PIC Microcontroller Trainer

By

Bruce Misner
Lakehead University
Electrical Engineering

August 1, 2002

Overview

The purpose of constructing this Printer Trainer is to give the student an experimental device that they can develop and test code using the PIC 16F877 microcontroller. The chioice to develop around the 16F877 is because it has a lot of features available in microcontrollers. The flash memory device was desirable because it can be rapidly reprogrammed and put back in circuit to be tested. The printer was used because it was there, but also because it was full of al sorts of components and hardware that all went into this project. If someone were to build a similar unit old dot matrix printers can be pulled out of any dumpster. The bottom case of the printer was ideal to house the carriage and stepper motor because that was what it was designed to do. Very little in the way of mechanical fabrication was required to build this. Electrical fabrication in way of PCB's and a little wiring was all that was needed to complete the project. The Development board was designed to stand alone and flexible so it could be used for any application. This was only one.

Table of Contents

The PIC Printer Trainer

The printer trainer started as an old dot matrix printer that was scavenged for parts. The bottom printer chasis, the metal base for the carriage, carriage and printhead base, and the stepper motor for X direction were the only components that remianed together. The stepper motor had a gear with a slotted wheel that was used in the belt drive that moved the print head. The belt drive was also left intact.

One of the components scavenged was a photo interupter that wasn't mounted on the slotted wheel, but it would be now. A small bracket also obtained from the printer was used to secure the interupter and was taped (that's right, taped) to the metal frame so that the slotted wheel was fully into the interupters gap. After some experimenting it was found that 270 Ohm current limiting resistor for the LED on the interupter and a 1 K resistor on the emitter of the photo transistor produced a nice 5V pulse that the PIC could count with TMR0 (used as an event counter).

The stepper motor in this printer was 4V 1.5A. It was more than my little AC adapter could manage, so a separate motor voltage supply was needed. Binding posts were installed into the base of the printer and wired to the stepper motor drive circuit via a terminal strip. The motor voltage is switched on and off using TIP31 power transistors with the base driven directly from the Port C pins of the PIC. With a 220 Ohm resistor in series with the base the PIC being able to source 200mA would provide ample base current. Be careful though, power transistors have low betas.

Small brackets had to be constructed to fasten the limit switches so when the print head assembly reached the end of the carriage the limmt switch would be contacted. The brackets were fashioned out of some scrap metal and fastened to the printer base through screw holes already on the base of the printer.

The position pushbutton and analog input potentiometer were mounted together on a PCB and then afixed to the case of the printer. This input board gives a front panel for manual control of the print head, or other application input.

The Printer Trainer has a parallel connection to the printer port of the computer. It connects an eight bit port on the PIC Development board to the eight data lines of the computer. The complexity of the interface can require several other bits to perform some sort of handshaking, but for my interface I required only the one additional bit. The programming of the interface will be covered in depth later. Whether you can live with this simplicity would be dependant upon your application.

Other PCBs designed and built to be used to work with this setup were an eight bit LED board and an analog output board. The analog output board is and 8 bit D/A that can be connected to an 8 bit port on the PIC Development board to give an output voltage of 0Vdc to 5Vdc. The LED board has 8 LEDs and current limiting resistors that again can be connected to an 8 bit port on the PIC Development board and was instrumental in debugging code. The schematics follow in the figures section.

The PIC Development Board

The PIC Development board was the starting point of all of this. I wanted my own development board because I wasn't totally happy with the PIC DEM2 board from Microchip even though it did serve its purpose. The PIC DEM2 board had certain inputs and certain outputs but somewhat locked you into using the controller in a certain way. The one thing that I would have changed was to have the output LEDs somewhere other than Port B because Port B has a lot of interupt capabilities, which would be inputs and you tie up this port outputting to the LEDs. Hence, I built my own.

First, we have a 40 pin socket to place the programmed chip. Then I crammed a ZIF socket into it. Why? For development purposes the ZIF made it very easy to add or remove the chip from the socket. The plan is, once you are happy with the operation you pull out the ZIF and put the controller into the regular socket where it will reside for a longer time and the ZIF can be used again.

The power for my board comes from a 9Vac wall adapter, which was handy, but will take the secondary of a transformer from 9 to 30Vac. The development board rectifies and regulates the voltage at 5Vdc required for the controller. The current rating on my wall adapter is 300mA. This could be taxed quite easily because the chip itself has fairly good drive characteristics up to 200mA on most outputs. The development board also has terminals to use the 5Vdc that other devices may use for power. So, caution about power consumption should be considered.

The oscillator can be a variety of input. Microchip provided with the PIC DEM2 board a 10MHz canned oscillator that I used in my development board. It has the same footprint as a 14 pin dip. It only uses the corner pins but made it easy when designing to just drop down a dip footprint.

Other than the programming voltage which I put a jumper on so you can do the low voltage in circuit programming, a feature I have not used yet but it is there if required. All the rest of the board consists of header pins that bring the I/O pins out and ready for connection to your devices you which to control.

The deveopment board schematic is the Document Number 1 in the figures section.

Other information related to the programming of the chip will be covered in a section to follow.

The Stepper Motor Drive Circuit

The stepper motor drive circuit used is probably the simplest drive circuit you can build. It consists of the drive transistor (TIP31), a freewheeling diode, and a base resistor.

Other circuits were tested but found to be too much trouble for the savings you would gain. If you were to use a stepper motor drive chip or a logic circuit using Flip Flops (both of which were tried) you merely need a pulse to make the step and a logic bit to change the direction (two bits on the microcontroller required). With this circuit you require 4 bits on the controller (one for each coil) and programming to pulse each coil in the proper order. This complicates the programming of the controller only slightly but the code is included for this circuit.

This stepper circuit simply drives the base of the drive transistor directly from one bit on a controller port. Most ports on this controller are capable of sourcing up to 200mA of current making it acceptable for the base current. The beta of the power transistor is only in the neighbourhood of 10 giving a maximum collector current of around 2A which is more than this stepper motor requires. The motor voltage, however, requires a separate supply to deliver this much current, so binding posts are provided to connect a separate motor supply.

Switch Circuitry

The PIC Printer trainer has four switches installed onboard, two limit switches, and two pushbuttons. The switches are connected in the same manner. The 5Vdc supply connects to one end of the switch. The other end of the switch is connected to a 10K resistor. The open end of the resistor is then taken to ground. The input to the PIC is taken inbetween the switch and the resistor. Hence, when the switch is open the voltage is dropped across the switch so the PIC doesn't see it leaving a logic low on the input to the PIC. When the switch closes current flows through the 10K resistor dropping the 5Vdc acoss it. This yields a logic high at the input to the PIC.

The limit switches are positioned at the ends of the carriage. This will allow us to test if we are at either end or not. The pushbuttons are located at the front of the trainer and would be used for manual positioning of the print head.

Analog Input

On the front panel attached to the same PCB as the pushbuttons is a 10K potentiometer with one end to the ground and the other to the 5Vdc supply. The wiper voltage can be taken to an Analog input on the PIC.

LED Board

An LED board was fabricated using 8 individual LED's and 8 270 Ohm resistors. Each LED resistor combination connects to an outport bit on the controller. Again, the drive capabilities of the PIC are adequate to light the LED. This board is a must for debugging your code, by sending status numbers to a port you can track your codes progression.

Analog Output Board

An analog output board was also constructed to give a 0 to 5Vdc output based on the 8 bit binary value written to it.

The D/A used was the 8 bit AD558. It is connected so that on any change in port value will automatically give a change in the analog output voltage. Connecting the Chip Enable and Chip Select lines to ground gives the constantly tracking feature. The analog output is from pins 14, 15, and 16 with the analog common as a common ground with the digital and PIC ground.

This board has not been incorporated into the trainer, however the usefulness of such a device warrants the fabrication. It has been tested to be functional.

Programming the PIC Microcontroller

Microchip has a large selection of microcontrollers so the first thing we must do is select a chip that suits our application. The 16F877 was selected to be purchased by the school because of its diversity. It is a fairly large controller and would be overkill for a lot of applications. However, the 16F877 has a lot of functions so that no matter what the student might need it will have it. It also has flash program memory for easy reprogrammability. The factors that might determine your selection, number of I/O lines required, analog measurement, pulse width modulation, event counters and timers, in circuit programming, master slave bus mastering, serial communications I2C and USART, and whatever else you can dream up. All of the mentioned features are all supported by the 16F877.

The next thing you will need is a programmer. There are third party programmers and it is not hard to build your own. Microchip has the PICstart Plus that we purchased. The PICstart Plus came with MPLAB software which supported the programmer and also came with an assembler. There are C compilers available but we wish to focus on the low level function of these devices which make assembly the way to go. However, C would speed development time. The main consideration between C compilers is there library functions and how complete they would be. If you were to buy a cheap C compiler you may be writing a lot of functions in assembly and building your own libraries. This defeats the purpose of the compiler.

The first thing that will be a sticking point for you is the Configuration bits. The configuration bits control some of the functionality of the device. The best advice I have is turn off all that are not used. My first encounter was simply displaying an eight bit value on the LED board when attached to Port B. If the in circuit programming is enabled it reconfigures PORT B Bit 4 and you will get no output. There are other functions tied to the device that can be disabled due to configuration bits, so turn them off if you don't need them and when you do enable a function be careful that you know the consequences of the action.

MPLAB Software

The MPLAB software is essentially a shell program that gives the user access to the function of the PIC programmer. It gives access to emulator, the configuration bits, the downloading of the code, the assembler, the device selection, the status and value viewing windows. After a short time you will be good with this software it is pretty straight forward.

The first thing to do is Project->New and create a new project , then name the project and then cancel the next window. Then File->New and create an asm file, then name it and save it. Then go back to Project->Edit and click Add Node. The Node is the asm file you just created, then say OK. Once you have code in your asm file to compile it you Project->Build All.

The simulator I have only used for small test code, but the capability to provide stimulus can allow for fairly good simulation. However, with the flash memory and the ZIF socket on the development board, reporgramming and in circuit testing is quite quick. The combination of the F5 F6 and F7 keys provide, halt reset and step functions.

The SFR RAM and ROM buttons are very useful. When simulating you can view register values with the SFR (Special Function Registers). After compiling, you can click the ROM button to view the code and how it will be stored on the chip. Be sure to keep track of the final address used by your program because you will want to know that when you download your code.

Once you have compiled your program and you are ready to download and test you must first enable the programmer. This is when you select the device and set your configuration bits. Once you have done this you can minimize these windows and get them out of your way. Next you click Programmer->Program and the window that comes up you put the end address of your program in the first box instead of 1FFF so that it only programs your code on to the chip and not a lot of zeroes after. This is a big time saver.

Architecture and Assembly

The data sheets for the 16F877 are in a following section. Familiarity with both is key to developing code for these controllers. The claim is that it is a simple RISC instruction set (35 instructions) and it so easy to use. What they don't tell you is that there are 512 possible funtion registers that depending on what feature of the chip you are using you will have to know the registers that control it. Now, not all 512 registers are used and there are places for you to define one of these registers for your own use. The registers are contained in four register banks. The easiest thing to do here is to look at some sample code.

```
bcf     STATUS,RP1
bsf     STATUS,RP0
movlw   B'11111111'
movwf   TRISB
bcf     STATUS,RP0
movf    PORTB,W
```

This code is something you will probably use frequently. It provides a simple 8 bit input on PORTB and saves the value in the W register. First, we will look at the first two instructions. The function register STATUS cantains two bits RP0 and RP1 that controls the register page you are dealing with. The bcf and bsf commands set or clear the bit specified in the instruction. In this case we clear RP1 and set RP0 giving us access to register page 1 where ther TRISB register is located. By writing 255 or B'11111111' to TRISB it defines all PORTB's bits as input. Writing 0's would make it output. Then we clear RP0 pointing us to register page 0 where we can now access PORTB register. Then the movf command reads the value from PORTB to the W register. W is like the accumulator or working register and will be frequently used.

```
bcf     STATUS,RP1
bsf     STATUS,RP0
movlw   B'00000000'
movwf   TRISB
bcf     STATUS,RP0
movlw   0xff
movwf   PORTB
```

This now defines PORTB as output and writes 0FFH to PORTB setting all bits high on the port pins. The movlw moves the following value into the W register. The movwf command writes whatever is in W to the register or port specified. Registers and ports are synonomous.

You can define your own register by using the set directive.

```
myreg   set     0x20
```

20H is an open register in bank 0 and we have assigned the name myreg to it. Now, as with PORTB register we can use myreg in the same manner. It creates in essence a variable to be used in the code.

I will not do much more at this time with this section, the data sheets for the controller follows in a separate section and my code as well, so you can take it from here.

Interupts

Interupts are an integral part of the microcontroller because so much of its uses are event driven. If you look at my code in the back the main prgram is just a loop waiting for something to happen. This is effective porgramming of a microcontroller (according to Bruce). You must at the beginning of the program define the interupt vector (there is only one) and it resides at memory locaton 0x4.

```
org 0x4
goto    int_vect
```

This defines the vector and then later in the program you will have a label int_vect that will signify the start of the interupt handling routine. When an interupt occurs we must figure out what caused the interupt. To make it easier we should disable or mask interupts we don't intend on using. This is done prodominantly using the INTCON register. Every bit in INTCON controls an interupt or another set of interupts. Interupts typically have an enable bit and a status bit. You set the enable bit for the interupt and when the event occurs it will vector to the interupt handler for you to deal with. Once an interupt occurs the first thing to do in the handler is to determine what caused the interupt using the status bits. Status bits are still used even if the interupt is disabled

Math

A quick word on the math capabilities of the controller. Essentially you have add and subtract. There are logic functions as well. The subtract function leaves the difference in the W register, but if there is a borrow the result is in twos complement form and if there is no borrow the difference is correct. See my code to find out how I have handled this.

This is all I am going to deal with as far as instruction sets and registers in an overall perspective. I will cover additional material in a more specific manner.

Using TMR0

TMR0 is one of three timers available on the 16F877, however as with most functions on the PIC microcontrollers it can be used used in several ways to give all kinds of functionality to the controller. The functions of TMR0 are interval timer, and extenal event counter. TMR0 is in a sense the main counter on the controller. TMR0s functions are controlled by OPTION_REG, INTCON, and TMR0 registers. TMR0 register is the current count, you can read and write to it. TMR0 register works in conjunction with an 8 bit prescaler to give up to a 16 bit count if desired. In the OPTION_REG you can select the prescaler value, select input source, and select whether it triggers on a rising or falling edge. The INTCON register is the main interupt control register. We can have TMR0 either generate an interupt or not and we can check for an overflow with the status bit. Whether the interupt is active or not the status bit is still affected by the overflow condition.

In our application we used TMR0 as an external event counter to count the pulses generated by the slotted wheel and the photo interupter. The input to TMR0 will be RA4, bit 4 on PORTA, and will trigger on the rising edge. Hence, the following must be set by the code to effect this condition.

```
bsf     OPTION_REG,T0CS     ;selects input as RA4 bit
bsf     OPTION_REG,T0SE     ;select rising edge
bcf     OPTION_REG,PSA      ;assign prescaler to TMR0
bcf     OPTION_REG,PS2      ;prescaler value 000
bcf     OPTION_REG,PS1      ;still divides input by 2
bcf     OPTION_REG,PS0
```

If our counter overflows bit 2 in INTCON will signal this and if we want to generate an interupt based on this we ould set bit 5.

```
bsf     INTCON,T0IE             ;enables interupt generation clear to disable

loop    btfss   INTCON,T0IF     ;stays in loop until Overflow occurs
        goto    loop
```

The other two timers belong in the peripheral interupt section which in certain respect operates the same, however, different registers are used.

PORTB for Input

One of the main reasons I designed my own Deveopment board is because the PIC-DEM2 Demo board had the eight LEDs connected to PORTB. This made no sense to me and I will explain why. Two other interupts that are available through the INTCON register are Port B change and RB0/INT, both of which I use in the trainer and found to be extremely useful. In order to use these they must be inputs. Hence, if I wish to monitor an input and interupt on an event change right away I think of using PORTB.

For the trainer I have the four switches, the two positioning switches and the two limit switches. All of these switches are hooked to the upper bits of PORTB. Any change in state of these switches will cause an interupt. The RB0/INT line I tie to my lpt interface and when a high signal is sent from the computer an interupt is generated and it knows to read data from the computer.

It is possible to porgram the port to have some input bits and some output bits, which is nice for smaller controllers with limited I/O, but even with bigger controller we may still need to do this. On the trainer the event counter input is on PORTA and a lot of the analog input is on PORTA, as well. So PORTA is programmed some as input and some as output.

Making Steps in the Right Direction

There are two considerations you must deal with if you want to make the stepper motor move, which order to pulse the coils and the length of time each pulse has to be.

First we must determine the pairs of coils with an ohmmeter, there are two pairs. In the schematic section in the back you can see I have labelled the coils A, B, C, and D. and the sequence you have to pulse them in is as follows:

	Step 1	Step 2	Step 3	Step 4
Coil A	ON	ON	OFF	OFF
Coil B	OFF	OFF	ON	ON
Coil C	ON	OFF	OFF	ON
Coil D	OFF	ON	ON	OFF

Now that we have determined the order in which to pulse the coils and we have assigned I/O bits (PORTC lower four bits) we can figure out the four bit values to be written to create this pulse order (5,9,10,6). This order will cause the motor to step in one direction. If the reverse direction is required you would reverse the order of the pulses (6,10,9,5). Once you have finished making your step you can leave the final value on the coil maintaining a holding current to hold the motor in position or if no torque is on the shaft of the motor you can turn off all coils and save power.

The other consideration was the length of the pulses required to energize the coil enough to cause a step. On source had said it required 20ms in order to energize the coil. What I did was continually decrease the pulse length until it wouldn't move anymore, then made it a little longer. What pulse length

did I end up with? I don't know. I used TMR1 to cause a delay (calldelay subroutine). We can look at how TMR1 operates, as it is different than TMR0. I didn't generate an interupt I simply polled the overflow status bit PIR1 bit 0 and disabled the interupt using PIE1 register bit 0. It was found by loading value 0xe800 into TMR1H and TMR1L provided the delay required when used with the 10MHz clock. If your clock is a different frequency your delay value will be different.

Communication Breakdown

It was chosen to use the LPT1 printer interface to communicate with the PIC Development board. We connected the 8 data bits from the LPT1 to PORTD and pin 16 of LPT1 the /INIT bit was connected to PORTB bit 0. I didn't build up a full fledged bi-directional interface but simply what was needed. It is bidirectional but only when the /INIT line is raised will the PIC send a value.

Let's go through what is done. The PIC, when first powered on, will search for the first limit switch and call it position 0. The PIC is currently outputting 0FFH to the computer. My program on the computer side knows that the number to follow 0FFH will be the terminal count. When the controller moves the print head over to the second limit switch to determine the terminal count and the PIC outputs the terminal count the computer reads it as such. The VB program then treats every subsequent number as a current location. Every movement of the print head is recorded by TMR0 and when the print head stops it outputs the new current position.

When the VB program wants to send a setpoint to the controller the /INIT line is raised causing an RB0/INT interupt on the controller side. Once the interupt is generated ports on both sides of communication have to be switched. First, on the VB prgram side setting bit changes the data port of the LPT connection to an output from what we have as normally an input (as a printer port it is normally an output port). The program ,then, outputs a new setpoint to the PIC and then after a time delay reverts back to an input port to continue to receive currnt location values from the PIC. On the PIC side, when the RB0/INT interupt occurs it changes PORTD to an input port so it can receive the new setpoint. The PIC then waits for the RB0 line to drop to a low value then changes PORTD back to an output and is then ready to resume as normal.

Analog Input Example

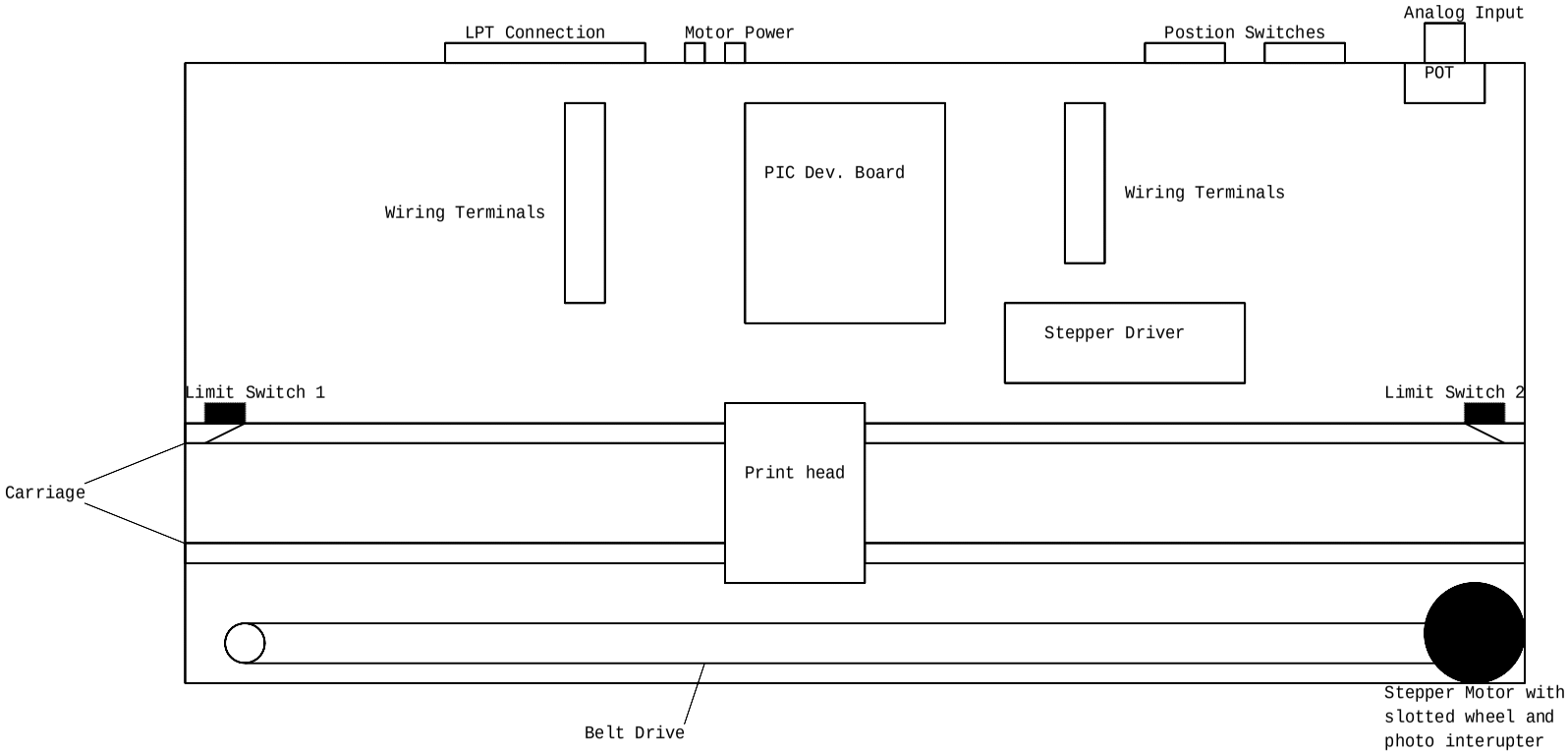
The analog input example was not incorporated as part of the printer trainer because it didn't really fit with anything that I was developing. So, using the PIC-DEM2 demo board I developed a small example of how to utilize this.

The A/D converter section of the microcontroller is controlled by seven different registers. Start with the INTCON register. If you wish to have the A/D cause an interupt you must have PEIE bit (bit 6) set to enable peripheral interupts. The PIE1 register you must set bit 6 to enable the A/D to generate an interupt. The PIR1 registers' bit 6 is the flag bit that gets set when the A/D is finished its conversion. PIR1 is tested to see if the A/D caused the interupt or if no interupts are used it still tells you when the A/D is finished. ADCON0 allows you to select the clock source of you're A/D, select which A/D channel you wish to use, and there is a bit you can turn the A/D on or off as desired. ADCON1 allows you to program what inputs are to be used for analog input and how the result is justified in the two regsters ADRESH and ADRESL.

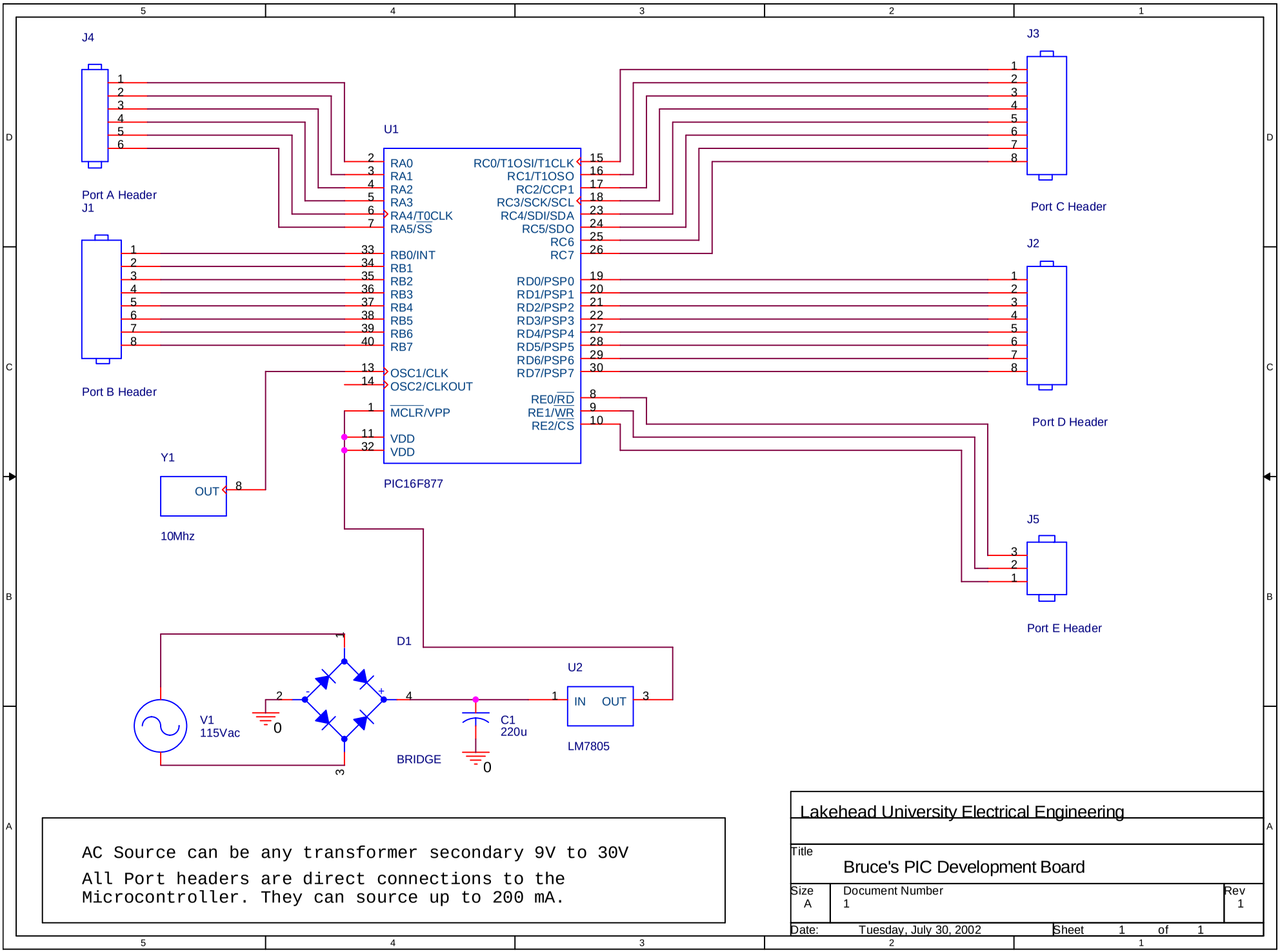
The A/D on the controller is 10 bit and that is why it requires two 8 bit registers to store the result. In our code example we justify the result left so that the eight most significant bits are all in ADRESH and that is the only register I read. If you require the full 10 bit precision you will have to deal with both registers.

To start a conversion you must first configure you're A/D by writing to all of these registers so you will get the desired results. Once this has been done you simply need to set ADCON0 register bit 2 (GO bit) and the A/D will start its conversion. When the A/D is done you can either test the GO bit again because when it drops low the A/D is finished. You can also tell if the A/D is done by looking at the flag bit PIR1 bit 6 (ADIF). When ADIF goes high the conversion is done and is valid if interupts are used or not.

Schematics



Lakehead University Electrical Engineering		
Title: PIC Printer Trainer		
Size: A	Document Number: 5	Rev: 1
Date: Friday, August 02, 2002	Sheet 1 of 1	



AC Source can be any transformer secondary 9V to 30V

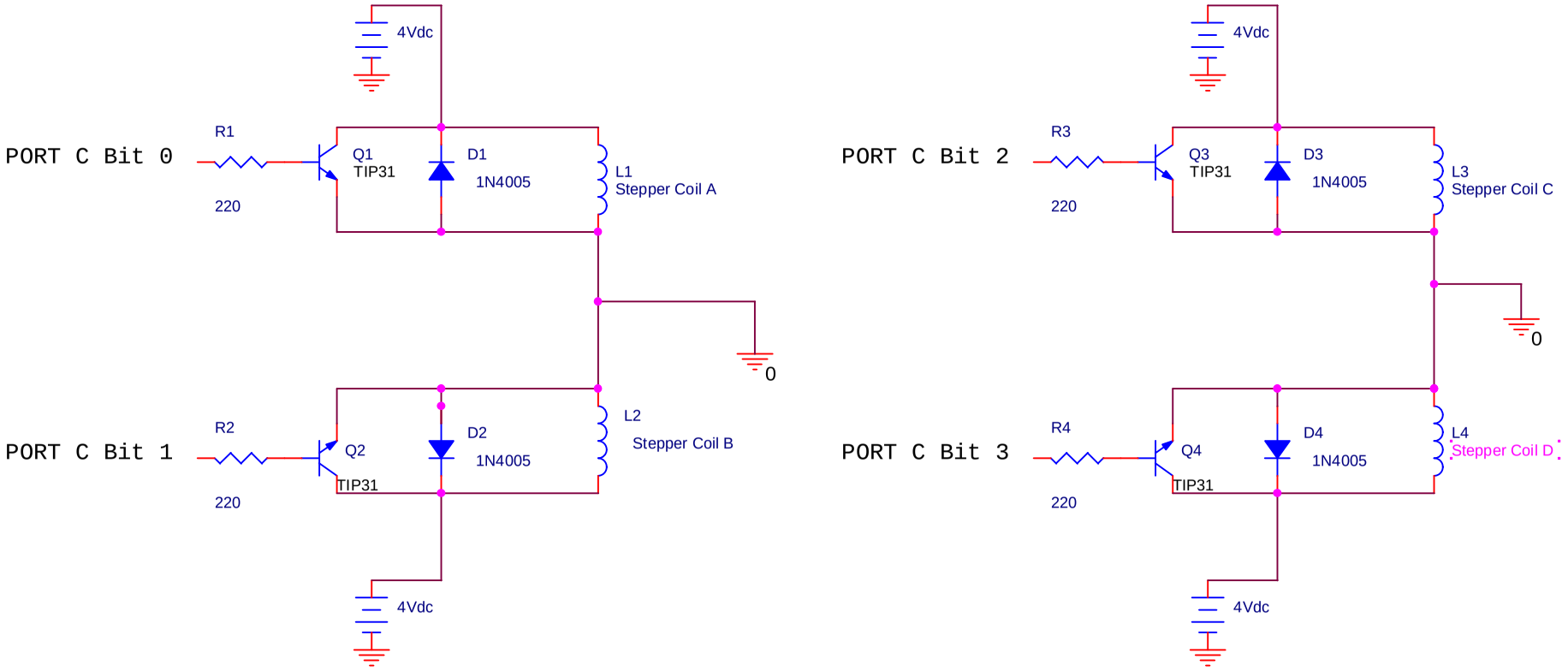
All Port headers are direct connections to the Microcontroller. They can source up to 200 mA.

Lakehead University Electrical Engineering

Title: Bruce's PIC Development Board

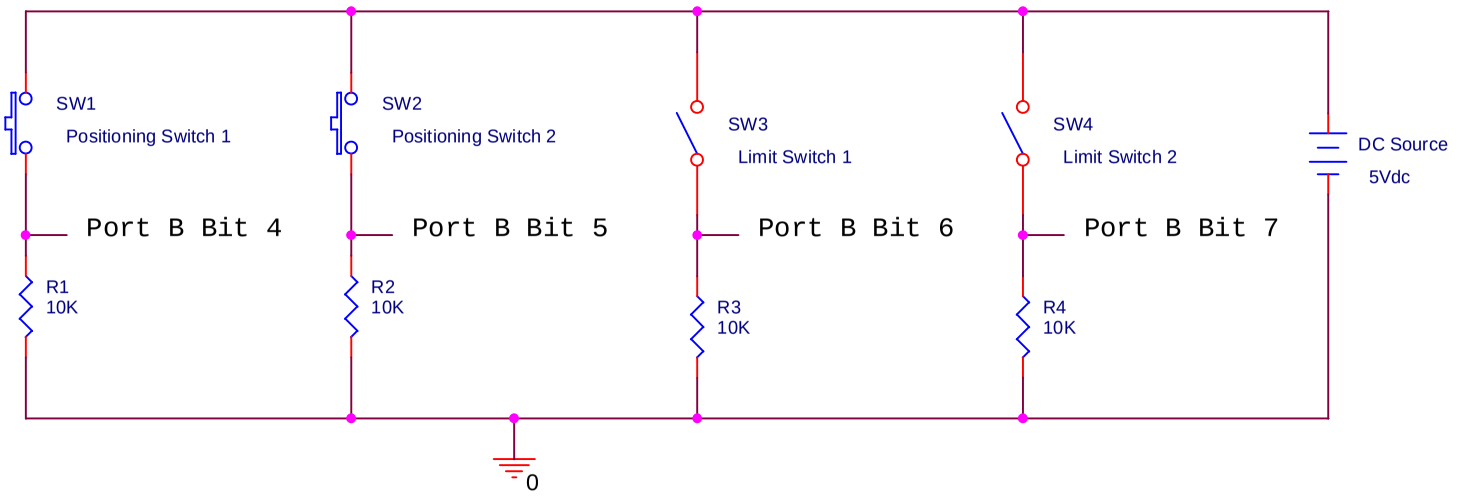
Size	Document Number	Rev
A	1	1

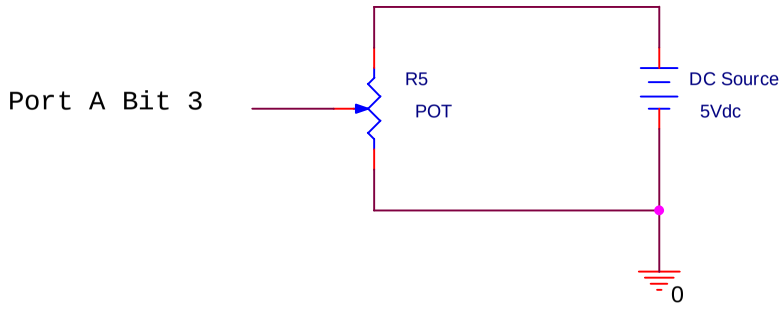
Date: Tuesday, July 30, 2002 Sheet 1 of 1



Lakehead University Electrical Engineering		
Title: Stepper Motor Drive Circuit		
Size: A	Document Number: 2	Rev: 1
Date: Thursday, August 01, 2002	Sheet 1 of 1	

Switch Circuitry





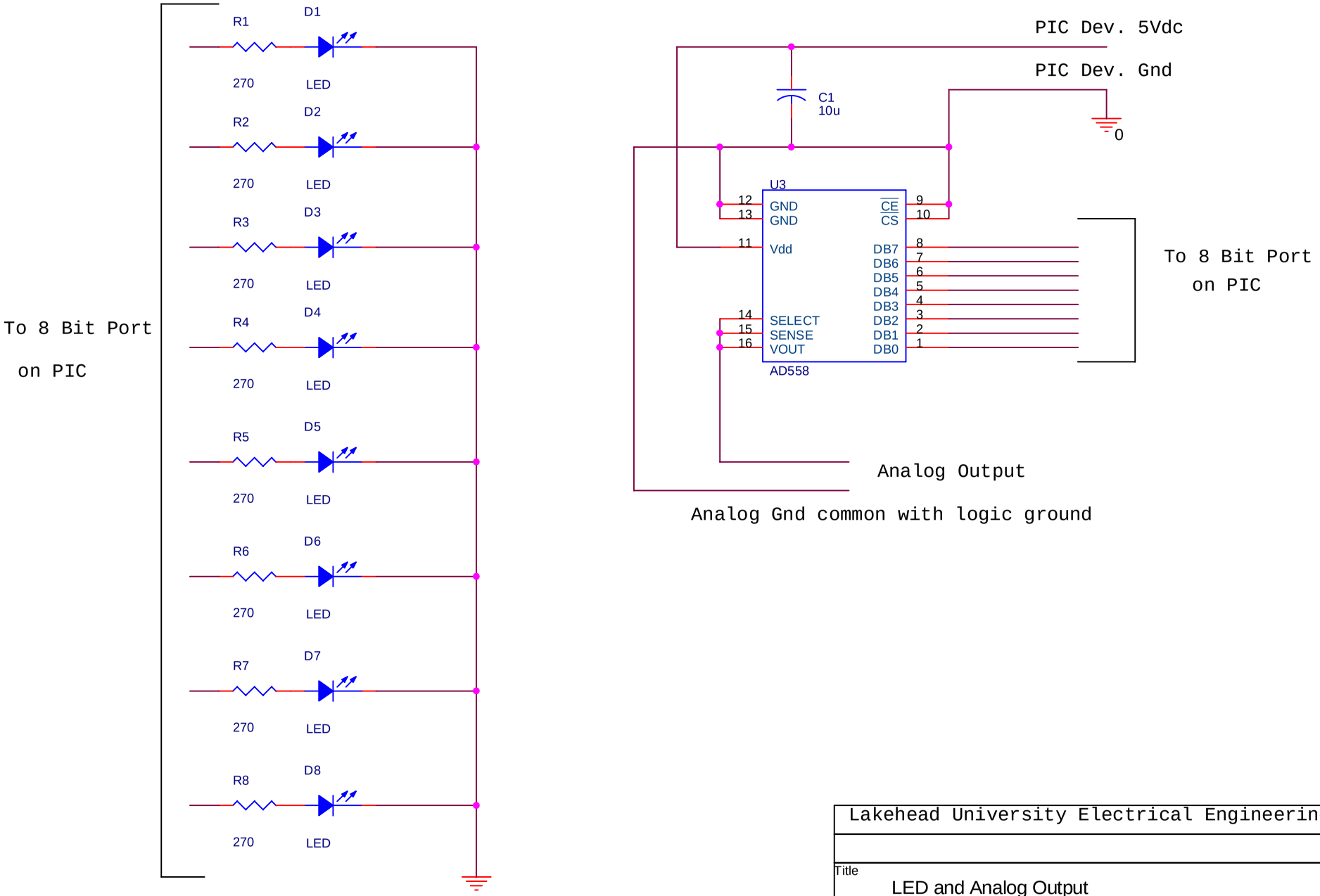
Analog Input Circuitry

Positiong switches and potentiometer located on Input board at the front of the Trainer

Limit switches wired on the terminal strips and switches located at each end of the carriage

5Vdc supply comes from the PIC Development Board

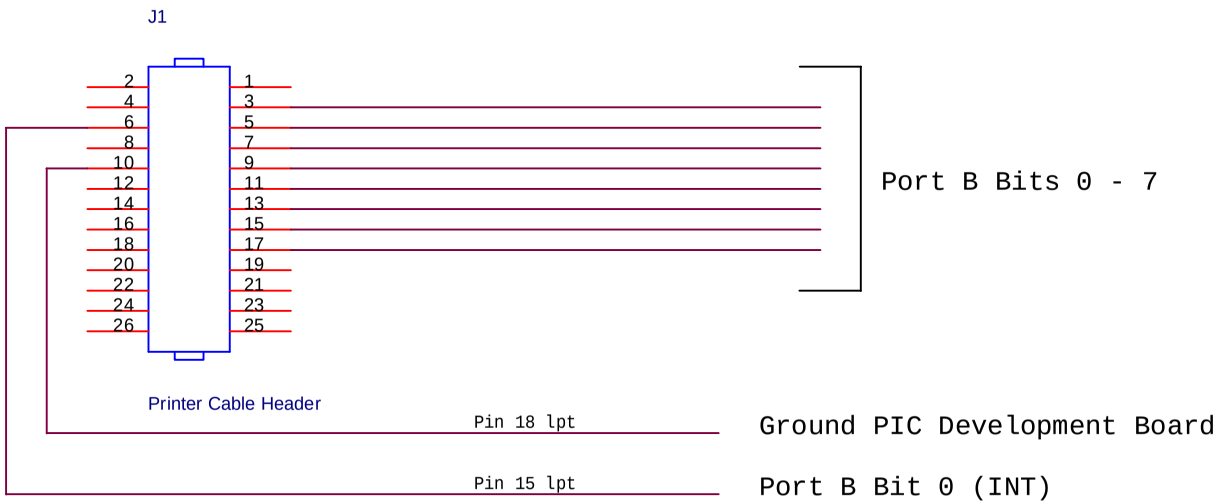
Lakehead University Electrical Engineering		
Title: Miscellaneous Circuitry		
Size: A	Document Number: 3	Rev: 1
Date: Thursday, August 01, 2002	Sheet 1 of 1	



Lakehead University Electrical Engineering		
Title: LED and Analog Output		
Size: A	Document Number: 6	Rev: 1
Date: Tuesday, August 06, 2002	Sheet 1 of 1	

Printer Side

PIC Side



Pins 3 - 17 Odd numbers are pins 2 - 9 on the DB25 connector on the PC

Pin 18 is the ground shared between computer and PIC

Pin 16 is the /INIT line on the printer side and is an output connected to Port B Bit 0 of the PIC. Triggers an interupt on low to high transition.

Lakehead University Electrical Engineering		
Title: LPT Connections for PIC Trainer		
Size: A	Document Number: 4	Rev: 1
Date: Thursday, August 01, 2002	Sheet 1 of 1	

Code Listing

Printer Trainer Code

```
;
;     Created by Bruce
;
;     Designed to run on the Bruce Dev. Board
;     and Printer Trainer
;
;
;     Uses 16F877 MicroController at 10 Mhz
;
;     IO Pin connections
;
;
;     Port A (ra)
;
;     pin 3 (ra3) - Analog input (not used in this)
;     pin 4 (ra4) - TMR0 input
;
;
;     Port B (rb)
;
;     pin 0 (rb0) - lpt direction signal (INT signal)
;     pin 4 (rb4) - direction pushbutton
;     pin 5 (rb5) - direction pushbutton
;     pin 6 (rb6) - limit switch
;     pin 7 (rb7) - limit switch
;
;
;     Port C (rc)
;
;     pin 0 - pin 3 (rc0:3) - Stepper motor coils, connects to
;                         driver board
;
;
;     Port D (rd)
;
;     pin 0 - pin 7 (rd0:7) - primarily output to send position to
;                         computer but input when computer sends
;                         new set point to PIC. Determined by
;                         rb0 INT signal
;
;
;     Define Part
;
      list p=16f877
      include "p16f877.inc"
;
;
;
endpos      set   0x20        ;user defined register
curpos      set   0x21        ;bank 0
setpos      set   0x22
tempval     set   0x23
tempva1     set   0x24
;
```

```
;
;     Reset Vector
;
      org   0x0
;
;
      goto Start
;
;
;     Interupt Vector
;
      org   0x4
      goto intvect
;
;
;     Start of program
;
;     First to initialize ports and interupts
;
Start
      bsf   STATUS,RP0        ;register bank 1
      bcf   STATUS,RP1

      movlw B'00000000'       ;define port c as output
      movwf TRISC             ;stepper motor port
      movlw B'00000000'       ;define port d as output
      movwf TRISD             ;lpt port connection 8 bits
;
      movlw B'11111111'       ;define port b as input
      movwf TRISB             ;pushbuttons(4,5)
                              ;limit switches(6,7)
                              ;lpt direction bit(0)
;
      movlw B'11100000'       ;option_reg byte
      movwf OPTION_REG        ;TMR0 input RA4
                              ;Port B pull ups off
                              ;0 for prescaler value
                              ;prescaler assigned to TMR0

      bcf   STATUS,RP0        ;register bank 0
      bcf   INTCON,0          ;clear port b change flag
      bcf   INTCON,3          ;do not enable port b change interupt
      bcf   INTCON,2          ;clear TMR0 overflow flag
      bcf   INTCON,5          ;do not enable TMR0 interupt
      bcf   INTCON,4
      bsf   INTCON,7          ;set global interupt
;
;
;
      movlw 0xff              ;send 0xff to PORT D
      movwf PORTD             ;signals lpt and VB app to
                              ;expect the next number to be
                              ;the end count
;
;     move print head to one end and call it count 0
;
findzeropt
```

```
      call  makefstep         ;move one step
      btfsc PORTB,7           ;test for limt switch contact
      goto  setupt0           ;if contacted goto
      goto  findzeropt        ;if not loop till contact
;
;
;     Find end count
;
setupt0
      movlw 0                 ;set TMR0 to 0
      movwf TMR0
;
findend
      call  makebstep         ;move one step in other dir.
      btfsc PORTB,6           ;check for other limit sw.
      goto  foundend          ;if contacted goto
      goto  findend           ;loop until contact
;
foundend
      movf  TMR0,W            ;save value of TMR0
      movwf endpos            ;save in user reg endpos
      movwf PORTD             ;send to lpt connection
      rrf   endpos,W          ;divide endpos by 2
      movwf setpos            ;endpos/2 = midpt or setpos
      bcf   setpos,7          ;ensure that the msb is clear
      movlw 0                 ;reset TMR0 to 0
      movwf TMR0
;
move2mid
      call  makefstep         ;move one step toward middle
;
      comf  TMR0,W            ;complement TMR0 value
      bcf   STATUS,2          ;clear Zero flag
      andwf setpos,W          ;and TMR0 with setpos
      btfss STATUS,2          ;test for zero
      goto  move2mid          ;if not zero not at midpt yet
                              ;loop and move another step
      movf  setpos,W          ;midpt found move setpos to W
      movwf PORTD             ;send midpt value to lpt conn.
      movwf curpos            ;and save value as curpos
;
;
;     Done start up routine now enable interupts
;     and start main program loop
;
;
      bcf   INTCON,0          ;clear rb change flag
      bcf   INTCON,1          ;clear INT flag
      bsf   INTCON,3          ;enable rb change interupt
      bsf   INTCON,4          ;enable INT interupt
      bsf   INTCON,7
;
;
;     Main Program Loop
;
;
loop
```

```
      goto  loop
;
;     Interupt Service Routine
;
;     an interupt has occured find out what caused it
;
intvect

      btfsc INTCON,1          ;is the computer wanting to
                              ;send a set point (INT bit)
      goto  getsetpt          ;if yes goto

      btfsc INTCON,0          ;was a pushbutton pressed
      goto  check4pb          ;if yes goto
      retfie                  ;neither caused interupt
                              ;disregard signal
;
;
;     Routine to handle a Pusbutton Press
;
;
check4pb
      bcf   INTCON,3          ;disable interupts
      bcf   INTCON,4
      btfss PORTB,4           ;was it this pushbutton that
                              ;was pressed
      goto  movforward        ;if yes move in one direction
      btfss PORTB,5           ;was it this pushbutton
      goto  movbackward       ;move in other direction
      goto  endint            ;if neither end interupt
;
;
;     Move in one direction
;
;
movforward
      bcf   INTCON,3          ;again disable interupts
      bcf   INTCON,4
      movlw 0                 ;initialize TMR0 to count
                              ;how far we move the print
                              ;head
      movwf TMR0
;
mfor
      btfss PORTB,5           ;is the pushbutton still
                              ;depressed
      goto  endfstep          ;if no then end routine
      btfsc PORTB,6           ;has the limit switch been
                              ;contacted
      goto  endfstep          ;if yes the end routine
      call  makefstep         ;move one step
      goto  mfor              ;loop while conditions are
                              ;still good
endfstep
      movf  curpos,W          ;move curpos to W
      subwf TMR0,W            ;subtract TMR0 count
                              ;measured how far we went
```

```
      movwf tempval           ;save in tempval
      decf  tempval,W         ;undo the twos complement
                              ;result dec and complement
      movwf tempval
      comf  tempval,W
      movwf curpos            ;save new current position
                              ;to curpos
      movwf PORTD             ;send new position to lpt conn.
      movlw 0                 ;reset TMR0 to 0
      movwf TMR0
      goto  endint            ;end routine
;
;     Pushbutton press move it in other direction
;
movbackward
      bcf   INTCON,3          ;disable interupts
      bcf   INTCON,4
      movlw 0                 ;reset TMR0 to 0
      movwf TMR0
;
mback
      btfss PORTB,4           ;is pushbutton still depressed
      goto  endbstep          ;if no then end routine
      btfsc PORTB,7           ;was the limit switch contacted
      goto  endbstep          ;if yes then end routine
      call  makebstep         ;move one step
      goto  mback             ;loop while conditions are
                              ;still good
endbstep
      movf  curpos,W          ;move curpos to W
      addwf TMR0,W            ;add TMR0
      movwf curpos            ;save as new current position
      movwf PORTD             ;send value to lpt
      movlw 0                 ;reset TMR0 to 0
      movwf TMR0
;
;
endstep
endint

      bcf   INTCON,0          ;clear rb change flag bit
      bsf   INTCON,3          ;enable rb change interupt
      bsf   INTCON,4          ;enable INT interupt
      movlw 0                 ;ensure motor coils are
      movwf PORTC             ;turned off
;
      retfie                  ;return from interupt
;
;
;     Routine to handle set point change from computer
;
;
getsetpt
      bcf   INTCON,3          ;disable RB change interupt
      bcf   INTCON,4          ;disable INT interupt
      bcf   INTCON,1          ;clear INT bit
      bsf   STATUS,RP0        ;change register banks
```

```
      movlw B'11111111'       ;change port d to inputs
      movwf TRISD
      bcf   STATUS,RP0        ;change register banks
      call  calldelay         ;delay for port timing
      movf  PORTD,W           ;mov portd value to setpos register
      movwf setpos
wait4end
      btfsc PORTB,0           ;test for INT bit to drop low
      goto  wait4end          ;wait for this to happen
      bsf   STATUS,RP0        ;change register banks
      movlw B'00000000'       ;change port d back to output
      movwf TRISD
      bcf   STATUS,RP0        ;change register banks
;
;
      movlw 0
      movwf tempva1
      movf  setpos,W          ;load setpos value into W
      subwf curpos,W          ;subtract the current position
                              ;from the set position
      btfss STATUS,0
      goto  skipthis
      movwf tempval
      goto  moveback
skipthis
      movf  setpos,W
      subwf curpos,W
      movwf tempval           ;save temp value
      decf  tempval,W         ;decrement tempval result to W
      movwf tempval           ;save as tempval again
      comf  tempval,W         ;complement tempval save in W
      movwf tempval           ;move result to tempval
      movlw 0                 ;initialize timer0 back to 0
      movwf TMR0
;
;
;     if setpos was greater than curpos
;
move2p1
      call  makebstep         ;move a step
      movf  tempval,W         ;complement tempval and and with TMR0
      subwf TMR0,W
      btfss STATUS,0          ;if zero reached setpos
      goto  move2p1           ;if not reached go make another step
      movf  curpos,W          ;retrieve curpos value
      addwf TMR0,W            ;add TMR0 to give new curpos
      movwf PORTD             ;send new curpos to computer
      movwf curpos            ;save new curpos
      bcf   INTCON,1          ;clear INT bit
      bsf   INTCON,3          ;enable RB change interupt
      bsf   INTCON,4          ;enable INT interupt
      bsf   INTCON,7

      retfie                  ;return from interupt
;
;
;     if curpos was greater than setpos
```

```
;
;
moveback
      movlw 0                 ;reset TMR0 to 0
      movwf TMR0
move3p2
      call  makefstep         ;make a step
      movf  tempval,W         ;complement tempval
      subwf TMR0,W            ;and with TMR0
      btfss STATUS,0          ;if not zero make another step
      goto  move3p2
      movf  curpos,W          ;if zero then found position
      subwf TMR0,W            ;add TMR0 value to curpos
      btfsc STATUS,0
      goto  skipagain
      movwf tempval
      decf  tempval,W
      movwf tempval
      comf  tempval,W
skipagain
      movwf curpos            ;save as new curpos
      movwf PORTD             ;write new curpos to PORTD computer
      bcf   INTCON,1          ;clear INT bit
      bsf   INTCON,3          ;set RB change interupt
      bsf   INTCON,4          ;set INT interupt
      bsf   INTCON,7

      retfie                  ;return from interupt
;
;
;
makefstep
      movlw 0x5               ;first in coil engergizing sequence
                              ;to step motor in one direction
      movwf PORTC
      call  calldelay         ;delay while motor coils energize
      movlw 0x9               ;next coil sequence
      movwf PORTC
      call  calldelay
      movlw 0xa               ;next coil sequence
      movwf PORTC
      call  calldelay
      movlw 0x6               ;final coil sequence
      movwf PORTC
      call  calldelay
      movlw 0                 ;turn off coils, may be require
                              ;coils energized in some situations
      movwf PORTC
      return                  ;return from call
;
;     move stepper one step in reverse direction
;
makebstep
      movlw 0x5         ;reverse the order of coils energized
      movwf PORTC
      call  calldelay
      movlw 0x6
```

```
      movwf PORTC
      call  calldelay
      movlw 0xa
      movwf PORTC
      call  calldelay
      movlw 0x9
      movwf PORTC
      call  calldelay
      movlw 0
      movwf PORTC
      return

calldelay                     ;delay routine
      movlw 0xe8              ;value drived from experimentation
                              ;any less stepper wont step
      movwf TMR1H             ;load timer 1 high byte
      movlw 0                 ;load timer0 low byte
      movwf TMR1L
      bcf   PIR1,0            ;clear TMR1 flag bit
      bsf   STATUS,RP0        ;change register page
      bcf   PIE1,0            ;just changed
      bcf   STATUS,RP0        ;change register page
      bsf   T1CON,0           ;enable timer1 to start counting
      bcf   INTCON,5          ;disable TMR0 interupt?

delayloop
      btfss PIR1,0            ;poll for time out
      goto  delayloop
      bcf   T1CON,0           ;disable TMR1
      bcf   PIR1,0            ;clear TMR1 flag bit
      bcf   INTCON,5
      return                  ;return from delay

      end
```

```
'
'     VB Program to give basic communication with the PIC
'
' Declare I/O routine so NT will allow access to the ports
'
Private Declare Function OutPortB Lib "ACCES32" Alias "VBOutPortB"
(ByVal Port As Long, ByVal Value As Byte) As Integer
Private Declare Function InPortB Lib "ACCES32" Alias "VBInPortB" (ByVal
Port As Long) As Integer
'
Dim getend As Boolean
'
' Command1 button Click
'
' clicked when a setpoint wants to be sent to the PIC
'
Private Sub Command1_Click()
'
    Dim newval As Byte
'
' disable this timer
'
    Timer1.Enabled = False
'
' set /INIT bit high
'
    Call OutPortB(&H37A, &HF)
'
' get new setpoint value from TEXT3 textbox
'
    newval = CByte(Int(Text3.Text))
'
' write out setpoint value
'
    Call OutPortB(&H378, newval)
'
' start Timer2 giving some time for the PIC to read the setpoint value
'
    Timer2.Enabled = True
'
End Sub
'
' On Form load set LPT port as input
'
Private Sub Form_Load()

    Call OutPortB(&H37A, &HF0)
    getend = False

End Sub
'
' Timer1 time out event
'
Private Sub Timer1_Timer()
```

```
    Dim tempstring As String
'
' read value from PIC
'
    i = InPortB(&H378)
'
' if value if 0FFH then the next number will be terminal count and
should be
' store in TEXT1 textbox
'
    If i = &HFF Then
        getend = True   'getend is a flag
    End If
'
' this gets a normal current position value and stores value in TEXT2
textbox
'
    If i <> &HFF And Not (getend) Then
        Text2.Text = i
    End If
'
' gets terminal count and stores in TEXT1 textbox
'
    If i <> &HFF And getend Then
        Text1.Text = i
        getend = False
    End If

End Sub
'
' generate a time interval to allow the PIC to read the setpoint value
then change
' LPT back to an input port
'
Private Sub Timer2_Timer()

    Timer2.Enabled = False
    Call OutPortB(&H37A, &HF0)
    Timer1.Enabled = True

End Sub;
```

Analog Input Example Program

```
;     Created by Bruce
;
;     Designed to run on the PIC-DEM2
;
;
;     Uses 16F877 MicroController at 10 Mhz
;
;     IO Pin connections
;
;
;     Port A (ra)
;
;     pin 0 (ra0) - Analog input
;
;     Define Part
;
      list p=16f877
      include "p16f877.inc"
;
;
TEMP  set   0x20        ;define temporary register
;
;
;
;     Reset Vector
;
      org   0x0
;
;
      goto Start        ;reset vector
;
;
;     Interupt Vector
;
      org   0x4         ;interupt vector
      goto intvect
;
;
Start

      bcf   STATUS,RP1  ;select register 1
      bsf   STATUS,RP0
;
;
      movlw B'11111111' ;define PORTA as input
      movwf TRISA
      movlw B'00000000' ;define PORTB as output
      movwf TRISB
;
;
      bcf   STATUS,RP0  ;select register page 0
;
      bcf   PIR1,ADIF   ;clear A/D interupt flag bit
;
```

```
      bsf   STATUS,RP0  ;select register 1
      bsf   PIE1,ADIE   ;set A/D interupt enable bit
      bcf   STATUS,RP0  ;select register page 0
;
      movlw 0           ;initialize PORTB as 0
      movwf PORTB
      movwf ADCON1      ;8 most significant bits in ADRESH
                        ;all inputs are analog
;
;
      movlw 0x01        ;A/D clock is Fosc/2
                        ;select RA0 for input
                        ;ADON bit set
      movwf ADCON0
;
      bsf   INTCON,PEIE ;enable peripheral interupts
      bsf   INTCON,GIE  ;enable global interupts
;
      call  SetupDelay  ;call delay subroutine
;
      bsf   ADCON0,GO   ;start first A/D conversion
;
;
;     Main Program loop
;
;
loop
      goto loop
;
;
;     Interupt Service Routine
;
;
intvect
      btfss PIR1,ADIF   ;check if A/D caused the interupt
      goto  endint            ;if not end routine
      movf  ADRESH,W    ;read upper 8 bits of A/D
      movwf PORTB       ;send to PORTB
      bcf   PIR1,ADIF   ;clear interupt flag bit
      bsf   INTCON,PEIE ;reset peripheral interupts
      call  SetupDelay  ;cal delay twice 2Tad
      call  SetupDelay
      bsf   ADCON0,GO   ;start new conversion
;
endint
;
      retfie                  ;end interupt service routine
;
;
SetupDelay
      movlw   .3              ; Load Temp with decimal 3
      movwf   TEMP
SD
      decfsz  TEMP, F         ; Delay loop
      goto    SD
      return
```

```
;
      end               ;end program
```