



**Part B:**

We want to run the buzzer at 349 Hz. Recall from the video that

$$\text{PWM period} = (\text{timer\_ticks} + 1)(4)(T_{\text{osc}})(T_{2\text{pre}})$$

We want a PWM period of \_\_\_\_\_ seconds, so we will use

$T_{\text{osc}}$  = period of oscillation of the timer (seconds) = \_\_\_\_\_ seconds or a timer frequency of \_\_\_\_\_ (Hz)

$T_{2\text{pre}}$  = prescaler for the timer = \_\_\_\_\_

timer\_ticks = number of ticks we want the timer (after prescaling) to count between pulses = \_\_\_\_\_

Now take a look at the changes to the code that we have labeled on the attached program. Go ahead and type in those changes. Make sure to ask if you don't understand why you're adding those pieces of code.

Run the code. It should make an annoying noise at a single frequency. Pull the jumper (J9) near the buzzer to make the buzzer be quiet.

**Part C:**

We want to only turn the buzzer on when we push RB0. We've attached a second program listing that shows the changes for this part. Go ahead and type in those changes. Make sure to ask if you don't understand why you're adding those pieces of code.

Make sure that jumper J9 is back on. Run the code. Now the board should only make noise when you push the button RB0.

**Part D:**

Now we want to have the potentiometer adjust the frequency. We already have the code to read the potentiometer and have the value go from 0 to 1023. We would like to have the maximum frequency be 880 Hz (high A) and the lowest frequency be 250 Hz. We need to adjust the number of timer\_ticks to get this frequency range, keeping the other settings the same as in Part C:

Potentiometer Reading (RA0result)	Frequency	timer_ticks?
0	880 Hz	
1023	250 Hz	

If we didn't have to worry about integer math issues, we could use the equation of a straight line to fit these numbers:

$$\text{timer\_ticks} = \text{RA0result} * (\text{_____}) + (\text{_____})$$

Sadly, we do need to worry about integer math, so we need to use something that won't overflow our ints. So we will use

$$\text{timer\_ticks} = \text{RA0result} * \text{_____/_____} + \text{_____}$$

Now that we know what formula to use, we can use `RA0result` in the function to start the buzzer at the appropriate frequency. Add this to your code (see attached code printout), and verify that it works.

Put both jumpers back on the board! Check here when you have done this: \_\_\_\_\_

## Part B

### example\_ADC\_PWM\_PartB.c

```
/* *****  
* FileName:          example ADC.c  
* Processor:        PIC18F4520  
* Compiler:         MPLAB C18 v. 3.06  
*  
* This file uses the LCD display to display the of the pot RA0  
* The AD conversion uses 10 bits and this program prints those bits  
* in decimal  
*  
* Author           Date           Comment  
*-----*/  
// David Fisher    9/9/07  
  
/** Header Files *****/  
#include <p18f4520.h>  
#include "LCD Module.h"  
#include <adc.h>  
#include <stdio.h>  
#include <timers.h> //We need the timers for the PWM  
#include <pwm.h> //The PWM makes the noise on the buzzer  
  
/** Define Constants Here *****/  
  
/** Local Function Prototypes *****/  
void low_isr(void);  
void high_isr(void);  
  
// =====  
// Configuration Bits  
#pragma config OSC = EC // External 4MHz crystal  
#pragma config WDT = OFF  
#pragma config LVP = OFF  
#pragma config BOREN = OFF  
  
/** Declare Interrupt Vector Sections *****/  
#pragma code high_vector=0x08  
void interrupt_at_high_vector(void)  
{  
    _asm goto high_isr _endasm  
}  
  
#pragma code low_vector=0x18  
void interrupt_at_low_vector(void)  
{  
    _asm goto low_isr _endasm  
}  
  
/** Global Variables *****/  
char line1[17];  
char line2[17];  
int RA0result, RB0result;  
  
/* *****  
* Function:         void main(void)  
* *****/  
#pragma code  
  
void main (void)  
{  
    XLCDInit();
```

example\_ADC\_PWM\_PartB.c

```
XLCDClear();

// configure A/D convertor
// config 1 = Setup the timing to a conservative value (you don't need to ever
change this)
// config 2 = Use channel 0, not interrupts off, use the power and ground as
references
// portconfig = 0x0E setup only analog 0 as a possible analog input pin

OpenADC(ADC_FOSC_8 & ADC_RIGHT_JUST & ADC_12_TAD,
        ADC_CH0 & ADC_INT_OFF & ADC_REF_VDD_VSS,
        0x00);

// This library function simply set the following SFR's
// ADCON0 = 0x01;          ///### Turn On ADC
// ADCON1 = 0x0E;          ///### Select Vref+ = VDD, Vref- = VSS, AN0 = Analog
Input
// ADCON2 = 0xA9;          ///### Acquisition delay 12 TAD, A/D conversion clock
8TOSC, Right Justified

// Play 349 Hz all the time
// Open Timer 2 with a 1:16 prescaler
OpenTimer2 (TIMER_INT_OFF & T2_PS_1_16);
// Open the PWM with 178 Timer Ticks from the formula
OpenPWM1(178);
// Set the duty cycle (affects volume) to 10% (10% of 1023)
SetDCPWM1(102);

while (1)
{
    SetChanADC( ADC_CH0 );          // Select the pin
    ConvertADC();                  // Start conversion
    while( BusyADC() );            // Wait for completion
    RA0result = ReadADC();         // Read result

    sprintf(Line1, "RA0 ADC -> %#4u", RA0result);
    XLCDL1home();
    XLCDPutRamString(Line1);

    SetChanADC ( ADC_CH12 );
    ConvertADC();
    while( BusyADC() );
    RB0result = ReadADC();

    sprintf(Line2, "RB0 ADC -> %#4u", RB0result);
    XLCDL2home();
    XLCDPutRamString(Line2);
}
}

/*****
* Function:      void high_isr(void)
* Input:
* Output:
* Overview:
*****/
#pragma interrupt high_isr          // declare function as high priority
ISR
void high_isr(void)
{
}
```

example\_ADC\_PWM\_PartB.c

```
/******  
* Function:          void Low_isr(void)  
* Input:  
* Output:  
* Overview:  
*****/  
#pragma interruptlow Low_isr          // declare function as low priority isr  
void Low_isr(void)  
{  
}
```

## Part C

### example\_ADC\_PWM\_PartC.c

```
/* *****  
* FileName:          example ADC.c  
* Processor:        PIC18F4520  
* Compiler:         MPLAB C18 v. 3.06  
*  
* This file uses the LCD display to display the of the pot RA0  
* The AD conversion uses 10 bits and this program prints those bits  
* in decimal  
*  
* Author            Date            Comment  
*-----*/  
// David Fisher      9/9/07  
  
/** Header Files *****/  
#include <p18f4520.h>  
#include "LCD Module.h"  
#include <adc.h>  
#include <stdio.h>  
#include <timers.h>           //We need the timers for the PWM  
#include <pwm.h>             //The PWM makes the noise on the buzzer  
  
/** Define Constants Here *****/  
#define PUSHBUTTON_THRESH 500  
  
/** Local Function Prototypes *****/  
void low_isr(void);  
void high_isr(void);  
void startTheBuzzer(void);  
void stopTheBuzzer(void);  
  
// =====  
// Configuration Bits  
#pragma config OSC = EC // External 4MHz crystal  
#pragma config WDT = OFF  
#pragma config LVP = OFF  
#pragma config BOREN = OFF  
  
/** Declare Interrupt Vector Sections *****/  
#pragma code high_vector=0x08  
void interrupt_at_high_vector(void)  
{  
    _asm goto high_isr _endasm  
}  
  
#pragma code low_vector=0x18  
void interrupt_at_low_vector(void)  
{  
    _asm goto low_isr _endasm  
}  
  
/** Global Variables *****/  
    char line1[17];  
    char line2[17];  
    int RA0result, RB0result;  
  
/* *****  
* Function:          void main(void)  
* *****/  
#pragma code  
  
void main (void)
```

example\_ADC\_PWM\_PartC.c

```

{
  XLCDInit();
  XLCDClear();

  // configure A/D convertor
  // config 1 = Setup the timing to a conservative value (you don't need to ever
change this)
  // config 2 = Use channel 0, not interrupts off, use the power and ground as
references
  // portconfig = 0x0E setup only analog 0 as a possible analog input pin

  OpenADC(ADC_FOSC_8 & ADC_RIGHT_JUST & ADC_12_TAD,
          ADC_CH0 & ADC_INT_OFF & ADC_REF_VDD_VSS,
          0x00);

  // This library function simply set the following SFR's
  // ADCON0 = 0x01;      //### Turn On ADC
  // ADCON1 = 0x0E;      //### Select Vref+ = VDD, Vref- = VSS, AN0 = Analog
Input
  // ADCON2 = 0xA9;      //### Acquisition delay 12 TAD, A/D conversion clock
8TOSC, Right Justified

  // Play 349 Hz all the time
  // Open Timer 2 with a 1:16 prescaler
  OpenTimer2 (TIMER_INT_OFF & T2_PS_1_16);
  // Open the PWM with 178 Timer Ticks from the formula
  OpenPWM1(178);
  // Set the duty cycle (affects volume) to 10% (10% of 1023)
  SetDCPWM1(102);

  while (1)
  {

    SetChanADC( ADC_CH0 );          // Select the pin
    ConvertADC();                  // Start conversion
    while( BusyADC() );            // Wait for completion
    RA0result = ReadADC();         // Read result

    sprintf(Line1, "RA0 ADC -> %#4u", RA0result);
    XLCDL1home();
    XLCDPutRamString(Line1);

    SetChanADC ( ADC_CH12 );
    ConvertADC();
    while( BusyADC() );
    RB0result = ReadADC();

    sprintf(Line2, "RB0 ADC -> %#4u", RB0result);
    XLCDL2home();
    XLCDPutRamString(Line2);

    if (RB0result < PUSHBUTTON_THRESH)
    {
      startTheBuzzer();
    }
    else
    {
      stopTheBuzzer();
    }
  }
}
/*****

```

example\_ADC\_PWM\_PartC.c

```
// Starts the buzzer
// FUNCTIONS MUST GO BEFORE INTERRUPTS IN THE CODE
*****/
void startTheBuzzer()
{
    //Make the duty cycle 10%
    SetDCPWM1(102);
}
/*****
// Stops the buzzer
// FUNCTIONS MUST GO BEFORE INTERRUPTS IN THE CODE
*****/
void stopTheBuzzer()
{
    //Make the duty cycle 0%
    SetDCPWM1(0);
}
/*****
* Function:      void high_isr(void)
* Input:
* Output:
* Overview:
*****/
#pragma interrupt high_isr           // declare function as high priority
ISR
void high_isr(void)
{
}

/*****
* Function:      void low_isr(void)
* Input:
* Output:
* Overview:
*****/
#pragma interrupt low_isr           // declare function as low priority
ISR
void low_isr(void)
{
}
```

## Part D

### example\_ADC\_PWM\_PartD.c

```
// Starts the buzzer
// FUNCTIONS MUST GO BEFORE INTERRUPTS IN THE CODE
*****/
void startTheBuzzer()
{
    int timer_ticks;
    // Re-open the PWM with a frequency based on the
    // potentiometer ASC value. Watch the integer math.
    timer_ticks=70+RA0resul t*32/183;
    OpenPWM1(timer_ticks);
    //Make the duty cycle 10%
    SetDCPWM1(102);
}
/*****
// Stops the buzzer
// FUNCTIONS MUST GO BEFORE INTERRUPTS IN THE CODE
*****/
void stopTheBuzzer()
{
    //Make the duty cycle 0%
    SetDCPWM1(0);
}
/*****
* Function:      void hi gh_i sr(voi d)
* Input:
* Output:
* Overview:
*****/
#pragma interrupt hi gh_i sr          // declare functi on as hi gh pri ori ty
i sr
void hi gh_i sr(voi d)
{
}

/*****
* Function:      void low_i sr(voi d)
* Input:
* Output:
* Overview:
*****/
#pragma interrupt low low_i sr      // declare functi on as low pri ori ty i sr
void low_i sr(voi d)
{
}
```