Bargraph Display

Adapted from Prof. Martin D. Fox’s ECE266 slides
Introduction

• Review of first project
• Second project
• You will learn to use the A/D converter, output to port D; create bar graph display
int main()
{
    Initial();
    for (;;)
    {
        Blink();
        LoopTime();
    }
}
Blink.c

```c
void Initial()
{
    ADCON1 = 0x04;  // Select PORTA pins for ADC or digital I/O
    TRISA = 0x0B;   // Set I/O for PORTA
    TRISB = 0x0F;   // Set I/O for PORTB
    TRISC = 0xB7;   // Set I/O for PORTC
    TRISD = 0x00;   // Set I/O for PORTD
    TRISE = 0x04;   // Set I/O for PORTE

    PR2 = 95;       // Set up Timer2 for a loop time of 10 ms

    PORTD = 0;      // Turn off LEDs
    T2CON = 0x5F;   // Finish set up of Timer2

    INTCON = 0xD0;  // Enable RB0/INT interrupts
}
```
Timer 2

FIGURE 7-1: TIMER2 BLOCK DIAGRAM

Note 1: TMR2 register output can be software selected by the SSP module as a baud clock.

1/7.37MHz ~ 0.136 uS

4 X / 7.37MHz = 10 mS;

X = 18425

How close can we come with timer 2?
T2CON

REGISTER 7-1: T2CON: TIMER2 CONTROL REGISTER (ADDRESS 12h)

<table>
<thead>
<tr>
<th>U-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TOUTPS3</td>
<td>TOUTPS2</td>
<td>TOUTPS1</td>
<td>TOUTPS0</td>
<td>TMR2ON</td>
<td>T2CKPS1</td>
<td>T2CKPS0</td>
</tr>
</tbody>
</table>

bit 7: **Unimplemented**: Read as ‘0’

bit 6-3: **TOUTPS3**: Timer2 Output Postscale Select bits
- 0000 = 1:1 Postscale
- 0001 = 1:2 Postscale
- 0010 = 1:3 Postscale
- ...
- 1111 = 1:16 Postscale

bit 2: **TMR2ON**: Timer2 On bit
- 1 = Timer2 is on
- 0 = Timer2 is off

bit 1-0: **T2CKPS1**: Timer2 Clock Prescale Select bits
- 00 = Prescaler is 1
- 01 = Prescaler is 4
- 1x = Prescaler is 16

R = Readable bit
W = Writable bit
U = Unimplemented bit, read as ‘0’
- n = Value at POR reset
### REGISTER 2-3: INTCON REGISTER (ADDRESS 0Bh, 8Bh, 10Bh, 18Bh)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Function Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td><strong>GIE</strong>: Global Interrupt Enable bit</td>
<td>1 = Enables all unmasked interrupts, 0 = Disables all interrupts</td>
</tr>
<tr>
<td>6</td>
<td><strong>PEIE</strong>: Peripheral Interrupt Enable bit</td>
<td>1 = Enables all unmasked peripheral interrupts, 0 = Disables all peripheral interrupts</td>
</tr>
<tr>
<td>5</td>
<td><strong>TMR0IE</strong>: TMR0 Overflow Interrupt Enable bit</td>
<td>1 = Enables the TMR0 interrupt, 0 = Disables the TMR0 interrupt</td>
</tr>
<tr>
<td>4</td>
<td><strong>INTE</strong>: RB0/INT External Interrupt Enable bit</td>
<td>1 = Enables the RB0/INT external interrupt, 0 = Disables the RB0/INT external interrupt</td>
</tr>
<tr>
<td>3</td>
<td><strong>RBIE</strong>: RB Port Change Interrupt Enable bit</td>
<td>1 = Enables the RB port change interrupt, 0 = Disables the RB port change interrupt</td>
</tr>
<tr>
<td>2</td>
<td><strong>TMR0IF</strong>: TMR0 Overflow Interrupt Flag bit</td>
<td>1 = TMR0 register has overflowed (must be cleared in software), 0 = TMR0 register did not overflow</td>
</tr>
<tr>
<td>1</td>
<td><strong>INF</strong>: RB0/INT External Interrupt Flag bit</td>
<td>1 = The RB0/INT external interrupt occurred (must be cleared in software), 0 = The RB0/INT external interrupt did not occur</td>
</tr>
<tr>
<td>0</td>
<td><strong>RBIF</strong>: RB Port Change Interrupt Flag bit</td>
<td>1 = At least one of the RB7:RB4 pins changed state; a mismatch condition will continue to set the bit. Reading PORTB will end the mismatch condition and allow the bit to be cleared (must be cleared in software), 0 = None of the RB7:RB4 pins have changed state</td>
</tr>
</tbody>
</table>
Blink.c

void Blink()
{
    unsigned char temp;
    if (blnkcnt-- == 0) {
        blnkcnt = MaxCount;

        temp = PORTD;   // Copy present state of LEDs to temp
        temp = temp & 0xE0;   // mask off top bits

        if ( temp == 0x40 )
            temp = 0xC0;   // if bit 6 was set, now set bit 7
        else if ( temp == 0x20 )
            temp = 0x60;   // if bit 5 was set, now set bit 6
        else
            temp |= 0x20;   // for everything else, set bit 5

        PORTD ^= temp;   // set LEDs to new value
    }
}
Blink.c

LoopTime()
{
    for (;;) {
        if ( TMR2IF ) { // Check whether ten milliseconds are up
            TMR2IF=0;  // Clear flag
            RA5 ^= 1;  // Toggle PORTA, bit 5
            return;
        }
    }
}
void interrupt handler(void) {
    if ( INTF ) { // Test RB0/INT flag
        INTF = 0; // Clear RB0/INT flag
        INTEDG ^= 0x40; // Toggle to catch opposite edge
        RD1 = RC0; // Copy Inc/Dec "direction" to
        PORTD, 1
        RD0 = RB0; // Copy interrupt signal to
        PORTD, 0
    }
}

Analog/Digital Conversion

A/D conversion is the process of sampling a continuous signal

Two significant implications

1. The information content of the sampled signal is less than the continuous signal

   The continuous signal contains an infinite number of independent samples, the sampling process reduces that to a finite number of independent samples

2. Uncertainty is added to the sampled data.

   Quantization error is part of the sampling process since the number of intervals is finite. This is analogous to truncating a number after a specific number of places
Analog-to-digital converters

Proportionality

Analog-to-digital

Digital to analog

Embedded Systems Design: A Unified Hardware/Software Introduction, (c) 2000 Vahid/Givargis
Analog/Digital Conversion

Example

Number of bits = 3
Number of intervals = $2^3$
Range = 0-10 volts
Resolution = 1.25 volts

Quantization error = +/- 0.625 volts
Variance = $(1.25)^2/12 = 0.130$ volts$^2$
Proportional Signals

Simple Equation

Assume minimum voltage of 0 V.

\( V_{\text{max}} = \) maximum voltage of the analog signal

\( a = \) analog value

\( n = \) number of bits for digital encoding

\( 2^n = \) number of digital codes

\( M = \) number of steps, either \( 2^n \) or \( 2^n - 1 \)

\( d = \) digital encoding

\( a / V_{\text{max}} = d / M \)
DAC vs. ADC

DAC:
- **n** digital inputs for digital encoding **d**
- analog input for \( V_{\text{max}} \)
- analog output **a**

ADC:

Given a \( V_{\text{max}} \) analog input and an analog input **a**, how does the converter know what binary value to assign to **d** in order to satisfy the ratio?

- Try and build circuitry that converts directly from **a** to **d**
  - Not easy
- Use a guessing mechanism
  - Use DAC to generate analog values for comparison with **a**
  - ADC “guesses” an encoding **d**, then checks its guess by inputting **d** into the DAC and comparing the generated analog output \( a' \) with original analog input **a**
  - How does the ADC guess the correct encoding?
Bit Weight

Each bit is weighted with an analog value, such that a 1 in that bit position adds its analog value to the total analog value represented by the digital encoding.

**Example:** -5 V to +5 V analog range, n=8

<table>
<thead>
<tr>
<th>Digital Bit</th>
<th>Bit Weight (V)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>10/2 = 5</td>
</tr>
<tr>
<td>6</td>
<td>10/4 = 2.5</td>
</tr>
<tr>
<td>5</td>
<td>10/8 = 1.25</td>
</tr>
<tr>
<td>4</td>
<td>10/16 = 0.625</td>
</tr>
<tr>
<td>3</td>
<td>10/32 = 0.313</td>
</tr>
<tr>
<td>2</td>
<td>10/64 = 0.157</td>
</tr>
<tr>
<td>1</td>
<td>10/128 = 0.078</td>
</tr>
<tr>
<td>0</td>
<td>10/256 = 0.039</td>
</tr>
</tbody>
</table>
A/D on the PIC

- Controlled by ADCON0 and ADCON1 registers
- Returns a 10-bit value in the ADRESH and ADRESL registers
- Data acquisition must settle - roughly 30 us after A/D is turned on
Analog-to-Digital Converter

- ADCS bits set A/D clock period $T_{AD}$
  - $T_{AD}$ must be at least $1.6\mu s$
  - 7.37 MHz clock equals $0.136\mu s \times 16 = 2.17\mu s$
  - ADCS0 = 1, ADCS1 = 0, ADCS2 = 1

- CHS bits select which analog input
  - Top potentiometer is connected to the RA1/AN1 input on the PIC
  - Set CHS2,CHS2,CHS0 to “001”
Analog-to-Digital Converter

- Set the ADCON0 register to 0x48:
  \[ \text{ADCON0} = 0x48; \]

- Set the ADCS0 and CHS0 bits to enable the ADC:
  \[ \text{ADCS0} = 1; \]
  \[ \text{CHS0} = 1; \]

- Turn on A/D by setting the ADON bit:
  \[ \text{ADON} = 1; \]

- Wait for roughly 30us:
  \[ \text{for (i=15; i; i--);} \]

- Start conversion:
  \[ \text{ADGO} = 1; \]

- Wait for conversion to complete:
  \[ \text{while (ADGO);} \]
ADCON1: set port pins

- ADCS2 bit sets A/D clock period
- ADFM decides ADRES justification
- CFG bits select configuration = “0100”
New BarTable

;;;;;;; BarTable subroutine

; This subroutine reads ADRESH and retains only the upper three bits. It
; uses them to access the table. It returns in W the bits to be output to PORTD
; which will be displayed as a bar chart.

BarTable
    movf ADRESH,W       ; move adresh to w
    btfsc STATUS,Z      ; Z bit =1 if result=0
    goto ZeroResult

    MOVFF ADRESH,TEMP   ; copy present state ADRES to F
    swapf TEMP,F        ; move bits 7,6,5 to bits 3,2,1
    rrf TEMP,W          ; and on to bits 2,1,0 of W
    andlw B'00000111'   ; keep only bits to be shifted
    addwf PCL,F         ; change PC with PCLATH and offset in W
    retlw B'00000001'    ;
    retlw B'00000011'    ;
    retlw B'00000111'    ;
    retlw B'00001111'    ;
    retlw B'00011111'    ;
    retlw B'00111111'    ;
    retlw B'11111111'    ;

ZeroResult
    retlw B'00000000'    ; return if all zeros in adresh

;
QIK-START wiring AD
Summary

• We have learned how to configure and use the AD converter

• Assignment: in the next lab, Bar Intensity, you will use the second trimpot to control the intensity of the LEDs in the Bar Chart.

• You should start thinking about how you will implement this.