

# **AUTOMATED GUIDED ROBOT (AGR)**

By

**Mohd Azlan Shah Bin Abd Rahim**

## **FINAL PROJECT REPORT**

Submitted to the Electrical & Electronics Engineering Programme  
in Partial Fulfillment of the Requirements  
for the Degree  
Bachelor of Engineering (Hons)  
(Electrical & Electronics Engineering)

Universiti Teknologi Petronas  
Bandar Seri Iskandar  
31750 Tronoh  
Perak Darul Ridzuan

© Copyright 2007

by

Mohd Azlan Shah Abd Rahim, 2007

# **CERTIFICATION OF APPROVAL**

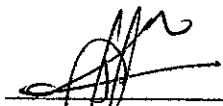
## **AUTOMATED GUIDED ROBOT (AGR)**

By

**Mohd Azlan Shah Bin Abd Rahim**

A project dissertation submitted to the  
Electrical & Electronics Engineering Programme  
Universiti Teknologi PETRONAS  
in partial fulfilment of the requirement for the  
Bachelor of Engineering (Hons)  
(Electrical & Electronics Engineering)

Approved:



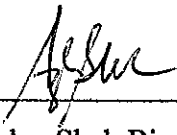
Ms. Nani Mohd Nawi  
Project Supervisor

UNIVERSITI TEKNOLOGI PETRONAS  
TRONOH, PERAK

December 2007

## **CERTIFICATION OF ORIGINALITY**

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.



---

Mohd Azlan Shah Bin Abd Rahim

## **ABSTRACT**

This project concerns the design and fabrication of the Automated Guided Robot (AGR) prototype, utilizing artificial intelligence (AI) and genetic algorithm (GA) as a mainframe in helping the robot to generate a self-understanding of the area of work and mobilization to a destination desired by the user. The main objective of this project is to create and develop a Path Planning Mobile Robot able to avoid obstacles in its path and reach a target designated position from its starting point utilizing 3 wheel-based rover body, sensors, linear motors and microcontrollers. Compared to manual mobile robots, AGRs require sensors and control systems that generate feedback for the re-evaluation of an unexpected situation and to detect obstacles in the path the AGR is required to follow. The paper describes the network algorithms developed and used in the design process of the AGR including simulations and circuit designs done for the prototype. A general robotics circuit construction of the mainframe target board for central processing, a controller board for the sensor feedbacks and a small base tri-wheeled structure has been fabricated by the author and continual troubleshooting and enhancement has been done for these components of the AGR. Algorithm conversion to C code programming has been done throughout the project for the obstacle avoidance and path planning algorithms based upon the GA platform of AI.

## **ACKNOWLEDGEMENTS**

Firstly, I would like to thank our Almighty God for giving me the strength and time to complete this Interim Report.

I would like to grant my humblest gratitude to my Supervisor, Ms. Illani Mohd Nawi for supervising my Final Year Project and giving me endless guidance and advice on the work that must be carried out in order to complete the project successfully. She has never failed to give me support and was always understanding despite all the errors and delays that I made during the completion of the project.

Apart from that, to the technicians Mrs. Siti Hawa, for the support and most importantly for the cooperation and help that had been given. Not forgetting credit to the Electrical and Electronics Faculty for their cooperation and contributions of individuals towards completing my final year project.

Also special thanks to my fellow colleagues who have aided me with valuable advises and opinions regarding the project. Last but not least, to my beloved parents and family for their continuous love and support.

Thank you and God Bless.

## **TABLE OF CONTENTS**

LIST OF TABLES .....	ix
LIST OF FIGURES.....	x
LIST OF ABBREVIATIONS .....	xi
CHAPTER 1 INTRODUCTION .....	1
1.1 Background of Study .....	1
1.2 Problem Statement.....	1
1.3 Objective.....	2
1.4 Scope of Study.....	2
CHAPTER 2 LITERATURE REVIEW .....	3
2.1 Methods of Detection and Sensor Systems as Feedback Analysis...	3
2.2 Differential drive and steering .....	4
2.2.1 Comparison Between Servo And Stepper Motor In Direction Control .....	5
Stepper Motors .....	5
Servo Motors.....	6
2.2.2 Pulse Width Modulation (PWM) Motor speed control.....	7
2.2.3 H-Bridge Motor Speed Controller Board.....	8
2.3 Path Planning and Obstacle Avoidance.....	10
2.3.1 3D arrays in the use of grid-based global positioning .....	10
2.3.2 Wisp628 Microcontroller Programmer Board .....	13
2.3.3 CMPS03 Electronic Magnetic Compass.....	14
CHAPTER 3 PROJECT WORK.....	17
3.1 Project Process Flow .....	17
3.2 Procedure Identification .....	19
3.3 Tools Required .....	20
3.3.1 Hardware Configuration .....	20
3.3.2 Software Requirements .....	20
CHAPTER 4 RESULTS & DISCUSSION.....	21
4.1 Body Structure and Mechanism .....	21
4.1.1 Generation of 3D design .....	21
4.1.2 Simulation of DD&GP using Simulink MATLAB .....	23

4.1.3 Actual fabrication of AGR prototype structure .....	27
4.2 Circuit Construction .....	28
4.2.1 Master and Slave Controller Target Board .....	28
4.2.2 Relay controlled motor driver .....	29
4.3 Algorithm Building and Coding Generation .....	31
4.3.1 Algorithm for differential drive movement control .....	31
4.3.2 Algorithm for obstacle avoidance .....	33
4.4 Troubleshooting and Testing .....	35
4.4.1 Mobility Test.....	35
4.4.2 Obstacle Avoidance and Path Planning Test .....	36
CHAPTER 5 CONCLUSION & RECOMMENDATIONS .....	39
REFERENCES .....	40
APPENDICES.....	41
Appendix A Microcontroller Datasheet .....	42
Appendix B MAX232 datasheet .....	70
Appendix C SRF04 Ultrasonic Range Sensor Datasheet .....	75
Appendix D CMPS03 Electronic Magnetic Compass Datasheet.....	77
Appendix E C Code Generated .....	80

**LIST OF TABLES**

**Table 1** Truth table for I/O of motor driver ..... 10

**Table 2** I2C interface 16bit registers ..... 16



## LIST OF FIGURES

<b>Figure 1</b> Differential drive robot with servo steering build.....	4
<b>Figure 2</b> Sample PWM waveforms for stated duty-cycles.....	8
<b>Figure 3</b> Relay controlled motor driver.....	9
<b>Figure 4</b> Node-wise 2D positioning .....	11
<b>Figure 5</b> 3D axis .....	11
<b>Figure 6</b> An example of a discrete probability distribution of vertex (sensor node) k for direction (action) "East"(i.e. right).....	12
<b>Figure 7</b> Wisp628 Schematic Diagram.....	14
<b>Figure 8</b> Connections to the compass module.....	15
<b>Figure 9</b> Process flow of project.....	18
<b>Figure 10</b> Procedure identification and flow diagram.....	19
<b>Figure 11</b> Top view of AGR base 2D sketch .....	22
<b>Figure 12</b> Full solidworks 3D model of AGR.....	23
<b>Figure 13</b> Full Differential drive and global positioning blockset using Simulink....	24
<b>Figure 14</b> The Main PID controller blockset breakdown with error constants .....	25
<b>Figure 15</b> The differential drive blockset breakdown for DC motor.....	25
<b>Figure 16</b> Results of simulation displayed as a figure.....	26
<b>Figure 17</b> Results for PID controller orientation.....	26
<b>Figure 18</b> Fabricated AGR isometric view.....	27
<b>Figure 19</b> Fabricated AGR side view .....	27
<b>Figure 20</b> PIC16F84A + PIC16F877A target board .....	28
<b>Figure 21</b> EAGLE schematic for AGR Target Board .....	29
<b>Figure 22</b> EAGLE Schematic for Motor driver.....	30
<b>Figure 23</b> Constructed circuit on veroboard.....	30
<b>Figure 24</b> Algorithm for Differential drive control .....	32
<b>Figure 25</b> Ultrasonic sensor placement .....	33
<b>Figure 26</b> Algorithm bubble for obstacle avoidance .....	34
<b>Figure 27</b> Mobility test grid.....	35
<b>Figure 28</b> possible path taken by the AGR in an undetermined space.....	37
<b>Figure 29</b> possible path taken by the AGR in a predetermined space.....	38

## **LIST OF ABBREVIATIONS**

<b>2D</b>	<b>Two-Dimensional</b>
<b>3D</b>	<b>Three-Dimensional</b>
<b>AGR</b>	<b>Automated Guided Robot</b>
<b>AI</b>	<b>Artificial Intelligence</b>
<b>BDCM</b>	<b>Brushed Direct Current Motor</b>
<b>CCP</b>	<b>Capture &amp; Compare</b>
<b>DC</b>	<b>Direct Current</b>
<b>DD&amp;GP</b>	<b>Differential Drive &amp; Global Positioning</b>
<b>EE</b>	<b>Electrical &amp; Electronics</b>
<b>GA</b>	<b>Genetic Algorithm</b>
<b>GPS</b>	<b>Global Positioning System</b>
<b>I2C</b>	<b>Inter-Integrated Circuit</b>
<b>I/O</b>	<b>Input/ Output</b>
<b>JPL</b>	<b>Jet Propulsion Laboratory</b>
<b>NASA</b>	<b>National Aeronautics and Space Administration</b>
<b>PIC</b>	<b>Programmable Integrated Circuit</b>
<b>PID</b>	<b>Proportional, Integral &amp; Derivative</b>
<b>PMDC</b>	<b>Permanent Magnet Direct Current</b>
<b>PWM</b>	<b>Pulse Width Modulation</b>
<b>U.S.A</b>	<b>United States of America</b>

# **CHAPTER 1**

## **INTRODUCTION**

### **1.1 Background of Study**

Robotics is known as a new revolution to the entity of beings that varies according to its uses. In modern day environments, robotics and automation are involved in almost every industrial activity and conveniently improve the efficiency, productivity and reliability of a system. Robotics is also implemented in medical practice, construction, outer-space exploration, household appliances and even as mobile transportation.

Currently there has been study of automated guided robots which is used in transportation and exploration that can be configured for different terrain. These designs are known as locomotion, Hopfield Neural Network, Genetic Algorithm and etc. For example, JPL (Jet Propulsion Laboratory, NASA) in U.S.A have developed many rovers. Sojourner which were landed in Mars in 1997 adopted rocker-bogie locomotion, Blue Rover took use of three-segment locomotion, the mini Mars rover Go-For has a active wheel-legged locomotion, Nano Rover utilize posable-truct chassis, and Elastic Loop Mobility System [1] was also designed as new type of locomotion for planetary exploration.

### **1.2 Problem Statement**

Robot Path Planning problem or robot Motion Planning problem is one of the famous problems in robot's offline decision making algorithms. In this problem, the aim is to find a collision free path, which the robot can follow to reach the target from its start position. Analysis and research on autonomous path planning has included innovative advancements in the use of artificial intelligence (AI) and genetic algorithm (GA).

With advancement in the study of this subject, technology with uncontrollable situations such as outer space exploration and deep sea excavation can be further improved. New technology such as autonomous vehicle systems may also be able to utilize such algorithms which are fail-safe.

### **1.3 Objective**

The main objective of this project is to create and develop a Path Planning Mobile Robot able to avoid obstacles in its path and reach a target designated position from its starting point utilizing 4-wheel based rover body, sensors, linear motors and microcontrollers.

Using statistical analysis, the eligibility and ingenuity of the AGR algorithm results will be compared with other existing system results at the end of the project period to satisfy the requirements of a path planning mobile robot.

### **1.4 Scope of Study**

The scope of this project includes robotics, control systems and artificial intelligence in the study of Obstacle avoidance using ultrasonic sonar sensing, path planning and global positioning using Genetic Algorithm (GA) with magnetic compass as direction feedback and differential drive train for linear movement using two brushed DC motors. The parameters of study extend to microcontrollers and design simulation of the AGR base which include stress tests, weighted load tests, variable speed versus torque tests, and dimension collision detection.

## **CHAPTER 2**

### **LITERATURE REVIEW**

#### **2.1 Methods of Detection and Sensor Systems as Feedback Analysis**

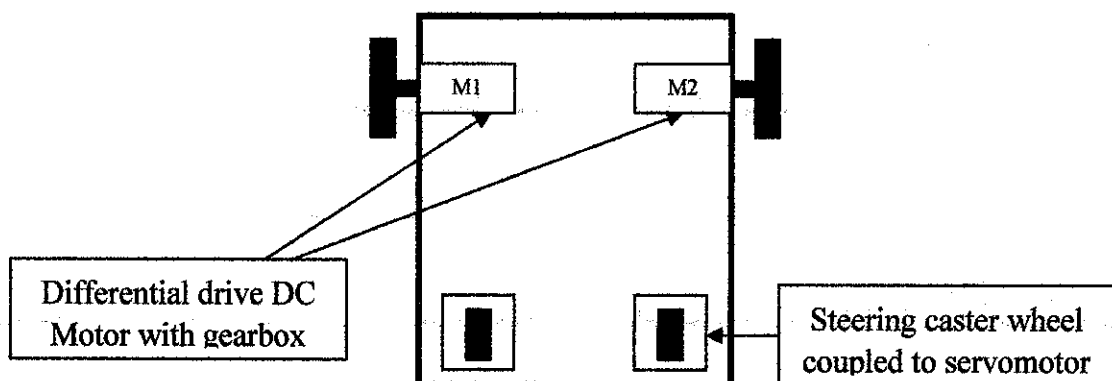
Sensors are an important aspect in the functionality of a robot. With sensors, robots are said to be able to obtain vision, sense of touch, balance, and even hearing. According to their tasks and application, robots are given the appropriate sensors that function as the feedback systems in a controller. Autonomous Guided Robot (AGR) systems are classified as rover based robotics that require vision-type and touch sensors. The AGR should be able to maneuver and counteract with the environment using sensors to detect the obstacles around, remember its current position and calculate a new path to take. [5]

The proposed sensors for the construction of the AGR in this project are ultrasonic proximity detectors, electronic magnetic compass, and optical tracking sensors. The function of the proximity detectors is to detect any objects surrounding the robot that is detected within a range specified. The proximity detectors will be placed on the front, left and right face of the robot to ensure wider sonar-like detection. An electronic magnetic compass detects the strongest magnetic flux generation and generates that as the North Pole. Using this compass, the AGR can generate sense of direction and will not lose its path once avoiding an obstacle. 3 ultrasonic sensors have been suggested for obstacle detecting and 1 magnetic compass for bearing change detection. Suggested sensor modules figures and datasheet can be referred to in Appendix C, D and E.

## 2.2 Differential drive and steering

Cars are manufactured with 4 wheels, each on each end of the car. This gives the automobile much more balance and stability than having only a single wheel at the front. The two wheels on the front help create grip when the car is driven forward by the back wheels connected to the engine. Maneuverability is done by steering the two wheels on the front, where the car is able to turn in a curve which cancels the requirement for it to stop to take a  $90^0$  turn. This not only increases the speed of movement, but also reduces power consumption. This exact design could also be used on the robot rover intended in this project.

This possible design was sketched and a brief idea on steering was developed in the progress. Instead of using a stepper motor which is ultimately heavier, it is suggested that two servo motors are used to steer individually the two front tires. But this individual steering is actually done in synchronization such as a car's, where both tires turn the same amount and degree. The standard servo motors would be able to turn  $180^0$ , which can be set to even  $90^0$  to the left and right. But as mentioned before, the rover would not require to stop and make a perfect  $90^0$  turn instead just take a corner when turning. The DC motor drivers will still be maintained to drive the rover and control the speed of the rover.



**Figure 1** Differential drive robot with servo steering build

### ***2.2.1 Comparison Between Servo And Stepper Motor In Direction Control***

#### **Stepper Motors**

Stepper motors are less expensive and basically easier to use than a servo motor of a similar size. It is called stepper motors because it moves in discrete steps. Controlling a stepper motor requires a stepper drive and a controller. You control a stepper motor by providing the drive with a step and direction signal. The drive then interprets these signals and drives the motor. Stepper motors can be run in an open loop configuration (no feedback) and are good for low-cost applications. In general, a stepper motor will have high torque at low speeds, but low torque at high speeds. Movement at low speeds is also choppy unless the drive has microstepping capability. At higher speeds, the stepper motor is not as choppy, but it does not have as much torque. When idle, a stepper motor has a higher holding torque than a servo motor of similar size, since current is continuously flowing in the stepper motor windings. [8]

Some of the advantages of stepper motors over servo motors are as follows:

- Low cost
- Can work in an open loop (no feedback required)
- Excellent holding torque (eliminated brakes/clutches)
- Excellent torque at low speeds
- Low maintenance (brushless)
- Very rugged - any environment
- Excellent for precise positioning control
- No tuning required

Some of the disadvantages of stepper motors in comparison with servo motors are as follows:

- Rough performance at low speeds unless you use microstepping
- Consume current regardless of load
- Limited sizes available
- Noisy
- Torque decreases with speed (you need an oversized motor for higher torque at higher speeds)
- Stepper motors can stall or lose position running without a control loop

## **Servo Motors**

One of the main differences between servo motors and stepper motors is that servo motors, by definition, run using a control loop and require feedback of some kind. A control loop uses feedback from the motor to help the motor get to a desired state (position, velocity, and so on). There are many different types of control loops. Generally, the PID (Proportional, Integral, Derivative) control loop is used for servo motors. When using a control loop such as PID, you may need to tune the servo motor. Tuning is the process of making a motor respond in a desirable way. Tuning a motor can be a very difficult and tedious process, but is also an advantage in that it lets the user have more control over the behavior of the motor.

Since servo motors have a control loop to check what state they are in, they are generally more reliable than stepper motors. When a stepper motor misses a step for any reason, there is no control loop to compensate in the move. The control loop in a servo motor is constantly checking to see if the motor is on the right path and, if it is not, it makes the necessary adjustments. In general, servo motors run more smoothly than stepper motors except when microstepping is used. Also, as speed increases, the torque of the servo remains constant, making it better than the stepper at high speeds (usually above 1000 RPM). [8]



Some of the advantages of servo motors over stepper motors are as follows:

- High intermittent torque
- High torque to inertia ratio
- High speeds
- Work well for velocity control
- Available in all sizes
- Quiet

Some of the disadvantages of servo motors compared with stepper motors are as follows:

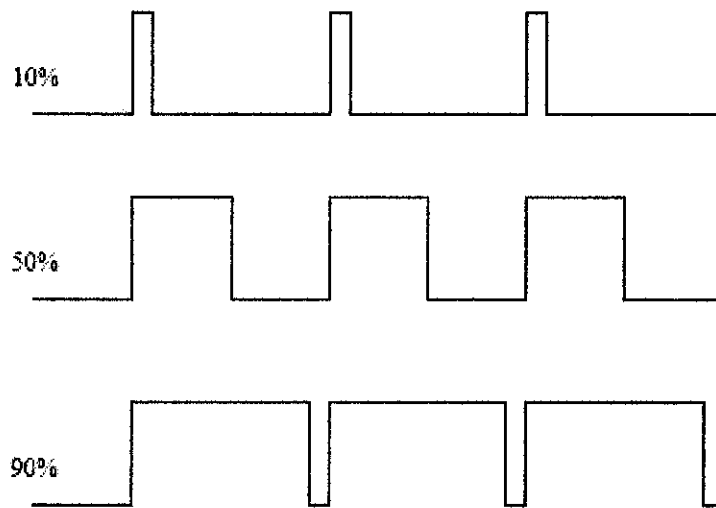
- More expensive than stepper motors
- Cannot work open loop - feedback is required
- Require tuning of control loop parameters
- More maintenance due to brushes on brushed DC motors

In this thesis, the author has weighted the advantages of servo motors to be much more appropriate for the use in AGR steering control because of the higher reliability and more control over the movement. Precision is not required in unexplored areas as movement would be random for the AGR, but accuracy and speed is essential to amount the proper control over the turning process of the AGR when detecting obstacles.

### ***2.2.2 Pulse Width Modulation (PWM) Motor speed control***

PWM (Pulse Width Modulation) is the most common technique that is used to control the speed of a PMDC motor. In this technique a rectangular voltage waveform of a certain amplitude and frequency is applied to motor armature windings. The duty-cycle of the waveform is made variable. An increase or decrease in duty-cycle increases or decreases the average power delivered to the armature windings. The speed of the motor varies in proportion to the average power delivered to the armature

winding. The PWM can be generated through programming the PIC16F877A microcontroller which already has preset PWM pins CCP1 and CCP2.



**Figure 2** Sample PWM waveforms for stated duty-cycles

### ***2.2.3 H-Bridge Motor Speed Controller Board***

There are number of ways to accomplish the electronic or electrically assisted direction control of motors, for example:

- *Relay Control*
- *Bipolar Transistor H-bridge Network*
- *Power MOSFET H-bridge Network*
- *Motor Bridge Control*

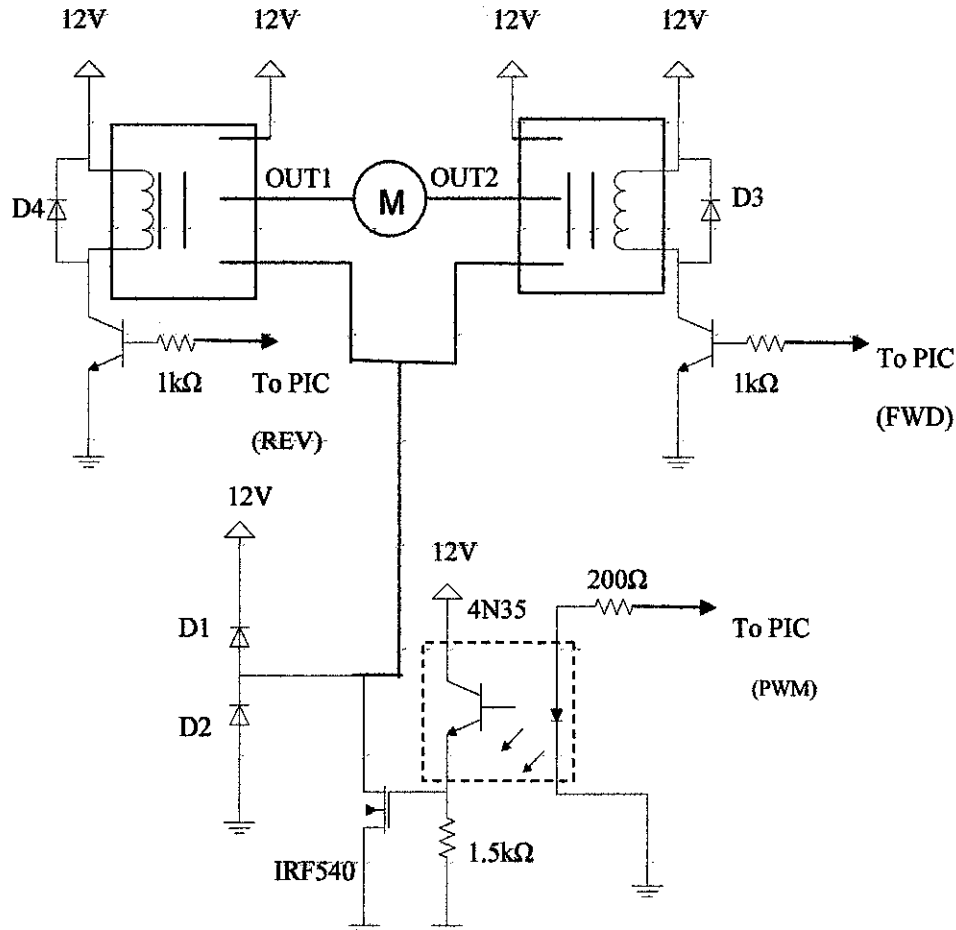
The clear advantage of using one of the above circuits is that they work as a buffer between the motor and the microcontroller or microprocessor. The motor draws a substantial current in loaded condition ranging from 1A for small DC motor to as much as 4A for a medium size DC motor. A microcontroller or microprocessor cannot provide this much amount of current. Therefore these buffers (also called *motor drivers*) are required to fulfill this need and save microprocessor or microcontroller from burning due to excessive heat produced due to large current sinking.

H-bridge is the most common method to control a DC motor. It can be implemented with Bipolar-transistor or with Power MOSFET. But their circuitry requires large heat sinks because in operation they dissipate considerable amount of heat, and therefore cover large space. [6]

The H-bridge used in this project is the Relay controlled Power MOSFET H-Bridge motor driver [7]. Below is the components list for the driver:

- 12V single-pole relay X 2
- 1N4148 fast-recovery zener diodes X 4
- IRF540 Power MOSFET X 1
- 4N35 Optocoupler X 1
- 2N3904 NPN Transistor X 2
- Resistors:  $1k\Omega$  X 2,  $1.5k\Omega$  X 1,  $200\Omega$  X 1

The schematic is as follows:



**Figure 3** Relay controlled motor driver

The truth table of the motor logical inputs and the outputs is shown below:

**Table 1** Truth table for I/O of motor driver

Input		Output	
FWD	REV	OUT1	OUT2
L	L	L	L
H	L	H	L
L	H	L	H
H	H	L	L

#### 1. Forward / reverse control

When FWD is HIGH and REV is LOW, current flows from OUT1 to OUT2. When FWD is LOW and REV is HIGH, current flows from OUT2 to OUT1 (refer to the truth table).

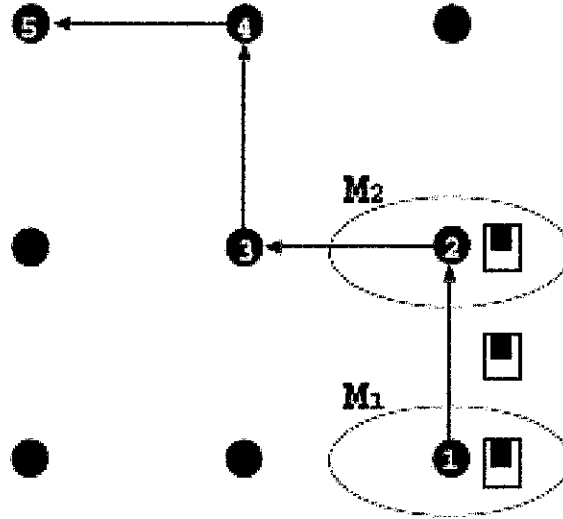
#### 2. Forced stop

By setting REV and FWD, either both HIGH, or both LOW, power supply to the motor is shut down and a brake is applied by absorbing the motor counter-electromotive force.

### 2.3 Path Planning and Obstacle Avoidance

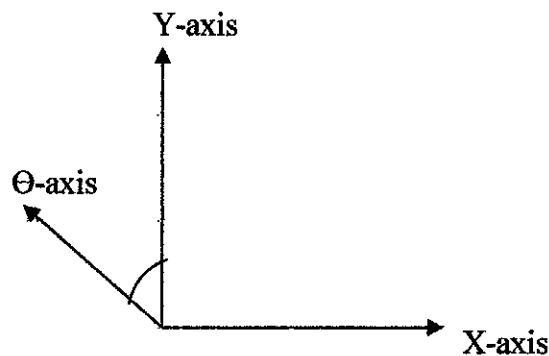
#### 2.3.1 3D arrays in the use of grid-based global positioning

In the development of global positioning for a mobile robot, the frequent use of 2D arrays commonly arise as the most basic and effective method. Logically in mapping a certain area, the normal axis that would be noticed is the X-axis and the Y-axis which will give suitable amount of information for the robot to perform a node-wise recognition and carry out global positioning. The figure below shows a node-wise 2D movement of a simulated robot from node 1 to 5 which uses sensor networks [3]:



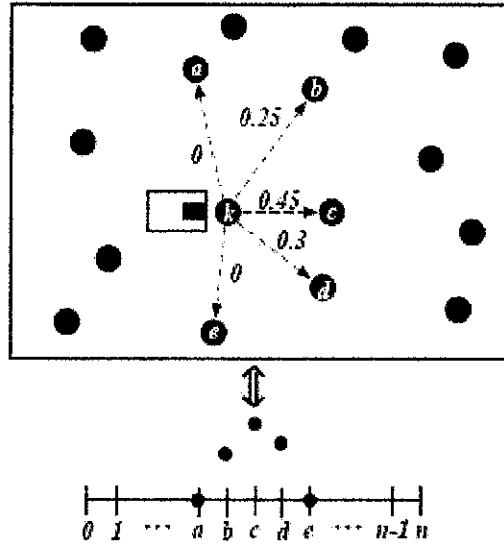
**Figure 4** Node-wise 2D positioning

The effectiveness of such networks have been proven to be 80% efficient and are reliable in a fast-response and rugged situation. In this project, a slightly more complex 3D form of array hopefully could be applied for the grid-based global positioning. The axis would consist of the normal X and Y-axis, and an additional radius ( $\Theta$ ) axis. This  $\Theta$ -axis is possible with the use of the CMPS03 magnetic compass which would feedback the bearing value of the robot. This bearing value is converted to degrees which would be used in determining the radius of perception for the AGR. With the use of this axis, certain error in 2D arrays could be dealt with which would have resulted from the robot having a slight bearing change which would ultimately change the accumulated X or Y value. Correction can be done using the  $\Theta$ -axis value.



**Figure 5** 3D axis

To apply such arrays, a probabilistic approach and mathematical function must be developed to enhance the array. In a journal written by Maxim A. Batalin and Gaurav S. Sukhatme [3], the theoretical iteration of probabilities would be applied according to the number of actions and nodes that could and would be developed by the artificial intelligence. This would of course be a finite number, which could be bounded by the wanted map of the area in discussion.



**Figure 6** An example of a discrete probability distribution of vertex (sensor node) k for direction (action) "East"(i.e. right).

FIGURE 6 shows an example given by [3] of a typical discrete probability distribution for a vertex (sensor node) per action (direction). Mass distribution of probability is a sum of 1, and nodes with obstacles will be stated as zero.

The mathematical model that I have developed is based upon the understanding that the 3rd axis is used as an error correction for X and Y, thus the system is the sum of changes of the bearing towards the probability matrix of XY:

$$U_{(t+1)} = \sum [P(t) * U(t)] \pm \arg \Theta(t)$$

Where  $U(t)$  = array iteration for actions possible

$P(t)$  = Probability matrix of XY

$\Theta(t)$  = Bearing change of compass (range)

### **2.3.2 Wisp628 Microcontroller Programmer Board**

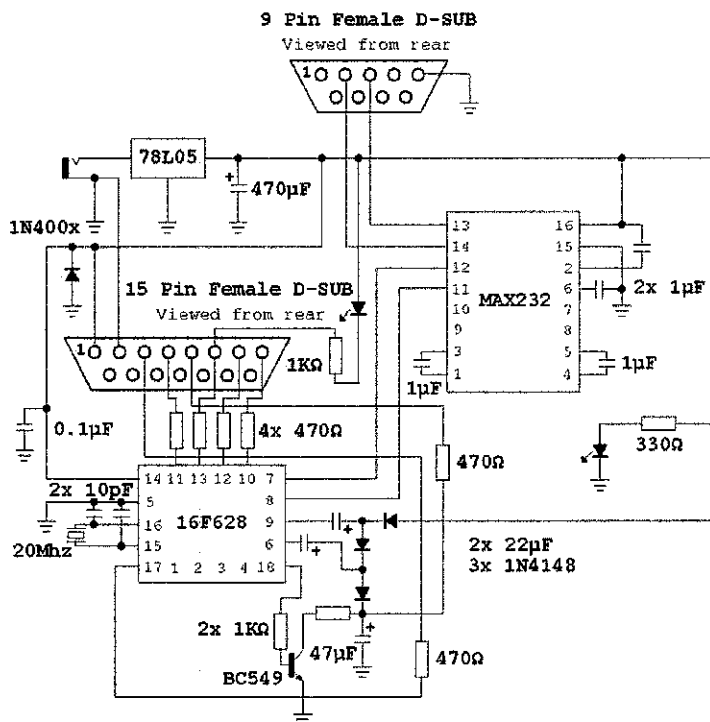
As a controlling platform for the system, the central processing is done through a master microcontroller which is the PIC16F877A microcontroller (Appendix A). The controller or more commonly known as target board is able to control all I/O devices integrated in to the AGR system including the sensors and the transducers. The target board created for the AGR requires an interfacing device for programming through the PC which is done by integrating the Wisp628 Microcontroller programmer board.

The Wisp628 PIC Microcontroller Programmer board is a multi-pin programmer utilizing RS232 as interface to the computer console. This programmer is a basic and vital requirement in this project as the programmer will convert C program files into assembly machine language saved into the microcontroller that will be used to control the behavior of the robot. The Wisp628 is a license-free based programmer that is shared online for use by freelance robotics enthusiast. The programmer is an interfacing system between the PC and the target board.

The programmer was built according to the instructions of the author and using the firmware provided by the author. Below is the components list for Wisp628 Programmer circuit:

- 6x 470 $\Omega$  1% resistors.
- 1x 330 $\Omega$  1% resistor.
- 3x 1K $\Omega$  1% resistors.
- 4x 10K $\Omega$  1% resistors.
- 1x 0.1 $\mu$ F disc ceramic capacitor.
- 4x 1 $\mu$ F electrolytic capacitors. (25V or more)
- 2x 22 $\mu$ F electrolytic capacitors.
- 1x 47 $\mu$ F electrolytic capacitor.
- 1x 470 $\mu$ F electrolytic capacitor.
- 1x 1N400x 1A power diode.
- 3x General purpose silicon diodes. (I used 1N4148)
- 1x LED for debugging.
- 1x 20Mhz parallel cut crystal.

- 1x NPN bipolar transistor. (most standard 100mA types will do, I used a BC549)
- 1x 78L05 100mA 5V voltage regulator.
- 1x MAX232 line driver or substitute.
- 1x PIC16F628 programmed with firmware.
- 1x Prototyping PCB. (E.g. RS 435-298, anything with 400 holes or more should be fine.)
- 1x DC power jack.



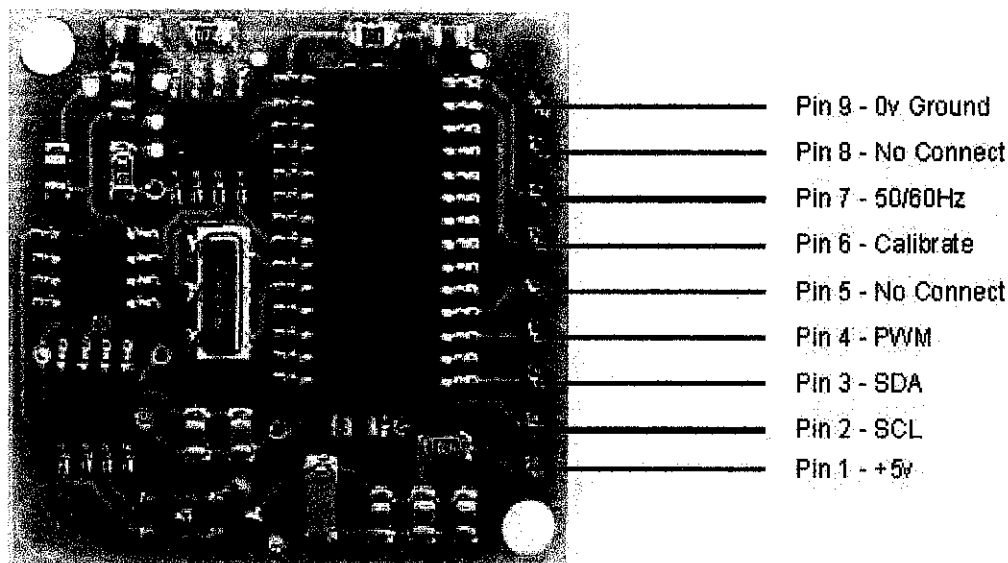
**Figure 7** Wisp628 Schematic Diagram

### 2.3.3 CMPS03 Electronic Magnetic Compass

As a feedback controller, the AGR requires certain sensing abilities to help it implement decisive movements from one point to another. The CMPS03 magnetic compass acts as a sensor to help the robot detect where it is facing. The compass uses the Philips KMZ51 magnetic field sensor, which is sensitive enough to detect the Earth's magnetic field. The output from two of them mounted at right angles to each other is used to compute the direction of the horizontal component of the Earth's magnetic field.



The CMPS03 is a dual-compatible sensor which can be interfaced with either using I2C or by reading the PWM signal from designated pins. Figure 1 shows the layout of the pins:



**Figure 8** Connections to the compass module

Since the microchip being used supports both interfaces, the I2C interface is chosen for the more simpler read-in data sent to the PIC. The PWM signal is a pulse width modulated signal with the positive width of the pulse representing the angle. This means it requires a special call function to convert this pulse into degree bearing. The I2C interface gives a direct bearing value of 0 to 255( $360^{\circ}$ ). The author has produced conversion codes to allow a degree bearing of 0 to  $360^{\circ}$  to be used.

I2C communication protocol with the compass module is the same as popular EEPROM's such as the 24C04. First send a start bit, the module address (0XC0) with the read/write bit low, then the register number you wish to read. This is followed by a repeated start and the module address again with the read/write bit high (0XC1). You now read one or two bytes for 8bit or 16bit registers respectively. 16bit registers are read high byte first. The compass uses an array of 16bit registers:

**Table 2** I2C interface 16bit registers

Register	Function
0	Software Revision Number
1	Compass Bearing as a byte, i.e. 0-255 for a full circle
2,3	Compass Bearing as a word, i.e. 0-3599 for a full circle, representing 0-359.9 degrees.
4,5	Internal Test - Sensor1 difference signal - 16 bit signed word
6,7	Internal Test - Sensor2 difference signal - 16 bit signed word
8,9	Internal Test - Calibration value 1 - 16 bit signed word
10,11	Internal Test - Calibration value 2 - 16 bit signed word
12	Unused - Read as Zero
13	Unused - Read as Zero
14	Unused - Read as Undefined
15	Calibrate Command - Write 255 to perform calibration step. See text.

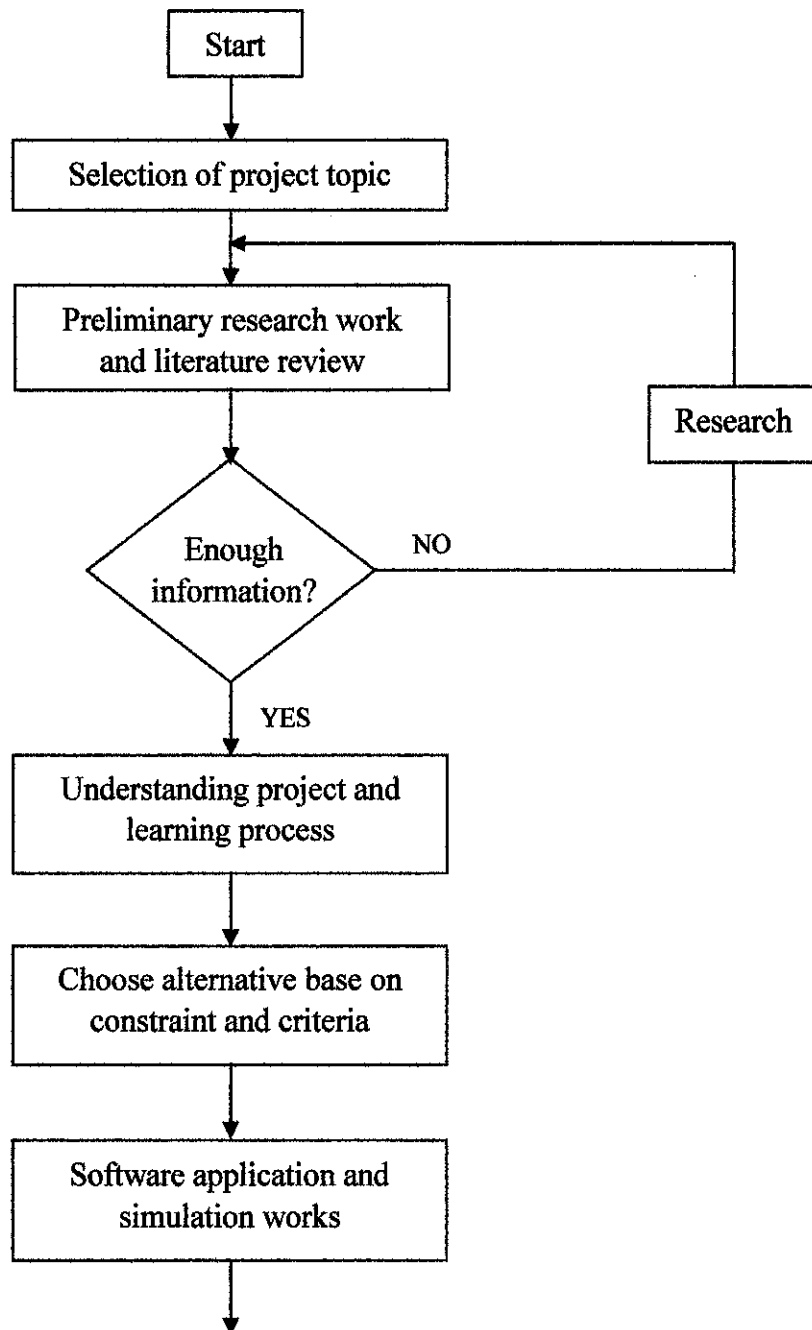
Clearly it can be noticed that the register that will be used is Register 1 which supplies the compass bearing as a byte. The read speed of the module is estimated at 60 $\mu$ s, which is sufficiently fast for a continuous feedback control of the AGR. Before running the device, it must be calibrated to read either a north bearing or a fixed position bearing. Refer to appendix D for more information on the module.

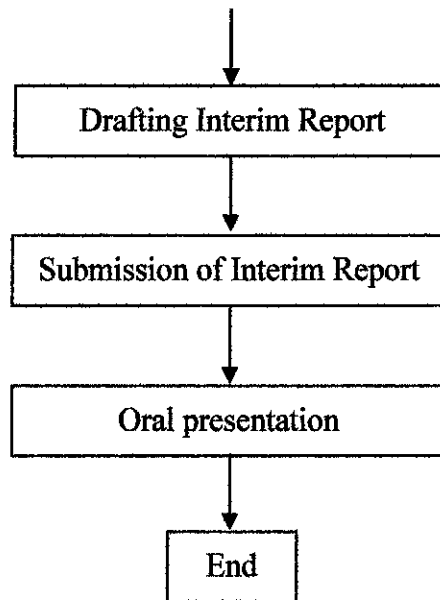
## CHAPTER 3

### PROJECT WORK

#### 3.1 Project Process Flow

The process flow of the project will be done based on the simple flow which is then applied throughout the project (Figure 6).

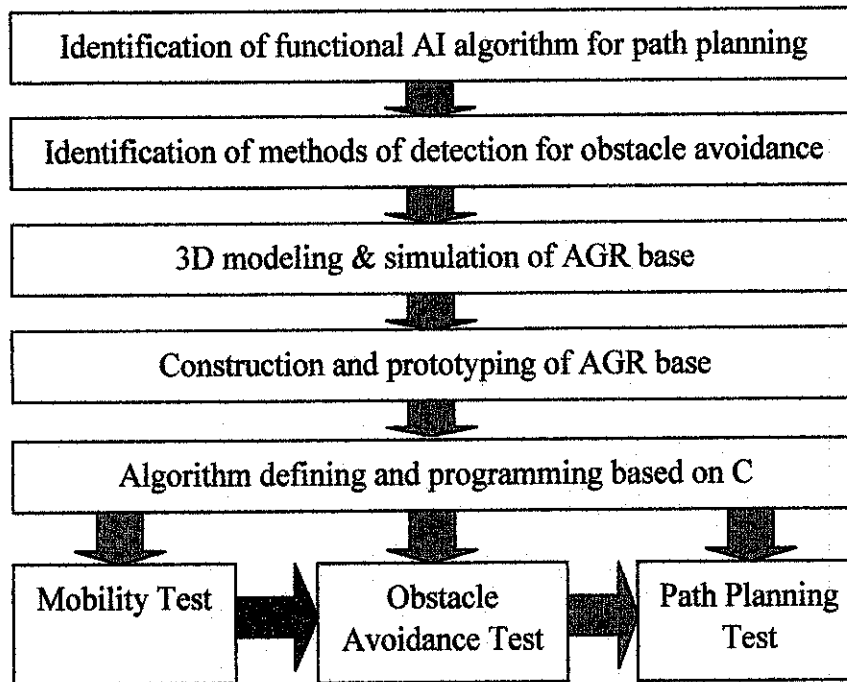




**Figure 9** Process flow of project

The first step that is taken into consideration is to select the project topic and thus conduct preliminary researches and literature review through the available source such as using the internet, text books, journals as well as conference papers. As the information is available, it is very crucial for the author to undergo learning process as robotics is not a familiar area to him/her. From the data gathered, the alternatives will be layered out and the best alternatives will be chosen based on constraint and criteria. The next step would be to transfer or apply the information gathered by using related software or doing some stimulation pertaining to the project. The following step is the drafting of interim report which has to be submitted before the final interim report is written and thus submitted to the committee. The final stage of the process flow of the project is the oral presentation which will take place towards the end of the semester.

### 3.2 Procedure Identification



**Figure 10** Procedure identification and flow diagram

The main task beforehand is the identification of control systems and methods of detection to be used for the AGR mobile path planning. This step requires data acquisition and comparison studies for existing systems and robots that utilize such technology. The next step is the 3D modeling and simulation of AGR base which requires hands-on training on 3D CAD works or Solidworks drawings. Also included in this step is the simulation and designing of electronic controller circuits for the AGR using PSpice and EAGLE layout editor. The advantage of this step is the ability to detect error or defects on the prototype model before construction phase is taken.

The next step is the construction and prototyping which will consist of two steps. The first step is the fabrication of a prototype board and circuits for testing and troubleshooting purposes. The second step is the fabrication of actual AGR base with PCB done circuitry which has been verified through the prototype. This step is completed in the next semester of the Final Year Project. The final step is the algorithm defining and programming of AGR controllers based on obstacle avoidance, path planning and global positioning algorithms. This step will be done throughout the project period and will also be accompanied by testing methods for

each control algorithm. These tests are done multiple times to attain quality feedback of the system and for troubleshooting purpose. The tests consist of three phases, which are the mobility, obstacle avoidance and path planning tests.

The main objective of the mobility test is to acquire information on the maneuverability of the AGR. The author selects 3 criteria for this test; that includes speed control, turning and forward alignment. Hardware and software configuration will be troubleshoot until the robot is able to carry out all criteria accurately before the next test phase is done. The second phase is the obstacle avoidance test. In this phase the objective is to complete random obstacle avoiding in a predetermined area. The AGR will be tuned and tested to avoid walls and objects according to the algorithm created. The third phase is the path planning test. The AGR is tested in two environments, predetermined and random area. In this phase, no obstacles are placed accept limiting walls and troubleshooting is done based on the bearing change of the compass. Once completed, a combined approach of both obstacle avoidance and path planning is carried out as the final test method.

### **3.3 Tools Required**

#### ***3.3.1 Hardware Configuration***

- Ultrasonic proximity detector (SRF04)
- Electronic Magnetic Compass (CMPS03)
- PIC16F877A Microcontroller + Target Board
- Planetary gear Brushed DC Motor
- Standard servo motor
- Robot rover base with 4 wheels

#### ***3.3.2 Software Requirements***

- PICC Compiler for Microcontrollers
- MATLab ®
- Solidworks 2006 3D Modeling software
- PSPICE A/D

## **CHAPTER 4**

### **RESULTS & DISCUSSION**

#### **4.1 Body Structure and Mechanism**

The body structure of the AGR was designed and constructed in 3 phases which are:

1. Generation of 3D Model design
2. Simulation of 3D Model and drive train
3. Fabrication of prototype structure using aluminum bars and other materials.

Utilizing simulation programs such as MATLAB SIMULINK and SolidWorks, the basic criterion and the capacity of the AGR could be analyzed and predicted before the actual model was made. Once tests were carried out in the simulation which included stress test, weight to torque ratio, etc, the components such as the DC motor and materials were purchased. These simulations actually help in reducing the cost and duration of the project as less/no modification needs to be done to the structure once a perfectly precise 3D model has been created.

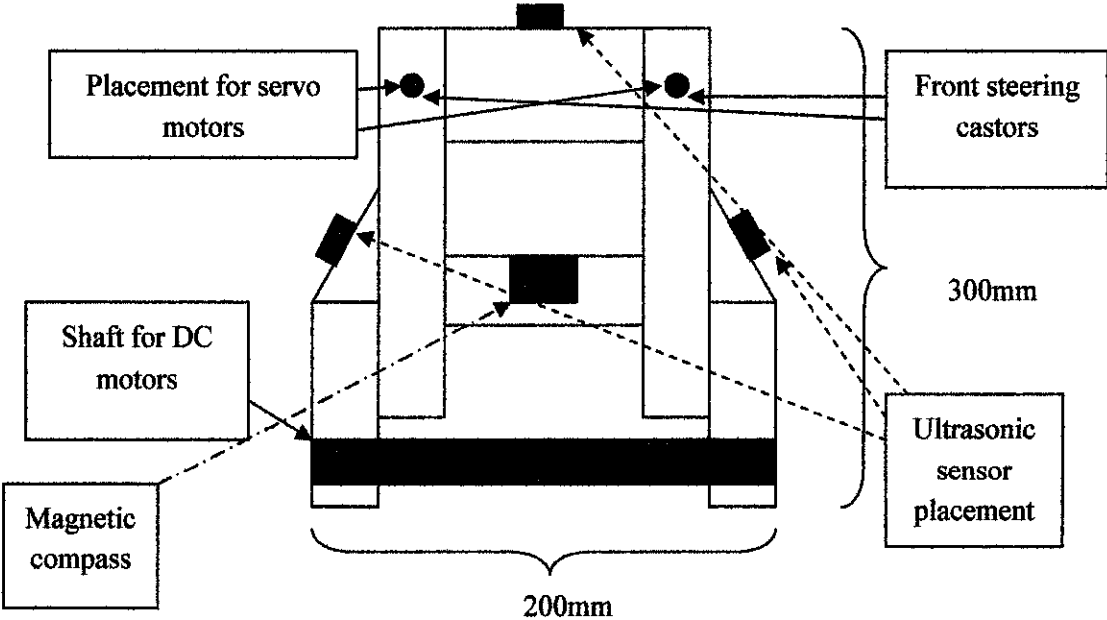
##### ***4.1.1 Generation of 3D design***

The generation of 3D model required a CAD or solid structure designing & analysis tool software and a sketch of the base design with constraints decided upon for the AGR. The software used for this purpose is Solidworks® 2006 3D modeling software. This software as stated by the developers is an award winning software that is in use at over 12,000 major educational institutions around the world that provides all the design engineering, data management, and communications tools required for 3D modeling.

There are also design validation tools embedded into Solidworks such as COSMOSWorks which is a powerful, easy-to-use design validation and optimization

software, COSMOSMotion which is a complete motion simulation and kinematics package, and COSMOSEMS™ which is a 3D-field simulator that allows electronic product designers to simulate the effect of components exposed to low frequency electromagnetic currents and fields.

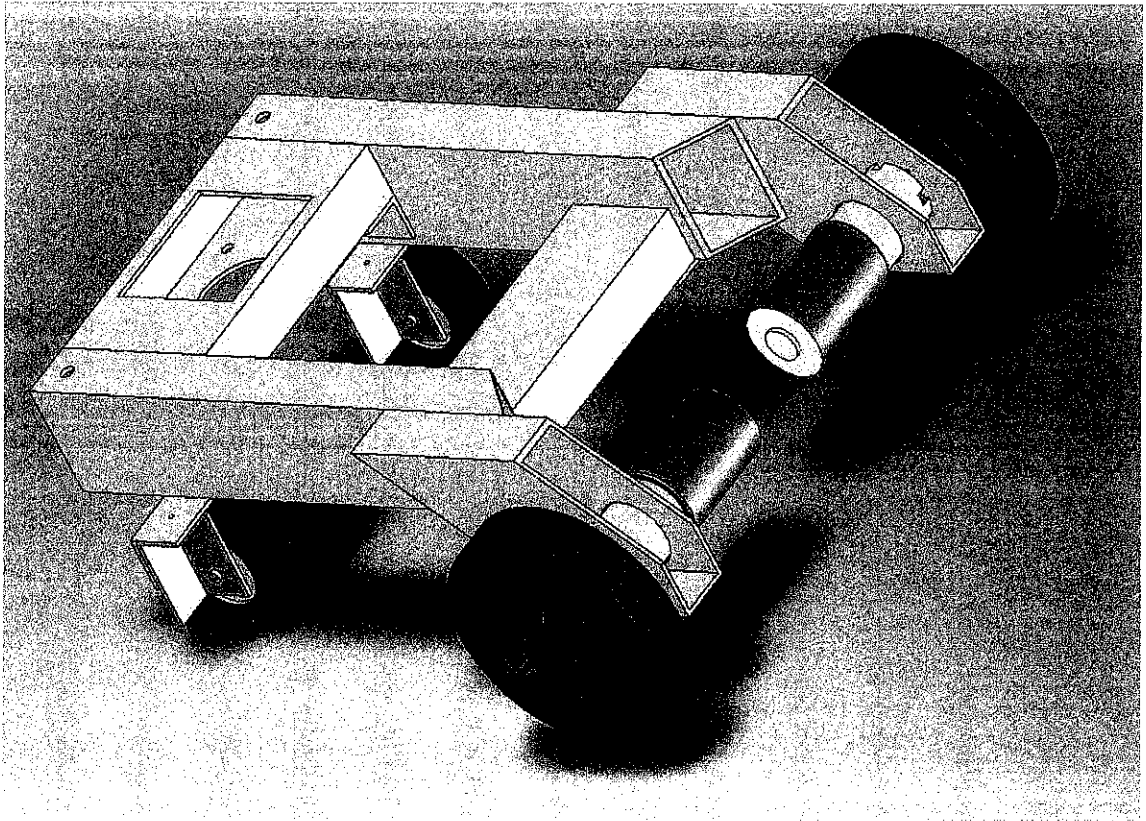
Before carrying out the design, a sketch was made with design constraints which include minimal dimension of 40mm<sup>2</sup> for the base; and components required and placement. The design also includes the dimensions for the material used which is rectangular hollow aluminum 2" x 1", rectangular hollow 3" x 1", L shaped aluminum 1" x 1" and Tube hollow aluminum 2" diameter. The components selected for the AGR are the RS stepper motor for steering control of the 2 front castor wheels, 2 DC brushed planetary geared motors for driving, the magnetic compass for direction feedback, 3 ultrasonic proximity sensors for obstacle detection and USDigital rotary encoder for distance feedback. Figure 11 shows the dimension constraint for the robot base:



**Figure 11** Top view of AGR base 2D sketch

The 3D model created is the base of the AGR. This model is according to the dimensions stated in Figure 11 with holes and structural couplings for components and base connection of the base. Figure 12 shows the 3D model created in Solidworks 2006:





**Figure 12** Full solidworks 3D model of AGR

#### ***4.1.2 Simulation of DD&GP using Simulink MATLAB***

The “Differential Drive and Global Positioning Blockset” is the tool which can be used with Simulink® for modeling, designing, and simulation of the dynamics of a type of vehicle robots called differentially driven vehicle robot. This tool carries the blocks with built-in Global Positioning algorithm (related to Dead-reckoning), which are useful in finding out the current position and heading of the vehicle robot. This blockset will allow the scientists and engineers to build and simulate their models of vehicle robot using differential drive steering method according to their own design and specifications, and will enable to visualize the results in the form of model-driven animation. [6]

This simulation is to calculate the appropriate loading and the balance of the robot with the use of rated brushed DC motors and the wheel radius. The ratings are calculated and entered into the blockset and a system is built using these blocksets to generate a differential driven robot for simulation purposes. The ‘Animation block’ generates the animation of the differentially driven vehicle robot. In the animation-figure the robot is represented in the form of a blue rectangular surface which moves

and turns according to the data provided by the Simulink model. Note that the area provided by the animation-figure is 4.5 meters in length and 6 meters in width, therefore, the dimensions of the robot should be small enough so that the movement of the robot can be easily observed in that area.

The ratings and values of gain for PID controller derived from the transfer function of the BDCM are entered in the blockset and the simulation is done. The transfer function is as follows:

$$\frac{\omega(s)}{V_a(s)} = \frac{K}{(Js + b)(Ls + R) + K^2}$$

Where,  $L_s$  = Armature Inductance (H)

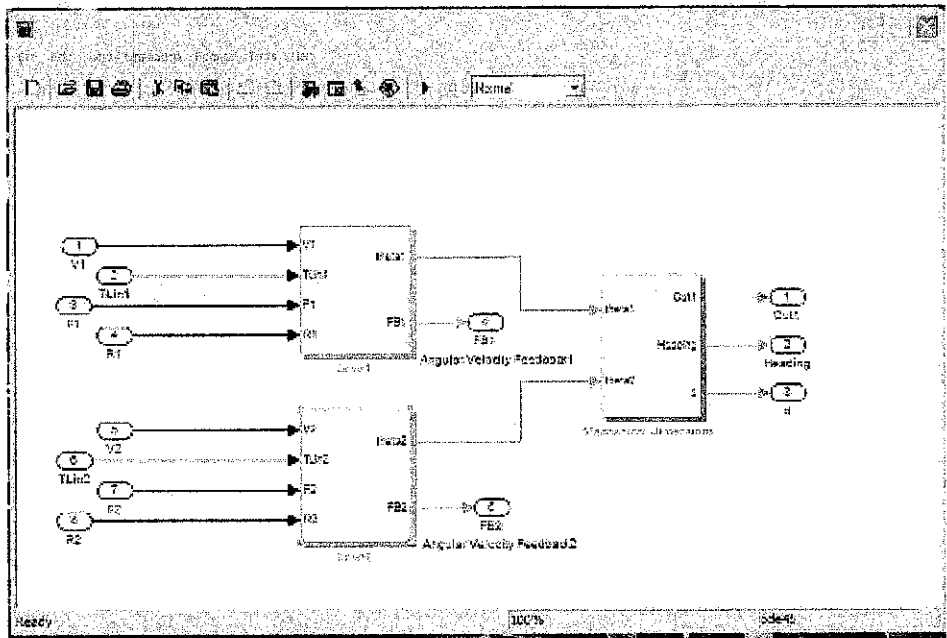
$R$  = Armature winding Resistance (Ohm)

$K_b = K_m = K$  = Motor torque constant (N-m/A)

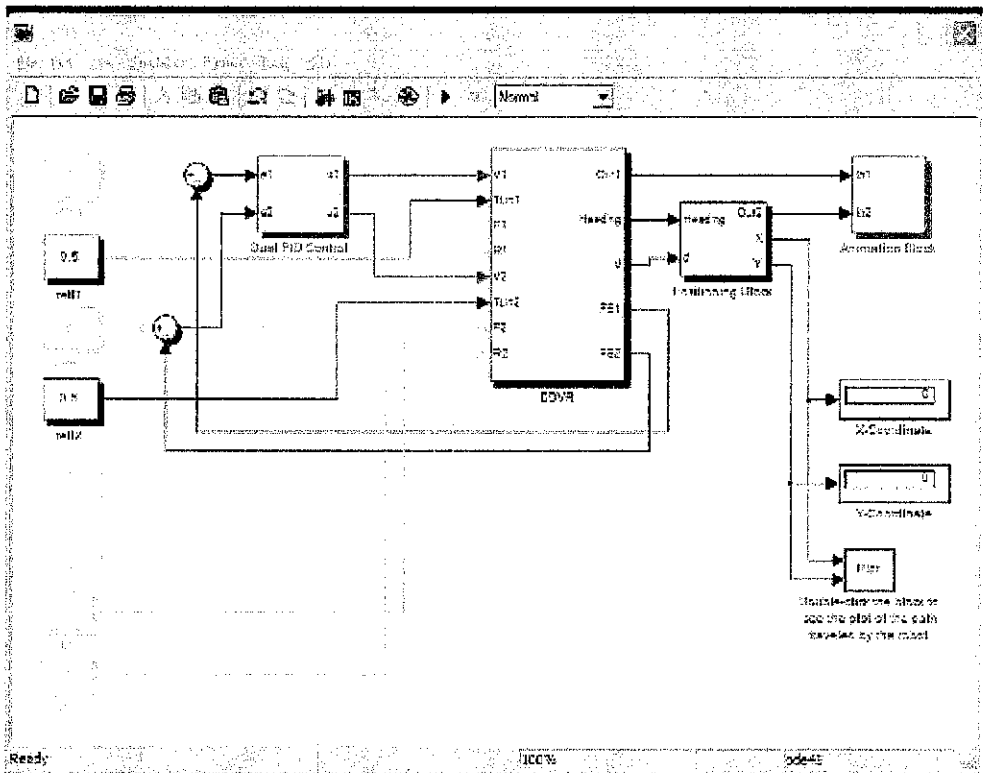
$J_s$  = Moment of inertia ( $\text{kg-m}^2$ )

$b$  = viscous friction coefficient (N-msec/rad)

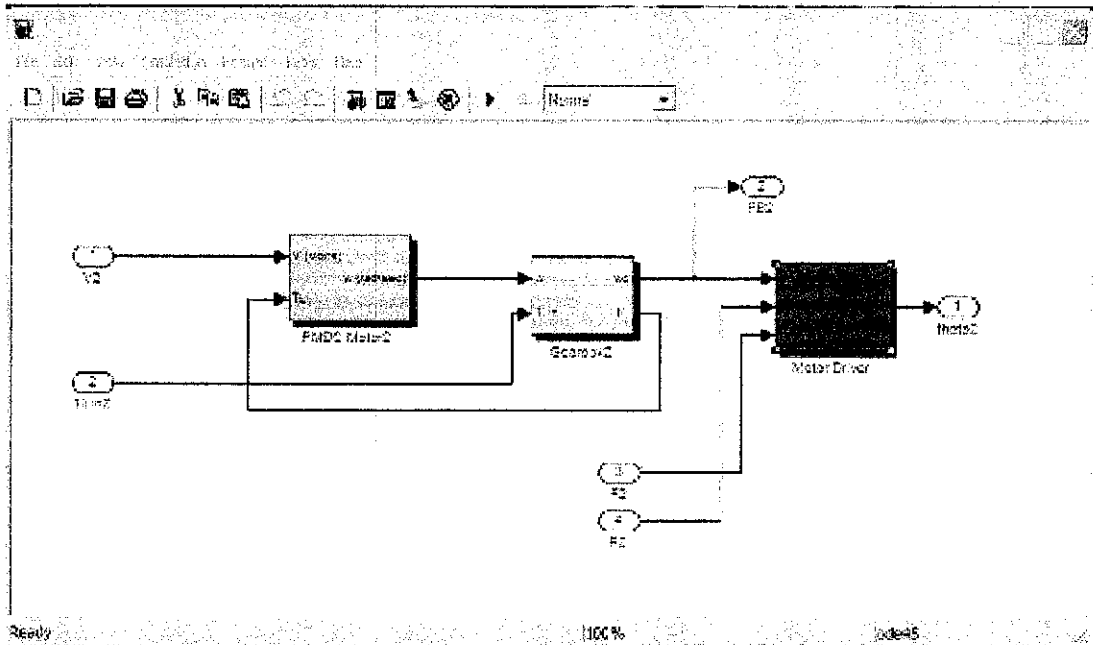
The simulation consists of variable values  $V_1$ ,  $V_2$ ,  $L_1$  and  $L_2$ .  $V_1$  and  $V_2$  are the variable speed drive of each corresponding motor and  $L_1$  and  $L_2$  are the loading capacity of the corresponding wheels. The simulation is also able to produce results for the angular arc movement described beforehand for obstacle avoidance which can be used directly.



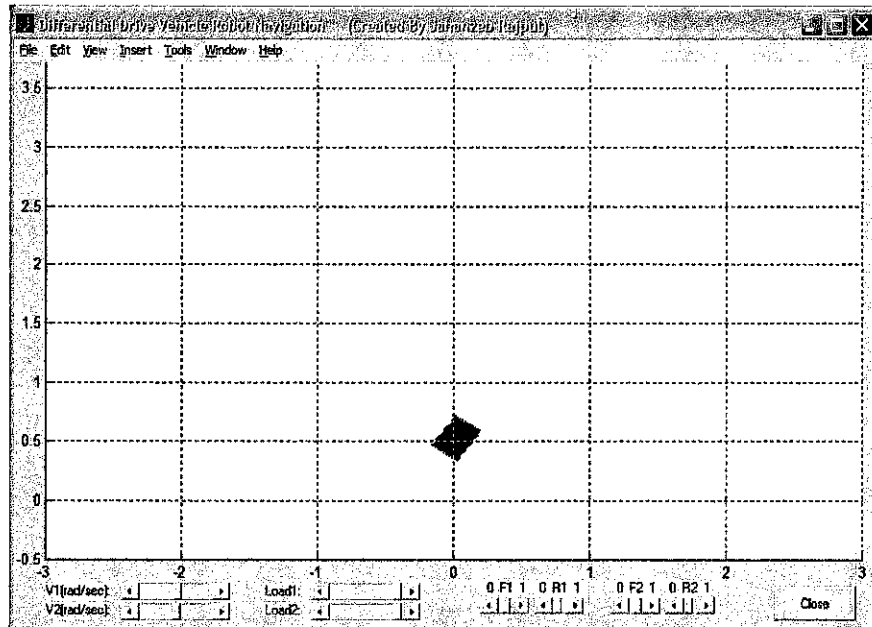
**Figure 13** Full Differential drive and global positioning blockset using Simulink



**Figure 14** The Main PID controller blockset breakdown with error constants

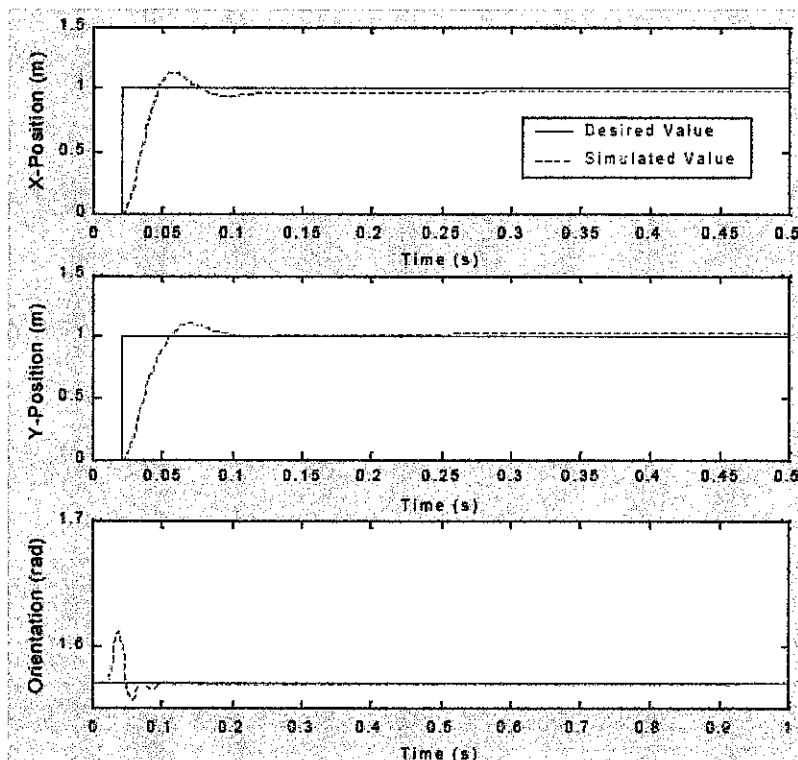


**Figure 15** The differential drive blockset breakdown for DC motor



**Figure 16** Results of simulation displayed as a figure

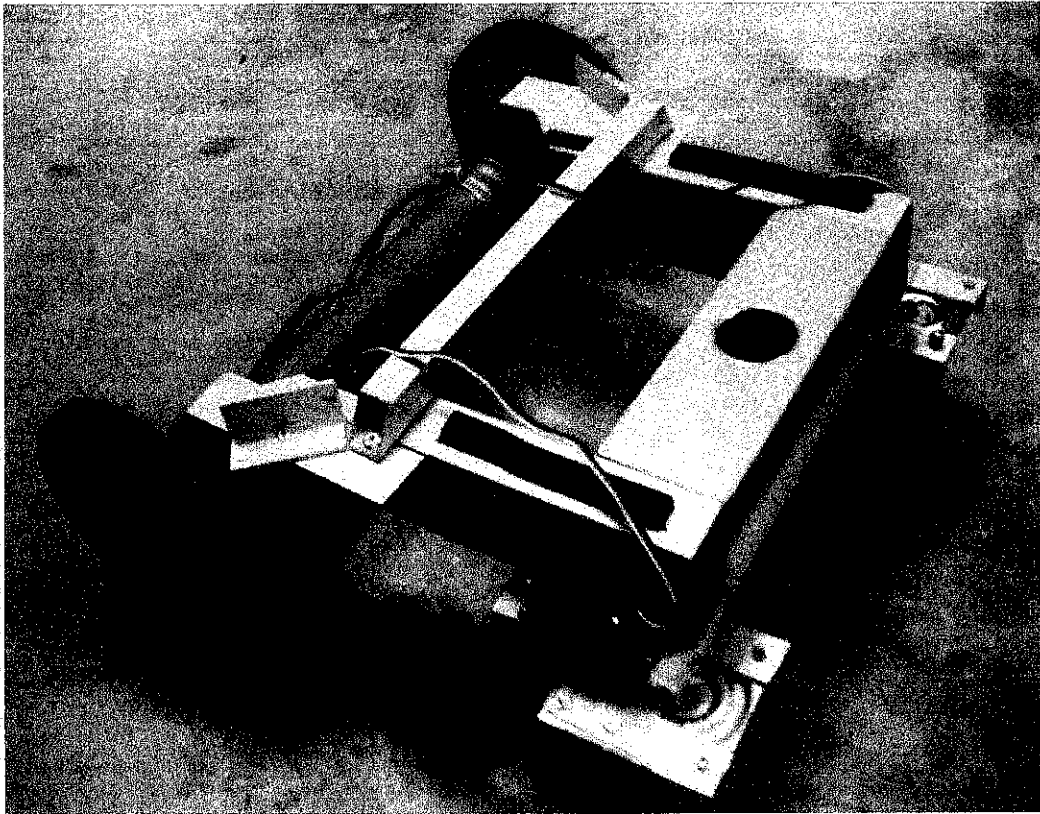
As seen from the graph figure 16, the blue box moves differentially by the change of  $V_1$ ,  $V_2$ , Load1 and Load2. Higher  $V_1$  and  $V_2$  values results in faster movement and the difference between the two results in change of direction. The figure below shows the graph of summarized results for PID controller orientation and error calculation according to the  $\Theta$ , X and Y axis:



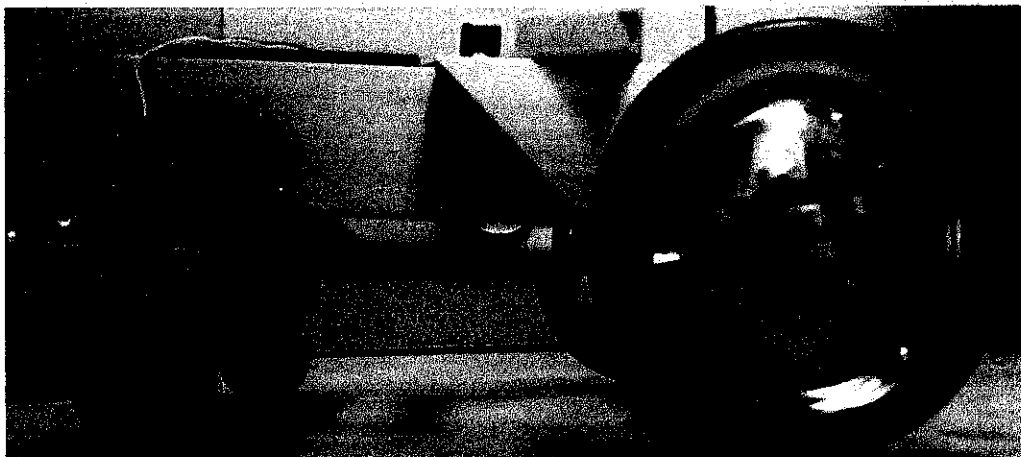
**Figure 17** Results for PID controller orientation

#### ***4.1.3 Actual fabrication of AGR prototype structure***

Using aluminium hollow and L bars, the AGR prototype was constructed and fabricated according to the design criteria and size. Certain optimizations were done to make sure the construction is solid such as the addition of a coupling pipe for both the DC motors to make sure the alignment of both tyres are the same. The mounting for the sensors were also fabricated, and the circuit mounting was also predetermined before construction. The fabricated prototype is shown in Figure 18:



**Figure 18** Fabricated AGR isometric view



**Figure 19** Fabricated AGR side view

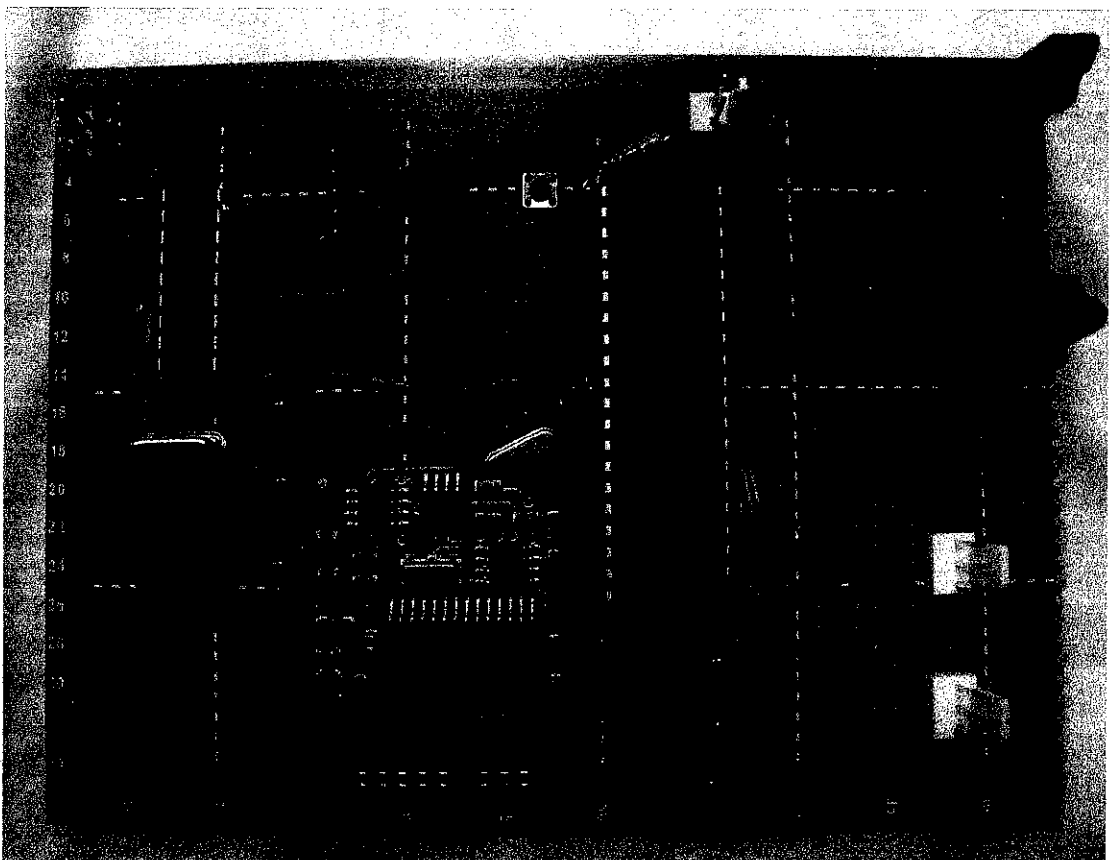
## 4.2 Circuit Construction

The construction of control circuits is based upon the circuits discussed in Chapter 2 of the report. The circuitry is designed and fabricated through a few phases which are:

1. Electronic Circuit design and simulation in PSPICE AD
2. Purchase/ acquire components and wires from EE store
3. Circuit testing and troubleshooting on breadboard
4. Veroboard finalized working circuit design.

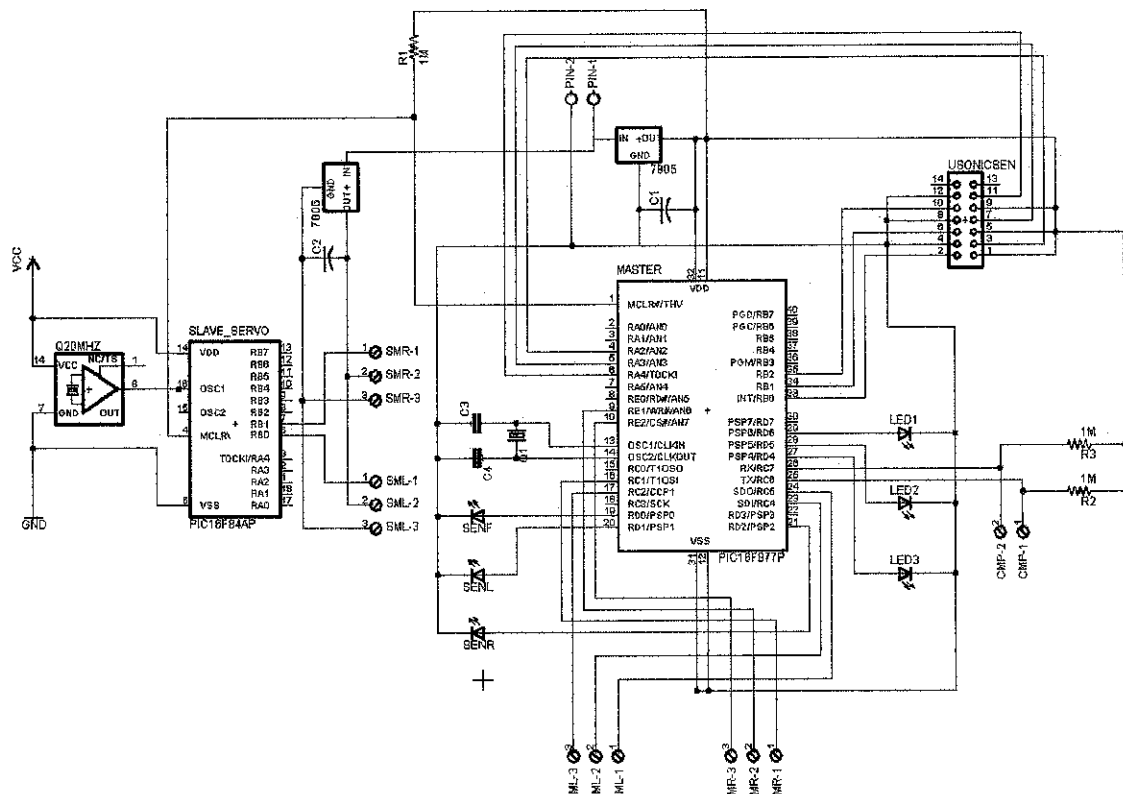
### 4.2.1 Master and Slave Controller Target Board

According to the schematic shown Chapter 2, the programmer was assembled on a prototyping board to test the components and the circuit itself. After utilizing the firmware, the programmer was verified by the interface program called “BumbleBee” that can be used on Microsoft Windows XP Operating System. After uploading and running a test C program that turns on a few LEDs on the target board, the wisp programmer board and target board were then converted into one board soldered on a veroboard as seen in Figure 20.



**Figure 20** PIC16F84A + PIC16F877A target board

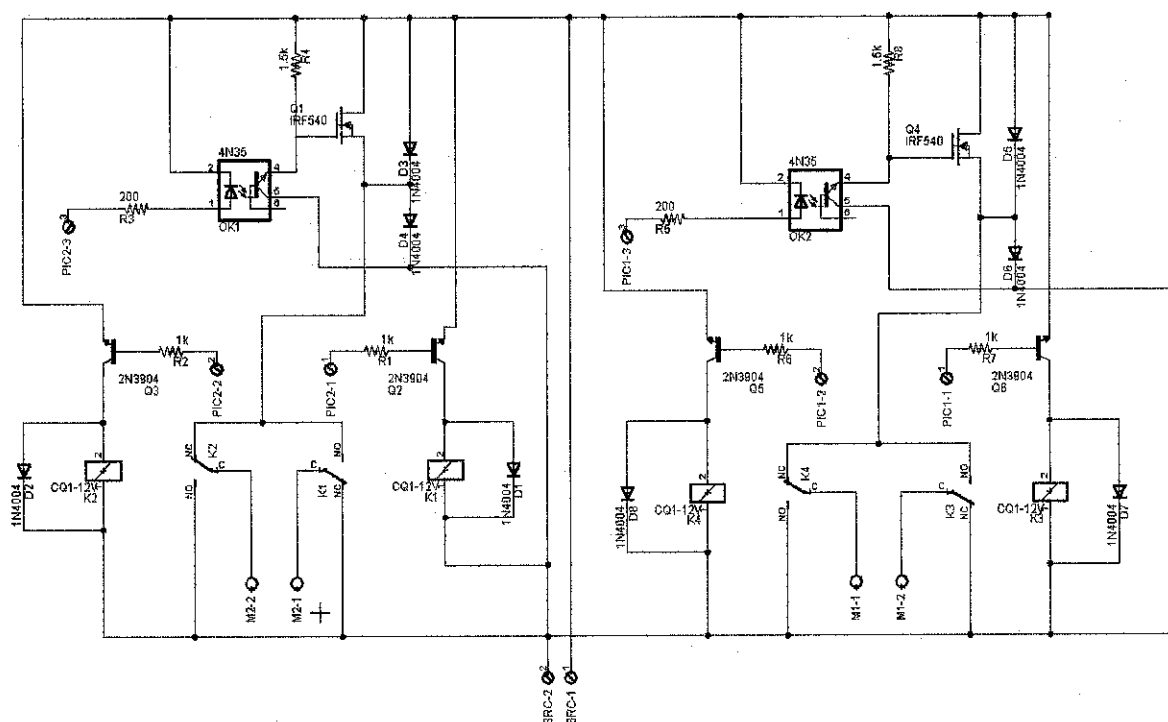
The target board connection is simulated through PSPICE with the microcontroller interfacing with the I/O sensors and DC Motor Controllers. A few sequencing LEDs were also included to enable program structure flow to be verified. The circuit design is as follows:



**Figure 21** EAGLE schematic for AGR Target Board

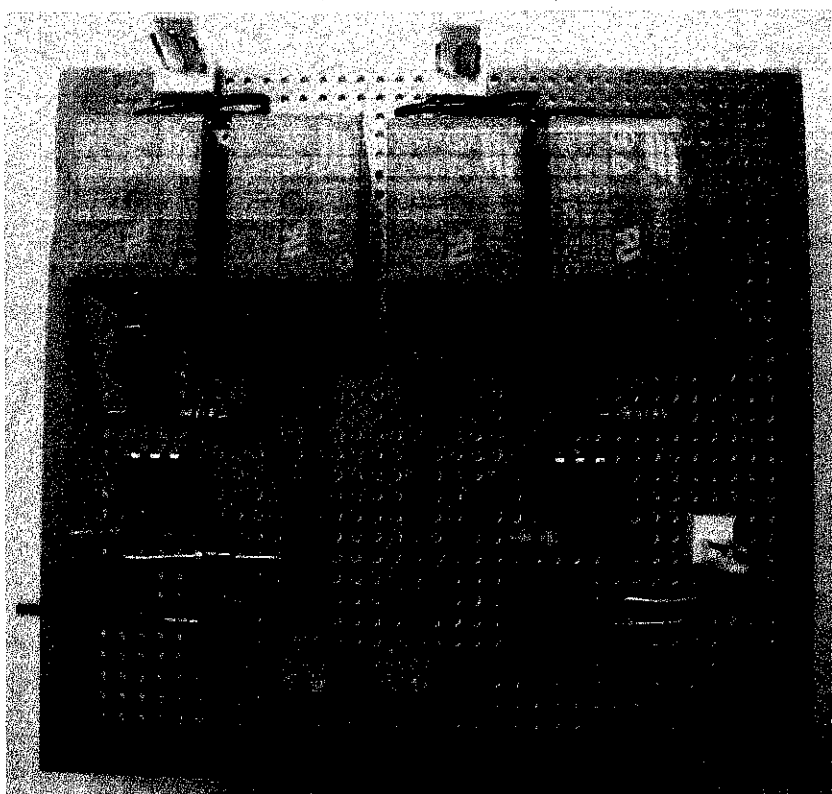
#### 4.2.2 Relay controlled motor driver

Design and analysis of the relay controlled motor driver was done with PSPICE and EAGLE. ERC checks were done on the schematic diagram done in EAGLE as seen in Figure 22. After analysis was done, the motor driver components were purchased according to the list stated in Chapter 2. The circuit was constructed and tested on breadboard before finalization.



**Figure 22** EAGLE Schematic for Motor driver

The resulting circuit constructed on veroboard is seen in Figure 14. The layout has 2 two-pin connectors for motor and power supply (12V) and a three-pin to the PIC. The motor speed control is done by the optocoupler which receives PWM signals from the PIC and varies the frequency of the output voltage to the motor accordingly.



**Figure 23** Constructed circuit on veroboard



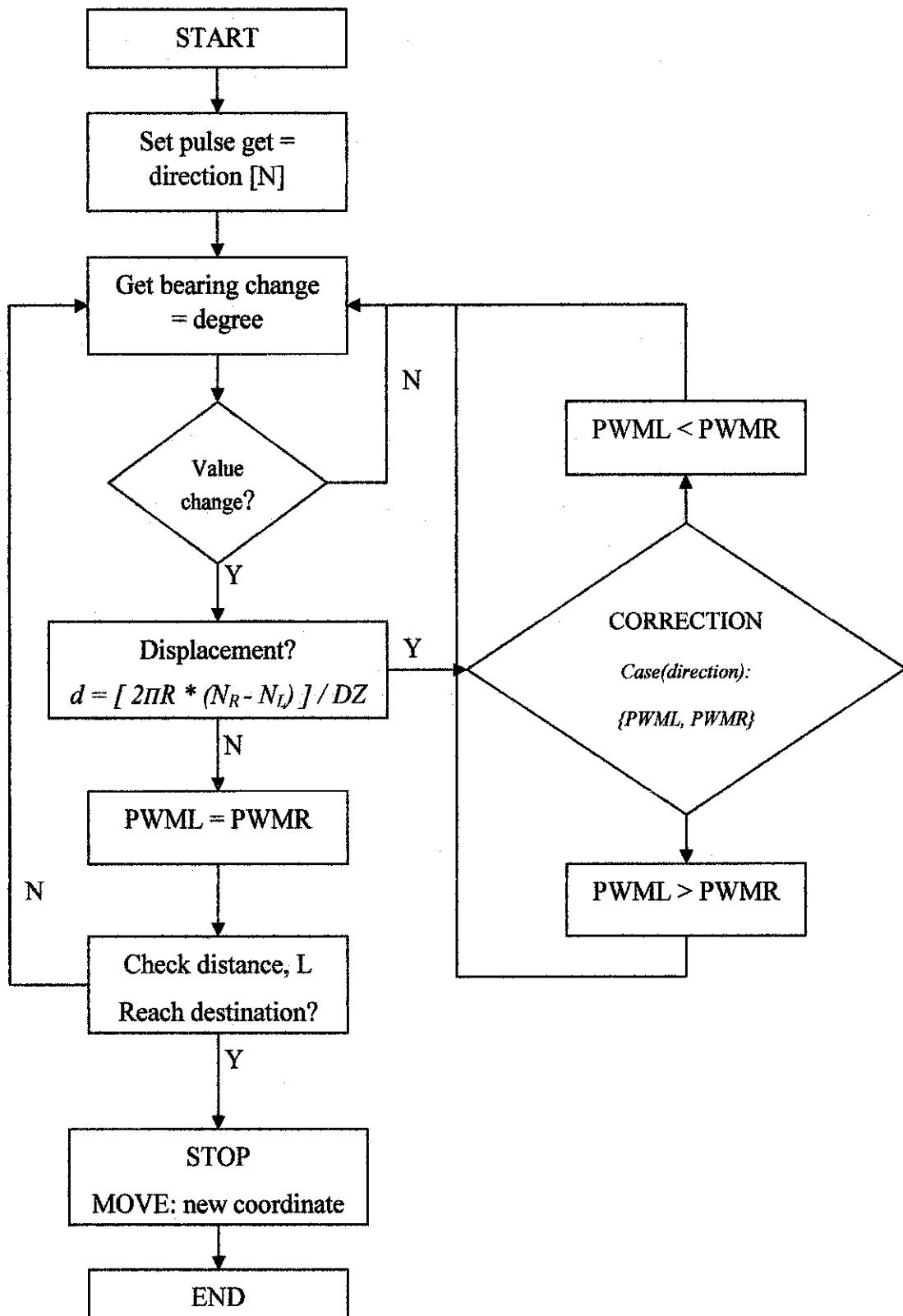
### **4.3 Algorithm Building and Coding Generation**

There are several algorithms integrated in the path planning of a mobile robot. Few such algorithms that are included in this project include the obstacle avoidance and global positioning algorithms. The algorithms are based upon process flow diagrams that help clarify the sequence of the algorithm looping for the robot.

From the generated algorithms, the coding language C is generated to be used for conversion to machine language (Hex) to be programmed into the microcontroller. These algorithms are explained in this section as follows.

#### ***4.3.1 Algorithm for differential drive movement control***

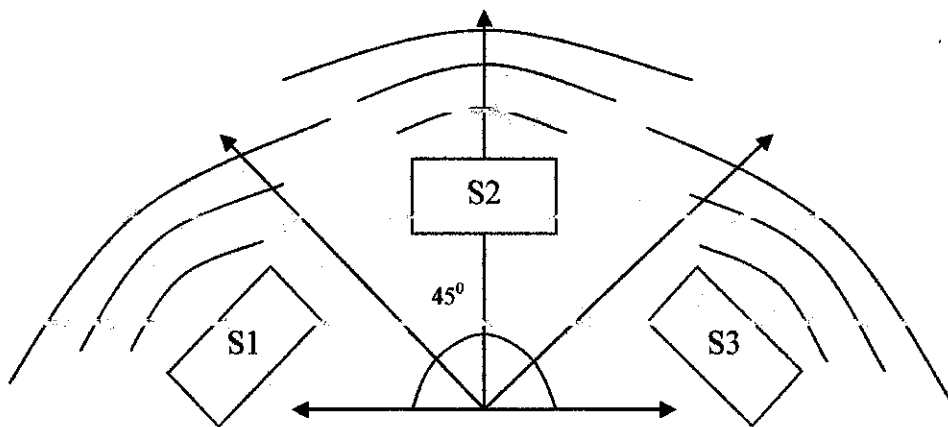
The speed and movement control of the DC brushed motors will be done by PWM signal from PIC as stated in the Chapter 2 of this report. The basic C program created for the use of PWM signaling is shown in Appendix E. Using the mathematical formula and theorems stated in Chapter 2; the author has generated an algorithm for differential drive as shown below:



**Figure 24** Algorithm for Differential drive control

### 4.3.2 Algorithm for obstacle avoidance

The obstacle avoidance algorithm basically functions to help the robot avoid reaching a dead end and resulting in collision. This integrated intelligence functions separately from the differential drive and global positioning algorithms as a single network but integrates as a system of networks for the robot in path planning. The important components that will be used in the obstacle avoidance are the ultrasonic sensors SRF04. 3 sensors will be mounted on the robot as follows:

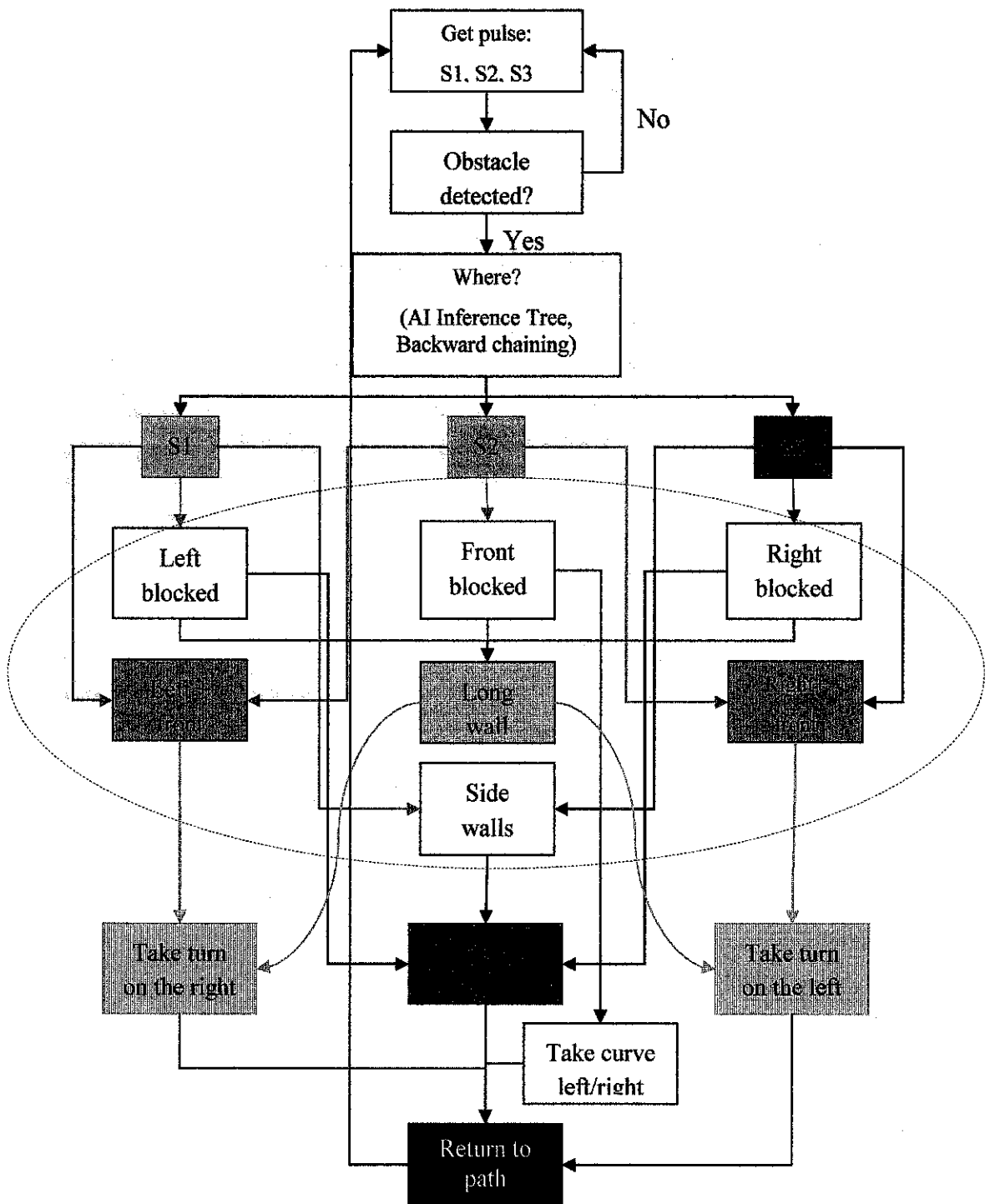


**Figure 25** Ultrasonic sensor placement

As seen in Figure 25, the sensors are placed  $45^{\circ}$  angle from the front sensors to allow  $180^{\circ}$  sensorable area. The sonar beam from the ultrasonic sensor has a  $\pm 38^{\circ}$  wide area, thus resulting in the total  $180^{\circ}$  if placed in such a way. The advantage of this placement is that the robot is fully alert of the surrounding area in front, and will be able to sense obstacles and walls from the front and the sides of its path. This will help the robot to determine which side to maneuver its turn to in cases where a wall is present on any of its sides. The predetermined action for each case is shown in Figure 18 below. The generated C coding for generating feedback signal from the ultrasonic sensors are shown in Appendix E.

When avoiding an obstacle, the robot must be able to turn away from the specified path and turn back into the path when assumed the obstacle is not present any longer. In most general cases, the robot will turn a full  $90^{\circ}$  to move to the x-axis and another  $90^{\circ}$  to return to the y-axis. Then repeat this step to return to the path. This will take up tremendous time and there is a possibility of the dead-reckoning to have high error

reading. The maneuvering of the AGR in this project will only require an angular arc which will be enough to avoid the obstacle and in the case of a dead end or junction, only then will the robot change direction of x and y-axis. Reminding that the placement of sensors plays an important part in this maneuver, the robot can differentiate between a single obstacle and a long wall. The generated algorithm bubble is shown below:



**Figure 26** Algorithm bubble for obstacle avoidance

## 4.4 Troubleshooting and Testing

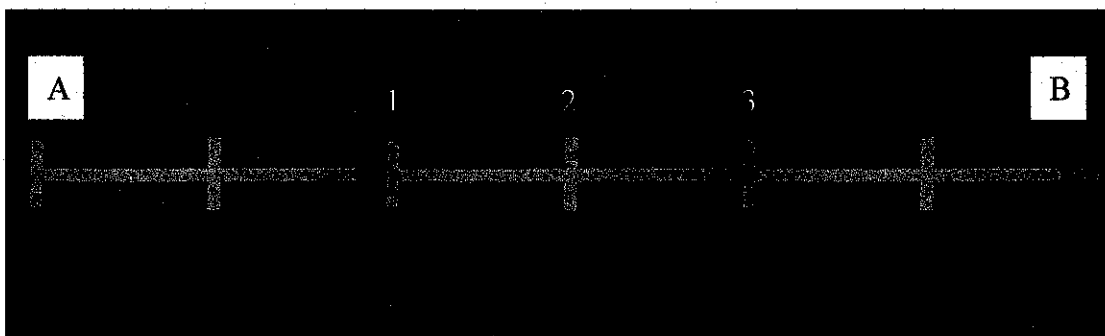
The establishment of a working prototype would not be complete without proper troubleshooting and repeated test or experiments carried out. Accuracy and repeatability is the main criteria in this project and fine-tuning of each variable control and feedback sensor must be carried out to achieve objectives. The author has divided the testing into 3 phases which are:

- Mobility Test
- Obstacle Avoidance Test
- Path Planning Test

Fine-tuning is done through the C language programming codes. The overall body structure is tested first to ensure errors do not arise from the motor alignments or body weight.

### 4.4.1 Mobility Test

Mobility test is categorized as the physical test of the AGR's ability to control speed, turn and move forward. A test grid was created for this test which is shown in figure below:



**Figure 27** Mobility test grid

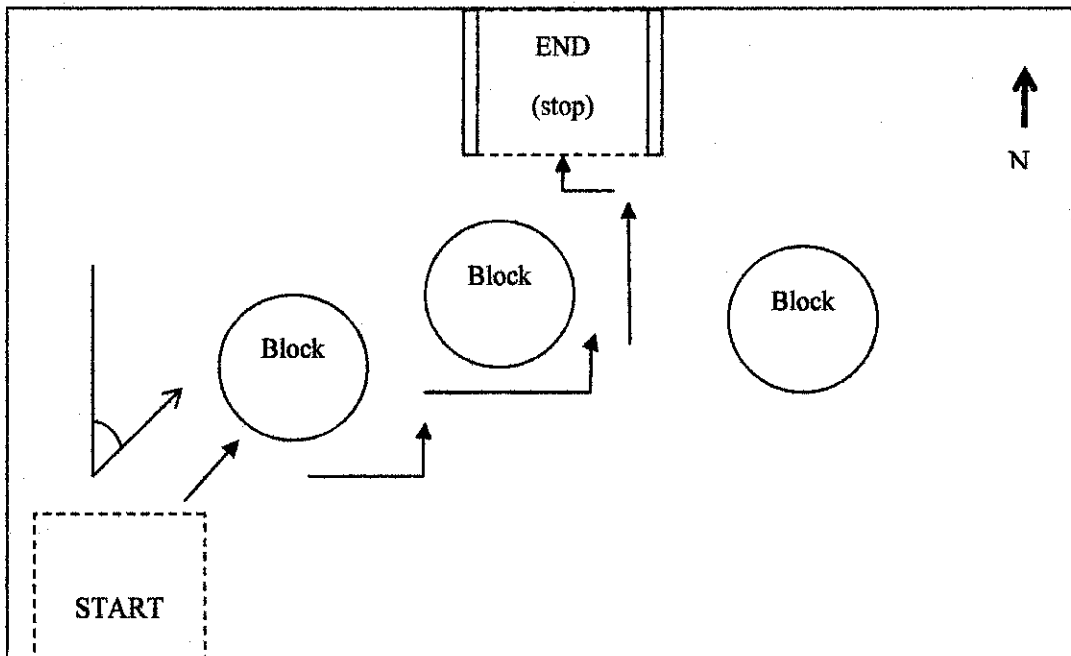
Turning test are done for left and right turning, and the initiative is to achieve a full 90° turn ratio before returning to forward movement. The main troubleshooting and modification done in this process is the bearing change reading and how the controller manipulates the data for the turning process. A few error readings which would occur could not be troubleshoot and thus repeatability of the AGR has been reduced to maximum 4 successive test runs. Using the same test grid (Figure 27), forward movement for the AGR is calibrated. The AGR is driven by 2 DC motors at

the back, thus a difference in speed or torque in these motors could lead to misalignment. If the AGR is not able to follow the white line from point A to point B without the use of sensors, then adjustment would be done to the speed of each motor according to the displacement. Optional movement including avoidance curve is also tested and troubleshoot, with the consideration of a  $180^\circ$  ideal curvature. If the robot is able to displace at waypoint 1 within the grid, missing waypoint 2 and re-entering the AB line at waypoint 3 then it is considered as a curvature movement. This move is considered as the hardest to troubleshoot since the movement is without feedback and only 1:3 ratios of successive runs had been carried out.

#### ***4.4.2 Obstacle Avoidance and Path Planning Test***

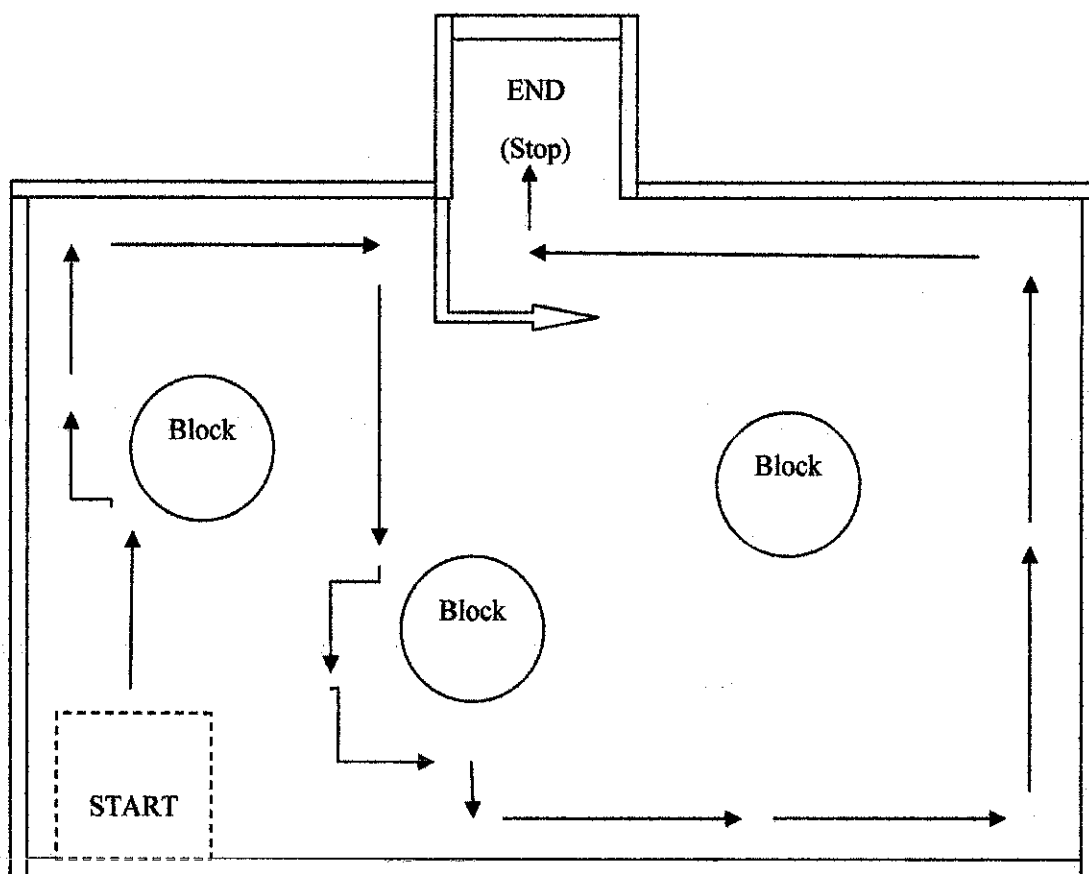
The AGR is able to sense two condition changes when moving which are the obstacles surrounding it and the North pole bearing change. Using the 3 ultrasonic sensors, the AGR basically has a  $180^\circ$  view of its area in front and will be able to avoid any obstacles facing it. There are two alternative ways in planning its next movement to a proposed end waypoint, which is either using a predetermined path scheduled by exactly informing the AGR of the obstacles it will face, or by using the compass bearing of the waypoint.

From its starting point, the distance from its end point could be determined as a bearing of the North. As seen in Figure 28, it is possible to move from the start point using the North bearing as a calibrating feedback to the controlled movement of the AGR. The AGR will move according to this bearing until it reaches the end point whilst avoiding any obstacles using the ultrasonic sensors as a secondary movement. When the AGR turns to avoid an obstacle, it will return to the North bearing after avoiding the obstacle. With this method, it is unnecessary to give a predetermined path to the controller and this will reduce the amount of error in movement in the case there are more obstacles present. The downside of this method is that the North bearing can be changed if there is a magnetic presence anywhere surrounding the AGR. As stated before, the electronic compass will detect the strongest magnetic frequency present as the North Pole. This result in error reading, thus the AGR may not be able to reach its end point.



**Figure 28** possible path taken by the AGR in an undetermined space

As a second alternative, random approaches are possible using the obstacle avoidance algorithm. As the AGR moves forward, it is constantly detecting any obstacles in front of it. An algorithm is developed to actually allow the AGR to move according to the arrows shown in figure 29 towards the end waypoint. The movement actually follows the limitation of space of the predetermined area, and it will follow the wall of which may lead to the end waypoint. When obstacles are detected, the AGR will avoid accordingly. Any amount of obstacles are possible, thus random obstacles can be placed in this situation. The AGR will continue moving until it reaches the end waypoint, which achieves the objective of this project. The downside is that the amount of time taken for each trial may vary, and may also be very long periods. The AGR may strafe away from the movement example shown and take a longer more winding route to the end point. This problem could have been overcome with the implementation of GA which gives the AGR the ability to 'learn' the paths it takes and carry out the best possible solution there is. The author was not able to implement AI in this project since the span of time allocated for the project was insufficient and thus the experimentation with GA was not carried out.



**Figure 29** possible path taken by the AGR in a predetermined space

The conclusive approach after testing has been made is using the North bearing as the primary feedback for path planning. Consecutive test results show that less error is made using this method and movement from the START point to the END point takes a shorter period of time. As a reminder, the AGR does not have the ability to memorize routes because it does not have enough memory and no additional external memory utilized in the system. This is due to insufficient time and funding for the project, thus the feasible objective of this project does not include memorizing routes in the control system of the robot.



## **CHAPTER 5**

### **CONCLUSION & RECOMMENDATIONS**

The author has achieved the objectives of the project with the completion of a fully working prototype by the end of the project period. The author has developed 3D and MATLAB simulations to predict the movement of the robot in certain conditions. Through these simulations and 3D models, the AGR was constructed and configured to carry out tasks according to the objectives of this project.

The prototype is structurally ready and troubleshooting process was carried out to fulfill the desired condition. The AI of the AGR dependant on the coding of the controller has been tested and verified to be able to carry out the tasks of obstacle avoidance and path planning. The AGR is able to move in a random behavior from one point and end at a final designated point using limited sensors without colliding with randomized objects within the path.

With the completion of the project, several recommendations for improvement of this project have been identified. The AGR control system has the tendency to reset or receive error spikes from the feedback sensors, thus isolation and protection circuits should be included to avoid such occurrence. The main and motor controller circuits should always be isolated and clean power supply voltage should be supplied using these protection circuits, thus ensuring safe and trouble-free functionality. In application of outer-space autonomous exploration, this would be a necessity as no troubleshooting or parts replacement could be carried out if a failure occurs.

A recommendation of upgrade for the control system is the inclusion of extended memory architecture using RAM or ROM IC to operate in memorizing application such as path waypoint planting. The global positioning could be further expanded to include this algorithm which would enable the AGR to evaluate and perform path evaluation and also use the saved waypoints to return to its starting point.

## REFERENCES

- [1] A. Fujimori, Peter N. Nikiforuk, and Madan M. Gupta (1997), "Adaptive Navigation of Mobile Robots with Obstacle Avoidance", IEEE Transactions On Robotics And Automation, Vol. 13.
- [2] Lance D.Chambers (1999). Genetic Algorithm .*Complex Coding Systems*. Volume 3: CRC Press.
- [3] Maxim A. Batalin, Gaurav S. Sukhatme and Myron Hattig, "Mobile Robot Navigation using a Sensor Network", IEEE International Conference on Robotics and Automation.
- [4] M. Qingchun, S. Honglian (1998), "Intelligent Control Based on Genetic Algorithms, Case Study on Mobile Robots", IEEE, Conference on Intelligent Systems.
- [5] K. Ohno, T. Tsubouchit , B. Shigematsut (2003), "Outdoor Navigation of a Mobile Robot between Buildings based on GPS and Odometry Data Fusion", 2003 IEEE ICRA.
- [6] Jahanzeb Rajput, "Differential Drive & Global Positioning blks User's Guide", University of Engineering and Technology, Lahore, Pakistan.
- [7] HobbyProjects, Electronics circuits & tutorials,  
[http://www.hobbyprojects.com/quick\\_circuits\\_reference/motor\\_and\\_general\\_control\\_schematics.html](http://www.hobbyprojects.com/quick_circuits_reference/motor_and_general_control_schematics.html)
- [8] Technology Comparison Engineering Reference, Catalog 8000-3/USA, Parker Hannifin Corporation, Compumotor Division, California.

## **APPENDICES**

**APPENDIX A**  
**MICROCONTROLLER DATASHEET**

**Microcontroller PIC16F877A Datasheet**



---

**PIC16F87XA**  
**Data Sheet**

28/40/44-Pin Enhanced Flash  
Microcontrollers



# PIC16F87XA

## 28/40/44-Pin Enhanced Flash Microcontrollers

### Devices Included in this Data Sheet:

- PIC16F873A
- PIC16F874A
- PIC16F877A
- 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 6K x 14 words of Flash Program Memory.  
Up to 388 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 I²C™ (Master/Slave)
- Universal Synchronous Asynchronous Receiver Transmitter (USART/SCI) with 9-bit address detection
- Parallel Slave Port (PSP) – 8 bits wide with external RD, WR and CS controls (40/44-pin only)
- Brown-out detection circuitry for Brown-out Reset (BOR)

### Analog Features:

- 10-bit, up to 8-channel Analog-to-Digital Converter (A/D)
- Brown-out Reset (BOR)
- Analog Comparator module with:
  - Two analog comparators
  - Programmable on-chip voltage reference (VREF) module
  - Programmable input multiplexing from device inputs and internal voltage reference
  - Comparator outputs are externally accessible

### Special Microcontroller Features:

- 100,000 erase/write cycle Enhanced Flash program memory typical
- 1,000,000 erase/write cycle Data EEPROM memory typical
- Data EEPROM Retention > 40 years
- Self-reprogrammable under software control
- In-Circuit Serial Programming™ (ICSP™) via two pins
- Single-supply 5V In-Circuit Serial Programming
- Watchdog Timer (WDT) with its own on-chip RC oscillator for reliable operation
- Programmable code protection
- Power saving Sleep mode
- Selectable oscillator options
- In-Circuit Debug (ICD) via two pins

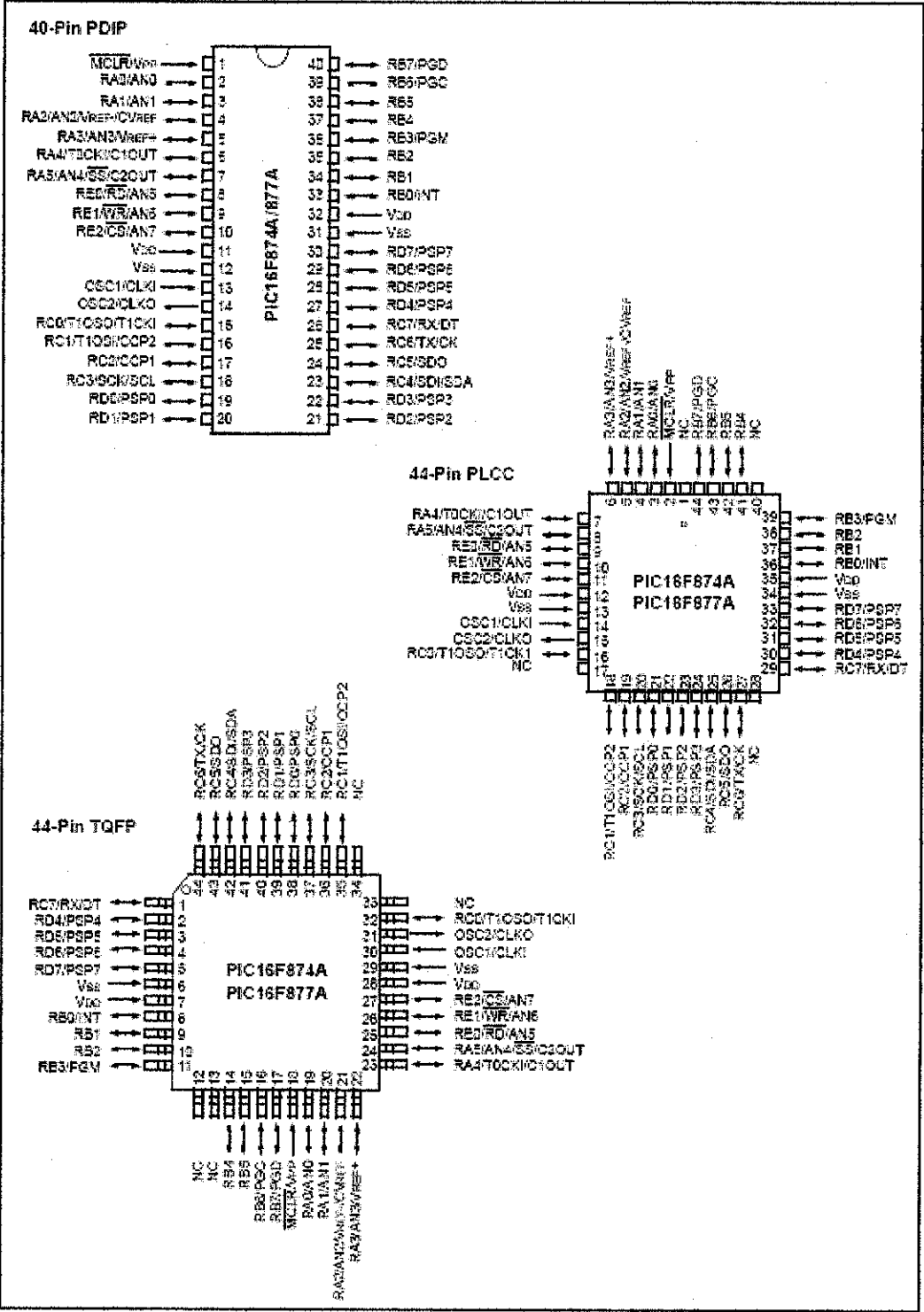
### CMOS Technology:

- Low-power, high-speed Flash/EEPROM technology
- Fully static design
- Wide operating voltage range (2.0V to 5.5V)
- Commercial and Industrial temperature ranges
- Low-power consumption

Device	Program Memory		Data SRAM (Bytes)	EEPROM (Bytes)	I/O	10-bit A/D (ch)	CCP (PWM)	MSSP		USART	Timers 8/16-bit	Comparators
	Bytes	# Single Word Instructions						SPI	Master PC			
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	6	2	Yes	Yes	Yes	2/1	2
PIC16F877A	14.3K	8192	368	256	33	8	2	Yes	Yes	Yes	2/1	2

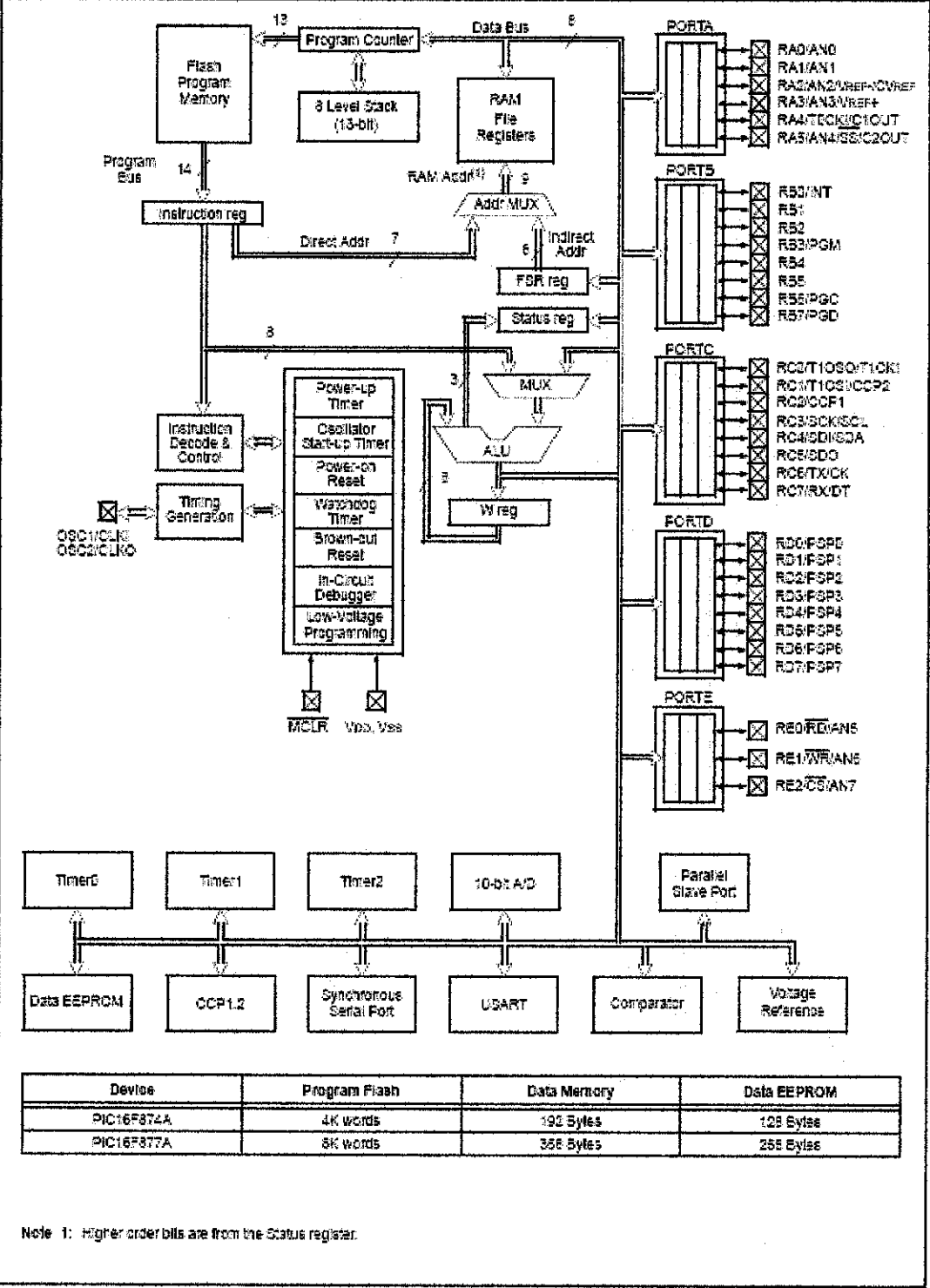
PIC16F87XA

Pin Diagrams (Continued)



PIC16F87XA

FIGURE 1-2: PIC16F874A/877A BLOCK DIAGRAM



# PIC16F87XA

TABLE 1-2: PIC16F873A/876A PINOUT DESCRIPTION

Pin Name	PDIP, SOIC, SSOP Pin#	QFN Pin#	I/O/P Type	Buffer Type	Description
OSC1/CLKI OSC1 CLKI	9	8	I  I	ST/CMOS <sup>(3)</sup>	Oscillator crystal or external clock input. Oscillator crystal input or external clock source input. ST buffer when configured in RC mode; otherwise CMOS. External clock source input. Always associated with pin function OSC1 (see OSC1/CLKI, OSC2/CLKO pins).
OSC2/CLKO OSC2 CLKO	10	7	O  O	—	Oscillator crystal or clock output. Oscillator crystal output. Connects to crystal or resonator in Crystal Oscillator mode. In RC mode, OSC2 pin outputs CLKO, which has 1/4 the frequency of OSC1 and denotes the instruction cycle rate.
MCLR/VPP MCLR VPP	1	28	I  P	ST	Master Clear (input) or programming voltage (output). Master Clear (Reset) input. This pin is an active low Reset to the device. Programming voltage input.
RA0/AN0 RA0 AN0 RA1/AN1 RA1 AN1 RA2/AN2/VREF- CVREF RA2 AN2 VREF- CVREF RA3/AN3/VREF+ RA3 AN3 VREF+ RA4/T0CKI/C1OUT RA4 T0CKI C1OUT RA5/AN4/SS/C2OUT RA5 AN4 SS C2OUT	2   3   4   5   6   7	27   26   1   2   3   4	I/O I  I/O I  I/O I  I/O I O  I/O I O	TTL   TTL   TTL   TTL   ST   TTL	PORTA is a bidirectional I/O port.  Digital I/O. Analog input 0.  Digital I/O. Analog input 1.  Digital I/O. Analog input 2. A/D reference voltage (Low) input. Comparator VREF output.  Digital I/O. Analog input 3. A/D reference voltage (High) input.  Digital I/O – Open-drain when configured as output. Timer0 external clock input. Comparator 1 output.  Digital I/O. Analog input 4. SPI slave select input. 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.



# PIC16F87XA

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

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

# PIC16F87XA

TABLE 1-3: PIC16F874A/877A PINOUT DESCRIPTION

Pin Name	PDIP Pin#	PLCC Pin#	TQFP Pin#	QFN Pin#	I/O/P Type	Buffer Type	Description
OSC1/CLKI OSC1  CLKI	13	14	20	32	I  I	ST/CMOS <sup>(1)</sup>	Oscillator crystal or external clock input. Oscillator crystal input or external clock source input. ST buffer when configured in RC mode; otherwise CMOS. External clock source input. Always associated with pin function OSC1 (see OSC1/CLKI, OSC2/CLKO pins).
OSC2/CLKO OSC2  CLKO	14	15	31	33	O  O	—	Oscillator crystal or clock output. Oscillator crystal output. Connects to crystal or resonator in Crystal Oscillator mode. In RC mode, OSC2 pin outputs CLKO, which has 1/4 the frequency of OSC1 and denotes the instruction cycle rate.
MCLR/VPP MCLR  VPP	1	2	18	18	I  P	ST	Master Clear (input) or programming voltage (output). Master Clear (Reset) input. This pin is an active low Reset to the device. Programming voltage input.
RA0/AN0 RA0 AN0	2	3	19	19	I/O I	TTL	PORTA is a bidirectional I/O port.  Digital I/O. Analog input 0.
RA1/AN1 RA1 AN1	3	4	20	20	I/O I	TTL	
RA2/AN2/VREF-/CVREF RA2 AN2 VREF- CVREF	4	5	21	21	I/O I I O	TTL	
RA3/AN3/VREF+ RA3 AN3 VREF+	5	6	22	22	I/O I I	TTL	
RA4/T0CKI/C1OUT RA4  T0CKI C1OUT	6	7	23	23	I/O  I O	ST	
RA5/AN4/SS/C2OUT RA5 AN4 SS C2OUT	7	8	24	24	I/O I I O	TTL	

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.

# PIC16F87XA

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
RB0/INT RB0 INT	33	36	2	9	I/O I	TTL/ST <sup>(1)</sup>	PORTB is a bidirectional I/O port. PORTB can be software programmed for internal weak pull-up on all inputs.  Digital I/O. 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 RB3 PGM	36	39	11	12	I/O I	TTL	Digital I/O. 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 RB6 PGC	39	43	16	16	I/O I	TTL/ST <sup>(2)</sup>	Digital I/O. In-circuit debugger and ICSP programming clock.
RB7/PGD RB7 PGD	40	44	17	17	I/O I/O	TTL/ST <sup>(2)</sup>	Digital 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.

# PIC16F87XA

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
RC0/T1OSC/T1CKI RC0 T1OSO T1CKI	15	16	32	34	I/O O I	ST	PORTC is a bidirectional I/O port.  Digital I/O. Timer1 oscillator output. Timer1 external clock input.
RC1/T1OSI/CCP2 RC1 T1OSI CCP2	16	16	35	35	I/O I I/O	ST	
RC2/CCP1 RC2 CCP1	17	19	36	38	I/O I/O	ST	Digital I/O. Capture2 input, Compare2 output, PWM2 output.
RC3/SCK/SCL RC3 SCK SCL	18	20	37	37	I/O I/O I/O	ST	Digital I/O. Synchronous serial clock input/output for SPI mode. Synchronous serial clock input/output for I <sup>2</sup> C mode.
RC4/SDI/SDA RC4 SDI SDA	23	25	42	42	I/O I I/O	ST	Digital I/O. SPI data in. I <sup>2</sup> C data I/O.
RC6/SDO RC6 SDO	24	26	43	43	I/O O	ST	Digital I/O. SPI data out.
RC8/TX/CK RC8 TX CK	25	27	44	44	I/O O I/O	ST	Digital I/O. USART asynchronous transmit. USART1 synchronous clock.
RC7/RX/DT RC7 RX DT	26	29	1	1	I/O I I/O	ST	Digital I/O. USART asynchronous receive. USART synchronous data.

Legend: I = input      O = output      IO = 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.

# PIC16F87XA

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
RD0/PSP0 RD0 PSP0	19	21	38	38	I/O I/O	ST/TTL <sup>(3)</sup>	PORTD is a bidirectional I/O port or Parallel Slave Port when interfacing to a microprocessor bus.  Digital I/O. Parallel Slave Port data.
RD1/PSP1 RD1 PSP1	20	22	39	39	I/O I/O	ST/TTL <sup>(3)</sup>	
RD2/PSP2 RD2 PSP2	21	23	40	40	I/O I/O	ST/TTL <sup>(3)</sup>	
RD3/PSP3 RD3 PSP3	22	24	41	41	I/O I/O	ST/TTL <sup>(3)</sup>	
RD4/PSP4 RD4 PSP4	27	30	2	2	I/O I/O	ST/TTL <sup>(3)</sup>	
RD5/PSP5 RD5 PSP5	28	31	3	3	I/O I/O	ST/TTL <sup>(3)</sup>	
RD6/PSP6 RD6 PSP6	29	32	4	4	I/O I/O	ST/TTL <sup>(3)</sup>	
RD7/PSP7 RD7 PSP7	30	33	5	5	I/O I/O	ST/TTL <sup>(3)</sup>	
RE0/RD/AN5 RE0 RD AN5	8	9	25	25	I/O I I	ST/TTL <sup>(3)</sup>	PORTE is a bidirectional I/O port.  Digital I/O. Read control for Parallel Slave Port. Analog input 5.
RE1/WR/AN6 RE1 WR AN6	9	10	26	26	I/O I I	ST/TTL <sup>(3)</sup>	
RE2/CS/AN7 RE2 CS AN7	10	11	27	27	I/O I I	ST/TTL <sup>(3)</sup>	
VSS	12, 31	13, 34	6, 29	6, 30, 31	P	—	Ground reference for logic and I/O pins.
VDD	11, 32	12, 35	7, 28	7, 8, 28, 29	P	—	Positive supply for logic and I/O pins.
NC	—	1, 17, 28, 40	12, 13, 33, 34	13	—	—	These pins are not internally connected. These pins should be left unconnected.

Legend: I = input    O = output    IO = 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.

## 8.0 CAPTURE/COMPARE/PWM MODULES

Each Capture/Compare/PWM (CCP) module contains a 16-bit register which can operate as a:

- 16-bit Capture register
- 16-bit Compare register
- PWM Master/Slave Duty Cycle register

Both the CCP1 and CCP2 modules are identical in operation, with the exception being the operation of the special event trigger. Table 8-1 and Table 8-2 show the resources and interactions of the CCP module(s). In the following sections, the operation of a CCP module is described with respect to CCP1. CCP2 operates the same as CCP1 except where noted.

### CCP1 Module:

Capture/Compare/PWM Register 1 (CCPR1) is comprised of two 8-bit registers: CCPR1L (low byte) and CCPR1H (high byte). The CCP1CON register controls the operation of CCP1. The special event trigger is generated by a compare match and will reset Timer1.

### CCP2 Module:

Capture/Compare/PWM Register 2 (CCPR2) is comprised of two 8-bit registers: CCPR2L (low byte) and CCPR2H (high byte). The CCP2CON register controls the operation of CCP2. The special event trigger is generated by a compare match and will reset Timer1 and start an A/D conversion (if the A/D module is enabled).

Additional information on CCP modules is available in the PICmicro<sup>®</sup> Mid-Range MCU Family Reference Manual (DS33023) and in application note AN594, "Using the CCP Module(s)" (DS00594).

TABLE 8-1: CCP MODE – TIMER RESOURCES REQUIRED

CCP Mode	Timer Resource
Capture	Timer1
Compare	Timer1
PWM	Timer2

TABLE 8-2: INTERACTION OF TWO CCP MODULES

CCPx Mode	CCPy Mode	Interaction
Capture	Capture	Same TMR1 time base
Capture	Compare	The compare should be configured for the special event trigger which clears TMR1
Compare	Compare	The compare(s) should be configured for the special event trigger which clears TMR1
PWM	PWM	The PWMs will have the same frequency and update rate (TMR2 interrupt)
PWM	Capture	None
PWM	Compare	None

## 8.3 PWM Mode (PWM)

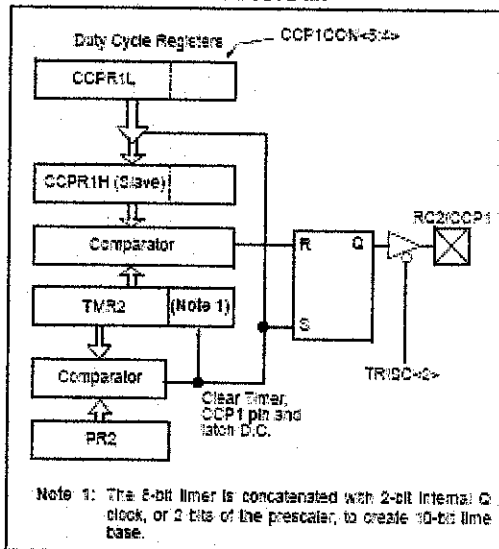
In Pulse Width Modulation mode, the CCPx pin produces up to a 10-bit resolution PWM output. Since the CCP1 pin is multiplexed with the PORTC data latch, the TRISC<2> bit must be cleared to make the CCP1 pin an output.

**Note:** Clearing the CCP1CON register will force the CCP1 PWM output latch to the default low level. This is not the PORTC I/O data latch.

Figure 8-3 shows a simplified block diagram of the CCP module in PWM mode.

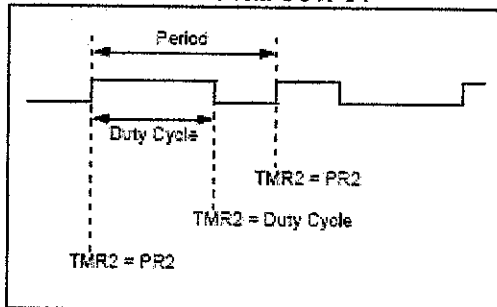
For a step-by-step procedure on how to set up the CCP module for PWM operation, see Section 8.3.3 "Setup for PWM Operation".

**FIGURE 8-3: SIMPLIFIED PWM BLOCK DIAGRAM**



A PWM output (Figure 8-4) has a time base (period) and a time that the output stays high (duty cycle). The frequency of the PWM is the inverse of the period (1/period).

**FIGURE 8-4: PWM OUTPUT**



### 8.3.1 PWM PERIOD

The PWM period is specified by writing to the PR2 register. The PWM period can be calculated using the following formula:

$$\text{PWM Period} = [(PR2) + 1] \cdot 4 \cdot T_{OSC} \cdot (\text{TMR2 Prescale Value})$$

PWM frequency is defined as 1/[PWM period].

When TMR2 is equal to PR2, the following three events occur on the next increment cycle:

- TMR2 is cleared
- The CCP1 pin is set (exception: if PWM duty cycle = 0%, the CCP1 pin will not be set)
- The PWM duty cycle is latched from CCPR1L into CCPR1H

**Note:** The Timer2 postscaler (see Section 7.1 "Timer2 Prescaler and Postscaler") is not used in the determination of the PWM frequency. The postscaler could be used to have a servo update rate at a different frequency than the PWM output.

### 8.3.2 PWM DUTY CYCLE

The PWM duty cycle is specified by writing to the CCPR1L register and to the CCP1CON<5:4> bits. Up to 10-bit resolution is available. The CCPR1L contains the eight MSBs and the CCP1CON<5:4> contains the two LSBs. This 10-bit value is represented by CCPR1L:CCP1CON<5:4>. The following equation is used to calculate the PWM duty cycle in time:

$$\text{PWM Duty Cycle} = (\text{CCPR1L:CCP1CON<5:4>}) \cdot T_{OSC} \cdot (\text{TMR2 Prescale Value})$$

CCPR1L and CCP1CON<5:4> can be written to at any time, but the duty cycle value is not latched into CCPR1H until after a match between PR2 and TMR2 occurs (i.e., the period is complete). In PWM mode, CCPR1H is a read-only register.

The CCPR1H register and a 2-bit internal latch are used to double-buffer the PWM duty cycle. This double-buffering is essential for glitch-free PWM operation.

When the CCPR1H and 2-bit latch match TMR2, concatenated with an internal 2-bit Q clock or 2 bits of the TMR2 prescaler, the CCP1 pin is cleared.

The maximum PWM resolution (bits) for a given PWM frequency is given by the following formula.

**EQUATION 8-1:**

$$\text{Resolution} = \frac{\log\left(\frac{F_{OSC}}{F_{PWM}}\right)}{\log(2)} \text{ bits}$$

**Note:** If the PWM duty cycle value is longer than the PWM period, the CCP1 pin will not be cleared.

# PIC16F87XA

## 8.3.3 SETUP FOR PWM OPERATION

The following steps should be taken when configuring the CCP module for PWM operation:

1. Set the PWM period by writing to the PR2 register.
2. Set the PWM duty cycle by writing to the CCPR1L register and CCP1CON<5:4> bits.
3. Make the CCP1 pin an output by clearing the TRISC<2> bit.
4. Set the TMR2 prescale value and enable Timer2 by writing to T2CON.
5. Configure the CCP1 module for PWM operation.

TABLE 8-3: EXAMPLE PWM FREQUENCIES AND RESOLUTIONS AT 20 MHz

PWM Frequency	1.22 kHz	4.88 kHz	19.53 kHz	78.12kHz	156.3 kHz	208.3 kHz
Timer Prescaler (1, 4, 16)	16	4	1	1	1	1
PR2 Value	0xFFh	0xFFh	0xFFh	0x3Fh	0x1Fh	0x17h
Maximum Resolution (bits)	10	10	10	8	7	5.5

TABLE 8-4: REGISTERS ASSOCIATED WITH CAPTURE, COMPARE AND TIMER1

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, 6Bh, 10Bh, 18Bh	INTCON	GIE	PEIE	TMRDIE	INTE	RBIE	TMR0IE	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF <sup>(1)</sup>	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
0Dh	PIR2	—	—	—	—	—	—	—	CCP2IF	---- --0	---- --0
8Ch	PIE1	PSPIE <sup>(1)</sup>	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
8Dh	PIE2	—	—	—	—	—	—	—	CCP2IE	---- --0	---- --0
87h	TRISC	PORTC Data Direction Register								1111 1111	1111 1111
0Eh	TMR1L	Holding Register for the Least Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	uuuu uuuu
0Fh	TMR1H	Holding Register for the Most Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	uuuu uuuu
10h	T1CON	—	—	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON	--00 0000	--uu uuuu
15h	CCPR1L	Capture/Compare/PWM Register 1 (LSB)								xxxx xxxx	uuuu uuuu
16h	CCPR1H	Capture/Compare/PWM Register 1 (MSB)								xxxx xxxx	uuuu uuuu
17h	CCP1CON	—	—	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0	--00 0000	--00 0000
1Bh	CCPR2L	Capture/Compare/PWM Register 2 (LSB)								xxxx xxxx	uuuu uuuu
1Ch	CCPR2H	Capture/Compare/PWM Register 2 (MSB)								xxxx xxxx	uuuu uuuu
1Dh	CCP2CON	—	—	CCP2X	CCP2Y	CCP2M3	CCP2M2	CCP2M1	CCP2M0	--00 0000	--00 0000

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by Capture and Timer1.

Note 1: The PSP is not implemented on 28-pin devices; always maintain these bits clear.



PIC16F87XA

TABLE 8-5: REGISTERS ASSOCIATED WITH PWM AND TIMER2

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, 2Bh, 10Bh, 12Bh	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	0000 000u
0Cn	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
0Dn	PIR2	—	—	—	—	—	—	—	CCP2IF	---- --0	---- --0
6Cn	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
6Dn	PIE2	—	—	—	—	—	—	—	CCP2IE	---- --0	---- --0
87h	TRISC	PORTC Data Direction Register								1111 1111	1111 1111
11h	TMR2	Timer2 Module's Register								0000 0000	0000 0000
92h	PR2	Timer2 Module's Period Register								1111 1111	1111 1111
12h	T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	-000 0000	-000 0000
15h	CCPR1L	Capture/Compare/PWM Register 1 (LSB)								xxxx xxxx	uuuu uuuu
16h	CCPR1H	Capture/Compare/PWM Register 1 (MSB)								xxxx xxxx	uuuu uuuu
17h	CCP1CON	—	—	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0	--00 0000	--00 0000
18h	CCPR2L	Capture/Compare/PWM Register 2 (LSB)								xxxx xxxx	uuuu uuuu
1Cn	CCPR2H	Capture/Compare/PWM Register 2 (MSB)								xxxx xxxx	uuuu uuuu
1Dn	CCP2CON	—	—	CCP2X	CCP2Y	CCP2M3	CCP2M2	CCP2M1	CCP2M0	--00 0000	--00 0000

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

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

# PIC16F87XA

## 9.4 I<sup>2</sup>C Mode

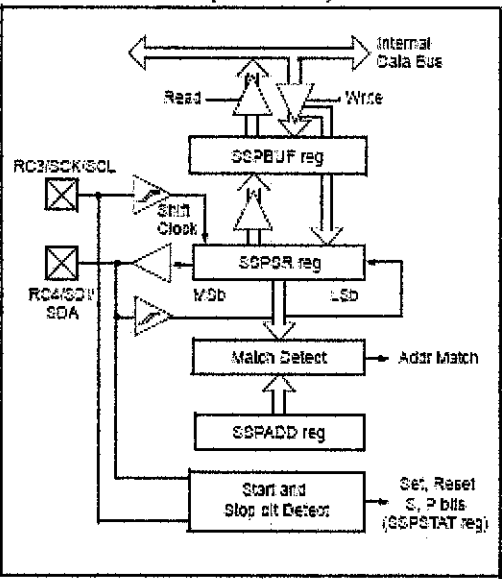
The MSSP module in I<sup>2</sup>C mode fully implements all master and slave functions (including general call support) and provides interrupts on Start and Stop bits in hardware to determine a free bus (multi-master function). The MSSP module implements the standard mode specifications, as well as 7-bit and 10-bit addressing.

Two pins are used for data transfer:

- Serial clock (SCL) – RC3/SCK/SCL
- Serial data (SDA) – RC4/SDI/SDA

The user must configure these pins as inputs or outputs through the TRISC<4:3> bits.

FIGURE 9-7: MSSP BLOCK DIAGRAM (I<sup>2</sup>C MODE)



## 9.4.1 REGISTERS

The MSSP module has six registers for I<sup>2</sup>C operation. These are:

- MSSP Control Register (SSPCON)
- MSSP Control Register 2 (SSPCON2)
- MSSP Status Register (SSPSTAT)
- Serial Receiver/Transmit Buffer Register (SSPBUF)
- MSSP Shift Register (SSPSR) – Not directly accessible
- MSSP Address Register (SSPADD)

SSPCON, SSPCON2 and SSPSTAT are the control and status registers in I<sup>2</sup>C mode operation. The SSPCON and SSPCON2 registers are readable and writable. The lower six bits of the SSPSTAT are read-only. The upper two bits of the SSPSTAT are read/write.

SSPSR is the shift register used for shifting data in or out. SSPBUF is the buffer register to which data bytes are written to or read from.

SSPADD register holds the slave device address when the SSP is configured in I<sup>2</sup>C Slave mode. When the SSP is configured in Master mode, the lower seven bits of SSPADD act as the baud rate generator reload value.

In receive operations, SSPSR and SSPBUF together create a double-buffered receiver. When SSPSR receives a complete byte, it is transferred to SSPBUF and the SSPIF interrupt is set.

During transmission, the SSPBUF is not double-buffered. A write to SSPBUF will write to both SSPBUF and SSPSR.

# PIC16F87XA

## 9.4.2 OPERATION

The MSSP module functions are enabled by setting MSSP Enable bit, SSPEN (SSPCON<5>).

The SSPCON register allows control of the I<sup>2</sup>C operation. Four mode selection bits (SSPCON<3:0>) allow one of the following I<sup>2</sup>C modes to be selected:

- I<sup>2</sup>C Master mode, clock = OSC/4 (SSPAD + 1)
- I<sup>2</sup>C Slave mode (7-bit address)
- I<sup>2</sup>C Slave mode (10-bit address)
- I<sup>2</sup>C Slave mode (7-bit address) with Start and Stop bit interrupts enabled
- I<sup>2</sup>C Slave mode (10-bit address) with Start and Stop bit interrupts enabled
- I<sup>2</sup>C Firmware Controlled Master mode, slave is Idle

Selection of any I<sup>2</sup>C mode, with the SSPEN bit set, forces the SCL and SDA pins to be open-drain, provided these pins are programmed to inputs by setting the appropriate TRISC bits. To ensure proper operation of the module, pull-up resistors must be provided externally to the SCL and SDA pins.

## 9.4.3 SLAVE MODE

In Slave mode, the SCL and SDA pins must be configured as inputs (TRISC<4:3> set). The MSSP module will override the input state with the output data when required (slave-transmitter).

The I<sup>2</sup>C Slave mode hardware will always generate an interrupt on an address match. Through the mode select bits, the user can also choose to interrupt on Start and Stop bits.

When an address is matched, or the data transfer after an address match is received, the hardware automatically will generate the Acknowledge (ACK) pulse and load the SSPBUF register with the received value currently in the SSPSR register.

Any combination of the following conditions will cause the MSSP module not to give this ACK pulse:

- The buffer full bit, BF (SSPSTAT<0>), was set before the transfer was received.
- The overflow bit, SSPOV (SSPCON<6>), was set before the transfer was received.

In this case, the SSPSR register value is not loaded into the SSPBUF, but bit SSPIF (PIR1<3>) is set. The BF bit is cleared by reading the SSPBUF register, while bit SSPOV is cleared through software.

The SCL clock input must have a minimum high and low for proper operation. The high and low times of the I<sup>2</sup>C specification, as well as the requirement of the MSSP module, are shown in timing parameter #100 and parameter #101.

### 9.4.3.1 Addressing

Once the MSSP module has been enabled, it waits for a Start condition to occur. Following the Start condition, the 8 bits are shifted into the SSPSR register. All incoming bits are sampled with the rising edge of the clock (SCL) line. The value of register SSPSR<7:1> is compared to the value of the SSPADD register. The address is compared on the falling edge of the eighth clock (SCL) pulse. If the addresses match, and the BF and SSPOV bits are clear, the following events occur:

1. The SSPSR register value is loaded into the SSPBUF register.
2. The Buffer Full bit, BF, is set.
3. An ACK pulse is generated.
4. MSSP Interrupt Flag bit, SSPIF (PIR1<3>), is set (interrupt is generated if enabled) on the falling edge of the ninth SCL pulse.

In 10-bit Address mode, two address bytes need to be received by the slave. The five Most Significant bits (MSbs) of the first address byte specify if this is a 10-bit address. Bit R/W (SSPSTAT<2>) must specify a write so the slave device will receive the second address byte. For a 10-bit address, the first byte would equal '11110 A9 A8 0', where 'A9' and 'A8' are the two MSbs of the address. The sequence of events for 10-bit address is as follows, with steps 7 through 9 for the slave-transmitter:

1. Receive first (high) byte of address (bits SSPIF, BF and bit UA (SSPSTAT<1>) are set).
2. Update the SSPADD register with second (low) byte of address (clears bit UA and releases the SCL line).
3. Read the SSPBUF register (clears bit BF) and clear flag bit SSPIF.
4. Receive second (low) byte of address (bits SSPIF, BF and UA are set).
5. Update the SSPADD register with the first (high) byte of address. If match releases SCL line, this will clear bit UA.
6. Read the SSPBUF register (clears bit BF) and clear flag bit SSPIF.
7. Receive Repeated Start condition.
8. Receive first (high) byte of address (bits SSPIF and BF are set).
9. Read the SSPBUF register (clears bit BF) and clear flag bit SSPIF.

## 9.4.3.2 Reception

When the  $\overline{R/W}$  bit of the address byte is clear and an address match occurs, the  $\overline{R/W}$  bit of the SSPSTAT register is cleared. The received address is loaded into the SSPBUF register and the SDA line is held low ( $\overline{ACK}$ ).

When the address byte overflow condition exists, then the No Acknowledge ( $\overline{ACK}$ ) pulse is given. An overflow condition is defined as either bit BF (SSPSTAT<0>) is set or bit SSPOV (SSPCON<6>) is set.

An MSSP interrupt is generated for each data transfer byte. Flag bit SSPIF (PIR1<3>) must be cleared in software. The SSPSTAT register is used to determine the status of the byte.

If SEN is enabled (SSPCON<0> = 1), RC3/SCK/SCL will be held low (clock stretch) following each data transfer. The clock must be released by setting bit CKP (SSPCON<4>). See Section 9.4.4 "Clock Stretching" for more detail.

## 9.4.3.3 Transmission

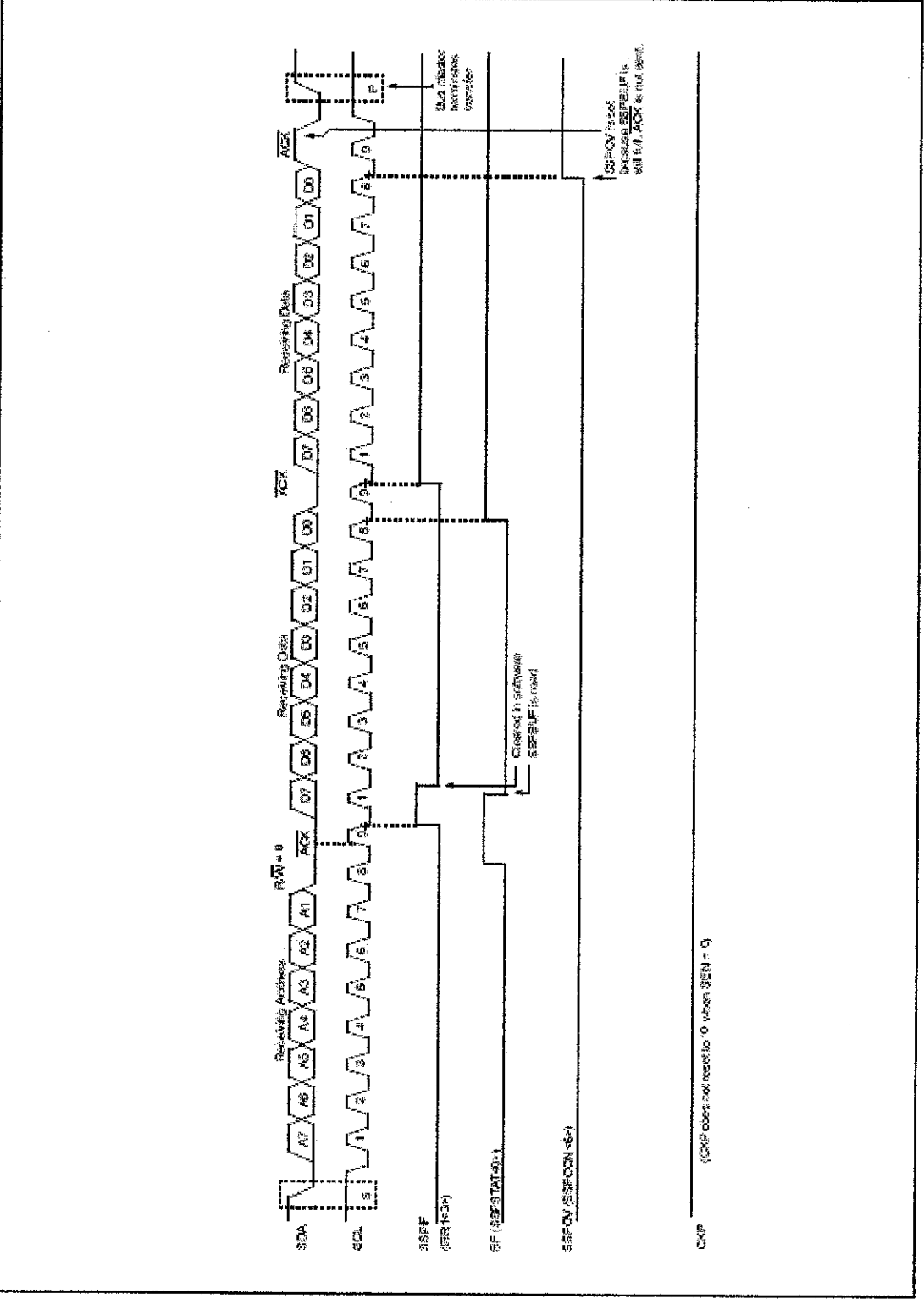
When the  $\overline{R/W}$  bit of the incoming address byte is set and an address match occurs, the  $\overline{R/W}$  bit of the SSPSTAT register is set. The received address is loaded into the SSPBUF register. The  $\overline{ACK}$  pulse will be sent on the ninth bit and pin RC3/SCK/SCL is held low regardless of SEN (see Section 9.4.4 "Clock Stretching" for more detail). By stretching the clock, the master will be unable to assert another clock pulse until the slave is done preparing the transmit data. The transmit data must be loaded into the SSPBUF register, which also loads the SSPSR register. Then pin RC3/SCK/SCL should be enabled by setting bit CKP (SSPCON<4>). The eight data bits are shifted out on the falling edge of the SCL input. This ensures that the SDA signal is valid during the SCL high time (Figure 9-9).

The  $\overline{ACK}$  pulse from the master-receiver is latched on the rising edge of the ninth SCL input pulse. If the SDA line is high (not  $\overline{ACK}$ ), then the data transfer is complete. In this case, when the  $\overline{ACK}$  is latched by the slave, the slave logic is reset (resets SSPSTAT register) and the slave monitors for another occurrence of the Start bit. If the SDA line was low ( $\overline{ACK}$ ), the next transmit data must be loaded into the SSPBUF register. Again, pin RC3/SCK/SCL must be enabled by setting bit CKP.

An MSSP interrupt is generated for each data transfer byte. The SSPIF bit must be cleared in software and the SSPSTAT register is used to determine the status of the byte. The SSPIF bit is set on the falling edge of the ninth clock pulse.

# PIC16F87XA

FIGURE 9-8: I<sup>2</sup>C SLAVE MODE TIMING WITH SEN = 0 (RECEPTION, 7-BIT ADDRESS)



14.11 Interrupts

The PIC16F87XA family has up to 15 sources of interrupt. The Interrupt Control register (INTCON) records individual interrupt requests in flag bits. It also has individual and global interrupt enable bits.

**Note:** Individual interrupt flag bits are set regardless of the status of their corresponding mask bit or the GIE bit.

A global interrupt enable bit, GIE (INTCON<7>), enables (if set) all unmasked interrupts or disables (if cleared) all interrupts. When bit GIE is enabled and an interrupt's flag bit and mask bit are set, the interrupt will vector immediately. Individual interrupts can be disabled through their corresponding enable bits in various registers. Individual interrupt bits are set regardless of the status of the GIE bit. The GIE bit is cleared on Reset.

The "return from interrupt" instruction, RETFIE, exits the interrupt routine, as well as sets the GIE bit, which re-enables interrupts.

The RB0/INT pin interrupt, the RB port change interrupt and the TMR0 overflow interrupt flags are contained in the INTCON register.

The peripheral interrupt flags are contained in the Special Function Registers, PIR1 and PIR2. The corresponding interrupt enable bits are contained in Special Function Registers, PIE1 and PIE2, and the peripheral interrupt enable bit is contained in Special Function Register, INTCON.

When an interrupt is responded to, the GIE bit is cleared to disable any further interrupt, the return address is pushed onto the stack and the PC is loaded with 0004h. Once in the Interrupt Service Routine, the source(s) of the interrupt can be determined by polling the interrupt flag bits. The interrupt flag bit(s) must be cleared in software before re-enabling interrupts to avoid recursive interrupts.

For external interrupt events, such as the INT pin or PORTB change interrupt, the interrupt latency will be three or four instruction cycles. The exact latency depends when the interrupt event occurs. The latency is the same for one or two-cycle instructions. Individual interrupt flag bits are set regardless of the status of their corresponding mask bit, PEIE bit or GIE bit.

FIGURE 14-10: INTERRUPT LOGIC

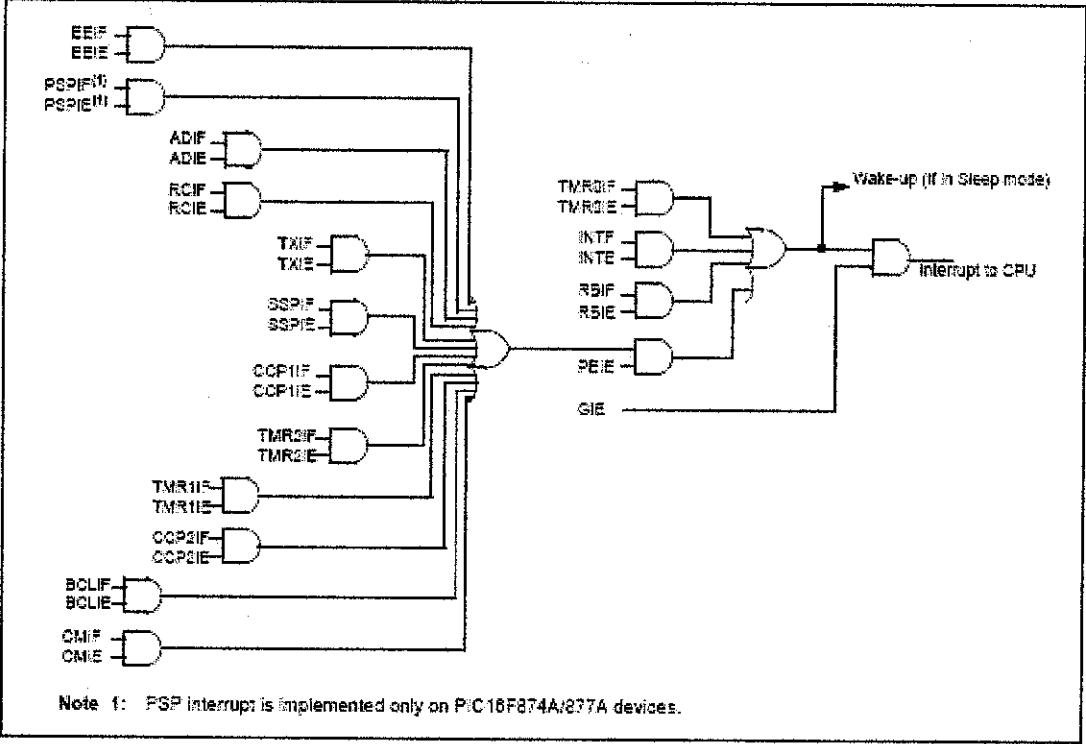
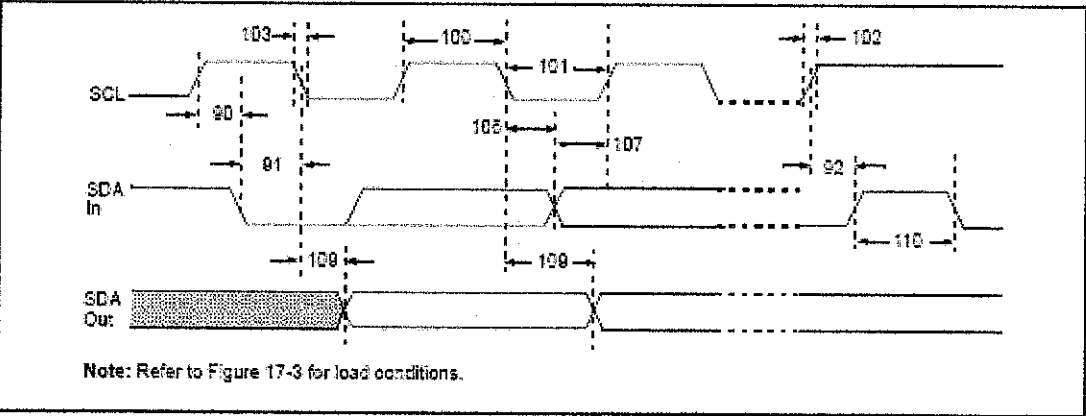


TABLE 17-10: I<sup>2</sup>C BUS START/STOP BITS REQUIREMENTS

Param No.	Symbol	Characteristic	Min	Typ	Max	Units	Conditions
90	TSU:STA	Start condition	100 kHz mode	4700	—	ns	Only relevant for Repeated Start condition
		Setup time	400 kHz mode	600	—	ns	
91	THD:STA	Start condition	100 kHz mode	4000	—	ns	After this period, the first clock pulse is generated
		Hold time	400 kHz mode	600	—	ns	
92	TSU:STO	Stop condition	100 kHz mode	4700	—	ns	
		Setup time	400 kHz mode	600	—	ns	
93	THD:STO	Stop condition	100 kHz mode	4000	—	ns	
		Hold time	400 kHz mode	600	—	ns	

FIGURE 17-16: I<sup>2</sup>C BUS DATA TIMING



# PIC16F87XA

TABLE 17-11: I<sup>2</sup>C BUS DATA REQUIREMENTS

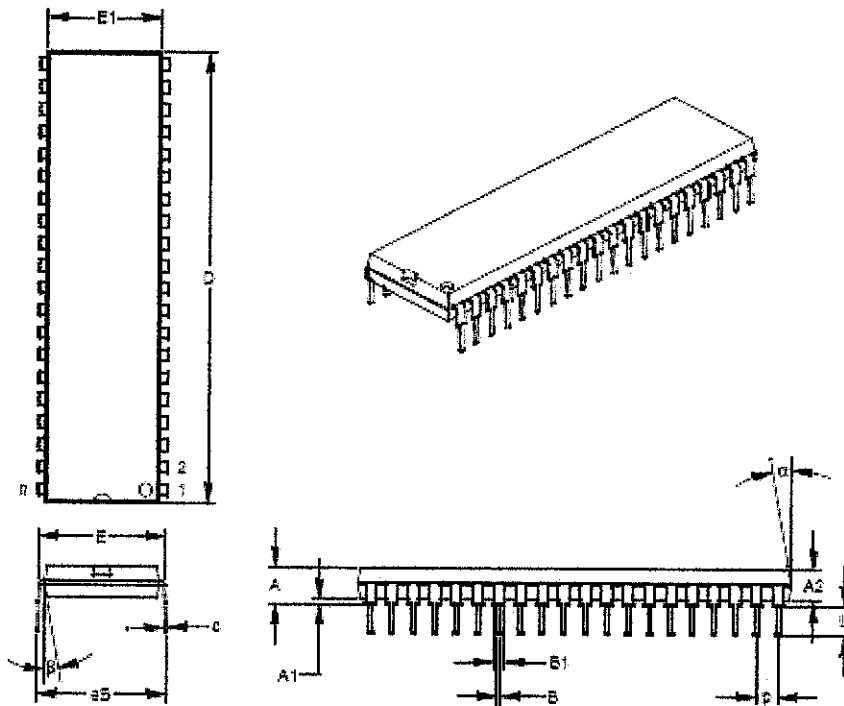
Param No.	Sym	Characteristic	Min	Max	Units	Conditions
100	THIGH	Clock High Time	100 kHz mode	4.0	—	μs
			400 kHz mode	0.6	—	μs
			SSP Module	0.5 Tcy	—	
101	TLOW	Clock Low Time	100 kHz mode	4.7	—	μs
			400 kHz mode	1.3	—	μs
			SSP Module	0.5 Tcy	—	
102	TR	SDA and SCL Rise Time	100 kHz mode	—	1000	ns
			400 kHz mode	20 + 0.1 Cb	300	ns Cb is specified to be from 10 to 400 pF
103	TF	SDA and SCL Fall Time	100 kHz mode	—	300	ns
			400 kHz mode	20 + 0.1 Cb	300	ns Cb is specified to be from 10 to 400 pF
90	TSU:STA	Start Condition Setup Time	100 kHz mode	4.7	—	μs
			400 kHz mode	0.6	—	μs
91	THD:STA	Start Condition Hold Time	100 kHz mode	4.0	—	μs
			400 kHz mode	0.6	—	μs
106	THD:DAT	Data Input Hold Time	100 kHz mode	0	—	ns
			400 kHz mode	0	0.9	μs
107	TSU:DAT	Data Input Setup Time	100 kHz mode	250	—	ns
			400 kHz mode	100	—	ns
92	TSU:STO	Stop Condition Setup Time	100 kHz mode	4.7	—	μs
			400 kHz mode	0.6	—	μs
109	TAA	Output Valid from Clock	100 kHz mode	—	3500	ns
			400 kHz mode	—	—	ns
110	TBUF	Bus Free Time	100 kHz mode	4.7	—	μs
			400 kHz mode	1.3	—	μs
	CB	Bus Capacitive Loading	—	400	pF	

- Note 1: As a transmitter, the device must provide this internal minimum delay time to bridge the undefined region (min. 300 ns) of the falling edge of SCL to avoid unintended generation of Start or Stop conditions.
- 2: A fast mode (400 kHz) I<sup>2</sup>C bus device can be used in a standard mode (100 kHz) I<sup>2</sup>C bus system, but the requirement that, TSU:DAT ≥ 250 ns, must then be met. This will automatically be the case if the device does not stretch the LOW period of the SCL signal. If such a device does stretch the LOW period of the SCL signal, it must output the next data bit to the SDA line, TR MAX. + TSU:DAT = 1000 + 250 = 1250 ns (according to the standard mode I<sup>2</sup>C bus specification), before the SCL line is released.



# PIC16F87XA

## 40-Lead Plastic Dual In-line (P) – 600 mil (PDIP)



Dimension Limits	Units	INCHES*			MILLIMETERS		
		MIN	NOM	MAX	MIN	NOM	MAX
Number of Pins	$\pi$		40			40	
Pitch	P		.100			2.54	
Top to Seating Plane	A	.160	.175	.190	4.06	4.45	4.83
Molded Package Thickness	A2	.140	.150	.160	3.56	3.81	4.06
Base to Seating Plane	A1	.015			0.38		
Shoulder to Shoulder Width	E	.555	.600	.625	15.11	15.24	15.88
Molded Package Width	E1	.530	.545	.560	13.46	13.84	14.22
Overall Length	D	2.045	2.058	2.065	51.94	52.26	52.45
Tip to Seating Plane	L	.120	.130	.135	3.05	3.32	3.43
Lead Thickness	$\phi$	.008	.012	.015	0.20	0.23	0.38
Upper Lead Width	B1	.030	.050	.070	0.76	1.27	1.78
Lower Lead Width	B	.014	.018	.022	0.36	0.45	0.56
Overall Row Spacing	$\phi$	.620	.650	.680	15.75	16.51	17.27
Mold Draft Angle Top	$\alpha$	5	10	15	5	10	15
Mold Draft Angle Bottom	$\beta$	5	10	15	5	10	15

\* Controlling Parameter

$\phi$  Significant Characteristic

Notes:

Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed .010" (0.254mm) per side.

JEDEC Equivalent: MO-011

Drawing No. C04-016



**PIC16F84A**  
**Data Sheet**

18-pin Enhanced FLASH/EEPROM  
8-bit Microcontroller



# PIC16F84A

## 18-pin Enhanced FLASH/EEPROM 8-Bit Microcontroller

### High Performance RISC CPU Features:

- Only 35 single word instructions to learn
- All instructions single-cycle except for program branches which are two-cycle
- Operating speed: DC - 20 MHz clock input  
DC - 200 ns instruction cycle
- 1024 words of program memory
- 66 bytes of Data RAM
- 64 bytes of Data EEPROM
- 14-bit wide instruction words
- 8-bit wide data bytes
- 15 Special Function Hardware registers
- Eight-level deep hardware stack
- Direct, indirect and relative addressing modes
- Four interrupt sources:
  - External RB0/INT pin
  - TMR0 timer overflow
  - PORTB<7:4> interrupt-on-change
  - Data EEPROM write complete

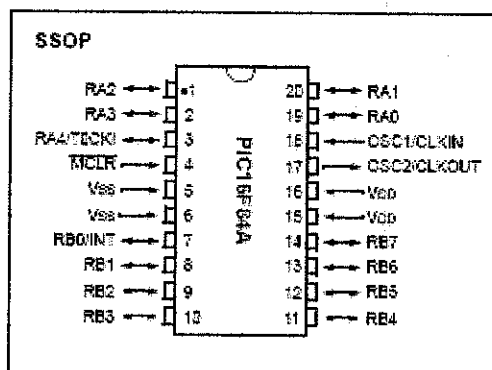
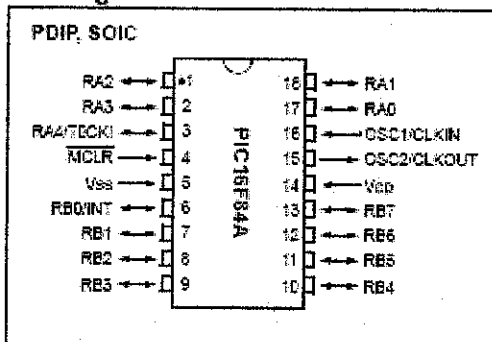
### Peripheral Features:

- 13 I/O pins with individual direction control
- High current sink/source for direct LED drive
  - 25 mA sink max. per pin
  - 25 mA source max. per pin
- TMR0: 8-bit timer/counter with 8-bit programmable prescaler

### Special Microcontroller Features:

- 10,000 erase/write cycles Enhanced FLASH Program memory typical
- 10,000,000 typical erase/write cycles EEPROM Data memory typical
- EEPROM Data Retention > 40 years
- In-Circuit Serial Programming™ (ICSP™) - via two pins
- Power-on Reset (POR), Power-up Timer (PWRT), Oscillator Start-up Timer (OST)
- Watchdog Timer (WDT) with its own On-Chip RC Oscillator for reliable operation
- Code protection
- Power saving SLEEP mode
- Selectable oscillator options

### Pin Diagrams



### CMOS Enhanced FLASH/EEPROM Technology:

- Low power, high speed technology
- Fully static design
- Wide operating voltage range:
  - Commercial: 2.0V to 5.5V
  - Industrial: 2.0V to 5.5V
- Low power consumption:
  - < 2 mA typical @ 5V, 4 MHz
  - 15 µA typical @ 2V, 32 kHz
  - < 0.5 µA typical standby current @ 2V

# PIC16F84A

## 1.0 DEVICE OVERVIEW

This document contains device specific information for the operation of the PIC16F84A device. Additional information may be found in the PICmicro™ Mid-Range Reference Manual, (DS33023), which may be downloaded from the Microchip website. The Reference Manual should be considered a complementary document to this data sheet, and is highly recommended reading for a better understanding of the device architecture and operation of the peripheral modules.

The PIC16F84A belongs to the mid-range family of the PICmicro® microcontroller devices. A block diagram of the device is shown in Figure 1-1.

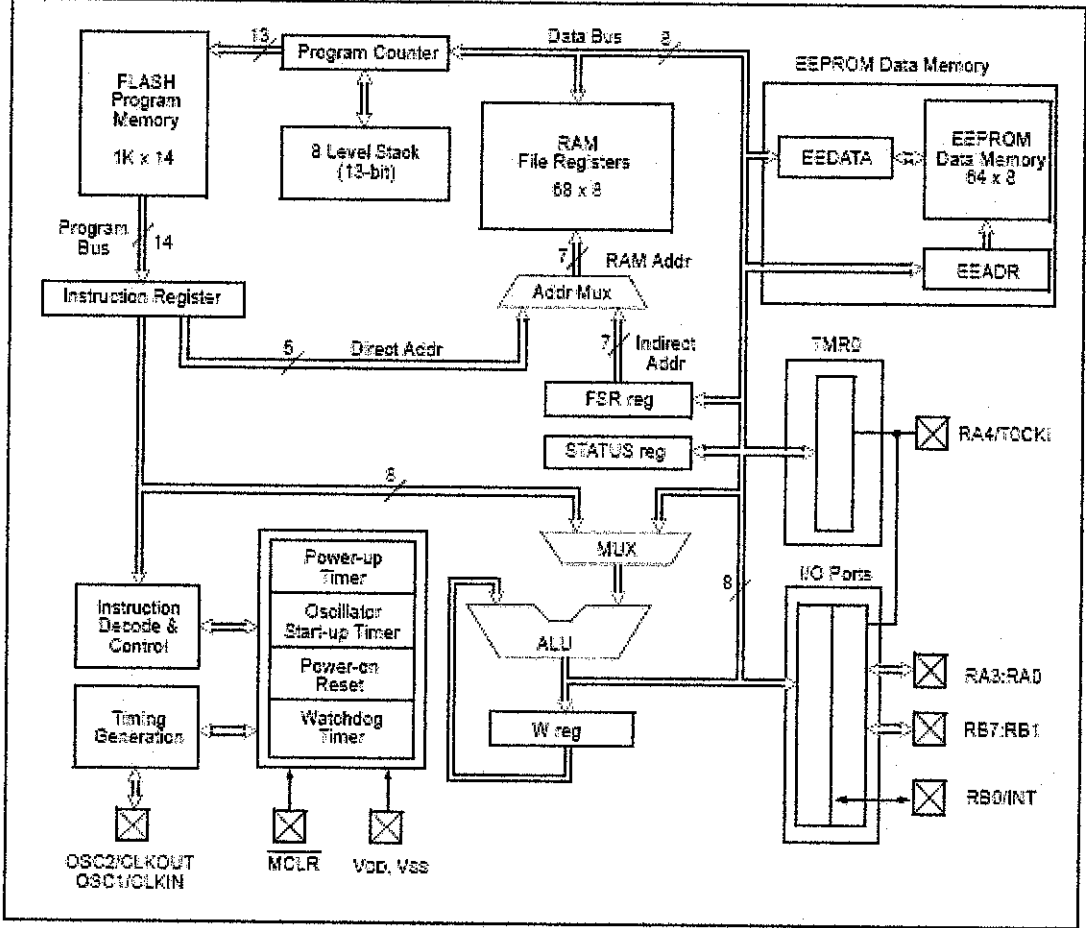
The program memory contains 1K words, which translates to 1024 instructions, since each 14-bit program memory word is the same width as each device instruction. The data memory (RAM) contains 68 bytes. Data EEPROM is 64 bytes.

There are also 13 I/O pins that are user-configured on a pin-to-pin basis. Some pins are multiplexed with other device functions. These functions include:

- External interrupt
- Change on PORTB interrupt
- Timer0 clock input

Table 1-1 details the pinout of the device with descriptions and details for each pin.

FIGURE 1-1: PIC16F84A BLOCK DIAGRAM



# PIC16F84A

TABLE 1-1: PIC16F84A PINOUT DESCRIPTION

Pin Name	PDIP No.	SOIC No.	SSOP No.	I/O/P Type	Buffer Type	Description
OSC1/CLKIN	16	16	18	I	ST/CMOS <sup>(3)</sup>	Oscillator crystal input/external clock source input.
OSC2/CLKOUT	15	15	19	O	—	Oscillator crystal output. Connects to crystal or resonator in Crystal Oscillator mode. In RC mode, OSC2 pin outputs CLKOUT, which has 1/4 the frequency of OSC1 and denotes the instruction cycle rate.
MCLR	4	4	4	I/P	ST	Master Clear (Reset) input/programming voltage input. This pin is an active low RESET to the device.
RA0	17	17	19	I/O	TTL	PORTA is a bi-directional I/O port.  Can also be selected to be the clock input to the TMR0 timer/counter. Output is open drain type.
RA1	18	18	20	I/O	TTL	
RA2	1	1	1	I/O	TTL	
RA3	2	2	2	I/O	TTL	
RA4/T0CKI	3	3	3	I/O	ST	
RB0/INT	6	6	7	I/O	TTL/ST <sup>(1)</sup>	PORTB is a bi-directional I/O port. PORTB can be software programmed for internal weak pull-up on all inputs.  RB0/INT can also be selected as an external interrupt pin.  Interrupt-on-change pin. Interrupt-on-change pin. Interrupt-on-change pin. Serial programming clock. Interrupt-on-change pin. Serial programming data.
RB1	7	7	8	I/O	TTL	
RB2	8	8	9	I/O	TTL	
RB3	9	9	10	I/O	TTL	
RB4	10	10	11	I/O	TTL	
RB5	11	11	12	I/O	TTL	
RB6	12	12	13	I/O	TTL/ST <sup>(2)</sup>	
RB7	13	13	14	I/O	TTL/ST <sup>(2)</sup>	
VSS	5	5	5,6	P	—	Ground reference for logic and I/O pins.
VDD	14	14	15,16	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.

## 2.3 Special Function Registers

The Special Function Registers (Figure 2-2 and Table 2-1) are used by the CPU and Peripheral functions to control the device operation. These registers are static RAM.

The special function registers can be classified into two sets, core and peripheral. Those associated with the core functions are described in this section. Those related to the operation of the peripheral features are described in the section for that specific feature.

TABLE 2-1: SPECIAL FUNCTION REGISTER FILE SUMMARY

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on RESET	Details on page
Bank 0											
00h	INDF	Uses contents of FSR to address Data Memory (not a physical register)								----	11
01h	TMR0	8-bit Real-Time Clock/Counter								XXXX XXXX	20
02h	PCL	Low Order 8 bits of the Program Counter (PC)								0000 0000	11
03h	STATUS <sup>(2)</sup>	IRP	RP1	RP0	TO	PD	Z	DC	C	0001 1XXX	8
04h	FSR	Indirect Data Memory Address Pointer 0								XXXX XXXX	11
05h	PORTA <sup>(4)</sup>	—	—	—	RA4/T0CKI	RA3	RA2	RA1	RA0	---x XXXX	16
06h	PORTB <sup>(5)</sup>	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0/INT	XXXX XXXX	18
07h	—	Unimplemented location, read as '0'								—	—
08h	EEDATA	EEPROM Data Register								XXXX XXXX	13,14
09h	EEADR	EEPROM Address Register								XXXX XXXX	13,14
0Ah	PCLATH	—	—	—	Write Buffer for upper 5 bits of the PC <sup>(1)</sup>				---0 0000	11	
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	10
Bank 1											
80h	INDF	Uses Contents of FSR to address Data Memory (not a physical register)								----	11
81h	OPTION_REG	RBP1	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	9
82h	PCL	Low order 8 bits of Program Counter (PC)								0000 0000	11
83h	STATUS <sup>(2)</sup>	IRP	RP1	RP0	TO	PD	Z	DC	C	0001 1XXX	8
84h	FSR	Indirect data memory address pointer 0								XXXX XXXX	11
85h	TRISA	—	—	—	PORTA Data Direction Register				---1 1111	16	
86h	TRISB	PORTB Data Direction Register								1111 1111	18
87h	—	Unimplemented location, read as '0'								—	—
88h	EECON1	—	—	—	EEIF	WRERR	WREN	WR	RD	---0 x000	13
89h	EECON2	EEPROM Control Register 2 (not a physical register)								----	14
0Ah	PCLATH	—	—	—	Write buffer for upper 5 bits of the PC <sup>(1)</sup>				---0 0000	11	
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	10

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0', q = value depends on condition

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a slave register for PC<12:8>. The contents of PCLATH can be transferred to the upper byte of the program counter, but the contents of PC<12:8> are never transferred to PCLATH.

2: The TO and PD status bits in the STATUS register are not affected by a MCLR Reset.

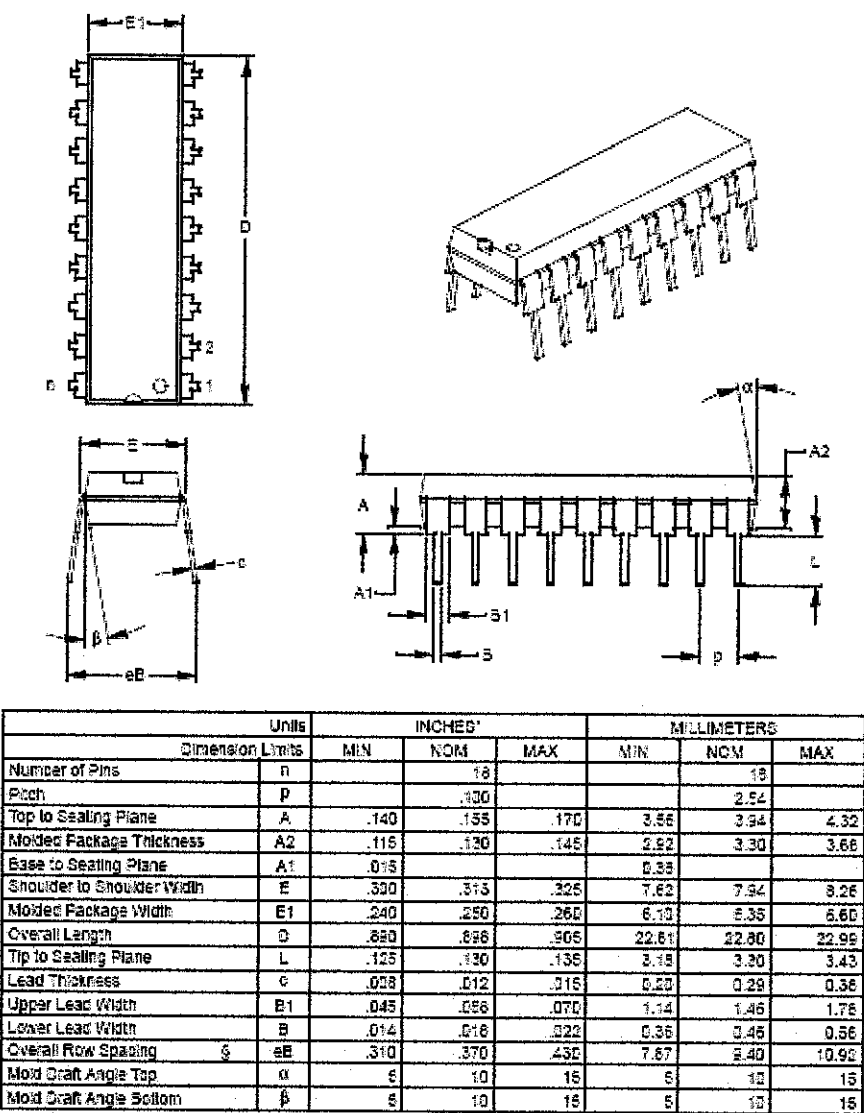
3: Other (non power-up) RESETS include: external RESET through MCLR and the Watchdog Timer Reset.

4: On any device RESET, these pins are configured as inputs.

5: This is the value that will be in the port output latch.

# PIC16F84A

18-Lead Plastic Dual In-line (P) – 300 mil (PDIP)



\* Controlling Parameter

§ Significant Characteristics

Notes:

Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed .010" (0.254mm) per side.

JEDEC Equivalent: MS-301

Drawing No. CD4-037

# APPENDIX B

## MAX232 DATASHEET

19-4223, Rev 7b, 1/97



### +5V-Powered, Multichannel RS-232 Drivers/Receivers

MAX220-MAX249

#### General Description

The MAX220-MAX249 family of line drivers/receivers is intended for all EIA/TIA-232E and V.28/V.24 communications interfaces, particularly applications where  $\pm 12V$  is not available.

These parts are especially useful in battery-powered systems, since their low-power shutdown mode reduces power dissipation to less than 5 $\mu$ W. The MAX225, MAX233, MAX235, and MAX245/MAX246/MAX247 use no external components and are recommended for applications where printed circuit board space is critical.

#### Applications

Portable Computers  
Low-Power Modems  
Interface Translation  
Battery-Powered RS-232 Systems  
Multi-Drop RS-232 Networks

#### Features

##### Superior to Bipolar

- † Operate from Single +5V Power Supply (+5V and +12V—MAX231/MAX239)
- † Low-Power Receive Mode in Shutdown (MAX223/MAX242)
- † Meet All EIA/TIA-232E and V.28 Specifications
- † Multiple Drivers and Receivers
- † 3-State Driver and Receiver Outputs
- † Open-Line Detection (MAX243)

#### Ordering Information

PART	TEMP. RANGE	PIN-PACKAGE
MAX220CPE	0°C to +70°C	16 Plastic DIP
MAX220CSE	0°C to +70°C	16 Narrow SO
MAX220CWE	0°C to +70°C	16 Wide SO
MAX220C/D	0°C to +70°C	Dice*
MAX220EPE	-40°C to +85°C	16 Plastic DIP
MAX220ESE	-40°C to +85°C	16 Narrow SO
MAX220EWE	-40°C to +85°C	16 Wide SO
MAX220EJE	-40°C to +85°C	16 CERDIP
MAX220MJE	-55°C to +125°C	16 CERDIP

Ordering Information continued at end of data sheet.

\*Contact factory for dice specifications.

#### Selection Table

Part Number	Power Supply (V)	No. of RS-232 Drivers/Rx	No. of Ext. Caps	Nominal Cap. Value ( $\mu$ F)	SHDN & Three-State	Rx Active in SHDN	Data Rate (kbps)	Features
MAX220	+5	2/2	4	4.7/10	No	—	120	Ultra-low-power, industry-standard pinout
MAX222	+5	2/2	4	0.1	Yes	—	200	Low-power shutdown
MAX223 (MAX213)	+5	4/5	4	1.0 (0.1)	Yes	✓	120	MAX241 and receivers active in shutdown
MAX225	+5	5/5	0	—	Yes	✓	120	Available in SO
MAX230 (MAX200)	+5	5/0	4	1.0 (0.1)	Yes	—	120	5 drivers with shutdown
MAX231 (MAX201)	+5 and +7.5 to +13.2	2/2	2	1.0 (0.1)	No	—	120	Standard +5/+12V or battery supplies; same functions as MAX232
MAX232 (MAX202)	+5	2/2	4	1.0 (0.1)	No	—	120 (C4)	Industry standard
MAX232A	+5	2/2	4	0.1	No	—	200	Higher slew rate, small caps
MAX233 (MAX203)	+5	2/2	0	—	No	—	120	No external caps
MAX233A	+5	2/2	0	—	No	—	200	No external caps, high slew rate
MAX234 (MAX204)	+5	4/0	4	1.0 (0.1)	No	—	120	Replaces 1498
MAX235 (MAX205)	+5	5/5	0	—	Yes	—	120	No external caps
MAX236 (MAX206)	+5	4/0	4	1.0 (0.1)	Yes	—	120	Shutdown, three state
MAX237 (MAX207)	+5	5/0	4	1.0 (0.1)	No	—	120	Complements IBM PC serial port
MAX238 (MAX208)	+5	4/4	4	1.0 (0.1)	No	—	120	Replaces 1498 and 1499
MAX239 (MAX209)	+5 and +7.5 to +13.2	2/5	2	1.0 (0.1)	No	—	120	Standard +5/+12V or battery supplies; single-package solution for IBM PC serial port
MAX240	+5	5/5	4	1.0	Yes	—	120	DIP or flatpack package
MAX241 (MAX211)	+5	4/5	4	1.0 (0.1)	Yes	—	120	Complete IBM PC serial port
MAX242	+5	2/2	4	0.1	Yes	✓	200	Separate shutdown and enable
MAX243	+5	2/2	4	0.1	No	—	200	Open-line detection simplifies cabling
MAX244	+5	8/10	4	1.0	No	—	120	High slew rate
MAX245	+5	8/10	0	—	Yes	✓	120	High slew rate, int. caps, two shutdown modes
MAX246	+5	8/10	0	—	Yes	✓	120	High slew rate, int. caps, three shutdown modes
MAX247	+5	8/9	0	—	Yes	✓	120	High slew rate, int. caps, nine operating modes
MAX248	+5	8/9	4	1.0	Yes	✓	120	High slew rate, selective half-chip enables
MAX249	+5	6/10	4	1.0	Yes	✓	120	Available in quad flatpack package



Maxim Integrated Products 1

For free samples & the latest literature: <http://www.maxim-ic.com>, or phone 1-800-998-8800.  
For small orders, phone 408-737-7600 ext. 3468.



## +5V-Powered, Multichannel RS-232 Drivers/Receivers

### ABSOLUTE MAXIMUM RATINGS—MAX220/222/232A/233A/242/243

Supply Voltage (V <sub>CC</sub> )	.....-0.3V to +6V	16-Pin Narrow SO (derate 8.70mW/°C above +70°C).....696mW
Input Voltages		16-Pin Wide SO (derate 9.62mW/°C above +70°C).....762mW
T <sub>IN</sub>	.....-0.3V to (V <sub>CC</sub> - 0.3V)	18-Pin Wide SO (derate 9.62mW/°C above +70°C).....762mW
R <sub>IN</sub>	.....±30V	20-Pin Wide SO (derate 10.00mW/°C above +70°C).....800mW
T <sub>OUT</sub> (Note 1)	.....±15V	20-Pin SSOP (derate 8.00mW/°C above +70°C).....640mW
Output Voltages		16-Pin CERDIP (derate 10.00mW/°C above +70°C).....800mW
T <sub>OUT</sub>	.....±15V	18-Pin CERDIP (derate 10.53mW/°C above +70°C).....842mW
R <sub>OUT</sub>	.....-0.3V to (V <sub>CC</sub> + 0.3V)	
Driver/Receiver Output Short Circuited to GND	.....Continuous	Operating Temperature Ranges
Continuous Power Dissipation (T <sub>A</sub> = +70°C)		MAX2__AC__ MAX2__C.....0°C to +70°C
16-Pin Plastic DIP (derate 10.53mW/°C above +70°C).....842mW		MAX2__AE__ MAX2__E.....40°C to +85°C
18-Pin Plastic DIP (derate 11.11mW/°C above +70°C).....889mW		MAX2__AM__ MAX2__M.....-55°C to +125°C
20-Pin Plastic DIP (derate 8.00mW/°C above +70°C).....640mW		Storage Temperature Range.....-65°C to +160°C
		Lead Temperature (soldering, 10sec).....+300°C

Note 1: Input voltage measured with T<sub>OUT</sub> in high-impedance state,  $\overline{\text{SHDN}}$  or V<sub>CC</sub> = 0V.

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

### ELECTRICAL CHARACTERISTICS—MAX220/222/232A/233A/242/243

(V<sub>CC</sub> = +5V ±10%, C1-C4 = 0.1μF, T<sub>A</sub> = T<sub>MIN</sub> to T<sub>MAX</sub>, unless otherwise noted.)

PARAMETER	CONDITIONS		MIN	TYP	MAX	UNITS
RS-232 TRANSMITTERS						
Output Voltage Swing	All transmitter outputs loaded with 3kΩ to GND		±5	±8		V
Input Logic Threshold Low				1.4	0.8	V
Input Logic Threshold High			2	1.4		V
Logic Pull-Up/Input Current	Normal operation			5	40	μA
	SHDN = 0V, MAX222/242, shutdown		±0.01	±1		
Output Leakage Current	VCC = 5.5V, SHDN = 0V, VOUT = ±15V, MAX222/242		±0.01	±10		μA
	VCC = SHDN = 0V, VOUT = ±15V		±0.01	±10		
Data Rate	All except MAX220, normal operation			200	116	kbits/ sec
	MAX220			22	20	
Transmitter Output Resistance	VCC = V+ = V- = 0V, VOUT = ±2V		300	10M		Ω
Output Short-Circuit Current	VOUT = 0V		±7	±22		mA
RS-232 RECEIVERS						
RS-232 Input Voltage Operating Range					±30	V
RS-232 Input Threshold Low	VCC = 5V	All except MAX243 R2IN	0.8	1.3		V
		MAX243 R2IN (Note 2)	-3			
RS-232 Input Threshold High	VCC = 5V	All except MAX243 R2IN		1.8	2.4	V
		MAX243 R2IN (Note 2)		-0.5	-0.1	
RS-232 Input Hysteresis	All except MAX243, VCC = 5V, no hysteresis in shdn.		0.2	0.5	1	V
	MAX243			1		
RS-232 Input Resistance			3	5	7	kΩ
TTLCMOS Output Voltage Low	IOUT = 3.2mA			0.2	0.4	V
TTLCMOS Output Voltage High	IOUT = -1.0mA		3.5	VCC - 0.2		V
TTLCMOS Output Short-Circuit Current	Sourcing VOUT = GND		-2	-10		mA
	Sinking VOUT = VCC		10	30		
TTLCMOS Output Leakage Current	SHDN = VCC or EN = VCC (SHDN = 0V for MAX222), 0V ≤ VOUT ≤ VCC			±0.05	±10	μA

## +5V-Powered, Multichannel RS-232 Drivers/Receivers

### ELECTRICAL CHARACTERISTICS—MAX223/MAX230-MAX241 (continued)

(MAX223/230/232/234/236/237/238/240/241,  $V_{CC} = +5V \pm 10\%$ ; MAX233/MAX235,  $V_{CC} = 5V \pm 5\%$ ,  $C_1$ - $C_4 = 1.0\mu F$ ; MAX231/MAX239,  $V_{CC} = 5V \pm 10\%$ ;  $V_+ = 7.5V$  to  $13.2V$ ;  $T_A = T_{MIN}$  to  $T_{MAX}$ ; unless otherwise noted.)

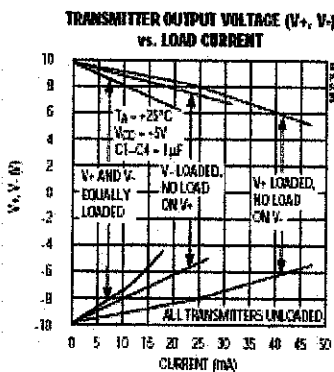
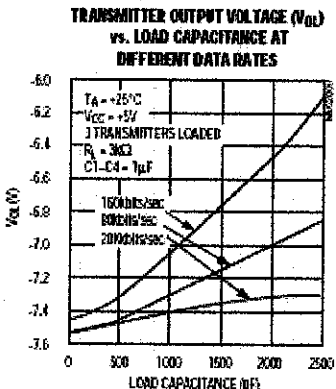
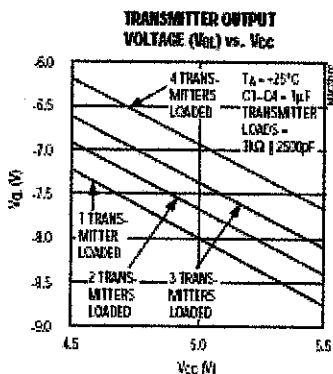
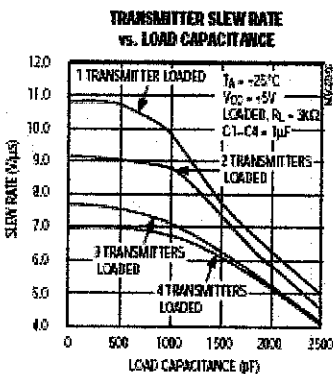
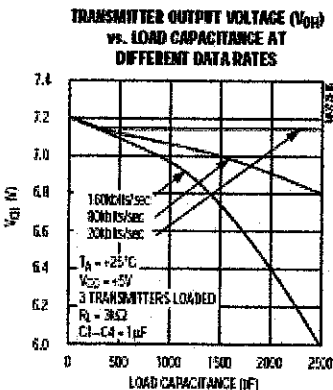
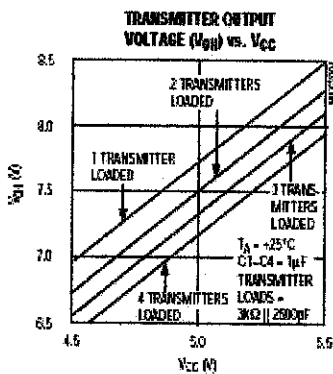
PARAMETER	CONDITIONS		MIN	TYP	MAX	UNITS
RS-232 Input Threshold Low	$T_A = +25^\circ C$ , $V_{CC} = 5V$	Normal operation SHDN = 5V (MAX223) SHDN = 0V (MAX235/236/240/241)	0.8	1.2		V
		Shutdown (MAX223) SHDN = 0V, EN = 5V ( $R_{4IN}$ , $R_{5IN}$ )	0.6	1.5		
RS-232 Input Threshold High	$T_A = +25^\circ C$ , $V_{CC} = 5V$	Normal operation SHDN = 5V (MAX223) SHDN = 0V (MAX235/236/240/241)		1.7	2.4	V
		Shutdown (MAX223) SHDN = 0V, EN = 5V ( $R_{4IN}$ , $R_{5IN}$ )		1.5	2.4	
RS-232 Input Hysteresis	$V_{CC} = 5V$ , no hysteresis in shutdown		0.2	0.5	1.0	V
RS-232 Input Resistance	$T_A = +25^\circ C$ , $V_{CC} = 5V$		3	5	7	k $\Omega$
TTL/CMOS Output Voltage Low	$I_{OUT} = 1.6mA$ (MAX231/232/233, $I_{OUT} = 3.2mA$ )				0.4	V
TTL/CMOS Output Voltage High	$I_{OUT} = -1mA$		3.5	$V_{CC} - 0.4$		V
TTL/CMOS Output Leakage Current	$0V \leq R_{OUT} \leq V_{CC}$ ; EN = 0V (MAX223); EN = $V_{CC}$ (MAX235-241)			0.05	$\pm 10$	$\mu A$
Receiver Output Enable Time	Normal operation	MAX223		600		ns
		MAX235/236/239/240/241		400		
Receiver Output Disable Time	Normal operation	MAX223		900		ns
		MAX235/236/239/240/241		250		
Propagation Delay	RS-232 IN to TTL/CMOS OUT, $C_L = 150pF$	Normal operation		0.5	10	$\mu s$
		SHDN = 0V (MAX223)	$t_{PHLS}$	4	40	
			$t_{PLHS}$	6	40	
Transition Region Slew Rate	MAX223/MAX230/MAX234-241, $T_A = +25^\circ C$ , $V_{CC} = 5V$ , $R_L = 3k\Omega$ to $7k\Omega$ , $C_L = 50pF$ to $2500pF$ , measured from $+3V$ to $-3V$ or $-3V$ to $+3V$		3	5.1	30	V/ $\mu s$
	MAX231/MAX232/MAX233, $T_A = +25^\circ C$ , $V_{CC} = 5V$ , $R_L = 3k\Omega$ to $7k\Omega$ , $C_L = 50pF$ to $2500pF$ , measured from $+3V$ to $-3V$ or $-3V$ to $+3V$			4	30	
Transmitter Output Resistance	$V_{CC} = V_+ = V_- = 0V$ , $V_{OUT} = \pm 2V$		300			$\Omega$
Transmitter Output Short-Circuit Current				$\pm 10$		mA

# **+5V-Powered, Multichannel RS-232 Drivers/Receivers**

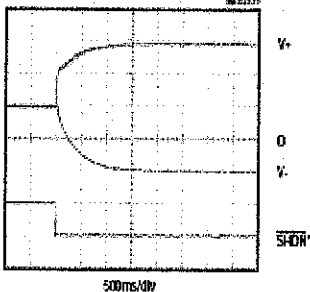
## **Typical Operating Characteristics**

**MAX223/MAX230-MAX241**

**MAX220-MAX249**



**V+, V- WHEN EXITING SHUTDOWN (1µF CAPACITORS)**



\*SHUTDOWN POLARITY IS REVERSED FOR NON MAX241 PARTS

# +5V-Powered, Multichannel RS-232 Drivers/Receivers

MAX220-MAX249

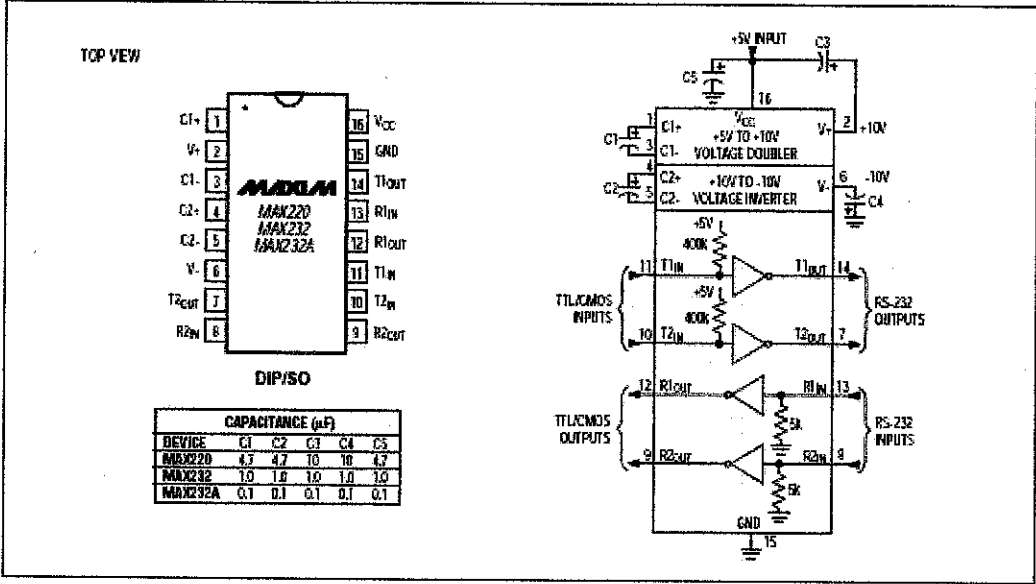


Figure 5. MAX220/MAX232/MAX232A Pin Configuration and Typical Operating Circuit

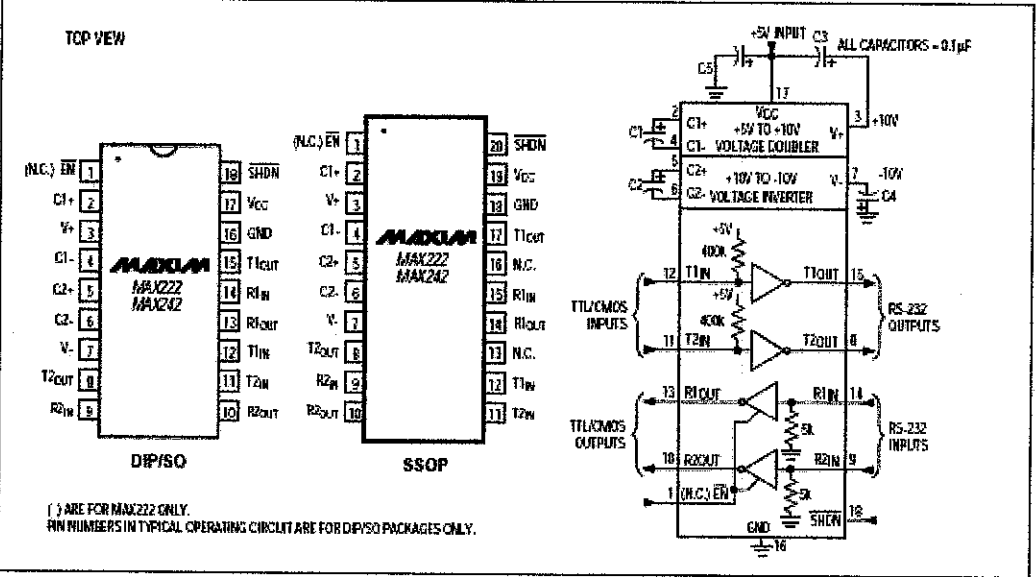


Figure 6. MAX222/MAX242 Pin Configurations and Typical Operating Circuit

MAXIM

## APPENDIX C

### SRF04 ULTRASONIC RANGE SENSOR DATASHEET



599 Menlo Drive, Suite 100  
Rocklin, California 95765, USA  
Office: (916) 624-8333  
Fax: (916) 624-8003

General: [info@parallax.com](mailto:info@parallax.com)  
Technical: [support@parallax.com](mailto:support@parallax.com)  
Web Site: [www.parallax.com](http://www.parallax.com)  
Educational: [www.stampsinclass.com](http://www.stampsinclass.com)

---

## Devantech SRF04 Ultrasonic Range Finder (#28015)

The Devantech SRF04 ultrasonic range finder provides precise, non-contact distance measurements from about 3 cm (1.2 inches) to 3 meters (3.3 yards). It is very easy to connect to BASIC Stamps or the Javelin, requiring only two I/O pins.<sup>1</sup> The SRF04 library makes this device very simple to use and is an ideal component for robotics applications.

The SRF04 works by transmitting an ultrasonic (well above human hearing range) pulse and measuring the time it takes to "hear" the pulse echo. Output from the SRF04 is in the form of a variable-width pulse that corresponds to the distance to the target.

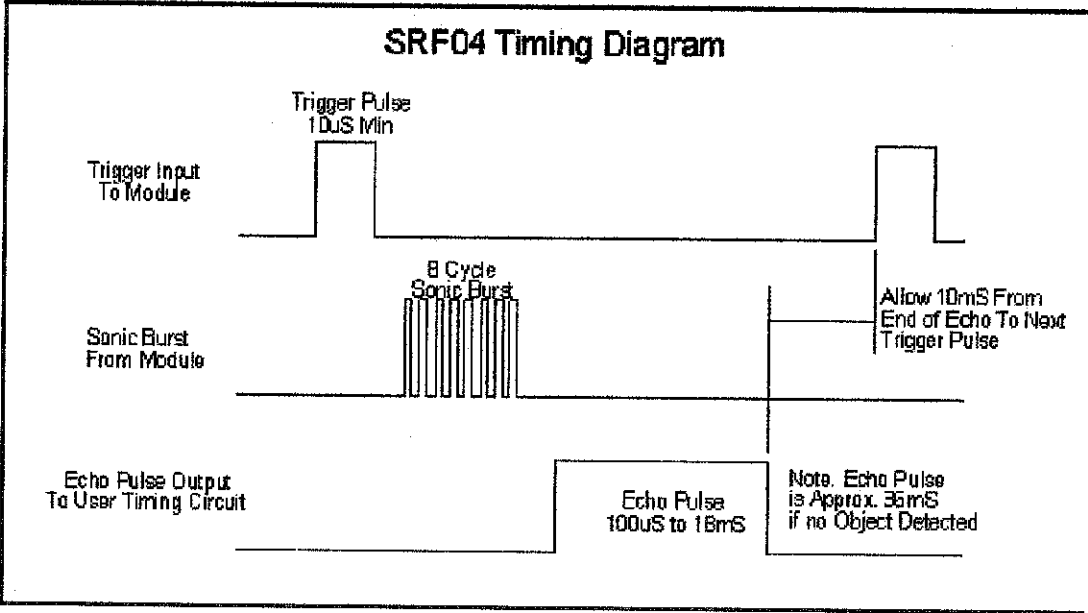
The SRF04 is designed and manufactured by Devantech, who provides additional technical resources for the device. Their web site is <http://www.robot-electronics.co.uk>.

### Features

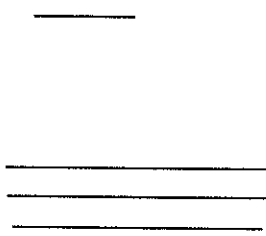
- Voltage – 5 v
- Current – 30 mA Typ. 50mA Max.
- Frequency – 40 kHz
- Max Range – 3 m
- Min Range – 3 cm
- Sensitivity – Detect 3 cm diameter broom handle at > 2 m
- Input Trigger – 10  $\mu$ s Min. TTL level pulse
- Echo Pulse – Positive TTL level signal, width proportional to range.
- Small Size – (1.7 in x .8 in x .7 in height) 43 mm x 20 mm x 17 mm height

---

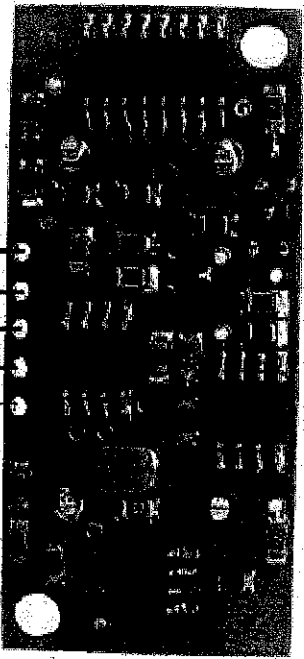
<sup>1</sup> For a Javelin Stamp application note see [www.javelinstamp.com](http://www.javelinstamp.com).



**SRF04  
Connections**



- 5v Supply
- Echo Pulse Output
- Trigger Pulse Input
- Do Not Connect
- 0v Ground

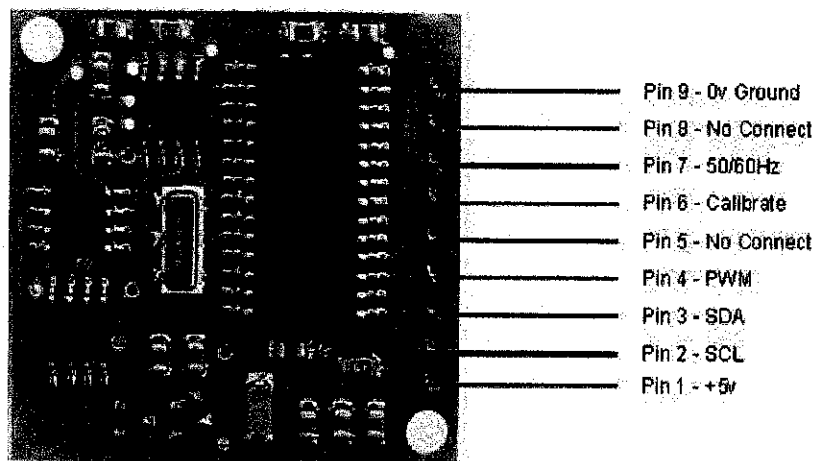


## APPENDIX D

### CMPS03 ELECTRONIC MAGNETIC COMPASS DATASHEET

This compass module has been specifically designed for use in robots as an aid to navigation. The aim was to produce a unique number to represent the direction the robot is facing. The compass uses the Philips KMZ51 magnetic field sensor, which is sensitive enough to detect the Earth's magnetic field. The output from two of them mounted at right angles to each other is used to compute the direction of the horizontal component of the Earth's magnetic field. We have examples of using the Compass module with a wide range of popular controllers.

Connections to the compass module

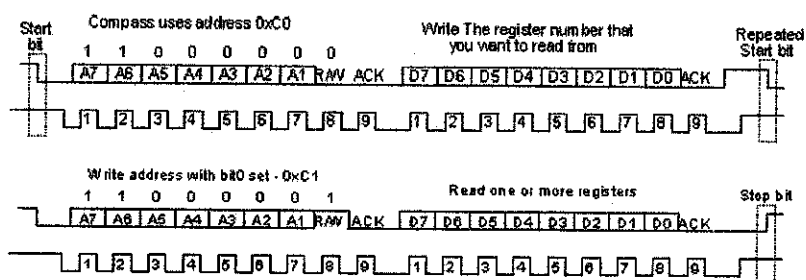


The compass module requires a 5v power supply at a nominal 15mA.

There are two ways of getting the bearing from the module. A PWM signal is available on pin 4, or an I2C interface is provided on pins 2,3.

The PWM signal is a pulse width modulated signal with the positive width of the pulse representing the angle. The pulse width varies from 1mS ( $0^\circ$ ) to 36.99mS ( $359.9^\circ$ ) – in other words  $100\mu\text{S}/^\circ$  with a +1mS offset. The signal goes low for 65mS between pulses, so the cycle time is 65mS + the pulse width - ie, 66ms-102ms. The pulse is generated by a 16 bit timer in the processor giving a 1 $\mu\text{S}$  resolution, however I would not recommend measuring this to anything better than  $0.1^\circ$  (10 $\mu\text{S}$ ). Make sure you connect the I2C pins, SCL and SDA, to the 5v supply if you are using the PWM, as there are no pull-up resistors on these pins.

Pin 2,3 are an I2C interface and can be used to get a direct readout of the bearing. If the I2C interface is not used then these pins should be pulled high (to +5v) via a couple of resistors. Around 47k is ok, the values are not at all critical.



I2C communication protocol with the compass module is the same as popular eeprom's such as the 24C04. First send a start bit, the module address (0XC0) with the read/write bit low, then the register number you wish to read. This is followed by a repeated start and the module address again with the read/write bit high (0XC1). You now read one or two bytes for 8bit or 16bit registers respectively. 16bit registers are read high byte first. The compass has a 16 byte array of registers, some of which double up as 16 bit registers as follows;

Register	Function
0	Software Revision Number
1	Compass Bearing as a byte, i.e. 0-255 for a full circle
2,3	Compass Bearing as a word, i.e. 0-3599 for a full circle, representing 0-359.9 degrees.
4,5	Internal Test - Sensor1 difference signal - 16 bit signed word
6,7	Internal Test - Sensor2 difference signal - 16 bit signed word
8,9	Internal Test - Calibration value 1 - 16 bit signed word
10,11	Internal Test - Calibration value 2 - 16 bit signed word
12	Unused - Read as Zero
13	Unused - Read as Zero
14	Unused - Read as Undefined
15	Calibrate Command - Write 255 to perform calibration step. See text.

Register 0 is the Software revision number (8 at the time of writing). Register 1 is the bearing converted to a 0- 255 value. This may be easier for some applications than 0-360 which requires two bytes. For those who require better resolution registers 2 and 3 (high byte first) are a 16 bit unsigned integer in the range 0-3599. This represents 0-359.9°. Registers 4 to 11 are internal test registers and 12,13 are unused. Register 14 is undefined. Don't read them if you don't want them - you'll just waste your I2C bandwidth. Register 15 is used to calibrate the compass. Full calibration information is here.

The I2C interface does not have any pull-up resistors on the board, these should be provided elsewhere, most probably with the bus master. They are required on both the SCL and SDA lines, but only once for the whole bus, not on each module. I suggest a value of 1k8 if you are going to be working up to 400KHz and 1k2 or even 1k if you are going up to 1MHz. The compass is designed to work at up to the standard clock speed (SCL) of 100KHz, however the clock speed can be raised to 1MHZ providing the following precaution is taken; At speeds above around 160KHz the CPU cannot respond fast enough to read the I2C data. Therefore a small delay of 50uS should be inserted either side of writing the register address. No delays are required anywhere else in the sequence. By doing this, I have tested the compass module up to 1.3MHz SCL clock speed. There is an example driver here using the HITECH PICC compiler for the PIC16F877. Note that the above is of no concern if you are using popular embedded language processors such as the OOPic. The compass module always operates as a slave, its never a bus master.

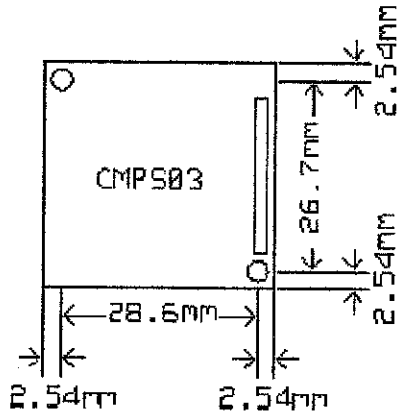


Pin 7 is an input pin selecting either 50Hz (low) or 60Hz (high) operation. I added this option after noticing a jitter of around  $1.5^\circ$  in the output. The cause was the 50Hz mains field in my workshop. By converting in synchronism with the mains frequency this was reduced to around  $0.2^\circ$ . An internal conversion is done every 40mS (50Hz) or every 33.3mS (60Hz). The pin has an on-board pull-up can be left unconnected for 60Hz operation. There is no synchronism between the PWM or I2C outputs and the conversion. They both retrieve the most recent internal reading, which is continuously converted, whether it is used or not. Pin 6 is used to calibrate the compass. The calibrate input (pin 6) has an on-board pull-up resistor and can be left unconnected after calibration. **Calibration is identical to the CMPS01 Rev7 procedure.** Full calibration information is here.

Pins 5 and 8 are No Connect. Actually pin 8 is the processor reset line and has an on-board pull-up resistor. It is there so that we can program the processor chip after placement on the PCB.

**PCB Drilling Plan**

The following diagram shows the CMPS03 PCB mounting hole positions.



# APPENDIX E

## C CODE GENERATED

### Master Microcontroller PIC16F877A Code

*Testing program for obstacle avoidance and path planning  
of Autonomaus Guided Robot (AGR)  
Created by: Mohd Azlan Shah Bin Abd Rahim (5146)  
Electrical & Electronics Engineering Department  
University of Technology PETRONAS*

#### NOTES:

*PIN\_B0 = triggering output to ultrasonic sensor front  
PIN\_B1 = triggering output to ultrasonic sensor left  
PIN\_B2 = triggering output to ultrasonic sensor right  
PIN\_A2 = input pulse from ultrasonic sensor front  
PIN\_A3 = input pulse from ultrasonic sensor left  
PIN\_A4 = input pulse from ultrasonic sensor right  
PIN\_B4 = output to servo controller (bit 0)  
PIN\_B5 = output to servo controller (bit 1)  
PIN\_B6 = output to servo controller (bit 2)  
PIN\_B7 = Input from servo controller  
PIN\_C1 = CCP1 PWM LEFTWHEEL output  
PIN\_C2 = CCP2 PWM RIGHTWHEEL output  
PIN\_E1 = LEFTWHEEL direction control output  
PIN\_E2 = LEFTWHEEL direction control output  
PIN\_C4 = RIGHTWHEEL direction control output  
PIN\_C5 = RIGHTWHEEL direction control output  
PIN\_D0 = Sequencing LED 0  
PIN\_D1 = Sequencing LED 1  
PIN\_D2 = Sequencing LED 2*

1	#include <16F877A.h>
2	#use delay(clock=20000000)
3	#fuses HS,NOPROTECT,NOWDT, NOBROWNOUT, NOPUT, NOLVP
4	#include <16F877A.h>
5	#use delay(clock=20000000)
6	#fuses HS,NOPROTECT,NOWDT, NOBROWNOUT, NOPUT, NOLVP
7	#use I2C(master, sda=PIN_D0, scl=PIN_D1)
8	
9	
10	#define slow 30
11	#define medium 70

```

12 #define fast 110
13
14 int32 value;
15 int32 distance;
16 signed int16 direction[2];
17 signed int16 difference;
18 int16 degree;
19 int32 frontsense;
20 int32 rightsense;
21 int32 leftsense;
22 int front;
23 int left;
24 int right;
25 int action;
26
27 void motor(int32 LEFTI,int32 RIGHTI)
28 {
29     set_pwm1_duty(LEFTI);
30     set_pwm2_duty(RIGHTI);
31 }
32
33 servotrl(int action)
34 {
35     switch(action)
36     {
37         case 1: output_bit(PIN_B4,0); //direction 0
38             output_bit(PIN_B5,0);
39             output_bit(PIN_B6,0);
40             break;
41         case 2: output_bit(PIN_B4,1); //direction -60
42             output_bit(PIN_B5,0);
43             output_bit(PIN_B6,0);
44             break;
45         case 3: output_bit(PIN_B4,0); //direction 60
46             output_bit(PIN_B5,0);
47             output_bit(PIN_B6,1);
48             break;
49         case 4: output_bit(PIN_B4,1); //direction -45
50             output_bit(PIN_B5,1);
51             output_bit(PIN_B6,0);
52             break;
53         case 5: output_bit(PIN_B4,0); //direction 45
54             output_bit(PIN_B5,1);
55             output_bit(PIN_B6,1);
56             break;
57         case 6: output_bit(PIN_B4,1); //curve -45
58             output_bit(PIN_B5,0);
59             output_bit(PIN_B6,1);
60             break;
61         case 7: output_bit(PIN_B4,1); //curve 45
62             output_bit(PIN_B5,1);
63             output_bit(PIN_B6,1);
64             break;
65         default: break;
66     }

```

```

67 }
68
69 int16 compass()
70 {
71
72     i2c_start();
73
74     i2c_write(0xC0);
75
76     i2c_write(0x01);
77
78     i2c_start();
79
80     i2c_write(0xC1);
81
82     direction[1] = i2c_read(0);
83     difference = direction[1] - direction[0];
84
85     if (direction[1] >= 63 && direction[1] <= 65)           //90degree
86     {
87         degree = 90;
88     }
89     if(direction[1] >= 191 && direction[1] <= 193)
90     {
91         degree = 270;
92     }
93     if(direction[1] >= 66 && direction[1] <= 190)           //more than 90degree
94     {
95         degree = 180;
96     }
97     if(direction[1] >= 31 && direction[1] <= 33)           //45degree
98     {
99         degree = 45;
100    }
101    if(direction[1] >= 223 && direction[1] <= 225)           //-45degree
102    {
103        degree = 315;
104    }
105    if(direction[1] == 0)                                     // 0degree
106    {
107        degree = 0;
108    }
109
110    i2c_stop();
111    direction[0] = direction[1];
112    return degree;
113 }
114
115 void forward()
116 {
117     output_high(PIN_E1);
118     output_low(PIN_E2);
119     set_pwm1_duty(medium);
120     output_low(PIN_C4);
121     output_high(PIN_C5);

```

```

122     set_pwm2_duty(medium);
123 }
124
125 void backward()
126 {
127     output_low(PIN_E1);
128     output_high(PIN_E2);
129     set_pwm1_duty(slow);
130     output_high(PIN_C4);
131     output_low(PIN_C5);
132     set_pwm2_duty(slow);
133 }
134
135 void stop()
136 {
137     output_low(PIN_E1);
138     output_low(PIN_E2);
139     output_low(PIN_C4);
140     output_low(PIN_C5);
141 }
142
143 void turn_left()
144 {
145     output_bit(PIN_D0,1);
146     output_bit(PIN_D1,1);
147     stop();
148     servoctrl(3);
149     backward();
150     delay_ms(1000);
151     stop();
152     servoctrl(2);
153     delay_ms(1000);
154     forward();
155     while(1)
156     {
157         if(compass()==270) break;
158         else motor(slow,slow);
159     }
160     servoctrl(1);
161     motor(medium,medium);
162     output_bit(PIN_D0,0);
163     output_bit(PIN_D1,0);
164 }
165
166 void turn_right()
167 {
168     output_bit(PIN_D1,1);
169     output_bit(PIN_D2,1);
170     stop();
171     servoctrl(2);
172     backward();
173     delay_ms(1000);
174     stop();
175     servoctrl(3);
176     delay_ms(1000);

```

```

177 forward();
178 while(1)
179 {
180   if(compass()==90) break;
181   else motor(slow,slow);
182 }
183 servotrl(1);
184 motor(medium,medium);
185 output_bit(PIN_D1,0);
186 output_bit(PIN_D2,0);
187 }
188
189 void curve_left()
190 {
191   servotrl(6);
192   while(1)
193   {
194     if(input(PIN_B7) == 1) break;
195     else motor(slow,slow);
196   }
197   servotrl(4);
198   delay_ms(1000);
199   servotrl(5);
200   while(1)
201   {
202     if(compass()==0) break;
203     else motor(slow,slow);
204   }
205   servotrl(1);
206   delay_ms(1000);
207   motor(fast,fast);
208
209 }
210
211 void curve_right()
212 {
213   servotrl(7);
214   while(1)
215   {
216     if(input(PIN_B7) == 0) break;
217     else motor(slow,slow);
218   }
219   servotrl(5);
220   delay_ms(1000);
221   servotrl(4);
222   while(1)
223   {
224     if(compass()==0) break;
225     else motor(slow,slow);
226   }
227   servotrl(1);
228   delay_ms(1000);
229   motor(fast,fast);
230
231 }

```

```

232
233
234 int32 ranger1()
235 {
236     value = 0;
237     output_bit(PIN_B0,1);
238     delay_us(10);
239     output_bit(PIN_B0,0);
240
241     while(1)
242     {
243         if(input(PIN_A2) == 1) break;
244         else;
245     }
246     while(1)
247     {
248         if(input(PIN_A2) == 0 || value > 36000) break;
249         else
250         {
251             value++;
252             delay_us(1);
253         }
254     }
255
256     if(value>0 && value<36000) distance = value/36;
257     else distance = 0;
258
259     return distance;
260
261     delay_ms(10);
262
263 }
264
265 int32 ranger2()
266 {
267     value = 0;
268     output_bit(PIN_B1,1);
269     delay_us(10);
270     output_bit(PIN_B1,0);
271
272     while(1)
273     {
274         if(input(PIN_A3) == 1) break;
275         else;
276     }
277     while(1)
278     {
279         if(input(PIN_A3) == 0 || value > 36000) break;
280         else
281         {
282             value++;
283             delay_us(1);
284         }
285     }
286

```

```

287   if(value>0 && value<36000) distance = value/36;
288   else distance = 0;
289
290   return distance;
291
292   delay_ms(10);
293
294   }
295
296   int32 ranger3()
297   {
298     value = 0;
299     output_bit(PIN_B2,1);
300     delay_us(10);
301     output_bit(PIN_B2,0);
302
303     while(1)
304     {
305       if(input(PIN_A4) == 1) break;
306       else;
307     }
308     while(1)
309     {
310       if(input(PIN_A4) == 0 || value > 36000) break;
311       else
312       {
313         value++;
314         delay_us(1);
315       }
316     }
317
318     if(value>0 && value<36000) distance = value/36;
319     else distance = 0;
320
321     return distance;
322
323     delay_ms(10);
324
325     }
326
327
328   int obstacle_front()    //obstacle in front algorithm
329   {
330     frontsense = ranger1();
331     if(frontsense>0 && frontsense<=3)
332     {
333       output_bit(PIN_D4,1);
334       return 1;
335     }
336     else if (frontsense<=0 || frontsense>3)
337     {
338       output_bit(PIN_D4,0);
339       return 0;
340     }
341     else;

```



```

342 }
343
344 int obstacle_left() //obstacle left algorithm
345 {
346     leftsense = ranger2();
347     if(leftsense>0 && leftsense<=3)
348     {
349         output_bit(PIN_D5,1);
350         return 1;
351     }
352     else if (leftsense<=0 || leftsense>3)
353     {
354         output_bit(PIN_D5,0);
355         return 0;
356     }
357     else;
358 }
359
360 int obstacle_right() //obstacle right algorithm
361 {
362     rightsense = ranger3();
363     if(rightsense>0 && rightsense<=3)
364     {
365         output_bit(PIN_D6,1);
366         return 1;
367     }
368     else if (rightsense<=0 || rightsense>3)
369     {
370         output_bit(PIN_D6,0);
371         return 0;
372     }
373     else;
374 }
375
376
377 void obstacle_avoidance()
378 {
379     int32 xtra = 0;
380     front = obstacle_front();
381     left = obstacle_left();
382     right = obstacle_right();
383
384     if(front==1 && left==1 && right==1) //long wall detected
385     {
386         output_bit(PIN_D0, 1);
387     }
388     else if(front==1 && right==1 && left==0) //obstacle front+right
389     {
390         stop();
391         delay_ms(1000);
392         turn_left();
393     avoidleft: do {
394         //curve_left;
395         xtra++;
396         right = obstacle_right();

```

```

397     }while(right==1);
398
399     if(obstacle_right()==1) goto avoidleft;
400     else turn_right();
401
402     output_bit(PIN_D1, 0);
403
404     while(xtra!=0)
405     {
406         obstacle_right();
407         if(right==1)
408         {
409             output_bit(PIN_D2, 1);
410             forward();
411             motor(260,260);
412         }
413         else
414         {
415             output_bit(PIN_D2, 0);
416             turn_right();
417             while(xtra!=0)
418             {
419                 forward();
420                 motor(250,320);
421                 xtra--;
422             }
423             turn_left();
424         }
425     }
426
427 }
428 else if(front==1 && left==1 && right==0) //obstacle front+left
429 {
430     stop();
431     delay_ms(1000);
432     turn_right();
433 avoidright: while(obstacle_left()==1)
434 {
435     curve_right();
436     xtra++;
437 }
438
439 if(obstacle_left()==0) turn_left();
440 else; // goto avoidright;
441
442 output_bit(PIN_D1, 0);
443
444 while(xtra!=0)
445 {
446     if(obstacle_left()==1)
447     {
448         output_bit(PIN_D2, 1);
449         forward();
450         motor(250,320);
451     }

```

```

452     else
453     {
454         output_bit(PIN_D2, 0);
455         turn_left();
456         while(xtra!=0)
457         {
458             forward();
459             motor(250,320);
460             xtra--;
461         }
462         turn_right();
463     }
464 }
465 }
466 else if(front==1 && left==0 && right==0) //obstacle front(take curve)
467 {
468
469     if(obstacle_front()==1) motor(310,104);
470     else motor(250,320);
471     delay_ms(100);
472
473     while(obstacle_left()==1)
474     {
475         output_bit(PIN_D0,1);
476         if(ranger2()>0 && ranger2()<3) motor(260,140);    //taking right curve
477                                                         //(dampening effect to keep distance from object 5cm)
478         else if(ranger2()==3) motor(200,200);
479         else if(ranger2()<0 && ranger2()>3) motor(140,260);
480     }
481
482     motor(104,310);
483     delay_ms(500);
484     motor(260,140);
485     delay_ms(200);
486     motor(260,260);
487
488     output_bit(PIN_D0,0);
489 }
490 else
491 {
492     forward();
493     motor(200,260); //motor control forward
494 }
495 }
496
497
498 void main()
499 {
500
501     setup_ccp1(CCP_PWM); //setup PWM pins CCP1 & CCP2
502     setup_ccp2(CCP_PWM);
503     setup_timer_2(T2_DIV_BY_16, 127, 1);
504
505     stop();

```

506	delay_ms(2000);
507	
508	while(1) //continuous loop of program
509	{
510	forward();
511	compass();
512	obstacle_front();
513	obstacle_left();
514	obstacle_right();
515	obstacle_avoidance();
516	}
517	}

### Servo Slave Microcontroller PIC16F84A Code

*Testing program for obstacle avoidance and path planning  
of Autonomous Guided Robot (AGR)*

*Created by: Mohd Azlan Shah Bin Abd Rahim (5146)*

*Electrical & Electronics Engineering Department*

*University of Technology PETRONAS*

#### NOTES:

*PIN\_B0 = output to master controller*

*PIN\_B4 = input from master controller (bit 0)*

*PIN\_B5 = input from master controller (bit 1)*

*PIN\_B6 = input from master controller (bit 2)*

*PIN\_A1 = output to servo (left)*

*PIN\_A2 = output to servo (right)*

1	
2	
3	#include <16f84A.h>
4	#fuses XT,NOWDT,NOPROTECT,NOPUT
5	#use delay (clock=4000000)
6	
7	int delay, count;
8	
9	void servocurve(int dir)
10	{
11	if(dir==1) //left curve
12	{
13	for(delay=1350;delay<=1800;delay++)
14	{
15	for(count=0;count<=48000;count++)
16	{
17	output_bit(PIN_A2,1);
18	output_bit(PIN_A1,1);
19	delay_us(delay);
20	output_bit(PIN_A2,0);
21	output_bit(PIN_A1,0);
22	delay_ms(20);

```

23     }
24     }
25     }else;
26
27     if(dir==2) //right curve
28     {
29         for(delay=1350;delay<=810;delay--)
30         {
31             for(count=0;count<=48000;count++)
32             {
33                 output_bit(PIN_A2,1);
34                 output_bit(PIN_A1,1);
35                 delay_us(delay);
36                 output_bit(PIN_A2,0);
37                 output_bit(PIN_A1,0);
38                 delay_ms(20);
39             }
40         }
41     }else;
42 }
43
44 void standard()
45 {
46     if (input(PIN_B5)==0 && input(PIN_B4)==0 && input(PIN_B6)==0)    //standard 0
47     {
48         output_bit(PIN_A2,1);
49         output_bit(PIN_A1,1);
50         delay_us(1350);
51         output_bit(PIN_A2,0);
52         output_bit(PIN_A1,0);
53         delay_ms(20);
54     }
55
56     else if(input(PIN_B5)==0 && input(PIN_B4)==0 && input(PIN_B6)==1) //standard 60
57     {
58         output_bit(PIN_A2,1);
59         output_bit(PIN_A1,1);
60         delay_us(1950);
61         output_bit(PIN_A2,0);
62         output_bit(PIN_A1,0);
63         delay_ms(20);
64     }
65
66     else if(input(PIN_B5)==0 && input(PIN_B4)==1 && input(PIN_B6)==0) //standard -60
67     {
68         output_bit(PIN_A2,1);
69         output_bit(PIN_A1,1);
70         delay_us(750);
71         output_bit(PIN_A2,0);
72         output_bit(PIN_A1,0);
73         delay_ms(20);
74     }
75
76     else if(input(PIN_B5)==1 && input(PIN_B4)==0 && input(PIN_B6)==1) //standard 45
77     {

```

```

78  output_bit(PIN_A2,1);
79  delay_us(810);
80  output_bit(PIN_A2,0);
81  delay_ms(20);
82  }
83
84  else if(input(PIN_B5)==1 && input(PIN_B4)==1 && input(PIN_B6)==0) //standard -45
85  {
86  output_bit(PIN_A2,1);
87  output_bit(PIN_A1,1);
88  delay_us(1800);
89  output_bit(PIN_A2,0);
90  output_bit(PIN_A1,0);
91  delay_ms(20);
92  }
93
94  else if(input(PIN_B5)==0 && input(PIN_B4)==1 && input(PIN_B6)==1) //curve -45
95  {
96  servocurve(1);
97  output_bit(PIN_B0,1); //if finish change to standard -45
98  }
99
100 else if(input(PIN_B5)==1 && input(PIN_B4)==1 && input(PIN_B6)==1) //curve 45
101 {
102 servocurve(2);
103 output_bit(PIN_B0,0); //if finish change to standard 45
104 }
105 else;
106 }
107
108 void main()
109 {
110 while(1)
111 {
112 standard(); //continuous loop
113 }
114 }

```