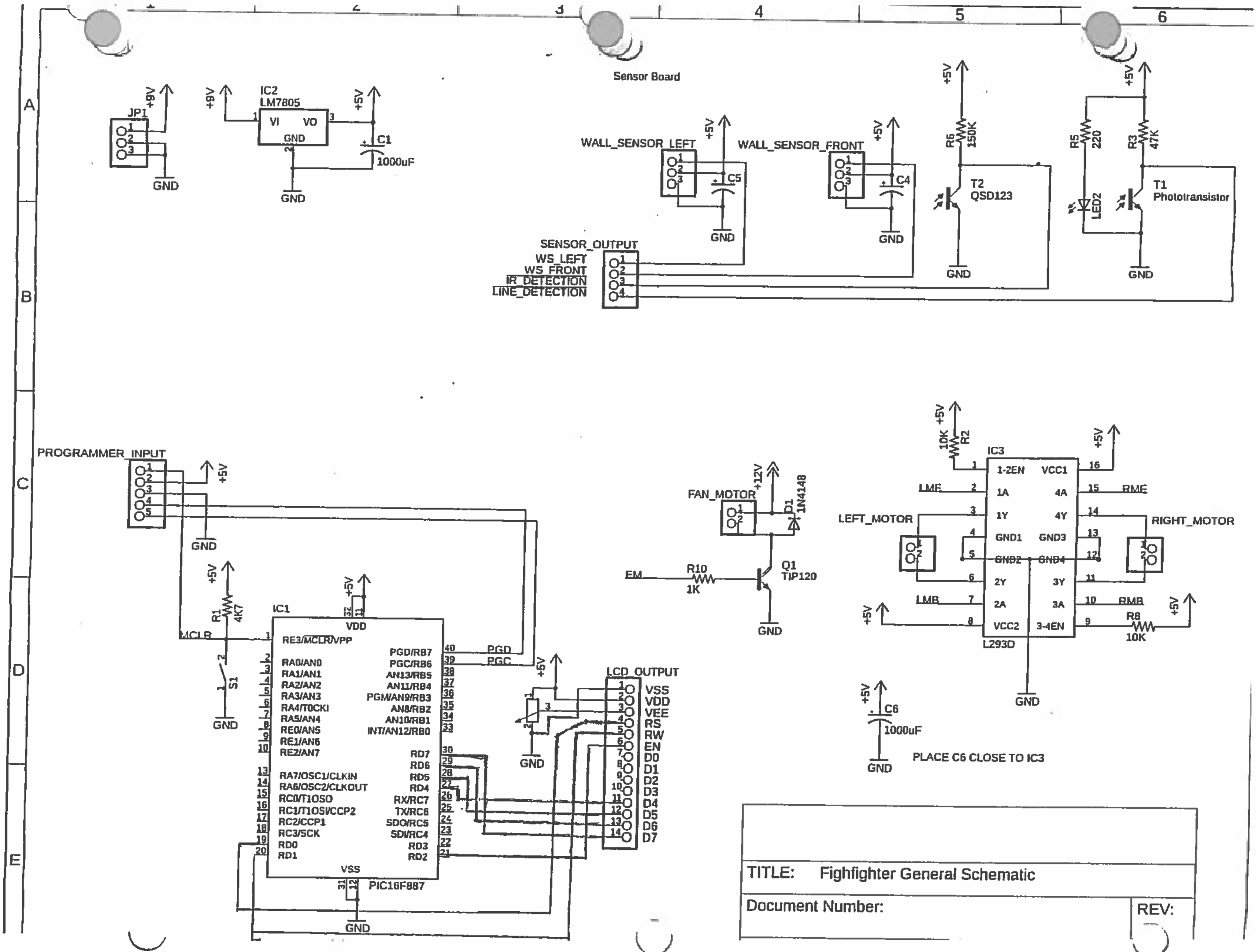

Sensor Board
JP1
+9V
GND
IC2
LM7805
VI
VO
C1
1000uF
+5V
WALL_SENSOR_LEFT
WALL_SENSOR_FRONT
C5
C4
R6
150K
T2
QSD123
R5
220
LED2
R3
47K
T1
Phototransistor
SENSOR_OUTPUT
WS_LEFT
WS_FRONT
IR_DETECTION
LINE_DETECTION
PROGRAMMER_INPUT
R1
4K7
MCLR
S1
IC1
PIC16F887
VDD
VSS
RE3/MCLR/VPP
RA0/AN0
RA1/AN1
RA2/AN2
RA3/AN3
RA4/T0CKI
RA5/AN4
RE0/AN5
RE1/AN6
RE2/AN7
RA7/OSC1/CLKIN
RA6/OSC2/CLKOUT
RC0/T1OSO
RC1/T1OSI/CCP2
RC2/CCP1
RC3/SCK
RD0
RD1
PGD/RB7
PGC/RB6
AN13/RB5
AN11/RB4
PGM/AN9/RB3
AN8/RB2
AN10/RB1
INT/AN12/RB0
RD7
RD6
RD5
RD4
RX/RC7
TX/RC6
SDO/RC5
SDI/RC4
RD3
RD2
PGD
PGC
LCD_OUTPUT
VSS
VDD
VEE
RS
RW
EN
D0
D1
D2
D3
D4
D5
D6
D7
FAN_MOTOR
+12V
D1
1N4148
EM
R10
1K
Q1
TIP120
IC3
L293D
R2
10K
1-2EN
1A
1Y
GND1
GND2
2Y
2A
VCC2
VCC1
4A
4Y
GND3
GND4
3Y
3A
3-4EN
LME
LMB
RME
RMB
LEFT_MOTOR
RIGHT_MOTOR
R8
10K
C6
1000uF
PLACE C6 CLOSE TO IC3
TITLE: Fighfighter General Schematic
Document Number:
REV:

Tips and Suggestions

By previous students

The following quotes are from previous students:

"Overall, the firefighter robot, although daunting at first, was not a difficult project to complete. By efficiently using our time, making good use of print and Internet resources, and consulting with students, the project became far more manageable and far less problematic. However, we could rarely have said that the project had become "easy" at any point, there were still times during which careful troubleshooting and analysis were required to solve issues. The entire project can be grouped into sections of work, each entailing its own challenges and involving different forms of troubleshooting."

"Designing and building the firefighter robot, as part of the Computer Engineering course, has been interesting: high fives were given, tears had been shed, and many hours were spent trying to get a robot to efficiently navigate a maze and extinguish a flame. The most important outcomes from this project is the significant amount of knowledge, the hands-on-experiments, and the daily experiences. The final product not only displays days upon days of hard work, but also days upon days of thinking, planning, and learning."

"I liked that this time around, Webb didn't really help us too much when were having troubles with the robot. He told us at the beginning of the course that he would limit the amount of help he gave which he indeed did. This was really beneficial as it helped us improve our troubleshooting. Whereas Webb would previously tell us that we were missing a capacitor, we would now have to spend a while looking for the mistake ourselves. We quickly got the hang of things, so by the end of the semester, most problems we encountered would be fixed pretty quickly. I also liked the fact that Webb was blunt with everyone and told them that, if they were going to slack off, they would have no hopes of finishing the project. Finally, I also liked that we had to breadboard our deadlines before we could make them as PCBs. It meant that we got familiar with the circuitry that we would be working with, and it helped us iron out mistakes that we made. Had we not had this, we probably would have made as many motherboards as Bob and Bill did."

Duotang:

- There is a lot of information in the duotang that is valuable except that it can be hard to find
- Don't forget about the fan in your circuit board (hint : TIP 120)
- Use the list of defines in your program
- Read about the analog to digital conversions

- Most basic programming can be found in the duotang. For a more complex code, see the MicroCode Studio help section

General Tips:

- Don't waste time in class
- Finish the breadboarding labs early so you can get a head start on the firefighter. You will appreciate the extra time later on.
- Although four months is given to complete the robot, if the time is not managed well it is going to be a scramble to the finish line. Indeed, managing your time throughout the project is essential.
- Take the time to properly select your group member.
- Simplicity is a key factor in a good design. Take the time to properly design and consider all aspects of the robot.
- Do your best to stay with the deadlines. Don't fall behind.
- Split the tasks between partners whenever possible.
- Buy your materials ahead of time.
- Think about using terminal blocks vs snappable headers.
- Use a ribbon cable to connect the robot to a power supply. You shouldn't use batteries because power supplies allow for all kinds of voltages and keep the flow smooth and constant.
- Know the uses for ribbon cables vs. snappable headers.
- Getting a good powerful fan can make a huge difference.
- Think about multiple circuit boards vs. one circuit board.
- Computer fans disperse air, instead of focusing it.
- When designing boards, consider the size of bot desired.
- Think about the options for a third wheel/balancing point.
- Use red wires for positive, black for negative, and blue for control.
- Braiding wires results in a much cleaner bot.
- Have the wheels farther apart. We saw other robots with increased distance between the motors travel in a straighter line.
- Don't waste time! We had relaxed a bit and fallen behind.

Finally, make sure to read the duotang that Webb provides you, and then refer back to specific sections as you encounter them later on in the project. It provides valuable information and can help you avoid redesigning your robot as well as your circuitry.

Getting Started

The most common problems with getting PIC microcontrollers running involved making sure the few external components are of the appropriate value and properly connected to the PIC. Following are some hints to help get things up and running.

Make sure the MCLR pin is connected to 5 V either through some kind of voltage protected reset circuit or simply with a 4.7k resistor. If you leave the pin unconnected, its level floats around and sometimes the PIC will work but usually it won't. The PIC has an on-chip power-on-reset circuit so in general just an external pull-up resistor is adequate. But in some cases the PIC may not power up properly and an external circuit may be necessary.

Make sure your power supply is up to the task. While the PIC itself consumes very little power, the power supply must be filtered fairly well. If the PIC is controlling devices that pull a lot of current from your power supply, as they turn on and off they can put enough of a glitch on the supply lines to cause the PIC to stop working properly. Even an LED display can create enough of an instantaneous drain to momentarily clobber a small power supply (like a 9 V battery) and cause the PIC to lose its mind.

Start small. Write short programs to test features you are unsure of or might be having trouble with. Once these smaller programs are working properly you can build on them.

Try doing things a different way. Sometimes what you are trying to do looks like it should work but doesn't, no matter how hard you pound on it. Usually there is more than one way to skin a program. Try approaching the problem from a different angle and maybe enlightenment will ensue.

Try not to use too many **GOTO**s. While **GOTO**s may be a necessary evil try to minimize their use as much as possible. Try to write your code in logical sections and not jump around too much. **GOSUB**s can be helpful in achieving this.

Firefighter Rules and Regulations

There are many rules and regulations to the Firefighter competition. A complete set of rules can be found with the teacher. Listed below are the main rules that you will need to be aware of :

1. All robots must fit in a Bounding Box with a base 31 x 31 cm square and 27 cm high.

2. The candle flame will be from 15 cm to 20 cm above the nominal floor level.

3. Your robot cannot touch the candle or its base during the run.

4. The robot must, in the opinion of the Judges, have found a candle before it attempts to put it out.

5. The robot must not use any destructive or dangerous methods to put out the candle.

6. The robot must come within 30 cm of the candle be-fore it extinguishes the flame.

Great Cow Basic

Before you begin to program your PIC chip you should know that an increasingly popular alternative to the PICBasic Pro language that we use in class is Great Cow Basic. This is an open-source programming language that you can download from the Internet. Many students before you have used this program and have spoken very positively about it.

For those of you that are looking at a career in a field related to programming you may want to consider Great Cow, if not now, at some point.

Some of the advantages are:

- You can download to your laptop and use it for your firefighter project
- Excellent support on the website with a great forum
- Many tutorials available
- It's a form of Basic so it is very similar to what we already use
- IT IS FREE !!

The big advantage to Great Cow is that when you leave this classroom you will be able to continue programming PICs whether at home or post secondary education.

Firefighter Robot
Final Report

1. The final report should include a cover page. This page should include the date of submission in the upper right corner, the page number in the lower right, your product name, team members, and the teacher's name.

2. Each group member will include a written reflection that pertains to anything they have learned during their years in tech., or a specific project, or tips to future gr 12 students in the course, or feedback regarding what you liked in the course, and it can even include feedback regarding what you feel we can do to improve the course. This part is quite open, just keep it constructive. This reflection will be a minimum of 1.5 pages, at a size 12 font, spacing at the default of 1.15, and either Times Roman Numeral or Aerial.

3. You will include pictures of your final firefighter. There must be at least 5 pictures that clearly show all sides of the bot.

4. The final program you used for your firefighter must be printed out and included. Your program should be well commented so any person reading your program will understand the logic.

5. All circuit boards will be printed out and included in your final report. The circuit boards should have all parts labelled.

The due date is the last day of regularly scheduled classes at the end of the semester.

PICBASIC PRO Basics

Variables

Variables are where temporary data is stored in a PICBASIC PRO program. They are created using the VAR keyword. Variables may be bit, byte, and word sized. Space for each variable is automatically allocated in the microcontroller's RAM.

Label **VAR** *Size*

Label is any unique identifier, excluding keywords in PICBASIC PRO. *Size* is **BIT, BYTE,** and **WORD.**

Some examples of creating variables are:

> Dog **VAR Bit**
> Cat **VAR BYTE**
> Bird **VAR WORD**

A **BIT** is 1 bit, and number are 0 or 1.
A **BYTE** is 8 bits, and numbers are 0 to 255.
A **WORD** is 16 bits and numbers are 0 to 65,535.

Constants

Named constants may be created in a similar manner to variables. It may be more convenient to use a name for a constant instead of using a constant number. If the number needs to be changed, it may be changed in only one place in the program; where the constant is defined. Variable data cannot be stored in a constant.

Label **CON** *Constant expression*

PICBASIC PRO Statements

ADCIN

ADCIN, *Channel, VAR*

Read the on-chip analog to digital converter *Channel*, and store the result in *Var.* While the ADC registers can be accessed directly, **ADCIN** makes the process a little easier.

Before **ADCIN** can be used, the appropriate TRIS register must be set to make the desired pins inputs. The **ADCON,** and/or **ANSEL** registers must also be set to assign the desired pins to analog inputs and in some cases to set the result format and clock source (set the clock source the same as the **DEFINE** specified for it). See the microchip data sheets for more information on these registers and things like the clock source and how to set them for the specified device.

Depending on the device, it may have an 8, 10, or 12 bit ADC. For many PIC Micros, the high bit of **ADCON0** or **ADCON1** controls whether the result is left or right justified. In most cases, 8-bit results should be right justified (**ADCON1=1**).

Several **DEFINES** may be used. The defaults are shown below:

```
DEFINE ADC_BITS 8          'Set number of bits in result (8, 10, or 12)
DEFINE ADC_CLOCK 3         'Set clock source (rc=3)
DEFINE ADC_SAMPLEUS 50     'Set sample time in microseconds
```

ADC_SAMPLEUS is the number of microseconds the program waits between setting the *Channel* and starting the analog to digital conversion. This is the sampling time.

```
TRISA=%11111111            ' Set PORTA to all input
ADCON1=0                   ' PORTA is analog
ADCIN 0, LW                ' Read channel 0 to LW
```

+++++++ **Webb's note** +++++++ When reading the **ADCIN** command students can sometimes be confused with the channel. The channel refers to the AN number on your chip. As an example, **ADCIN** 0, lw refers to the analog to digital bit 0, This is seen as AN0 on your chip diagram. On the 16F887 you will see this as pin 2. The result of this would then be stored in the variable you created, in this case — lw. There are many AN bits on the 16F887, it does not refer to a specific port. One of the easiest, and most efficient ways to use the chip would be to use A and E ports for your analog to digital conversion needs i.e. wall and flame detections. This is because you need to set distances with these. The other consideration you need to pay attention to is the **ADCON** set up. The datasheet is included in this duotang, and is used to set the A and E ports for digital or analog, YOU WANT IT IN ANALOG!!!!!

attention to is the **ADCON** set up. The datasheet is included in this duotang, and is used to set the A and E ports for digital or analog, YOU WANT IT IN ANALOG!!!!!
Use this method for counting lines in the maze:

LINE DETECTION

X = X + 1 'This is extremely useful for counting lines. You set the variable up as X, then it adds 1 to the count each time it executes that part of your program.

Example:

```
X var byte

    main:
    If portc.0 = 0 then gosub line
    goto main

    line:
    x=x+1
    Pause 20
    If x=6 then goto room 4
    Return
```

FOR..NEXT

FOR *Count* = *Start* **TO** *End*

NEXT *Count*

The **FOR..NEXT** loop allows programs to execute a number of statements some number of times using a variable as a counter. Due to its complexity and versatility, FOR..NEXT is best described step by step:

1) The value of *Start* is assigned to the index variable, *Count*. *Count* can be a variable of any type.

2) The *Body* is executed. The *Body* is optional and can be omitted (perhaps for a delay loop).

If the loop needs to *Count* to more than 255, a word variable must be used.

```
FOR i = 1 TO 10                'Count from 1 to 10
NEXT i                         'Go back to and do next count

FOR B2 = 20 TO 10 STEP -2      'Count from 20 to 10 by 2
NEXT B2                        'Go back to and do next count
```

+++++++ Webb's note +++++++ This command is great for creating a sweeping motion for your blower when you have found the flame.

```
For y = 1 to 10          'set count to 10
portb.7=1                'blower on
Pause 200
portb=%00000010          'rotate left
Pause 100
Next y
```

You can then reverse this action to sweep back right.

GOSUB

GOSUB *Label*

Jump to the subroutine at *Label* saving its return address on the stack. Unlike **GOTO,** when a **RETURN** statement is reached after executing a **GOSUB,** execution resumes with the statement following that last executed **GOSUB** statement.

An unlimited number of subroutines may be used in a program. Subroutines may also be nested. In other words, it is possible for a subroutine to **GOSUB** to another subroutine. Such subroutine nesting must be restricted to no more than four nested levels for 12 and 14 bit core devices.

```
GOSUB Beep               'Execute subroutine named Beep

Beep:
PORTB=%00000100          'Turn on B2
PAUSE 1000               'Hold for 1 second
RETURN                   'Go back to main routine that called Beep
```

GOTO

GOTO *Label*

Program execution continues with the statements at *Label.*

```
GOTO send                  'Jump to statement labelled send.

send:
PORTB=%00000001            'Turn on B1
```

HIGH

HIGH *Pin*

Make the specified *Pin* high. *Pin* is automatically made an output. *Pin* may be a constant, 0 -15, or a variable that contains a number 0 - 15 (e.g B0) or a pin name (e.g. PORTA.0)

```
        HIGH 0             'Make Pin 0 an output and set it high

        HIGH PORTA.0       'Make PORTA.0, PIN 0 an output and set it high

Led Var PORTB.0            'Define Led pin
        HIGH Led           'Make Led pin an output and set it high
```

Alternatively, if the pin is already an output, a much quicker and shorter way (from a generated code standpoint) to set it high would be:

```
PORTB.0 = 1                'Set PORTB pin 0 high
```

IF..THEN

IF *Comp* (**AND/OR** *Comp...*) **THEN** *Statement*
IF *Comp* (**AND/OR** *Comp...*) **THEN** *Label*
ELSIF *Comp* (**AND/OR** *Comp...* **THEN**
Statement...

ENDIF

Performs one or more comparisons. Each *Comp* term can relate a variable to a constant or other variable and includes one of the comparison operators listed previously.

IF..THEN evaluates the comparison terms for true or false. If it evaluates the comparison terms for true or false. It it evaluates to rue, the operation after the **THEN** is executed. If it evaluates to false, the operation after the **THEN** is not executed. Comparisons that evaluate to 0 are considered false. Any other value is considered true.

LCDOUT

LCDOUT *Item* (Item...)

Display *Item* on an intelligent Liquid Crystal Display. PBP supports LCD modules with a Hitachi 44780 controller or equivalent. These LCDs usually have a 14 or 16 pin single or double row header at one edge.

If a pound sign (#) precedes an *Item*, the ASCII representation for each digit is sent to the LCD.

A program should wait for up to half a second before sending the first command to an LCD. It can take quite a while for an LCD to start up.

The LCD is initialized the first time any character or command is sent to it using LCDOUT. If it is powered down and then powered back up for some reason during operation, an internal flag can be reset to tell the program to reinitialize it the next time it uses LCDOUT.

FLAGS=0

Commands are sent to the LCD by sending a $FE followed by the command. Some useful commands are listed in the following table:

Command	Operation
$FE, 1	Clear display
$FE, 2	Return home
$FE, $0C	Cursor off

$FE, $0E	**Underline cursor on**
$FE, $0F	**Blinking cursor on**
$FE, $10	**Move cursor left one position**
$FE, $14	**Move cursor right one position**
$FE, $80	**Move cursor to beginning of first line**
$FE, $C0	**Move cursor to beginning of second line**
$FE, $94	**Move cursor to beginning of third line**
$FE, $D4	**Move cursor to beginning of fourth line**

Note that there are commands to move the cursor to the beginning of the different lines of a multi-line display. For most LCDs, the displayed characters and lines are not consecutive in display memory, there can be a break in between locations. For most 16 x 2 displays, the first line starts at $80 and the second line starts at $C0. The command:

LCDOUT $FE, $80 + 4

sets the display to start writing characters at the fourth position of the first line. 16 x 1 displays are usually formatted as 8 x 2 displays with a break between the memory locations for the first and second 8 characters. 4 line displays also have a mixed up memory map, as shown in the table above.

```
LCDOUT $FE, 1, "Hello"        'Clear display and show "Hello"

LCDOUT $FE, $C0, "World"      'Jump to second line and show "World"

LCDOUT $FE, 1, #wall          'Display the numerical value of the wall variable
```

The LCD may be connected to the PIC microcontroller using either a 4-bit bus or an 8-bit bus. All bits must be on one port.

PBP (PICBasic Pro) assumes the LCD is connected to specific pins unless told otherwise using DEFINES. It assumes the LCD will be used with a 4-bit bus with data lines DB4-DB7, connected to PIC MCU (micro) PORTA.0 - PORTA.3, Register Select to PORTA.4 and Enable to PORTB.3.

It is also preset to initialize the LCD to a 2 line display.

To change this setup, place one or more of the following **DEFINEs**, all in upper-case at the top of your PICBASIC PRO. Program.

```
DEFINE LCD_DREG PORTA          'Set LCD Data port
DEFINE LCD_DBIT 0       'Set starting Data bit (0 or 4) if 4-bit bus
DEFINE LCD_RSREG PORTA         'Set LCD Register Select port
DEFINE LCD_RSBIT 4             'Set LCD_RSBIt 4
DEFINE LCD_EREG PORTB          'Set LCD Enable port
DEFINE LCD_EBIT 3       'Set LCD Enable bit
DEFINE LCD_RWREG PORTE         'Set LCD Read/Write port
DEFINE LCD_RWBIT 2             'Set LCD Read/Write bit
DEFINE LCD_BITS 4       'Set LCD bus size (4 or 8 bits)
DEFINE LCD_LINES 2             'Set number of lines on LCD
DEFINE LCD_COMMANDUS 1500 'Set command delay time in µs
DEFINE LCD_DATAUS 44           'Set data delay time in µs
```

+++++++ Webb's note +++++++ Many students get confused with the LCD defines. They misunderstand the defines to think that they are laid out for them to copy, THEY ARE NOT!!!!! You must understand what each define does and to what Port and bit# you have wired it to. The defines listed above are examples and likely will not correspond to your specific wiring.

PAUSE

PAUSE *Period*

Pause the program for *Period* milliseconds. *Period* is 16 bits using PBP, so delays can be up to 65,535 milliseconds (a little over a minute).

PAUSE assumes an oscillator frequency of 4Mhz. If an oscillator other than 4Mhz is used, PBP must be told using a **DEFINE OSC** command.

```
PAUSE 1000            'Delay for 1 second
```

PAUSEUS

PAUSEUS *Period*

Pause the program for *Period* microseconds. *Period* is 16 bit, so delays can be up to 65,535 microseconds. Unlike the other delay functions, **PAUSEUS** doesn't put the microcontroller into low power mode. Thus, **PAUSEUS** consumes more power but is also much more accurate.

Because **PAUSEUS** takes a minimum number of cycles to operate, depending on the frequency of the oscillator, delays of less than a minimum number of microseconds are not possible using **PAUSEUS**.

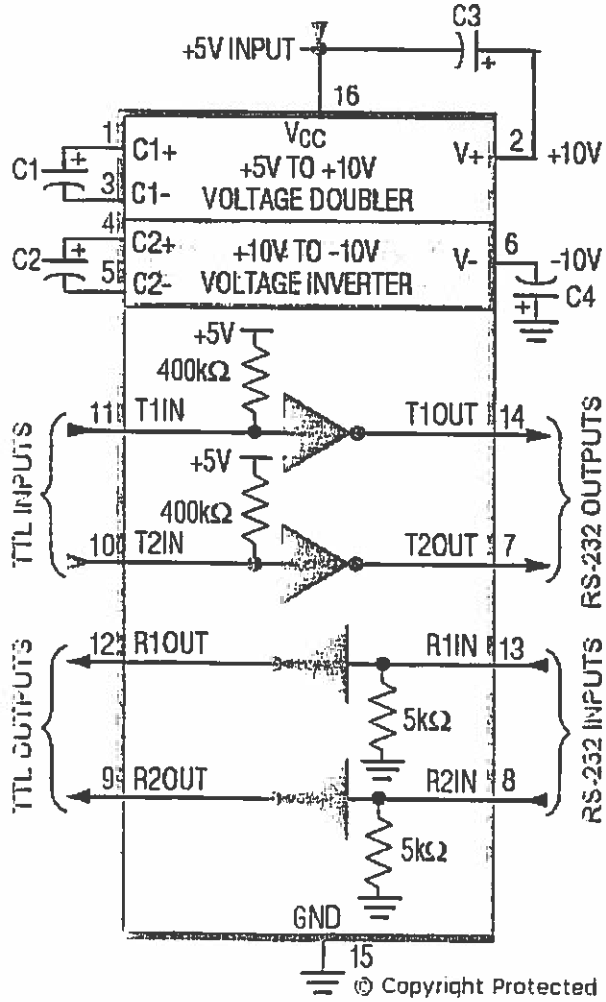
OSC	Minimum Delay
3	20 us
4	24 us
8	12 us
10	8 us
12	7 us
16	5 us
20	3 us
24	3 us
25, 32, 33	2 us

Pauseus assumes an oscillator frequency of 4 MHZ. If an oscillator other than 4 MHZ is used, PBP must be told using a **DEFINE** OSC command.

```
PAUSEUS 1000          ' Delay for 1 millisecond
```

Max 232 Pins/Voltages

Pin 1	=	7.6	→	6.9 volts
2	=	9.5	→	8.9 v
3	=	2.8	→	2.4 v
4	=	3.9	→	4.5 v
5	=	-5	→	-4 v
6	=	-9	→	-8 v
7	=	-9	→	-8 v
8	=	0 v		
9	=	5 v		
10	=	3.6 v		
11	=	5 v		
12	=	5 v		
13	=	-10 v		
14	=	-8.5	→	-7.2 v
15	=	0 v		(common ground)
16	=	5 v		



Chapter 4. General Setup

4.1 INTRODUCTION

How to get started using the PICkit 3 programmer/debugger is discussed.

- Starting the MPLAB IDE Software
- Creating a Project
- Viewing the Project
- Building the Project
- Setting Configuration Bits
- Setting the Debugger or Programmer
- Debugger/Programmer Limitations

4.2 STARTING THE MPLAB IDE SOFTWARE

After installing the MPLAB IDE software (**Section 3.2 "Installing the Software"**), invoke it by using any of these methods:

- Select Start>Programs>Microchip>MPLAB IDE vx.xx>MPLAB IDE, where vx.xx is the version number.
- Double click the MPLAB IDE desktop icon.
- Execute the file `mplab.exe` in the `mplab ide\core` subdirectory of the MPLAB IDE installation directory.

For more information on using the software, see:

- *"MPLAB IDE User's Guide"* (DS51519) – Comprehensive guide for using MPLAB IDE.
- The on-line help files – The most up-to-date information on MPLAB IDE and PICkit 3 programmer/debugger.
- Readme files – Last minute information on each release is included in `Readme for MPLAB IDE.txt` and `Readme for PICkit 3 Debugger.txt`. Both files are found in the `Readmes` subdirectory of the MPLAB IDE installation directory.

4.3 CREATING A PROJECT

The easiest way to create a new project is to select Project>Project Wizard. With the help of the Project Wizard, a new project and the language tools for building that project can be created. The wizard will guide you through the process of adding source files, libraries, etc., to the various "nodes" on the project window. See MPLAB IDE documentation for more detail on using this wizard. The basic steps are provided here:

- Select your device (e.g., PIC18F45K20)
- Select a language toolsuite (e.g., Microchip C Compiler Toolsuite)
- Name the project
- Add application files (e.g., `program.c`, `support.s`, `counter.asm`)

Note:	If you do not have a custom linker script in your project, the Project Manager will select the appropriate linker script for you.

4.4 VIEWING THE PROJECT

After the Project Wizard has created a project, the project and its associated files are visible in the Project window. Right click on any line in the project window tree to pop up a menu with additional options for adding and removing files.

See MPLAB IDE documentation for more detail on using the Project window.

4.5 BUILDING THE PROJECT

After the project is created, the application needs to be built. This will create object (hex) code for the application that can be programmed into the target by the PICkit 3 programmer/debugger.

To set build options, select Project>Build Options>Project.

> **Note:** On the Project Manager toolbar (View>Toolbars>Project Manager), select "Debug" from the drop-down list when using the PICkit 3 as a debugger, or select "Release" when using it as a programmer.

When done, choose Project>Build All to build the project.

4.6 SETTING CONFIGURATION BITS

Although device Configuration bits may be set in code, they also may be set in the MPLAB IDE Configuration window. Select Configure>Configuration Bits. By clicking on the text in the "Settings" column, these can be changed.

Some Configuration bits of interest are:

- **Watchdog Timer Enable** – On most devices, the Watchdog Timer is enabled initially. It is usually a good idea to disable this bit.
- **Comm Channel Select** – For some devices, you will need to select the communications channel for the device, e.g., PGC1/EMUC1 and PGD1/EMUD1. Make sure the pins selected here are the same ones physically connected to the device.
- **Oscillator** – Select the configuration setting that matches the target oscillator.

4.7 SETTING THE DEBUGGER OR PROGRAMMER

Select Debugger>Select Tool>PICkit 3 to choose the PICkit 3 programmer/debugger as the debug tool. The Debugger menu and MPLAB IDE toolbar will change to display debug options once the tool is selected. Also, the Output window will open and messages concerning PICkit 3 status and communications will be displayed on the **PICkit 3** tab. For more information, see **Section 9.2 "Debugging Functions"** and **Section 9.3 "Debugging Dialogs/Windows"**.

Select Programmer>Select Programmer>PICkit 3 to choose the PICkit 3 programmer/debugger as the programmer tool. The Programmer menu and MPLAB IDE toolbar will change to display programmer options once the tool is selected. Also, the Output window will open and messages concerning ICE status and communications will be displayed on the **PICkit 3** tab. For more information, see **Section 9.4 "Programming Functions"**.

Select Debugger>Settings or Programmer>Settings to open the Settings dialog (**Section 9.5 "Settings Dialog"**) and set up options as needed.

If errors occurs, see:

- **Chapter 8. "Error Messages"**
- **Chapter 7. "Frequently Asked Questions (FAQs)"**

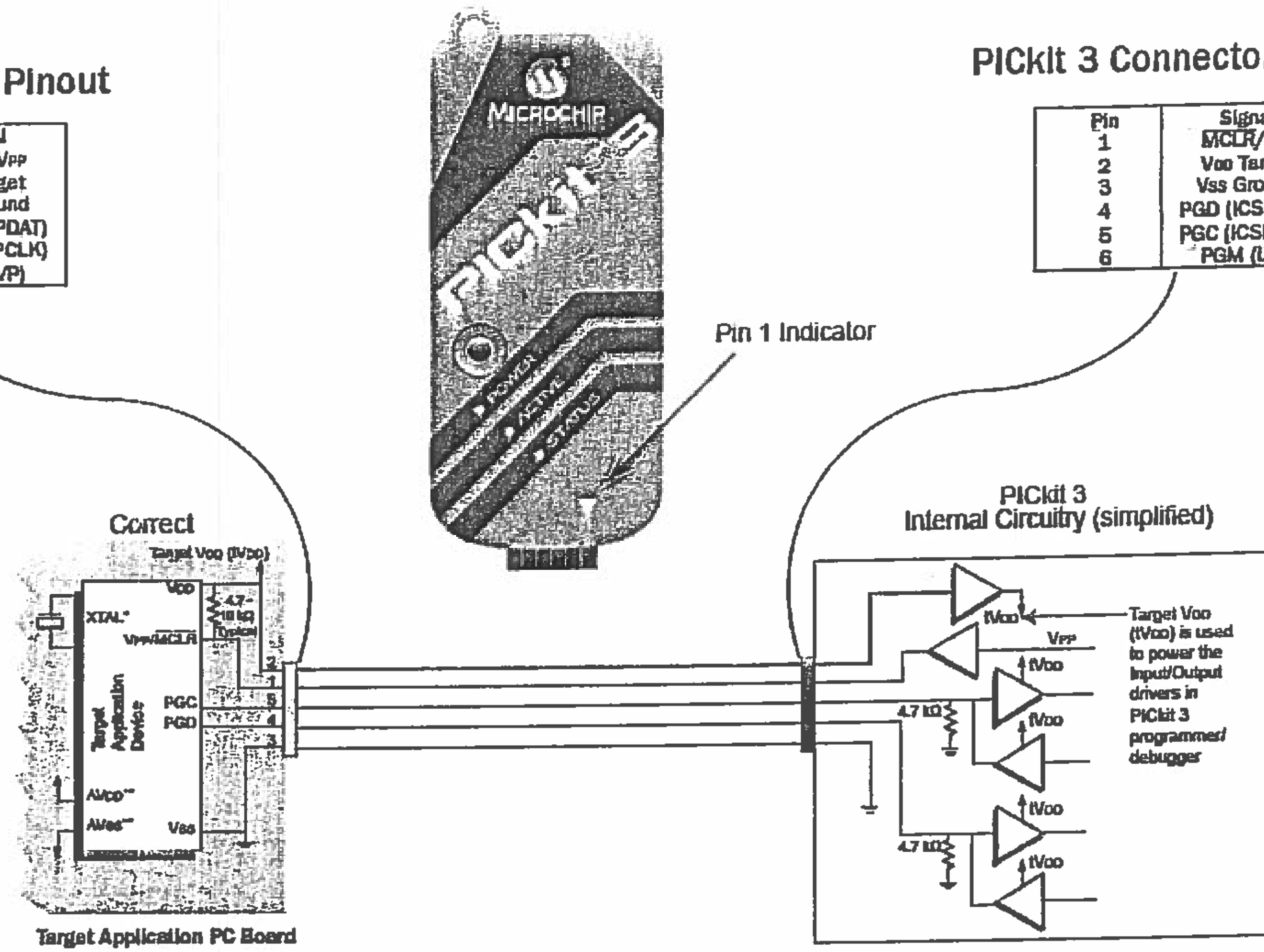
Circuitry and Connector Pinouts

Target Connector Pinout

Pin	Signal
1	$\overline{MCLR}$/V_{PP}
2	V_{DD} Target
3	V_{SS} Ground
4	PGD (ICSPDAT)
5	PGC (ICSPCLK)
6	PGM (LVP)

PICkit 3 Connector Pinout

Pin	Signal
1	$\overline{MCLR}$/V_{PP}
2	V_{DD} Target
3	V_{SS} Ground
4	PGD (ICSPDAT)
5	PGC (ICSPCLK)
6	PGM (LVP)



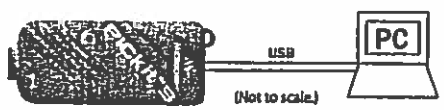
USING PICkit™ 3 IN-CIRCUIT DEBUGGER

1 Install the Latest Software

Install the MPLAB® IDE software onto your PC using the MPLAB IDE CD-ROM or download the software from the MPLAB IDE page of the Microchip web site (www.microchip.com/MPLAB). Check the latest Release Notes for additional information.

2 Configure PC USB Communications

Connect the PICkit™ 3 development programmer/debugger to a PC USB port via a USB cable. PICkit 3 uses the standard HID USB Windows® driver.
Note: If a USB hub is used, the hub must be powered with its own power supply.



3 Build Your Project

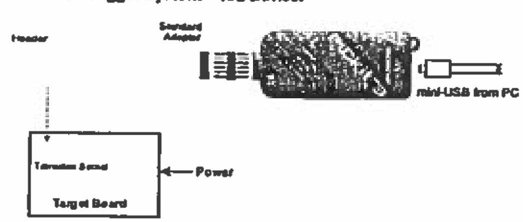
1. Launch MPLAB IDE.
2. Load your project or use the Project Wizard to create a new one.
4. Build your project based on your configurations and options.
5. Select the PICkit 3 as either a debugger (Debugger>Select Tool>PICkit 3) or as a programmer (Programmer>Select Programmer>PICkit 3).

4 Connect to Target and Power

1. Attach the PICkit 3 to the PC using the USB cable, if not already.
2. Attach the communications cable between the debugger and target board.
3. Connect power to the target board.

Typical Debugger System – Device With On-Board ICE Circuitry:

Alternate Debugger System – ICE Device:



5 Program and Debug

1. Program your device.
2. As a programmer, PICkit 3 will automatically run your code. As a debugger, you can run, halt, single step and set breakpoints in your code.

Note: For information on Reserved Resouces used by the debugger, see the PICkit 3 on-line help.

ADDITIONAL INFORMATION

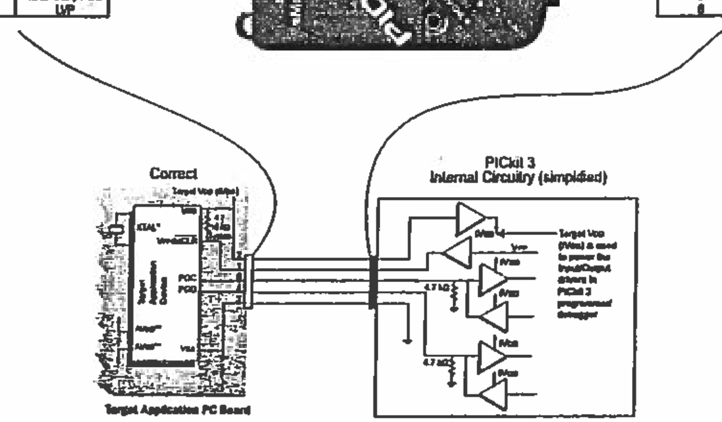
Circuitry and Connector Pinouts

Target Connector Pinout

Pin	Signal
1	MCLR/Vpp
2	VDD Target
3	VSS Ground
4	ICSPDAT/PGD
5	ICSPCLK/PGC
6	LVP

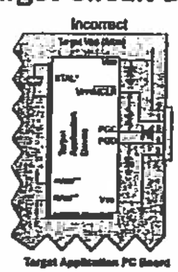
PICkit 3 Connector Pinout

Pin	Signal
1	MCLR/Vpp
2	VDD Target
3	VSS Ground
4	ICSPDAT/PGD
5	ICSPCLK/PGC
6	LVP



*Target device must be running with an oscillator for the debugger to function as a debugger.
**If the device has AVDD and AVSS lines, they must be connected for the debugger to operate.

Target Circuit Design Precautions



- Do not use multiplexing on PGC/PGD – they are dedicated for communications to PICkit 3.
- Do not use pull-ups on PGC/PGD – they will divide the voltage levels since these lines have 4.7 kΩ pull-down resistors in PICkit 3.
- Do not use capacitors on PGC/PGD – they will prevent fast transitions on data and clock lines during programming and debug communications.
- Do not use capacitors on MCLR – they will prevent fast transitions of VPP.
- Do not use diodes on PGC/PGD – they will prevent bidirectional communication between PICkit 3 and the target PIC® MCU.

Recommended Settings

COMPONENT	SETTING
Oscillator	• OSC bits set properly • Running
Power	Supplied by target
WDT	Disabled (device dependent)
Code Protect	Disabled
Table Read Protect	Disabled
LVP	Disabled
BOD	VDD > BOD VDD min
JTAG	Disabled
AVDD and AVSS	Must be connected
PGCx/PGDx	Proper channel selected, if applicable
Programming	VDD voltage levels meet programming specs

Note: See the PICkit 3 User's Guide for more component and setting information.

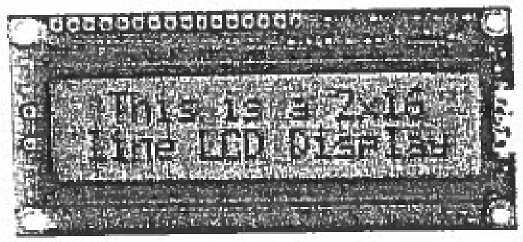
www.microchip.com

MICROCHIP

DS51792A

Driving An LCD

A **liquid-crystal display (LCD)** is a flat panel display, electronic visual display, or video display that uses the light modulating properties of liquid crystals.



LCD modules come in various configurations, but 99% of them use the same interface chip, the Hitachi 4870 LCD character driver. These screens come in a variety of configurations including 8 x 1, which is one row of eight characters, 16 x 2 and 20 x 4. LCD screens are limited to text only and are often used in copiers, fax machines, laser printers, industrial test equipment, networking equipment such as routers and storage devices.

The LCD can be controlled in an 8-bit mode or a 4-bit mode, which requires eight I/O or four I/O, respectively. Most people want to save I/O so they use the 4-bit mode.

A typical pin configuration for an LCD would be as follows:

1. VSS Ground
2. VCC or VDD (+ 3.3V to + 5V)
3. VO Contrast Adjustment – brightness of the text on screen (grounding it sets it to maximum contrast, using a variable resistor allows you to adjust the contrast)
4. RS Register Select – RS=0: Command, RS = 1: Data (connected to the PIC, used to tell the LCD if a character or LCD command is coming from the PIC) 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.
5. R/W Read/Write R/W=0: Write only mode, R/W=1: Data. This line is pulled low in order to write commands or character data to the module, or pulled high to read character data or status information from its registers.
6. E Clock (Enable) – This input is used to initiate the actual transfer of commands or character data between the module and the data lines.
7. DB0 Bit 0 – not used in 4 bit operation

8. DB1 Bit 1 – not used in 4 bit operation
9. DB2 Bit 2 – not used in 4 bit operation
10. DB3 Bit3 – not used in 4 bit operation
11. DB4 Bit 4
12. DB5 Bit 5
13. DB6 Bit 6
14. DB& Bit 7
15. BLA Backlight Anode (+)
16 BLK Backlight Cathode (-)

Wall and Flame Detection

Analog-to-Digital Conversion

The firefighter uses one of the most useful features of the PIC16F887, the analog-to-digital (A/D) converter.

Almost everything in the real world is not digital but instead analog. To control something in the real world, or to understand something in the PIC, we have to convert that real-world analog data into the digital for the PIC understands. That is done with an A/D converter. For example, if you have to read a temperature, or light levels (Infra Red), you will need both a sensor (Sharp GP2D12 or IR phototransistor) to convert the measurement into a variable voltage, and an A/D converter to change the resulting voltage into a digital value.

An A/D register's digital output will have a resolution to it. That means it can output an 8-bit digital value, 10-bit digital value, or larger if required. The PIC 16F887 has a 10 bit resolution A/D register, but can also operate as an 8-bit. We will use it as an 8-bit since it's a little easier to understand. Eight bits fit into one byte, and that's much easier to manipulate in code.

For the example of code we will assume that a wall detection sensor is wired in RA2 of the PIC.

We start off with the **DEFINE** statements required by PBPro. The first is to set the output result to eight bits. Then we set the clock source to RC, and then finally we add a sample time that sets when we check the status of the A/D conversion.

We then set our TRIS statements, porta will be inputs, and portb will be outputs
Next we establish the variable we will use to store the A/D result.
Then we can set up the **ADCON** register to make all inputs of Port A work with the A/D register rather than as digital I/O. You will need to check the PIC18F887 datasheet which can be found at www. microchip,com . As luck would have it, your teacher is a really good guy and has taken the time to sift through all the unwanted garble of almost 300 pages and included the 2 sheets you need.
Then we set up the **ADCIN** commands. Within this command we define which A/D port to read and where to put the result (adval).
After that we set port B to be all outputs and initialize the motors.

```
Define ADC_BITS 8                       'set number of bits in result
Define ADC_CLOCK 3                      'set clock source (3 = rc)
Define ADC_SAMPLEUS 50                  'set sampling time in uS (microseconds)

trisa=%11111111                         'set all A port to inputs
trisb=%00000000                         'set all B port to outputs

SW VAR byte                             'create the variable SW and make it 8 bits

Main:
ADCON1=0                                'set ports A & E to analog
ADCIN 0, SW                             'read on pin 2 (A0) and store in the variable SW

portb=%00000110                         'motor forward
If SW > 13 then portb=%00000010         'turn on right motor to correct left
If SW < 13 then portb=%00000100         'turn on left motor to correct right
If SW = 13 then portb=%00000110         'go forward
```

Another way of controlling the motors for wall hugging is to use **GOTO**'s, shown below.

```
main:
If SW > 13 then GOTO left
If SW < 13 then GOTO right
goto main

left:
portb=%00000001                         'jog left
Pause 20
goto main

right:
portb=%00001000                         'jog right
pause 20
goto main
```

Check for 887 sheets

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

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

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

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

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

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

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

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

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

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DONE	—	ADON
bit 7							bit 0

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

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

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

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

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

bit 2 **GO/DONE:** A/D Conversion Status bit

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

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

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

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

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

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

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

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

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

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

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

bit 5-4 **Unimplemented:** Read as '0'

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

PCFG <3:0>	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0	VREF+	VREF-	C/R
0000	A	A	A	A	A	A	A	A	VDD	VSS	8/0
0001	A	A	A	A	VREF+	A	A	A	AN3	VSS	7/1
0010	D	D	D	A	A	A	A	A	VDD	VSS	5/0
0011	D	D	D	A	VREF+	A	A	A	AN3	VSS	4/1
0100	D	D	D	D	A	D	A	A	VDD	VSS	3/0
0101	D	D	D	D	VREF+	D	A	A	AN3	VSS	2/1
011x	D	D	D	D	D	D	D	D	—	—	0/0
1000	A	A	A	A	VREF+	VREF-	A	A	AN3	AN2	6/2
1001	D	D	A	A	A	A	A	A	VDD	VSS	6/0
1010	D	D	A	A	VREF+	A	A	A	AN3	VSS	5/1
1011	D	D	A	A	VREF+	VREF-	A	A	AN3	AN2	4/2
1100	D	D	D	A	VREF+	VREF-	A	A	AN3	AN2	3/2
1101	D	D	D	D	VREF+	VREF-	A	A	AN3	AN2	2/2
1110	D	D	D	D	D	D	D	A	VDD	VSS	1/0
1111	D	D	D	D	VREF+	VREF-	D	A	AN3	AN2	1/2

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

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

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

* when doing your flame extinguisher it is important to understand that not all transistors are the same. Hint - current & pinou

2N3904

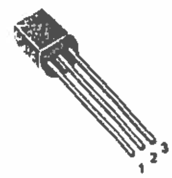
SMALL SIGNAL NPN TRANSISTOR

PRELIMINARY DATA

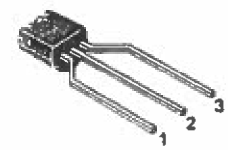
Ordering Code	Marking	Package / Shipment
2N3904	2N3904	TO-92 / Bulk
2N3904-AP	2N3904	TO-92 / Ammopack

- SILICON EPITAXIAL PLANAR NPN TRANSISTOR
- TO-92 PACKAGE SUITABLE FOR THROUGH-HOLE PCB ASSEMBLY
- THE PNP COMPLEMENTARY TYPE IS 2N3906

APPLICATIONS
- WELL SUITABLE FOR TV AND HOME APPLIANCE EQUIPMENT
- SMALL LOAD SWITCH TRANSISTOR WITH HIGH GAIN AND LOW SATURATION VOLTAGE

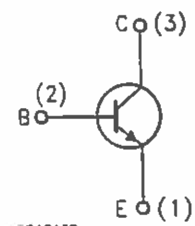


TO-92
Bulk



TO-92
Ammopack

INTERNAL SCHEMATIC DIAGRAM



DS10130

ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_{CBO}	Collector-Base Voltage ($I_E = 0$)	60	V
V_{CEO}	Collector-Emitter Voltage ($I_B = 0$)	40	V
V_{EBO}	Emitter-Base Voltage ($I_C = 0$)	6	V
I_C	Collector Current	200	mA
P_{tot}	Total Dissipation at $T_C = 25\ ^oC$	625	mW
T_{stg}	Storage Temperature	-65 to 150	^oC
T_j	Max. Operating Junction Temperature	150	^oC

MAXIMUM RATINGS

Rating	Symbol	TIP120, TIP125	TIP121, TIP126	TIP122, TIP127	Unit
Collector–Emitter Voltage	V_{CEO}	60	80	100	Vdc
Collector–Base Voltage	V_{CB}	60	80	100	Vdc
Emitter–Base Voltage	V_{EB}	5.0			Vdc
Collector Current – Continuous – Peak	I_C	5.0 8.0			Adc
Base Current	I_B	120			mAdc
Total Power Dissipation @ T_C = 25°C Derate above 25°C	P_D	65 0.52			W W/°C
Total Power Dissipation @ T_A = 25°C Derate above 25°C	P_D	2.0 0.016			W W/°C
Unclamped Inductive Load Energy (Note 1)	E	50			mJ
Operating and Storage Junction, Temperature Range	T_J, T_{stg}	–65 to +150			°C

THERMAL CHARACTERISTICS

Characteristic	Symbol	Max	Unit
Thermal Resistance, Junction–to–Case	$R_{\theta JC}$	1.92	°C/W
Thermal Resistance, Junction–to–Ambient	$R_{\theta JA}$	62.5	°C/W

Stresses exceeding those listed in the Maximum Ratings table may damage the device. If any of these limits are exceeded, device functionality should not be assumed, damage may occur and reliability may be affected.

1. I_C = 1 A, L = 100 mH, P.R.F. = 10 Hz, V_{CC} = 20 V, R_{BE} = 100 Ω

ELECTRICAL CHARACTERISTICS (T_C = 25°C unless otherwise noted)

Characteristic		Symbol	Min	Max	Unit
OFF CHARACTERISTICS					
Collector–Emitter Sustaining Voltage (Note 2) (I_C = 100 mAdc, I_B = 0)	TIP120, TIP125 TIP121, TIP126 TIP122, TIP127	$V_{CEO(sus)}$	60 80 100	– – –	Vdc
Collector Cutoff Current (V_{CE} = 30 Vdc, I_B = 0) (V_{CE} = 40 Vdc, I_B = 0) (V_{CE} = 50 Vdc, I_B = 0)	TIP120, TIP125 TIP121, TIP126 TIP122, TIP127	I_{CEO}	– – –	0.5 0.5 0.5	mAdc
Collector Cutoff Current (V_{CB} = 60 Vdc, I_E = 0) (V_{CB} = 80 Vdc, I_E = 0) (V_{CB} = 100 Vdc, I_E = 0)	TIP120, TIP125 TIP121, TIP126 TIP122, TIP127	I_{CBO}	– – –	0.2 0.2 0.2	mAdc
Emitter Cutoff Current (V_{BE} = 5.0 Vdc, I_C = 0)		I_{EBO}	–	2.0	mAdc
ON CHARACTERISTICS (Note 2)					
DC Current Gain (I_C = 0.5 Adc, V_{CE} = 3.0 Vdc) (I_C = 3.0 Adc, V_{CE} = 3.0 Vdc)		h_{FE}	1000 1000	– –	–
Collector–Emitter Saturation Voltage (I_C = 3.0 Adc, I_B = 12 mAdc) (I_C = 5.0 Adc, I_B = 20 mAdc)		$V_{CE(sat)}$	– –	2.0 4.0	Vdc
Base–Emitter On Voltage (I_C = 3.0 Adc, V_{CE} = 3.0 Vdc)		$V_{BE(on)}$	–	2.5	Vdc
DYNAMIC CHARACTERISTICS					
Small–Signal Current Gain (I_C = 3.0 Adc, V_{CE} = 4.0 Vdc, f = 1.0 MHz)		h_{fe}	4.0	–	–
Output Capacitance (V_{CB} = 10 Vdc, I_E = 0, f = 0.1 MHz	TIP125, TIP126, TIP127 TIP120, TIP121, TIP122	C_{ob}	– –	300 200	pF

Product parametric performance is indicated in the Electrical Characteristics for the listed test conditions, unless otherwise noted. Product performance may not be indicated by the Electrical Characteristics if operated under different conditions.

2. Pulse Test: Pulse Width ≤ 300 μs, Duty Cycle ≤ 2%

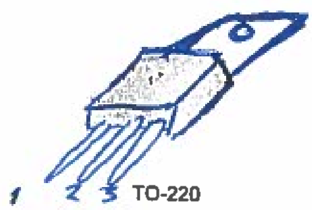
FAIRCHILD

November 2014

TIP120 / TIP121 / TIP122
NPN Epitaxial Darlington Transistor

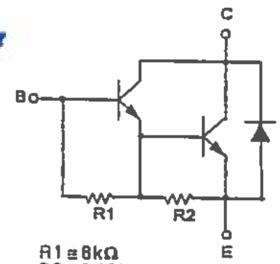
Features

- Medium Power Linear Switching Applications
- Complementary to TIP125 / TIP126 / TIP127



1.Base 2.Collector 3.Emitter

Equivalent Circuit



R1 ≅ 8kΩ
R2 ≅ 0.12kΩ

Ordering Information

Part Number	Top Mark	Package	Packing Method
TIP120	TIP120	TO-220 3L (Single Gauge)	Bulk
TIP120TU	TIP120	TO-220 3L (Single Gauge)	Rail
TIP121	TIP121	TO-220 3L (Single Gauge)	Bulk
TIP121TU	TIP121	TO-220 3L (Single Gauge)	Rail
TIP122	TIP122	TO-220 3L (Single Gauge)	Bulk
TIP122TU	TIP122	TO-220 3L (Single Gauge)	Rail

Absolute Maximum Ratings

Stresses exceeding the absolute maximum ratings may damage the device. The device may not function or be operable above the recommended operating conditions and stressing the parts to these levels is not recommended. In addition, extended exposure to stresses above the recommended operating conditions may affect device reliability. The absolute maximum ratings are stress ratings only. Values are at T_C = 25°C unless otherwise noted.

Symbol	Parameter		Value	Unit
V_{CBO}	Collector-Base Voltage	TIP120	60	V
		TIP121	80	
		TIP122	100	
V_{CEO}	Collector-Emitter Voltage	TIP120	60	V
		TIP121	80	
		TIP122	100	
V_{EBO}	Emitter-Base Voltage		5	V
I_C	Collector Current (DC)		5	A
I_{CP}	Collector Current (Pulse)		8	A
I_B	Base Current (DC)		120	mA
T_J	Junction Temperature		150	°C
T_{STG}	Storage Temperature Range		-65 to 150	°C

Linearizing The Sharp Sensor

As good as the Sharp GP2D12 analog sensor is for its' ability to sense objects it does have a challenge when using the output information.

The Sharp sensor has a non-linearizing voltage output and it is very helpful when we can put this output into a linear form. One approach that works well is to use sophisticated mathematics programs to generate a curve fit. The functions that such programs generate are quite good but usually require floating point math and a good math library in order to implement them. This isn't much help when using controllers that lack floating point capabilities.

Another good approach is to use a piecewise linear approximation to convert the output voltage to a range value. This involves breaking up the response into small straight lines and doing a separate approximation for each line. Straight-line approximations are simple to compute and can be implemented with fairly good accuracy by applying integer math. The disadvantage is they take up code space.

Ideally, it would be nice to have a single approximation function that works well with integer math, so, here it is:

$$R = (((6787/(V-3)))-4)/5$$

Example:

```
V var byte
swall var byte

sidewall:
ADCIN 0, V                       'read analog voltage value from port A0 and assign to V
swall = (((6787/(V-3)))-4)/5     "calculate distance and assign to variable swall
GOTO sidewall
```

PARALLAX INC

Web Site: www.parallax.com
Forums: forums.parallax.com
Sales: sales@parallax.com
Technical: support@parallax.com

Office: (916) 624-8333
Fax: (916) 624-8003
Sales: (888) 512-1024
Tech Support: (888) 997-8267

Sharp GP2D12 Analog Distance Sensor (#605-00003)

General Description

The Sharp GP2D12 is an analog distance sensor that uses infrared to detect an object between 10 cm and 80 cm away. The GP2D12 provides a non-linear voltage output in relation to the distance an object is from the sensor and interfaces easily using any analog to digital converter.

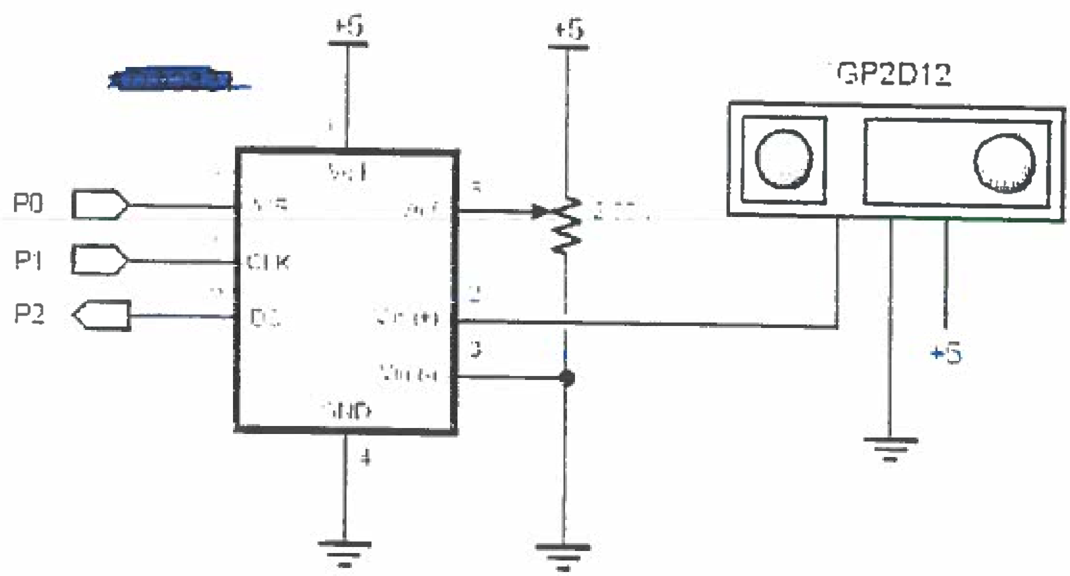
Features

- High immunity to ambient light and color of object
- No external control circuitry required
- Sensor includes convenient mounting holes
- Compatible with all BASIC Stamp® and SX microcontrollers

Application Ideas

- Robot range finder
- Halloween prop activation

Quick Start Circuit



Sensitivity

The usable range of the GP2D12 is between 10 cm and 80 cm. The readings for objects closer than 10 cm are unstable and therefore not usable.

Resources and Downloads

Check out the Sharp GP2D12 Analog Distance Sensor product page for example programs, the manufacturer datasheet and more:

http://www.parallax.com/detail.asp?product_id=605-00003

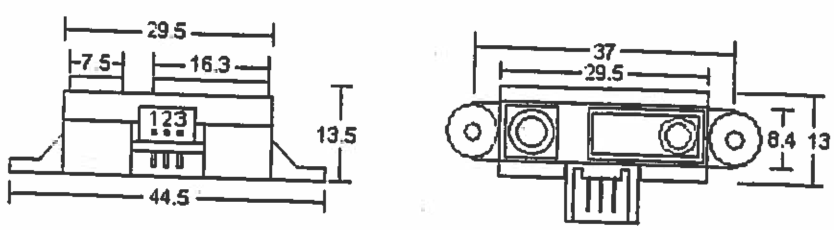
Specifications

Symbol	Quantity	Minimum	Typical	Maximum	Units
Vcc	Supply Voltage †	4.5	5.0	5.5	V
Topr	Operating Temperature †	-10	-	+60	°C
Tstg	Storage Temperature †	-40	-	+70	°C
ΔL	Distance Measuring Range †	10	-	80	cm
Vo	Output Terminal Voltage (L=80 cm) †	0.25	0.4	0.55	V
ΔVo	Output change at L=80 cm to 10 cm †	1.75	2.0	2.25	V
Icc	Average Dissipation Current (L=80cm) †	-	33	50	mA

† data obtained from Sharp's GP2D12 datasheet

Pin Definitions and Ratings

Pin	Name	Function
1	Vo	Voltage Output
2	GND	Ground
3	Vcc	Supply Voltage



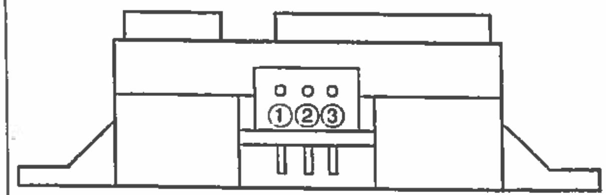
*All units in millimeters (mm)

FEATURES

- Analog output
- Effective Range: 10 to 80 cm
- LED pulse cycle duration: 32 ms
- Typical response time: 39 ms
- Typical start up delay: 44 ms
- Average current consumption: 33 mA
- Detection area diameter @ 80 cm: 6 cm

DESCRIPTION

The GP2D12 is a distance measuring sensor with integrated signal processing and analog voltage output.



PIN	SIGNAL NAME
①	V_O
②	GND
③	V_{CC}

GP2D12-6

Figure 1. Pinout

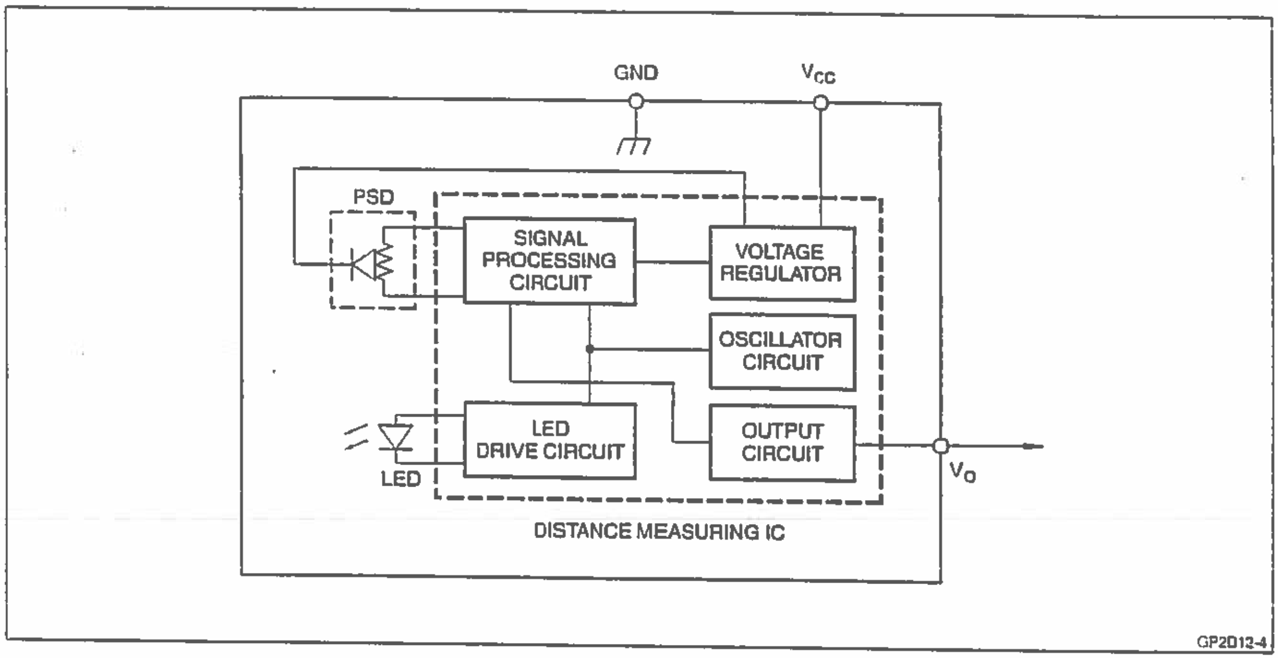


Figure 2. Block Diagram

Sensitivity

The usable range of the GP2D12 is between 10 cm and 80 cm. The readings for objects closer than 10 cm are unstable and therefore not usable.

Resources and Downloads

Check out the Sharp GP2D12 Analog Distance Sensor product page for example programs, the manufacturer datasheet and more:

http://www.parallax.com/detail.asp?product_id=605-00003

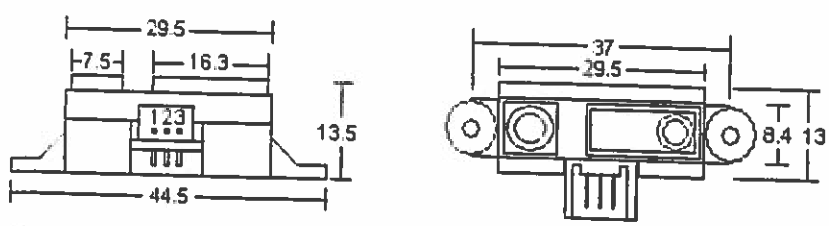
Specifications

Symbol	Quantity	Minimum	Typical	Maximum	Units
Vcc	Supply Voltage †	4.5	5.0	5.5	V
Topr	Operating Temperature †	-10	-	+60	°C
Tstg	Storage Temperature †	-40	-	+70	°C
ΔL	Distance Measuring Range †	10	-	80	cm
Vo	Output Terminal Voltage (L=80 cm) †	0.25	0.4	0.55	V
ΔVo	Output change at L=80 cm to 10 cm †	1.75	2.0	2.25	V
Icc	Average Dissipation Current (L=80cm) †	-	33	50	mA

† data obtained from Sharp's GP2D12 datasheet

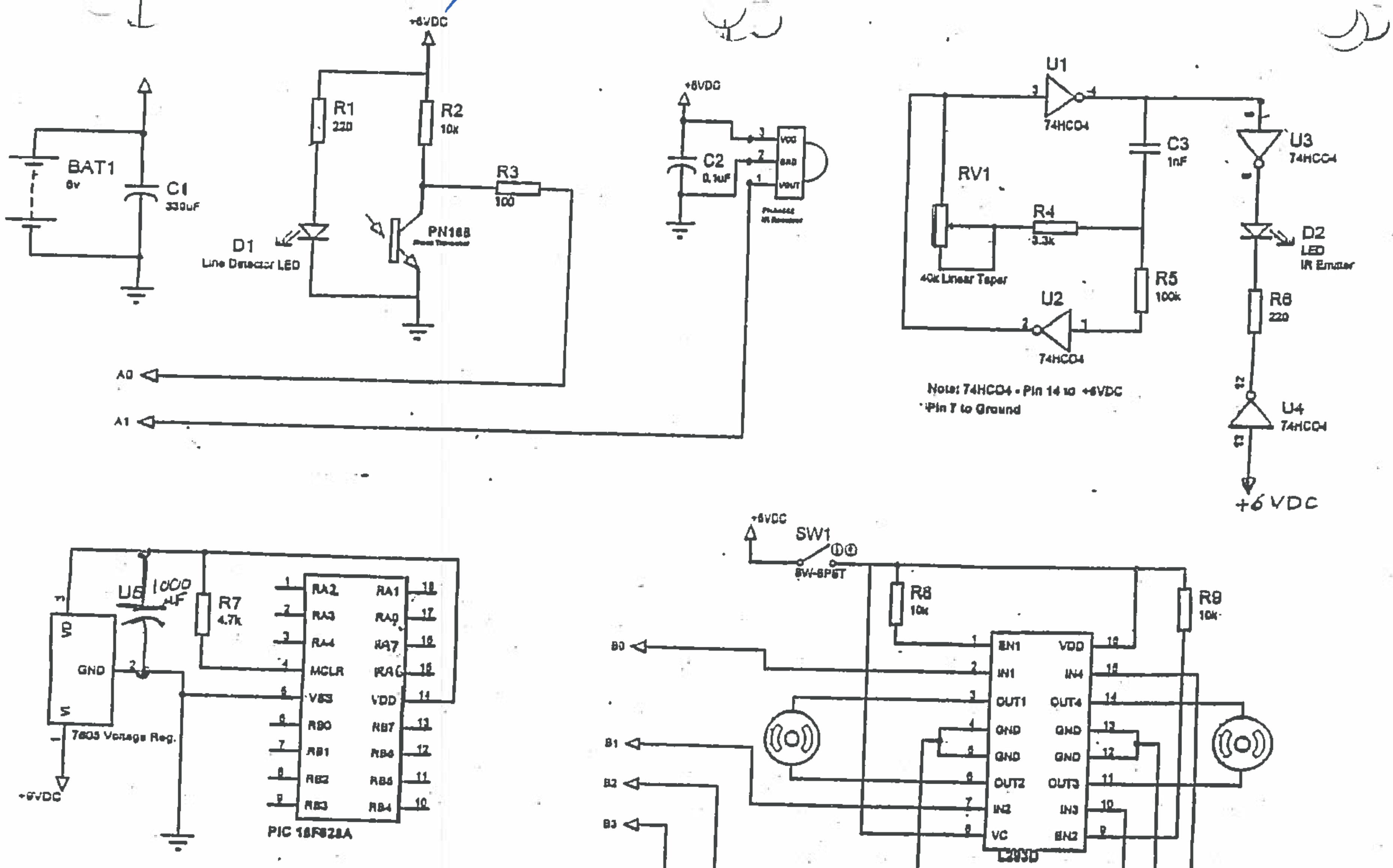
Pin Definitions and Ratings

Pin	Name	Function
1	Vo	Voltage Output
2	GND	Ground
3	Vcc	Supply Voltage



*All units in millimeters (mm)

* Not for Firefighter
- Only for reference for motor driver



Sumo Bot