

Contactless Measurement of Angular - Speed

By

DOVLET BAZAROV

1624

Dissertation submitted in partial fulfillment of
the requirement for the
Bachelor of Engineering (Hons)
(Electrical & Electronics Engineering)

April 2004

Universiti Teknologi Petronas
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

t
TJ
223
P76
B362
2004

1. Programmable Controllers
2. Microprocessors - Programming
3. EEE - Thesis

CERTIFICATION OF APPROVAL

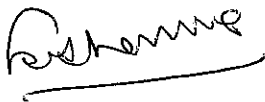
Contactless Measurement of Angular-Speed

By

Dovlet Bazarov

A project dissertation submitted to the
Electrical & Electronics Engineering Programme
Universiti Teknologi Petronas
In partial fulfillment of the requirement for the
BACHELOR OF ENGINEERING (Hons)
(ELECTRICAL & ELECTRONICS ENGINEERING)

Approved by,



Prof. Dr. Pramod Chandra Sharma

Dr. P.C. Sharma
Professor
Electrical Engineering Dept.
Universiti Teknologi PETRONAS
31750 Tronoh
Perak Darul Ridzuan, MALAYSIA


UNIVERSITI TEKNOLOGI PETRONAS

TRONOH, PERAK

April 2004

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 s specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.



Dovlet Bazarov

ACKNOWLEDGEMENTS

I would sincerely like to thank my FYP Supervisor Prof. Dr. Pramod Chandra Sharma very much. I owe the success of my project to him. He has been completely supportive throughout my project. I wish for him the lifetime of happiness.

I thank our FYP Coordinator Mr. Zuki for his generous assistance. I also feel grateful to my senior Arslan Rozyyev for his valuable advices.

ABSTRACT

The control technology has been improved to enormous scales so far. People are trying to apply control over everything they do, so that the level of accuracy and efficiency increase. The simplest form of control could be a control of a rotating shaft. We might want to keep its speed stable at some rps (rounds per second). This can be achieved in numerous methods. Regardless of any method chosen, the first step to achieve this goal would always be being able to measure the speed and then to apply control measures to it.

In this project, I am investigating *contactless measurement of angular speed* of a rotating shaft. My main objective is to demonstrate how to measure the angular speed of a rotating shaft contactlessly. Contactless measurement of angular speed is particularly critical when dealing with load-sensitive devices such as mobile antennas.

I am achieving the contactless measurement through using IR (infrared) emitter-detector circuitry. The IR pair generates pulses, which are galvanized by the rotation of the shaft. Slots on a disk mounted to the shaft permits light transmission between IR emitter and detector, which are placed on both sides of the disk. Hence, when the disk rotates it is possible to get stream of pulses from the circuitry.

I used a PIC microcontroller to accept the pulses and obtain the angular speed in terms of rps. This value is then displayed on 7-segment LEDs.

Lastly, this project involved a lot of microprocessor concepts. I tried my best to present concepts involved as comprehensible as possible. I expect the reader to have no difficulty while reading it.

TABLE OF CONTENTS

CERTIFICATION OF APPROVAL	ii
CERTIFICATION OF ORIGINALITY	iii
ACKNOWLEDGEMENTS	iv
ABSTRACT	v
CHAPTER 1:INTRODUCTION	1
1.1 Problem Statement	2
1.2 Objectives and Scope of Study	3
1.3 Literature Review	3
1.3.1 The Design Overview	3
1.3.2 PIC16F84 and its Architecture	5
1.3.3 PIC16F84 Interfacing	8
1.3.4 Alternative Methods	8
CHAPTER 2: METHODOLOGY AND SYSTEM DESCRIPTION	11
2.1 Methodology	11
2.2 The System Diagram and Process Flow Description	12
CHAPTER 3: DISCUSSION AND RESULTS	16
CHAPTER 4: CONCLUSION AND RECOMMENDATIONS	20
REFERENCES	21
APPENDIX A: ASSEMBLY SOURCE CODES	22
APPENDIX B: GANNT CHART FOR FYP, SEMESTER 2	28

LIST OF FIGURES

		Page
Figure 1	The system components	3
Figure 2	IR emitter-detector circuitry	4
Figure 3	Simplified internal layout of PIC16F84	5
Figure 4	The PIC16F84 pins	6
Figure 5	The PIC16F84 interfacing	8
Figure 6	Block diagram of the system	12
Figure 7	The system flow diagram	13
Figure 8	Interrupt Service Routine Flow Diagram	15
Figure 9	IR emitter-detector circuitry	17
Figure 10	PIC16F84 inputs and outputs	18

LIST OF TABLES

Table 1	Pin description for PIC16F84	6
Table 2	The comparison of possible methods	9
Table 3	Design components and specifications	16

TERMINOLOGY

There are number of technical acronyms that are used quite frequently throughout this report. Therefore I provided those words below with their explanations so as to use them in the rest of the report without explanations.

FYP	: Final Year Project
PIC	: Programmable microcontroller from Microchip®
PIC 16F84	: PIC model used in this project
CPU	: Central Processing Unit
RAM	: Read-Only Memory
ISR	: Interrupt Service Routine
I/O	: Input/Output
LED	: Light Emitting Diode
IR	: Infrared
rps	: Rounds per second (shaft speed)

CHAPTER 1

INTRODUCTION

In this project, I study and implement the contactless measurement of angular speed of a rotating shaft. There are plenty of methods available to measure a shaft speed. I achieve this by implementing IR emitter - detector circuitry.

It is the known fact that anything attached, other than the original load to a machine shaft introduces extra load. This leads to certain degree of complexities when trying to control the speed, which in case results in unpredictable level of stability in the machines' operation that we obviously do not want. The importance of accuracy even unveils if the design is of rather scientific nature such as mobile dish antennas, microchip robotics and the like. These fields require extensive precision in the contactless speed measurement.

In this project, I am designing a digital system, whose input is train of pulses and output is the speed of the shaft displayed on LED devices. In order to achieve the project objective the system is required to be able to perform the following functions:

- Count received pulses.
- Extract speed (in rps).
- Send the result to output devices.

The system should perform the following functions sequentially and continuously until the power supply is cut. Now, extracting speed from received pulses requires a mathematical calculation to be performed, which depends on the number of pulses and number of slots present on the disk. Therefore, the system is expected to be able to perform this calculation to obtain the speed.

Calculating for the speed:

- p : number of pulses received.
- t : time elapsed (in seconds).
- n : number of slots on the disk.
- Rps : speed in rps

$$Rps = p / (t n)$$

Although, it is rather a simple calculation, it needs a dedicated environment to correctly process it. This leads to the selection of correct device, which can carry out this operation and the rest of the tasks as well. With no doubt, only a processor can perform the above mentioned tasks. Since it is a simple calculation, a simpler processor would be the best device chosen for this project. I chose PIC 16F84 device to be my processor for the design. It is an 18-pin Enhanced Flash/EEPROM 8-bit microcontroller.

The project now has a microcontroller that needs to be programmed in order to execute tasks in the certain order. Therefore, the design needs both hardware and software to be developed.

1.1 PROBLEM STATEMENT

We can measure the speed of a revolving shaft of a machine using tachometer. However, the machine gets loaded due to the pressure applied. This may change the speed of the machine. An alternative method to this is to mount a lightweight slotted disk on the shaft. A light source may be placed on one side of the slotted disk. The revolving slotted disk will interrupt the light. Thus, a train of light pulses are produced on the other side of the slotted disk. The number of pulses received per minute is therefore function of the speed of the revolving disk and the number of slots on the disk. We can develop a system, where the pulses of light drive a light dependent transducer on the other side of the slotted disk to activate a counting circuitry. The counter counts the number of pulses per minute, determines the speed (rps) and displays it on a display device.

1.2 OBJECTIVES AND SCOPE OF STUDY

The objectives of this project are as stated below:

- To investigate the contactless measurement of a machine speed.
- To keep the project cost at minimum.
- To complete all project work within two semesters.
- To achieve the objectives of this course.

The scope of the project:

- To identify and locate all project hardware.
- To develop both hardware design and software program.
- To construct a design, that demonstrates the theory of the project.

1.3 LITERATURE REVIEW

1.3.1 The Design Overview

The system's input is train of pulses and output is digital signals to output devices. Therefore, the design is made up of three main components; input generator, input receiver and manipulator, and an output device (**Figure 1**).

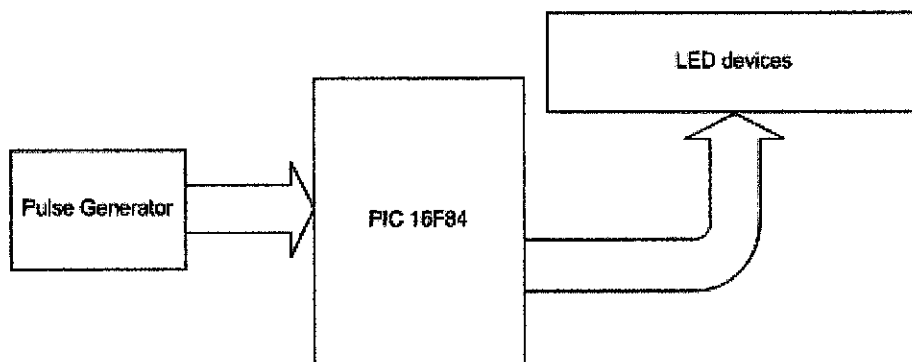


Figure 1: The system components.

The input to the system is train of pulses generated from an IR emitter-detector circuitry. The simplest form of the circuitry could be built as shown in **Figure 2**.

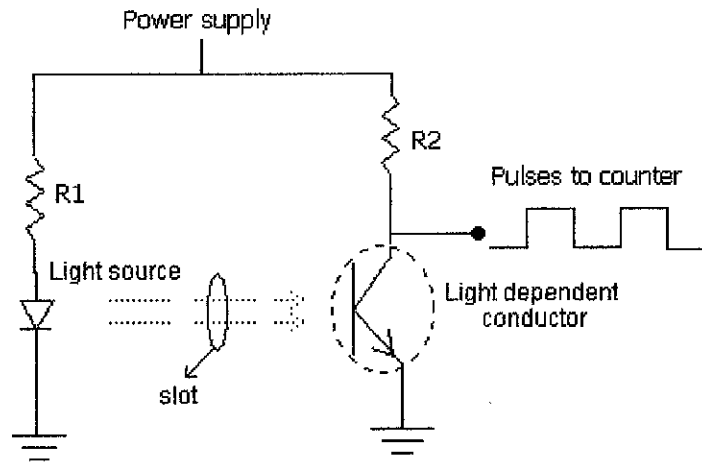


Figure 2: IR emitter-detector circuitry.

The IR light is emitted from the light source and detected by the light-dependent conductor. The transmission of the IR light is interrupted by the revolving motion of the light disk (or fan blades) for the light can reach the detector when the slots are positioned in between the IR emitter and detector.

The input receiver and manipulator is a device, which can accept pulses and perform calculations. The PIC microcontroller device is used as the input receiver and manipulator. It provides all necessary elements required by the design.

The output from the PIC is digital signals to be sent to an output device. Since the output is in terms of digits, the output device can be 7-segment LEDs (each representing one digit).

1.3.2 PIC16F84 and its Architecture

The PIC16F84 belongs to the mid-range family of the MICROCHIP®'s microcontroller family. A simplified PIC16F84 internal layout is shown in **Figure 3**. It is equipped with all features that my design needs. They are as listed below:

- High performance CPU and its working environment (registers and memory)
- External Interrupt handler
- Timer/Counter module
- I/O terminals
- Programmable in assembly language

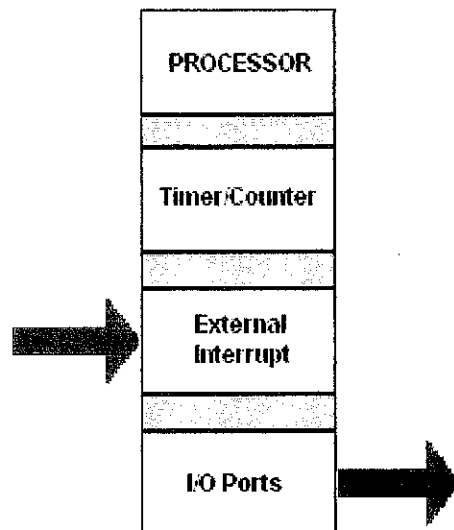


Figure 3: Simplified internal layout of PIC16F84.

The PIC16F84 microcontroller has a processor to manipulate data, an external interrupt handler to receive pulses, a Timer0 (timer/counter) module to either count pulses generated internally or externally. It is interfaced to other devices through its I/O terminals. The Timer0 (timer/counter) module requires to be correctly set through programming to have it work according to the desired specifications. It can be used to provide a time delay. The I/O terminals are entirely programmable and the direction of data is configured through programming. The programming language used is **Assembly language** (C can be used as well).

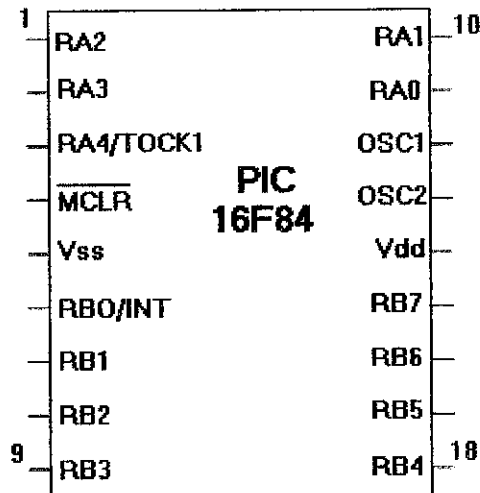


Figure 4: The PIC16F84 pins.

Table 1: Pin description for PIC16F84.

Pin No.	Pin Name	Description
1	PORTA pin 2	PORTA is a bi-directional I/O port
2	PORTA pin 3	
3	PORTA pin 4	
4	MCLR'	Master Clear (Reset) input/programming voltage input.
5	Ground	Ground reference
6	PORTB pin 0	PORTB is a bi-directional I/O port
7	PORTB pin 1	
8	PORTB pin 2	
9	PORTB pin 3	
10	PORTA pin 1	PORTA is a bi-directional I/O port
11	PORTA pin 0	
12	OSC1/CLKIN	Oscillator crystal input/external clock source input
13	OSC2/CLKOUT	Oscillator crystal output. Connects to crystal or resonator.
14	Positive supply	Positive supply
15	PORTB pin 7	PORTB is a bi-directional I/O port
16	PORTB pin 6	
19	PORTB pin 5	
18	PORTB pin 4	

There are lots of features that PIC16F84 possesses. These features make it so flexible that not only they are used in small scale designs but also in many branches of the industry today. Some of important PIC16F84 features are listed below:

- High performance RISC (Reduced Instruction Set Computer) CPU.
- Only 35 single word instruction set.
- All instructions take single-cycle (except for program branches).
- Operating speed: DC – 20MHz clock input, DC – 200 ns instruction cycle.
- 1024 words of program memory.
- 68 bytes of Data RAM.
- 64 bytes of Data EEPROM.
- 14-bit wide instruction words, 8-bit wide data bytes.
- 15 Special Function Hardware registers.
- 8-level deep hardware stack.
- Direct, indirect and relative addressing modes.
- 4 interrupt sources (External RB0/INT pin, TMR0 timer overflow, PORTB<4:7> interrupt-on-change, Data EEPROM write complete).
- 13 I/O pins with individual direction control.
- High current sink/source:
 - 25 mA sink maximum per pin.
 - 25 mA source maximum per pin.
- TMR0 module: 8-bit timer/counter with 8-bit programmable prescaler.

1.3.3 PIC16F84 Interfacing

The PIC16F84 is interfaced to other devices through its I/O terminals. It requires appropriate power supply, grounding and an external clock signal to function properly. For the clarity reason I omitted those inputs to the microcontroller (See **Figure 5**). The LED and the pulse generator also require proper power supply and grounding based on their specifications.

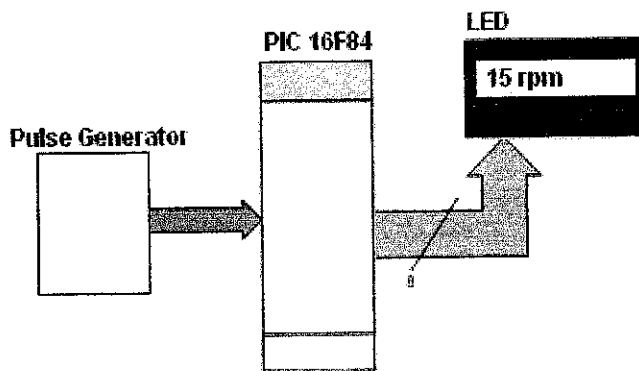


Figure 5: The PIC16F84 interfacing.

When other devices are interfaced to the PIC16F84 through its I/O terminals, they must maintain appropriate voltage and current levels that the PIC16F84 can handle (Refer to the features).

1.3.4 Alternative Methods

There are several methods to obtain the speed from received pulses. Although the methods vary only in how the PIC receives signals and determines the speed, the design itself remains essentially the same for all methods. Next are the possible methods and advantages/disadvantages over one another.

Table 2: The comparison of possible methods.

Method Description	Advantages	Disadvantages
<p><i>Using TMR0 module:</i> We can use the TMR0 module to accept external pulses. It generates overflow interrupt when reaches overflow.</p>	<p>It can be programmed to be as an external counter, which generates overflow interrupt.</p>	<p>It has a fixed prescaler (operates like a sampler) that increase in two's powers such as 1:2, 1:4, 1:8. For odd slot numbered disks, this feature is useless.</p> <p>Since this method used TMR0 module itself, we have to write extra code to provide time delay.</p>
<p><i>Using polling method:</i> In this method we can accept pulses in any pin and try to detect changes by writing an appropriate code for it.</p>	<p>We can use the nearest pin.</p>	<p>This method requires extra complex coding to detect the pin-state change.</p>
<p><i>Using PORTB pin-state change interrupt:</i> PORTB most significant pins raise an interrupt when their state-change from high to low or vice versa. We can use this feature to recognize pin-state changes.</p>	<p>PIC can easily detect state changes.</p>	<p>PIC requires checking whether a high state or low state is the current state.</p>

<p><i>Using RB0/INT Interrupt:</i> The RB0/INT pin generates an interrupt when it receives an external signal in appropriate level.</p>	<p>In this method, the PIC processes each signal individually. Thus, there is no need for performing calculations. In this mode, the PIC is totally flexible to be modified to cater any disk type. TMR0 can provide time delays.</p>	<p>This method introduces more interrupts thus slowing down the overall performance of the PIC.</p>
---	---	---

Among the alternative methods, the best method would be implementing RB0/INT method since it provides more flexibility than others and has no difficulty with performing calculations. This method treats each signal individually. Actually, this method eliminates most calculations by only checking boundaries of variables every time an interrupt is raised. Here, TMR0 module can be programmed to provide required time delays through TMR0 overflow interrupt.

Other methods all require calculations. The PIC16F84 microcontrollers are not good at performing calculations due to their limited instruction set. They can only add and subtract to provide the very basic needs.

CHAPTER 2

METHODOLOGY AND SYSTEM DESCRIPTION

This section discusses about the entire system design. It discusses about the design components and process flow. Through the system block diagram it explains how the system components interact with each other to achieve the common goal set.

2.1 METHODOLOGY

The overall system design is built and components are identified. The function of each component is well defined. Their interaction with each other is explained in the next section.

The system is ready to accept inputs, which are external pulses, after a successful initialization. It performs calculations and boundary checking, based on the functions it was programmed, on the received input and sends the result to the output devices, which in this case are 7-segment LEDs. The system is required to repeat the same flow continuously until it is interrupted.

2.2 THE SYSTEM DIAGRAM AND PROCESS FLOW DESCRIPTION

This section discusses the entire design at and explains how the system works.

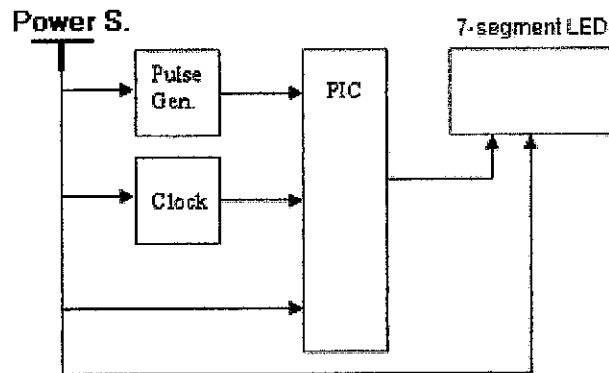


Figure 6: Block diagram of the system

In **Figure 6**, the PIC, the LED and the clock (crystal oscillator) are powered by a proper power supply. Since all the three components require around +5 Vdc, we can supply power to them through same source. The clock component generates fine frequency required for the PIC16F84 to work. Its frequency is 4MHz.

The output of the pulse generator is the input to the PIC, which is train of pulses generated from IR emitter-detector circuitry (depicted as Pulse Gen block in **Figure 6**). The External Interrupt handler receives the pulses and counts them. This value is then passed to the PIC processor to perform further calculations. The Counter/Timer module in the PIC16F84 is programmed to count the internal clock cycles and provide timely delays. The processor uses this time delays as a measure of time period intervals to determine the rps value. Finally, the output as a speed in rps is displayed on the display component. The output is sent in terms of digital signals through I/O terminals to LEDs.

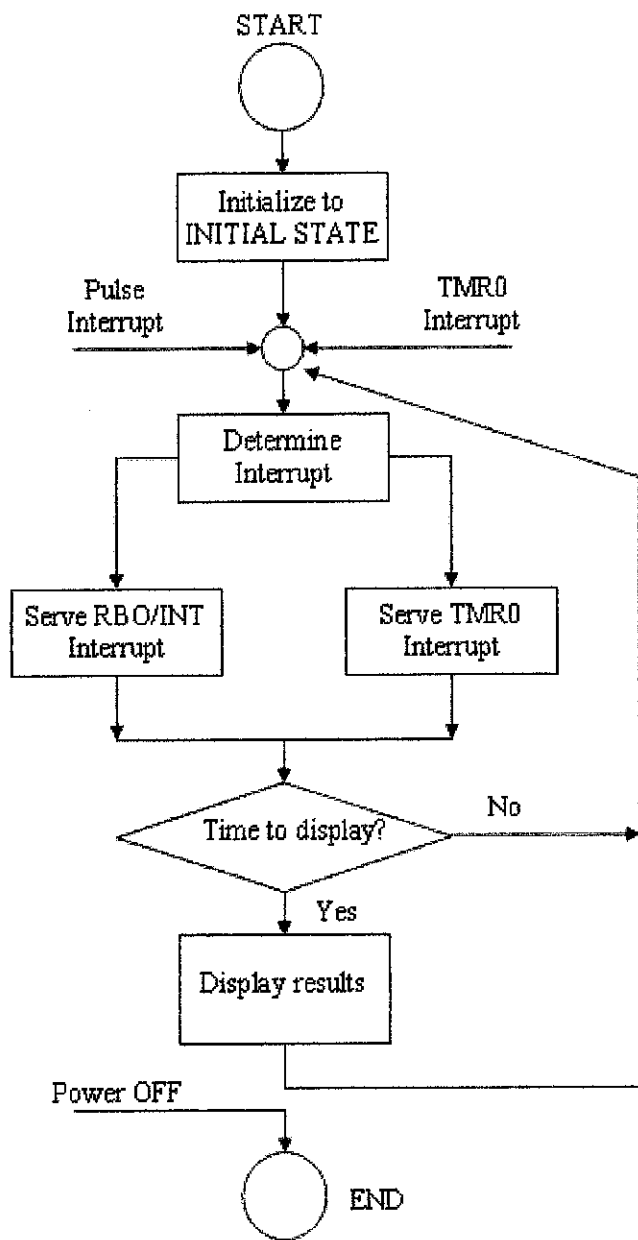


Figure 7: The system flow diagram.

We can see the whole system flow in **Figure 7**. When it is initialized it firstly sets its configurations to the Initial State. It configures its both PORTA and PORTB data directions, arranges interrupt handlings and sets TMR0 module to the proper configuration. Once the initial settings are successfully reached, it is ready to accept inputs and serve for interrupts.

When any interrupt is raised it first attempts to recognize the interrupt originator by checking its status register. Then it serves the correct ISR (Interrupt Service Routine) as shown in **Figure 8**. Next, it checks if it is about time to display results. If the time has elapsed, it displays the results on the LEDs. As we can observe in the system flow diagram (**Figure 7**), the system is not required to halt on itself. It continuously listens for the interrupts and serves them. It only stops when the power supply is cut off. We can imagine this as if the power supply is connected to the system through a push button. When pushed on it activates and when off it halts.

The system enters to an ISR whenever an interrupt is raised. It first determines the source of the interrupt as was mentioned above. We can see the entire ISR subroutine in **Figure 8**. It shows exactly what path in the ISR the system follows and what procedures it performs. If the interrupt is INT interrupt, it means it just received an external pulse. The INT ISR is executed, which is the left path in ISR. If TMR0 interrupt raises, it follows the right path in ISR. The details of each route are depicted in **Figure 8**. Once an interrupt is served, it returns to the main program where it listens for the next interrupt.

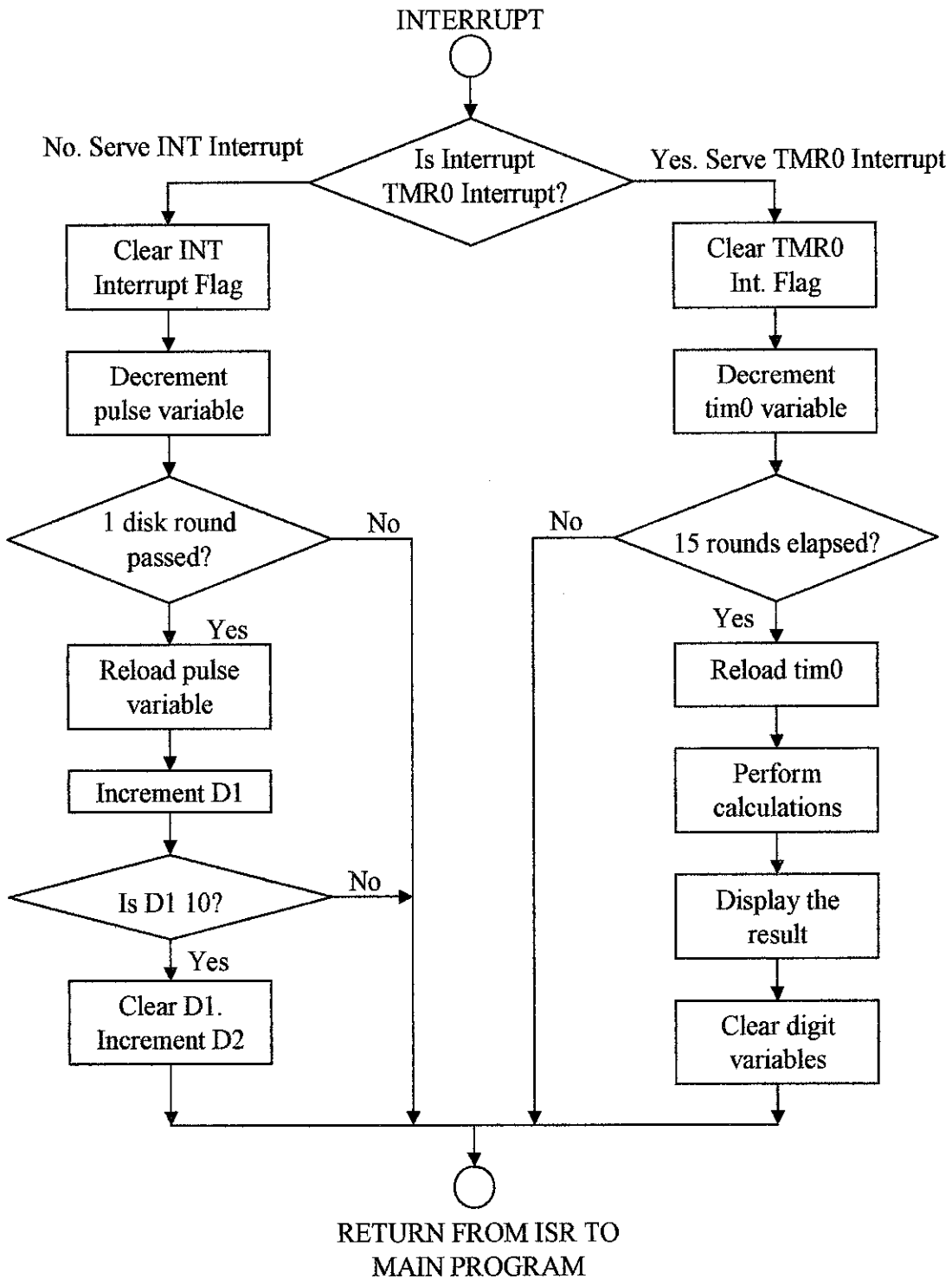


Figure 8: Interrupt Service Routine Flow Diagram.

The design scope is to demonstrate contactless measurement of angular speed. This design is developed to display two-digit speed values, which is up to 99 rps.

CHAPTER 3

DISCUSSION AND RESULTS

The system was designed to achieve the objective of the project. The components have been identified and design has been completed. Please refer to **Table 3** for design components and specifications.

Table 3: Design components and specifications

Component	Specification	Quantity
PIC	PIC16F84 Microchip®	1
LED	7-segment LED	5
LED decoder	Model 74LS4x	2
A motor with slotted disk	7 slots	1
Clock pulse generator	4 MHz Crystal Oscillator	1
Battery	5VDC and 12VDC	2
Circuit board	-	1
IR emitter	$V_F(\text{max})=1.7\text{VDC}$, $V_R(\text{max})=5\text{VDC}$, $I_F(\text{max})=100\text{mA}$, $P_{\text{DISS}}=100\text{mW}$	1
IR detector	$V_R(\text{max})=60\text{VDC}$, $P_{\text{DISS}}=100\text{mW}$, Light current min = 6.5 μA , Light current max = 15 μA	1
Other auxiliary components	-	-

Last item in **Table 3**, which are auxiliary components, is actually the sum of all secondary components such as resistors and wires used.

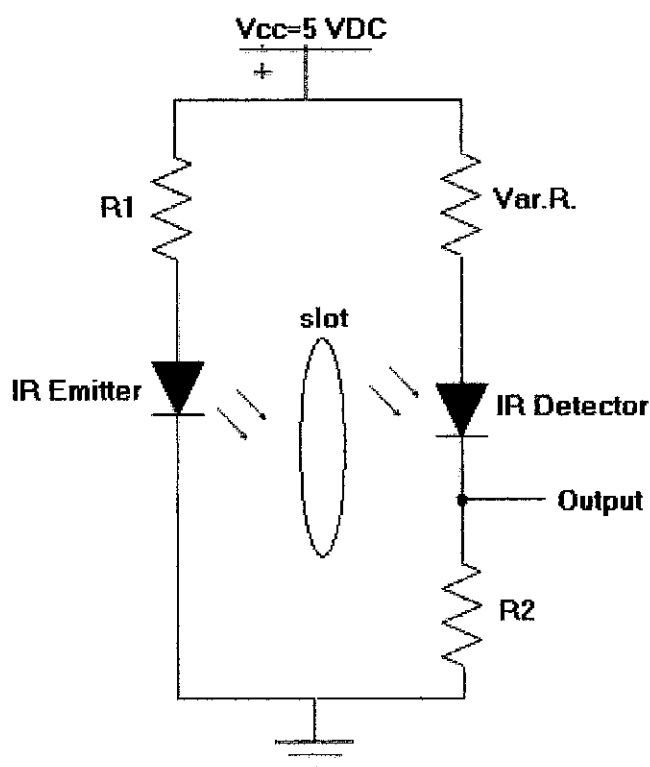


Figure 9: IR emitter-detector circuitry

Calculating for R1 and R2:

$$R1 = (V_{cc} - V_F) / I_F$$

Please refer to **Table 3** for IR emitter specifications.

$$R1 = (5 - 1.7) / 100\text{m} = 33 \Omega$$

Using the same formula:

$$R2 \text{ max} = (5 - 1.7) / 6.5\mu = 508 \Omega$$

$$R2 \text{ min} = (5 - 1.7) / 15 \mu = 220 \Omega$$

The Variable Resistor in **Figure 9** is to tune the output voltage level.

When the slotted disk rotates, it is possible to get pulse train form the output in the IR emitter-detector circuitry (as shown in **Figure 9**), which is then fed to the PIC16F84 through RBO/INT pin (pin 6).

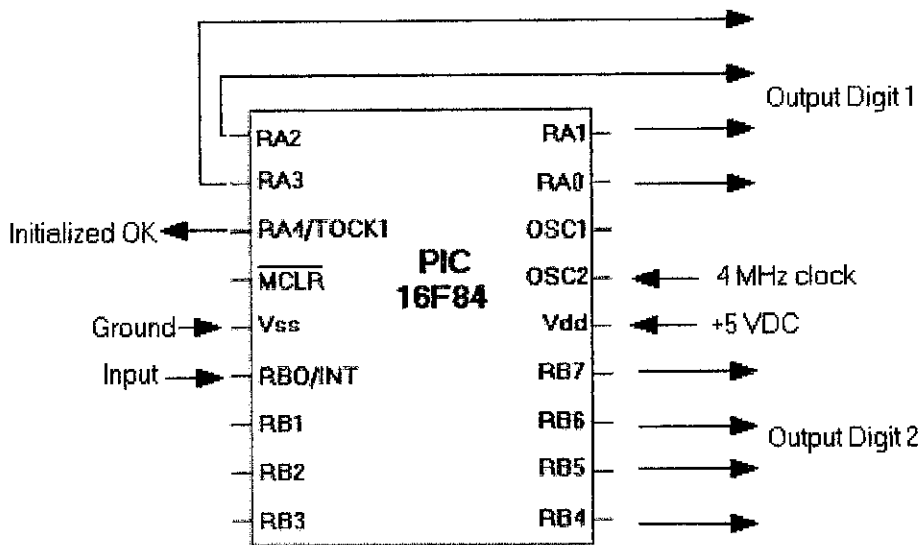


Figure 10: PIC16F84 inputs and outputs

Figure 10 shows all connections of the PIC16F84 and I/O directions. The output pulse from the IR circuitry is the input to the PIC16F84 through pin 6. The output is sent to two 7-segment LEDs (through LED decoders) through the PIC I/O ports. The first digit of the result is sent through the lower bits of PORTA and the second digit is sent through the upper bits of PORTB.

The output is refreshed twice in a second. This is the closest possible precision that the TMR0 module can provide. I got this by assigning 1:128 value to TMR0 prescaler. With this setting TMR0 provides 0.49152 second (about 0.5 second) time delay in fifteen (15) rounds. So every time a TMR0 overflow interrupt occurs, the PIC increments an internal variable to detect whether fifteen (15) rounds have elapsed.

The external clock for PIC16F84 is 4 MHz. The PIC16F84 divides this into four (4) parts that yields 1 MHz execution speed. Thus, each instruction execution cycle takes exactly 1 microsecond. I set the TMR0 to increment in every 128 instruction cycles. Since the TMR0 is an 8-bit register it can take values from 0 to 255 ($2^8 - 1$). This gives $128 \times 256 = 32768$ micro second time delay in one (1) round and 15 rounds will give $15 \times 32768 = 491520$ micro seconds (or 0.49152 second). The theoretical error is

1.696 %, which can be considered very acceptable for PIC devices. Actually, the practical error is rather smaller than this value. Because some branch executions during the program execution, which are not expressible, take time to further reduce the error.

This design is developed to display two-digit speed values. It can display up to 99 rps. We follow the same procedures to display more than two-digit values. Only, there will be simple modifications in the program. All it requires is adding more boundary-checking routines to INT ISR and allocating extra I/O pins for output. The PIC16F84 has thirteen (13) I/O pins. This should be kept in mind during design process. If more I/O pins are required different PIC should be used such as PIC16F62 or PIC16C74.

Initially the system used an LCD display as its output device. The source code was to support the LCD, since it needed proper programming to function. I could not locate in second semester so I used 7-segment LEDs to replace it. This required some alterations in the code and needed some extra components (LED decoders). The complete source code for the project is completed and is attached to appendices. The code is fully commented to explain what the lines perform.

CHAPTER 4

CONCLUSION AND RECOMMENDATIONS

The project period is two semesters. In the first semester, the design part was completed and all components were identified. The PIC block diagrams and program source code for the project are as well completed and simulated. The design was modified to cater the output device change in the second semester.

The objectives of the project are achieved. Both the hardware and software parts for the project are developed and tested with a sample code. There were some challenges faced during the project period. The code development, PIC troubleshooting, components integration and tuning could be considered as some of them.

The project objectives and scope were relatively clear. The design was not so complicated once the components were allocated. That is the reason I chose PIC16F84, which is one of the simplest PIC family to use. If a more complex design is to be developed, I would recommend Motorola® 68000 processor. It has more complex instruction set and powerful addressing modes.

My experience from previous subjects and my supervisor's advices were very useful in completing this project. I hope I accomplished the objectives of this subject. Lastly, this particular project involved many microcontroller concepts. I did my best to present them as clearly as possible so that the reader should have no difficulty while examining it.

REFERENCES

1. [MCHP99] MICROCHIP®, 1999, “Datasheet for PIC16F84”
2. Design with PIC Microcontrollers, 1999, John B. Peatman, Prentice Hall
3. PIC microcontrollers for beginners, MICROCHIP®
4. Microprocessor II EEB5303 Lecture Notes, Semester July-2003
5. From the Web:
 - <http://www.microchip.com>
 - <http://www.boondog.com>
 - <http://www.maxmon.com>
 - Other personal sites.

APPENDICES

A. SOURCE CODE

```
LIST P=PIC16F84 ; Use PIC 16F84
; radix hex ; Gives warning: Radix superceded by command line.
; #include "P16F84.inc" ; Include this file
; _CONFIG_CP_OFF&_WDT_OFF&_PWRTE_ON&_XT_OSC
; Code-protect off, no watchdog, uses XT oscillator

w equ H'0000'
f equ H'0001'

;---- Register Files-----
porta equ 0x05
portb equ 0x06
trisa equ 0x85
trisb equ 0x86

timer0 equ 0x01
regSTAT equ 0x03
intcon equ 0x0B
regOPT equ 0x81

;---- STATUS Bits -----
RPO equ H'0005' ; regSTAT REG bits
Z equ H'0002' ; Zero bit
C equ H'0000' ; Carry/'Borrow bit

;---- INTCON Bits -----
GIE equ H'0007' ; Global Interrupt Enable bit
TOIE equ H'0005' ; Timer0 Overflow Interrupt Enable bit
INTE equ H'0004' ; RBO/INT External Interrupt Enable bit
TOIF equ H'0002' ; Timer0 Overflow Interrupt Flag bit
INTF equ H'0001' ; RBO/INT External Interrupt Flag bit

;---- My Variables -----
W_temp equ 0x20 ; Holds temporary W content
S_temp equ 0x21 ; Holds temporary regSTAT register's content

d1 equ 0x22 ; Holds 1st digit to be displayed
d2 equ 0x23 ; Holds 2nd digit to be displayed
d3 equ 0x24 ; Holds 3rd digit to be displayed
tim0 equ 0x25 ; Timer0 cycle counter
pulse equ 0x26 ; Holds pulse number which represents 1 round
```

```

;----- Define Macros -----
Bank0 macro
    bcf    0x03, RPO
    endm

Bank1 macro
    bsf    0x03, RPO
    endm

Push  macro                ; Important: All macro operations happend in Bank0
    movwf W_temp
    swapf 0x03, w
    movwf S_temp
    endm

Pop   macro
    swapf S_temp,    w
    movwf 0x03
    swapf W_temp,    f
    swapf W_temp,    w
    endm

INTen macro
    movlw B'10110000' ; Set Intcon with binary value 10110000
    movwf intcon      ; GIE enabled, TOIE enabled, INTE enabled
    endm              ; All flags cleared
;-----
    org    0x00
    goto   MAIN

    org    0x04
    goto   ISR

MAIN Bank0                ; Start at Bank 0
    movlw 0x00             ; Clear W
    bcf   intcon, GIE     ; Disable interrupts until everything is set. Wait until it
displays 000
    clrf  porta           ; Clear buffers
    clrf  portb
    clrf  d1              ; Initially d1, d2 and d3 are 0 (zero)
    clrf  d2
    clrf  d3
    movlw 0x0F            ; Load tim0 with decimal 15
    movwf tim0
    movlw 0x07           ; Load pulse with decimal 7
    movwf pulse

```

```

Bank1          ; Switch to 1st bank
clrf trisa     ; Set PortA bits all output
movlw 0x01     ; Set PortB bits all output except RB0/INT pin
movwf trisb

movlw B'11000110' ; Set Timer0 (Prescaler 1:128. Gives 0.49152s when 15
times occurs )
movwf regOPT

Bank0
INTen         ; Set and enable Interrupts

bsf porta, 4  ; Initialization successful

_wait goto _wait ; Hang program for interrupts

;----- Interrupt Service Routine (ISR) -----
ISR bcf intcon, GIE ; Macro to disable interrupts
Push ; Macro to save the "environment"
;-----

btfss intcon, INTF ; Determine which interrupt has just occurred
goto _sTMR0 ; Timer0 Overflow Interrupt has occurred
; INT Interrupt has occurred
_sINT bcf intcon, INTF ; Clear INTF to get the next interrupt

decf pulse, f
btfss 0x03, Z ; Test if 1 full round is passed? 7 pulses represent a round.
goto _pop ; No, the round is incomplete
; Yes, 1 round has passed

movlw 0x07 ; Reload pulse with decimal 7
movwf pulse

;-----
incf d1, f ; Increment d1
movf d1, w
xorlw 0x0A ; Is it 10?
btfss 0x03, Z ; Check if the result is 0
goto _pop ; No it is not 10
; Yes d1 was 10
clrf d1 ; d1 is cleared to zero

incf d2, f ; Increment d2
;movf d2, w ; d3 is disabled for now
;xorlw 0x0A
;btfss 0x03, Z

```



```

;goto _pop

;clrf d2
;incf d3 ; Increment d3
; Until here it can display up to 999

goto _pop ; End _sINT

_sTMR0 bcf intcon, T0IF ; Clear flag

decf tim0, f
btfss 0x03, Z ; Test if 15 cycles finished?
goto _pop ; No, 15 cycles not reached. Do nothing.

movlw 0x0F ; 1/2 seconds passed.
movwf tim0 ; Reload tim0 with decimal 15

;----- Perform calculations

movf d2, w ; Multiply d2 by 2
addwf d2, f

movf d1, w ; Multiply d1 by 2
addwf d1, f

movlw 0x0A ; Load W with decimal 10

subwf d1, w ; subtract 10 from d1. Save result in W

btfss 0x03, C ; Check the result in C bit
goto _done ; Result < 0. Do nothing

btfsc 0x03, Z ; Result = 0 OR Result > 0
goto _zero ; Result = 0

movlw d1 ; Result > 0
incf d2, f
goto _done

_zero clrf d1
incf d2, f

_done ; Calculations done

```

```

;----- Display results

    movf  d1,  w    ; Display results on LED (Output refreshed at every 1/2
seconds)
    movlw porta    ; Output d1 on porta lower bits

    swapf d2,  w    ; Swap upper 4 bits with lower ones and save in w (d2
remains intact)
    movlw portb    ; Output d2 on portb higher bits

    clrf  d1        ; Clear d1, d2
    clrf  d2

;----- Return from ISR with original environment

_pop  Pop          ; Macro to regain the original "environment" (context)
      INTen        ; Macro to enable interrupts back

      retfie       ; Return from ISR
;-----
      end

```

TEST CODE

```

LIST P=PIC16F84 ; Use PIC 16F84
; radix hex

w    equ  H'0000'
f    equ  H'0001'

porta equ  0x05
portb equ  0x06
trisa equ  0x85
trisb equ  0x86
regSTAT equ  0x03
RPO   equ  H'0005' ; regSTAT REG bits

Bank0 macro
    bcf  0x03, RPO
endm

Bank1 macro
    bsf  0x03, RPO

```

```

    endm

    org    0x00
    goto  MAIN

MAIN movlw 0x00      ; Clear W
    clrf  porta     ; Clear buffers
    clrf  portb

    Bank1           ; Switch to 1st bank
    movlw 0xFF
    movwf trisa

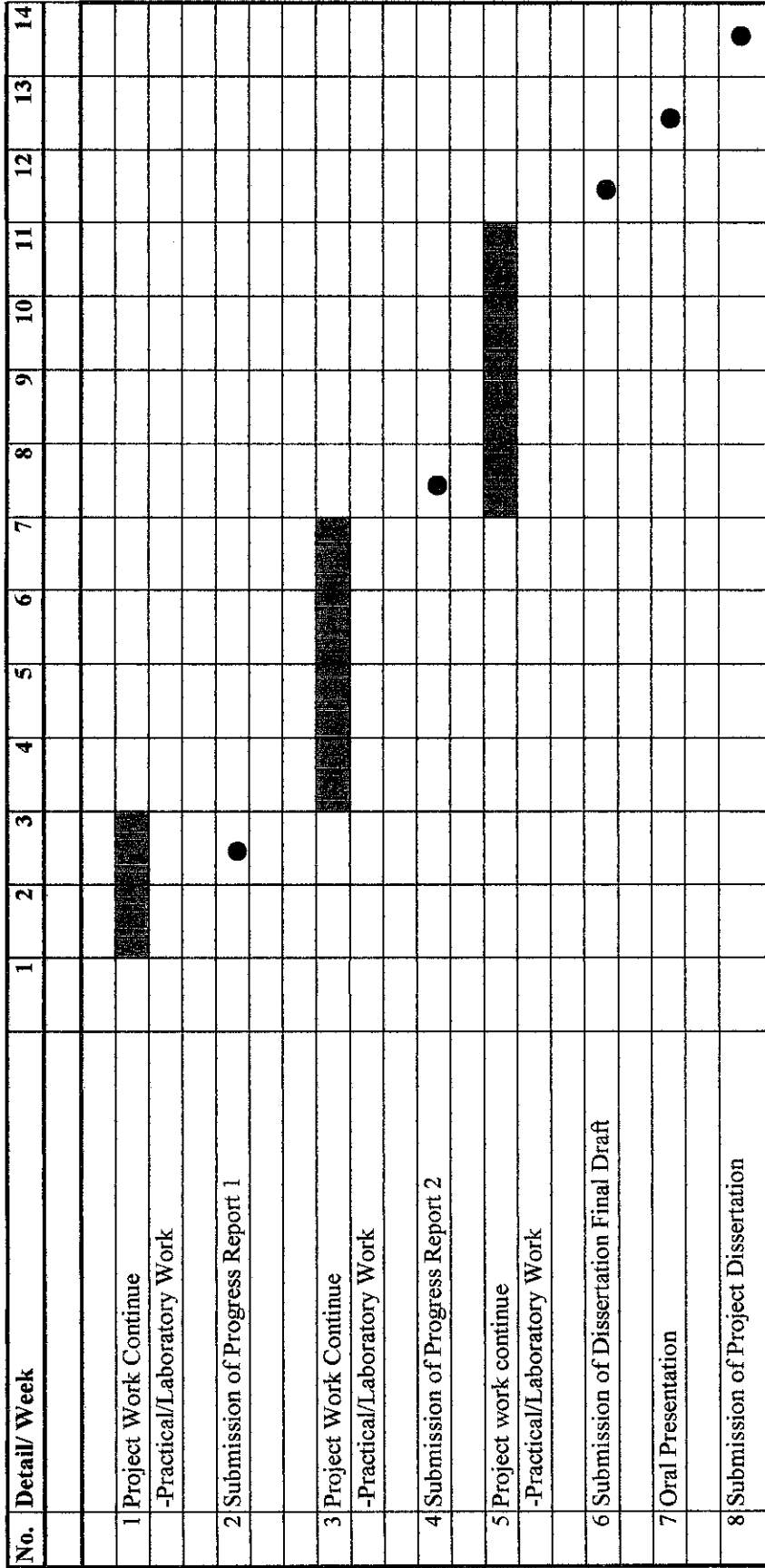
    clrf  trisb ;movwf    trisb

    Bank0
;//////////////////////Start Test
here  movf  porta, w
      movwf portb
      goto here
;//////////////////////End Test

    end

```

B. GANTT CHART



● Suggested milestone
 ■ Process