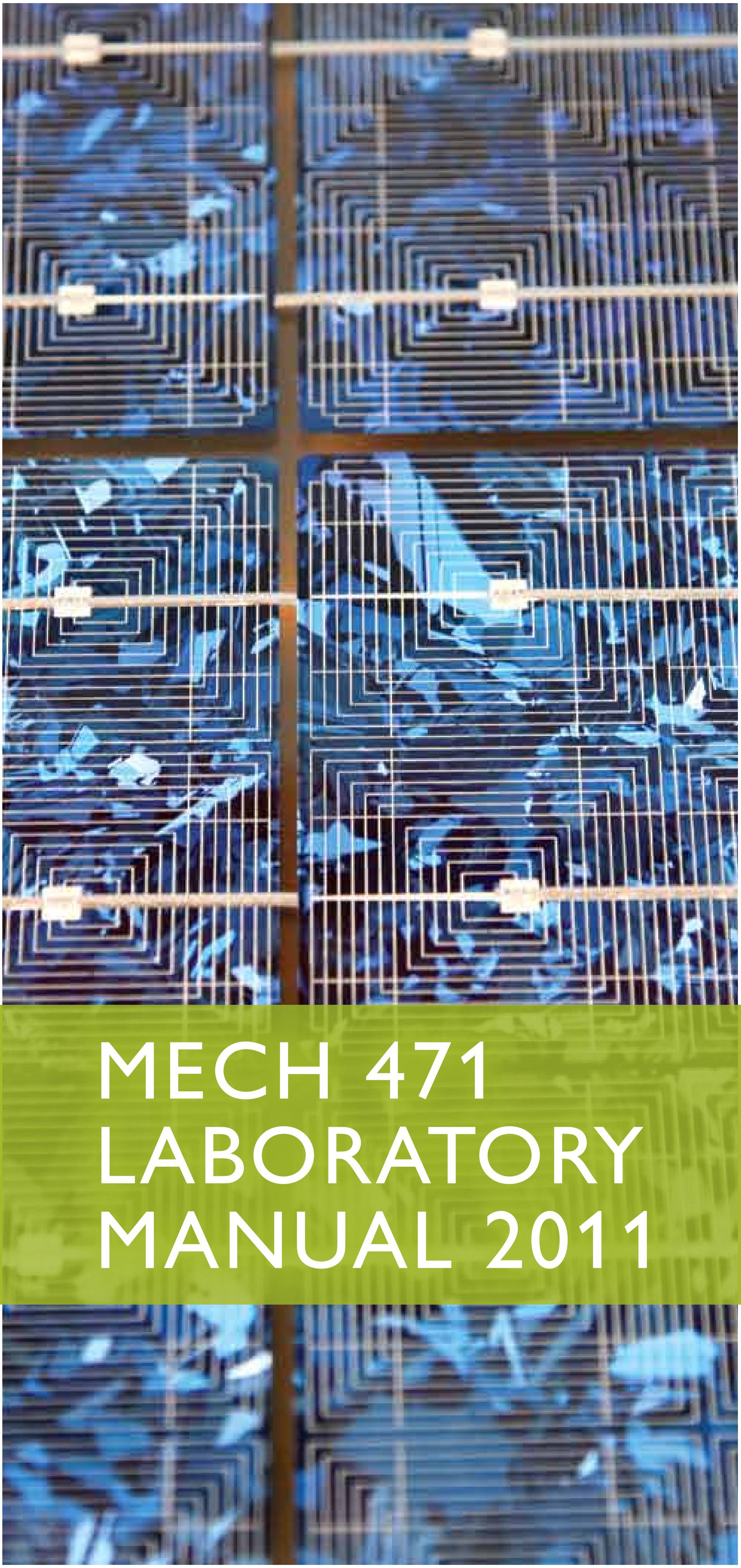


Concordia University

Engineering and Computer Science

Department of Mechanical and Industrial Engineering

MECH 471 LABORATORY MANUAL 2011

MICROCONTROLLERS FOR MECHATRONICS

BELAL M. IBRAHIM

Laboratory Specialist

GENERAL SAFETY RULES

Electric and electronic circuits can be dangerous. Safe practices are necessary to prevent electrical shock, fires, explosions, mechanical damage, and injuries resulting from the improper use of equipment.

Experienced people know that even moderate voltages may cause a serious shock (i.e. if the human skin is moist its resistance to the flow of electricity can drop drastically). They also know that so-called low-voltage equipment may have a high-voltage section. Therefore, never assume protective devices are working. Also, never assume a circuit is off even though the switch is in the "OFF" position. The switch could be defective.

As your knowledge and experience grows, you will learn many specific safety procedures for dealing with electricity and electronics. In the meantime:

- always follow procedures carefully;
- investigate everything before you act;
- when in doubt, do not act. Ask your instructor or supervisor.

Safe practices will protect you and your fellow students. Please do not hesitate to ask your instructor about anything that you are not sure of or do not understand. Many accidents occur when people rush into things and cut corners. Take the time required to protect yourself and others. Running, horseplay, and practical jokes are strictly forbidden in laboratories. Circuits and equipment must be treated with respect. Learn how they work and the proper way of working on them.

In case of an emergency use the internal phone to call security by dialing 3717. Security will connect you to the appropriate emergency service and immediately dispatch security personnel.

Or you can use your cellular phone to call 514.848.3717.

The civic address is:
Concordia University, 1455 De Maisonneuve West, H3G1M8
Room_?????_________

The technician responsible/ in charge of this laboratory

is: _????_________________________ tel:_???__________________________

Safety Regulations for Students in All Mechanical and Industrial Engineering Laboratories;

Standard lab safety must be followed in all laboratories.

a. First discuss your experiment regarding possible hazards or problems, with the demonstrator, or the MIE technical staff, or your professor.

b. Do not work alone. Work with another person in a lab that has running machinery, machine tools, conveyors, hydraulics, lifting equipment, voltage hazards, or where chemicals are in use.

c. Safety glasses must be worn in the vicinity of pneumatics, machine tools grinders, power saws, and drills.
Users of lasers need special safety glasses for the particular wavelength of the laser.

d. No equipment or machine may be operated by anyone unless they have received adequate instruction from a qualified instructor eg. machine tools, hydraulics, chemicals, lasers, running machinery, robots. Undergraduate students may not use any machine or equipment unless a Department technical staff member is present. Graduate students are the responsibility of their immediate academic supervisor.

e. Workplace Hazardous Material training must be obtained before using chemicals or compressed gasses. Contact Dainius Juras tel: 3128 for training.

f. All appropriate safety accessories (lab coats, safety glasses, gloves, etc..) must be used when handling chemicals. No open toe shoes are permitted in laboratories.

g. No chemicals to be left unattended or unlabeled according to WHMIS. All chemicals must be stored properly.

h. Long term unattended tests must be fail safe.

i. When the university is officially closed, you may not work in a lab unless your supervisor or a technical staff member is present.

j. No eating in laboratories.

k. Major accidents and injuries must be reported at once to Security tel: 3717 the Safety Officer (tel: 3128), the Professor (Supervisor) or the Department Administrator (tel: 7975) should then be informed.

l. During working hours all minor accidents should be reported to the Safety Officer (tel: 3128), the Professor (Supervisor) or the Department Administrator (tel: 7975).

m. An "Incident Report" must be filled out by the person involved, for all accidents and injuries.

LABORATORY RULES

Considering the large number of students attending the labs and in order for the lab to operate properly, the students are asked to abide by the following rules:

1. No eating or drinking is permitted in the laboratory.
2. Overcoats and briefcases are not permitted in the laboratory.
3. Students should bring their own laboratory manual.
4. No equipment is allowed to be exchanged from one bench to another.
5. Upon entering and when leaving the laboratory students should check equipment against the list posted at each station.
6. All damaged or missing equipment and cables must be reported immediately to the demonstrator. Failure to do so will result in students being charged for damages or losses.
7. Writing or taping on work benches will result in ejection from the laboratory.
8. All data must be recorded neatly in the laboratory on a clean piece of paper and must be signed by the demonstrator.
9. No more than three students are allowed to occupy one laboratory setup.
10. Any student who is more than 30 minutes late will not be permitted into the laboratory.
11. After your laboratory session is completed all components, connecting jumpers, and cables must be returned to their respective places.

SCOPE OF THE LABORATORY

The main purposes of microcontroller laboratory work are as follows:

- To provide experience in writing and developing assembly language program.
- To guide student on how to use the hardware features of the 8-bit MCUs.
- To provide practical experience in handling sensors, DC motors, and servo motor.
- To provide practical experience in Microcontroller interfacing techniques.
- To introduce Microchip development tool MPLAB IDE.

ORGANIZATION OF THIS MANUAL

This manual is divided into 5 sections, each section describing one experiment. Sections are broken down into parts as follows:

1. Objectives
2. Introduction
3. Experimental procedure
4. Optional work

The first part gives the objectives of the experiment. The second part provides a comprehensive introduction on how to do different part of the experiment. Relevant subroutines are often included in this part for the convenience of the student. The third part describes the experimental procedure to be adopted and is itself broken down into subsections. Some of these subsections indicate to the student how to connect and test a particular circuit. Other subsections require the student to carry out a number of preliminary calculations and preparation for the experiment. The fourth part gives an optional extra work relevant to the experiment to enhance the idea discussed or to widen the student's background and ability to add more applications.

EXECUTION OF THE EXPERIMENTS

Each experiment must be studied in advance and prepare a draft of the program. If the material is understood the student knows exactly what to expect in an experiment and a flawless program can be obtained very quickly. Although, some of the experiment procedure may not broken into steps to follow, the student must draw his own flowchart. It is a very good engineering practice to draw such chart during the advance reading of the experiment. In this way discrepancies can be immediately detected and checked.

Devices are invariably characterized with maximum voltage, current, and power ratings. These should never be exceeded. Otherwise, the properties of a device may be impaired. The datasheets of all devices that are being used in the experiment found in the Appendix section. If in doubt about the use of a particular instrument, the operating instructions provided by the manufacturer should be read. Defective equipment must be reported immediately.

Each group is required to work at the same bench location each week. Equipment and components must be returned to their places. The benches must be left clean at the end of the experiment.

Since the laboratory represents a significant portion of the student's practical training, it is imperative that the students perform all the experiments. If a student has missed an experiment due to circumstances entirely beyond his/her control, that student will have the opportunity to perform it at the end of the term. However, it is most unlikely that arrangements can be made for any individual to perform more than one experiment at this time.

THE LAB REPORT

For each experiment a lab report must be written which can be regarded as a record of all activities, observations and discussions pertaining to the experiment. Lab reports should above all be legible and should contain as much information as possible. A lab report should consist of papers stapled together with a title page identifying the course, experiment, date, student's name and demonstrator's name.

Each lab report should be divided into five parts as follows:

- Objectives
- Introduction
- Experimental results
- Your suggestions
- Conclusions

Objectives

They have to be stated clearly and can be copied from the lab manual.

Introduction

Should prepare the reader as to what is done during the experiment. It should be brief.

Experimental Results

Copy of your program as described later, a clear flowchart and graphs if any.

Your Suggestions

In this section write what you imagine about other applications or improvement of the existing one. You could briefly explain the idea with the help of the flowchart.

Conclusions

Should indicate success or failure and any difficulties you had in the experiment.

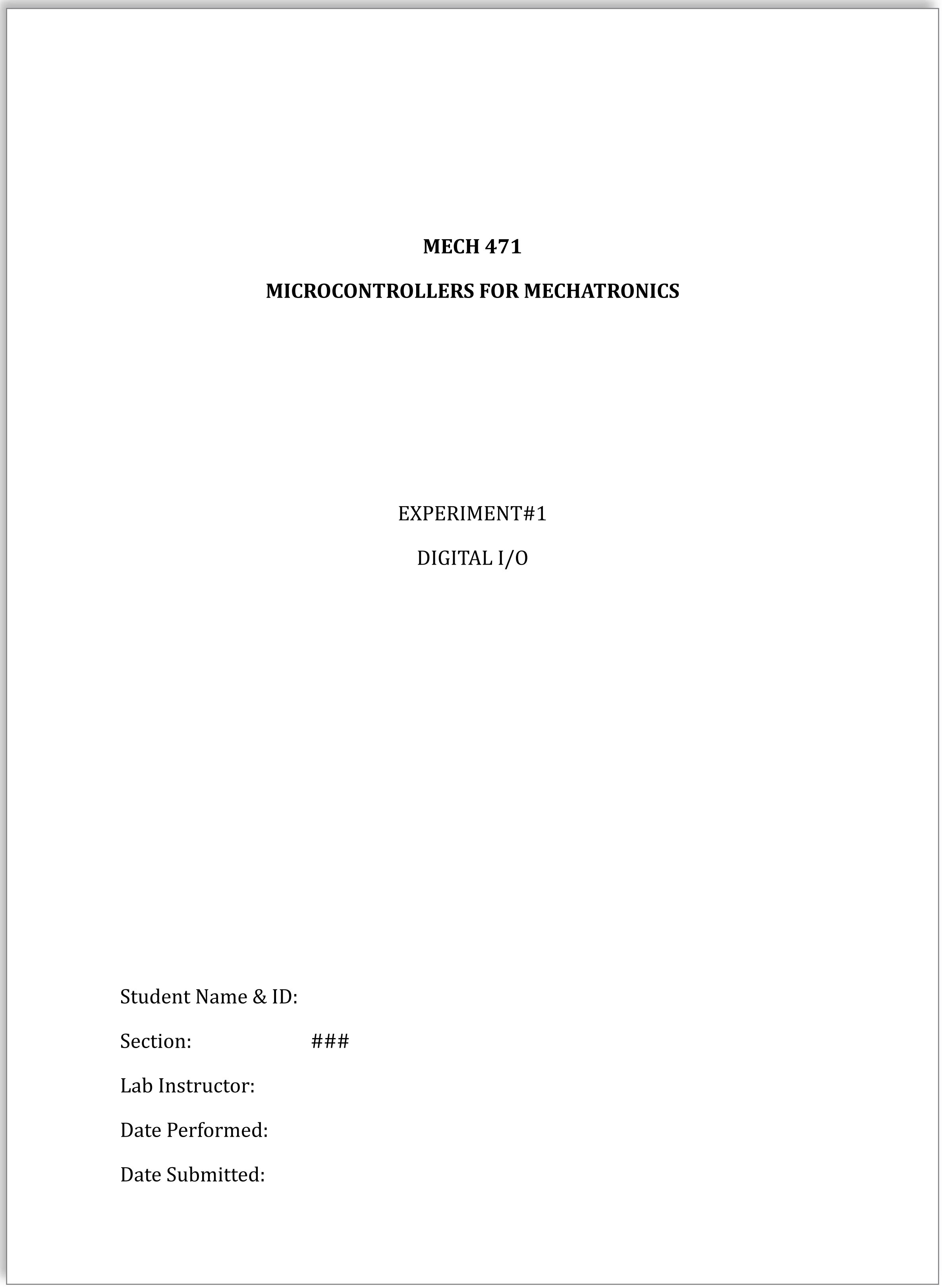
GRADING SCHEME

Laboratory reports for each experiment are to be submitted within two weeks, the same day that the experiment was performed. The report will be returned at the day that the next experiment. Each lab report will be marked out of ten. The grading scheme is as follows:

Objectives, introduction	10%
Results	50%
Suggestions and Conclusion	10%
Preparation and participation*	30%

**It is important that the student prepare for each experiment by reading the instructions before the student goes to the laboratory. Therefore, both the preparation and the participation will be evaluated during the laboratory.*

COVER PAGE SAMPLE



MECH 471

MICROCONTROLLERS FOR MECHATRONICS

EXPERIMENT#1

DIGITAL I/O

Student Name & ID:

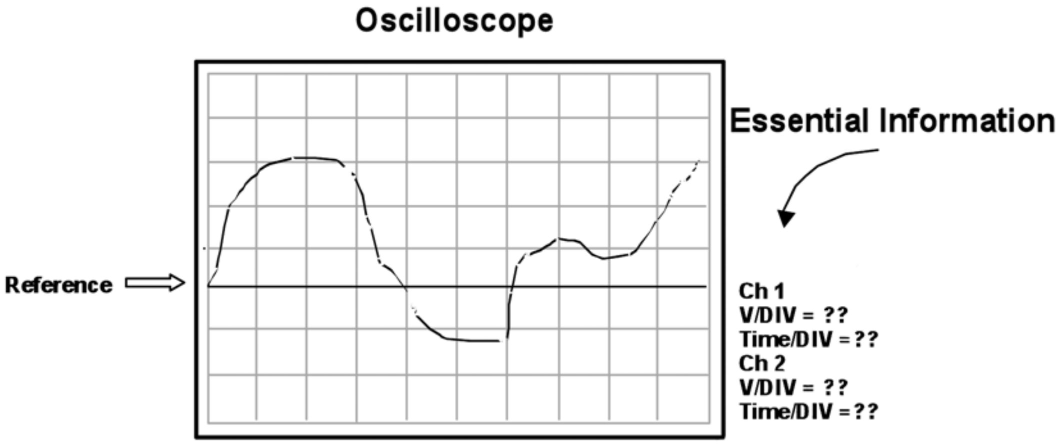
Section: ###

Lab Instructor:

Date Performed:

Date Submitted:

PRESENTATION OF THE GRAPHICAL RECORDS



Explain the above results.

....

....

....

....

....

....

PRESENTATION OF THE PROGRAM

Paste a copy of your program here with your name.

Divide your code into sections with proper header for each to indicating its functionality and the goal you want achieve.

Explain all sections above and show how you achieved program's goal.

....

....

....

....

....

Show how to improve the above code and suggest other applications for it.

....

....

....

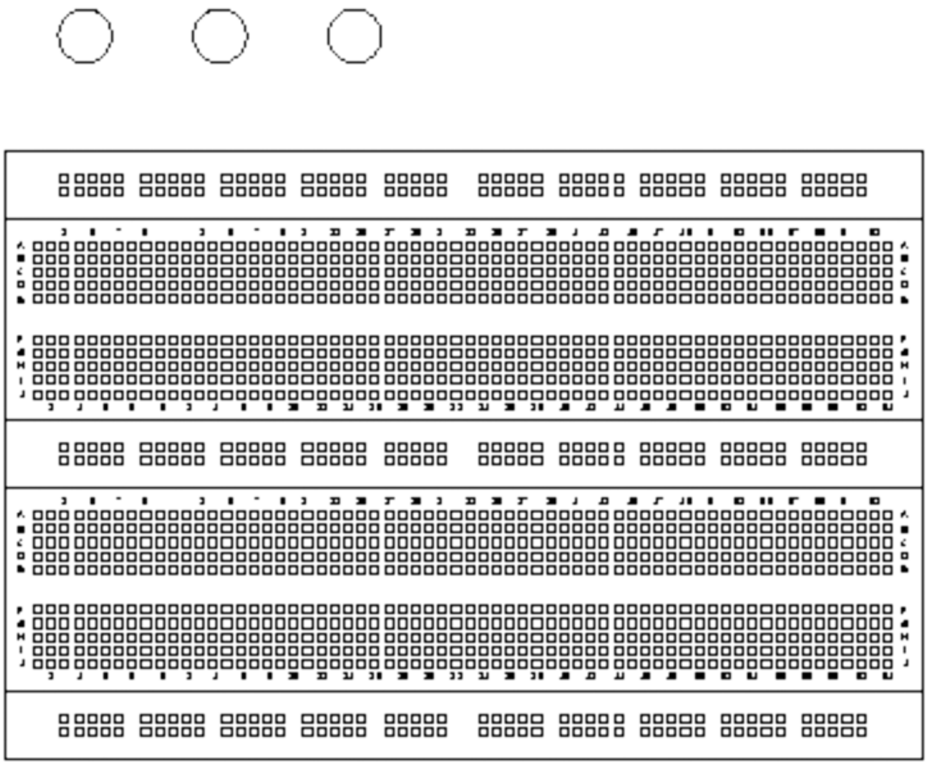
....

....

Please note that:

- The presentation of any step in the experimental procedure must be completed and followed by the necessary theoretical calculations and comments.
- The comment statement must indicate the correctness and the validation of the results through a meaningful discussion.

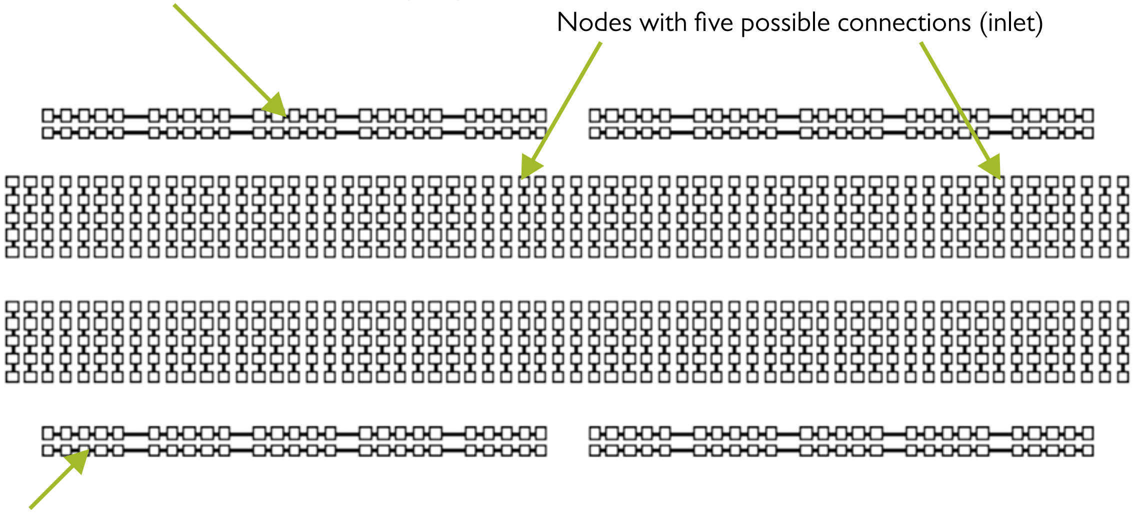
BREADBOARD



Breadboard external view

Bus line with 25 possible connections (inlet)

Nodes with five possible connections (inlet)

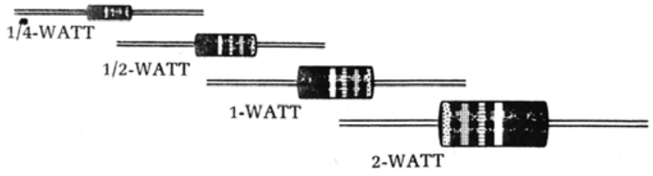


Bus line can be used as a large common node for the circuits under test, for example one bus line can be used for +Vcc and another can be used for –Vcc or Ground

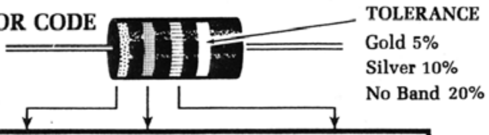
Breadboard internal metal connections of common bus lines and nodes

RESISTORS

Resistors come in several sizes and shapes, each one with its color code or value printed on it. The Manual calls out the value, and color code when used, of each resistor at the time it is installed.

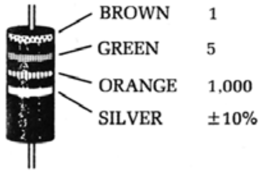


RESISTOR COLOR CODE

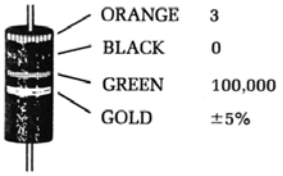


COLOR	1st DIGIT	2nd DIGIT	MULTIPLY BY
BLACK	0	0	1
BROWN	1	1	10
RED	2	2	100
ORANGE	3	3	1,000
YELLOW	4	4	10,000
GREEN	5	5	100,000
BLUE	6	6	1,000,000
VIOLET	7	7	10,000,000
GRAY	8	8	100,000,000
WHITE	9	9	1,000,000,000
GOLD			.1
SILVER			.01

EXAMPLES:



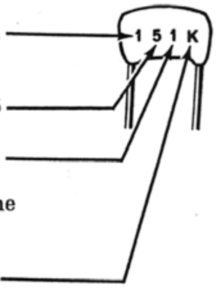
15 × 1,000 = 15,000 Ω (15,000 OHMS), or "15 k"



30 × 100,000 = 3,000,000 Ω (OR 3 MΩ)
3 MΩ = 3 MEGOHMS

CAPACITORS

Capacitors come in many sizes and types. The Manual will tell the type and value of each one, and show what it looks like. This page shows how you can read the code printed on some capacitors.



First digit of capacitor's value: 1

Second digit of capacitor's value: 5

Multiplier: Multiply the first & second digits by the proper value from the Multiplier Chart.

To find the tolerance of the capacitor, look up this letter in the Tolerance columns.

EXAMPLES:

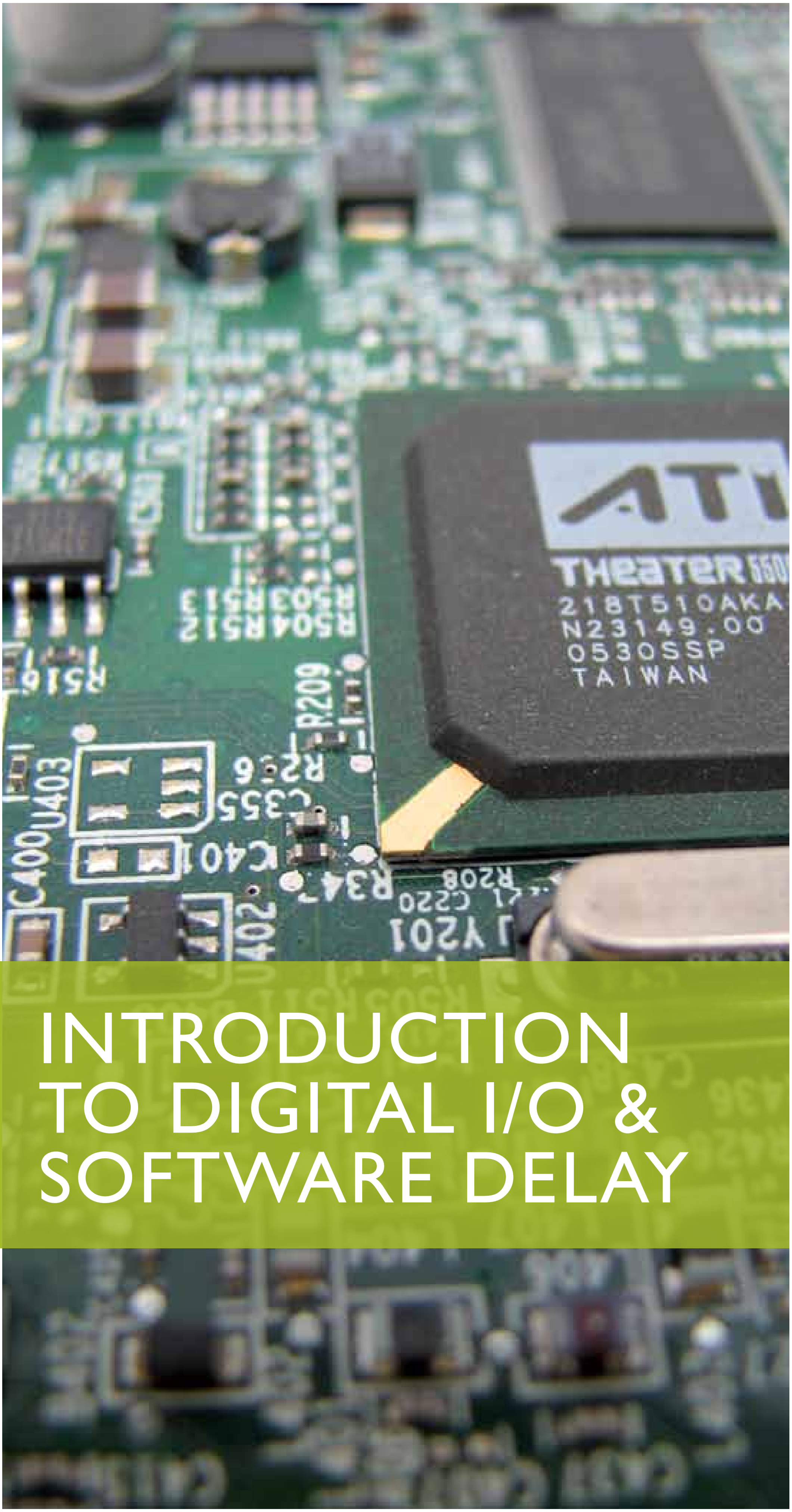
151K = 15 × 10 = 150 pF
759 = 75 × 0.1 = 7.5 pF

NOTE: The letter "R" may be used at times to signify a decimal point; as in: 2R2 = 2.2 (pF or μF).

pF = picofarads
μF = microfarads

MULTIPLIER		TOLERANCE OF CAPACITOR		
FOR THE NUMBER:	MULTIPLY BY:	10pF OR LESS	LETTER	OVER 10pF
0	1	±0.1pF	B	
1	10	±0.25pF	C	
2	100	±0.5pF	D	
3	1000	±1.0pF	F	±1%
4	10,000	±2.0pF	G	±2%
5	100,000		H	±3%
			J	±5%
8	0.01		K	±10%
9	0.1		M	±20%

TABLE OF CONTENTS



EXPERIMENT 1

INTRODUCTION TO DIGITAL I/O & SOFTWARE DELAY

OBJECTIVE

To get familiar with a select set of the PIC18F4431 microcontroller (MCU) instructions, learn how to use and program each port and set it as an input or output port, learn how to generate short (micro seconds) and long (mille seconds) delays, introduce two of the circuit's elements that can be used to interface with MCU such as, light emitting diodes (LED) and switches. Finally, learn how to eliminate the noise usually caused by switches (mechanical ones) in the circuit.

INTRODUCTION

Microcontroller has been widely used in industrial and consumer products that require a built- in automated process. MCUs are used in household appliances such as dishwashers, fridges, washing machines and microwaves. In automobiles, On-Board Diagnostics system or "OBD" is a computer-based system built into all 1996 and later light-duty vehicles and trucks, as required by the Clean Air Act Amendments of 1990. OBD systems are designed to monitor the performance of some major components of an engine including those responsible for controlling emissions. The multi-sensor system adaptively generates a set of parameters to optimally control the firing time, the fuel-oxygen mix, the exhaust system emission, and the traction control. Separate MCUs could be used for Automobile ABS system and communicates with the main OBD. These are a few examples that show how MCUs have become essential devices embedded in all equipments and devices we have nowadays. This is made possible because the microcontroller is a stand-alone device that has a microprocessor, a memory, input/output ports, and special function devices such as an analog-to-digital converter and timers, all connected, packaged and delivered to us on a single semi-conductor chip. The benefit appears in the circuit design. In order to have a self controlled running system, all you have to do is build your circuit or attach the components you want to control to the device, write a program, store the program on the chip, and power it up.

I/O PORT CONFIGURATION

PIC18F4431 microcontroller is a 40-pin or 44-pin device depending on the package type. Due to the limited number of pins in the package, most pins are multiplexed with an alternate function from the peripheral features on the device. For example, pin RA6 is multiplexed with the main oscillator pin. There are 5 ports available on the device: PORTA, PORTB, PORTC, PORTD, and   PORTE. Each port has three registers essential for its operation. These registers are:

- **TRIS** register (tri-state device controls the direction of the data in bidirectional ports).
- **PORT** register (reads the Voltage level on the pins of the device).
- **LAT** register (stores the output data until further read-modify-write operation).

Port initialization is done in steps as follows:

1. Clear all data in the latch register **LATX** and on the port **PORTX** using the instruction **CLRF**.
2. Load the proper byte to the WREG using the instruction **MOVLW**.
3. Load the content of WREG into **TRISX** using the instruction **MOVWF**.

Assume we want to set PORTD (D0 to D7) as outputs, the byte in step 2 will be 0X00H. An alternative way is to clear TRISD (all bits =0) "**CLRF TRISD**". As a result, all bits on PORTD are set as outputs (i.e., put the content of the output latch register LATD on the corresponding pins <0:7> of PORTD). So, if we send "10101010" to the port while connecting a pre-built 8 light emitting diodes (LEDs) board to PORTD pins, all LEDs attached to D0, D2, D4 and D6 will be on and the others will be off. To set the port as inputs, the byte in step 2 will be 0XFFH. Now you're ready to output the desired data on PORTD using working register W as follows:

```
MOVLW       B'11001100'
MOVWF       PORTD
```

LIGHT-EMITTING DIODES

The Light emitting diode(LED), is a semiconductor device consisting of two different types of semiconducting material called N-type and P-type fused together to form an PN-junction. The P-type refers to the anode and the N-type refers to the cathode see figure (1). This junction gives the device or the diode a unique electric characteristic such as conducting the current only in one direction (the forward direction), and preventing the current from passing through the junction if connected in the reverse direction. The forward and reverse direction configurations are obtained when we connect the power supply to the diode's pins. If the positive terminal of power supply or battery is connected to the P-type and the negative to the N-type, then the diode is now in forward direction and the current is flowing through the circuit with a minimum resistance (few Ohms for general purpose diodes). In the vice versa, almost no current will flow and the diode will show maximum resistance (in terms of mega Ohms). To limit the forward current, we have to connect the diode in series with a relatively small resistance (330Ω). See figure (1-b) below.

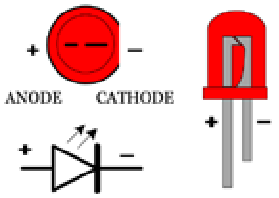


Figure 1.1a: Shape and symbol for LED

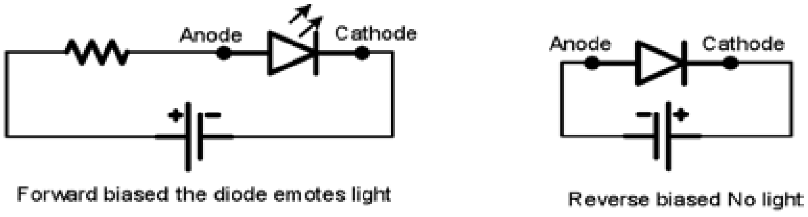


Figure 1.1b: LED in circuit

USING SWITCHES

Figure (2-a) shows the circuit connection of the Push-Button key usually found in most electronic circuits. The metal contact of the mechanical switch tends to vibrate or bounce before taking its final position as shown in figure (2-b). This could happen when closing or releasing the switch. The bounces will be read as multiple inputs by the microcontroller. The consequence is clear in a telephone set with a push button. When you push number 2, the controller will read 222 because of the bounces. Multiple inputs can be eliminated by a key-debounce technique, using either hardware or software. In the hardware technique, several circuits are available to eliminate the bounce. In the hardware technique, an external or debounce device is placed between the switch or key and the MCU input port. For example, two Schmitt NAND gates (CD4093 gates with delay or hysteresis) are placed between the push-button and the input port of MCU to provide a clean square wave input to the MCU.

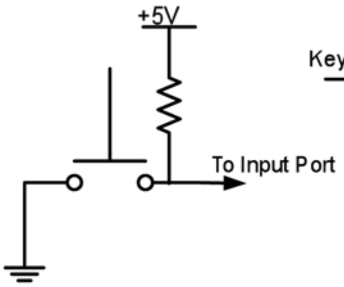


Figure 1.2a

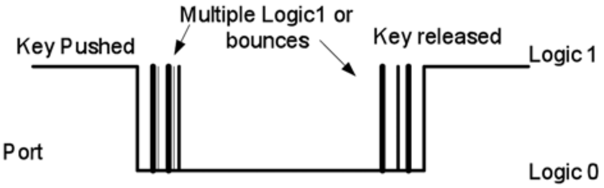


Figure 1.2b

In the software debounce-technique, the program goes in delay loop for 10 to 20 ms after it detects a switch closer, and reads the switch closer again. If it finds the reading to be still low, the program accepts and processes the reading. In other words, the program will wait for all noises or bounces to go away and only then begin reading.

GENERATING TIME DELAY

PIC18F4431 has multiple built-in timers. Employing these timers to generate a time delay is known as the hardware method. Another method is called software method, when the loops and nested loops are used to generate delay. In this experiment, we use loops to generate the time delay required for debounce. The basic idea is to use the time period of the internal (or external) oscillator of the MCU as a building block to construct the required delay. The operating frequency of PIC18F4431 ranges from DC-to- 40MHz. The execution time of any instruction is called instruction cycle. Most of the instructions for PIC18 are executed in a single instruction cycle. One instruction cycle consists of four oscillator periods. The following instructions will be used in a delay loop:

• NOP	takes 1 cycle
• GOTO	takes 2 cycles
• CALL	takes 2 cycles
• RETURN	takes 2 cycles
• DECFSZ	see below *
• MOVLW or MOVWF	takes 1 cycle

**The DECFSZ (decrement F and skip if zero) instruction takes 2 cycles if ZERO otherwise, it takes 1 instruction cycle. So, overall, the DECFSZ and the GOTO takes 3 instruction cycles.*

Now assume the MCU operating frequency is 8 MHz and you want to generate delay of 10ms, 20ms, and 100ms. The delay is very long compared to the time period of the operating frequency. How shall you approach this? The solution lies in using the loop and nested loop (loop-within-the loop). To construct a loop, you need a register loaded with decimal numbers to start decrementing the content by one, until the register content becomes zero, in order to be out of the loop. The largest Hexadecimal number that can be loaded into 8-bits register is 0XFFh or D'255'. Let us calculate the total delay for the following delay loop;

```
COUNT1    EQU       D'200'    ;
REG10     EQU       0x10      ; Register 10 is labelled as REG10
REG11     EQU       0x11      ; Define data register address
          ORG       0x20      ; Begin assembly at program memory 20H
          MOVLW     COUNT1    ; Load deciman count in W
          MOVMF     REG10     ; Set up REG10 as a counter
          MOVWF     REG11     ; Set up REG11 as a counter
Delay-X   DECFSZ    REG10     ; Decrement REG10 takes 1 cycle and 2 when zero
          GOTO      Delay-X   ;Takes 2 cycles
          END
```

To calculate the time required to execute Delay-X loop do the following:

- Calculate the time period of clock pulses of the MCU oscillator "8MHz" which is $T = 1/f = 1/8 \text{ x } 10^6$ =125 nS.
- Calculate the total instruction cycles in Delay-X loop is 3 cycles OR 3 x 4 clock pulses for each cycle = 12 Clk.
- Compute the time to execute the loop once = 12 x 125 nS = 1.5 μS.
- Multiply Delay-X by 200 (executed 200 times). Total time delay is T_L = 1.5μS X 200 = 300μS.
- Add 1 cycle to be exact = 125 nS x 4 = 500 nS time to the total time delay, which can be ignored compared to 300μS.

We can increase the loop delay by adding **NOP** instruction inside the loop. This will only be possible if we split the **DECFSZ** into instruction **DECF** and **BNZ**. Rewrite the above loop using these new instructions:

```
Delay-X   DECF      REG10     ; Decrement REG10, takes 1 cycle.
          NOP                 ; One extra cycle added to step 2 above.
          NOP                 ; Another extra cycle added to step 2 above.
          BNZ       Delay-X   ; Branch to Delay-X if not zero, takes 2 cycles.
```

Figure 1.1: HT33 Tubular Heat Exchanger

Step 2 above will be 5 instruction cycles in Delay-X OR 5 x 4 clock pulses per cycle = 20 CIK. The new time delay is 500μs.

As we can see from the above calculation, one delay loop is not enough to obtain the required delay in ms. The solution to that would be the nested loop. Rewrite the above program and add another outer delay loop called Delay-X1 with REG11 as a counter.

```
Delay-X1    DECFSZ    REG11      ; Decrement REG10 takes 1 cycle and 2 when zero.
Delay-X     DECFSZ    REG10      ; Decrement REG10 takes 1 cycle and 2 when zero.
            GOTO      Delay-X    ; Takes 2 cycles
            GOTO      Delay-X1   ; Takes 2 cycles
            END
```

The execution time of the above nested delay loop is 60 mS. Now you can modify the content of the REG10 and REG11 and add NOP to obtain the required delay.

EXPERIMENT

The microcontroller Lab; is equipped with several tools to allow students to focus on the objective of each experiment while, acquiring some hands on experience. Other than original equipments used in electronics lab such as Oscilloscopes, Power Supplies, Multi-meters, and Signal generators, there are a number of specialized sets of tools and equipments. The microcontroller development system in the lab consists of a desktop computer, software from Microchip called MPLAB IDE (Integrated Development Environment), and an In Circuit Emulator ICE2000. Rather than using the real MCU in the circuit, the later one "ICE2000" is used. Its features mimic a large number of Microchip MCU devices on a pin level without the need to have the real MCU integrated circuits. After running the program several times, debugging and developing the functionality of the program using ICE2000, Students will be able to burn the program in the chip using MCU programmer and replace the emulator with the real MCU in the circuit. Please refer to the appendix for a brief introduction to MPLAB IDE & ICE2000.

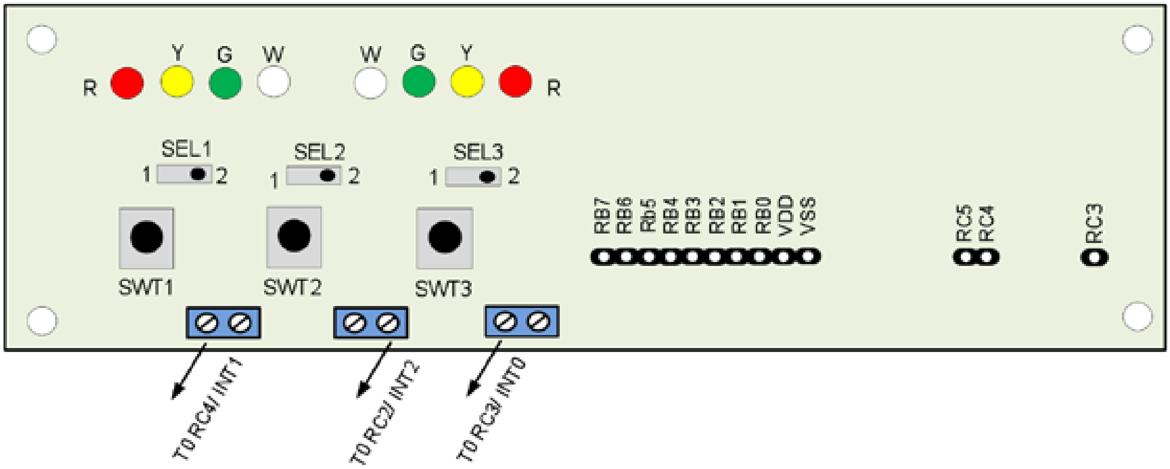


Figure 1.4: Layout of the pre-build circuit board

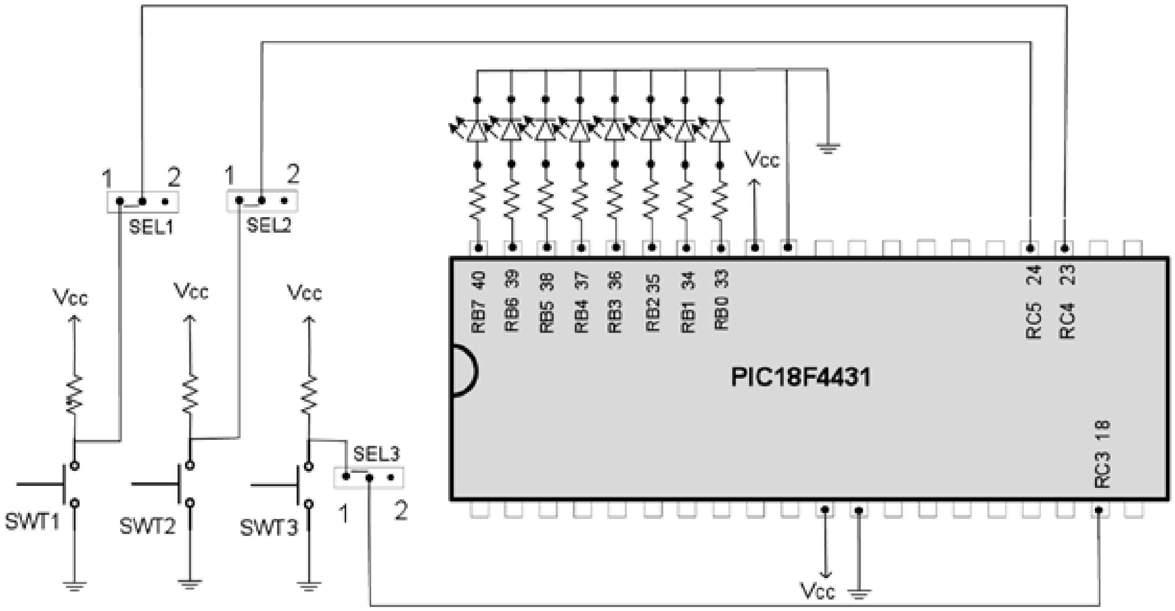


Figure 1.4: Schematic diagram of the pre-build circuit board

In order to minimize the error and time consumed in building the external hardware, the external hardware is pre-built on a PCB with a socket matching the pins on the transition socket of the ICE2000. The students have to first connect the pre-built circuit to the ICE2000 transition socket, and then connect the power supply.

Figures (3) and (4) shows the layout and the schematic diagram of the pre-built circuit which will be used to practice binary display using 8 LEDs and control switches. Anodes of the 8 LEDs are connected to PORTB via limiting resistors "330Ω" and Cathodes are connected to the ground. SWT 1, SWT2, and SWT3 are connected to pins RC4, RC5, and RC3 respectively.

PROCEDURE

1. Write an assembly program to light-up LEDs 0,1,2,3 for 500 ms, when SWT1 is pushed. Use 10ms or more delay for switch debounce. (Hint: loop until SWT1 is detected and send the binary byte "01010101" to PORTB. Keep these data for 500 ms CLER PORTB.
2. Modify step one program to light-up LEDs 4,5,6,7 for 500 ms when SWT2 is pushed. (Hint: Loop until SWT2 is detected and send the binary byte "10101010" to PORTB, CLEAR PORTB after 500 ms).
3. Add the following function to the assembly program: When SWT3 is pushed LEDs 0,1,2,3 will light-up for 500 ms and then off for 500 ms. Afterwards, LEDs 4,5,6,7 lights-up for 500 ms and then turns off for 500 ms. Continue toggling until SWT3 is pushed again.

PEDESTRIAN CROSSWALK

4. Write a program to simulate a pedestrian crosswalk switch. The crosswalk consists of two traffic lights mounted at the corner of cross roads. The traffic light is simulated by 4 LEDs RED-Yellow-Green-White. The first three lights (R-Y-G) are used to control the traffic flow, while the White LED is for the pedestrian control. The cross-section consists of one-way roads with no left or right turns. The pedestrian is allowed to cross when the green and white lights are on. There is no traffic light sequence implemented. The light will stay on in one direct until the pedestrian pushes a button to demand crossing. After pushing a switch to cross, the light will stay on in the desired direction until another pedestrian pushes the button to demanding crossing in the opposite direction. On the pre-built board, the two lights are mounted on one raw. SWT1, SWT2, and SWT3 are mounted on the second row.

5. Add another feature to the program of step4. When SWT3 is pushed LEDs 0, 4 (RED-light) flashes for 5 time, then stop and return to normal traffic control.

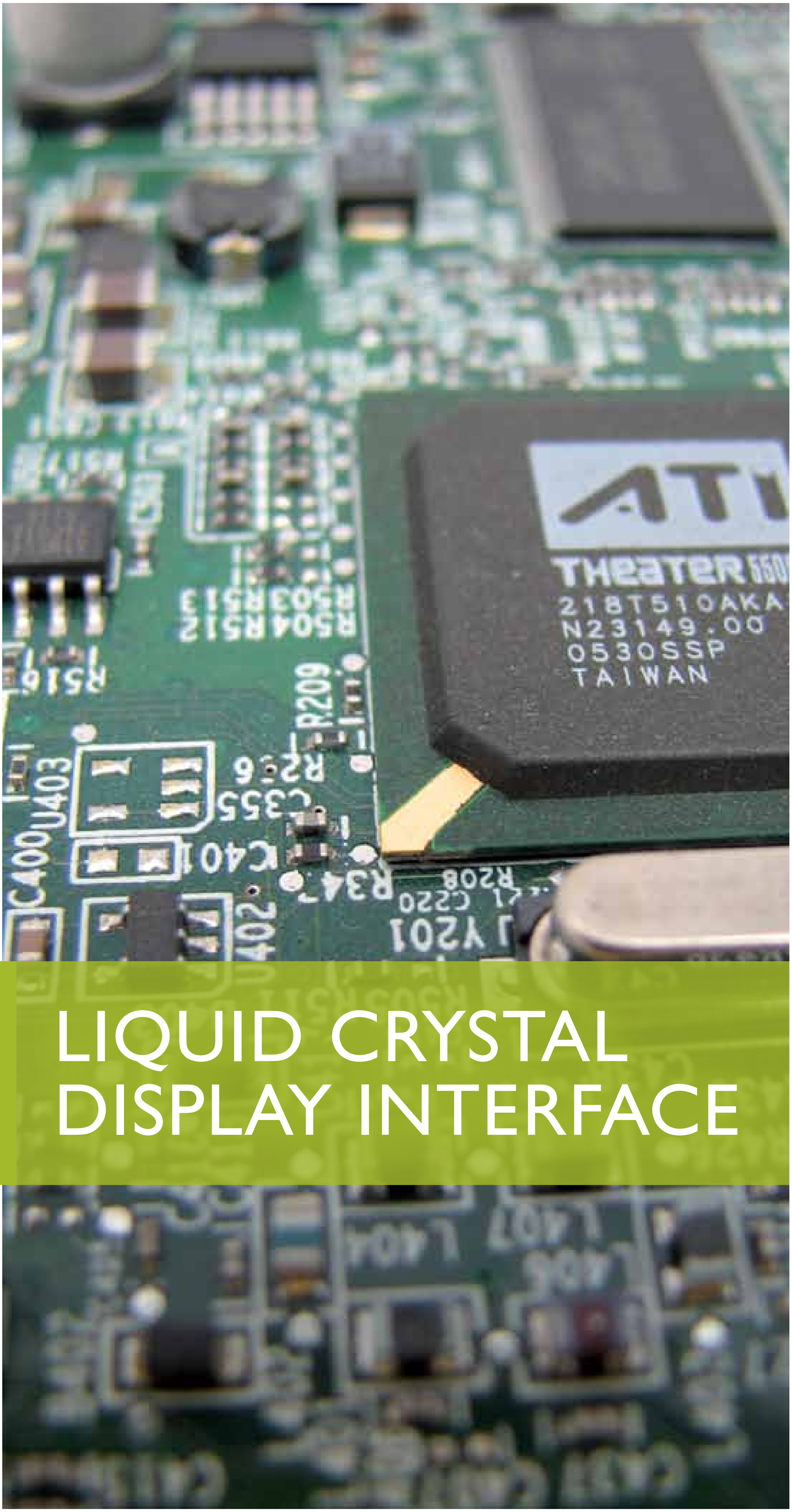
FURTHER WORK *(Optional)*

This part is outside the scope of the experiment therefore, you are not obliged to do it.

6. When SWT3+SWT1 are pushed simultaneously, LEDs 0,2,4,6 light-up for then off and, LEDs 1,3,5,7 light-up then off. This sequence will continue as if you are shifting the lights to the left.

7. When SWT3+SWT2 are pushed simultaneously, LEDs 1,3,5,7 light-up and then turn off and, LEDs 0,2,4,6 light-up then off. This sequence will continue as if you are shifting the lights to the right.

8. You can apply the same concept when two switches are pushed simultaneously on multi-color LEDs to produce a light show.



EXPERIMENT

2

LIQUID CRYSTAL DISPLAY INTERFACE

OBJECTIVE

To build up some knowledge about Liquid crystal display, develop and test device drivers for accessing a liquid crystal display (LCD), learn how to initialize the LCD and properly manage the display, as well as learn about ASCII codes, and how to display words and numbers.

INTRODUCTION

Liquid Cristal Display Modules are used in a wide variety of applications due to its thin, light weight, low voltage, and low power consumption. The LCD modules can be programmed to display not only numbers but also letters, words, and all sort of special characters and symbols. This makes them more versatile than the familiar 7-segment light emitting diode (LED) display. There are two types of such displays; one called alphanumeric LCD and the other called graphic LCD. An LCD is a passive device and relies on ambient light to be visible. LED Backlight can make LCD visible both day and night. A series of LED's are arranged either along the edge of the LCD panel or behind a diffuser to provide a bright, and even backlight to the display. LEDs are powered by a 5V supply, moderately bright and very long lasting.

An alphanumeric module exits in a wide variety of shapes and sizes. Line lengths of 8, 16, 20, 24, 32, and 40 characters are all standard in one, two and four-line versions. The HD44780 display controller (built-in LCD circuit) is commonly used in a variety of LCDs made by manufacturers such as Hitachi, Optrex, Amperex, Densitron, and Epson.

Unlike the 7-segment display, in all instances where an LCD is to be used in design, a microcontroller will be needed to drive it. In this lab, we will deal with the 20 X 2 lines alphanumeric LCD modules which use the Hitachi HD44780 (or compatible) controller chip.

CONNECTIONS

Most LCD modules provide a similar user-interface of a 14-pin access for LCD with no backlight and a 16-pin access for LCD with backlight. The connections are laid out in one of the two common configurations; either two rows of seven pins (the backlight has a separate connection) or a single row of 14pins or 16 pins. Figure (1) shows the two layouts. On most displays, the pins are numbered on the LCD's Printed circuit board but if not, it is quite easy to locate pin 1. Since this pin is connected to the ground, it is easy to trace it to the main ground connection on the board. The pins on LCD with a built-in driver (Hitachi HD44780) include three power connections (pin1-Pin3), three control signals (pin4-pin6), and an 8-bit data bus (pin7-pin14). If the LCD has a backlight, then you will have two extra pins (15 and 16). A brief description of the pin connections is given in table 1.

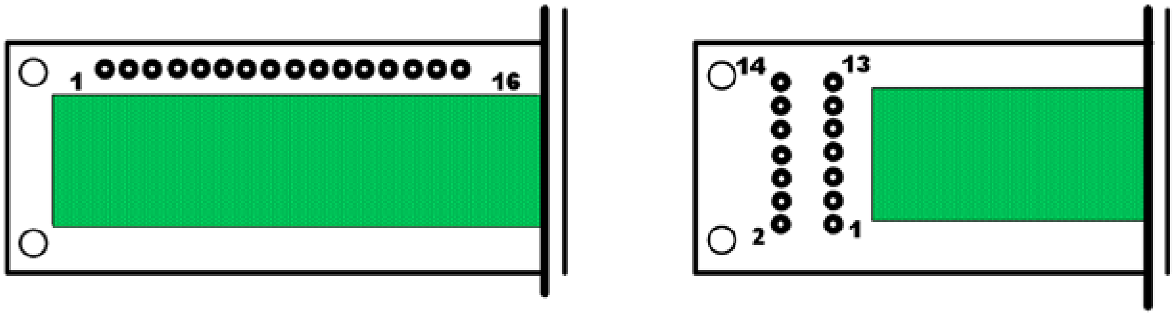


Figure 2.1

Table 2.1

Pin	Label	Function
1, 2, 3	Vss, Vdd, Vee	Ground, +Ve supply, Contrast
4, 5, 6	Rs, R/W, E	Register select, Read/write, Enable
7 : 14	D0 ::::: D7	Data bit 0 ::::: Data bit 7

Pin2 should be connected to the +Ve supply (5V) and Vss to ground or 0V (common).

Pin3 is a contrast control pin, (Vee) which is used to alter the contrast of the display. As recommended, you should connect this pin to a variable voltage supply. This can be achieved by using a potentiometer between +V and ground of supply or +V & -V supply and the wiper connected to the Vee as shown in figure (2). For simplicity, connect Vee to the ground or 0V.

Let's get back to the built-in controller HD44780. The driver has two 8-bit internal registers; one called instruction register (IR) and the other called data register (DR). The microcontroller writes into the IR to set up the LCD parameters for the desired operation (clear display, cursor shift, etc…) and into the DR to display ASCII characters.

Pin4 is the register select (RS) line. It is the first of the three command control inputs. When this line is low, data bytes transferred to the display are treated as commands and data bytes read from the display indicate its status. By setting the RS line high, character data can be transferred to and from the module.

Pin5 is Read/Write (R/W) line. This input is used to initiate the actual transfer of commands or character data between the module and the data lines. When writing to the display, data is transferred only on the high to low transition of this signal. When reading from the display, data will become available shortly after the low to high transition and remain available until the signal falls low again.

Pin6 is the enable line. This input is used to initiate the actual transfer of instructions or character data between the module and the data lines. When writing to the display, data is transferred only on the high to low transition of this signal. However, when reading from the display, data will become available shortly after the low to high transition and remain available until the signal falls low again.

Pins 7 to 14 are the data bus lines (D0:D7). Data can be transferred to and from the display, either as a single byte (8-bit operation mode) or nibble (4-bit operation mode). In the later case, only the upper four data lines (D4:D7) are used. The 4-bit mode is useful when using a microcontroller with fewer input/output ports or lines.

DATA VS INSTRUCTION

The display control lines are set up before sending either an instruction word or a data word. The RS (Pin4) line is set to 0 for sending an instruction or set to 1 for sending data. This must be done in advance before making the enable line HI. Also, the data or instruction byte must be stable at the PORT prior to making the enable line HI.

Table 2.2: Instructions which are needed to control the display

Instruction	Binary										Hex
	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0	
Clear Display	0	0	0	0	0	0	0	0	0	1	01
Display & Cursor Home	0	0	0	0	0	0	0	0	1	X	02 or 03
Character Entry Mode	0	0	0	0	0	0	0	1	1/D	S	04 to 07
Display On/Off & Cursor	0	0	0	0	0	0	1	D	U	B	08 to 0F
Display/Cursor Shift	0	0	0	0	0	1	D/C	R/L	X	X	10 to 1F
Function Set	0	0	0	0	1	8/4	2/1	10/7	X	X	20 to 3F
Set CGRAM Address	0	0	0	1	A	A	A	A	A	A	40 to 7F
Set Display Address	0	0	1	A	A	A	A	A	A	A	80 to FF

Initialization Settings:

1/D	1 = Increment, 0 = Decrement	D/C	1 = Display shift, 0 = Cursor move*
D	1 = Display on, 0 = Off	R/L	1 = Right shift, 0 = Left shift*
S	1 = Display shift on, 0 = Off	8/4	1 = 8-bits mode, 0 = 4-bits mode
U	1 = Cursor underline on, 0 = Off	2/1	1 = 2 line mode, 0 = 1 line mode
B	1 = Cursor blink on, 0 = Off	10/7	1 = 5x10 dot format, 0 = 5x7 dot format

* = not initialization settings   X = don't care   CGRAM = Character Generator RAM

Please note that the register select line (RS) and the read/write line (R/W) are set to zero. This is a necessary setting for sending instructions to the LCD. To send data, the setting will be RS=1 and R/W =0.

CHARACTER ADDRESSES

We will be using either 2x16 or 2X40 LCDs in this lab. When the LCD is powered up, the cursor is positioned at the beginning of the first line. Every time a character is entered, the cursor moves on to the next address. This auto-increment of the cursor address makes entering string of characters very easy, since it is not necessary to specify a separate address for each character. However, it may be necessary to position a string of characters somewhere other than the beginning of the first line. Let us assume the middle of the screen or the left side of the screen. You must consider the limitation of the LCD being used in your lab. For example, in case of the LCD 2x16, only 32 characters can be laid out as 16 characters (on each line) in two line mode or 32 characters in one line mode.

The relationship between DDRAM addresses and display locations are shown below.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	Display digit
Line 1	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	DDRAM
Line 2	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	Address

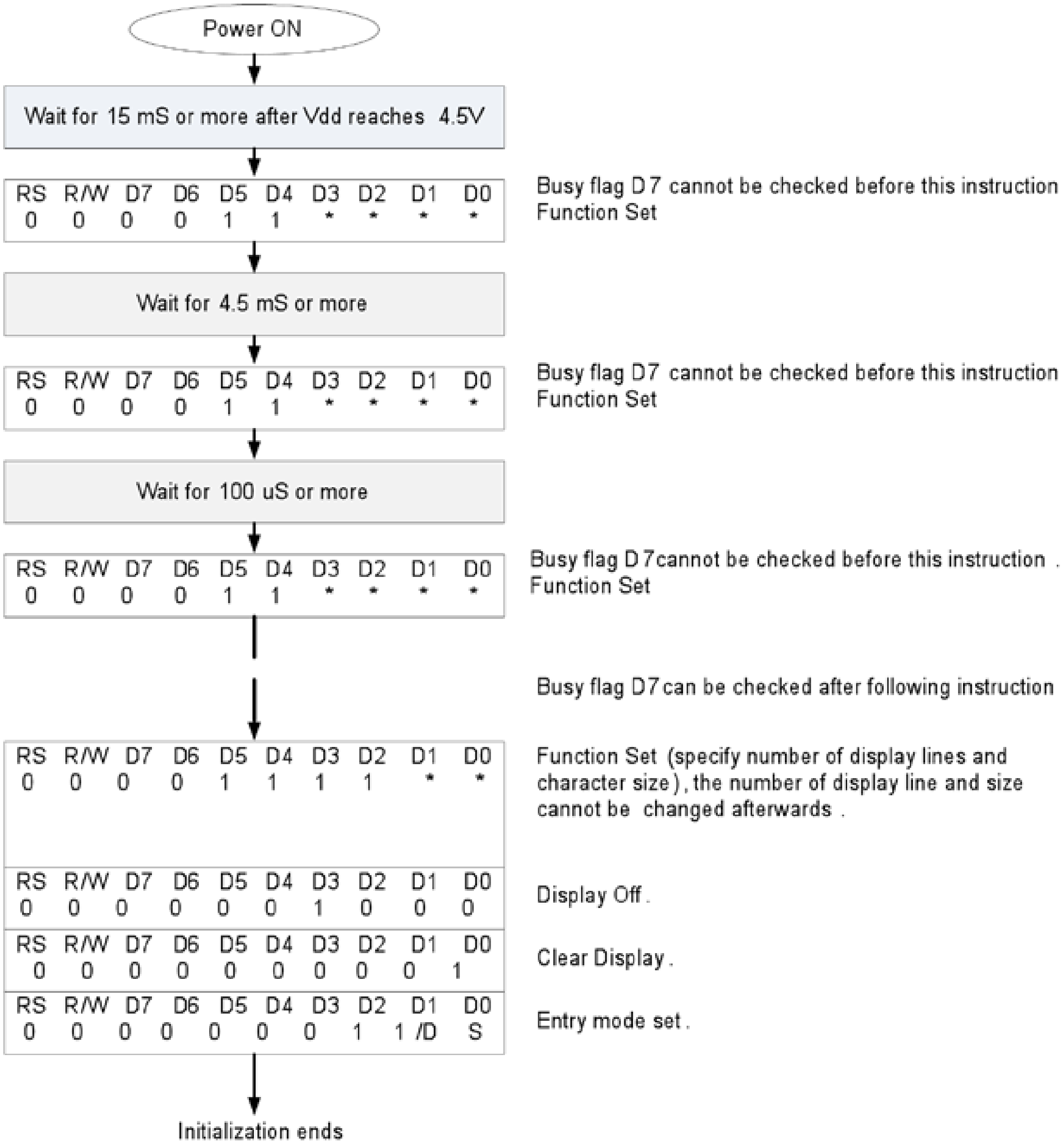
Display Size	Visible	
	Character Positions	DDRAM Addresses
2x16	00 → 15	0x00 → 0x0F + 0x40 → 0x4F
2x20	00 → 19	0x00 → 0x13 + 0x40 → 0x53
2x24	00 → 23	0x00 → 0x17 + 0x40 → 0x57
2x32	00 → 31	0x00 → 0x1F + 0x40 → 0x5F
2x40	00 → 39	0x00 → 0x27 + 0x40 → 0x67

LCD OPERATION

Prior to displaying a message on the LCD, you should prepare or initialize the display. The display module resets itself to an initial state (manufacture default settings) when power is applied. This type of initialization is called initialization by internal reset circuit. During the execution of this initialization or automatic reset, the built-in controller on the LCD sets the data line D7 high, as a busy flag, indicating that the controller is busy completing the internal operation. When the controller completes the operation, it resets the data line D7. The busy state lasts 10ms after VDD rises to 4.7V. The following instructions are executed in the initialization:

1. Display Clear
2. Function Set 8/4=1, 2/1=0, 10/7=0
3. Display On/Of control D=B=0
4. Entry Mode Set I/D=1, S=0

Another type of initialization is called Initializing by Instruction. IT will be part of the program you write to display a message on the LCD. Before sending an instruction or a data byte, you need to check the busy flag on the D7 line. The flow chart below shows the initialization sequence followed by the assembly code. Please note that PORTB will be used for data D0:D7 and three pins of PORTD will be assigned for RS=RD2, R/W=RD3, and E=RD4.



Wait for 15 mS or more after Vdd reaches 4.5V

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	1	1	*	*	*	*

Busy flag D7 cannot be checked before this instruction
Function Set

Wait for 4.5 mS or more

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	1	1	*	*	*	*

Busy flag D7 cannot be checked before this instruction
Function Set

Wait for 100 uS or more

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	1	1	*	*	*	*

Busy flag D7cannot be checked before this instruction .
Function Set

Busy flag D7can be checked after following instruction

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	1	1	1	1	*	*

Function Set (specify number of display lines and character size), the number of display line and size cannot be changed afterwards .

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	1	0	0	0

Display Off .

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0	0	1

Clear Display .

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	1	1 /D	S

Entry mode set .

INITIALIZATION CODE

The following initialization code is needed for the display to perform all the necessary settings the moment after power up. Due to the slow operation nature of the LCD, the delay routine provides timing balance between the faster MCU and the slower LCD. The routine keeps the MCU occupied while the LCD is initializing. Use TEMP register to store values.

Initialization Routine

```
CALL    Delay_15ms                  ; delay 15ms after power on
        MOVLW       B'00110000'     ; Function Set, the first instruction to
        MOVWF       TEMP            ; LCD to set up 8-bit mode interface.
        CALL        LCD_FS          ; Send instruction
        CALL        Delay_4.5ms     ; delay 4.5mS while LCD initializes
        MOVF        TEMP,W          ; get Same instruction from TEMP
        CALL        LCD_FS          ; Send instruction
        CALL        Delay_100us     ; short delay 100uS while LCD is busy
        MOVF        TEMP,W
        CALL        LCD_FS          ; Send instruction
        MOVLW       B'00111000'     ; Set 8-bit mode, 2 lines and 5x7 dots
        CALL        LCD_CMD
        MOVLW       B'00001000'     ; Display off
        CALL        LCD_CMD
        MOVLW       B'00000001'     ; Clear display
        CALL        LCD_CMD
        MOVLW       B'00111000'     ; Entry mode. Shift and increment cursor.
        CALL        LCD_CMD
        MOVLW       B'00001100'     ; Display On
        CALL        LCD_CMD
        RETURN
```

For information on how to write a delay routine, please refer to Lab1. The above initialization program uses subroutines LCD_FS and LCD_CMD. The subroutines LCD_FS, and LCD_CMD are practically considered as the ones that set the control lines RS=R/W=0, put the instruction byte on PORTB and pulse E line. The difference between LCD_CMD and LCD_FS is the later skips the call for CHK_BF subroutine. This subroutine structure with very little change will be used to send out a character or number to the display.

The following routine is used during the LCD initialization to send instructions Or commands, to the LCD. Note that RS=0 during initialization.

```
LCD_CMD:    MOVWF       TEMP        ; save command code in W
            CALL        CHK_BF      ; check if D7 is low
LCD_FS:     BCF         PORTD,2     ; RS=0 select instruction register (IR)
            BCF         PORTD,3     ; set R/W line to 0
            BSF         PORTD,4     ; take E line high
            NOP                     ; hold it high for 1 instruction cycle
            ……                      ; you may need to add more delay here
            MOVF        TEMP,W      ; Get code in W
            MOVWF       PORTB       ; send command code out
            BCF         PORTD,4     ; take E line low again
            RETURN
```

The following Routine “CHK_BF” is used to check the busy flag on pin D7.

```
CHK_BF   SETF      TRISB        ; set PORT B as inputs
         BCF       PORTD,2      ; set RS line to zero (PORT D bit 3)
         BSF       PORTD,3      ; R/W=1 to read D7 flag
READ:    BSF       PORTD,4      ; take E line high
         NOP                    ; hold it high for 1 instruction cycle
         BCF       PORTD,4      ; take E line low again
         MOVF      PORTB,W      ; read PORTB and store in W
         BTFSC     PORTB,7      ; check if LCD busy (D7=1)
         BRA       READ         ; if D7=1, go back and check again
         CLRF      TRISB        ; set PORT B as output
```

Notice the difference in the setting of the R/W line. In LCD_CMD (LCD command) and LCD_FS (LCD function set) subroutines, the R/W line is set to zero (write mode). However, in CHK_BF (check busy flag) routine, the R/W line is set to one in order to be able to read the busy flag on pin D7.

DISPLAYING A MESSAGE

The above LCD_CMD subroutine can be easily modified to obtain the new subroutine LCD_DATA shown below. The LCD_DATA routine will be used to send a message to the LCD. All instructions in the LCD_CMD routine will be used except the BTFSC instruction, which is not necessary at this point. Also, there is a change in setting of the RS line. The LCD_CMD writes into IR (RS=0) while LCD_DATA writes into DR (RS=1). To display a message on the LCD, the data must be retrieved and stored in WREG before calling the LCD_DATA routine.

The following routine is used to send a message to the LCD

```
LCD_DATA   MOVWF     TEMP         ; save command code in W
           CALL      CHK_BF       ; check if D7 is low
           BSF       PORTD,2      ; RS=1 select data register (DR)
           BCF       PORTD,3      ; set R/W line to 0
           BSF       PORTD,4      ; take E line high
           NOP                    ; hold it high for 1 instruction cycle
           ……                     ; you may need to add more delay here
           MOVF      TEMP,W       ; Get code in W
           MOVWF     PORTB        ; send command code out
           BCF       PORTD,4      ; take E line low again
           RETURN
```

Computer can only understand two numbers {0, 1}. The above assembly codes or codes in C, basic, and any high level computer language are converted into a machine code before given to the computer. The machine code consists of a series of zeros and ones. This series is what the microcontrollers or processors understand. Therefore, typing characters and numbers will never be recognized by computer unless there is a binary {0, 1} representation of a character such as 'a' or '@' or an action of some sort. The American Standard Code for Information Interchange (ASCII) is a character-encoding scheme based on the ordering of the English alphabet. ASCII codes represent text in computers, communication equipment, and other devices that use text such as intelligent typewriters and LCDs. Every intelligent display has a built-in table known as the ASCII table (character generator look-up table). This table can be found in the appendix of any computer or logic textbook. It contains 8-bit binary codes associated with characters; two nibbles (4-bit) arranged in a matrix form, the most significant 4-bit for row, and the least significant 4-bit for column.

For example, the binary code for the letter M is 0100 1101 and the equivalent hexadecimal is 0x4D.

Most ASCII characters may be defined in MPASM using the MOVLW instruction. To display a character or message use either of the following ways:

1. Display a character

```
                MOVLW       'M'             ; Define uppercase "M"
;;;;;;;;;;;;;OR MOVLW       0x4D            ;Or use the hexadecimal code for "M"
                CALL        LCD_DATA
                .........   .........
                MOVLW       'T'
                CALL        LCD_DATA
```

3. Output a message (using table)

```
                MOVLW       0               ; Table address of start of message
message:
                MOVWF       TEMP1           ; TEMP1 holds start of message address
                CALL        Table           ; Table message of 0xFF length
                ANDLW       0xFF            ; Check is at end of message
                BTFSC       SATAUS, Z       ; Zero returned at end
                GOTO        OUT             ; Skip the instruction if not zero
                CALL        LCD_DATA        ; Display character
                MOVF        TEMP1, W        ; Point to next character
                ADDLW       1               ; increment WREG by 1
                GOTO        message
OUT:
                ...........  ...........
Table
                ADDWF       PCL             ; Add W to the PC to create a jump
                RETLW       'M'             ; Begin table, return literal (M) to W
                .
                .
                RETLW       'A'
                RETLW       0               ; End of the table w=0
```

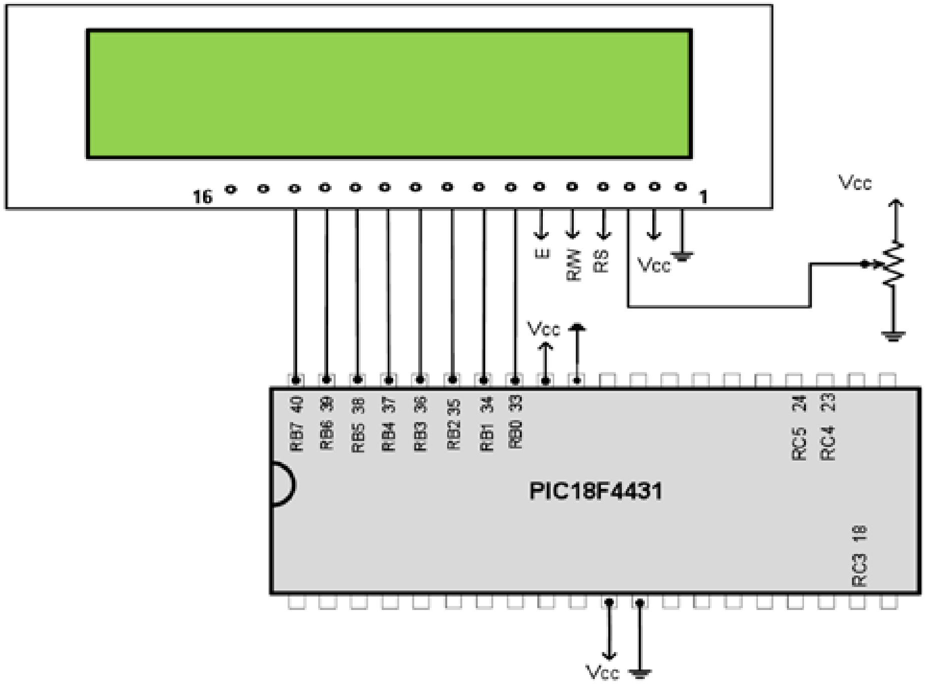
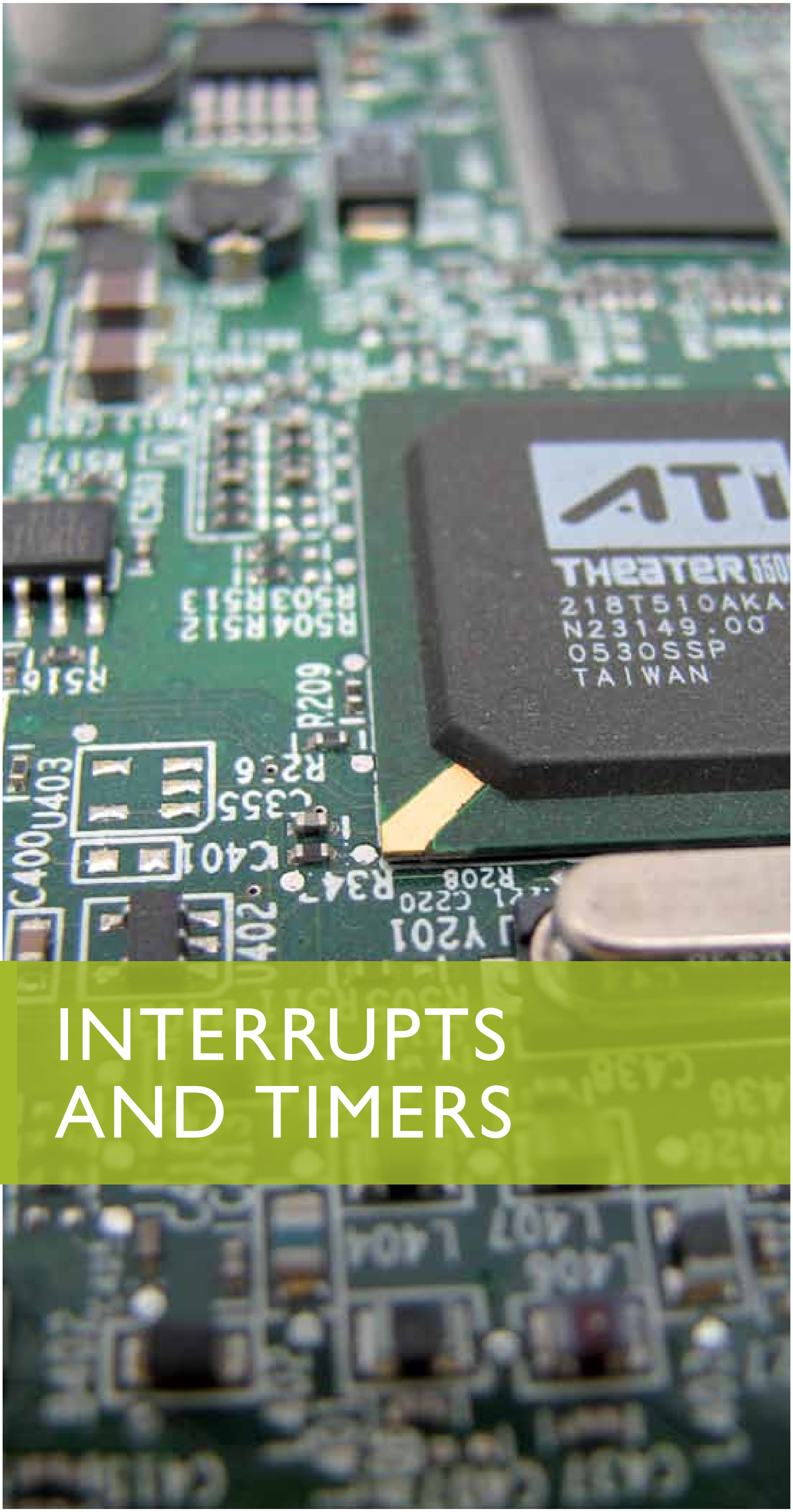


Figure 2

EXPERIMENT

1. Select a suitable place next to the transition socket and, connect the LCD's socket to the breadboard.
2. Connect PORTB (RB0...RB7) pins to D0….D7 pins on the LCD.
3. Connect the three pins of PORTD (RD2, RD3, RD4) to the RS, R/W, and E pins on the LCD.
4. Connect the following pins of the LCD: pin1 to the ground, pin2 to Vcc, and pin3 to the contract control variable resistance as shown in the figure above.
5. Write a program to display your Family name in the middle of the first line, and your course number in the middle of the second line.
6. Run the program and adjust the contract.
7. Alternate the position of the name and the course number (i.e. place your name in the middle of the second line and the course number on the first). Use the appropriate delay to make it visible (without any flickering). Wait for about 200ms and then return to your position in step5.
8. Display your name at the beginning of the first line and the course number at the end of the second line.
9. Rotate to the right the data of the fist line and rotate to the left the data on the second line.
10. After two rotations stop shifting the display, keeping your name in the middle of fist line and blinking course number in the middle of the second line.
11. Write down all the subroutine with corresponding explanation to all the used commands.



EXPERIMENT 3

INTERRUPTS AND TIMERS

OBJECTIVE

Introduce the interrupt process in microcontrollers/microprocessors in a simple plan text and assembly language program. Learn about the sources of interrupt and the associate control registers. By the end of this lab you will be able to implement interrupts into your own assembly program.

INTRODUCTION

The interrupt is a process or a signal that stops a microcontroller/microprocessor from what it is presently doing so that something else can happen or be done. The form of interrupt that resembles the MCU interrupt in real life is shown in the following example. Assume while chatting to a friend, your telephone rang. You stop chatting, and answer the telephone. When you have finished, you go back to chatting to the friend starting from the point when the telephone rang. The analogy here is very clear, the chatting is the main program, and the telephone ring is the cause or the signal for the interrupt. When the interrupt has ended, the MCU goes back to the main program. The interrupt process in MCU can be classified into two main groups, External interrupt and internal interrupt. The PIC18F4431 can handle the following interrupt sources;

1. A rising and falling voltage pulse on pins RC3/ITN0, RC4/ITN1, and RC5/ITN2.
2. A change in one or more of the voltage levels on the group of pins RB4:RB7.
3. An overflow of the timer registers TIMER0, TIMER1, TIMER2, and TIMER5.

Besides having multiple interrupt sources the PIC18 has a feature that allows each interrupt source to be assigned a high level or low priority level. The high priority interrupt event will override or interrupt any low priority interrupt that may be in progress. The high priority interrupts are directed to the interrupt vector location 000008h and the low priority to the interrupt vector location 000018h.

Generally, each interrupt source has three control bits to declare before its interrupt request granted by the microcontroller:

1. Flag bit to indicate that an interrupt event occurred.
2. Enable bit that allows program execution to branch to the interrupt vector address.
3. Priority bit to select high or low priority.

The interrupt operation is controlled by ten registers. Theses registers are:

RCON,	INTCON,	INTCON2,	INTCONT3	; Rest and Interrupt CONtrol registers
PIR1,	PIR2,	PIR3		; Peripheral Interrupt Request (flag) registers
PIE1,	PIE2,	PIE3,		; Peripheral Interrupt Enable registers
IPR1,	IPR2	IPR3		; Peripheral Interrup Priority registers

Some other sources of interrupt will not be covered in this lab. In the following sections we will examine in details both external and internal interrupts and the associated registers.

EXTERNAL INTERRUPT AND INTCON REGISTER

In the external interrupt, certain pins are assigned to accept a signal from outside circuits. Examining pin diagram of PIC18F4431 microcontroller, you will see that pin 18 shows it is RC3/TOCKI/T5CKI/INT0. RC3 is PORT C bit 3. The INT0 indicates that it can be configured as an external interrupt pin. Looking at other pins such as pin 23 & pin 24, you will recognize that bits 4 & 5 of the same port can be configured as external interrupt pins. Before Using the INT0 or other PORTC pins, we need to tell the MCU that we are going to use interrupts, and to specify which PORTC pin we will be using as an interrupt and not as an I/O pin. Also the MCU needs to be told whether to trigger on the rising or a falling voltage edge RC3, RC4 or RC5. Telling MCU is done through initial settings in some of its special function registers.

Assume we chose RC3/INT0 to be the source of external interrupt. To recognize the occurrence of the interrupt request, the MCU needs to check three bits, the flag, the enable, and the priority. To set the three bits(Flag, Enable and Priority) for RC3/ INT0 we have to set the interrupt Priority enable in RCON register, then set B7,B4, B1 in INTCON, and set B6 in INTCON2. As shown in the table below.

	B7	B6	B5	B4	B3	B2	B1	B0
RCON	IPEN	—	—	—	—	—	—	—
INTCON	GIE/GIEH*	—	—	INT0IE†	—	—	INT0IF††	—
INTCON2	—	INTEDG0**	—	—	—	—	—	—
INTCON3	—	—	—	—	—	—	—	—

*High Priority bit **falling edge †Enable bit ††interrupt flag

Now we are ready to write an assembly program with an External interrupt request. The source as mentioned before is INT0 or pin #3 of port C. A push button key (as shown in figure (1) lab 1) is connected to RC3 or pin 18 of PIC18F4431. The interrupt service routine INT0_ISR is a set of instruction to light up one LED on RB0 of PORTB when the push button is pressed.

```
             ORG       0X00
             GOTO      MAIN
             ORG       0X0008             ; High priority interrupt vector
             GOTO      INT0_ISR           ; Go to interrupt service routine
MAIN:        BSF       RCON, IPEN         ; Enable priority –RCON<7>
             BSF       INTCON, GIEH       ; Enable high priority –INTOCN<7>
             BSF       INCON, INT0IE      ; Set enable bit for INT0
             BCF       INTCON2, INTEDG0   ; Interrupt on the falling edge
;;;;;;;;;;;; MOVLW     B'1001000'         ; Alternative way on initializing INTCON
;;;;;;;;;;;; MOVWF     INTCON             ; Notice that INT0IF the flag bit is rested
             CLEAR     LATB               ; Initialize PORTB
             CLEAR     TRISB              ; all PORTB pins as outputs
WAIT         GOTO      WAIT               ; Loop for an interrupt to occur
             ORG       0x100
INT0_ISR     BCF       INT0IF             ; First clear the interrupt flag
             MOVLW     B'000000001
             MOVWF     PORTB              ; LED connected to RB0 is on
             RETFIE    FAST               ; Retrieve W, STSUS, BSR and enable interrupt
                                          ; bit and return to main program
             END
```

When you first power up the PIC, or if there is a reset, the Program Counter (PC) points to address 0000h, which is the start of program memory. That is the first command in the code ORG 0x00 then we need to skip over the other command using GOTO instruction "GOTO MAIN". In the main program the first few instructions are specific to INT0 initialization mode of interruption. Now in order for any of the interrupt signals (high or low priority, and internal or external) to have any effect, it is necessary to set the global interrupt bit GIE. The INT0 is enabled in high-priority mode and can be interrupted by falling edge of incoming signal. The last instructions include clearing PORT B and an endless loop to wait for an interrupt to occur (when the push-button is pressed).

When the interrupt occurs, three events will take place:

1. The microcontroller detects the high priority interrupt and immediately pushes PC (the return address), W register, STATUS register and BSR onto the stack.
2. Sets the flag bit INT0IF in INTCON (B1=1).
3. Branches to the interrupt vector at address 0008h in the program memory and starts executing at that address.

The interrupt service routine INT0_ISR code must contain two important instructions; one is BCF to clear the interrupt flag bit "INT0IF" in the INTCON register and the second instruction is "RETFIE FAST" or "RETFIE 1" return from interrupt. The stack is popped and the top of the stack (TOS) is loaded into the program counter.

The above program can be modified to allow multiple external interrupt to occur. Let us assume three push-button are connected to RC3 (pin #18), RC4 (pin# 23) and RC5 (pin# 24) and the corresponding service routines are INT0_ISR, INT1_ISR and INT2_ISR respectively. In this case more bits will be set for INTCON2, and INTCON3 as shown in the table below.

	B7	B6	B5	B4	B3	B2	B1	B0
RCON	IPEN	—	—	—	—	—	—	—
INTCON	GIE/GIEH	—	—	INT0IE	—	—	INT0IF	—
INTCON2	—	INTEDG0	INTEDG0	INTEDG2	—	—	—	—
INTCON3	INT2IP	INT1IP	—	INT2IE	INT1IE	—	INT2IF	INT1IF

Bits to be declared in the main program

Flags to be checked

The high priority interrupt vector needs to be modified as well and is given below.

```
ORG      0X0008             ; High priority interrupt vector
BTFSC    INTCON, INT0IF     ; Check INT0 flag, skip if it is clear
GOTO     INT0_ISR           ; Go to interrupt service routine
BTFSC    INTCON3, INT1IF    ; Check INT1 flag, skip if it is clear
GOTO     INT1_ISR
BTFSC    INTCON3, INT2IF    ; Check INT2 flag, skip if it is clear
GOTO     INT2_ISR
```

INTERNAL INTERRUPTS AND RELATED REGISTERS

PIC18 MCU has several internal devices (peripherals) that can interrupt the central processing unit, among them timers and A/D converter. The interrupt process for internal devices is similar to that of the external interrupts. As explained above, the priority, the interrupt flag and the enable bits are required for the interrupt request to be granted by the MCU. These bits are set in three different control registers given below:

1.	IPR	(IPR1, IPR2 and IPR3)	Interrupt Priority register
2.	PIR	(PIR1, PIR2 and PIR3)	Peripheral Interrupt Request (Flag)
3.	PIE	(PIE1, PIE2 and PIE3)	Peripheral Interrupt Enable

Please refer to PIC18F2331/2431/4331/4431 Datasheet Figure 1-2 page 13, and Table 4-3 page 50 for internal devices and the associated control registers. There are four timers: Timer0, Timer1, Timer2, and Timer5 that can be enabled as an interrupt sources. Assume timer 1 and timer 2 are enabled as low priority interrupt sources. In this case, bits in 5 control registers must be declared or checked as given in the table below.

	B7	B6	B5	B4	B3	B2	B1	B0
RCON	IPEN*	—	—	—	—	—	—	—
INTCON	—	PEIE/GIEL*	—	—	—	—	—	—
IPR1	—	—	—	—	—	—	TMR2IP*	TMR1IP*
PIE1	—	—	—	—	—	—	TMR2IE*	TMR1IE*
PIR1	—	—	—	—	—	—	TMR2IF	TMR1IF

*Bits to be declared in the main program

Flags to be checked

The deference between the high and low priority interrupt can be highlighted in the following points:

1. The interrupt vector addresses are 0008h in high-priority and 0018h in low-priority.
2. W, STATUS, and BSR registers are saved automatically on the stack in high-priority and not saved in low-priority (must be saved in data registers at the beginning of the ISR and retrieve before the end)
3. The end instructions of the ISR are "RETFIE FAST" in high-priority and "RETFIE" in low-priority interrupt.
4. INTCON<7> is set for high-priority and INTCON<6> in low-priority.

Use the template program below in the case of employing low-priority multiple interrupt sources. Notice that there are two ISR TMR1_ISR and TMR2_ISR with two separate data saving instructions for W, STATUS, and BSR. The question is can we use only one set of data save instructions for all low-priority interrupt service routines? The key to this is to test the interrupt flags and skip once again inside the ISR and branch to the target ISR. Try to find out if it is possible or not.

```
            ORG       0X00
            GOTO      MAIN
            ORG       0X0018                 ; Low-priority interrupt vector
            BTFSC     PIR1, TMR1IF           ; Check Timer1 flag
            GOTO      TMR1_ISR
            BTFSC     PIR2, TMR2IF           ; Check Timer2 flag
            GOTO      TMR2_ISR
MAIN:       ;;;;;;    ;;;;;;                 ; main assembly program start here
            CLRF      PIR1                   ; Good habit to clear all flags
            BSF       RCON, IPEN             ; Enable priority RCON<7>=1
            BSF       INTCON, GIEL           ; Enable global low-priority INTCON<6>=1
            BCF       IPR1, TMR1IP           ; Timer1 Low-priority
            BSF       PIE1, TMR1IE           ; Enable Timer1 overflow interrupt
            BCF       PIR1, TMR2IP           ; Timer2 Low-priority
            BSF       PIE1, TMR2IE           ; Enable Timer2 overflow interrupt
WAIT:       GOTO      WAIT                   ; Wait for an interrupt
TMR1_ISR:   MOVFF     STATUS, STATUS_TEMP    ; Save registers
            MOVWF     WREG_TEMP
            MOVFF     BSR, BSR_TEMP          ; BSR_TMEP located anywhere
            BCF       PIR1, TMR1IF           ; Clear TMR1 flag
            ;;;;      ;;;;;                  ; Your interrupt service routine
            ;;;;      ;;;;                   ; code is placed here
            MOVFF     BSR_TEMP, BSR          ; Restore BSR
            MOVF      WREG_TEMP, W           ; Retrieve registers
            MOVFF     STATUS_TEMP, STATUS
            RETFIE                           ; Go back to MAIN
TMR2_ISR:   MOVFF     STATUS, STATUS_TEMP    ; Save registers
            MOVWF     WREG_TEMP
            MOVFF     BSR, BSR_TEMP          ; BSR_TMEP located anywhere
            BCF       PIR1, TMR2IF           ; Clear TMR2 flag
            ;;;;      ;;;;;                  ; Your interrupt service routine
            ;;;;      ;;;;                   ; code is placed here
            MOVFF     BSR_TEMP, BSR          ; Restore BSR
            MOVF      WREG_TEMP, W           ; Retrieve registers
            MOVFF     STATUS_TEMP, STATUS
            RETFIE                           ; Go back to MAIN
            END
```

It is recommended to have a delay of 4 instruction cycles between each interrupt.

TIMER APPLICATIONS

PICs generally have several timers or counters, each with different capabilities. For the PIC18F4431, Timer0, Timer1, and Timer5 can be used as counters as well as timers, while Timer2 can only be used as a timer. The difference between using a Timer in timer mode or counter mode is simply the source of the pulses — a timer runs off the system clock, while a counter increments when it sees a rising/falling edge on a certain pin. Timers on the PIC can only count up. Timer2 and Timer5 can generate interrupts on a period match (Interrupt on TMR2 match with PR2 and Interrupt on TMR5 match with PR5). In digitally controlled systems, timers are considered essential part of its operation. There are several applications of timers such as time delay, pulse waveform generation, pulse width measurement, frequency measurement, and counting events. We will examine briefly the generation of the time delay and the pulse wave using TIMR0. The set up of the features and the mode of operation for each of the above mentioned timers requires an additional control register for each timer. The control registers associated with TIMER0, TIMER1, TIMER2, and TIMER5 are given below:

TIMER0 < > T0CON	TIMER1 < > T1CON
TIMER2 < > T2CON	TIMER5 < > T5CON

Please refer to the datasheet PIC18F2331/2431/4331/4431 for a detailed explanation of the operation mode and the features of these timers and the definition each bit of the associated control registers.

TIMER0 can be set as timer/up-counter, 8-bit/16-bit timer with 8 options of pre-scale values (bit 2-0 in T0CON). It is readable and writable and generates an interrupt when overflows from 0xFF/0xFFFF to 0x00/0x0000. To generate a time delay we have to load a count for a given delay. When the count reaches 0xFF/0xFFFF the interrupt flag will be set and the delay time in this case is the time between loading and the appearance of the flag. When the timer runs on the internal clock, it is updated on every instruction cycle (1 instruction cycle= 4 internal clock cycles).

Assume the internal clock for the MCU is set to 8MHz and TIMER0 in 16-bit mode is set to generate a high-priority interrupt every 700ms. If the desired pre-scale is 1:64, what are the necessary calculation steps to achieve this delay?

1. The internal clock period T is = 1/8MHz = 125ns
   The instruction period = 4T = 125ns x 4 = 500ns
2. Number of instruction cycles needed to generate 700ms = 700ms/500ns = 1400000
3. For pre-scale of 1:64 the count is = 1400000/64 = 21875
4. Since the timer counts only up from a loaded number until 0xFFFF(or 65535d), and rolls over to 0x0000, therefore the pre-loaded number should be 65535−21875=43660 or AA8CH
5. Verify the time delay Td   Td = instruction period x pre-scale x count
   Td = 500ns x 64 x 21875 =700ms

The flowing program uses TIMER0 to realize 700ms delay (hardware delay) with high –priority interrupt.

```
             ORG      00
             GOTO     MAIN
             ORG      0x0008          ; High-priority interrupt vector
             MOVLW    0xAA            ; Load high count of 0xAA8C
             MOVWF    TMR0H           ; Load high count in Timer0 high register
             MOVLW    0x8C            ; Load low count of 0xAA8C
             MOVWF    TMR0L           ; Load low count in Timer0 low register
             BCF      INTCON, TMR0IF  ; Clear TMR0 overflow flag
             RETFIE   FAST            ; go back to wait (main program)
MAIN:        ;;;;;    ;;;;;           ; Program starts here
             BSF      RCON, IPEN      ; Enable priority
             MOVLW    B'11100000'     ; Set global and peripheral priority, Enable TIMER0
             MOVWF    INTCON,1        ; overflow interrupt bit
             BSF      INTCON2, TMR0IP ; Set Timer0 as high-priority
             MOVLW    B'10000101'     ; Enable Timer0 16-bit mode and
             MOVWF    T0CON           ; pre-scale 1:64
Delay_700ms  MOVLW    0xAA
             MOVWF    TMR0H
             MOVLW    0x8C
             MOVWF    TMR0L           ; Your interrupt service routine
             BCF      INTCON, TMR0IF  ; Clear TMR0 overflow flag and start counter
WAIT:        GOTO     WAIT
             END
```

EXPERIMENT

PART 1: EXTERNAL INTERRUPT

1. Connect the socket which is mounted on the pre-built circuit board of experiment 1 to the matched socket's pins on the transition socket.
2. Connect the socket pins labeled RC3/INT0, RC4/INT1, and RC5/INT2 on the pre-built board to the pins # 18, 23, and 24 respectively on the transition socket.
3. Set the selector switches SEL1, SEL2, and SEL3 to position 2 see figure (1.4) in lab1.
4. Set pins 18, 23, and 24 on the transition socket as high priorities, falling edges external sources of interrupt.
5. Use the 8 LEDs as a binary Display. Write assembly program to increment/decrement the binary counter (count up/down) when SWT1/SWT2 is pushed respectively. The repeated push of SWT1 will be shown on the display as binary count up, similarly SWT2 as binary count down.

PART2: TIMERS AND INTERNAL INTERRUPT

Consider an automatic packaging system consist of a conveyer belt and a packaging machine in a marble-packaging factory. The Packaging line is set to package each batch of 4 marbles in a box and the packaging machine requires 300ms to complete packaging the batch in box. There is a shutter installed in the conveyer to stop the flow of the marble during the packaging machine busy period. A sensor placed on the conveyer to count the marbles as shown in figure (3). Also the packaging line equipped with buzzer that sound an alarm during the busy period of packaging machine (300ms).

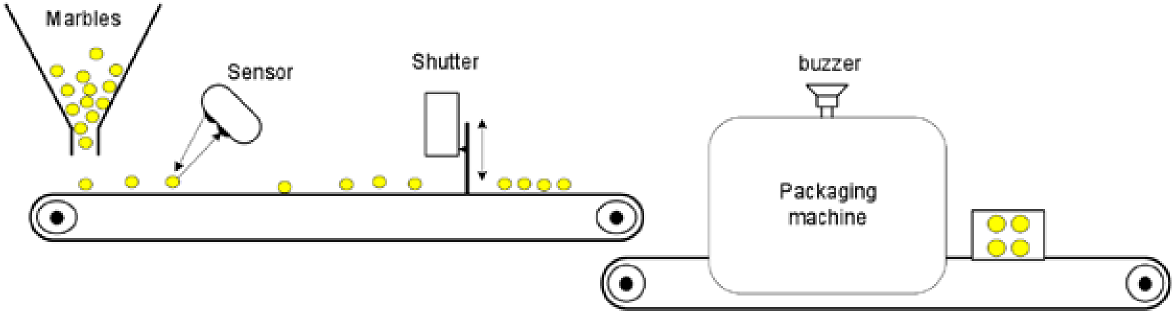


Figure 3

The available hardware devices are TIMER0, TIMER1 and TIMER2, in addition to 3 external interrupts INT0, INT1 and INT2. Port B connected to 8 LEDs with different color (RYGW). SWT1, SWT2 and SWT3 will be deselected when SEL1, SEL2, and SEL3 are set at position 2 respectively. This will make the external interrupt pins RC3, RC4, and RC5 available, see schematic diagram (1.4) in Lab1. Design an interrupt-driven interface for this system using any of the available devices and/or port terminals. For example one can use TIMER0 to implement delay of 300ms and RC4 to interface the phototransistor, RC3 and RC5 for buzzer and shutter control. Make your own choice clear and write it down on the top of your program. To simplify the connections, an interface board is labeled and attached to the setup. Signals from PIC18 emulator socket and the power supply wires are to be connected.

FURTHER WORK (OPTIONAL)

1. Rewrite the pedestrian traffic light interface of lab1 and make use of INT0, INT1, INT2, TIMER0, and TIMER1.
2. Use external interrupt (any one INT0, INT1 or INT2) to measure the RPM of motor shaft.

A photo-reflective sensor figure (a) below can be used to sense the position (velocity and acceleration) of a rotating shaft by simply attaching reflective tape to the shaft. As the shaft rotates, a pulse is generated every time the reflective tape passes in front of the sensor. Another alternative way is to attach a disc with opening or black and white strips. A pulse will be generated whenever object interrupt, reflect a light or IR beam passing across the gap of the sensor as shown in figure (b).

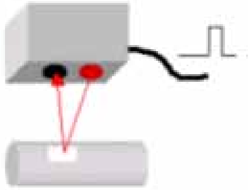


Figure 3-2a: Photo Reflective Sensor

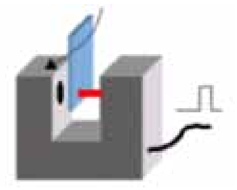
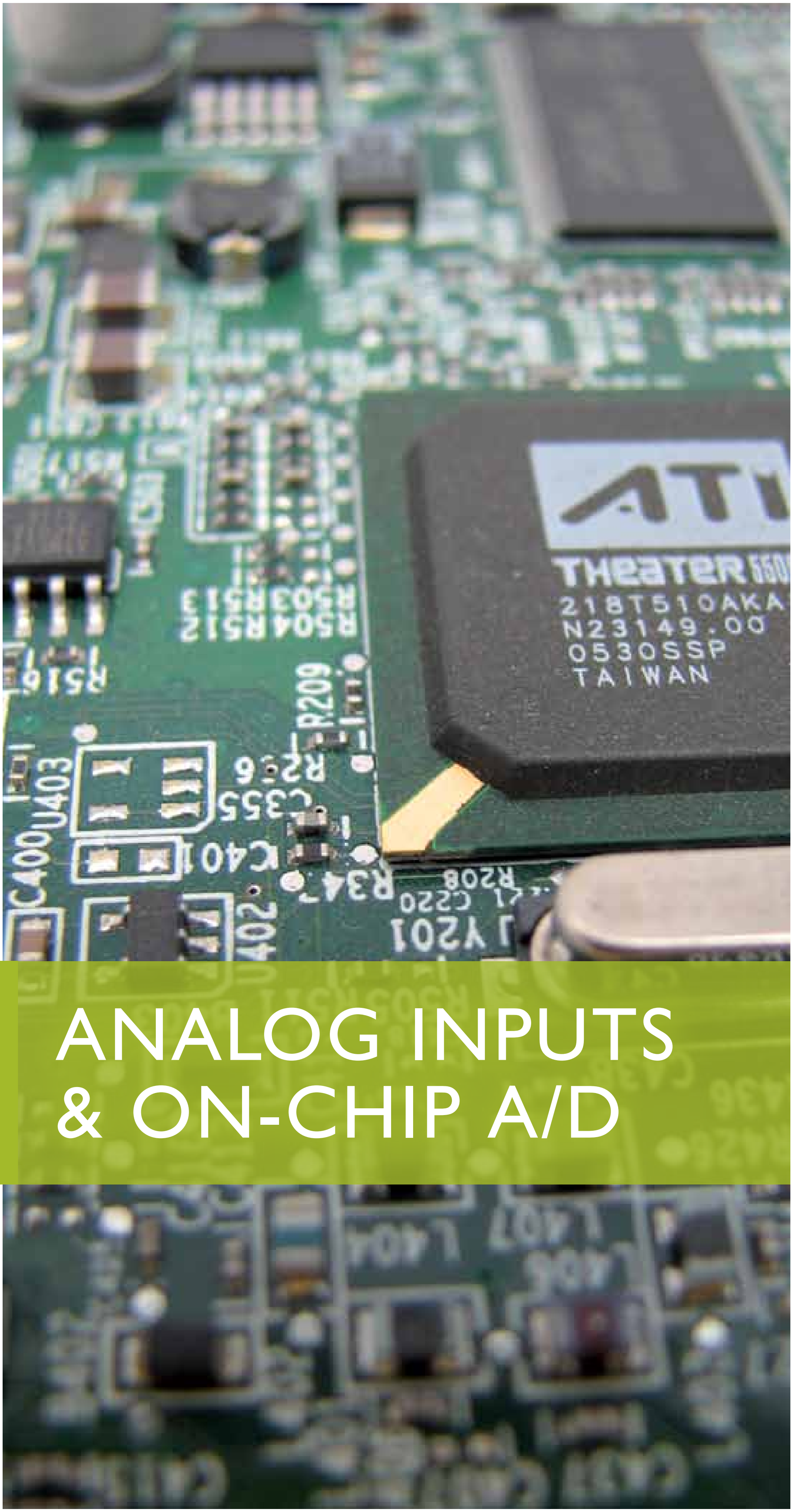


Figure 3-2b: Interrupter



EXPERIMENT 4

ANALOG INPUTS & ON-CHIP A/D

OBJECTIVE

Explore the on-chip analog-to-digital converter (ADC), learn how to select and configure analog channels, learn how to configure the ADC to work with analog sensors such as the LM35 series (precision integrated-circuit temperature sensors), and finally interface, convert and display, the real-time analog signal.

INTRODUCTION

Observing any natural events, such as lightening, storm, change in pressure or temperature over a very short period of time (time window) shows (reveal that) that the change is gradual and continuous in shape (analog form) over time. For example lightening, the static discharge between clouds and earth can be measured as continuous rising then falling of an electric current in a very short time.

Pressure, temperature, light, audio signal (microphone), force and acceleration are measured by analog devices known as sensors, transducers and pickups. When these are configured in a circuit, it will produce an analog signal (output voltage) proportional to the change of temperature, pressure, etc.

Microcontrollers (computers) are digital in nature. They are designed and built of digital circuits. The digital circuit handle signals consist of only two levels HIGH or LOW, logic 1 or logic 0. In Logic system, particularly TTL (transistor transistor logic), the signal threshold is 0.8V. When signal level less than 0.8v (0 to 0.8), it considered as logic 0 and when signals level greater that 0.8 (up to 5V max), it considered as Logic 1. So we cannot process or apply analog signal directly to or from MCU. To solve this problem MCU have on board or built-in Analog-To-Digital converter (ADC). The analog signal at the input of this module will come out as 1s or 0s. This enables us to easily interface all sort of analog devices with MCUs. PIC18F4431 MUC has a high-speed 10-bit ADC module that support 9 input channels labeled as AN0 to AN8. ADC module has 9 registers:

ADRESH:ADRESL	where the result is loaded at the completion of A/D conversion.
ADCON0:ADCON3	4 control register
ADCHS	Channel Select Register
ANSEL0:ANSEL1	2 Analog I/O select registers

As shown in the figure below reference voltage Vref- , Vref+ specifies the minimum and the maximum range of analog input respectively. The reference voltage is software selectable (register ADCON1 controls the voltage reference settings) to either the device‘s positive and negative analog power supply voltage (AVDD and AVss) or external source.

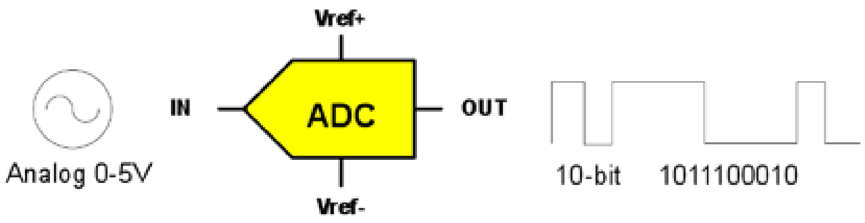


Figure 4.1

As we pointed out the operating range (input range) of ADC can be set to 0-5V, which restrict the input signal swing to be bounded to 0V to 5V. Any part of input signal outside this range will be lost.

SIGNAL CONDITIONING

To make best use of the ADC, the input voltage should traverse as much of its input range as possible, without exceeding it. Most signal sources, say a microphone or thermocouple, produce very small voltages. Therefore, in many cases amplification is needed to exploit the range to best effect. Voltage level shifting may also be required, for example if the signal source is bipolar (positive and negative) while the ADC input is unipolar (voltage is positive only). If the signal being converted is periodic, then a fundamental requirement of conversion is that the conversion rate (PIC18 sampling rate 200KHz) must be at least twice the highest signal frequency. This is known as the Nyquist sampling criterion. If this criterion is not met, then a deeply unpleasant form of signal corruption takes place, known as aliasing If there are large number of analog input (more than the available 9 channels) to be processed by MCU, such case exist in data logging or data acquisition systems, then analog multiplexer (semiconductor device) is used. The multiplexer acts like selector switch, sequentially select one input, say out of N inputs and connect to one analog channel of the DAC. It is important to know that the analog multiplexer is not an Ohmic switch (a mechanical switch with 0 resistance when it is ON), when switched 'ON", it has an internal series resistance, which can range from tens to thousands of ohms. This must be taken into the consideration during input data process.

ADC RESOLUTION & REFERENCE VOLTAGE

Resolution (quantization step size) specifies how accurately the ADC measures the analog input signal. Common ADCs are: 8-bit, 10 bit and 12 bit. If the reference voltage is selected as the device's power supply 0-5V (Vref-=0V and Vref+=5V), then resolution of each ADC's are:

1. 8-bit range 0-265 it can measure input signal as small as (step size=) 5V256=19mV
2. 10-bit range 0-1024 it can measure input signal as small as 5V1024=4.8mV
3. 12-bit range 0-4095 it can measure input signal as small as 5V4095=1.2mV

We can see that the 8-bit ADC can't tell the difference between 1mV and 18mV input signal since this input signal range lies within the step size of 19mV. So the digital output of the ADC will stay same as previous level no increment or decrement. Compare this to the 10-bit ADC, the change of 18mV in the input signal level will make ADC's digital output to jump 3 digital levels (steps). So the smaller steps size the higher resolution and in return more accurate digital representation of the input signal.

Bit 7-6 (VCFG1-VCFG0) in the A/D control register ADCON1 are the selection bits for the A/D V_{ref} (AV_{ref}) source selection. Example, if VCFG1=0 and VCFG0=0, this makes $V_{ref+}=AV_{DD}$ and $V_{ref-}=AV_{ss}$. The V_{ref-} & V_{ref+} pins are available in pin #4 & pin #5 of the PIC18F4431 chip.

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ADCON1	VCFG1	VCFG0	—	—	—	—	—	—

ADC CHANNEL SELECT & PIN FUNCTION

The A/D channels are grouped into sets of 2 or 3 channels. For PIC18F4431 device, the channels group are as follows:

Group A	AN0, AN4 and AN8	(GASEL1:GASEL0 in ADCHS register)
Group B	AN1 and AN5	(GBSEL1:GBSEL0 in ADCHS)
Group C	AN2 and AN6	(GCSEL1:GCSEL0 in ADCHS)
Group D	AN3 and AN7	(GDSEL1:GDSEL0 in ADCHS)

To set pin AN4 of the PIC18F4431 as analog input, you must locate its group (Group A), then set bit GASEL1=0 and GASEL0=1. Next, in the ANSEL0 register, set bit 3 to 1 (ANS3=1). This will disable the digital input buffer. Finally set the corresponding TRISA bit 3 for an input.

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ADHS	—	—	—	—	—	—	GASEL1=0	GASEL0=1
ANSEL0	—	—	—	—	ANS3=1	—	—	—

Channels in each group are selected as analog channels by setting the corresponding two bits in the ADCHS (A/D channel select). Example clearing all bits of the ADCHS register, will select channels AN0, AN1, AN2, and AN3 as analog channels.

```
MOVLW     0x04
MOVWF     TRISA                ; set pin RA4 of Port A as inputs
BCF       ADCHS, GASEL1
BSF       ADCHS, GASEL0        ; select analog channels AN0, AN1, AN2, AN3
BSF       ANSEL0, ANS3         ; select AN4 as analog input
```

ACQUISITION TIME

The process of ADC includes; applying analog signal to the selected input channels, then passing this input into a sample and hold circuit (S&H circuit found inside ADC module). The S&H takes a sample of the analog input voltage, like a snapshot or a slice, and then holds it steady for the duration of the conversion. Please refer to figure (20-0) in PIC18F4431 datasheet. For the A/D converter to meet its specified accuracy, the charge holding capacitor (CHOLD) must be allowed to fully charge to the input channel voltage level. When the conversion is started, the holding capacitor is disconnected from the input pin (which may be changing) and is used as a fixed “copy” of the input being sampled, during the successive approximation process. So before starting conversion you must give this capacitor time to charge. This is referred to as the acquisition or sampling time T_{ACQ}. In other words, the channel must be sampled for at least the minimum acquisition time (T_{ACQ}) before starting conversion. T_{ACQ} is also defined as the delay required before each A/D conversion. For PIC18F4431, the minimum required acquisition time (T_{ACQ}) is 0.75us (Please refer to PIC18F4431 datasheet for detailed specification).

ADC CLOCK

ADC requires a clock source to do its conversion, this is called ADC clock. The time period of the ADC clock is called T_{AD}. It is also the time required to generate 1 bit of conversion. The ADC operates by successive approximation, this means that the input voltage is fed to a comparator, and if the input voltage is higher than 50% of the range (reference), the MSB of the result is set high. The voltage is then checked against the mid-point of the remaining range, and the next bit set high or low accordingly, and so on for 10 bits. This takes a significant amount of time. IF the A/D conversion time per bit is defined as T_{AD}. The A/D conversion requires 12 T_{AD} per 10-bit conversion. For correct A/D conversions, the ADC clock (T_{AD}) must be as short as possible, but greater than the minimum T_{AD}. The minimum value for T_{AD} is 385 ns (see table25-21 in datasheet). It can be derived from the MCU clock (T_{OSC}) by dividing by suitable division factor. The device operating frequency or the MCU clock is divided by 2, 8, 16, 32, or 64 as necessary to satisfy the minimum time requirement for T_{AD}. Assume the MCU system is clocked at 8 MHz. This gives a clock period of 0.125 μs. We need a conversion time of at least 0.385 μs, if we select the divide by 4 option, the ADC clock period will then be 4x0.125 μs =0.5 μs, which is just longer than the minimum required.

The division factor for PIC clock (T_{OSC}) and the acquisition time (T_{ACQ}= 0.75 us) can be selected using A/D control register2 (ADCON2). In the above example we have;

T_{OSC} = 0.125us Division factor of 4 to satisfy the minimum time requirement for T_{AD}.
T_{AD} = 0.385us Multiply by 2 to satisfy the minimum time requirement for T_{ACQ}.
T_{ACQ} = 0.75 us

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ADCON2	ADFM	0	1	1	0	0	0	1

TRIGGERING A/D CONVERSIONS

The PIC18F4431 device is capable of triggering conversions from many different sources. The same method used by all other microcontrollers of setting the GO/DONNE bit still works. Sources are:

- RC3/INT0 pin
- Timer5 Overflow
- Input Capture 1 (IC1)
- CCP2 Compare Match
- Power Control PWM rising edge

These triggers are enabled using the SSRC<4:0> bits in ADCON3 register. Any combination of the five sources can trigger a conversion by simply setting the corresponding bit in ADCON3. When the trigger occurs, the GO/DONNE bit is automatically set by the hardware and then cleared once the conversion completes.

ADC RESULT REGISTER

The 10-bit result of the conversion is stored in two registers: ADRESH and ADRESL. This register is 16-bits wide. There are two ways to store the 10-bit ADC output in 16-bit result register: Left-justification and Right-justification. The A/D Result Format select bit “ADFM” is bit7 in the A/D control register ADCON2.

ADRESH							
R9	R8	R7	R6	R5	R4	R3	R2

ADRESL							
R1	R0	0	0	0	0	0	0

If ADFM=0, the result is left-justified, with the most significant 8-bits of the result in the ADRESH, and the least significant 2-bits of the result in the upper two bits of ADRESL. The unused 6-bits read as “0”.

The left-justification is useful when you are not concern with the full 10-bit resolution; you can simply treat ADCRESH as holding 8-bit result, and ignore the least significant two bits held in ADCRESL.

ADRESH							
0	0	0	0	0	0	R9	R8

ADRESL							
R7	R6	R5	R4	R3	R2	R1	R0

If ADFM=1, the result is Right-justified, with the least significant 8-bits of the result in the ADRESL, and the most significant 2-bits of the result in the lower two bits of ADRESH. The unused 6-bits read as “0”.

The right-justification is useful when you want to consider the full 10-bit resolution. All calculation in this case will be 16-bit arithmetic operations. If only 8 bits resolution is required, the process can be simplified. If the result is right justified, the low 8 bits ADRESL will record the low bits of the conversion, meaning that only the analog signal up to 25% of the full range (defined Vref-&Vref+) will be processed, but at full resolution. If the result is left justified, the high byte will be processed, which will represent the full analog signal range, but at reduced resolution. Example set the reference voltage Vref+= 2.046V, Vref-=0, this give a range of 0-2.046V. With this reference voltage, and maximum binary result of 1023, the conversion factor will be 2046/1023=2mv per bit. If the maximum value of the analog signal is 500mv, then 8-bit result from ADRESL will be at full resolution, because 500mv input signal is 25% of the range 0-2.046V.

ADC MODULE INITIALIZATION STEPS

The following steps should be followed to initialize the A/D module:

1. Configure the A/D module:
    a. Configure analog pins , voltage reference and digital I/O
    b. Select A/D input channels
    c. Select A/D Auto-conversion mode (single-shot or continuous loop)
    d. Select A/D conversion clock
    e. Select A/D conversion trigger
2. Configure A/D interrupt (if required):
    a. Set GIE bit
    b. Set PEIE bit
    c. Set ADIE bit
    d. Clear ADIF
    e. Select A/D trigger setting
    f. Select A/D interrupt priority
3. Turn ON ADC
    a. Set ADON bit in ADCON0 register
    b. Wait the required power-up setup time about 5-10 μs
4. Start sample/conversion sequence:
    a. Sample for minimum of $2T_{AD}$ and start conversion by setting the GO/DONNE bit. The GO/DONNE bit is set by the user in software or by module if initiated by a trigger.
    b. If T_{ACQ} is assigned a value (multiple of T_{AD}), then setting the GO/DONNE bit starts a sample period of the T_{ACQ} value, then starts a conversion.
5. Wait for A/D conversion/conversions to complete using one to complete using one of the following options:
    a. Poll for the GO/DONNE bit to be cleared if in Single-shot mode.
    b. Wait for the A/D interrupt flag (ADIF) to be set.
    c. Poll for the BEEMT bit to be cleared to signify that at least the first conversion has completed.
6. Read A/D results, clear ADIF flag, reconfigure trigger.

BINARY TO BCD CONVERSION

The conversion of 8-bit result is much easier than handling the 16-bit arithmetic operation when considering the 10-bit result. For this reason, the binary to BCD algorithm will be implemented for 8-bit result. The conversion method we will adopt is called successive subtraction. Successive subtraction: subtract the number successively so that the hundred, ten and unit locations can be recovered. The method code is given below:

```
                MOVWF       RESULT              ; get ADC result
                CLR         huns                ; zero huns digit
                CLRF        tens                ; zero tens digit
                CLRF        ones                ; zero ones digit
; hundreds digit
                BSF         STATUS, C           ; set carry for subtraction
                MOVLW       D'100'              ; load 100
sub1            SUBWF       RESULT              ; subtract 100 from the result
                INCF        huns                ; count number of loops
                BTFSC       STATUS, C           ; check if carry is cleared "done"
                GOTO        sub1                ; no, go back and subtract 100 again
                ADDWF       RESULT              ; yes, add 100 back on
                DECF        huns                ; and correct loop count
; tens digit
                BSF         STATUS, C           ; set carry for subtraction
                MOVLW       D'10'               ; load 10
sub2            SUBWF       RESULT              ; subtract 10 from the result
                INCF        tens                ; count number of loops
                BTFSC       STATUS, C           ; check if carry is cleared "done"
                GOTO        sub2                ; no, go back and subtract 10 again
                ADDWF       RESULT              ; yes, add 10 back on
                DECF        tens                ; and correct loop count
; ones digit
                MOVF        RESULT              ; load the reminder
                MOVWF       ones                ; and store as ones digit
```

BCD TO ASCCII CONVERSION

In BCD, a numerical digit is usually represented by four bits which, in general, represent the decimal digits 0 through 9. The conversion of 0 to 9 into ASCII can be done by adding 0x30H to each digit, for example the BCD digit 9 is 0x39H in ASCCII. The conversion code is given below:

```
MOVLW     030                   ; load ASCII offset
ADDWF     huns                  ; convert hundreds to ASCII
ADDWF     tens                  ; convert tens to ASCCII
ADDWF     ones                  ; convert ones to ASCCII
```

The program outline for 8-bit conversion is given below

```
ADC using low 8-bits
        Convert the analogue input to 8-bits and display
        Hardware: P18F4431 (8MHz), Vref+ = 2.048V or 4.096V, 16x2 LCD
Initialise:
        PortA = Analogue inputs
        PortB = LCD outputs
        ADC = Select f/4, RA0 input, right justify result, enable
        LCD = default setup (include LCD driver routines in lab2)
Main:
        REPEAT
                Get ADC 8-bit input
                Convert to BCD
                Display on LCD
        ALWAYS
Subroutines:
        Get ADC 8-bit input
                Start ADC and wait for done
                Store result
        Convert to BCD
                Calculate hundreds digit
                Calculate tens digit
                Remainder = ones digit
        Display on LCD
                Cursor position
                Convert BCD to ASCII
                Send hundreds, tens, decimal point, ones
Include:        LCD routine
```

EXPERIMENT

A hydraulic pump located at the base of a Dialysis Machine requires a temperature control system to maintain the temperature of the hydraulic fluid under 29 °C. The nominal operating temperature of the fluid is 20 °C. When the temperature reaches 22 °C a cooling fan starts. As the temperature rise up to 25 °C, a second cooling fan starts. When the fluid temperature reaches 30 °C an audible buzzer come off. The temperature is measured by a precision Centigrade Temperature Sensors LM35D of range 0 to 100 °C, and 10 mV/ °C.

- Use the pre-built circuit board shown in figure (4.2), the LCD, and the PIC18F4431 emulator to design an ON/OFF temperature controller system.
- Calculate the suitable external reference voltage Vref+ for using the lower 8-bit ADC result.
- Connect the power source (+ 5V) to the circuit board.
- Adjust the potentiometer 1 to obtain the reference voltage.
- Insert the socket of the circuit board into the breadboard with the matched pin number of the emulator socket.
- Use AN4, the analog input channel of PORTA for temperature sensor, PORTB for the LCD, and RD0-RD1 to control the cooling fans.

Please refer to the PIC18F4431 datasheet to section “20.3 A/D Acquisition Requirements“ pages 255-257 and to the temperature sensor LM35D datasheet in the appendix.

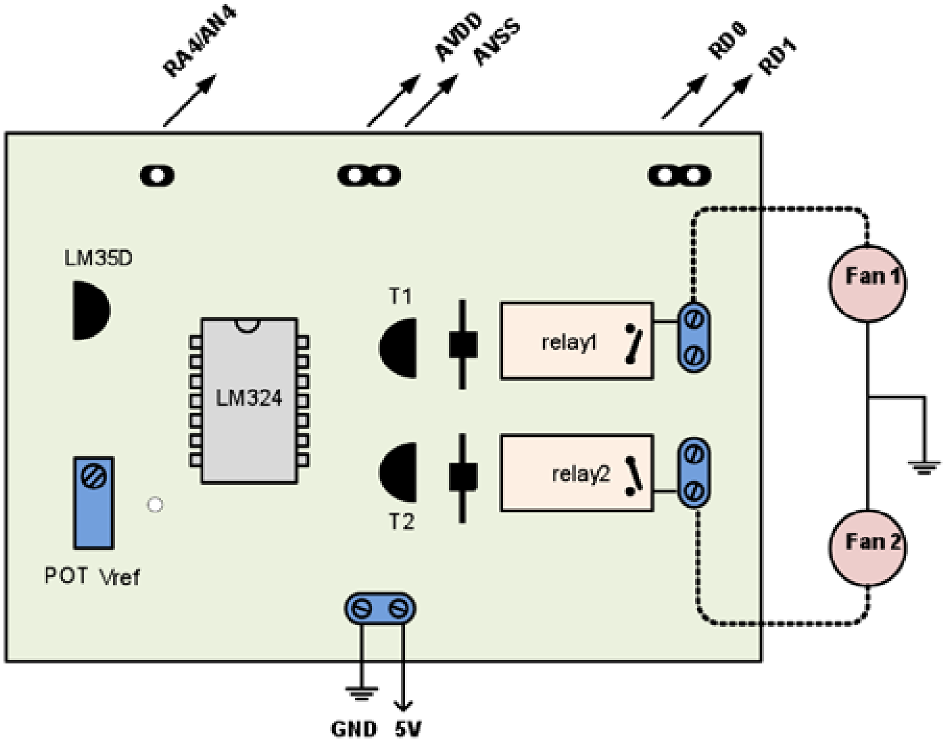
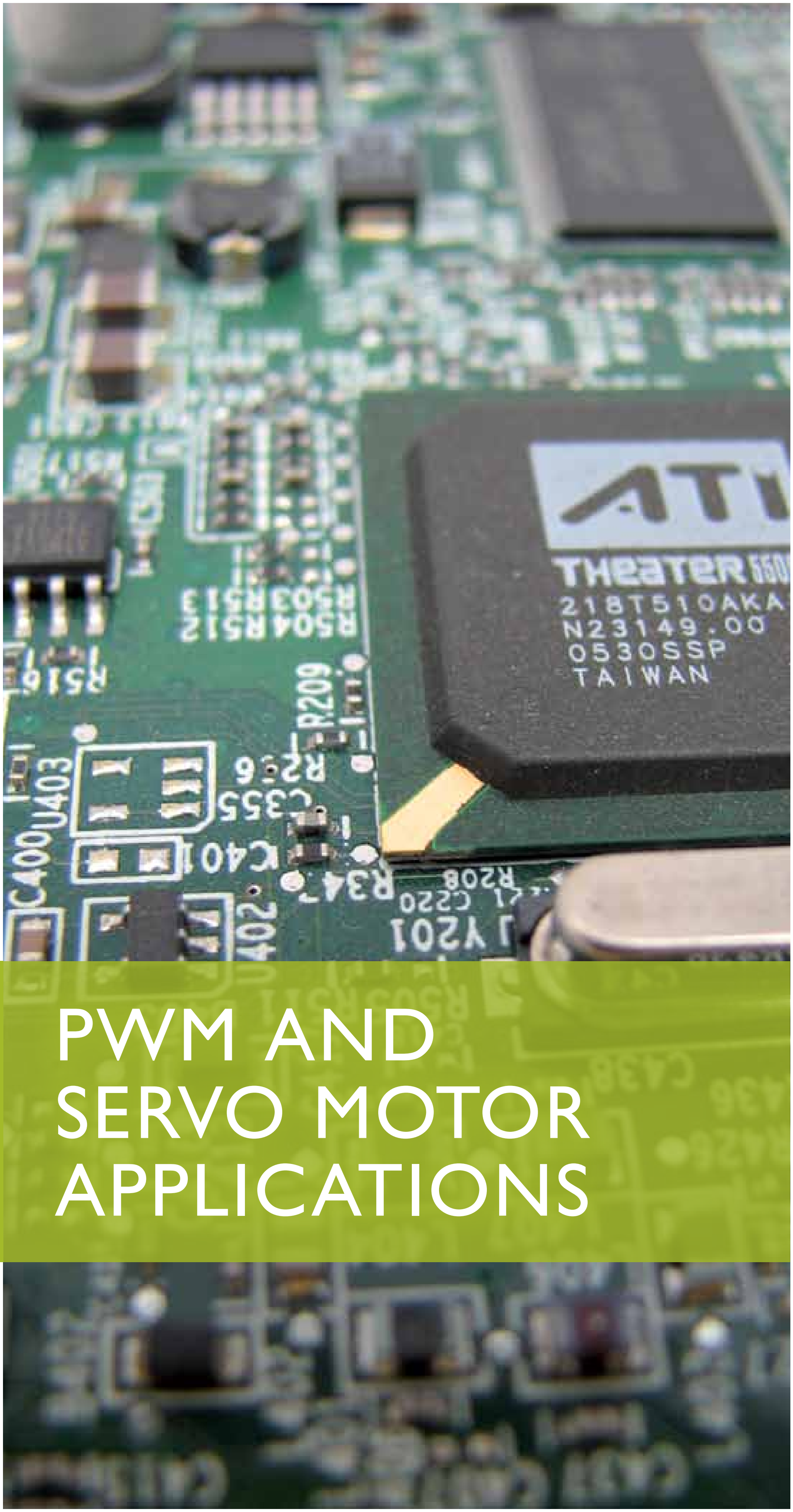


Figure 4.2



EXPERIMENT

5

PWM AND SERVO MOTOR APPLICATIONS

OBJECTIVE

Learn about servo, step, and DC motors, learn how to generate pulse waveforms, and Pulse Width Modulation (PWD), understand the concept of Capture, Compare, and PWM (CCP) modules, and Write instructions to generate pulse waveforms and PWM.

INTRODUCTION

DC MOTORS

A dc motor is an electric motor that runs on direct current (dc) electricity. It works on the principle of the force or torque experienced between two magnetic fields. One of the two magnetic fields is generated directly from the DC power supply (electrical magnet), and the other from stationary magnets. The DC motors can be classified into two groups:

1. The brushed DC motor generates torque directly from DC power supplied to the motor by using internal commutation, stationary permanent magnets, and rotating electrical magnets. The brushed DC motor is characterized as low cost, simple control of motor speed, low life-span for high intensity use, and high maintenance.

2. The brushless DC motors (BLDC motors, BL motors) use a rotating permanent magnet in the rotor, and stationary electrical magnets on the motor housing. Also known as electrically commutated motors powered by DC and having electronic commutation systems, rather than mechanical commutation and brushes. The current-to-torque and voltage-to-speed relationships of BLDC motors are linear.

SYNCHRONOUS ELECTRIC MOTOR

Asynchronous electric motor is an AC motor distinguished by a rotor spinning with coils passing magnets at the same rate as the alternating current and resulting rotating magnetic field which drives it. An alternative definition is that it has zero slip under usual operating conditions. The brushless DC motors are similar to AC synchronous motors. The major difference is that synchronous motors develop a sinusoidal back EMF, as compared to a rectangular, or trapezoidal, back EMF for brushless DC motors. Both have stator created rotating magnetic fields producing torque in a magnetic rotor.

STEP MOTOR AND SERVO MOTRO

A stepper motor (step motor) is a brushless, synchronous electric motor that can divide a full rotation into a large number of steps. The rotor moves in discrete steps as commanded, rather than rotating continuously like a conventional motor. When stopped but energized, a step motor holds its load steady with a holding torque. The motor's position can be controlled precisely without any feedback mechanism (see figure below). The predecessor of the step motor was the servo motor. A servo is an automatic device that uses error-sensing feedback to correct the performance of a mechanism. A closed feedback servo system is shown in the figure below.

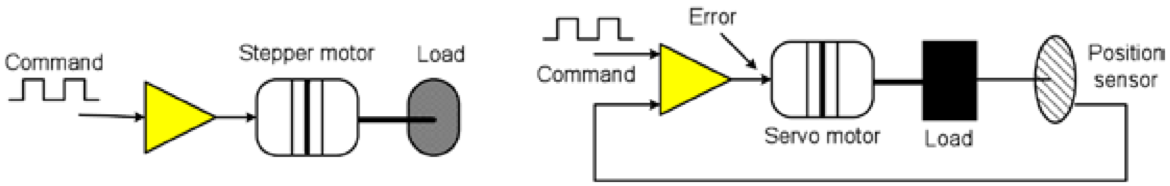


Figure 5.1

The servo motor is widely used in model hobbyist such as airplane R/C model for moving the rudder, ailerons, elevators and acceleration control or in the car R/C model for steering and acceleration control.

The servo motor basically is a high quality geared DC motor equipped with electronic circuit for controlling the DC motor rotation direction and position. Currently there are two types of servo motor available on the market, the first one is called standard servo and the other one is called continues servo; standard servo can rotate to maximum (clockwise or counter-clockwise) of 120 degrees while continues servo can rotate up to 360 degrees in both direction.

Servo motor shaft is positioned with pulse width modulated signals (PWM). All servos come with three wires, Red Power +V supply, Black Ground -V supply, and Whit PWM control signal. Usually in hobby servos with rotation angle 45°/90°/120° signal width varies between 1 and 2ms. The servo expects to see a pulse every 20ms (period 1/50Hz=20ms second). The signal you are going to give the servo is one that is high (5V) for 1-2ms and low (0V) for the remainder of the 20ms period. The duration of the high signal determines the position that the servo attempts to maintain. Note that the servo must continually receive this signal in order to maintain its position. The 20ms period is called the refresh rate of the control pulses. Information about the PWM control signal is given in table1.

The refresh rate of the control pulses and the servo motor feedback will ensure that the servo does not drift from the desired position. If there is too much time between control signals, the servo will drift, and too fast a signal will make the servo chatter. Check the datasheet for your servo's optimal refresh.

Table 1 shows the excel rotation angels of the servo motor versus the high pulse width.

Table 1

Pulse Width	Period	Duty Cycle	Servo Motor Position
1.0 ms or <	20ms	5% or < 7.5%	+45° (clockwise rotation) ø
1.5 ms	20 ms	7.5%	Zero position (neutral position) ⌽
2.0 ms or >	20 ms	10% or > 7.5%	-45° (contraclockwise rotation) ⦸

Please note that stepper and brushless DC motors require a controller cards to operate. In case of stepper DC motor, the required controller is called a driver card. The digital sequence from microprocessor or MCU is applied to the driver card rather than apply directly to the winding of the step motor. Applying the control signal from MCU directly to the winding of a step motor will damage the MCU PORT due to the limited fan-out (current supplied by MCU PORT circuit) of PORT circuit.

Table 2: Comparison between 4 types of DC motors

Type	Advantage	Disadvantage	Typical Applications	Drive
Stepper DC	Precision positioning; high holding torque	High initial cost; requires a controller	Positining in printers and floppy drives	DC
Brushless DC	Long lifespan; low maintenance; high efficiency	High initial cost; requires a controller	Hard drives, CD/DVD players, electric vehicles	DC
Brushed DC	Low initial cost; simple speed control	Maintenance (brushes); medium lifespan	Treadmill exercisers, automotive motors (seat, blowers, windows)	Direct DC or PWM
Pancake DC	Compact design; simple speed control	Medium cost; medium lifespan	Office equipment, fans/ pumps	Direct DC of PWM

PIC18 SQUARE PULSE GENERATOR

Square pulse can be generated as follow: set bit 0 of PORTA or any digital port, stay on for a certain period of time by using delay1, reset bit 0 the PORTA, and stay for another period or delay2 then loop back to set bit 0 again. This approach is not efficient but it can be used once or twice in the program to trigger some event (such as flashing LED or triggering sound alarm or closing gates). The “on” delay1 and the “off” delay2 will be used in definition of the duty cycle, the period, and the frequency of the square pulse or in general of any digital signal.

Duty cycle = Delay1/(Delay1 + Delay 2) * 100%

Period = Delay1 + Delay2

Frequency = 1/(Delay1 + Delay2)

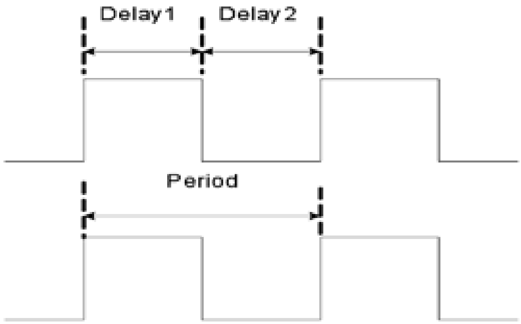


Figure 5.2

From the figure above, a square pulse with 50% duty cycle can be generated by setting delay1=delay2. For a 10% duty cycle delay1 will be set to 10ms for example and delay2 to 90ms. For a 70% delay1=70ms and delay2=30ms.

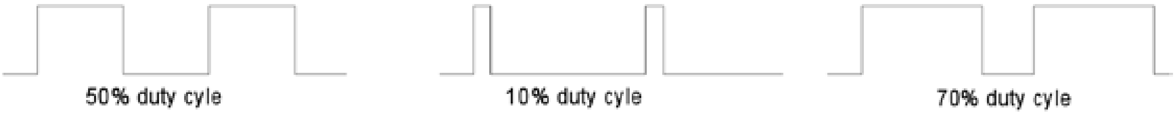


Figure 5.3

From the timing diagram above, it is clear that the square pulse width varies from 70ms to 10ms according to the desired duty cycle, while the waveform period is maintained constant at 100ms. This type of pulse width alteration is given a technical name Pulse Width Modulation "PWM". By changing pulse width we could change the average voltage supplied to the DC motor. The wider the pulse width, the higher the average voltage supplied to the DC motor. Therefore by varying the input voltage to the DC motor we could vary the DC motor speed.

The DC motor driver is an integrated monolithic circuit that is designed to accept a PWM controlling signal from the MCU to drive the DC motor at certain speed. It is high-voltage, high-current motor drive and come in half or full bridge. The full-bridge will drive and control the speed of the DC motor in both directions forward and reverse. The commercial number of the full-bridge is L298.

Assume PIC18F4431 runs at 8 MHz, the program below will generate a 100Khz square pulse 50%, 10%, and 70% duty cycle on PORTD bit 1 (pin#20).

Table (3) shows the concept of pulse wave generation.

```
        BCF       TRISA, 1    ; Set RD1 to Output
LOOP:   MOVLW     1           ; Turn on the LED on Port D
        MOVWF     PORTD
        MOVLW     5           ; 5 refer to 5us delay routine
        CALL      DELAY1      ; 5us/1us/or 7us delay 1 for 50%, 10% or 70% duty cycle
        MOVLW     0           ; Turn off the LED on Port D
        MOVWF     PORTA
        MOVLW     5           ; 5us delay routine
        CALL      DELAY2      ; 5us/9us/or 3us delay2 for 50%, 10% or 70% duty cycle
        GOTO      LOOP
```

CAPTURE, COMPARE, AND PWM (CCP) MODULES

PIC18 microcontroller's family includes one or more CCP modules. The CCP modules are 16-bit (or two 8-bit) registers that are specially designed to perform three function in conjunction with times.

CAPTURE MODE

In this mode, the associated CCP pin is set as an input to record the arrival time of either a rising or falling pulse. When an edge of pulse is sensed, the CCP module records the timer value and sets a flag or generates an interrupt.

COMPARE MODE

In this mode, the associated CCP pin is set as an output and a count is loaded in the CCP register. This count is compared with the timer register at every clock cycle, and when a match is found, the CCP pin can be driven low, high, or toggled. This mode is commonly used to generate pulses or periodic waveforms.

PWM MODE

In this mode, the associated CCP pin must be set as an output; also the necessary count for a PWM period and a duty cycle must be loaded in registers. The CCP module includes two registers: one specify the period of the waveform and the other to specify the duty cycle. The CCP1 pin produces up to 10-bit resolution PWM output, which means you have 1024 different steps from zero to full control range. The PWM mode is used in applications, such as controlling average power delivered to the load, in controlling the speed of the DC motor or controlling servo motor rotation's direction.

The heart of PIC18F4431 PWM lays on the TIMER2 module. This timer is used as the PWM time base for the PWM mode of the CCP module. The TMR@ register is readable and writable, and is cleared on any device Reset. As show in the simplified diagram below, the input clock (Fosc/4) has a pre-scale option of 1,4,16. For details please refer to the datasheet.

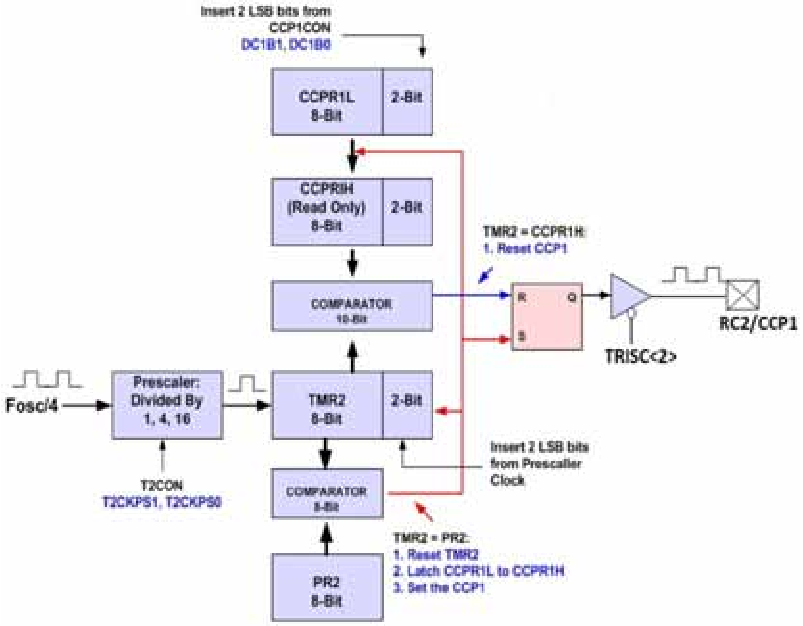


Figure 5.4: Simplified PWM block diagram

The TMR2 counter register clock is supplied by the pre-scale circuit which can be selected using the T2CKPS1 and T2CKPS0 bits in the T2CON register, the TMR2 register value is compared to the PR2 register which determine the TOP value of TMR2 counter register. When the TMR2 value is equal to the PR2 value, then the TMR2 counter register will be reset to 0, set the CCP1 output pin to high level (logic "1"), and copy CCPR1L to CCPR1H. See figure (5.4).

At the same time the value of TMR2 counter register is compared to the CCPR1L register value (CCPR1L= CCPR1H register value), when the TMR2 value equals the CCPR1L register value, the comparator will reset the CCP1 output pin to low level (logical "0") and when the TMR2 counter register equals to the PR2 register value, it will set the CCP1 output pin to high level (logical "1"). Therefore the PR2 register determines the PWM period, and the value of the CCPR1L determines the PWM pulse width. The change in the PR2 value will change the PWM period or PWM frequency and the change in the CCPR1L value will change the PWM pulse width. See the figure below.

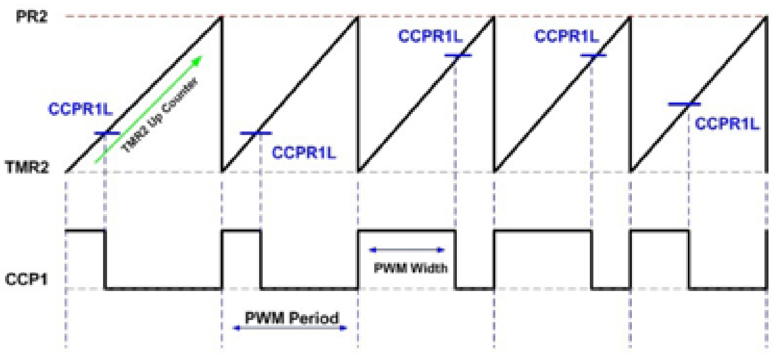


Figure 5.5: PWM output

SETUP FOR PWM OPERATION

1. Set the PWM period by writing to the PR2 register. The PWM could be calculated using this following formula:

   PWM period=[(PR2+1) * 4 * TOSC * (TMR2 prescale value)]

   Where TOSC is the system clock period in seconds.
   PWM frequency= FPWM = 1 / PWM Period Hz

2. Set the PWM duty cycle by writing to the CCPR1L register and CCP1CON<5:4> bits.

   PWM duty cycle = (CCPR1L:CCP1CON < 5:4 >) * Tosc * (TMR2 prescale values)

   The following example will show how to determine the 10-bit binary of (CCPR1L:CCP1CON) register. Assume PIC18 runs at 8MHz a 50Hz wanted at 5% duty cycle. TMR2 pre-scale is 16 find the 10-bit binary.

   PWM period=150=0.02s, PWM duty cycle = 5% of 0.02s =0.001s, TMR2 prescale = 16
   (CCPR1L:CCP1CON < 5:4 >) = PWM duty cycle/(Tosc * (TMR2 prescale values))
   (CCPR1L:CCP1CON < 5:4 >) = 0.001/ (1.25x10(superscript -7) * 16)
   (CCPR1L:CCP1CON < 5:4 >) = 500
   500 in 10-bit binary = 0111110100
   CCPR1L = 01111101 and CCP1CON<5:4> = 00

3. Make the CCP1 pin an output by clearing the TRISC<2> bit.

4. Set TMR@ pre-scale value and enable Timer2 by writing to T2CON.

5. Configure the CCp1 module for PWM operation.

The assembly script for the setup PWM procedure is given below. We assume PIC18F4431 runs at 8 MHz; will generate a PWM output on RC2/CCP1 pin (#17 on the PIC18 device). Select 1:16 pre-scale for TMR2, and load the PR2 register with 250 value (maximum value is 255 for 8-bit). Plug all the values in the PWM period equation in step 1 to obtain the PWM period of 2ms or PWM frequency of 500Hz.

Notice that this is the maximum PWM period (or minimum PWM frequency) we can obtain using TMR2. Also for duty cycle of 50% the 10-bit binary will be CCPR1L=01111101, and CCP1CON=00

```
MOVLW     d’250’             ; when TMR2 = PR2 = end of the PWM period
MOVWF     PR2
MOVLW     b’01111101         ; Load CCPR1 with the duty cycle binary value
MOVWF     CCPR1L
BCF       CCP1CON, DC1B1     ; Set the two LSbs (bit 1 and bit 0) of the 10-bit
BCF       CCP1CON, DC1B0     ; PWM duty cycle
MOVLW     b’11111011’        ; Make RC2/CCP1 pin an output*
ANDWF     TRISC
MOVLW     b’00000111’        ; set TMR2 ON and select the per-scale 1:16
MOVWF     T2CON
MOVF      CCP1CON,W          ; configure the CCP1 module for PWM
ANDLW     b’00110000’        ; Mask all but the previously set Duty cycle bits
IORLW     b’00001111’        ; and enable PWM mode
MOVLW     CCP1CON
```

* CCP2CON register can be configured to make RC1/CCP2 pin an additional PWM output

The PWM hardware has up to 10-bit resolution, which means that you can have up to 1024 different steps or voltage levels from zero to the full range. The following formula is used to determine the maximum PWM resolution (MAX bits = 10 bits).

PWM Resolutionmax=log(Fosc/Fpwm)/log(2) bits

The maximum PWM resolution for FOSC = 8MHz, and PWM frequency = 500Hz is 10 bits.

DRIVING SERVO MOTOR

The above example shows that TMR2 can be used to generate PWM with minimum PWM frequency of 500Hz (because PR2 is set to near max value of 255). This frequency is still too high from the servo motor working frequency of 50Hz. To obtain lower PWM frequency choose one of the following approaches:

1. Create your own PWM function to mimic the PWM signal as follow: turn on (set to logic 1) the PORT, make some 2 ms delay, turn off (set to logic 0) the PORT, and make some 18 ms delay and so forth. Although this is an easy approach but is not the efficient way to do it.
2. Keep using the PIC PWM peripheral and lower the operation frequency by setting the OSCCON register and PR2 register until it meets the servo motor frequency requirement. This approach will scarify the program execution speed as we will operate the PIC Microcontroller with the 500 kHz clock speed.

3. Use the PIC18 microcontroller's TIMER0 (have wider per-scale 1:256) with the interrupt to generate timing interval of 20ms. Each timer0 interrupt calls a pulse generation function that sets the position of the servo (1ms-2ms).
4. Use the PIC microcontroller's TIMER0 and TIMER1 to generate repetitive interrupts.

The idea in step 3 and 4 is to set TIMER0 to generate a continuously running series of accurately timed interrupt. Such series of interrupts on overflow serves as base PWM pulse generator or a clock tick for many time-based operations. Let us consider first the approach in step3.

USING TIMER0 AND VARIABLE COUNTERS TO GENERATE PWM

Assume PIC18F4431 runs at 8 MHz, TIMER0 period could be calculated using this formula bellow:

$$\text{TIMER0 period} = [(\text{TMR0} + 1)] \times 4 \times \text{Tosc} \times (\text{TIMER0 Prescaler value}) \text{ second}$$

By selecting the TIMER0 prescaler of 2 (T0PS2=0, T0PS1=0 and T0PS0=0 bits in T0CON) and initial the TMR0 register value to 156 (99 more counts to reach its maximum value of 255) with the system frequency clock of 8 Mhz, the PIC microcontroller TIMER0 overflow period can be calculated as follow:

$$\text{TIMER0 period} = [((255 - 156) + 1)] \times 4 \times 1/8000000 \times 2 = 0.0001 \text{ second} = 0.1 \text{ ms}$$

The idea is to use this clock tick of 0.1ms to generate the 20ms time period that is required by the servo motor as periodic update period of 50 Hz frequency. This can be done by counting these ticks up to maximum 200, this will give us the constant 20ms or 50Hz frequency. To generate the required pulse, use additional count variables and increment or decrement every clock tick, and then compare the updated value with a threshold. When a match occurs, the RC2 pin is driven high or high. This approach is similar to the operation of TIMER2 in the PWM mode shown in figure (5.6), when CCPR1L is constantly compared against the TMR2 (8-bit) value. When a mach occurs, the RC2/CCP1 pin is driven low.

In order to generate the PWM pulse, we need to have the following variable counter:

1. The INTcount variable counter is set to increment (or decrement if initial value is 200) each time an overflow TMR0 interrupt occurs (every 0.1ms). It will hold this count and compare to the MAX "= 200" (or compare zero). When a match occurs it will reset the RC2 port (logic "0") which connected to the servo motor.
2. The SerUpdate variable counter will be used to generate 1sec delay before sending a new pulse width to the servo. Due to the servo motor characteristics such as the servo transit time, the pulse width must be maintained for about 50 consecutive cycles. Total delay =50 x 20ms=1sec.
3. The ONtime variable counter will be used to determine the PWM pulse width.
4. The PWM-pulse variable counter will be compared to the ONtime variable, if it equal then we will set the RC2 port (logic "1"). See figure (4) and the assembly program below.

Typical values for ONtime are:

MAX-10 = 190	1ms pulse → CW-Rotation
MAX-20 =180	2ms pulse → CCW-Rotation
MAX-15 =175	1.5ms pulse → neutral position
MAX = 200	Stop the servo motor

Check the datasheet of the servo motor on hand for the correct pulse duration versus the angular position. For example a servo motor requires that the pulse stream of periodic period of 20ms (50Hz). The input pulse width of 1.25ms leads to an output shaft position of 0°, 1.5ms to an output shaft position of 90° and 1.75ms to an output shaft position of 180°. If the transit time (the amount of time it takes for the servo to move 60° is 0.17sec/60° at no load. This means that it will take about 0.5sec to rotate 180°. The following program gives an idea about how to put these data into a workable assembly program. The TMR0 initialization and ISR (interrupt service routine) are given with side explanations, and some part of the program will be omitted.

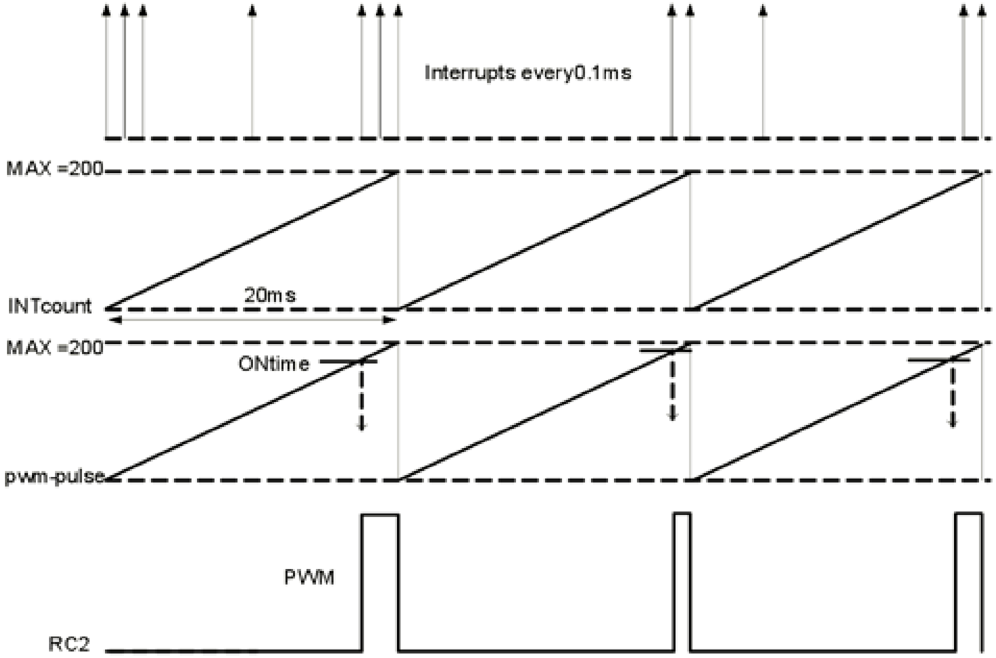


Figure 5.6

Table 4: ISR for Servo motor position controller

	ORG	0x0008	; high-priority interrupt vector
	GOTO	ISR	
....			
MAIN:	MOVLW	0x72	; Set the frequency to BMHz
	MOVWF	OSCCON	
	CLRF	INTcount	; Clear all incremental counters
	CLRF	pwm-pulse	;
	CLRF	SerUpdate	;
	BSF	RCON, IPEN	; Enable priority
	MOVLW	B'11100000'	; Set global and peripheral priority, enable TIMER0
	MOVWF	INTCON,1	; overflow interrupt bit
	BSF	INTCON2, TMR0IP	; Set Timer0 as high-priority
	MOVLW	B'11000000'	; Enable Timer0 & 8-bit mode
	MOVWF	T0CON	; pre-scale value 1:2
	MOVLW	d'156'	: Value to be loaded in TMR0L buffer
	MOVWF	TMR0L	; 156 in TMR0L buffer for 0.1ms interrupt
	BCF	INTCON, TMR0IF	; Clear TMR0 overflow flag and start count
ISR	INCF	INTcount, 1	; Increment the two counters every 0.1ms
	INCF	pwm-pulse, 1	
	MOVLW	D'200'	
	CPFSEQ	INTcount	; Compare and check if the peiod is 20ms
	GOTO	Test1	
	BCF	PORTC, 2	; Rest the RC2 pin of PORTC to logic 0 at the
	BCF	INTCON, TMR0IF	; beginning of the period
	CLRF	INTcount	
	INCF	SerUpdate	
	GOTO	INTend	
Test1	MOVF	ONtime, 0	
	CPFSEQ	pwm-pulse	; Compare check with the ONtime threshold
	GOTO	Test2	
	BSF	PORTC, 2	; Set the RC2 pin of PORTC to logic 1 and
	BCF	INTCON, TMR0IF	; keep it to the end of the period
	GOTO	INTend	
Test2	MOVLW	D'50'	
	CPFSEQ	SerUpdate	; Compare to see if we can update the width
	GOTO	Test3	; of applied pulses or not
	INCF	Pointer, 1	
TEST3	MOVF	Pointer, 0	; Move to the next width (ONtime) in the POSITable.
	ANDLW	0x07	
	CALL	POSITable	
	MOVWF	ONtime	

```
INTend      BCF       INTOCN, TMR0IF     ; Clear TMR0 overflow flag
            MOVWF     TMR0L
            BCF       PORTC, 2           ; Set RC2=0
            RETFIE    FAST               ; go back to the program
POSITable   ADDWF     PCL                ; Set the values to position the servo motor shaft:
            RETLW     D' '               ; 0 degrees
            RETLW     D' '               ; 45 degrees
            RETLW     D' '               ; 90 degrees
            RETLW     D' '               ; 135 degrees
            RETLW     D' '               ; 180 degrees "neutral position"
            RETLW     D' '               ; 135 degrees
            RETLW     D' '               ; 90 degrees
            RETLW     D' '               ; 45 degrees
            END
```

USING TWO TIMERS TIMER0 AND TIMER1 TO GENERATE PWM

Configure timer0 to interrupt at exactly 20ms. When Timer0 initiates a high priority interrupt, it will enable Timer1 (low priority) to count up to low priority interrupt. The position control pulses for the servo will be generated as follows: when Timer0 overflows, the servo pin is set high (logic "1") and at the same moment, timer 0 initiate timer1 count. When Timer1 overflows, the servo pin is set low (logic"0"). Figure (5.7) shows the wave diagram for the output pulse generated to control the servo motor.

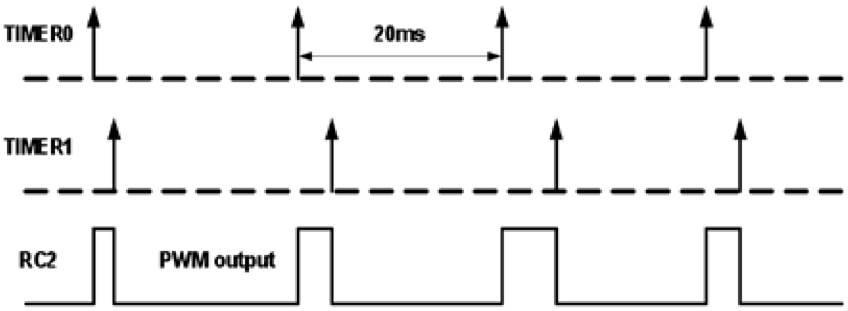


Figure 5.7

In this approach, you can load time1 with the demanded pulse width and continue sending the pulses for 1sec, then search for another one. If a change in position is required, get the variable and loaded into Timer1. As for timer0, it will run continuously to provide a trigger every 20ms for timer1. The fact that Timer0 interrupt is slow means that the microcontroller has plenty of time to execute other code in the program. The ISR (interrupt service routine) code should be kept short, efficient and avoid implant software delay in it so that, the lower priority tasks receive enough processing time. Also DO NOT forget to put the instruction "BCF INT0IF" in the first line of IRS so that the subsequent interrupts can be recognized. This action usually needed when you have a multiple interrupt situation.

Assume PIC18F4431 runs at 8 MHz, TIMER0 period could be calculated using this formula bellow:

TIMER0 period = [(TMR1 + 1)] x 4 x Tosc x (TIMER0 Prescaler value) second

By selecting the TIMER1 prescaler of 256 (T0PS2=1, T0PS1=1 and T0PS0=1 bits in T0CON) and initial the TMR0 register value to 99 (156 more counts to reach its maximum value of 255) with the system frequency clock of 8 Mhz, the PIC microcontroller TIMER0 overflow period can be calculated as follow:

TIMER0 period = [((255 - 99) + 1)] x 4 x 1/8000000 x 256 = 0.0001 second = 20.096ms

Timer1 will be configured to interrupt-on-overflow from FFFFh to 00000h using TMR1 register pair (TMR1H:TMR1L). TMR1 interrupt can be enable/disable when TMR0 interrupt by setting/clearing Timer1 interrupt enable bit, TMR1E (PIE1<0>). Also we set the prescaler of 8 (T1CKPS1=1 and T1CKPS0=1 bits in T1CON). To calculate the initial value to load into the register do the following:

1. Find the numerical value TMR1 in the above formula that generate 1ms/2ms

If 4 x 1/8000000 x 8= 4us, then 1ms/4us = 250 = (TMR1+1), TMR1=249, this value will generate exact 1ms, however, choosing the value of TMR1=255, will make no difference in time (about 0.024ms) but it will save us few instructions in ISR.

2. Determine the initial values to load into each register FFFF-d'255'= FFFF-FF = FF00, therefore the preload for TMR1H is FFh and the preload of TMR1L is 00. The later can be set using one instruction CLRF TMR1L.

Similarly for 2ms TMR1H=FE and TMR1L=00, the exact time is 2.048ms. Try your best to optimize your program and eliminate any unnecessary instructions.

EXPERIMENT

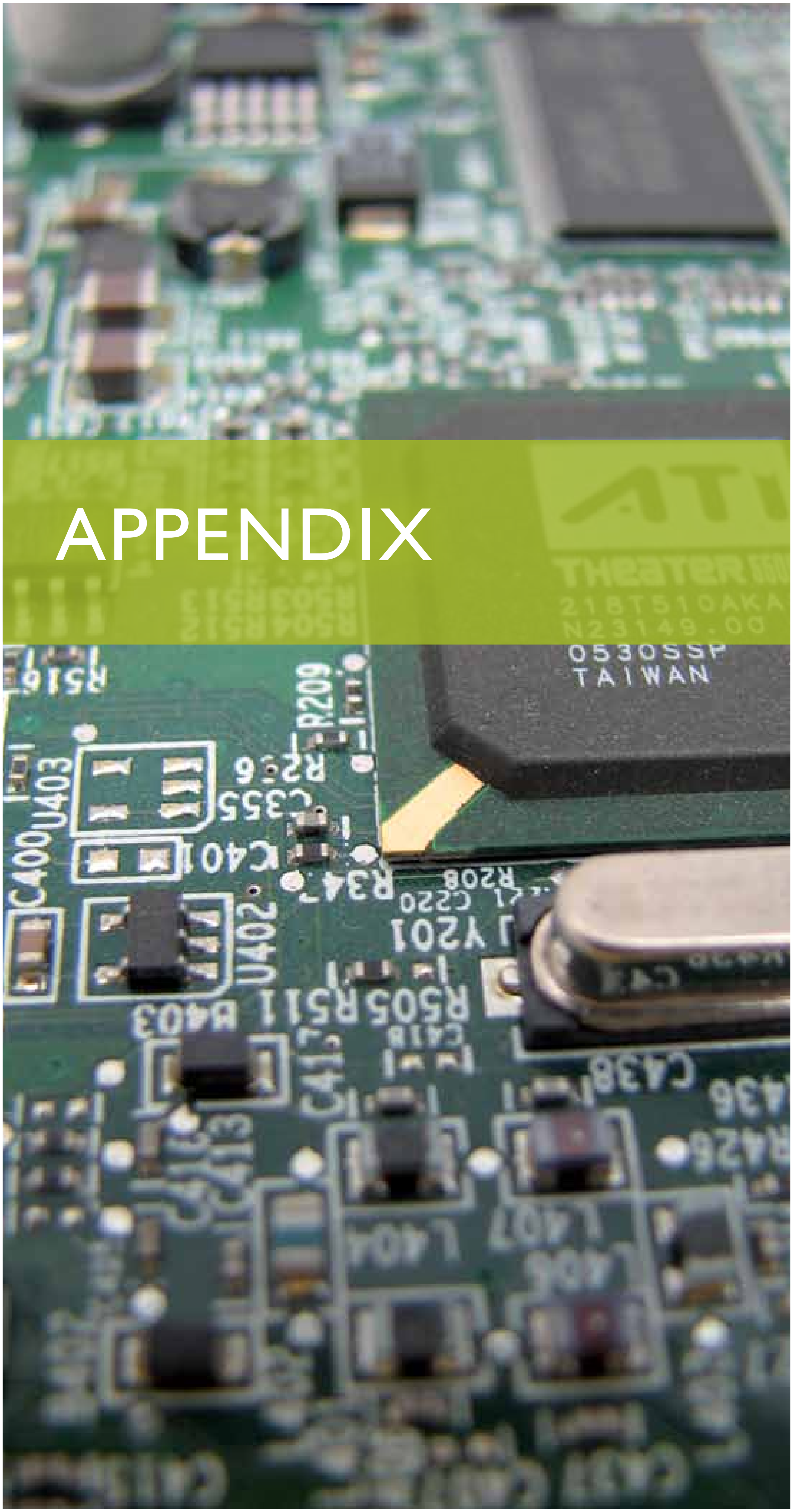
PART 1: SERVO MOTOR CONTROLLER

1. Given: the internal oscillator of the PIC MCU is 8KHz, and the output pin is RC1.
2. Use a suitable approach to write a program for controlling angular position of the hoppy servo motor. The angular position must be started from neutral position, move to the left, and back to the neutral, and then move to the right, and back to the neutral. The motion in this case is similar to the antenna's motion of the 180°(90°) navigational radar system.
3. Select a suitable delay to make the motion smooth and continues.
4. Display and examine the output on pin RC1 using MPLAB logic analyzer.
5. Select the scan angle based on the available servo motor type and characteristics.
6. Include a copy of your program and the waveform in the lab report and explain each step.
7. Discuss possible application to this type of motion. For example one might suggest that Ultra-Sound sensor can be mounted on the top of the motor to scan the area in front of the moving RCV for any obstacles.

PART 2: SQUARE PULSE GENERATOR & DC MOTOR SPEED CONTROL *(optional)*

1. Write a program using Timer 0 to generate a 1 KHz with 10%, 20%, and 40% duty cycle square pulse. Set the internal oscillator of the PIC18 MCU to 8MHz, and use the PWM hardware features of the PIC.
2. Display and examine the frequency using the MPLAB logic Analyzer. Display the output in a sequence of pulses with a 10% duty cycle for 1sec, then change to a 20% for 1sec, followed by a 40% for 1sec, and back to a 10% duty cycle.
3. Select a suitable delay between pulses to have s smooth transition from 10% to 40%.
4. Include a copy of your program in the lab report with an explanation to each step.
5. Disable two of the duty cycle values and connect the output pin to the DC motor control board shown in figure () below.
6. Observe the motor speed.
7. Repeat step 6 for another duty cycle, compare the result and comment.

APPENDIX



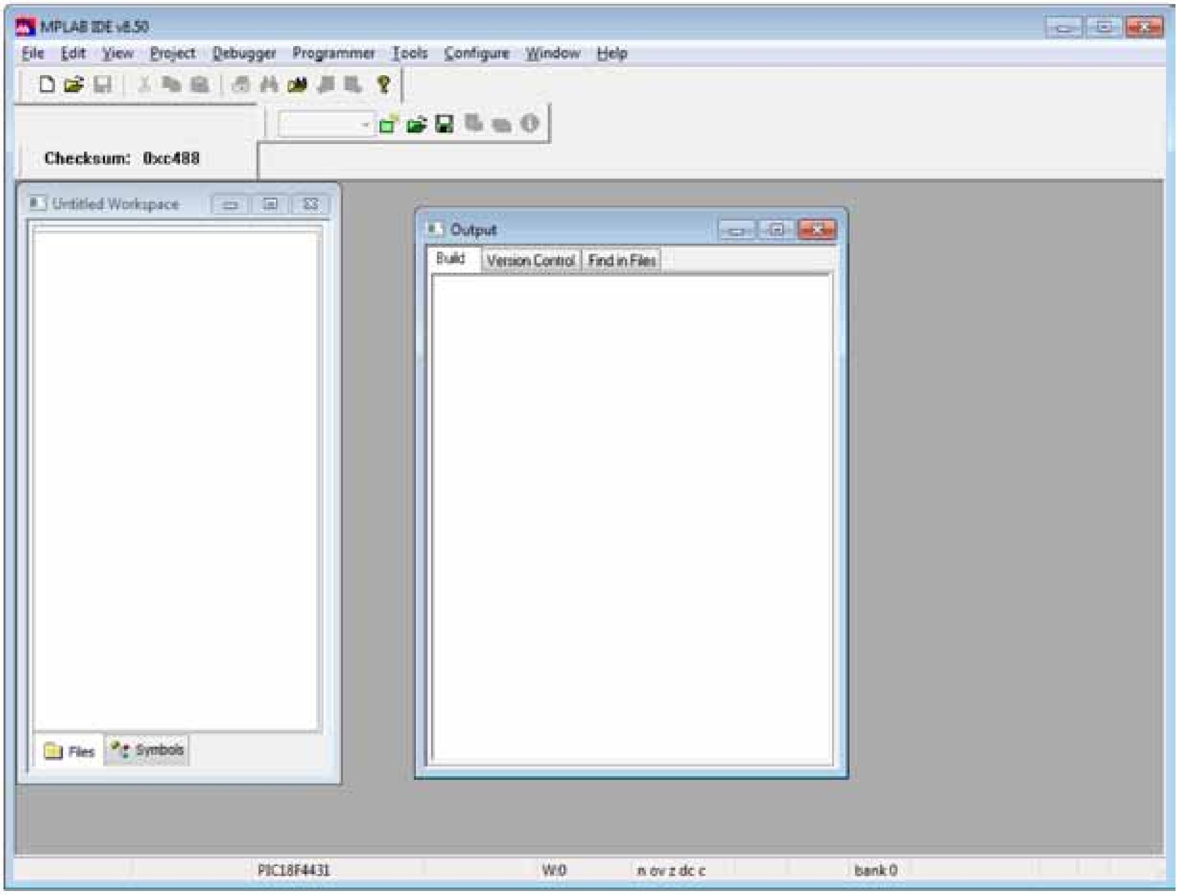
PIC PROJECT IDEAS

1. A model train controller or digital event sequencer for traffic light or toys.
2. A sequencer to switch different light patterns for a light show.
3. The switching of the water valves is in synchronization with music.
4. A sequencer to switch an explosive in a controlled demolition system.
5. A controller for an elevator of a building.
6. A controller of a light and ventilation systems after working hours of a building.
7. An automatic security gate controller equipped with two way of audio communications and video surveillance cameras or par code identification to open and close gates.
8. Temperature, humidity, and shelf shaker controller of an egg hatcher in a chicken farm.
9. A burglar alarm can be developed that reads the status of sensors and activate an alarm and signalling police station.
10. A fire alarm can be designed to locate the hot zone in the high rise building to cut the electric power, start fire distinguish system and signalling the fire department.
11. A sequencer to switch lights and audio systems on or off in a random sequence at random times to give the impression that a home is occupied.
12. A solar tracking system to position a concave (or flat) mirror arrays perpendicular to the sun rays during the day and in all seasons. This can be used to heat water or to generate electricity using sun energy.
13. A speed control of small DC motor and angular position controller for remotely controlled RCV equipped with IR and ultrasound sensors.
14. A programmable battery management and power control systems in electric cars.
15. A digitally controlled bed, dentist seat, or operating room bench in hospitals.
16. A controller for battery-powered wheelchair, this includes, battery and power management systems and speed controller.

GETTING STARTED WITH MPLAB IDE

MPLAB IDE is a windows-based software program that runs on a PC to develop applications for Microchip microcontrollers and digital signal controllers. It is called an Integrated Development Environment, or IDE. Download and install on your computer the latest version from microchip site www.microchip.com. The MPLAB IDE provides all what the developer needs from writing the program, debug and test before the final application takes shape. This development cycle is essential and time consuming process for designing imbedded applications. Using MPLAB IDE helps to develop flawless code then converted into machine code to be burned into a device. The MPLAB has a project manager that organize the files to be edited and other associated files so they can be sent to the language tools for assembly or compilation, and ultimately to a linker.

To run the MPLAB double click on the icon, the window shown below will open. It consists of 3 windows: the main MPLAB window with the program version and top menu, the untitled workspace and output windows.



Checksum: 0xc488

PIC18F4431 W:0 n ov z dc c bank 0

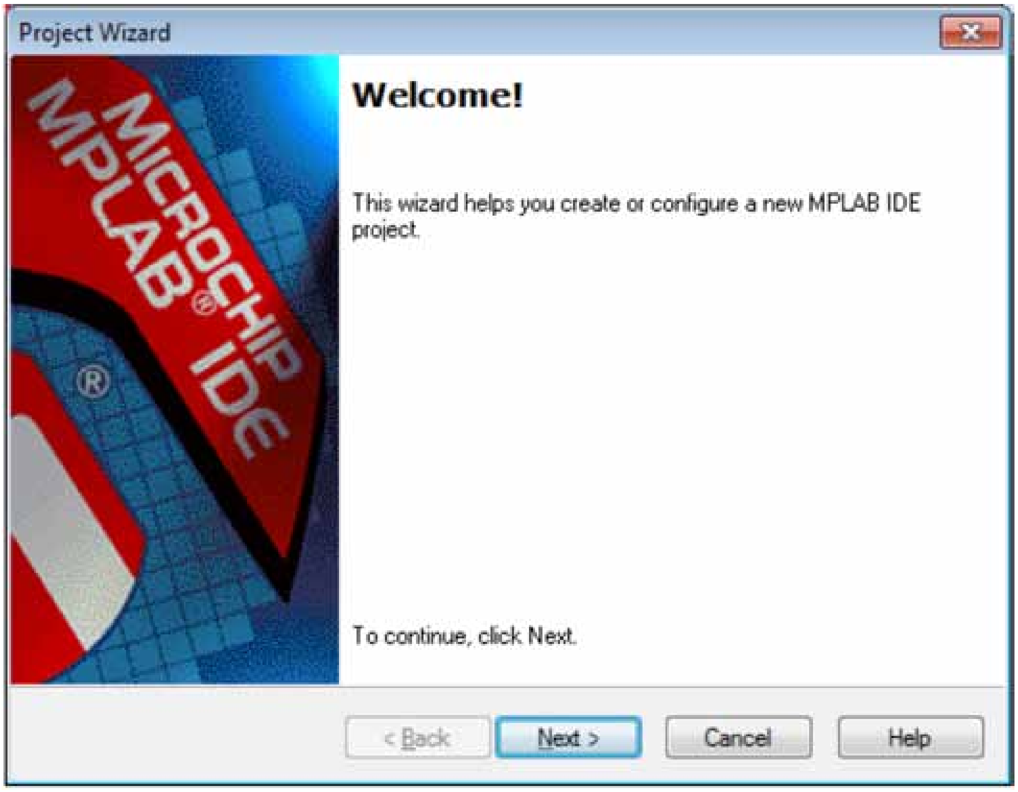
To use the MPLAB IDE follow these steps:

1. Create a new project: Folder contains all files
2. Add files: Add source code or files to the project.
3. Build the project: Assemble or compile (c code) and generate all necessary files Hex and object code file (or machine code file).
4. Run and debug: Execute the program using simulator MPLAB SIM and debug.

Before you proceed, create a directory on the hard disc of your computer and name it C:\"Your initials PROJET" LAB for example the full path to my folder will look like C:\Users\umroot\Desktop\BMILAB. Follow the steps below:

CREATING NEW PROJECT

From the top menu click on PROJECT → PROJECT WIZARD. The welcome screen is displayed.

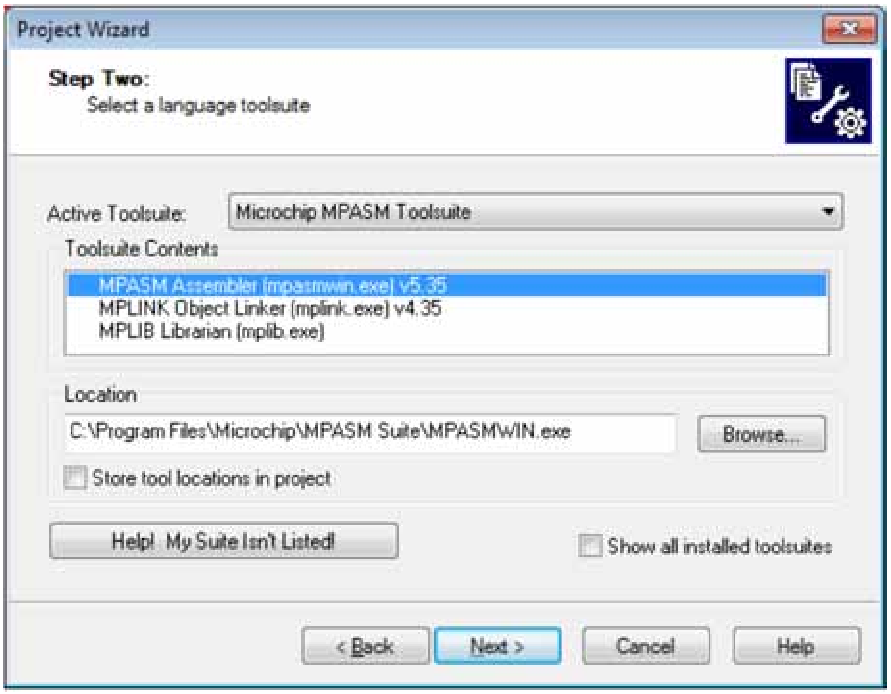


Welcome!

This wizard helps you create or configure a new MPLAB IDE project.

To continue, click Next.

Click next to proceed, a Device selection window will popup, select PIC18F4431 then click Next. The Project Wizard window or Step Two window is displayed to select a Language Toolsuite. Notice that C compiler is neither free nor included in this software. If you try to select for example C compiler,   every item in the Toolsuite Contents will be proceeded with red X in the front. This means it is not available or not working. Step two figures shown below.



Project Wizard

Step Two:
Select a language toolsuite

Active Toolsuite: Microchip MPASM Toolsuite

Toolsuite Contents

- MPASM Assembler (mpasmwin.exe) v5.35
- MPLINK Object Linker (mplink.exe) v4.35
- MPLIB Librarian (mplib.exe)

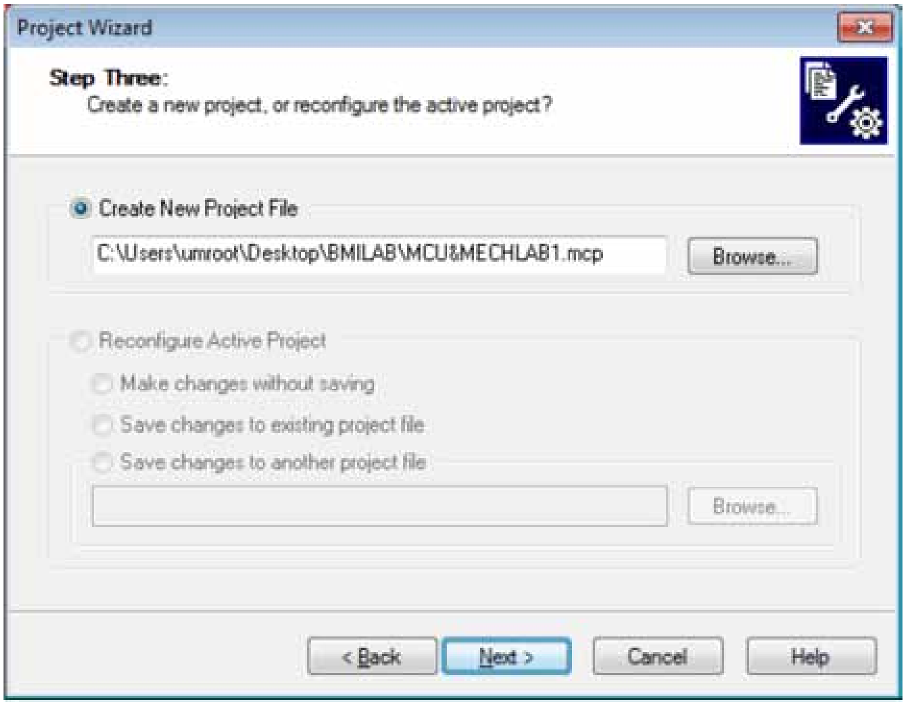
Location

C:\Program Files\Microchip\MPASM Suite\MPASMWIN.exe

□ Store tool locations in project

□ Show all installed toolsuites

Click on next for step three window Name your project. Browse C:\ to locate the project folder (in my case BMILAB) and type MCU&MECH LAB1 with no extension, then on click next.



Project Wizard

Step Three:
Create a new project, or reconfigure the active project?

● Create New Project File

C:\Users\umroot\Desktop\BMILAB\MCU&MECHLAB1.mcp

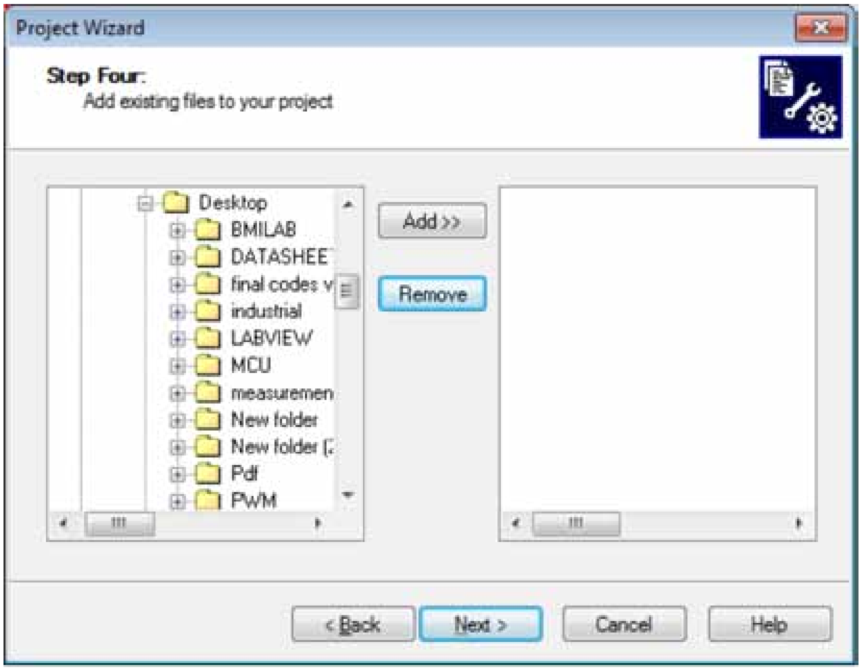
○ Reconfigure Active Project

○ Make changes without saving

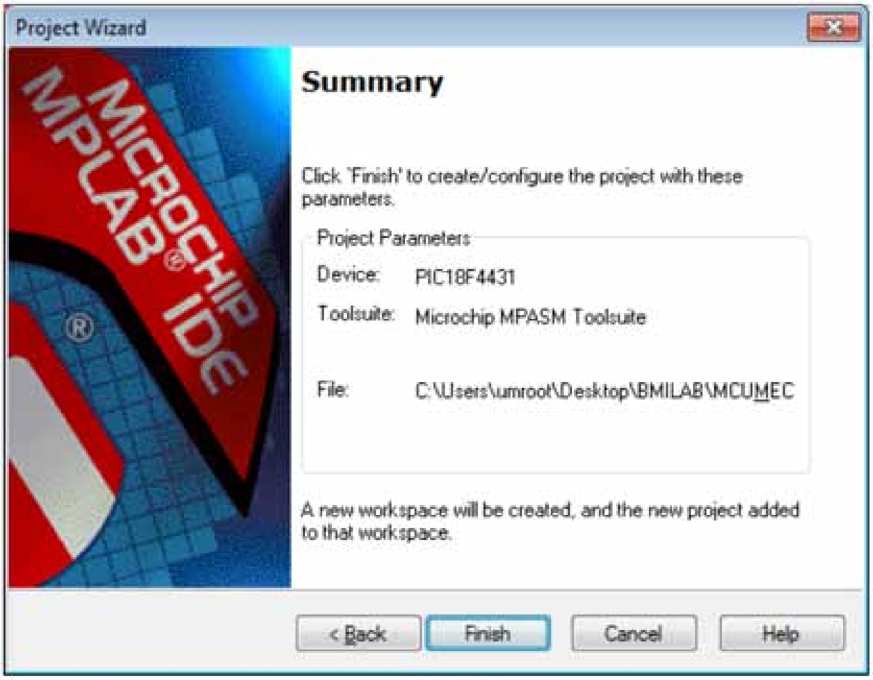
○ Save changes to existing project file

○ Save changes to another project file

Click next to go to Step Four "Add existing files to your project", if you don't have click Next.



The Project Wizard Summary will popup all information about the project you have been creating.



Summary

Click 'Finish' to create/configure the project with these parameters.

Project Parameters

Device: PIC18F4431

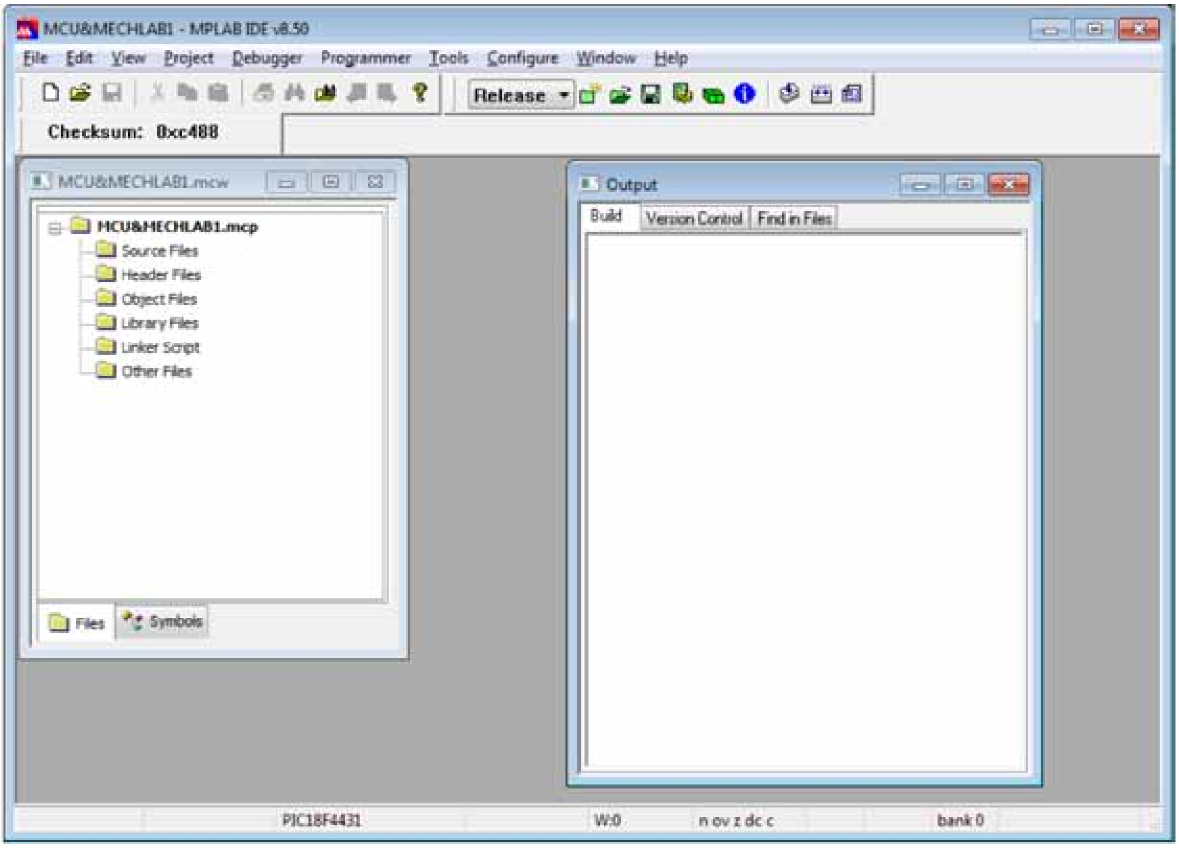
Toolsuite: Microchip MPASM Toolsuite

File: C:\Users\umroot\Desktop\BMILAB\MCUMEC

A new workspace will be created, and the new project added to that workspace.

Click finish and notice the change in the MPLAB main window. Now the Untitled Workspace is changed to MCU&MECHLAB1.mcw with the extension mcw (workspace). Also notice that the root directory of project is given the same name but with an extension mcp (P for project). As you proceed with your work you have many files added to the folders below the project folder.

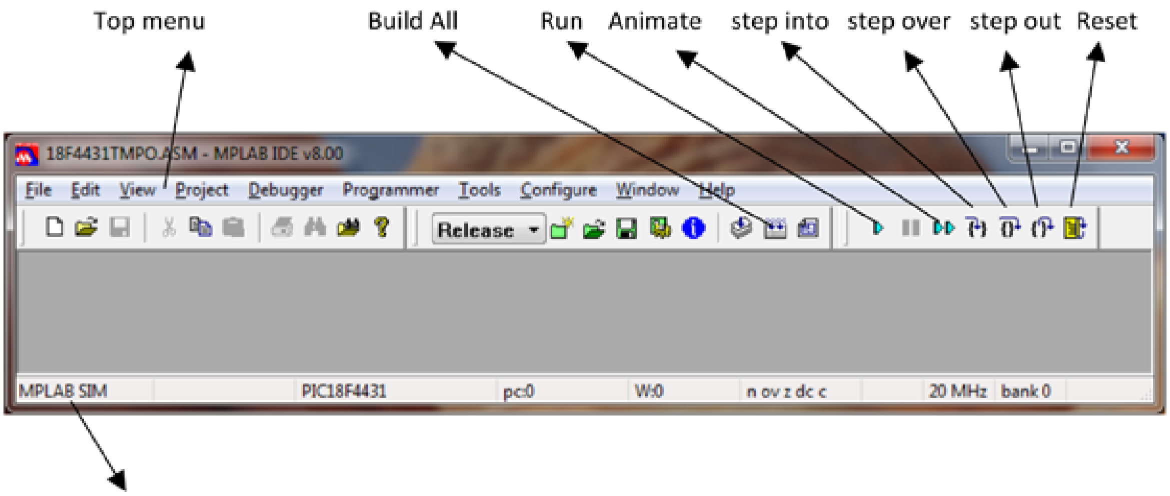
After editing, saving, assembling, and run the program, the MPLAB will add the file under the prospective folder. For example under Source Files folder a file with an extension .asm will appear right after editing and saving the file. Also after building a file with an extension .cod (object) will be added to the folder Object files.



TOP MENU AND CONTROL BUTTONS

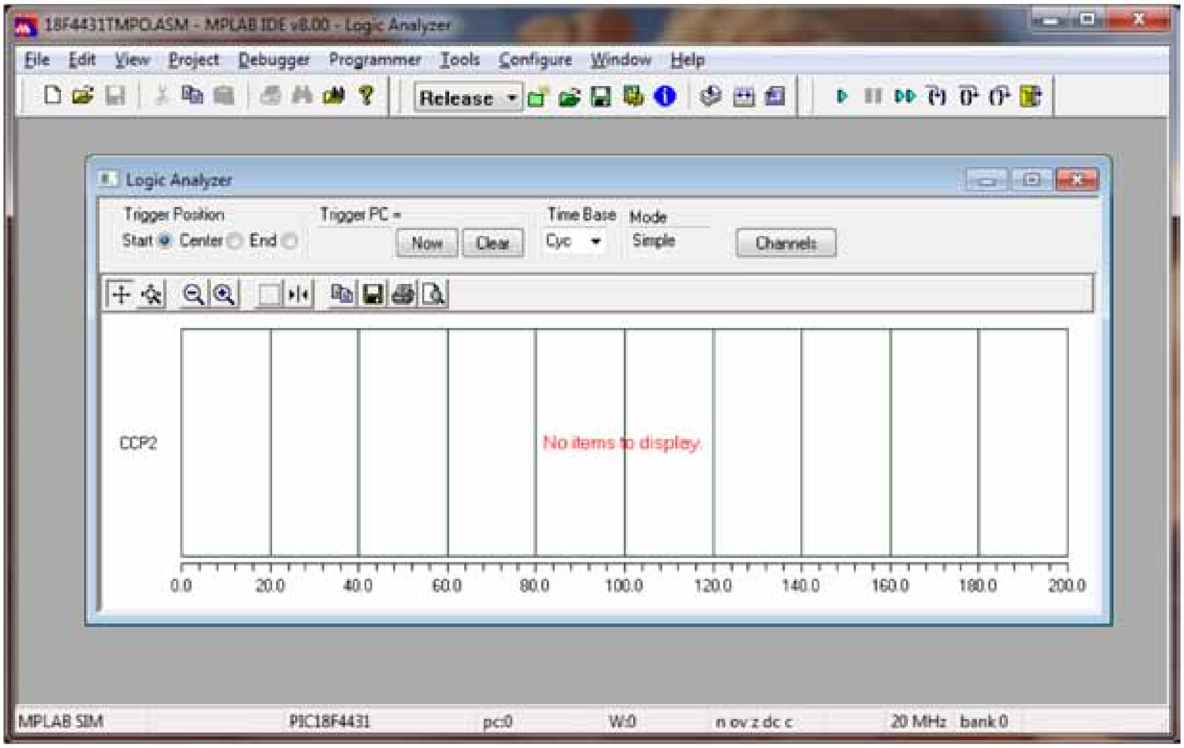
The menu of the MPLAB IDE is self-explanatory. If you forget the function of any toolbar button, position the mouse cursor over a toolbar button to see a brief description of the button's function. To explore the top menu, click view, an additional menu items will pop up, the toolbar item will allow you to add or remove set of control buttons. You select a watch window or logic analyzer and more from view menu.

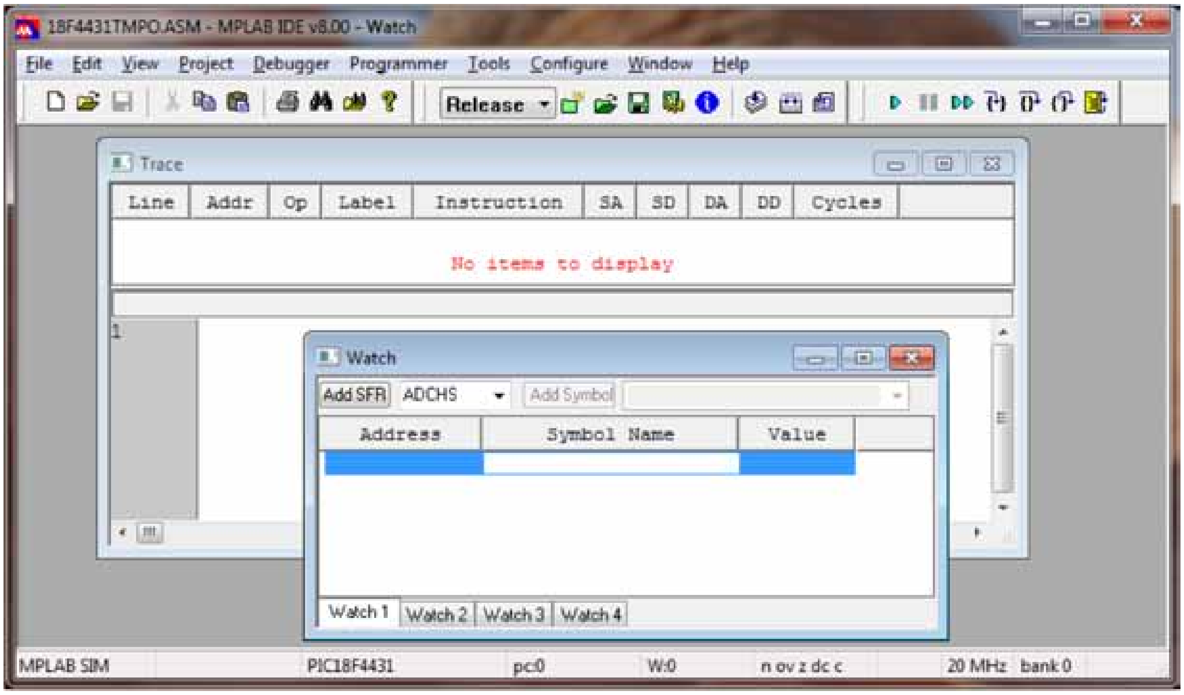
Explore Debugger menu, click Debugger, select tool, from this you can choose to simulate the assembly program and watch all registers and even observe the result on logic analyzer. When you are sure about the program, you can choose MPLAB ICE2000 as a tool to have a real time run for your assembly program.



The status bar on the bottom of the MPLAB IDE window should change to “MPLAB SIM”. Other information about your MPLAB IDE configuration will appear subsequently on other fields. For example device selection “PIC18F4431”, Frequency “20MHz”, register bank selection... etc.

The screen shot of the logic analyzer and watch and trace windows are shown below.




18F4431TMPO.ASM - MPLAB IDE v8.00 - Watch
File Edit View Project Debugger Programmer Tools Configure Window Help
Release
Trace
Watch
Add SFR
ADCHS
Add Symbol
Watch 1
Watch 2
Watch 3
Watch 4

Line	Addr	Op	Label	Instruction	SA	SD	DA	DD	Cycles

No items to display

Address	Symbol Name	Value

MPLAB SIM PIC18F4431 pc:0 W:0 n ov z dc c 20 MHz bank 0

LOAD THE TEMPLATE AND LINKER FILES

You can add a template file to the Source files folder and a linker file to the Linker Script folder. In the Step Four window “Add existing files to your project” scroll up to program files directory.

Go to this directory C:\Program Files\Microchip\MPASM Suite\Template\Code then select 18F4431TEMP.ASM click Add. In The Code directory a readme text file explains the content and the use of the template files is copied below.

Code Directory

This directory contains template files for absolute code development. This means that program memory code and RAM variables will be placed exactly where you specifically define them to be, using directives such as ORG. These directives are interpreted during assembly and place the code following the directive in a specified portion of program memory.

Template files containing absolute code generally use an ORG directive to place code at a reset vector and an ORG directive to place code at each interrupt vector location. The interrupt vector location in program memory precedes the code of the “main” program in program memory.

Object Directory

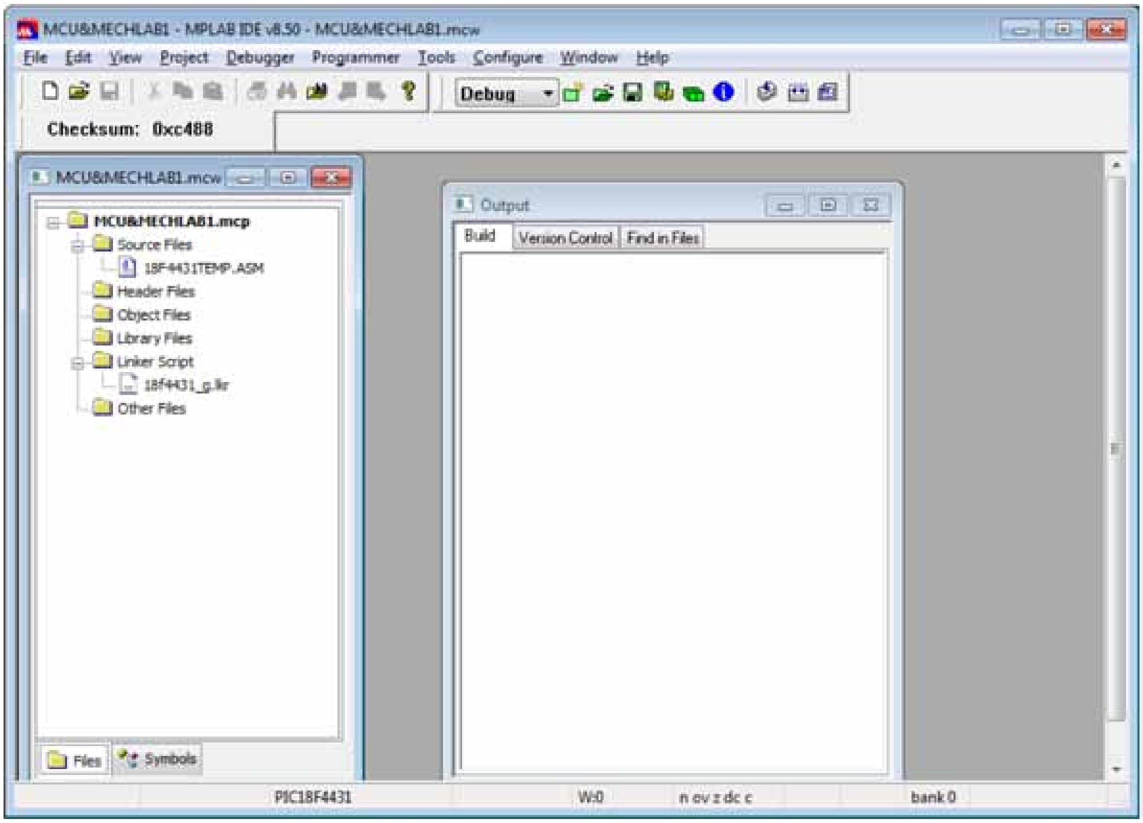
This directory contains template files for relocatable code development. This means that program memory code and RAM variables will be placed by the linker using directives such as CODE. The assembler builds portions of code and then the location of the code is determined by a linker script included in the project being built.

Template files containing relocatable code do not use an ORG directive to place code within program memory. Instead, the CODE directive is used to place code at specific locations in program memory as well as making the linker script decide where the code gets placed within program memory.

Absolute vs. Relocatable Code

For more on absolute and relocatable code, see the MPASM assembler documentation (user's guide or on-line help file) under "Assembler Operation".

The MCU&MECHLAB1 MPLAB project window after adding the Template and the linker files to the project.



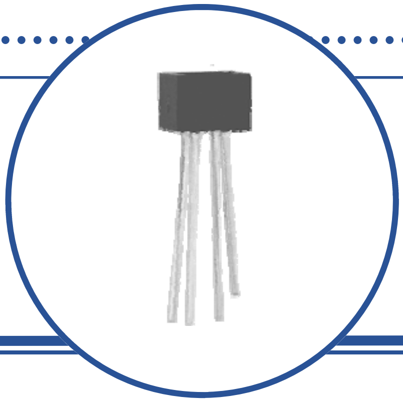
The TEMPLATE file is self explained and given below:

```
;******************************************************************************
;   This file is a basic template for assembly code for a PIC18F4431. Copy     *
;   this file into your project directory and modify or add to it as needed.   *
;                                                                              *
;   Refer to the MPASM User's Guide for additional information on the          *
;   features of the assembler.                                                 *
;                                                                              *
;   Refer to the PIC18F2331/2431/4331/4431 Data Sheet for additional           *
;   information on the architecture and instruction set.                       *
;******************************************************************************
;                                                                              *
;    Filename:                                                                 *
;    Date:                                                                     *
;    File Version:                                                             *
;                                                                              *
;    Author:                                                                   *
;    Company:                                                                  *
;                                                                              *
;*******************************************************************************
;                                                                               *
;    Files Required: P18F4431.INC                                               *
;                                                                               *
;*******************************************************************************
          LIST P=18F4431                   ;directive to define processor
          #include <P18F4431.INC>          ;processor specific variable definitions
;*******************************************************************************
;   Configuration bits
;   Microchip has changed the format for defining the configuration bits, please
;   see the .inc file for further details on notation.  Below are a few examples.

;   Oscillator Selection:
    CONFIG      OSC = LP         ;LP

;*******************************************************************************
;   Variable definitions
;   These variables are only needed if low priority interrupts are used.
;   More variables may be needed to store other special function registers used
;   in the interrupt routines.

                CBLOCK          0x080
                WREG_TEMP       ;variable used for context saving
                STATUS_TEMP     ;variable used for context saving
                BSR_TEMP        ;variable used for context saving
                ENDC
                CBLOCK      0x000
                EXAMPLE                 ;example of a variable in access RAM
                ENDC

;******************************************************************************
;   EEPROM data
;   Data to be programmed into the Data EEPROM is defined here

                ORG     0xf00000
                DE      "Test Data",0,1,2,3,4,5

;******************************************************************************
;   Reset vector
;   This code will start executing when a reset occurs.

                ORG     0x0000
                goto    Main            ;go to start of main code

;******************************************************************************
;   High priority interrupt vector
;   This code will start executing when a high priority interrupt occurs or
;   when any interrupt occurs if interrupt priorities are not enabled.

                ORG     0x0008
                bra     HighInt         ;go to high priority interrupt routine

;******************************************************************************
;   Low priority interrupt vector and routine
;   This code will start executing when a low priority interrupt occurs.
;   This code can be removed if low priority interrupts are not used.

                ORG     0x0018

                movff   STATUS,STATUS_TEMP          ;save STATUS register
                movff   WREG,WREG_TEMP              ;save working register
                movff   BSR,BSR_TEMP                ;save BSR register

;           *** low priority interrupt code goes here ***


                movff   BSR_TEMP,BSR                ;restore BSR register
                movff   WREG_TEMP,WREG              ;restore working register
                movff   STATUS_TEMP,STATUS          ;restore STATUS register
                retfie

;******************************************************************************
```

```
;   High priority interrupt routine
;   The high priority interrupt code is placed here to avoid conflicting with
;   the low priority interrupt vector.

HighInt:

;           *** high priority interrupt code goes here ***

                    retfie     FAST

;*********************************************************************************************************
;   Start of main program
;   The main program code is placed here.

Main:

;           *** main code goes here ***

;*********************************************************************************************************
;   End of program

                    END
```

Reflective Object Sensor

OPB706A, OPB706B, OPB706C
OPB707A, OPB707B, OPB707C



Features:

- Choice of Phototransistor (OPB706) or Photodarlington (OPB707) output
- Unfocused for sensing diffuse surface
- Low cost plastic housing
- Designed for use with PCBoards or connectors

Description:

The **OPB706** consists of an infrared Light Emitting Diode (LED) and an NPN silicon Phototransistor mounted "side-by-side" on parallel axes in a black plastic housing. The **OPB707** consists of an infrared LED and an NPN silicon Photodarlington mounted "side-by-side" on parallel axes in a black plastic housing.

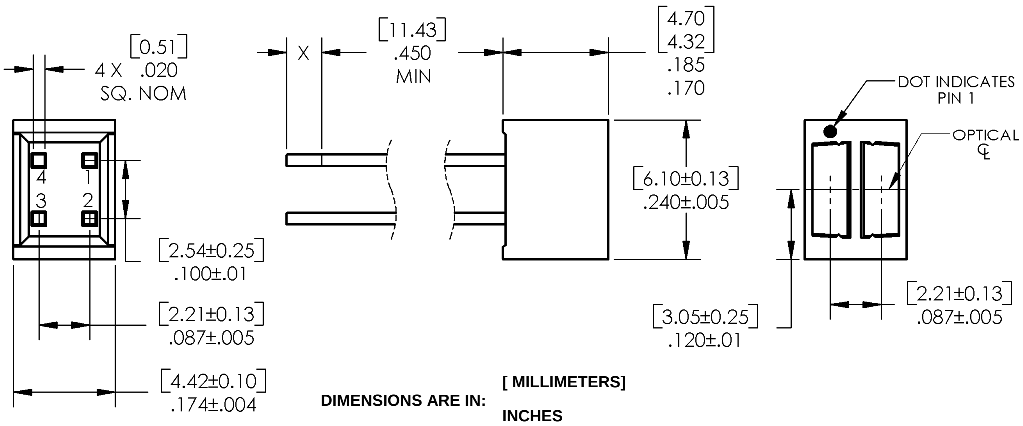
On both **OPB706** and **OPB707**, the LED and Phototransistor / Photodarlington are molded using dark infrared transmissive plastic to reduce ambient light noise. The Phototransistor / Photodarlington responds to light from the emitter when a reflective object passes within its field of view of the device.

Custom electrical, wire and cabling and connectors are available. Contact your local representative or OPTEK for more information.

Applications:

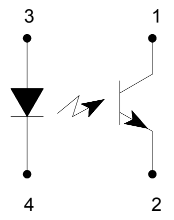
- Non-contact reflective object sensor
- Assembly line automation
- Machine automation
- Machine safety
- End of travel sensor
- Door sensor

Part Number	LED Peak Wavelength	Sensor	Reflection Distance	Lead Length / Spacing
OPB706A	935 nm	Transistor	0.050" (1.27mm)	0.45" / 0.087", 0.100"
OPB706B				
OPB706C				
OPB707A		Darlington		
OPB707B				
OPB707C				

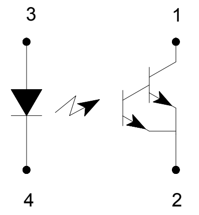


DIMENSIONS ARE IN: [MILLIMETERS]
INCHES

OPB706



OPB707



Pin #	LED	Pin #	Transistor
3	Anode	1	Collector
4	Cathode	2	Emitter

RoHS

OPTEK reserves the right to make changes at any time in order to improve design and to supply the best product possible.

Absolute Maximum Ratings (T_A=25°C unless otherwise noted)

Storage and Operating Temperature Range	-40° C to +85° C
Lead Soldering Temperature [1/16 inch (1.6mm) from the case for 5 sec. with soldering iron][(1)]	260° C

Input Diode

Forward DC Current	50 mA
Peak Forward Current (1 µs pulse width, 300 pps)	3 A
Reverse DC Voltage	2 V
Power Dissipation[(2)]	75 mW

Output Phototransistor (OPB706) | Output Photodarlington (OPB707)

Collector-Emitter Voltage OPB706 OPB707	 24 V 15 V
Emitter-Collector Voltage	5 V
Collector DC Current OPB706 OPB707	 25 mA 125 mA
Power Dissipation OPB706[(2)] OPB707[(3)]	 75 mW 100 mW

Notes:

(1) RMA flux is recommended. Duration can be extended to 10 seconds maximum when flow soldering.
(2) Derate linearly 1.25 mW/°C above 25 ° C.
(3) Derate linearly 1.67 mW/°C above 25 ° C.

OPTEK reserves the right to make changes at any time in order to improve design and to supply the best product possible.

Electrical Characteristics (T$_A$=25°C unless otherwise noted)

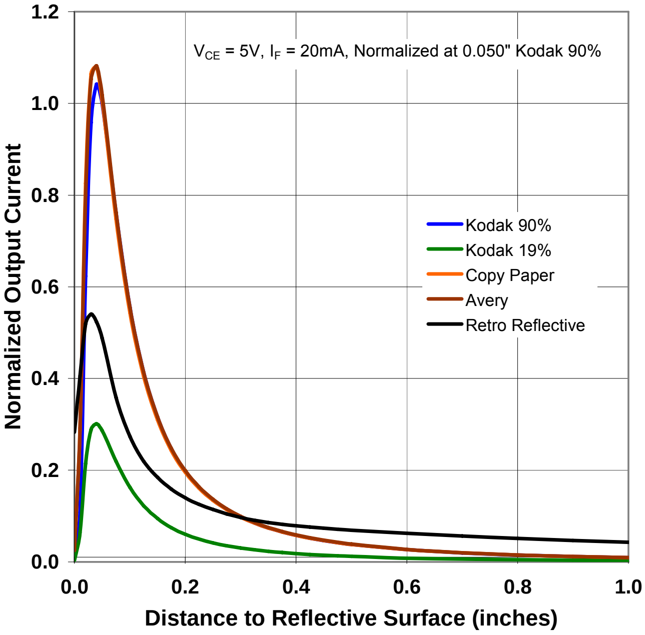
SYMBOL	PARAMETER	MIN	TYP	MAX	UNITS	TEST CONDITIONS
Input Diode (see OP165W for additional information)						
V_F	Forward Voltage	-	-	1.7	V	I_F = 20 mA
I_R	Reverse Current	-	-	100	µA	V_R = 2 V
Output Phototransistor (see OP505W for additional information) | **Photodarlington** (see OP535 for additional information)						
$V_{(BR)CEO}$	Collector-Emitter Brealdown Voltage OPB706 OPB707	24 15	- -	- -	V	I_C = 100 µA
$V_{(BR)ECO}$	Emitter-Collector Breakdown Voltage	5	-	-	V	I_E = 100 µA
I_{CEO}	Collector Dark Current OPB706 OPB707	- -	- -	100 250	nA	V_{CE} = 5 V, I_F = 0, $E_E \leq 0.1$ µW/cm^2
Combined						
I_{CX}	Crosstalk OPB706 OPB707	- -	- -	200 10	mA µA	I_F = 20 mA, V_{CE} = 5 V, No reflecting surface$^{(1)}$
$I_{C(ON)}$	On-State Collector Current OPB706A OPB706B OPB706C	500 350 250	- - -	- - -	µA	I_F = 20 mA, V_{CE} = 5V , d = 0.05" (1.27 mm)$^{(2)\,(3)}$
	OPB707A OPB707B OPB707C	25 17 10	- - -	- - -	mA	
$V_{CE(SAT)}$	Collector-Emitter Saturation Voltage OPB706 OPB707	0.4 1.1	- -	- -	V	I_F = 20 mA, d = 0.05" (1.27 mm)$^{(2)\,(3)}$ $I_{C(ON)}$ = 100µA $I_{C(ON)}$ = 2 mA

Notes:

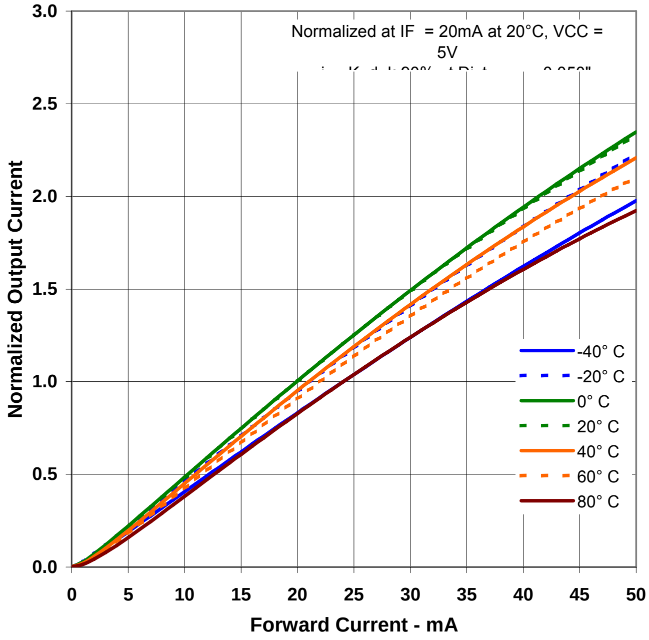
(1) Crosstalk (I_{CX}) is the collector current measured with the indicated current in the input diode and with no reflecting surface.
(2) The distance from the assembly face to the reflective surface is "d".
(3) Measured using Eastman Kodak neutral white test card with 90% diffuse reflectance as a reflecting surface. Reference: Eastman Kodak, Catalog #E 152 7795.
(4) Lower curve is a calculated worst case condition rather than the conventional -2 Ω limit.
(5) All parameters tested using pulse techniques.

OPTEK reserves the right to make changes at any time in order to improve design and to supply the best product possible.

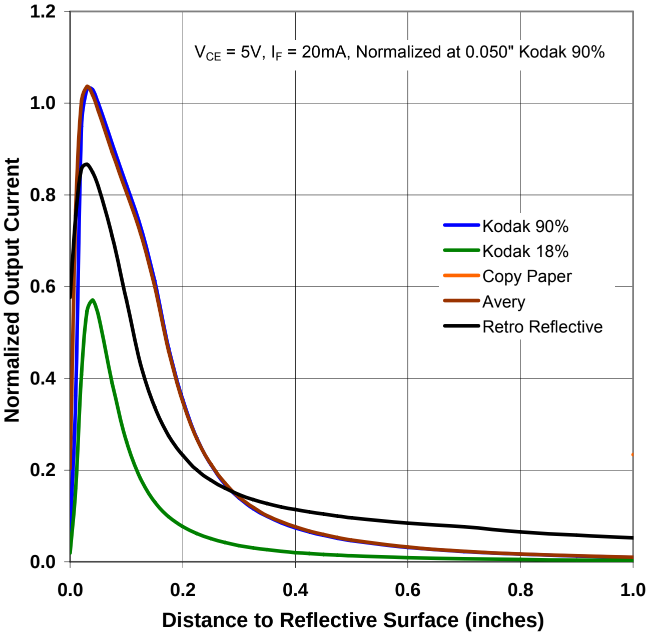
OPB706 - Normalized Collector Current vs. Object Distance



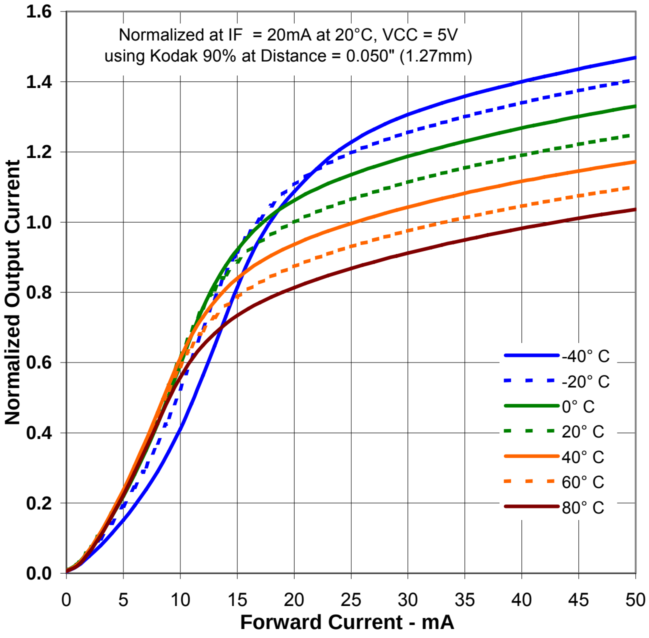
OPB706 - Output Current vs Forward Current vs Temperature



OPB707 - Normalized Collector Current vs. Object Distance



OPB707 - Output Current vs Forward Current vs Temperature



OPTEK reserves the right to make changes at any time in order to improve design and to supply the best product possible.

National Semiconductor

November 2000

LM35
Precision Centigrade Temperature Sensors

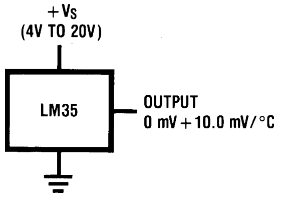
General Description

The LM35 series are precision integrated-circuit temperature sensors, whose output voltage is linearly proportional to the Celsius (Centigrade) temperature. The LM35 thus has an advantage over linear temperature sensors calibrated in ° Kelvin, as the user is not required to subtract a large constant voltage from its output to obtain convenient Centigrade scaling. The LM35 does not require any external calibration or trimming to provide typical accuracies of ±¼°C at room temperature and ±¾°C over a full −55 to +150°C temperature range. Low cost is assured by trimming and calibration at the wafer level. The LM35's low output impedance, linear output, and precise inherent calibration make interfacing to readout or control circuitry especially easy. It can be used with single power supplies, or with plus and minus supplies. As it draws only 60 µA from its supply, it has very low self-heating, less than 0.1°C in still air. The LM35 is rated to operate over a −55° to +150°C temperature range, while the LM35C is rated for a −40° to +110°C range (−10° with improved accuracy). The LM35 series is available packaged in hermetic TO-46 transistor packages, while the LM35C, LM35CA, and LM35D are also available in the plastic TO-92 transistor package. The LM35D is also available in an 8-lead surface mount small outline package and a plastic TO-220 package.

Features

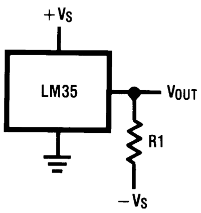
- Calibrated directly in ° Celsius (Centigrade)
- Linear + 10.0 mV/°C scale factor
- 0.5°C accuracy guaranteeable (at +25°C)
- Rated for full −55° to +150°C range
- Suitable for remote applications
- Low cost due to wafer-level trimming
- Operates from 4 to 30 volts
- Less than 60 µA current drain
- Low self-heating, 0.08°C in still air
- Nonlinearity only ±¼°C typical
- Low impedance output, 0.1 Ω for 1 mA load

Typical Applications



DS005516-3

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



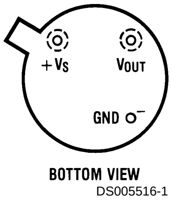
DS005516-4

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

FIGURE 2. Full-Range Centigrade Temperature Sensor

Connection Diagrams

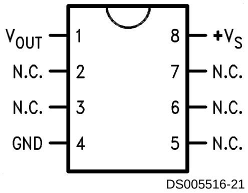
TO-46
Metal Can Package*



*Case is connected to negative pin (GND)

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

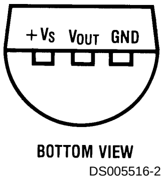
SO-8
Small Outline Molded Package



N.C. = No Connection

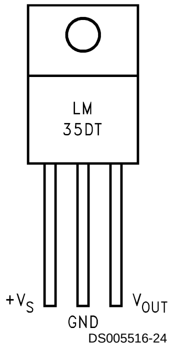
Top View
Order Number LM35DM
See NS Package Number M08A

TO-92
Plastic Package



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

TO-220
Plastic Package*



*Tab is connected to the negative pin (GND).

Note: The LM35DT pinout is different than the discontinued LM35DP.

Order Number LM35DT
See NS Package Number TA03F

Absolute Maximum Ratings (Note 10)

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

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

Electrical Characteristics

(Notes 1, 6)

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

Electrical Characteristics

(Notes 1, 6)

Parameter	Conditions	LM35			LM35C, LM35D			Units
		Typical	Tested Limit (Note 4)	Design Limit (Note 5)	Typical	Tested Limit (Note 4)	Design Limit (Note 5)	(Max.)
Accuracy,	T_A=+25˚C	±0.4	±1.0		±0.4	±1.0		˚C
LM35, LM35C	T_A=−10˚C	±0.5			±0.5		±1.5	˚C
(Note 7)	$T_A=T_{MAX}$	±0.8	±1.5		±0.8		±1.5	˚C
	$T_A=T_{MIN}$	±0.8		±1.5	±0.8		±2.0	˚C
Accuracy, LM35D	T_A=+25˚C				±0.6	±1.5		˚C
(Note 7)	$T_A=T_{MAX}$				±0.9		±2.0	˚C
	$T_A=T_{MIN}$				±0.9		±2.0	˚C
Nonlinearity	$T_{MIN}≤T_A≤T_{MAX}$	**±0.3**		**±0.5**	**±0.2**		**±0.5**	˚C
(Note 8)								
Sensor Gain	$T_{MIN}≤T_A≤T_{MAX}$	**+10.0**	**+9.8,**		**+10.0**		**+9.8,**	mV/˚C
(Average Slope)			**+10.2**				**+10.2**	
Load Regulation	T_A=+25˚C	±0.4	±2.0		±0.4	±2.0		mV/mA
(Note 3) 0≤I_L≤1 mA	$T_{MIN}≤T_A≤T_{MAX}$	**±0.5**		**±5.0**	**±0.5**		**±5.0**	mV/mA
Line Regulation	T_A=+25˚C	±0.01	±0.1		±0.01	±0.1		mV/V
(Note 3)	4V≤V_S≤30V	**±0.02**		**±0.2**	**±0.02**		**±0.2**	mV/V
Quiescent Current	V_S=+5V, +25˚C	56	80		56	80		µA
(Note 9)	V_S=+5V	**105**		**158**	**91**		**138**	µA
	V_S=+30V, +25˚C	56.2	82		56.2	82		µA
	V_S=+30V	**105.5**		**161**	**91.5**		**141**	µA
Change of	4V≤V_S≤30V, +25˚C	0.2	2.0		0.2	2.0		µA
Quiescent Current	4V≤V_S≤30V	**0.5**		**3.0**	**0.5**		**3.0**	µA
(Note 3)								
Temperature Coefficient of Quiescent Current		**+0.39**		**+0.7**	**+0.39**		**+0.7**	µA/˚C
Minimum Temperature for Rated Accuracy	In circuit of *Figure 1*, I_L=0	+1.5		+2.0	+1.5		+2.0	˚C
Long Term Stability	$T_J=T_{MAX}$, for 1000 hours	±0.08			±0.08			˚C

Note 1: Unless otherwise noted, these specifications apply: −55˚C≤T_J≤+150˚C for the LM35 and LM35A; −40˚≤T_J≤+110˚C for the LM35C and LM35CA; and 0˚≤T_J≤+100˚C for the LM35D. V_S=+5Vdc and I_{LOAD}=50 µA, in the circuit of *Figure 2*. These specifications also apply from +2˚C to T_{MAX} in the circuit of *Figure 1*. Specifications in **boldface** apply over the full rated temperature range.

Note 2: Thermal resistance of the TO-46 package is 400˚C/W, junction to ambient, and 24˚C/W junction to case. Thermal resistance of the TO-92 package is 180˚C/W junction to ambient. Thermal resistance of the small outline molded package is 220˚C/W junction to ambient. Thermal resistance of the TO-220 package is 90˚C/W junction to ambient. For additional thermal resistance information see table in the Applications section.

Note 3: Regulation is measured at constant junction temperature, using pulse testing with a low duty cycle. Changes in output due to heating effects can be computed by multiplying the internal dissipation by the thermal resistance.

Note 4: Tested Limits are guaranteed and 100% tested in production.

Note 5: Design Limits are guaranteed (but not 100% production tested) over the indicated temperature and supply voltage ranges. These limits are not used to calculate outgoing quality levels.

Note 6: Specifications in **boldface** apply over the full rated temperature range.

Note 7: Accuracy is defined as the error between the output voltage and 10mv/˚C times the device's case temperature, at specified conditions of voltage, current, and temperature (expressed in ˚C).

Note 8: Nonlinearity is defined as the deviation of the output-voltage-versus-temperature curve from the best-fit straight line, over the device's rated temperature range.

Note 9: Quiescent current is defined in the circuit of *Figure 1*.

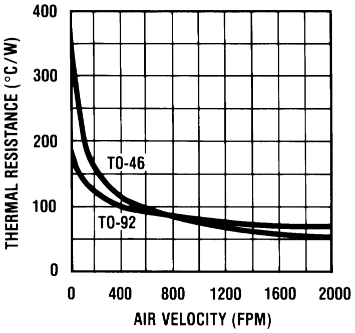
Note 10: Absolute Maximum Ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications do not apply when operating the device beyond its rated operating conditions. See Note 1.

Note 11: Human body model, 100 pF discharged through a 1.5 kΩ resistor.

Note 12: See AN-450 "Surface Mounting Methods and Their Effect on Product Reliability" or the section titled "Surface Mount" found in a current National Semiconductor Linear Data Book for other methods of soldering surface mount devices.

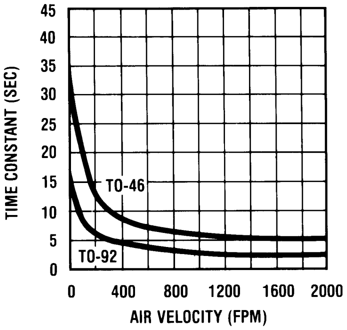
Typical Performance Characteristics

Thermal Resistance Junction to Air



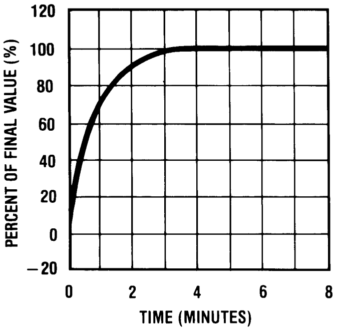
DS005516-25

Thermal Time Constant



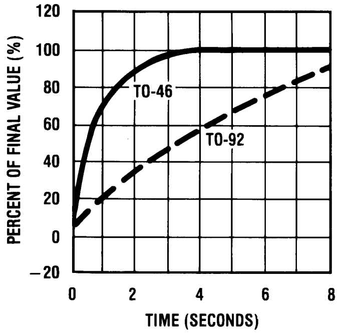
DS005516-26

Thermal Response in Still Air



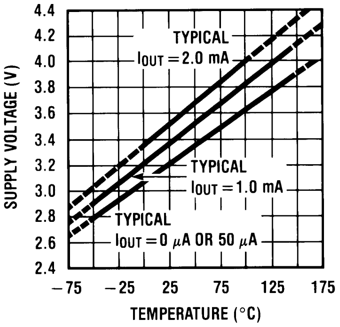
DS005516-27

Thermal Response in Stirred Oil Bath



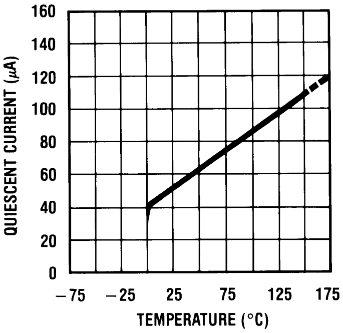
DS005516-28

Minimum Supply Voltage vs. Temperature



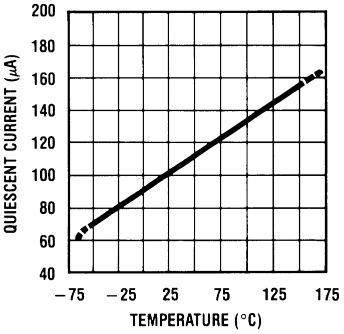
DS005516-29

Quiescent Current vs. Temperature (In Circuit of *Figure 1*.)



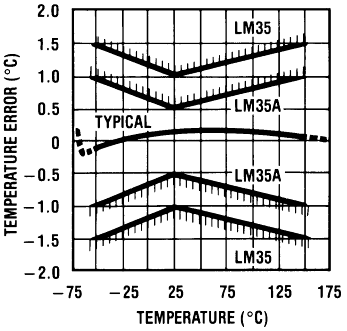
DS005516-30

Quiescent Current vs. Temperature (In Circuit of *Figure 2*.)



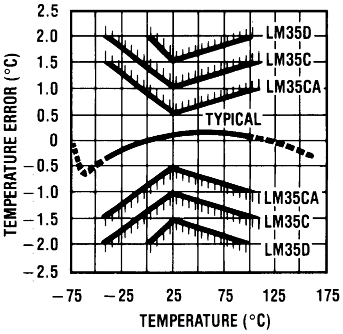
DS005516-31

Accuracy vs. Temperature (Guaranteed)



DS005516-32

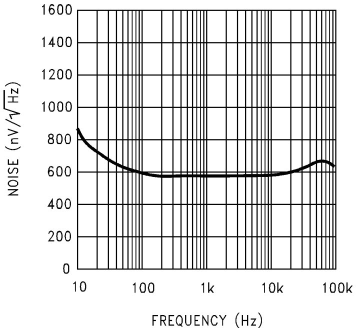
Accuracy vs. Temperature (Guaranteed)



DS005516-33

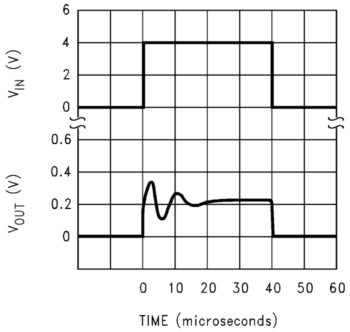
Typical Performance Characteristics (Continued)

Noise Voltage



DS005516-34

Start-Up Response



DS005516-35

Applications

The LM35 can be applied easily in the same way as other integrated-circuit temperature sensors. It can be glued or cemented to a surface and its temperature will be within about 0.01˚C of the surface temperature.

This presumes that the ambient air temperature is almost the same as the surface temperature; if the air temperature were much higher or lower than the surface temperature, the actual temperature of the LM35 die would be at an intermediate temperature between the surface temperature and the air temperature. This is expecially true for the TO-92 plastic package, where the copper leads are the principal thermal path to carry heat into the device, so its temperature might be closer to the air temperature than to the surface temperature.

To minimize this problem, be sure that the wiring to the LM35, as it leaves the device, is held at the same temperature as the surface of interest. The easiest way to do this is to cover up these wires with a bead of epoxy which will insure that the leads and wires are all at the same temperature as the surface, and that the LM35 die's temperature will not be affected by the air temperature.

The TO-46 metal package can also be soldered to a metal surface or pipe without damage. Of course, in that case the V− terminal of the circuit will be grounded to that metal. Alternatively, the LM35 can be mounted inside a sealed-end metal tube, and can then be dipped into a bath or screwed into a threaded hole in a tank. As with any IC, the LM35 and accompanying wiring and circuits must be kept insulated and dry, to avoid leakage and corrosion. This is especially true if the circuit may operate at cold temperatures where condensation can occur. Printed-circuit coatings and varnishes such as Humiseal and epoxy paints or dips are often used to insure that moisture cannot corrode the LM35 or its connections.

These devices are sometimes soldered to a small light-weight heat fin, to decrease the thermal time constant and speed up the response in slowly-moving air. On the other hand, a small thermal mass may be added to the sensor, to give the steadiest reading despite small deviations in the air temperature.

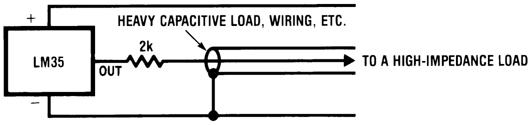
Temperature Rise of LM35 Due To Self-heating (Thermal Resistance,θ_{JA})

	TO-46,	**TO-46*,**	**TO-92,**	**TO-92**,**	**SO-8**	**SO-8****	**TO-220**
	no heat sink	**small heat fin**	**no heat sink**	**small heat fin**	**no heat sink**	**small heat fin**	**no heat sink**
Still air	400˚C/W	100˚C/W	180˚C/W	140˚C/W	220˚C/W	110˚C/W	90˚C/W
Moving air	100˚C/W	40˚C/W	90˚C/W	70˚C/W	105˚C/W	90˚C/W	26˚C/W
Still oil	100˚C/W	40˚C/W	90˚C/W	70˚C/W			
Stirred oil	50˚C/W	30˚C/W	45˚C/W	40˚C/W			
(Clamped to metal,							
Infinite heat sink)	(24˚C/W)				(55˚C/W)		

*Wakefield type 201, or 1" disc of 0.020" sheet brass, soldered to case, or similar.

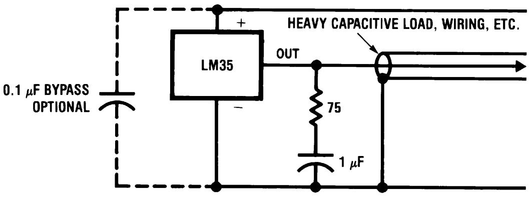
**TO-92 and SO-8 packages glued and leads soldered to 1" square of 1/16" printed circuit board with 2 oz. foil or similar.

Typical Applications



DS005516-19

FIGURE 3. LM35 with Decoupling from Capacitive Load



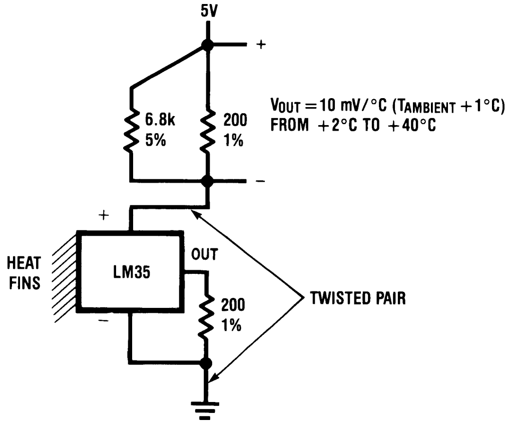
DS005516-20

FIGURE 4. LM35 with R-C Damper

CAPACITIVE LOADS

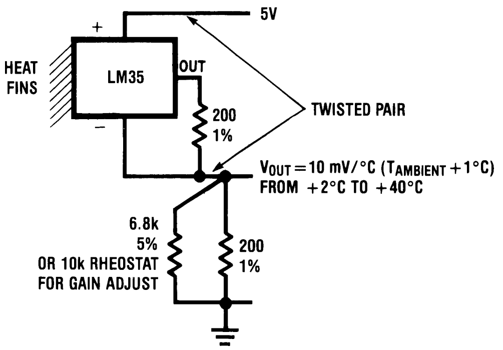
Like most micropower circuits, the LM35 has a limited ability to drive heavy capacitive loads. The LM35 by itself is able to drive 50 pf without special precautions. If heavier loads are anticipated, it is easy to isolate or decouple the load with a resistor; see *Figure 3*. Or you can improve the tolerance of capacitance with a series R-C damper from output to ground; see *Figure 4*.

When the LM35 is applied with a 200Ω load resistor as shown in *Figure 5*, *Figure 6* or *Figure 8* it is relatively immune to wiring capacitance because the capacitance forms a bypass from ground to input, not on the output. However, as with any linear circuit connected to wires in a hostile environment, its performance can be affected adversely by intense electromagnetic sources such as relays, radio transmitters, motors with arcing brushes, SCR transients, etc, as its wiring can act as a receiving antenna and its internal junctions can act as rectifiers. For best results in such cases, a bypass capacitor from V_{IN} to ground and a series R-C damper such as 75Ω in series with 0.2 or 1 µF from output to ground are often useful. These are shown in *Figure 13*, *Figure 14*, and *Figure 16*.



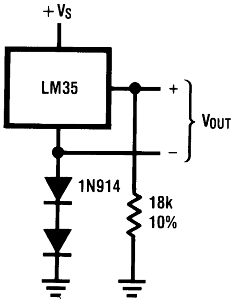
DS005516-5

FIGURE 5. Two-Wire Remote Temperature Sensor (Grounded Sensor)



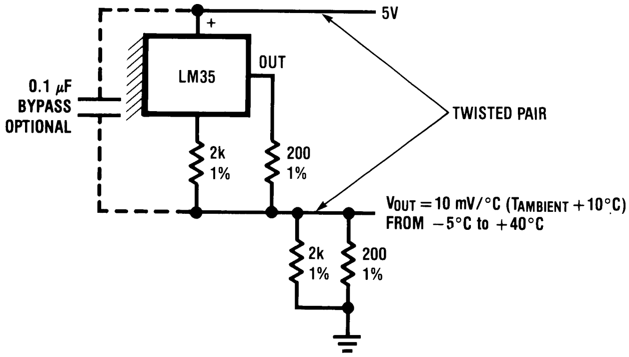
DS005516-6

FIGURE 6. Two-Wire Remote Temperature Sensor (Output Referred to Ground)



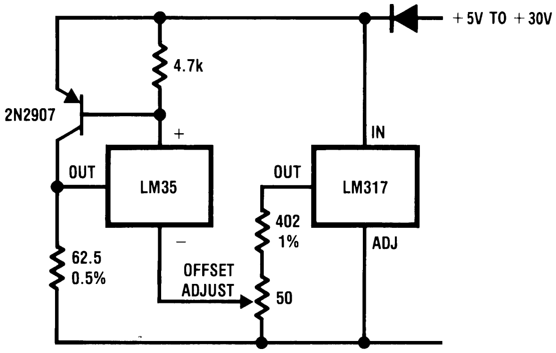
DS005516-7

FIGURE 7. Temperature Sensor, Single Supply, −55˚ to +150˚C



DS005516-8

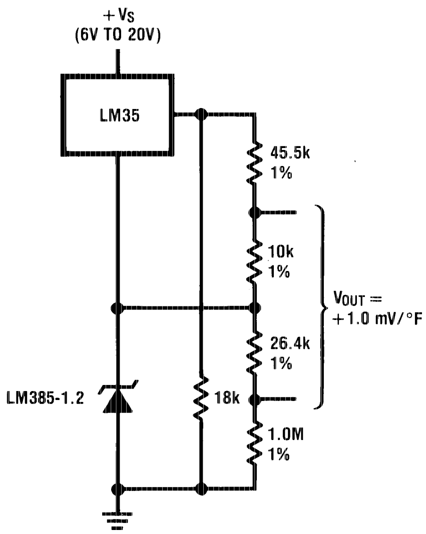
FIGURE 8. Two-Wire Remote Temperature Sensor (Output Referred to Ground)



DS005516-9

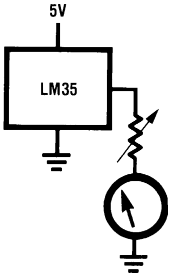
FIGURE 9. 4-To-20 mA Current Source (0˚C to +100˚C)

Typical Applications (Continued)



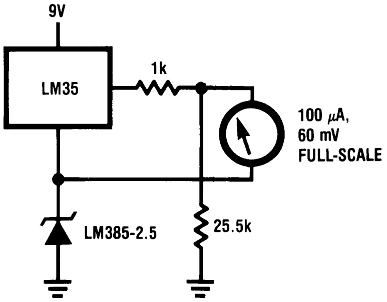
DS005516-10

FIGURE 10. Fahrenheit Thermometer



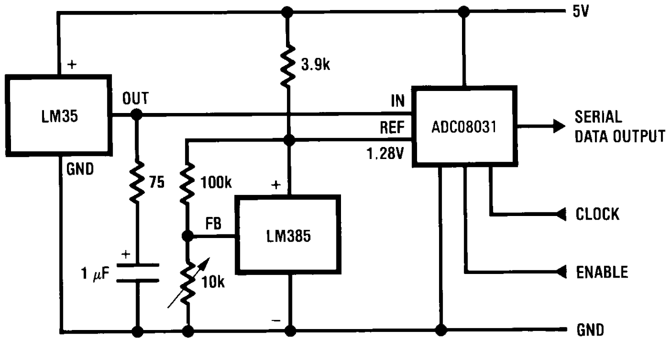
DS005516-11

FIGURE 11. Centigrade Thermometer (Analog Meter)



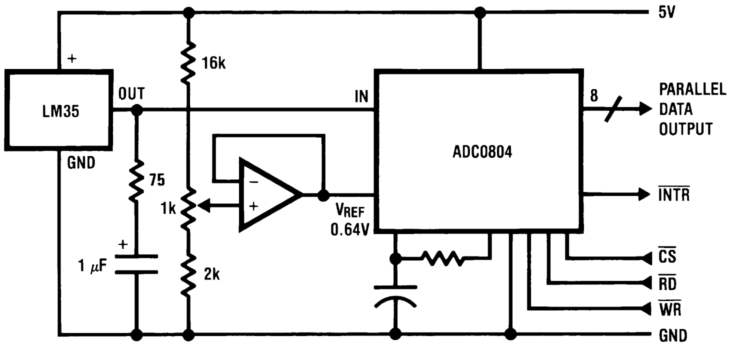
DS005516-12

FIGURE 12. Fahrenheit ThermometerExpanded Scale Thermometer (50˚ to 80˚ Fahrenheit, for Example Shown)



DS005516-13

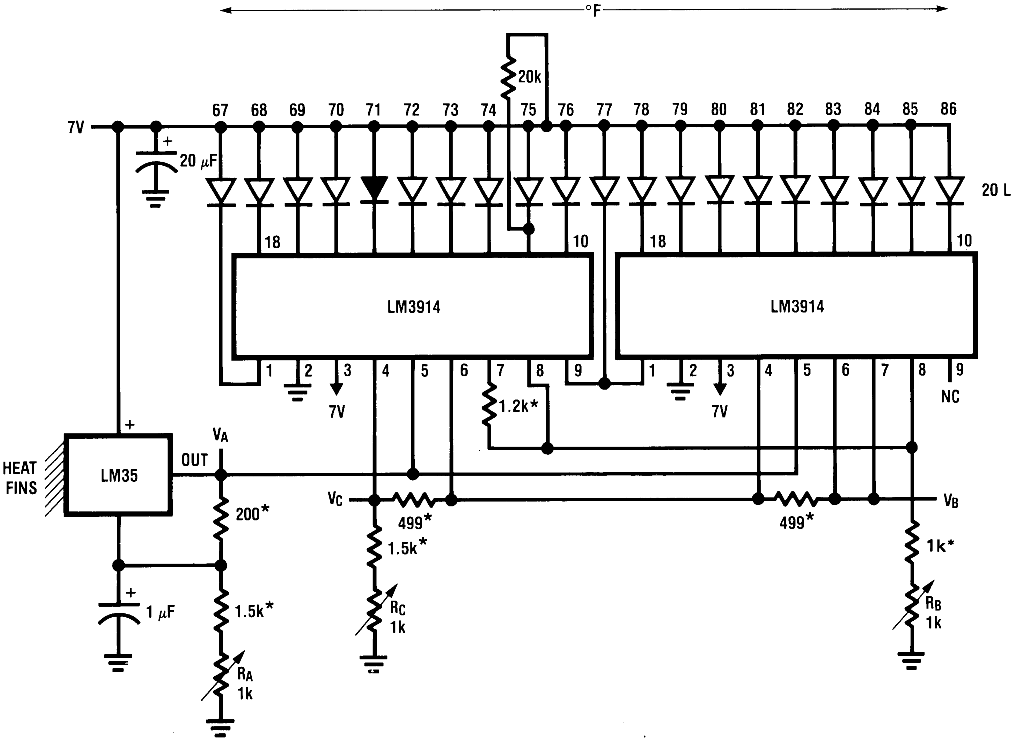
FIGURE 13. Temperature To Digital Converter (Serial Output) (+128˚C Full Scale)



DS005516-14

FIGURE 14. Temperature To Digital Converter (Parallel TRI-STATE™ Outputs for Standard Data Bus to μP Interface) (128˚C Full Scale)

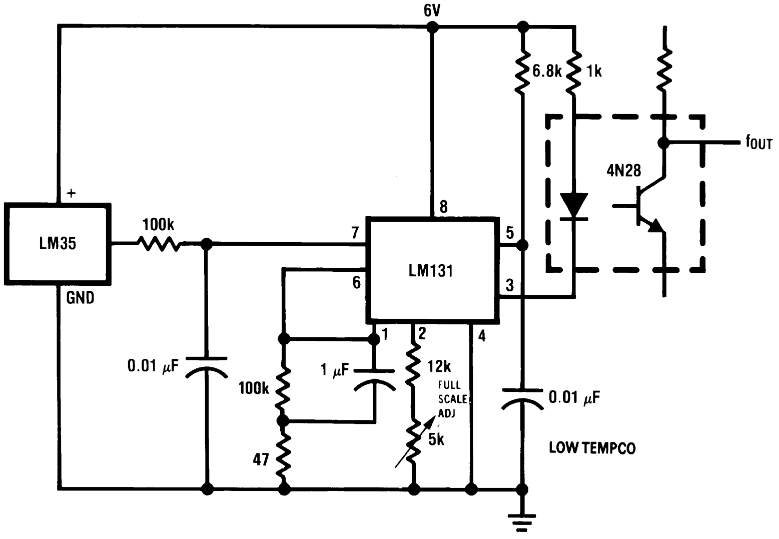
Typical Applications (Continued)



DS005516-16

*=1% or 2% film resistor
Trim R_B for V_B=3.075V
Trim R_C for V_C=1.955V
Trim R_A for V_A=0.075V + 100mV/˚C x $T_{ambient}$
Example, V_A=2.275V at 22˚C

FIGURE 15. Bar-Graph Temperature Display (Dot Mode)



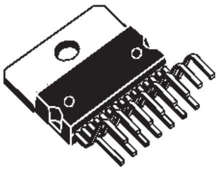
DS005516-15

FIGURE 16. LM35 With Voltage-To-Frequency Converter And Isolated Output (2˚C to +150˚C; 20 Hz to 1500 Hz)

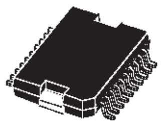
L298

DUAL FULL-BRIDGE DRIVER

- OPERATING SUPPLY VOLTAGE UP TO 46 V
- TOTAL DC CURRENT UP TO 4 A
- LOW SATURATION VOLTAGE
- OVERTEMPERATURE PROTECTION
- LOGICAL "0" INPUT VOLTAGE UP TO 1.5 V (HIGH NOISE IMMUNITY)



Multiwatt15



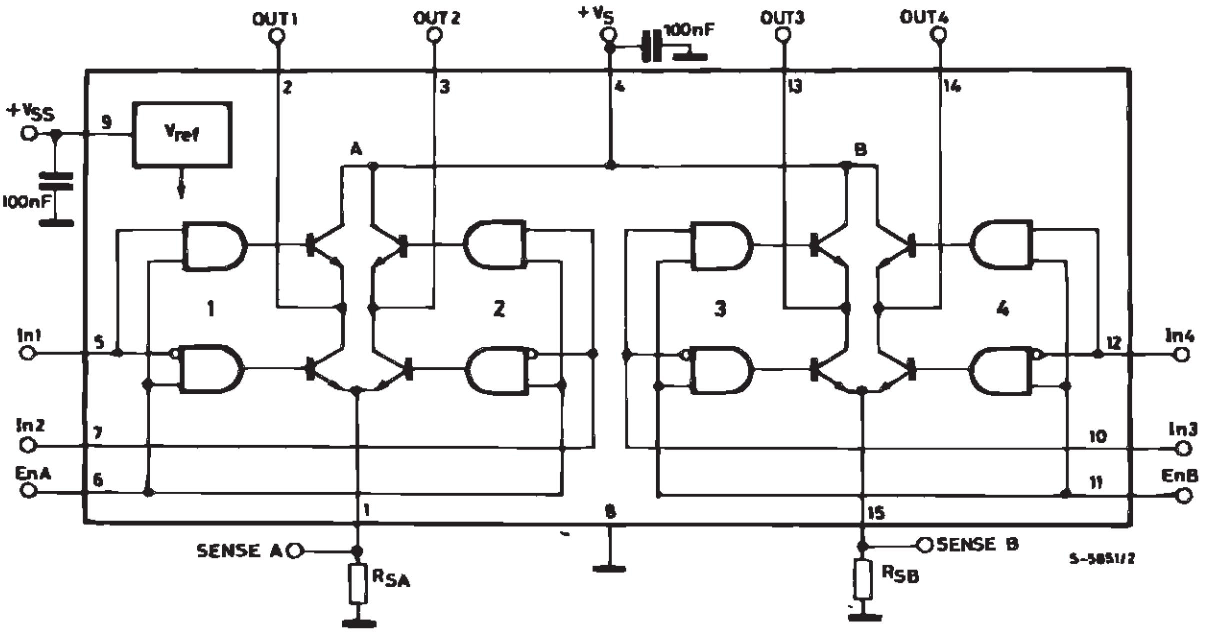
PowerSO20

ORDERING NUMBERS : L298N (Multiwatt Vert.)
L298HN (Multiwatt Horiz.)
L298P (PowerSO20)

DESCRIPTION

The L298 is an integrated monolithic circuit in a 15-lead Multiwatt and PowerSO20 packages. It is a high voltage, high current dual full-bridge driver designed to accept standard TTL logic levels and drive inductive loads such as relays, solenoids, DC and stepping motors. Two enable inputs are provided to enable or disable the device independently of the input signals. The emitters of the lower transistors of each bridge are connected together and the corresponding external terminal can be used for the connection of an external sensing resistor. An additional supply input is provided so that the logic works at a lower voltage.

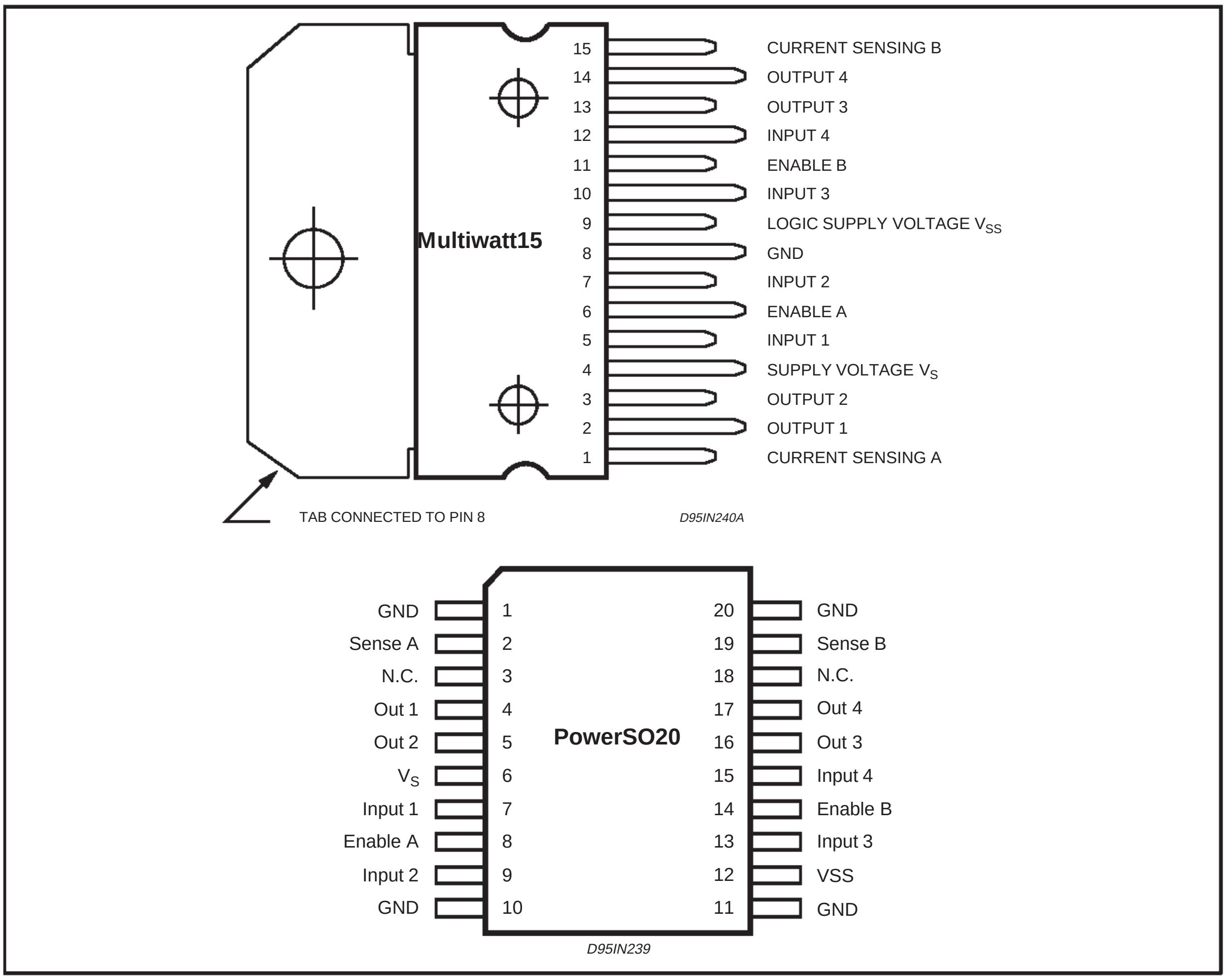
BLOCK DIAGRAM



ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_S	Power Supply	50	V
V_{SS}	Logic Supply Voltage	7	V
V_I,V_{en}	Input and Enable Voltage	–0.3 to 7	V
I_O	Peak Output Current (each Channel) – Non Repetitive (t = 100µs) –Repetitive (80% on –20% off; t_{on} = 10ms) –DC Operation	 3 2.5 2	 A A A
V_{sens}	Sensing Voltage	–1 to 2.3	V
P_{tot}	Total Power Dissipation (T_{case} = 75°C)	25	W
T_{op}	Junction Operating Temperature	–25 to 130	°C
T_{stg}, T_j	Storage and Junction Temperature	–40 to 150	°C

PIN CONNECTIONS (top view)



THERMAL DATA

Symbol	Parameter		PowerSO20	Multiwatt15	Unit
$R_{th\ j\text{-}case}$	Thermal Resistance Junction-case	Max.	–	3	°C/W
$R_{th\ j\text{-}amb}$	Thermal Resistance Junction-ambient	Max.	13 (*)	35	°C/W

(*) Mounted on aluminum substrate

PIN FUNCTIONS (refer to the block diagram)

MW.15	PowerSO	Name	Function
1;15	2;19	Sense A; Sense B	Between this pin and ground is connected the sense resistor to control the current of the load.
2;3	4;5	Out 1; Out 2	Outputs of the Bridge A; the current that flows through the load connected between these two pins is monitored at pin 1.
4	6	V_S	Supply Voltage for the Power Output Stages. A non-inductive 100nF capacitor must be connected between this pin and ground.
5;7	7;9	Input 1; Input 2	TTL Compatible Inputs of the Bridge A.
6;11	8;14	Enable A; Enable B	TTL Compatible Enable Input: the L state disables the bridge A (enable A) and/or the bridge B (enable B).
8	1,10,11,20	GND	Ground.
9	12	VSS	Supply Voltage for the Logic Blocks. A100nF capacitor must be connected between this pin and ground.
10; 12	13;15	Input 3; Input 4	TTL Compatible Inputs of the Bridge B.
13; 14	16;17	Out 3; Out 4	Outputs of the Bridge B. The current that flows through the load connected between these two pins is monitored at pin 15.
–	3;18	N.C.	Not Connected

ELECTRICAL CHARACTERISTICS (V_S = 42V; V_{SS} = 5V, T_j = 25°C; unless otherwise specified)

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
V_S	Supply Voltage (pin 4)	Operative Condition	V_{IH} +2.5		46	V
V_{SS}	Logic Supply Voltage (pin 9)		4.5	5	7	V
I_S	Quiescent Supply Current (pin 4)	V_{en} = H; I_L = 0 V_i = L V_i = H		13 50	22 70	mA mA
		V_{en} = L V_i = X			4	mA
I_{SS}	Quiescent Current from V_{SS} (pin 9)	V_{en} = H; I_L = 0 V_i = L V_i = H		24 7	36 12	mA mA
		V_{en} = L V_i = X			6	mA
V_{iL}	Input Low Voltage (pins 5, 7, 10, 12)		–0.3		1.5	V
V_{iH}	Input High Voltage (pins 5, 7, 10, 12)		2.3		VSS	V
I_{iL}	Low Voltage Input Current (pins 5, 7, 10, 12)	V_i = L			–10	µA
I_{iH}	High Voltage Input Current (pins 5, 7, 10, 12)	Vi = H ≤ V_{SS} –0.6V		30	100	µA
V_{en} = L	Enable Low Voltage (pins 6, 11)		–0.3		1.5	V
V_{en} = H	Enable High Voltage (pins 6, 11)		2.3		V_{SS}	V
I_{en} = L	Low Voltage Enable Current (pins 6, 11)	V_{en} = L			–10	µA
I_{en} = H	High Voltage Enable Current (pins 6, 11)	V_{en} = H ≤ V_{SS} –0.6V		30	100	µA
$V_{CEsat(H)}$	Source Saturation Voltage	I_L = 1A I_L = 2A	0.95	1.35 2	1.7 2.7	V V
$V_{CEsat(L)}$	Sink Saturation Voltage	I_L = 1A (5) I_L = 2A (5)	0.85	1.2 1.7	1.6 2.3	V V
V_{CEsat}	Total Drop	I_L = 1A (5) I_L = 2A (5)	1.80		3.2 4.9	V V
V_{sens}	Sensing Voltage (pins 1, 15)		–1 (1)		2	V

ELECTRICAL CHARACTERISTICS (continued)

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
T_1 (V_i)	Source Current Turn-off Delay	0.5 V_i to 0.9 I_L (2); (4)		1.5		µs
T_2 (V_i)	Source Current Fall Time	0.9 I_L to 0.1 I_L (2); (4)		0.2		µs
T_3 (V_i)	Source Current Turn-on Delay	0.5 V_i to 0.1 I_L (2); (4)		2		µs
T_4 (V_i)	Source Current Rise Time	0.1 I_L to 0.9 I_L (2); (4)		0.7		µs
T_5 (V_i)	Sink Current Turn-off Delay	0.5 V_i to 0.9 I_L (3); (4)		0.7		µs
T_6 (V_i)	Sink Current Fall Time	0.9 I_L to 0.1 I_L (3); (4)		0.25		µs
T_7 (V_i)	Sink Current Turn-on Delay	0.5 V_i to 0.9 I_L (3); (4)		1.6		µs
T_8 (V_i)	Sink Current Rise Time	0.1 I_L to 0.9 I_L (3); (4)		0.2		µs
fc (V_i)	Commutation Frequency	I_L = 2A		25	40	KHz
T_1 (V_{en})	Source Current Turn-off Delay	0.5 V_{en} to 0.9 I_L (2); (4)		3		µs
T_2 (V_{en})	Source Current Fall Time	0.9 I_L to 0.1 I_L (2); (4)		1		µs
T_3 (V_{en})	Source Current Turn-on Delay	0.5 V_{en} to 0.1 I_L (2); (4)		0.3		µs
T_4 (V_{en})	Source Current Rise Time	0.1 I_L to 0.9 I_L (2); (4)		0.4		µs
T_5 (V_{en})	Sink Current Turn-off Delay	0.5 V_{en} to 0.9 I_L (3); (4)		2.2		µs
T_6 (V_{en})	Sink Current Fall Time	0.9 I_L to 0.1 I_L (3); (4)		0.35		µs
T_7 (V_{en})	Sink Current Turn-on Delay	0.5 V_{en} to 0.9 I_L (3); (4)		0.25		µs
T_8 (V_{en})	Sink Current Rise Time	0.1 I_L to 0.9 I_L (3); (4)		0.1		µs

1) 1)Sensing voltage can be –1 V for $t \leq 50$ µsec; in steady state V_{sens} min ≥ –0.5 V.
2) See fig. 2.
3) See fig. 4.
4) The load must be a pure resistor.

Figure 1 : Typical Saturation Voltage vs. Output Current.

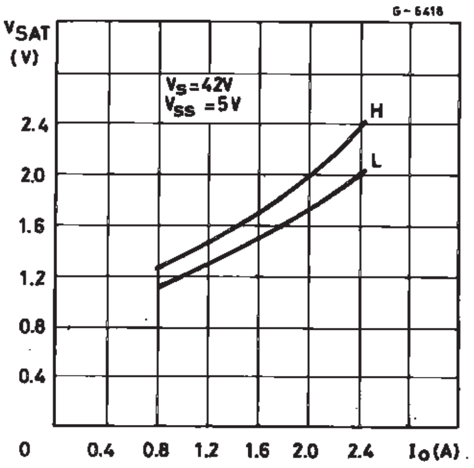
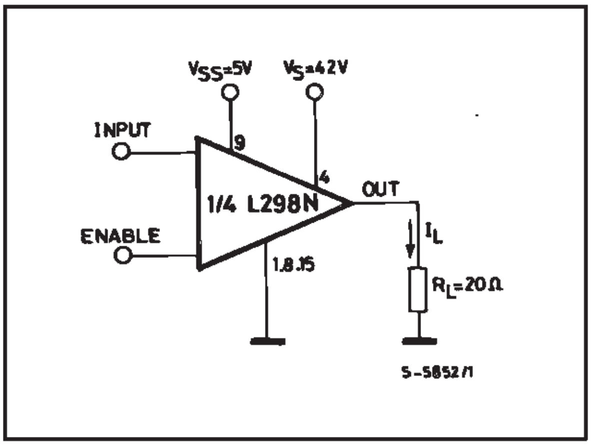


Figure 2 : Switching Times Test Circuits.



Note : For INPUT Switching, set EN = H
For ENABLE Switching, set IN = H

Figure 3 : Source Current Delay Times vs. Input or Enable Switching.

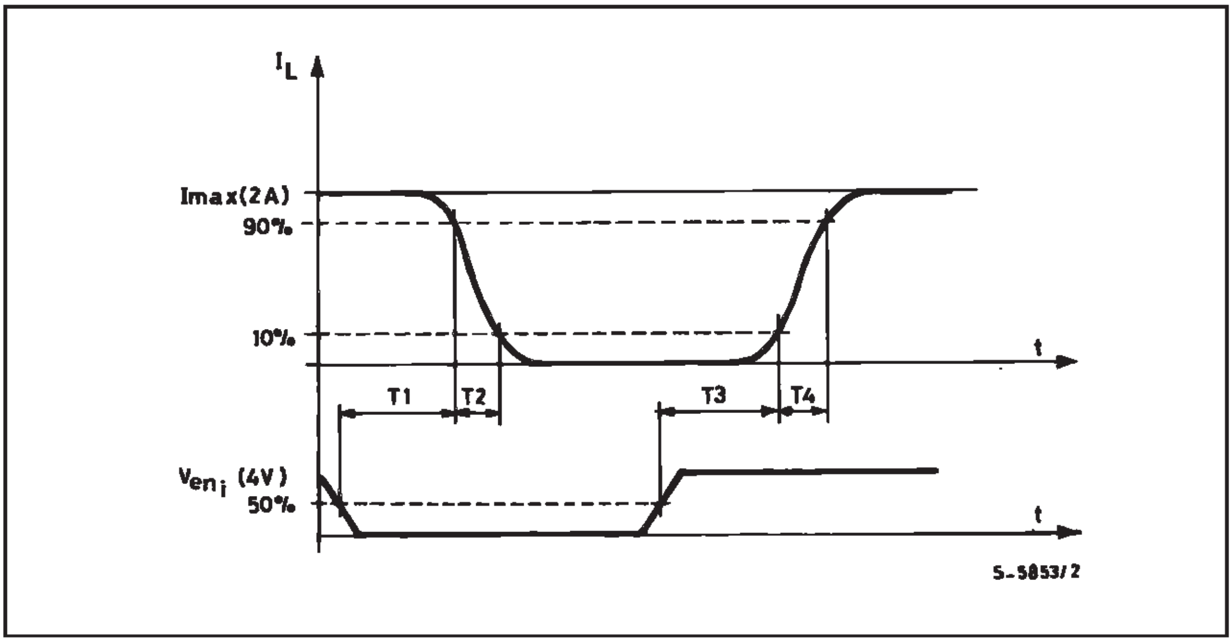
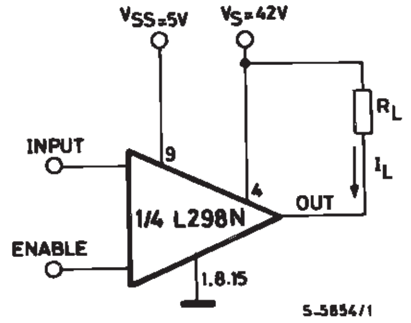


Figure 4 : Switching Times Test Circuits.



Note : For INPUT Switching, set EN = H
For ENABLE Switching, set IN = L

Figure 5 : Sink Current Delay Times vs. Input 0 V Enable Switching.

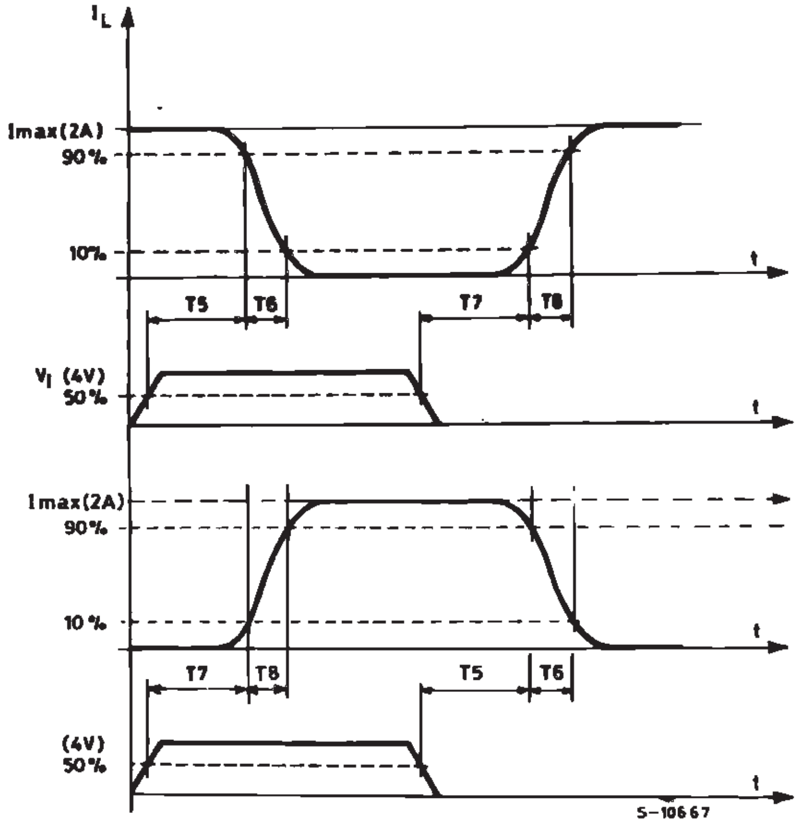
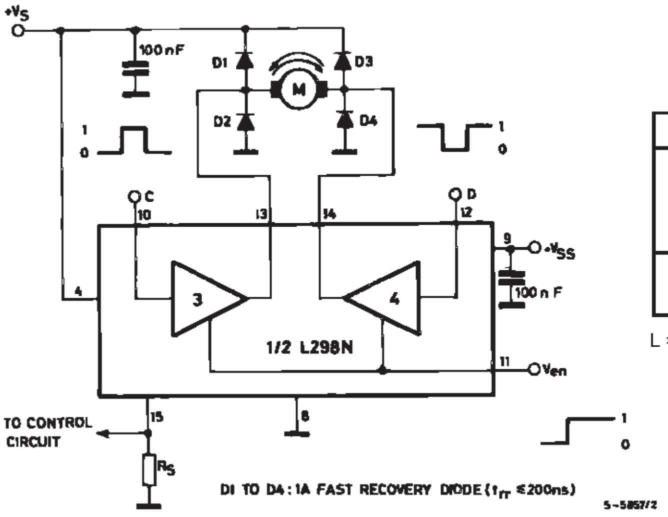


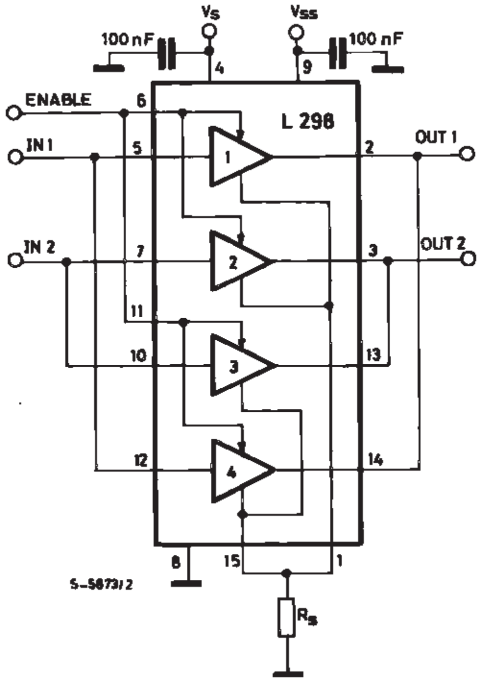
Figure 6 : Bidirectional DC Motor Control.



Inputs		Function
V_{en} = H	C = H ; D = L	Forward
	C = L ; D = H	Reverse
	C = D	Fast Motor Stop
V_{en} = L	C = X ; D = X	Free Running Motor Stop

L = Low H = High X = Don't care

Figure 7 : For higher currents, outputs can be paralleled. Take care to parallel channel 1 with channel 4 and channel 2 with channel 3.



APPLICATION INFORMATION (Refer to the block diagram)

1.1. POWER OUTPUT STAGE

The L298 integrates two power output stages (A ; B). The power output stage is a bridge configuration and its outputs can drive an inductive load in common or differenzial mode, depending on the state of the inputs. The current that flows through the load comes out from the bridge at the sense output : an external resistor (R_{SA} ; R_{SB}.) allows to detect the intensity of this current.

1.2. INPUT STAGE

Each bridge is driven by means of four gates the input of which are In1 ; In2 ; EnA and In3 ; In4 ; EnB. The In inputs set the bridge state when The En input is high ; a low state of the En input inhibits the bridge. All the inputs are TTL compatible.

2. SUGGESTIONS

A non inductive capacitor, usually of 100 nF, must be foreseen between both Vs and Vss, to ground, as near as possible to GND pin. When the large capacitor of the power supply is too far from the IC, a second smaller one must be foreseen near the L298.

The sense resistor, not of a wire wound type, must be grounded near the negative pole of Vs that must be near the GND pin of the I.C.

Each input must be connected to the source of the driving signals by means of a very short path.

Turn-On and Turn-Off : Before to Turn-ON the Supply Voltage and before to Turn it OFF, the Enable input must be driven to the Low state.

3. APPLICATIONS

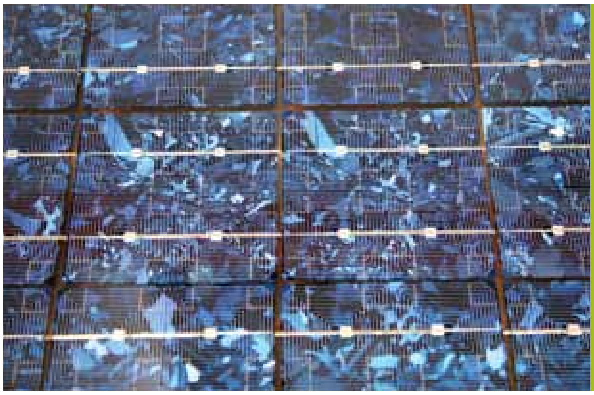
Fig 6 shows a bidirectional DC motor control Schematic Diagram for which only one bridge is needed. The external bridge of diodes D1 to D4 is made by four fast recovery elements (trr ≤ 200 nsec) that must be chosen of a VF as low as possible at the worst case of the load current.

The sense output voltage can be used to control the current amplitude by chopping the inputs, or to provide overcurrent protection by switching low the enable input.

The brake function (Fast motor stop) requires that the Absolute Maximum Rating of 2 Amps must never be overcome.

When the repetitive peak current needed from the load is higher than 2 Amps, a paralleled configuration can be chosen (See Fig.7).

An external bridge of diodes are required when inductive loads are driven and when the inputs of the IC are chopped ; Shottky diodes would be preferred.



UNIVERSITÉ
Concordia
UNIVERSITY

www.me.concordia.ca