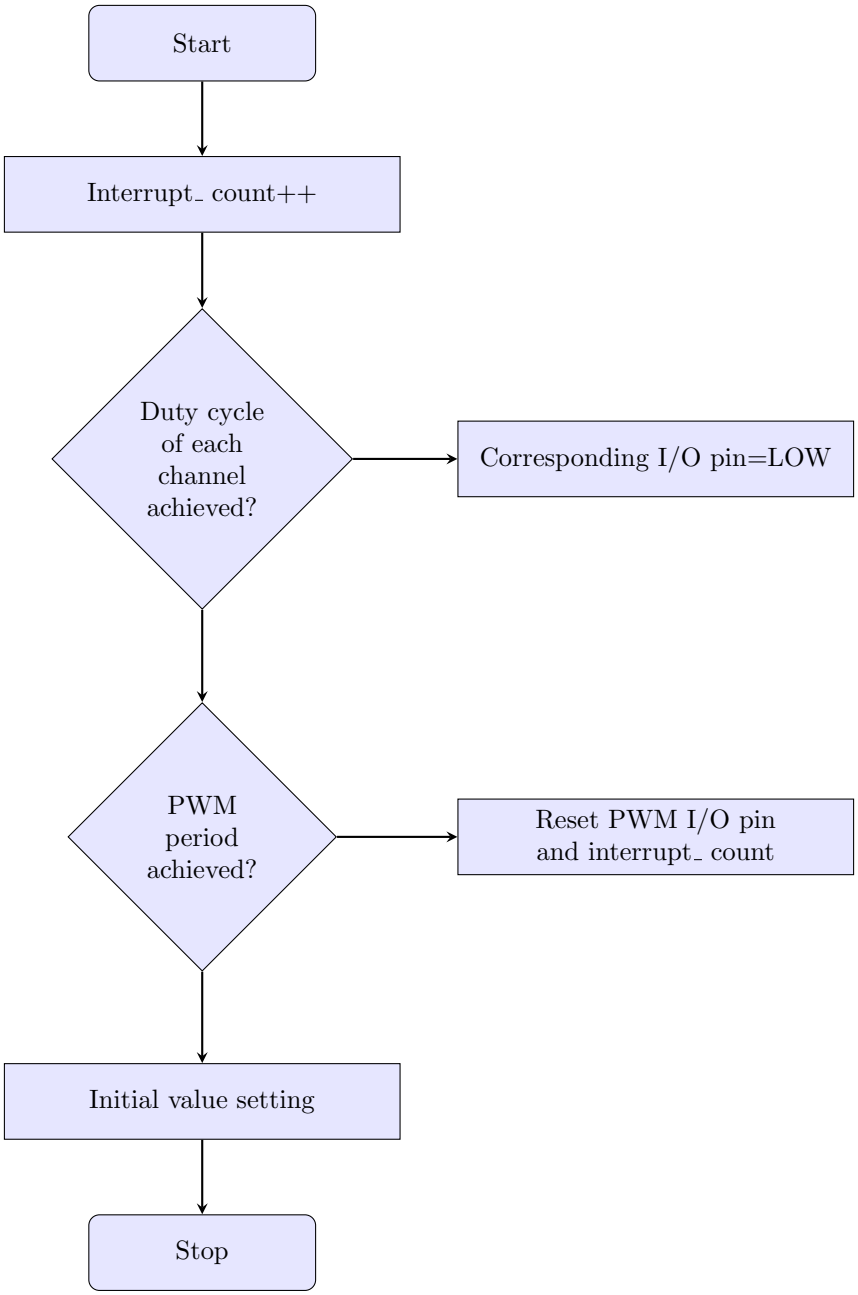


Figure 3.9: Interrupt subroutine flow chart

3.4 GUI creation using Matlab

The guided user interface for the control of the robot was created using Matlab GUI tool. The GUI code used for the project is included in the annexures The final GUI is shown in figure 3.10

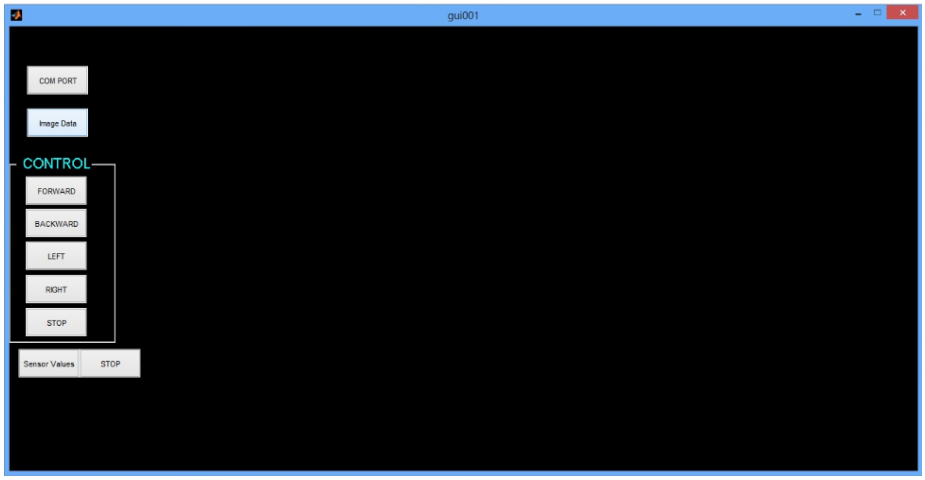


Figure 3.10: GUI created using Matlab

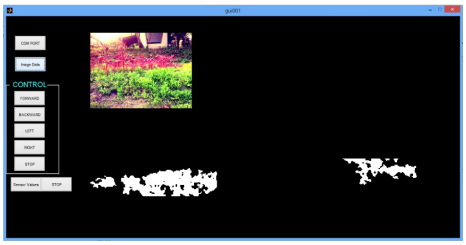
The buttons used are:

- *COM PORT* : To set up zigbee with pc ports
- *Image Data* : To obtain output of weed detection
- CONTROLS
  - *Forward* : Forward command
  - *Backward* : Backward command
  - *Left* : Left movement
  - *Right* : Right movement
  - *Stop* :Stop motion
- *Sensor Data* : To display the sensor values
- *STOP* : To stop reading sensor values

3.5 Weed detection using Matlab

Figure 3.11 shows obtained output for a sample input

(a) Sample Input



(b) Obtained output

Figure 3.11: Weed Detection using Matlab

In the whole image the red plants are supposed to be the crops and the green plants in between them are supposed to be weeds. As shown in figure 3.11b the matlab code succesfully identified the weeds. The code is based on simple colour detection principle. In the future we are planning to expand the project and making it capable of detecting more types of weeds using leaf pattern detection and Neural networks.

Steps involved

1. The intensity of the RGB values are increased using the function imadjust
2. The green part of the image is considered as foreground. Then the im-subtract operation is done between the 'G' dimension values and the grey scale image
3. To suppress the background median filtering is done
4. The resulting image is converted into black and white. Thus the green pixels are now seen as white pixels and the rest of the background is changed to black

5. Next the image is divided into three segments and each segment is checked for white pixels

6. If the number of white pixels is beyond a fixed threshold then the robot decides that there are weeds in the area and destroys the weeds by drilling the roots with a dc motor. Otherwise if the value is less than the threshold then the robot moves on

CHAPTER 4
CONCLUSION AND FUTURE SCOPE

The goal of this project was to create a Hexapod robot for Agricultural applications. The long term goal was to make it intelligent to fully take over all the operations in a farm. The main target operations included:

- Tills the soil within a few centimeters of the seed
- Plants Seeds at precise intervals and records the position of planting
- Scouts the field for weed detection
- Weeding : Applies weedicide at only areas where there is high concentration of weeds

We were able to build a satisfactory prototype of the hexapod robot with six legs each controlled by two servo motors. The hexapod Tripod gait algorithm was also succesfully implemented. The final robot is found to be highly manoeuvrable and versatile and it could traverse even rough terrain easily.
Coming to the Agricultural applications; we successfully implemented the following systems in our prototype:

- Weed detection capability was examined using Matlab colour detection principles
- Obstacle avoidance algorithms were successfully implemented. This could allow the robot to navigate without human interference guided by the boundaries of the farm.
- Sensors on-board worked perfectly showing the possibilities of remote data collection

Building such robot provide an infinite possibilities of applications such as SAR (Search and Rescue) , RFID(Radio-frequency identification) tracker, face detection, fire detector, mine detector and many other military applications. But first we are planning to add micro-sprayers, a mechanical hand and other add-ons to improve the capabilities of the robot in the field of Agriculture.
Some of our other more ambitious plans are listed below:

1. Each leg of the hexapod can be fitted with pressure sensors . The input from these pressure sensors can be used to implement algorithms than change the step lengths and the walking patterns (gaits) of the hexapod according to the external environment

2. It is possible to extract length measurements of real life objects from the captured image using Digital Image Processing . The collected data can be used by the hexapod to avoid too deep holes or very high obstacles

3. In Agriculture; productivity can be improved by using Swarm Robotics. The hexapod in this project has the ability to perform sowing seeds, removing weeds and harvesting more efficiently as discussed in this paper. A swarm of hexapods can thus replace the farmers and perform almost all the farming procedures.

4. Artificial Intelligence algorithms can be used to improve the efficiency of image processing algorithms

5. If the servo motors are replaced by pneumatic pistons or electronic muscle fibres then the robot can support more weight

6. If its possible to design a life sized Hexapod then it will be a more stable means of locomotion for the physically challenged. It will allow them to climb steps and move through rocky terrain without the need of getting down from the machine

REFERENCES

[1] Chin-Pao Hung,Wei-Ging Liu,Hong-Jhe Su,Jia-Wei Chen and Bo-Ming Chiu, *PIC-Based Multi-Channel PWM Signal Generation Method and Application to Motion Control of Six Feet Robot Toy*' International Journal Of Circuits, Systems and Signal Processing, Issue 2,Volume 3, 2009.

[2] Ing.R.Woering,*Simulating the first steps of a walking hexapod robot*,Technische Universiteit Eindhoven,Department Mechanical Engineering,Control Systems Technology Group,Eindhoven,Eindhoven,January,2011.

[3] Pedersen S.M,Fountas S and Blackmore S, *Agricultural Robots-Applications and Economic Prespectives*,University of Copenhagen,Institute of Food and Resource Economics, University of Thessaly,Denmark,Greece.

[4] Tareq Mamkegh,Ahmad Hindash and Mohammad Al-Jabari,*Hexapod Robot-Robot Design,model and control*,German Jordanian University,May 2011.

[5] Michael Margolis, *Arduino Cookbook*, O'Reilly, Second Edition, December 2011, ISBN 13: 978-93-5023-612-3

[6] *Arduino Examples*, http://arduino.cc/en/Tutorial/HomePage

Appendices

Appendix A

16f877a Datasheet

PIC16F87XA

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^2C™ (Master/Slave)
- Universal Synchronous Asynchronous Receiver Transmitter (USART/SCI) with 9-bit address detection
- Parallel Slave Port (PSP) – 8 bits wide with external $\overline{RD}$, $\overline{WR}$ and $\overline{CS}$ 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 (V_{REF}) 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

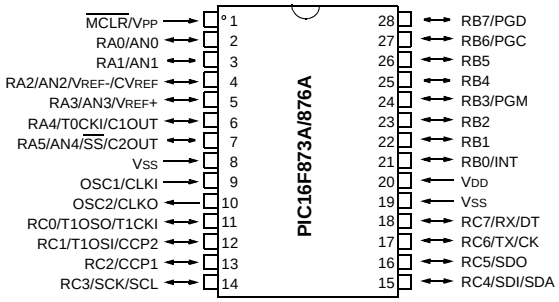
CMOS Technology:

- 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

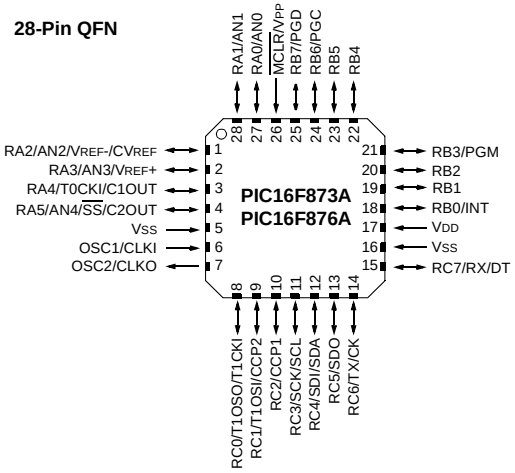
Device	Program Memory		Data SRAM (Bytes)	EEPROM (Bytes)	I/O	10-bit A/D (ch)	CCP (PWM)	MSSP		USART	Timers 8/16-bit	Comparators
	Bytes	# Single Word Instructions						SPI	Master I^2C			
PIC16F873A	7.2K	4096	192	128	22	5	2	Yes	Yes	Yes	2/1	2
PIC16F874A	7.2K	4096	192	128	33	8	2	Yes	Yes	Yes	2/1	2
PIC16F876A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

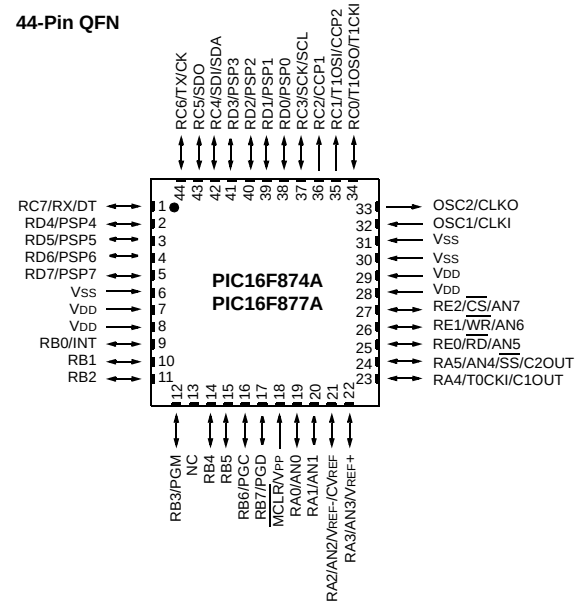
28-Pin PDIP, SOIC, SSOP



28-Pin QFN



44-Pin QFN



Pin Diagrams (Continued)

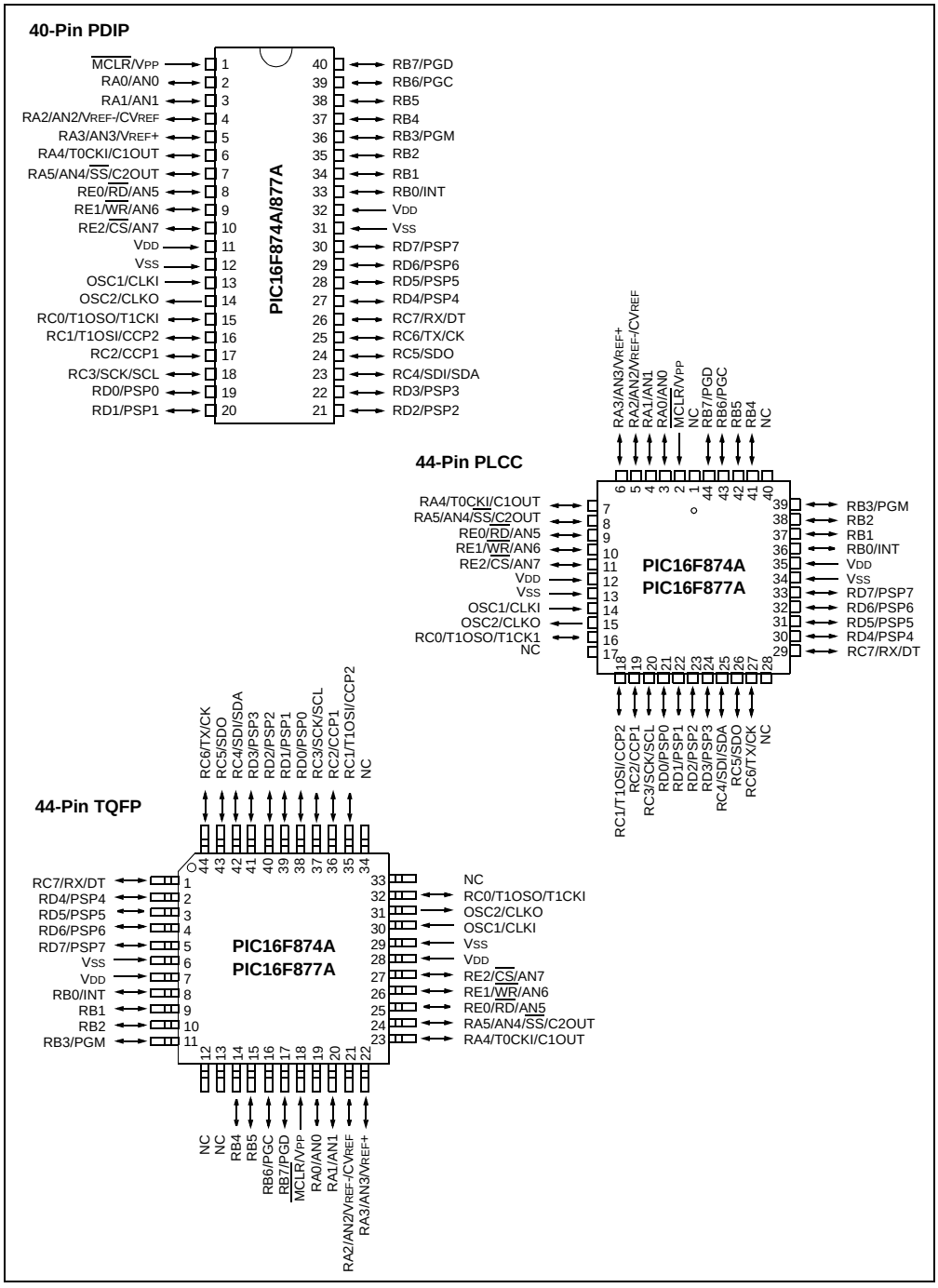
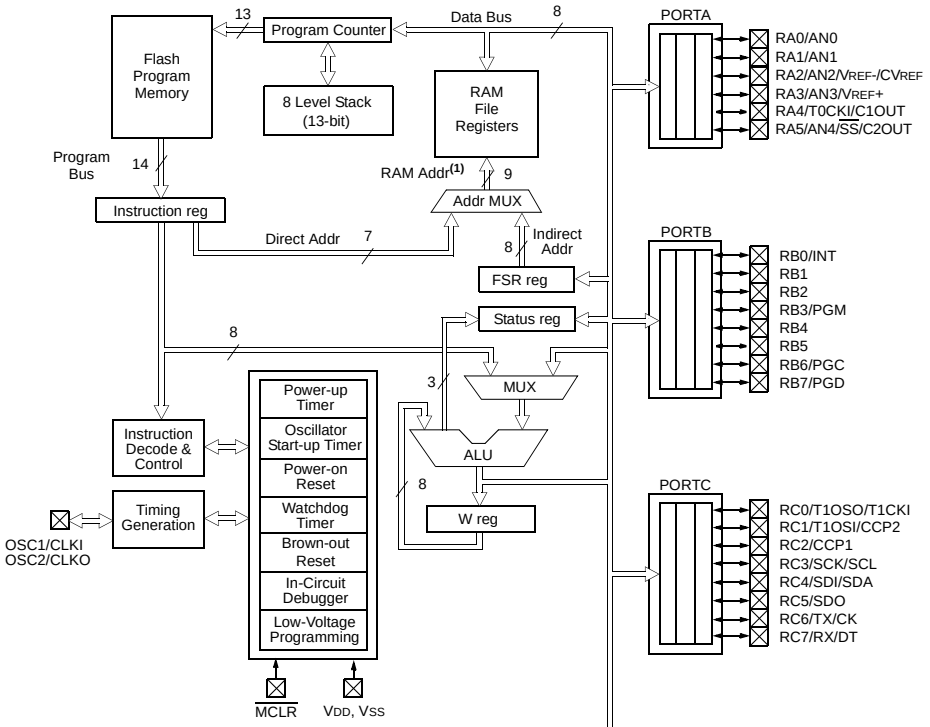


FIGURE 1-1: PIC16F873A/876A BLOCK DIAGRAM



Device	Program Flash	Data Memory	Data EEPROM
PIC16F873A	4K words	192 Bytes	128 Bytes
PIC16F876A	8K words	368 Bytes	256 Bytes

Note 1: Higher order bits are from the Status register.

TABLE 1-2: PIC16F873A/876A PINOUT DESCRIPTION

Pin Name	PDIP, SOIC, SSOP Pin#	QFN Pin#	I/O/P Type	Buffer Type	Description
OSC1/CLKI	9	6		ST/CMOS(3)	Oscillator crystal or external clock input.
OSC1			I		Oscillator crystal input or external clock source input. ST buffer when configured in RC mode; otherwise CMOS.
CLKI			I		External clock source input. Always associated with pin function OSC1 (see OSC1/CLKI, OSC2/CLKO pins).
OSC2/CLKO	10	7		—	Oscillator crystal or clock output.
OSC2			O		Oscillator crystal output. Connects to crystal or resonator in Crystal Oscillator mode.
CLKO			O		In RC mode, OSC2 pin outputs CLKO, which has 1/4 the frequency of OSC1 and denotes the instruction cycle rate.
$\overline{MCLR}$/VPP	1	26		ST	Master Clear (input) or programming voltage (output).
$\overline{MCLR}$			I		Master Clear (Reset) input. This pin is an active low Reset to the device.
VPP			P		Programming voltage input.
					PORTA is a bidirectional I/O port.
RA0/AN0	2	27		TTL	
RA0			I/O		Digital I/O.
AN0			I		Analog input 0.
RA1/AN1	3	28		TTL	
RA1			I/O		Digital I/O.
AN1			I		Analog input 1.
RA2/AN2/VREF-/ CVREF	4	1		TTL	
			I/O		Digital I/O.
RA2			I		Analog input 2.
AN2			I		A/D reference voltage (Low) input.
VREF-			O		Comparator VREF output.
CVREF					
RA3/AN3/VREF+	5	2		TTL	
RA3			I/O		Digital I/O.
AN3			I		Analog input 3.
VREF+			I		A/D reference voltage (High) input.
RA4/T0CKI/C1OUT	6	3		ST	
RA4			I/O		Digital I/O – Open-drain when configured as output.
T0CKI			I		Timer0 external clock input.
C1OUT			O		Comparator 1 output.
RA5/AN4/$\overline{SS}$/C2OUT	7	4		TTL	
RA5			I/O		Digital I/O.
AN4			I		Analog input 4.
$\overline{SS}$			I		SPI slave select input.
C2OUT			O		Comparator 2 output.

Legend: I = input O = output I/O = input/output P = power
— = Not used TTL = TTL input ST = Schmitt Trigger input

Note 1: This buffer is a Schmitt Trigger input when configured as the external interrupt.
2: This buffer is a Schmitt Trigger input when used in Serial Programming mode.
3: This buffer is a Schmitt Trigger input when configured in RC Oscillator mode and a CMOS input otherwise.

TABLE 1-2: PIC16F873A/876A PINOUT DESCRIPTION (CONTINUED)

Pin Name	PDIP, SOIC, SSOP Pin#	QFN Pin#	I/O/P Type	Buffer Type	Description
					PORTB is a bidirectional I/O port. PORTB can be software programmed for internal weak pull-ups on all inputs.
RB0/INT	21	18		TTL/ST(1)	
RB0			I/O		Digital I/O.
INT			I		External interrupt.
RB1	22	19	I/O	TTL	Digital I/O.
RB2	23	20	I/O	TTL	Digital I/O.
RB3/PGM	24	21		TTL	
RB3			I/O		Digital I/O.
PGM			I		Low-voltage (single-supply) ICSP programming enable pin.
RB4	25	22	I/O	TTL	Digital I/O.
RB5	26	23	I/O	TTL	Digital I/O.
RB6/PGC	27	24		TTL/ST(2)	
RB6			I/O		Digital I/O.
PGC			I		In-circuit debugger and ICSP programming clock.
RB7/PGD	28	25		TTL/ST(2)	
RB7			I/O		Digital I/O.
PGD			I/O		In-circuit debugger and ICSP programming data.
					PORTC is a bidirectional I/O port.
RC0/T1OSO/T1CKI	11	8		ST	
RC0			I/O		Digital I/O.
T1OSO			O		Timer1 oscillator output.
T1CKI			I		Timer1 external clock input.
RC1/T1OSI/CCP2	12	9		ST	
RC1			I/O		Digital I/O.
T1OSI			I		Timer1 oscillator input.
CCP2			I/O		Capture2 input, Compare2 output, PWM2 output.
RC2/CCP1	13	10		ST	
RC2			I/O		Digital I/O.
CCP1			I/O		Capture1 input, Compare1 output, PWM1 output.
RC3/SCK/SCL	14	11		ST	
RC3			I/O		Digital I/O.
SCK			I/O		Synchronous serial clock input/output for SPI mode.
SCL			I/O		Synchronous serial clock input/output for I^2C mode.
RC4/SDI/SDA	15	12		ST	
RC4			I/O		Digital I/O.
SDI			I		SPI data in.
SDA			I/O		I^2C data I/O.
RC5/SDO	16	13		ST	
RC5			I/O		Digital I/O.
SDO			O		SPI data out.
RC6/TX/CK	17	14		ST	
RC6			I/O		Digital I/O.
TX			O		USART asynchronous transmit.
CK			I/O		USART1 synchronous clock.
RC7/RX/DT	18	15		ST	
RC7			I/O		Digital I/O.
RX			I		USART asynchronous receive.
DT			I/O		USART synchronous data.
VSS	8, 19	5, 6	P	—	Ground reference for logic and I/O pins.
VDD	20	17	P	—	Positive supply for logic and I/O pins.

Legend: I = input O = output I/O = input/output P = power
— = Not used TTL = TTL input ST = Schmitt Trigger input

Note 1: This buffer is a Schmitt Trigger input when configured as the external interrupt.
2: This buffer is a Schmitt Trigger input when used in Serial Programming mode.
3: This buffer is a Schmitt Trigger input when configured in RC Oscillator mode and a CMOS input otherwise.

TABLE 1-3: PIC16F874A/877A PINOUT DESCRIPTION (CONTINUED)

Pin Name	PDIP Pin#	PLCC Pin#	TQFP Pin#	QFN Pin#	I/O/P Type	Buffer Type	Description
							PORTB is a bidirectional I/O port. PORTB can be software programmed for internal weak pull-up on all inputs.
RB0/INT	33	36	8	9		TTL/ST(1)	
RB0					I/O		Digital I/O.
INT					I		External interrupt.
RB1	34	37	9	10	I/O	TTL	Digital I/O.
RB2	35	38	10	11	I/O	TTL	Digital I/O.
RB3/PGM	36	39	11	12		TTL	
RB3					I/O		Digital I/O.
PGM					I		Low-voltage ICSP programming enable pin.
RB4	37	41	14	14	I/O	TTL	Digital I/O.
RB5	38	42	15	15	I/O	TTL	Digital I/O.
RB6/PGC	39	43	16	16		TTL/ST(2)	
RB6					I/O		Digital I/O.
PGC					I		In-circuit debugger and ICSP programming clock.
RB7/PGD	40	44	17	17		TTL/ST(2)	
RB7					I/O		Digital I/O.
PGD					I/O		In-circuit debugger and ICSP programming data.

Legend: I = input O = output I/O = input/output P = power
— = Not used TTL = TTL input ST = Schmitt Trigger input

Note 1: This buffer is a Schmitt Trigger input when configured as the external interrupt.
2: This buffer is a Schmitt Trigger input when used in Serial Programming mode.
3: This buffer is a Schmitt Trigger input when configured in RC Oscillator mode and a CMOS input otherwise.

2.2.2 SPECIAL FUNCTION REGISTERS

The Special Function Registers are registers used by the CPU and peripheral modules for controlling the desired operation of the device. These registers are implemented as static RAM. A list of these registers is given in Table 2-1.

The Special Function Registers can be classified into two sets: core (CPU) and peripheral. Those registers associated with the core functions are described in detail in this section. Those related to the operation of the peripheral features are described in detail in the peripheral features section.

TABLE 2-1: SPECIAL FUNCTION REGISTER SUMMARY

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Details on page:
Bank 0											
00h(3)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	31, 150
01h	TMR0	Timer0 Module Register								xxxx xxxx	55, 150
02h(3)	PCL	Program Counter (PC) Least Significant Byte								0000 0000	30, 150
03h(3)	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	22, 150
04h(3)	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	31, 150
05h	PORTA	—	—	PORTA Data Latch when written: PORTA pins when read						--0x 0000	43, 150
06h	PORTB	PORTB Data Latch when written: PORTB pins when read								xxxx xxxx	45, 150
07h	PORTC	PORTC Data Latch when written: PORTC pins when read								xxxx xxxx	47, 150
08h(4)	PORTD	PORTD Data Latch when written: PORTD pins when read								xxxx xxxx	48, 150
09h(4)	PORTE	—	—	—	—	—	RE2	RE1	RE0	---- -xxx	49, 150
0Ah(1,3)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	30, 150
0Bh(3)	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	24, 150
0Ch	PIR1	PSPIF(3)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	26, 150
0Dh	PIR2	—	CMIF	—	EEIF	BCLIF	—	—	CCP2IF	-0-0 0--0	28, 150
0Eh	TMR1L	Holding Register for the Least Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	60, 150
0Fh	TMR1H	Holding Register for the Most Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	60, 150
10h	T1CON	—	—	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{T1SYNC}$	TMR1CS	TMR1ON	--00 0000	57, 150
11h	TMR2	Timer2 Module Register								0000 0000	62, 150
12h	T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	-000 0000	61, 150
13h	SSPBUF	Synchronous Serial Port Receive Buffer/Transmit Register								xxxx xxxx	79, 150
14h	SSPCON	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0	0000 0000	82, 82, 150
15h	CCPR1L	Capture/Compare/PWM Register 1 (LSB)								xxxx xxxx	63, 150
16h	CCPR1H	Capture/Compare/PWM Register 1 (MSB)								xxxx xxxx	63, 150
17h	CCP1CON	—	—	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0	--00 0000	64, 150
18h	RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 000x	112, 150
19h	TXREG	USART Transmit Data Register								0000 0000	118, 150
1Ah	RCREG	USART Receive Data Register								0000 0000	118, 150
1Bh	CCPR2L	Capture/Compare/PWM Register 2 (LSB)								xxxx xxxx	63, 150
1Ch	CCPR2H	Capture/Compare/PWM Register 2 (MSB)								xxxx xxxx	63, 150
1Dh	CCP2CON	—	—	CCP2X	CCP2Y	CCP2M3	CCP2M2	CCP2M1	CCP2M0	--00 0000	64, 150
1Eh	ADRESH	A/D Result Register High Byte								xxxx xxxx	133, 150
1Fh	ADCON0	ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{DONE}$	—	ADON	0000 00-0	127, 150

Legend: x = unknown, u = unchanged, q = value depends on condition, - = unimplemented, read as '0', r = reserved. Shaded locations are unimplemented, read as '0'.

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a holding register for the PC<12:8>, whose contents are transferred to the upper byte of the program counter.

2: Bits PSPIE and PSPIF are reserved on PIC16F873A/876A devices; always maintain these bits clear.

3: These registers can be addressed from any bank.

4: PORTD, PORTE, TRISD and TRISE are not implemented on PIC16F873A/876A devices, read as '0'.

5: Bit 4 of EEADRH implemented only on the PIC16F876A/877A devices.

TABLE 2-1: SPECIAL FUNCTION REGISTER SUMMARY (CONTINUED)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Details on page:
Bank 1											
80h(3)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	31, 150
81h	OPTION_REG	$\overline{RBPU}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	23, 150
82h(3)	PCL	Program Counter (PC) Least Significant Byte								0000 0000	30, 150
83h(3)	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	22, 150
84h(3)	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	31, 150
85h	TRISA	—	—	PORTA Data Direction Register						--11 1111	43, 150
86h	TRISB	PORTB Data Direction Register								1111 1111	45, 150
87h	TRISC	PORTC Data Direction Register								1111 1111	47, 150
88h(4)	TRISD	PORTD Data Direction Register								1111 1111	48, 151
89h(4)	TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction bits			0000 -111	50, 151
8Ah(1,3)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	30, 150
8Bh(3)	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	24, 150
8Ch	PIE1	PSPIE(2)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	25, 151
8Dh	PIE2	—	CMIE	—	EEIE	BCLIE	—	—	CCP2IE	-0-0 0--0	27, 151
8Eh	PCON	—	—	—	—	—	—	$\overline{POR}$	$\overline{BOR}$	---- --qq	29, 151
8Fh	—	Unimplemented								—	—
90h	—	Unimplemented								—	—
91h	SSPCON2	GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN	0000 0000	83, 151
92h	PR2	Timer2 Period Register								1111 1111	62, 151
93h	SSPADD	Synchronous Serial Port (I^2C mode) Address Register								0000 0000	79, 151
94h	SSPSTAT	SMP	CKE	D/$\overline{A}$	P	S	R/$\overline{W}$	UA	BF	0000 0000	79, 151
95h	—	Unimplemented								—	—
96h	—	Unimplemented								—	—
97h	—	Unimplemented								—	—
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	111, 151
99h	SPBRG	Baud Rate Generator Register								0000 0000	113, 151
9Ah	—	Unimplemented								—	—
9Bh	—	Unimplemented								—	—
9Ch	CMCON	C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0	0000 0111	135, 151
9Dh	CVRCON	CVREN	CVROE	CVRR	—	CVR3	CVR2	CVR1	CVR0	000- 0000	141, 151
9Eh	ADRESL	A/D Result Register Low Byte								xxxx xxxx	133, 151
9Fh	ADCON1	ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0	00-- 0000	128, 151

Legend: x = unknown, u = unchanged, q = value depends on condition, - = unimplemented, read as '0', r = reserved. Shaded locations are unimplemented, read as '0'.

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a holding register for the PC<12:8>, whose contents are transferred to the upper byte of the program counter.

2: Bits PSPIE and PSPIF are reserved on PIC16F873A/876A devices; always maintain these bits clear.

3: These registers can be addressed from any bank.

4: PORTD, PORTE, TRISD and TRISE are not implemented on PIC16F873A/876A devices, read as '0'.

5: Bit 4 of EEADRH implemented only on the PIC16F876A/877A devices.

TABLE 4-5: PORTC FUNCTIONS

Name	Bit#	Buffer Type	Function
RC0/T1OSO/T1CKI	bit 0	ST	Input/output port pin or Timer1 oscillator output/Timer1 clock input.
RC1/T1OSI/CCP2	bit 1	ST	Input/output port pin or Timer1 oscillator input or Capture2 input/ Compare2 output/PWM2 output.
RC2/CCP1	bit 2	ST	Input/output port pin or Capture1 input/Compare1 output/ PWM1 output.
RC3/SCK/SCL	bit 3	ST	RC3 can also be the synchronous serial clock for both SPI and I^2C modes.
RC4/SDI/SDA	bit 4	ST	RC4 can also be the SPI data in (SPI mode) or data I/O (I^2C mode).
RC5/SDO	bit 5	ST	Input/output port pin or Synchronous Serial Port data output.
RC6/TX/CK	bit 6	ST	Input/output port pin or USART asynchronous transmit or synchronous clock.
RC7/RX/DT	bit 7	ST	Input/output port pin or USART asynchronous receive or synchronous data.

Legend: ST = Schmitt Trigger input

TABLE 4-6: SUMMARY OF REGISTERS ASSOCIATED WITH PORTC

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
07h	PORTC	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	xxxx xxxx	uuuu uuuu
87h	TRISC	PORTC Data Direction Register								1111 1111	1111 1111

Legend: x = unknown, u = unchanged

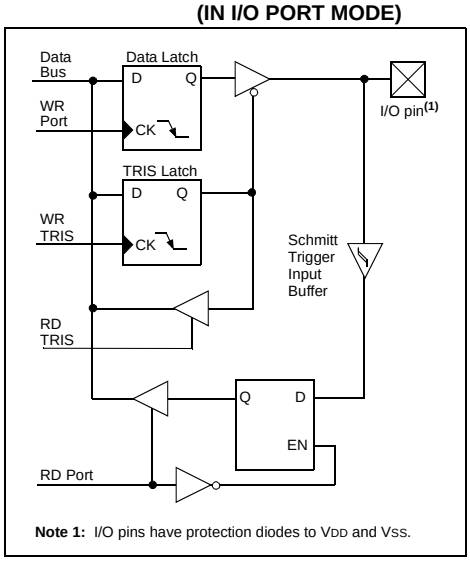
4.4 PORTD and TRISD Registers

Note:	PORTD and TRISD are not implemented on the 28-pin devices.

PORTD is an 8-bit port with Schmitt Trigger input buffers. Each pin is individually configurable as an input or output.

PORTD can be configured as an 8-bit wide microprocessor port (Parallel Slave Port) by setting control bit, PSPMODE (TRISE<4>). In this mode, the input buffers are TTL.

FIGURE 4-8: PORTD BLOCK DIAGRAM (IN I/O PORT MODE)



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

TABLE 4-7: PORTD FUNCTIONS

Name	Bit#	Buffer Type	Function
RD0/PSP0	bit 0	ST/TTL(1)	Input/output port pin or Parallel Slave Port bit 0.
RD1/PSP1	bit 1	ST/TTL(1)	Input/output port pin or Parallel Slave Port bit 1.
RD2/PSP2	bit2	ST/TTL(1)	Input/output port pin or Parallel Slave Port bit 2.
RD3/PSP3	bit 3	ST/TTL(1)	Input/output port pin or Parallel Slave Port bit 3.
RD4/PSP4	bit 4	ST/TTL(1)	Input/output port pin or Parallel Slave Port bit 4.
RD5/PSP5	bit 5	ST/TTL(1)	Input/output port pin or Parallel Slave Port bit 5.
RD6/PSP6	bit 6	ST/TTL(1)	Input/output port pin or Parallel Slave Port bit 6.
RD7/PSP7	bit 7	ST/TTL(1)	Input/output port pin or Parallel Slave Port bit 7.

Legend: ST = Schmitt Trigger input, TTL = TTL input

Note 1: Input buffers are Schmitt Triggers when in I/O mode and TTL buffers when in Parallel Slave Port mode.

TABLE 4-8: SUMMARY OF REGISTERS ASSOCIATED WITH PORTD

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
08h	PORTD	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0	xxxx xxxx	uuuu uuuu
88h	TRISD	PORTD Data Direction Register								1111 1111	1111 1111
89h	TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction Bits			0000 -111	0000 -111

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

4.5 PORTE and TRISE Register

Note: PORTE and TRISE are not implemented on the 28-pin devices.

PORTE has three pins (RE0/$\overline{RD}$/AN5, RE1/$\overline{WR}$/AN6 and RE2/$\overline{CS}$/AN7) which are individually configurable as inputs or outputs. These pins have Schmitt Trigger input buffers.

The PORTE pins become the I/O control inputs for the microprocessor port when bit PSPMODE (TRISE<4>) is set. In this mode, the user must make certain that the TRISE<2:0> bits are set and that the pins are configured as digital inputs. Also, ensure that ADCON1 is configured for digital I/O. In this mode, the input buffers are TTL.

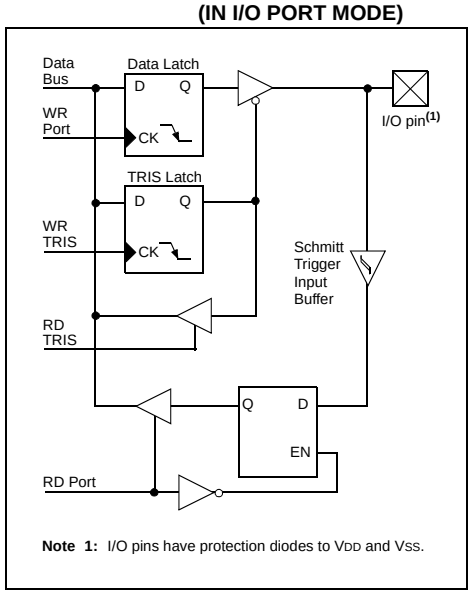
Register 4-1 shows the TRISE register which also controls the Parallel Slave Port operation.

PORTE pins are multiplexed with analog inputs. When selected for analog input, these pins will read as '0's.

TRISE controls the direction of the RE pins, even when they are being used as analog inputs. The user must make sure to keep the pins configured as inputs when using them as analog inputs.

Note: On a Power-on Reset, these pins are configured as analog inputs and read as '0'.

FIGURE 4-9: PORTE BLOCK DIAGRAM (IN I/O PORT MODE)



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

TABLE 4-9: PORTE FUNCTIONS

Name	Bit#	Buffer Type	Function
RE0/$\overline{RD}$/AN5	bit 0	ST/TTL(1)	I/O port pin or read control input in Parallel Slave Port mode or analog input: $\overline{RD}$ 1 = Idle 0 = Read operation. Contents of PORTD register are output to PORTD I/O pins (if chip selected).
RE1/$\overline{WR}$/AN6	bit 1	ST/TTL(1)	I/O port pin or write control input in Parallel Slave Port mode or analog input: $\overline{WR}$ 1 = Idle 0 = Write operation. Value of PORTD I/O pins is latched into PORTD register (if chip selected).
RE2/$\overline{CS}$/AN7	bit 2	ST/TTL(1)	I/O port pin or chip select control input in Parallel Slave Port mode or analog input: $\overline{CS}$ 1 = Device is not selected 0 = Device is selected

Legend: ST = Schmitt Trigger input, TTL = TTL input

Note 1: Input buffers are Schmitt Triggers when in I/O mode and TTL buffers when in Parallel Slave Port mode.

9.4.6 MASTER MODE

Master mode is enabled by setting and clearing the appropriate SSPM bits in SSPCON and by setting the SSPEN bit. In Master mode, the SCL and SDA lines are manipulated by the MSSP hardware.

Master mode of operation is supported by interrupt generation on the detection of the Start and Stop conditions. The Stop (P) and Start (S) bits are cleared from a Reset or when the MSSP module is disabled. Control of the I^2C bus may be taken when the P bit is set or the bus is Idle, with both the S and P bits clear.

In Firmware Controlled Master mode, user code conducts all I^2C bus operations based on Start and Stop bit conditions.

Once Master mode is enabled, the user has six options.

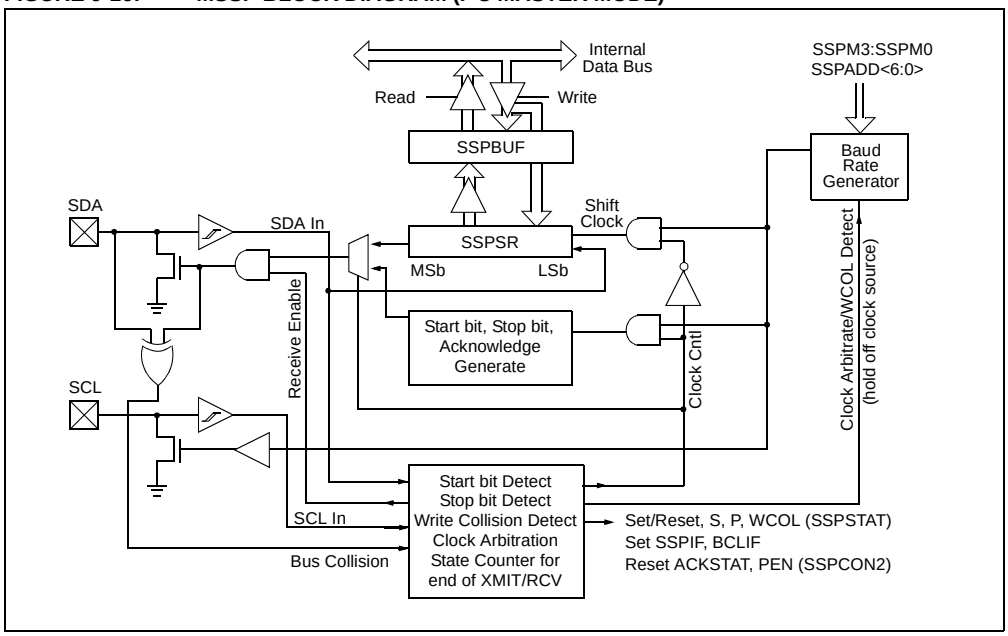
1. Assert a Start condition on SDA and SCL.
2. Assert a Repeated Start condition on SDA and SCL.
3. Write to the SSPBUF register, initiating transmission of data/address.
4. Configure the I^2C port to receive data.
5. Generate an Acknowledge condition at the end of a received byte of data.
6. Generate a Stop condition on SDA and SCL.

Note: The MSSP module, when configured in I^2C Master mode, does not allow queueing of events. For instance, the user is not allowed to initiate a Start condition and immediately write the SSPBUF register to initiate transmission before the Start condition is complete. In this case, the SSPBUF will not be written to and the WCOL bit will be set, indicating that a write to the SSPBUF did not occur.

The following events will cause SSP Interrupt Flag bit, SSPIF, to be set (SSP interrupt if enabled):

- Start condition
- Stop condition
- Data transfer byte transmitted/received
- Acknowledge transmit
- Repeated Start

FIGURE 9-16: MSSP BLOCK DIAGRAM (I^2C MASTER MODE)



9.4.6.1 I²C Master Mode Operation

The master device generates all of the serial clock pulses and the Start and Stop conditions. A transfer is ended with a Stop condition or with a Repeated Start condition. Since the Repeated Start condition is also the beginning of the next serial transfer, the I²C bus will not be released.

In Master Transmitter mode, serial data is output through SDA while SCL outputs the serial clock. The first byte transmitted contains the slave address of the receiving device (7 bits) and the Read/Write (R/$\overline{W}$) bit. In this case, the R/$\overline{W}$ bit will be logic '0'. Serial data is transmitted 8 bits at a time. After each byte is transmitted, an Acknowledge bit is received. Start and Stop conditions are output to indicate the beginning and the end of a serial transfer.

In Master Receive mode, the first byte transmitted contains the slave address of the transmitting device (7 bits) and the R/$\overline{W}$ bit. In this case, the R/$\overline{W}$ bit will be logic '1'. Thus, the first byte transmitted is a 7-bit slave address followed by a '1' to indicate the receive bit. Serial data is received via SDA while SCL outputs the serial clock. Serial data is received 8 bits at a time. After each byte is received, an Acknowledge bit is transmitted. Start and Stop conditions indicate the beginning and end of transmission.

The baud rate generator used for the SPI mode operation is used to set the SCL clock frequency for either 100 kHz, 400 kHz or 1 MHz I²C operation. See **Section 9.4.7 "Baud Rate Generator"** for more detail.

A typical transmit sequence would go as follows:

1. The user generates a Start condition by setting the Start Enable bit, SEN (SSPCON2<0>).
2. SSPIF is set. The MSSP module will wait the required Start time before any other operation takes place.
3. The user loads the SSPBUF with the slave address to transmit.
4. Address is shifted out the SDA pin until all 8 bits are transmitted.
5. The MSSP module shifts in the ACK bit from the slave device and writes its value into the SSPCON2 register (SSPCON2<6>).
6. The MSSP module generates an interrupt at the end of the ninth clock cycle by setting the SSPIF bit.
7. The user loads the SSPBUF with eight bits of data.
8. Data is shifted out the SDA pin until all 8 bits are transmitted.
9. The MSSP module shifts in the ACK bit from the slave device and writes its value into the SSPCON2 register (SSPCON2<6>).
10. The MSSP module generates an interrupt at the end of the ninth clock cycle by setting the SSPIF bit.
11. The user generates a Stop condition by setting the Stop Enable bit, PEN (SSPCON2<2>).
12. Interrupt is generated once the Stop condition is complete.

9.4.7 BAUD RATE GENERATOR

In I^2C Master mode, the Baud Rate Generator (BRG) reload value is placed in the lower 7 bits of the SSPADD register (Figure 9-17). When a write occurs to SSPBUF, the Baud Rate Generator will automatically begin counting. The BRG counts down to 0 and stops until another reload has taken place. The BRG count is decremented twice per instruction cycle (TCY) on the Q2 and Q4 clocks. In I^2C Master mode, the BRG is reloaded automatically.

Once the given operation is complete (i.e., transmission of the last data bit is followed by $\overline{ACK}$), the internal clock will automatically stop counting and the SCL pin will remain in its last state.

Table 9-3 demonstrates clock rates based on instruction cycles and the BRG value loaded into SSPADD.

FIGURE 9-17: BAUD RATE GENERATOR BLOCK DIAGRAM

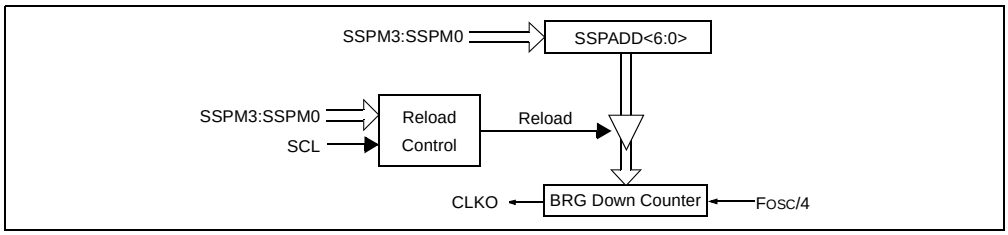


TABLE 9-3: I^2C CLOCK RATE W/BRG

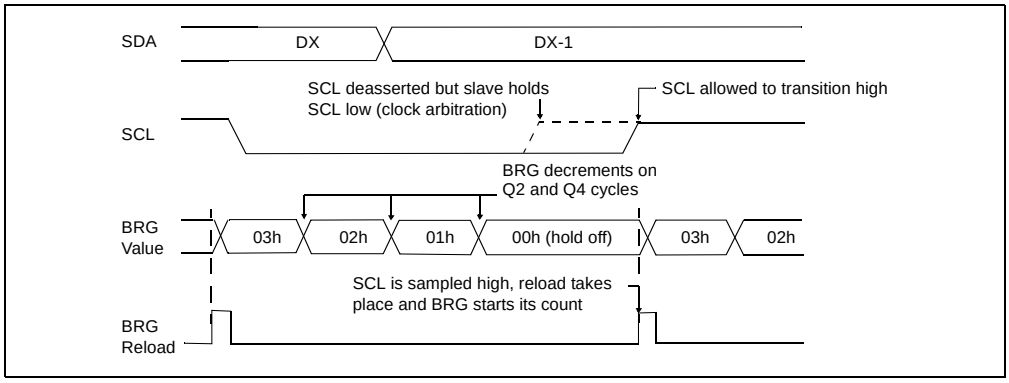
FCY	FCY*2	BRG Value	FSCL (2 Rollovers of BRG)
10 MHz	20 MHz	19h	400 kHz(1)
10 MHz	20 MHz	20h	312.5 kHz
10 MHz	20 MHz	3Fh	100 kHz
4 MHz	8 MHz	0Ah	400 kHz(1)
4 MHz	8 MHz	0Dh	308 kHz
4 MHz	8 MHz	28h	100 kHz
1 MHz	2 MHz	03h	333 kHz(1)
1 MHz	2 MHz	0Ah	100 kHz
1 MHz	2 MHz	00h	1 MHz(1)

Note 1: The I^2C interface does not conform to the 400 kHz I^2C specification (which applies to rates greater than 100 kHz) in all details, but may be used with care where higher rates are required by the application.

9.4.7.1 Clock Arbitration

Clock arbitration occurs when the master, during any receive, transmit or Repeated Start/Stop condition, deasserts the SCL pin (SCL allowed to float high). When the SCL pin is allowed to float high, the Baud Rate Generator (BRG) is suspended from counting until the SCL pin is actually sampled high. When the SCL pin is sampled high, the Baud Rate Generator is reloaded with the contents of SSPADD<6:0> and begins counting. This ensures that the SCL high time will always be at least one BRG rollover count, in the event that the clock is held low by an external device (Figure 9-17).

FIGURE 9-18: BAUD RATE GENERATOR TIMING WITH CLOCK ARBITRATION



9.4.8 I²C MASTER MODE START CONDITION TIMING

To initiate a Start condition, the user sets the Start condition enable bit, SEN (SSPCON2<0>). If the SDA and SCL pins are sampled high, the Baud Rate Generator is reloaded with the contents of SSPADD<6:0> and starts its count. If SCL and SDA are both sampled high when the Baud Rate Generator times out (TBRG), the SDA pin is driven low. The action of the SDA being driven low, while SCL is high, is the Start condition and causes the S bit (SSPSTAT<3>) to be set. Following this, the Baud Rate Generator is reloaded with the contents of SSPADD<6:0> and resumes its count. When the Baud Rate Generator times out (TBRG), the SEN bit (SSPCON2<0>) will be automatically cleared by hardware, the Baud Rate Generator is suspended, leaving the SDA line held low and the Start condition is complete.

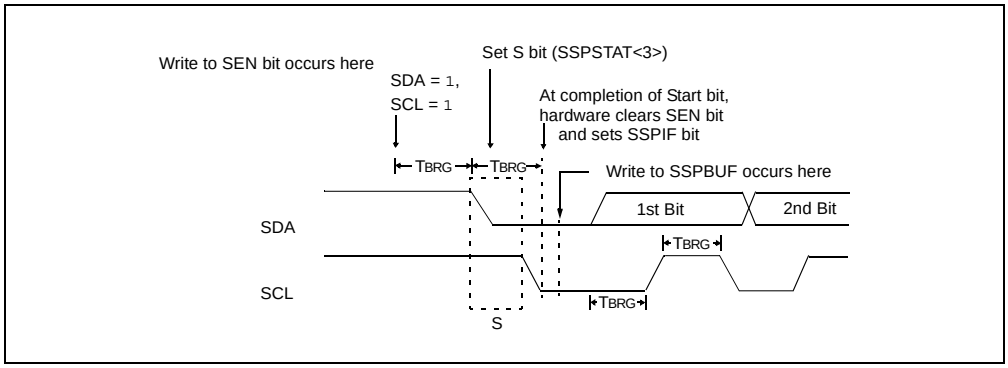
Note:	If at the beginning of the Start condition, the SDA and SCL pins are already sampled low, or if during the Start condition, the SCL line is sampled low before the SDA line is driven low, a bus collision occurs, the Bus Collision Interrupt Flag (BCLIF) is set, the Start condition is aborted and the I²C module is reset into its Idle state.

9.4.8.1 WCOL Status Flag

If the user writes the SSPBUF when a Start sequence is in progress, the WCOL is set and the contents of the buffer are unchanged (the write doesn't occur).

Note:	Because queueing of events is not allowed, writing to the lower 5 bits of SSPCON2 is disabled until the Start condition is complete.

FIGURE 9-19: FIRST START BIT TIMING



9.4.9 I²C MASTER MODE REPEATED START CONDITION TIMING

A Repeated Start condition occurs when the RSEN bit (SSPCON2<1>) is programmed high and the I²C logic module is in the Idle state. When the RSEN bit is set, the SCL pin is asserted low. When the SCL pin is sampled low, the Baud Rate Generator is loaded with the contents of SSPADD<5:0> and begins counting. The SDA pin is released (brought high) for one Baud Rate Generator count (TBRG). When the Baud Rate Generator times out, if SDA is sampled high, the SCL pin will be deasserted (brought high). When SCL is sampled high, the Baud Rate Generator is reloaded with the contents of SSPADD<6:0> and begins counting. SDA and SCL must be sampled high for one TBRG. This action is then followed by assertion of the SDA pin (SDA = 0) for one TBRG while SCL is high. Following this, the RSEN bit (SSPCON2<1>) will be automatically cleared and the Baud Rate Generator will not be reloaded, leaving the SDA pin held low. As soon as a Start condition is detected on the SDA and SCL pins, the S bit (SSPSTAT<3>) will be set. The SSPIF bit will not be set until the Baud Rate Generator has timed out.

Note 1: If RSEN is programmed while any other event is in progress, it will not take effect.

2: A bus collision during the Repeated Start condition occurs if:

- SDA is sampled low when SCL goes from low to high.
- SCL goes low before SDA is asserted low. This may indicate that another master is attempting to transmit a data '1'.

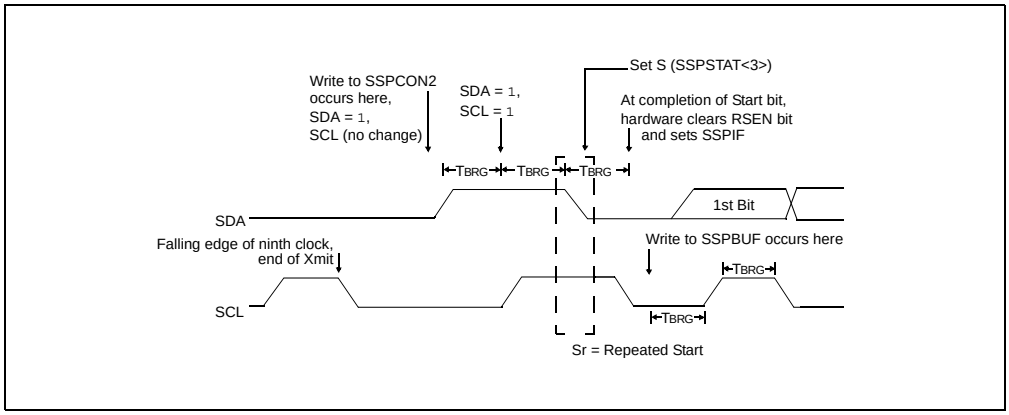
Immediately following the SSPIF bit getting set, the user may write the SSPBUF with the 7-bit address in 7-bit mode or the default first address in 10-bit mode. After the first eight bits are transmitted and an ACK is received, the user may then transmit an additional eight bits of address (10-bit mode) or eight bits of data (7-bit mode).

9.4.9.1 WCOL Status Flag

If the user writes the SSPBUF when a Repeated Start sequence is in progress, the WCOL is set and the contents of the buffer are unchanged (the write doesn't occur).

Note: Because queueing of events is not allowed, writing of the lower 5 bits of SSPCON2 is disabled until the Repeated Start condition is complete.

FIGURE 9-20: REPEAT START CONDITION WAVEFORM



9.4.10 I²C MASTER MODE TRANSMISSION

Transmission of a data byte, a 7-bit address or the other half of a 10-bit address is accomplished by simply writing a value to the SSPBUF register. This action will set the Buffer Full flag bit, BF, and allow the Baud Rate Generator to begin counting and start the next transmission. Each bit of address/data will be shifted out onto the SDA pin after the falling edge of SCL is asserted (see data hold time specification, parameter #106). SCL is held low for one Baud Rate Generator rollover count (TBRG). Data should be valid before SCL is released high (see data setup time specification, parameter #107). When the SCL pin is released high, it is held that way for TBRG. The data on the SDA pin must remain stable for that duration and some hold time after the next falling edge of SCL. After the eighth bit is shifted out (the falling edge of the eighth clock), the BF flag is cleared and the master releases SDA. This allows the slave device being addressed to respond with an $\overline{ACK}$ bit during the ninth bit time, if an address match occurred or if data was received properly. The status of $\overline{ACK}$ is written into the ACKDT bit on the falling edge of the ninth clock. If the master receives an Acknowledge, the Acknowledge Status bit, ACKSTAT, is cleared. If not, the bit is set. After the ninth clock, the SSPIF bit is set and the master clock (Baud Rate Generator) is suspended until the next data byte is loaded into the SSPBUF, leaving SCL low and SDA unchanged (Figure 9-21).

After the write to the SSPBUF, each bit of address will be shifted out on the falling edge of SCL, until all seven address bits and the R/$\overline{W}$ bit are completed. On the falling edge of the eighth clock, the master will deassert the SDA pin, allowing the slave to respond with an Acknowledge. On the falling edge of the ninth clock, the master will sample the SDA pin to see if the address was recognized by a slave. The status of the ACK bit is loaded into the ACKSTAT status bit (SSPCON2<6>). Following the falling edge of the ninth clock transmission of the address, the SSPIF is set, the BF flag is cleared and the Baud Rate Generator is turned off until another write to the SSPBUF takes place, holding SCL low and allowing SDA to float.

9.4.10.1 BF Status Flag

In Transmit mode, the BF bit (SSPSTAT<0>) is set when the CPU writes to SSPBUF and is cleared when all eight bits are shifted out.

9.4.10.2 WCOL Status Flag

If the user writes the SSPBUF when a transmit is already in progress (i.e., SSPSR is still shifting out a data byte), the WCOL is set and the contents of the buffer are unchanged (the write doesn't occur).

WCOL must be cleared in software.

9.4.10.3 ACKSTAT Status Flag

In Transmit mode, the ACKSTAT bit (SSPCON2<6>) is cleared when the slave has sent an Acknowledge ($\overline{ACK}$ = 0) and is set when the slave does Not Acknowledge ($\overline{ACK}$ = 1). A slave sends an Acknowledge when it has recognized its address (including a general call) or when the slave has properly received its data.

9.4.11 I²C MASTER MODE RECEPTION

Master mode reception is enabled by programming the Receive Enable bit, RCEN (SSPCON2<3>).

Note:	The MSSP module must be in an Idle state before the RCEN bit is set or the RCEN bit will be disregarded.

The Baud Rate Generator begins counting and on each rollover, the state of the SCL pin changes (high to low/low to high) and data is shifted into the SSPSR. After the falling edge of the eighth clock, the receive enable flag is automatically cleared, the contents of the SSPSR are loaded into the SSPBUF, the BF flag bit is set, the SSPIF flag bit is set and the Baud Rate Generator is suspended from counting, holding SCL low. The MSSP is now in Idle state, awaiting the next command. When the buffer is read by the CPU, the BF flag bit is automatically cleared. The user can then send an Acknowledge bit at the end of reception by setting the Acknowledge Sequence Enable bit, ACKEN (SSPCON2<4>).

9.4.11.1 BF Status Flag

In receive operation, the BF bit is set when an address or data byte is loaded into SSPBUF from SSPSR. It is cleared when the SSPBUF register is read.

9.4.11.2 SSPOV Status Flag

In receive operation, the SSPOV bit is set when 8 bits are received into the SSPSR and the BF flag bit is already set from a previous reception.

9.4.11.3 WCOL Status Flag

If the user writes the SSPBUF when a receive is already in progress (i.e., SSPSR is still shifting in a data byte), the WCOL bit is set and the contents of the buffer are unchanged (the write doesn't occur).

FIGURE 9-21: I²C MASTER MODE WAVEFORM (TRANSMISSION, 7 OR 10-BIT ADDRESS)

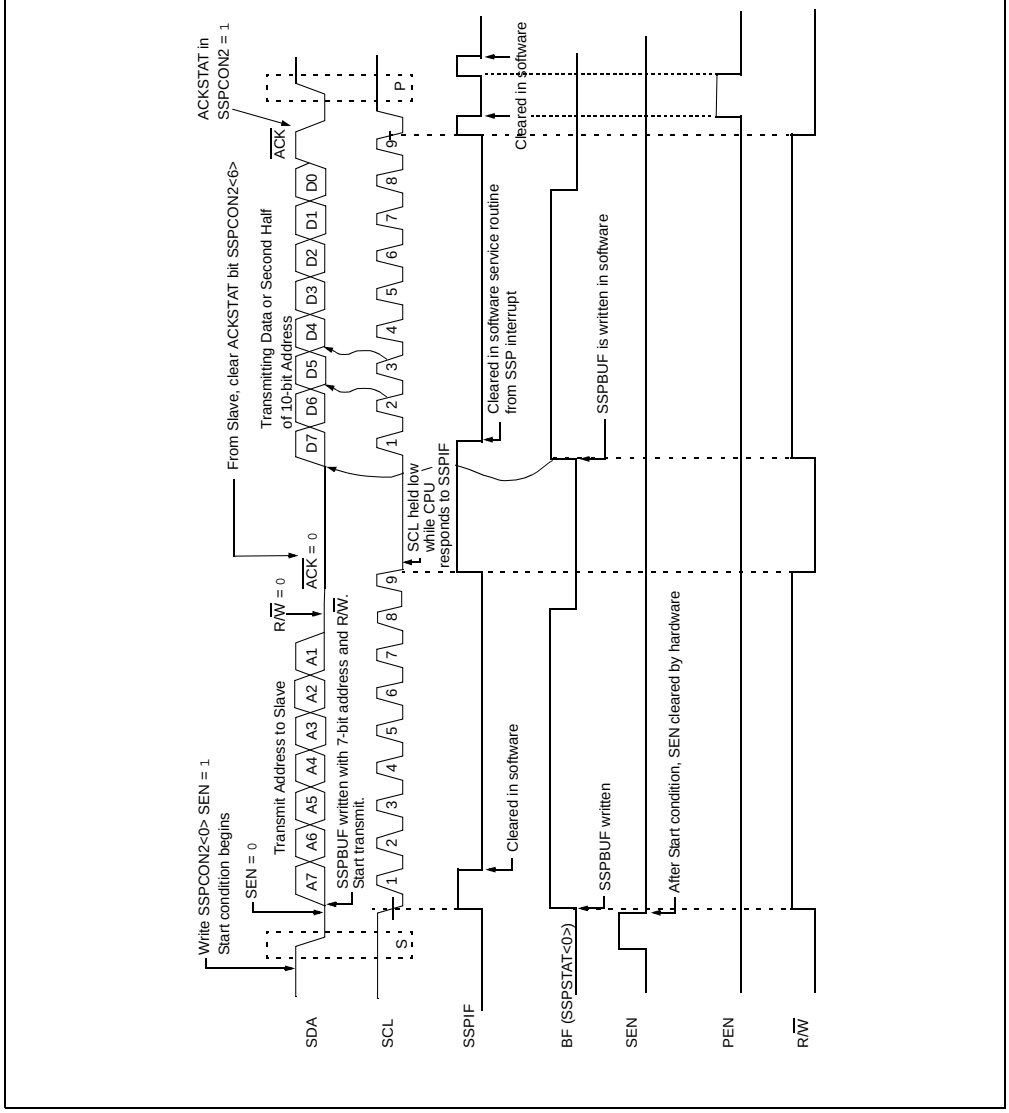
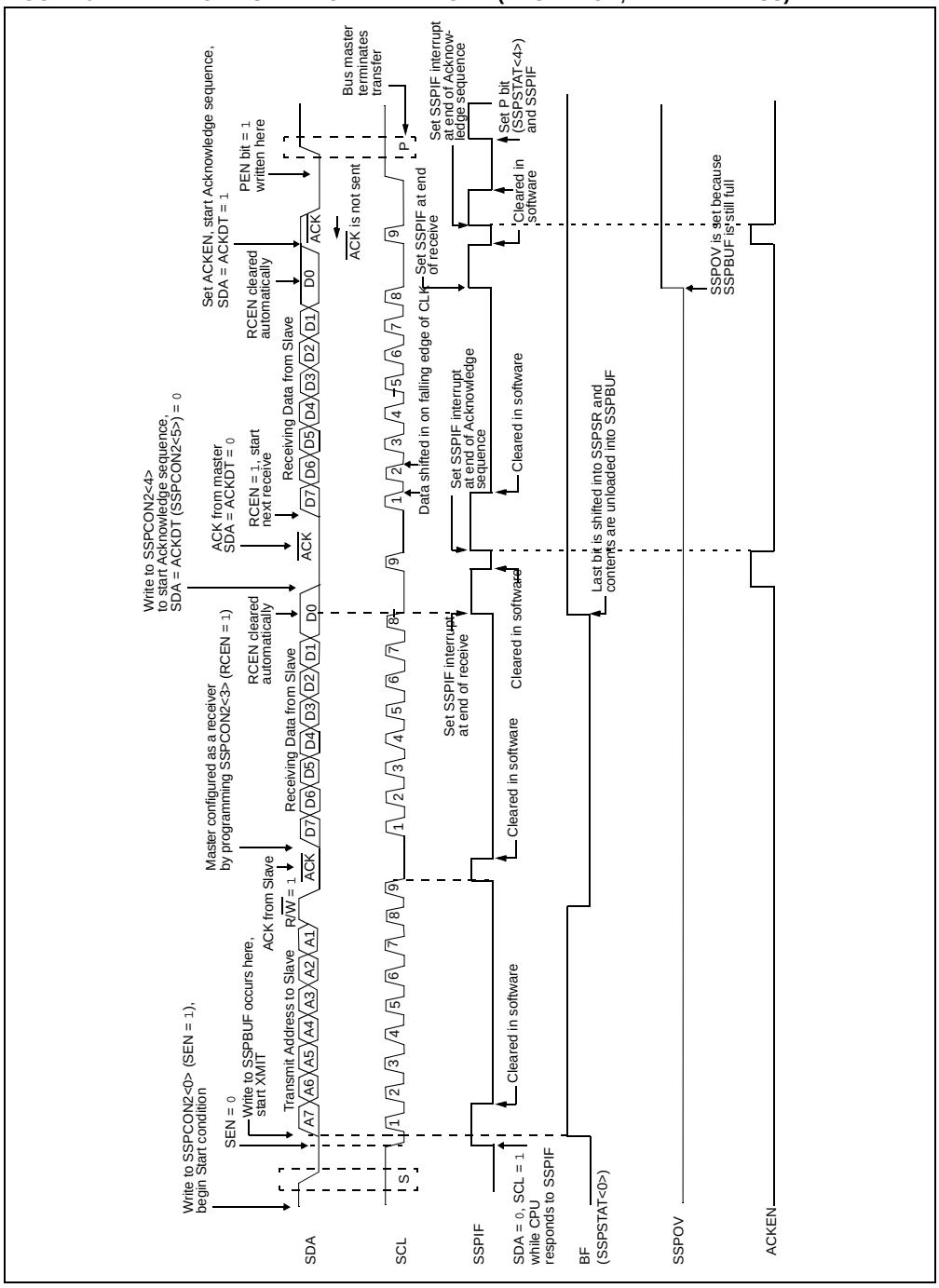


FIGURE 9-22: I²C MASTER MODE WAVEFORM (RECEPTION, 7-BIT ADDRESS)



9.4.12 ACKNOWLEDGE SEQUENCE TIMING

An Acknowledge sequence is enabled by setting the Acknowledge Sequence Enable bit, ACKEN (SSPCON2<4>). When this bit is set, the SCL pin is pulled low and the contents of the Acknowledge data bit are presented on the SDA pin. If the user wishes to generate an Acknowledge, then the ACKDT bit should be cleared. If not, the user should set the ACKDT bit before starting an Acknowledge sequence. The Baud Rate Generator then counts for one rollover period (TBRG) and the SCL pin is deasserted (pulled high). When the SCL pin is sampled high (clock arbitration), the Baud Rate Generator counts for TBRG. The SCL pin is then pulled low. Following this, the ACKEN bit is automatically cleared, the baud rate generator is turned off and the MSSP module then goes into Idle mode (Figure 9-23).

9.4.12.1 WCOL Status Flag

If the user writes the SSPBUF when an Acknowledge sequence is in progress, then WCOL is set and the contents of the buffer are unchanged (the write doesn't occur).

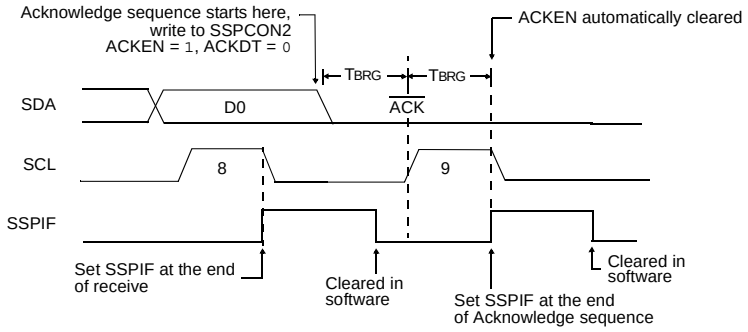
9.4.13 STOP CONDITION TIMING

A Stop bit is asserted on the SDA pin at the end of a receive/transmit by setting the Stop Sequence Enable bit, PEN (SSPCON2<2>). At the end of a receive/transmit, the SCL line is held low after the falling edge of the ninth clock. When the PEN bit is set, the master will assert the SDA line low. When the SDA line is sampled low, the Baud Rate Generator is reloaded and counts down to 0. When the Baud Rate Generator times out, the SCL pin will be brought high and one TBRG (Baud Rate Generator rollover count) later, the SDA pin will be deasserted. When the SDA pin is sampled high while SCL is high, the P bit (SSPSTAT<4>) is set. A TBRG later, the PEN bit is cleared and the SSPIF bit is set (Figure 9-24).

9.4.13.1 WCOL Status Flag

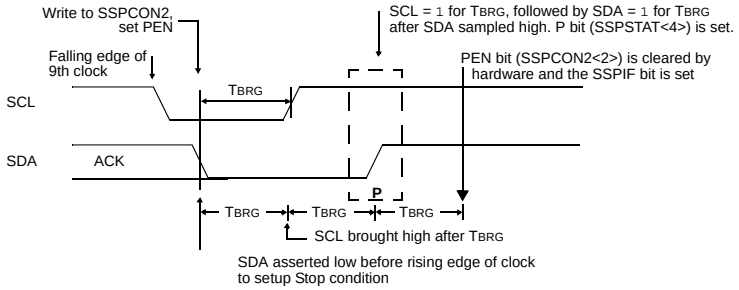
If the user writes the SSPBUF when a Stop sequence is in progress, then the WCOL bit is set and the contents of the buffer are unchanged (the write doesn't occur).

FIGURE 9-23: ACKNOWLEDGE SEQUENCE WAVEFORM



Note: TBRG = one Baud Rate Generator period.

FIGURE 9-24: STOP CONDITION RECEIVE OR TRANSMIT MODE



Note: TBRG = one Baud Rate Generator period.

10.0 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

10.1 USART Baud Rate Generator (BRG)

The BRG supports both the Asynchronous and Synchronous modes of the USART. It is a dedicated 8-bit baud rate generator. The SPBRG register controls the period of a free running 8-bit timer. In Asynchronous mode, bit BRGH (TXSTA<2>) also controls the baud rate. In Synchronous mode, bit BRGH is ignored. Table 10-1 shows the formula for computation of the baud rate for different USART modes which only apply in Master mode (internal clock).

Given the desired baud rate and FOSC, the nearest integer value for the SPBRG register can be calculated using the formula in Table 10-1. From this, the error in baud rate can be determined.

It may be advantageous to use the high baud rate (BRGH = 1) even for slower baud clocks. This is because the FOSC/(16 (X + 1)) equation can reduce the baud rate error in some cases.

Writing a new value to the SPBRG register causes the BRG timer to be reset (or cleared). This ensures the BRG does not wait for a timer overflow before outputting the new baud rate.

10.1.1 SAMPLING

The data on the RC7/RX/DT pin is sampled three times by a majority detect circuit to determine if a high or a low level is present at the RX pin.

TABLE 10-1: BAUD RATE FORMULA

SYNC	BRGH = 0 (Low Speed)	BRGH = 1 (High Speed)
0	(Asynchronous) Baud Rate = FOSC/(64 (X + 1))	Baud Rate = FOSC/(16 (X + 1))
1	(Synchronous) Baud Rate = FOSC/(4 (X + 1))	N/A

Legend: X = value in SPBRG (0 to 255)

TABLE 10-2: REGISTERS ASSOCIATED WITH BAUD RATE GENERATOR

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
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
18h	RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 000x	0000 000x
99h	SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

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

TABLE 10-3: BAUD RATES FOR ASYNCHRONOUS MODE (BRGH = 0)

BAUD RATE (K)	Fosc = 20 MHz			Fosc = 16 MHz			Fosc = 10 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	-	-	-	-	-	-	-	-	-
1.2	1.221	1.75	255	1.202	0.17	207	1.202	0.17	129
2.4	2.404	0.17	129	2.404	0.17	103	2.404	0.17	64
9.6	9.766	1.73	31	9.615	0.16	25	9.766	1.73	15
19.2	19.531	1.72	15	19.231	0.16	12	19.531	1.72	7
28.8	31.250	8.51	9	27.778	3.55	8	31.250	8.51	4
33.6	34.722	3.34	8	35.714	6.29	6	31.250	6.99	4
57.6	62.500	8.51	4	62.500	8.51	3	52.083	9.58	2
HIGH	1.221	-	255	0.977	-	255	0.610	-	255
LOW	312.500	-	0	250.000	-	0	156.250	-	0

BAUD RATE (K)	Fosc = 4 MHz			Fosc = 3.6864 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	0.300	0	207	0.3	0	191
1.2	1.202	0.17	51	1.2	0	47
2.4	2.404	0.17	25	2.4	0	23
9.6	8.929	6.99	6	9.6	0	5
19.2	20.833	8.51	2	19.2	0	2
28.8	31.250	8.51	1	28.8	0	1
33.6	-	-	-	-	-	-
57.6	62.500	8.51	0	57.6	0	0
HIGH	0.244	-	255	0.225	-	255
LOW	62.500	-	0	57.6	-	0

TABLE 10-4: BAUD RATES FOR ASYNCHRONOUS MODE (BRGH = 1)

BAUD RATE (K)	Fosc = 20 MHz			Fosc = 16 MHz			Fosc = 10 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	-	-	-	-	-	-	-	-	-
1.2	-	-	-	-	-	-	-	-	-
2.4	-	-	-	-	-	-	2.441	1.71	255
9.6	9.615	0.16	129	9.615	0.16	103	9.615	0.16	64
19.2	19.231	0.16	64	19.231	0.16	51	19.531	1.72	31
28.8	29.070	0.94	42	29.412	2.13	33	28.409	1.36	21
33.6	33.784	0.55	36	33.333	0.79	29	32.895	2.10	18
57.6	59.524	3.34	20	58.824	2.13	16	56.818	1.36	10
HIGH	4.883	-	255	3.906	-	255	2.441	-	255
LOW	1250.000	-	0	1000.000		0	625.000	-	0

BAUD RATE (K)	Fosc = 4 MHz			Fosc = 3.6864 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	-	-	-	-	-	-
1.2	1.202	0.17	207	1.2	0	191
2.4	2.404	0.17	103	2.4	0	95
9.6	9.615	0.16	25	9.6	0	23
19.2	19.231	0.16	12	19.2	0	11
28.8	27.798	3.55	8	28.8	0	7
33.6	35.714	6.29	6	32.9	2.04	6
57.6	62.500	8.51	3	57.6	0	3
HIGH	0.977	-	255	0.9	-	255
LOW	250.000	-	0	230.4	-	0

10.2 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

10.2.1 USART ASYNCHRONOUS TRANSMITTER

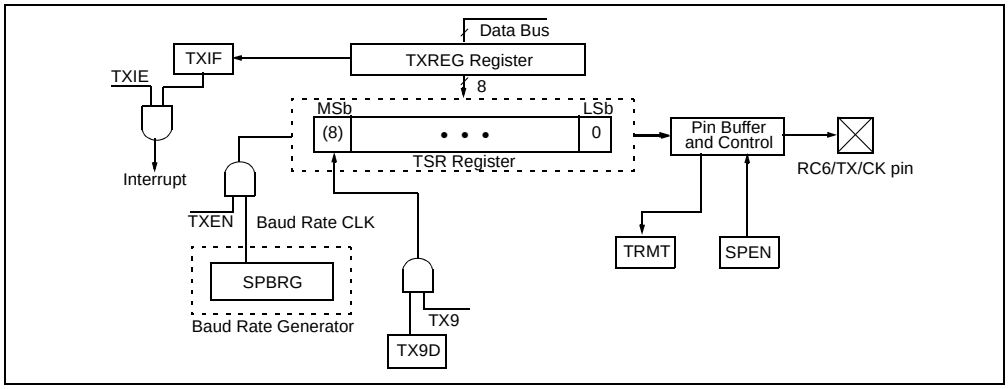
The USART transmitter block diagram is shown in Figure 10-1. The heart of the transmitter is the Transmit (Serial) Shift Register (TSR). The shift register obtains its data from the Read/Write Transmit Buffer, TXREG. The TXREG register is loaded with data in software. The TSR register is not loaded until 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.
2:	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 immediate 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 register 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



When setting up an Asynchronous Transmission, follow these steps:

1. Initialize the SPBRG register for the appropriate baud rate. If a high-speed baud rate is desired, set bit BRGH (**Section 10.1 "USART Baud Rate Generator (BRG)"**).
2. Enable the asynchronous serial port by clearing bit SYNC and setting bit SPEN.
3. If interrupts are desired, then set enable bit TXIE.
4. If 9-bit transmission is desired, then set transmit bit TX9.
5. Enable the transmission by setting bit TXEN, which will also set bit TXIF.
6. If 9-bit transmission is selected, the ninth bit should be loaded in bit TX9D.
7. Load data to the TXREG register (starts transmission).
8. If using interrupts, ensure that GIE and PEIE (bits 7 and 6) of the INTCON register are set.

FIGURE 10-2: ASYNCHRONOUS MASTER TRANSMISSION

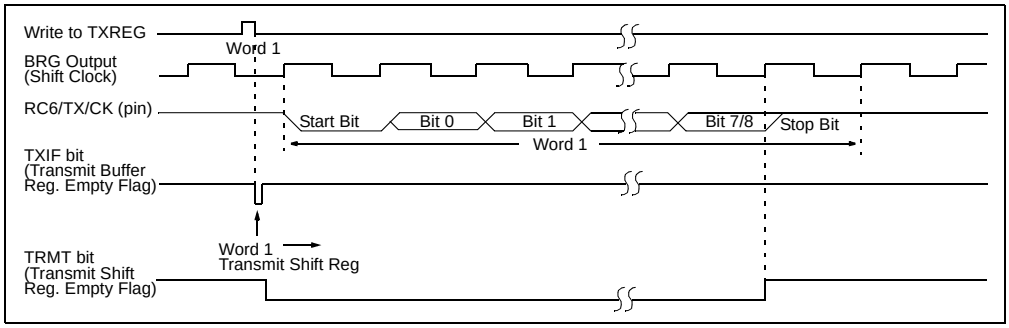
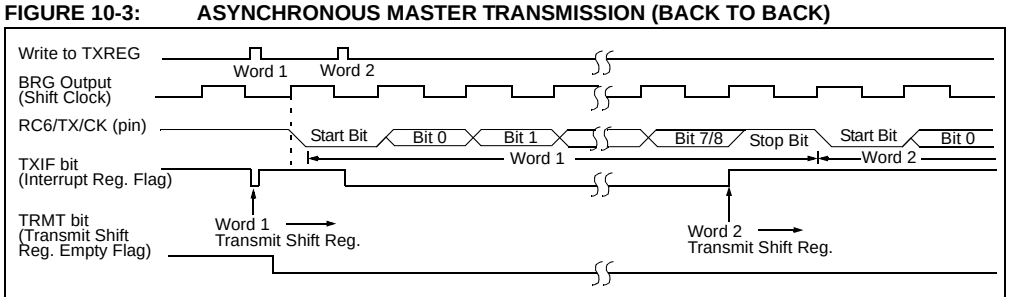


FIGURE 10-3: ASYNCHRONOUS MASTER TRANSMISSION (BACK TO BACK)



Note: This timing diagram shows two consecutive transmissions.

TABLE 10-5: REGISTERS ASSOCIATED WITH ASYNCHRONOUS TRANSMISSION

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	R0IF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
18h	RCSTA	SPEN	RX9	SREN	CREN	—	FERR	OERR	RX9D	0000 -00x	0000 -00x
19h	TXREG	USART Transmit Register								0000 0000	0000 0000
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
99h	SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

Legend: x = unknown, - = unimplemented locations read as '0'. Shaded cells are not used for asynchronous transmission.

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

Appendix B

Source Code - PIC 16f877a

```
// Hexapod.C
#include<pic.h>
int Timer_count=0,i;
char  array[6];
char  array_value,k,angle;
//char j;
void timer0_int(void);
void usart_init(void);
int convert_angle(char angle);
void servo_init(void);
void servo_246_init(void);
void servo_135_init(void);
void angle_init(void);

void forward();
void Backward();
void Left(void);
void Right(void);
void Clockwise();
void Anti_Clockwise();
void start(void);
void Stop();
void display(char );
 void MSdelay(unsigned int val);
char RECDATA;
int adcvalue=0;
int Temp_value=0;
int Humd_value=0;
int LDR_value=0;
int IR_value=0;
int mems_value=0;
int obstacle_threshold=300;
int  ADC_read(char channel );
void obstacles_avoidance(void);
```

```
/***************OUTER SERVOS***************/
#define servo_motor1   RD2
#define servo_motor2   RD3
#define servo_motor3   RD4
#define servo_motor4   RD5
#define servo_motor5   RD6
#define servo_motor6   RB7
/***************INNER SERVOS***************/
#define servo_motor11    RB2
#define servo_motor22   RB3
#define servo_motor33   RB4
#define servo_motor44   RB5
#define servo_motor55   RB6
#define servo_motor66   RB1

#define Head_servo_motor   RD7;
char Head_servo_motor_value;

char servo_motor1_value;
char servo_motor2_value;
char servo_motor3_value;
char servo_motor4_value;
char servo_motor5_value;
char servo_motor6_value;

char servo_motor11_value;
char servo_motor22_value;
char servo_motor33_value;
char servo_motor44_value;
char servo_motor55_value;
char servo_motor66_value;

char servo_motor1_angle;
char servo_motor2_angle;
char servo_motor3_angle;
char servo_motor4_angle;
char servo_motor5_angle;
char servo_motor6_angle;

char servo_motor11_angle;
char servo_motor22_angle;
char servo_motor33_angle;
char servo_motor44_angle;
char servo_motor55_angle;
char servo_motor66_angle;
char Head_servo_motor_angle;
```

```
void main()
{
  TRISD = 0X00;

  TRISB = 0X00;
    TRISA = 0Xff;
  timer0_int();
  usart_init();
  angle_init();
  servo_init();
  ADCON1=0X82;
  MSdelay(1000);
  //start();
  //Clockwise();//
  //forward();
  //Backward();
  //Left();
while (1)
{
if(RECDATA=='F'){forward();obstacles_avoidance();}
if(RECDATA=='B'){Backward();}
if(RECDATA=='L'){Left();}
if(RECDATA=='R'){Right();}

if(RECDATA=='C'){Clockwise();}
if(RECDATA=='A'){Anti_Clockwise();}

if(RECDATA=='S'){Stop();}


  MSdelay(100);
   Temp_value=ADC_read( 0X81);
   Temp_value=(int)(Temp_value*.4);
   LDR_value=ADC_read( 0X89);
   Humd_value=ADC_read( 0X91);
   Humd_value=(int)(Humd_value/6.67);

  mems_value=ADC_read( 0XA1);

% TXREG='#';while(TRMT!=1);
  TXREG='$';while(TRMT!=1);
  display(Temp_value);

  TXREG='$';while(TRMT!=1);
  display(LDR_value);
```

```
  TXREG='$';while(TRMT!=1);
   display(Humd_value);
  TXREG='$';while(TRMT!=1);
   display(mems_value);
  TXREG='$';while(TRMT!=1);
  TXREG='*';while(TRMT!=1);
   MSdelay(1000);
}
}
void interrupt ad()
{
if(T0IF==1)
{
T0IF=0;
Timer_count++;
if(Timer_count<125){
TMR0  = 213;            //Timer value for 20micro second
}
if(Timer_count==servo_motor1_value)
  servo_motor1=0;
if(Timer_count==servo_motor2_value)
  servo_motor2=0;
if(Timer_count==servo_motor3_value)
  servo_motor3=0;
if(Timer_count==servo_motor4_value)
  servo_motor4=0;
if(Timer_count==servo_motor5_value)
  servo_motor5=0;
if(Timer_count==servo_motor6_value)
  servo_motor6=0;
if(Timer_count==servo_motor11_value)
  servo_motor11=0;
if(Timer_count==servo_motor22_value)
  servo_motor22=0;
if(Timer_count==servo_motor33_value)
  servo_motor33=0;
if(Timer_count==servo_motor44_value)
  servo_motor44=0;
if(Timer_count==servo_motor55_value)
  servo_motor55=0;
if(Timer_count==servo_motor66_value)
  servo_motor66=0;
if(Timer_count==Head_servo_motor_value)
//   Head_servo_motor=0;

if(Timer_count>=125)
```

```
{
  PS2=1;
  PS1=1;
  PS0=0;
  TMR0=122;
}
if( Timer_count == 130 )
  {
  PORTD=0XFF;
  PORTB=0XFF;
  PS2=0;
  PS1=0;
  PS0=0;
      TMR0=213;
      Timer_count = 0;
  }
}
if(RCIF==1)
{
RCIF=0;
RECDATA=RCREG;


}
}
void timer0_int()
{
  PSA    = 0;
  PS2    = 0;
  PS1      = 0;
  PS0    = 0;
  GIE    = 1;
  PEIE   = 1;
  T0IE   = 1;
  TMR0   = 215;
  T0CS   = 0;
}
void usart_init()
{
  TXEN   =1;
  BRGH   =1;
  SPBRG  =10;
  SYNC   =0;
  SPEN   =1;
  CREN   =1;
  GIE    =1;
```

```
  PEIE   =1;
  RCIE   =1;
  TRISC  =0X80;
}
void display(char angle )
{
  char   array_value1;
  array_value1=angle;
  for(i=1;i<=3;i++)
  {
  array[i]=array_value1%10;
  array_value1=array_value1/10;
  }
  for(i=3;i>=1;i--)
  {
  TXREG=array[i]+'0';
  while(TRMT!=1);
  }
  TXREG=0x0D;while(TRMT!=1);
  TXREG=0x0A;while(TRMT!=1);
}
void servo_init()
{

   servo_motor1_angle=29;
   servo_motor2_angle=28;
   servo_motor3_angle=25;
   servo_motor4_angle=28;
   servo_motor5_angle=26;
   servo_motor6_angle=30;
   servo_motor11_angle=26;
   servo_motor22_angle=18;
   servo_motor33_angle=25;
   servo_motor44_angle=26;
   servo_motor55_angle=20;
   servo_motor66_angle=30;


   servo_motor1_value = convert_angle( servo_motor1_angle
   );
   servo_motor2_value = convert_angle( servo_motor2_angle
   );
   servo_motor3_value = convert_angle( servo_motor3_angle
   );
   servo_motor4_value = convert_angle( servo_motor4_angle
   );
```

```
    servo_motor5_value = convert_angle( servo_motor5_angle
    );
    servo_motor6_value = convert_angle( servo_motor6_angle
    );

    servo_motor11_value = convert_angle (
    servo_motor11_angle);
    servo_motor22_value = convert_angle(
    servo_motor22_angle);
    servo_motor33_value = convert_angle(
    servo_motor33_angle);
    servo_motor44_value = convert_angle(
    servo_motor44_angle);
    servo_motor55_value = convert_angle(
    servo_motor55_angle);
    servo_motor66_value = convert_angle(
    servo_motor66_angle);
}
void angle_init()
{
    servo_motor1_angle=29;
    servo_motor2_angle=28;
    servo_motor3_angle=26;
    servo_motor4_angle=28;
    servo_motor5_angle=27;
    servo_motor6_angle=30;
    servo_motor11_angle=26;
    servo_motor22_angle=18;
    servo_motor33_angle=25;
    servo_motor44_angle=26;
    servo_motor55_angle=20;
    servo_motor66_angle=29;

}

void servo_135_init()
{
//    servo_motor1_angle=29;
//    servo_motor3_angle=25;
//    servo_motor5_angle=27;
//    servo_motor11_angle=26;
//    servo_motor33_angle=25;
//    servo_motor55_angle=20;

    servo_motor1_angle=29;
    servo_motor3_angle=25;
```

```
    servo_motor5_angle=27;
    servo_motor11_angle=26;
    servo_motor33_angle=25;
    servo_motor55_angle=20;

    servo_motor1_value = convert_angle( servo_motor1_angle
    );
    servo_motor3_value = convert_angle( servo_motor3_angle
    );
    servo_motor5_value = convert_angle( servo_motor5_angle
    );
    servo_motor11_value = convert_angle (
    servo_motor11_angle);
    servo_motor33_value = convert_angle(
    servo_motor33_angle);
    servo_motor55_value = convert_angle(
    servo_motor55_angle);
}
void servo_246_init()
{
//    servo_motor2_angle=28;
//    servo_motor4_angle=28;
//    servo_motor6_angle=30;
//    servo_motor22_angle=18;
//    servo_motor44_angle=26;
//    servo_motor66_angle=30;


    servo_motor2_angle=28;
    servo_motor4_angle=28;
    servo_motor6_angle=30;
    servo_motor22_angle=18;
    servo_motor44_angle=26;
    servo_motor66_angle=29;
    servo_motor2_value = convert_angle( servo_motor2_angle
    );
    servo_motor4_value = convert_angle( servo_motor4_angle
    );
    servo_motor6_value = convert_angle( servo_motor6_angle
    );
    servo_motor22_value = convert_angle(
    servo_motor22_angle);
    servo_motor44_value = convert_angle(
    servo_motor44_angle);
    servo_motor66_value = convert_angle(
    servo_motor66_angle);
```

```
}
/***************/
/*
                          Forward movemenht of Robort
*/
/***************/
void forward()
{

for (k=0;k<15;k++)
{
  servo_motor1_value=convert_angle(servo_motor1_angle++);
  servo_motor3_value=convert_angle(servo_motor3_angle++);
  servo_motor5_value=convert_angle(servo_motor5_angle++);
  MSdelay(35);
}

for (k=0;k<15;k++)
{

  servo_motor11_value = convert_angle(servo_motor11_angle
   ++);
  servo_motor33_value=convert_angle(servo_motor33_angle
   --);
  servo_motor55_value=convert_angle(servo_motor55_angle
   --);
  MSdelay(35);
}

for (k=0;k<15;k++)
{

  servo_motor1_value=convert_angle(servo_motor1_angle--);
  servo_motor3_value=convert_angle(servo_motor3_angle--);
  servo_motor5_value=convert_angle(servo_motor5_angle--);
  MSdelay(35);
}

///***************/
for (k=0;k<15;k++)
{

  servo_motor2_value=convert_angle(servo_motor2_angle++);
  servo_motor4_value=convert_angle(servo_motor4_angle++);
  servo_motor6_value=convert_angle(servo_motor6_angle++);
  MSdelay(35);
```

```
}

for (k=0;k<15;k++)
{
  MSdelay(35);
  servo_motor11_value=convert_angle(servo_motor11_angle
    --);
  servo_motor33_value=convert_angle(servo_motor33_angle
    ++);
  servo_motor55_value=convert_angle(servo_motor55_angle
    ++);
}

servo_135_init();


for (k=0;k<15;k++)
{
  MSdelay(35);
  servo_motor22_value=convert_angle(servo_motor22_angle
    ++);
  servo_motor44_value=convert_angle(servo_motor44_angle
    --);
  servo_motor66_value=convert_angle(servo_motor66_angle
    ++);
}
for (k=0;k<15;k++)
{

  servo_motor2_value=convert_angle(servo_motor2_angle--);
  servo_motor4_value=convert_angle(servo_motor4_angle--);
  servo_motor6_value=convert_angle(servo_motor6_angle--);
  MSdelay(35);
}


for (k=0;k<15;k++)
{
  MSdelay(35);
  servo_motor1_value=convert_angle(servo_motor1_angle++);
  servo_motor3_value=convert_angle(servo_motor3_angle++);
  servo_motor5_value=convert_angle(servo_motor5_angle++);
}
for (k=0;k<15;k++)
{
  MSdelay(35);
```

```
  servo_motor22_value=convert_angle(servo_motor22_angle
    --);
  servo_motor44_value=convert_angle(servo_motor44_angle
    ++);
  servo_motor66_value=convert_angle(servo_motor66_angle
    --);

}
servo_246_init();


}
/***************/
/*
                  Backward movemenht of Robort
*/
/***************/
void Backward()
{

//for (k=0;k<15;k++)
//{
//  servo_motor1_value=convert_angle(servo_motor1_angle
   ++);
//  servo_motor3_value=convert_angle(servo_motor3_angle
   ++);          /*        motot1,3,5 raise upward
   direction           */
//  servo_motor5_value=convert_angle(servo_motor5_angle
   ++);
//MSdelay(35);
//}
for (k=0;k<15;k++)
{

  servo_motor11_value = convert_angle(servo_motor11_angle
    --);
  servo_motor33_value=convert_angle(servo_motor33_angle
    --);
  servo_motor55_value=convert_angle(servo_motor55_angle
    ++);
MSdelay(35);
}

for (k=0;k<15;k++)
{
```

```
  servo_motor1_value=convert_angle(servo_motor1_angle--);
  servo_motor3_value=convert_angle(servo_motor3_angle--);
  servo_motor5_value=convert_angle(servo_motor5_angle--);
MSdelay(35);
}

/***************/
for (k=0;k<15;k++)
{

  servo_motor2_value=convert_angle(servo_motor2_angle++);
  servo_motor4_value=convert_angle(servo_motor4_angle++);
  servo_motor6_value=convert_angle(servo_motor6_angle++);
MSdelay(35);
}
//
for (k=0;k<15;k++)
{
  MSdelay(35);
  servo_motor11_value=convert_angle(servo_motor11_angle
   ++);
  servo_motor33_value=convert_angle(servo_motor33_angle
   ++);
  servo_motor55_value=convert_angle(servo_motor55_angle
   --);
}

servo_135_init();


for (k=0;k<15;k++)
{
  MSdelay(35);
  servo_motor22_value=convert_angle(servo_motor22_angle
   --);
  servo_motor44_value=convert_angle(servo_motor44_angle
   ++);
  servo_motor66_value=convert_angle(servo_motor66_angle
   ++);
}
for (k=0;k<15;k++)
{

  servo_motor2_value=convert_angle(servo_motor2_angle--);
  servo_motor4_value=convert_angle(servo_motor4_angle--);
  servo_motor6_value=convert_angle(servo_motor6_angle--);
```

```
MSdelay(35);
}


for (k=0;k<15;k++)
{
  MSdelay(35);
  servo_motor1_value=convert_angle(servo_motor1_angle++);
  servo_motor3_value=convert_angle(servo_motor3_angle++);
  servo_motor5_value=convert_angle(servo_motor5_angle++);
}
for (k=0;k<15;k++)
{
  MSdelay(35);
  servo_motor22_value=convert_angle(servo_motor22_angle
   ++);
  servo_motor44_value=convert_angle(servo_motor44_angle
   --);
  servo_motor66_value=convert_angle(servo_motor66_angle
   --);
}
servo_246_init();
}

/***************/
/*             left movements                          */
/***************/
void Left()
{
/*for (k=0;k<15;k++)
{
  servo_motor1_value=convert_angle(servo_motor1_angle++);
  servo_motor3_value=convert_angle(servo_motor3_angle++);
  servo_motor5_value=convert_angle(servo_motor5_angle++);
  MSdelay(35);
}*/
//start();
for (k=0;k<15;k++)
{

  servo_motor11_value = convert_angle(servo_motor11_angle
   --);
  servo_motor33_value=convert_angle(servo_motor33_angle
   --);
  servo_motor55_value=convert_angle(servo_motor55_angle
   ++);
```

```
  MSdelay(35);
}


for (k=0;k<15;k++)
{

  servo_motor1_value=convert_angle(servo_motor1_angle--);
  servo_motor3_value=convert_angle(servo_motor3_angle--);
  servo_motor5_value=convert_angle(servo_motor5_angle--);
  MSdelay(35);
}

///***************/
for (k=0;k<15;k++)
{

  servo_motor2_value=convert_angle(servo_motor2_angle++);
  servo_motor4_value=convert_angle(servo_motor4_angle++);
  servo_motor6_value=convert_angle(servo_motor6_angle++);
  MSdelay(35);
}

for (k=0;k<15;k++)
{

  servo_motor11_value = convert_angle(servo_motor11_angle
   ++);
  servo_motor33_value=convert_angle(servo_motor33_angle
   ++);
  servo_motor55_value=convert_angle(servo_motor55_angle
   --);
  MSdelay(35);
}
servo_135_init();


for (k=0;k<15;k++)
{
  MSdelay(35);
  servo_motor22_value=convert_angle(servo_motor22_angle
   --);
  servo_motor44_value=convert_angle(servo_motor44_angle
   ++);
  servo_motor66_value=convert_angle(servo_motor66_angle
   ++);
```

```
}
for (k=0;k<15;k++)
{

  servo_motor2_value=convert_angle(servo_motor2_angle--);
  servo_motor4_value=convert_angle(servo_motor4_angle--);
  servo_motor6_value=convert_angle(servo_motor6_angle--);
MSdelay(35);
}


for (k=0;k<15;k++)
{
  MSdelay(35);
  servo_motor1_value=convert_angle(servo_motor1_angle++);
  servo_motor3_value=convert_angle(servo_motor3_angle++);
  servo_motor5_value=convert_angle(servo_motor5_angle++);
}
for (k=0;k<15;k++)
{
  MSdelay(35);
  servo_motor22_value=convert_angle(servo_motor22_angle
   ++);
  servo_motor44_value=convert_angle(servo_motor44_angle
   --);
  servo_motor66_value=convert_angle(servo_motor66_angle
   --);

}
servo_246_init();
}

void Right()
{
/*for (k=0;k<15;k++)
{
  servo_motor1_value=convert_angle(servo_motor1_angle++);
  servo_motor3_value=convert_angle(servo_motor3_angle++);
  servo_motor5_value=convert_angle(servo_motor5_angle++);
  MSdelay(35);
}*/
//start();
for (k=0;k<15;k++)
{
```

```
  servo_motor11_value = convert_angle(servo_motor11_angle
   ++);
  servo_motor33_value=convert_angle(servo_motor33_angle
   ++);
  servo_motor55_value=convert_angle(servo_motor55_angle
   --);
  MSdelay(35);
}


for (k=0;k<15;k++)
{

  servo_motor1_value=convert_angle(servo_motor1_angle--);
  servo_motor3_value=convert_angle(servo_motor3_angle--);
  servo_motor5_value=convert_angle(servo_motor5_angle--);
  MSdelay(35);
}

///***************/
for (k=0;k<15;k++)
{

  servo_motor2_value=convert_angle(servo_motor2_angle++);
  servo_motor4_value=convert_angle(servo_motor4_angle++);
  servo_motor6_value=convert_angle(servo_motor6_angle++);
  MSdelay(35);
}

for (k=0;k<15;k++)
{

  servo_motor11_value = convert_angle(servo_motor11_angle
   --);
  servo_motor33_value=convert_angle(servo_motor33_angle
   --);
  servo_motor55_value=convert_angle(servo_motor55_angle
   ++);
  MSdelay(35);
}
servo_135_init();


for (k=0;k<15;k++)
{
  MSdelay(35);
```

```
  servo_motor22_value=convert_angle(servo_motor22_angle
    ++);
  servo_motor44_value=convert_angle(servo_motor44_angle
    --);
  servo_motor66_value=convert_angle(servo_motor66_angle
    --);
}
for (k=0;k<15;k++)
{

  servo_motor2_value=convert_angle(servo_motor2_angle--);
  servo_motor4_value=convert_angle(servo_motor4_angle--);
  servo_motor6_value=convert_angle(servo_motor6_angle--);
MSdelay(35);
}


for (k=0;k<15;k++)
{
  MSdelay(35);
  servo_motor1_value=convert_angle(servo_motor1_angle++);
  servo_motor3_value=convert_angle(servo_motor3_angle++);
  servo_motor5_value=convert_angle(servo_motor5_angle++);
}
for (k=0;k<15;k++)
{
  MSdelay(35);
  servo_motor22_value=convert_angle(servo_motor22_angle
    --);
  servo_motor44_value=convert_angle(servo_motor44_angle
    ++);
  servo_motor66_value=convert_angle(servo_motor66_angle
    ++);

}
servo_246_init();
}



void Clockwise()
{
for (k=0;k<15;k++)
{
  servo_motor1_value=convert_angle(servo_motor1_angle++);
```

```
  servo_motor3_value=convert_angle(servo_motor3_angle++);
              /*        motot1,3,5 raise upward direction
             */
  servo_motor5_value=convert_angle(servo_motor5_angle++);
  servo_motor11_value = convert_angle(servo_motor11_angle
   ++);
  servo_motor33_value=convert_angle(servo_motor33_angle
   ++);
  servo_motor55_value=convert_angle(servo_motor55_angle
   ++);
  MSdelay(35);
}
for (k=0;k<15;k++)
{
  servo_motor1_value=convert_angle(servo_motor1_angle--);
  servo_motor3_value=convert_angle(servo_motor3_angle--);
              /*        motot1,3,5 raise upward direction
             */
  servo_motor5_value=convert_angle(servo_motor5_angle--);
//  servo_motor22_value=convert_angle(servo_motor22_angle
   --);
//  servo_motor44_value=convert_angle(servo_motor44_angle
   --);
//  servo_motor66_value=convert_angle(servo_motor66_angle
   --);
MSdelay(35);
}
servo_135_init();
for (k=0;k<15;k++)
{
  servo_motor2_value=convert_angle(servo_motor2_angle++);
  servo_motor4_value=convert_angle(servo_motor4_angle++);
  servo_motor6_value=convert_angle(servo_motor6_angle++);
  servo_motor22_value=convert_angle(servo_motor22_angle
   ++);
  servo_motor44_value=convert_angle(servo_motor44_angle
   ++);
  servo_motor66_value=convert_angle(servo_motor66_angle
   ++);
  MSdelay(35);
}
for (k=0;k<15;k++)
{
  servo_motor2_value=convert_angle(servo_motor2_angle--);
  servo_motor4_value=convert_angle(servo_motor4_angle--);
  servo_motor6_value=convert_angle(servo_motor6_angle--);
```

```
//  servo_motor11_value = convert_angle(
    servo_motor11_angle--);
//  servo_motor33_value=convert_angle(servo_motor33_angle
    --);
//  servo_motor55_value=convert_angle(servo_motor55_angle
    --);
  MSdelay(35);
}
servo_246_init();
}

void Anti_Clockwise()
{
for (k=0;k<15;k++)
{
  servo_motor1_value=convert_angle(servo_motor1_angle++);
  servo_motor3_value=convert_angle(servo_motor3_angle++);
            /*        motot1,3,5 raise upward direction
           */
  servo_motor5_value=convert_angle(servo_motor5_angle++);
  servo_motor11_value = convert_angle(servo_motor11_angle
    --);
  servo_motor33_value=convert_angle(servo_motor33_angle
    --);
  servo_motor55_value=convert_angle(servo_motor55_angle
    --);
  MSdelay(35);
}
for (k=0;k<15;k++)
{
  servo_motor1_value=convert_angle(servo_motor1_angle--);
  servo_motor3_value=convert_angle(servo_motor3_angle--);
            /*        motot1,3,5 raise upward direction
           */
  servo_motor5_value=convert_angle(servo_motor5_angle--)
    ;;
  MSdelay(35);
}
servo_246_init();
for (k=0;k<15;k++)
{
  servo_motor2_value=convert_angle(servo_motor2_angle++);
  servo_motor4_value=convert_angle(servo_motor4_angle++);
  servo_motor6_value=convert_angle(servo_motor6_angle++);
  servo_motor22_value=convert_angle(servo_motor22_angle
    --);
```

```
  servo_motor44_value=convert_angle(servo_motor44_angle
   --);
  servo_motor66_value=convert_angle(servo_motor66_angle
   --);
  MSdelay(35);
}
for (k=0;k<15;k++)
{
  servo_motor2_value=convert_angle(servo_motor2_angle--);
  servo_motor4_value=convert_angle(servo_motor4_angle--);
  servo_motor6_value=convert_angle(servo_motor6_angle--);
  MSdelay(35);
}
servo_135_init();
}


void start(void)
{
for (k=0;k<15;k++)
{
  servo_motor1_value=convert_angle(servo_motor1_angle++);
  servo_motor3_value=convert_angle(servo_motor3_angle++);
            /*        motot1,3,5 raise upward direction
           */
  servo_motor5_value=convert_angle(servo_motor5_angle++);
  MSdelay(35);
}
}
void Stop()
{
  servo_135_init();
  servo_246_init();
}
int  convert_angle(char k)
{
  char timer_value;
  int  temp;
  temp=k*5;
  timer_value = temp/9;
  timer_value=timer_value+25;
  return(timer_value);
}
 void MSdelay(unsigned int val)
  {
  unsigned int del,del1;
```

```
  for(del=1;del<=val;del++)
  {
  for(del1=0;del1<=331;del1++);
  }
 }
int  ADC_read(char channel )
{
ADCON0=channel;
MSdelay(15);
ADGO=1;
while(ADGO==1);
adcvalue=ADRESH;
adcvalue=adcvalue<<8;
adcvalue=(adcvalue|ADRESL);
return(adcvalue);
}
void obstacles_avoidance(void)
{
unsigned int adc_threshold;
unsigned int peak_left_threshold;
unsigned int peak_right_threshold;
adc_threshold = ADC_read(0X99);
if( ( adc_threshold > obstacle_threshold ) )//|| (
   obstacle == 1 )
  {
          RE0=1;
          MSdelay(40);
          RE0=0;
      MSdelay(1000);
  peak_left_threshold = adc_threshold;
  peak_right_threshold = adc_threshold;

      for(k=0 ;k < 45;k++ )
        {
       Head_servo_motor_value=convert_angle(
    Head_servo_motor_angle--);
        MSdelay(20);
        adc_threshold = ADC_read(0X99);
        if( adc_threshold > peak_left_threshold +10)
          {

          peak_left_threshold = adc_threshold;
          }
        }
      for(k=0 ;k < 45;k++ )
        {
```

```
          Head_servo_motor_value=convert_angle(
    Head_servo_motor_angle++);
         MSdelay(20);
         }

       for(k=0 ;k < 45;k++ )
           {

           Head_servo_motor_value=convert_angle(
    Head_servo_motor_angle++);
           MSdelay(20);
           adc_threshold = ADC_read(0X99);
           if( adc_threshold > peak_right_threshold+10 )
             {
             peak_right_threshold = adc_threshold;
             }
           }
       for(k=0 ;k < 45;k++ )
           {
            Head_servo_motor_value=convert_angle(
    Head_servo_motor_angle--);
           MSdelay(20);
           }

         if( peak_left_threshold > peak_right_threshold )
           {
           Backward();Backward();
           Right();  Right();Right();

           }
         if( peak_right_threshold > peak_left_threshold )
           {
           Backward();Backward();
           Left();Left();Left();
           }
   peak_left_threshold = 0;
   peak_right_threshold = 0;
//adc_threshold = ADC_read(0X99);
     }
/*else if( adc_threshold < obstacle_threshold )
     {
     forward();
     } */}
```

Appendix C

Matlab Weed Detection

```
clc
clear all
RGB1=imread('20140219_165816.jpg');
RGB2 = imadjust(RGB1,[.2.3 0; .6.7 1],[]);
[x y z]=size(RGB2)
data=RGB2;
subplot(221);imshow(data)
diff_im = imsubtract(data(:,:,2), rgb2gray(data));
diff_im = medfilt2(diff_im, [3 3]);
diff_im = im2bw(diff_im,0.1);
b= bwareaopen(diff_im,166);
m=b(1:50,:);
n=b(50:100,:);
o=b(100:142,:);
f1=find(m==1);f2=find(n==1);f3=find(o==1);
subplot(222);imshow(m)
subplot(223);imshow(n)
subplot(224);imshow(o)
if length(f1)>100
sersent('Z')
pause(1)
sersent('Z')
sersent('F')
pause(5)
else
disp('Forward')
sersent('F')
pause(5)
end
if length(f2)>100
sersent('Z')
pause(1)
sersent('Z')
sersent('F')
pause(5)
```

```
else
disp('Forward')
sersent('F')
pause(5)
end
if length(f3)>100
sersent('Z')
pause(1)
sersent('Z')
sersent('F')
pause(5)
else
disp('Forward')
sersent('F')
pause(5)
end
sersent('S')
disp('Finished')
```