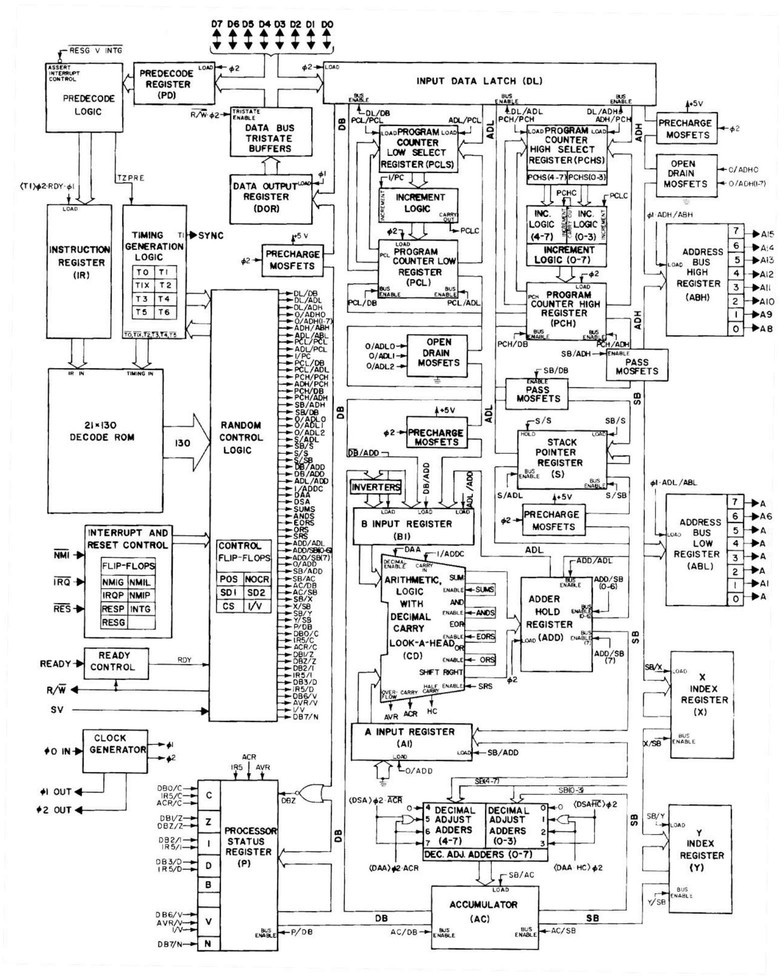
CE 167 - Sensing & Sensor Technologies

Lab 0: Hello World & Speaker Tones

By: Kyle Jeffrey



D7 D6 D5 D4 D3 D2 D1 D0
INPUT DATA LATCH (DL)
PREDECODE REGISTER (PD)
PREDECODE LOGIC
DATA BUS TRISTATE BUFFERS
DATA OUTPUT REGISTER (DOR)
INSTRUCTION REGISTER (IR)
TIMING GENERATION LOGIC
21×130 DECODE ROM
RANDOM CONTROL LOGIC
INTERRUPT AND RESET CONTROL
READY CONTROL
CLOCK GENERATOR
PROCESSOR STATUS REGISTER (P)
PROGRAM COUNTER LOW SELECT REGISTER (PCLS)
INCREMENT LOGIC
PROGRAM COUNTER LOW REGISTER (PCL)
PROGRAM COUNTER HIGH SELECT REGISTER (PCHS)
PROGRAM COUNTER HIGH REGISTER (PCH)
PRECHARGE MOSFETS
OPEN DRAIN MOSFETS
PASS MOSFETS
STACK POINTER REGISTER (S)
INVERTERS
B INPUT REGISTER (BI)
ARITHMETIC LOGIC WITH DECIMAL CARRY LOOK-A-HEAD (CD)
ADDER HOLD REGISTER (ADD)
A INPUT REGISTER (AI)
DECIMAL ADJUST ADDERS (4-7)
DECIMAL ADJUST ADDERS (0-3)
ACCUMULATOR (AC)
ADDRESS BUS HIGH REGISTER (ABH)
ADDRESS BUS LOW REGISTER (ABL)
X INDEX REGISTER (X)
Y INDEX REGISTER (Y)
NMI
IRQ
RES
READY
R/W
SV
φ0 IN
φ1 OUT
φ2 OUT

Introduction

The Lab introduced the PIC 32 Microcontroller and basic circuit design. Several circuits were made using a speaker to create different frequencies using onboard peripherals like buttons and a potentiometer.

Part 1: Hello World

Write a program that can print out “Hello World!” in serial on the PIC32 board.

Procedure

Use the built in serial “printf()” function provided by the stdio legacy library. The class also provides the library for initializing the PIC32 board for use using the provided BOARD.h and BOARD.c files. Calling “BOARD_Init()” will initialize the board and then the program is a simple printf. I will not detail this in subsequent parts as every PIC32 program follows this convention.

Pseudo-Code

```
#include BOARD

#include stdio

main(){

        BOARD_INIT()

        printf(“Hello World@”)

        while(1);

        Return 0

}
```

Notice the use of a while(1) in hardware programming. PIC32’s have no OS so will fall off, crash and restart if main returns.

Part 2: Hello A/D

Print the Audio to Digital reading from the potentiometer on the microcontroller shield using the AD library.

Procedure

Use the AD and OLED libraries to print readings from the potentiometer on to the OLED screen. The AD and OLED both require initialize functions using OledInit() and AD_Init(). The potentiometer uses the AD_A0 analog to digital pin to run so use the AddPin() function to add AD_A0. To continually update the potentiometer digital reading, use the function ReadADPin() on AD_A0 to receive the unsigned 10bit digital value which reaches max value 1023.

For the Oled use functions DrawString() and update() in tandem to continually update the screen within the main loop. The int value of the potentiometer just needs to be converted to a char* before drawing to the Oled. Use sprintf() to accomplish this.

Notice that the potentiometer reading can be sporadic. Later this will be accounted for.

Pseudo-Code

```
#include BOARD
#include AD
#include Oled
main(){
    //Initialize all the peripherals and BOARD
    while(1){
        Pot = ReadADPin(AD_A0) //Convert Pot to char*
        OledDrawString(Pot)
        OledUpdate()
    }
}
```

Part 3: Tone Out Speaker Hard-Coded

Generate a tone with the speaker using the tone generator library.

Procedure

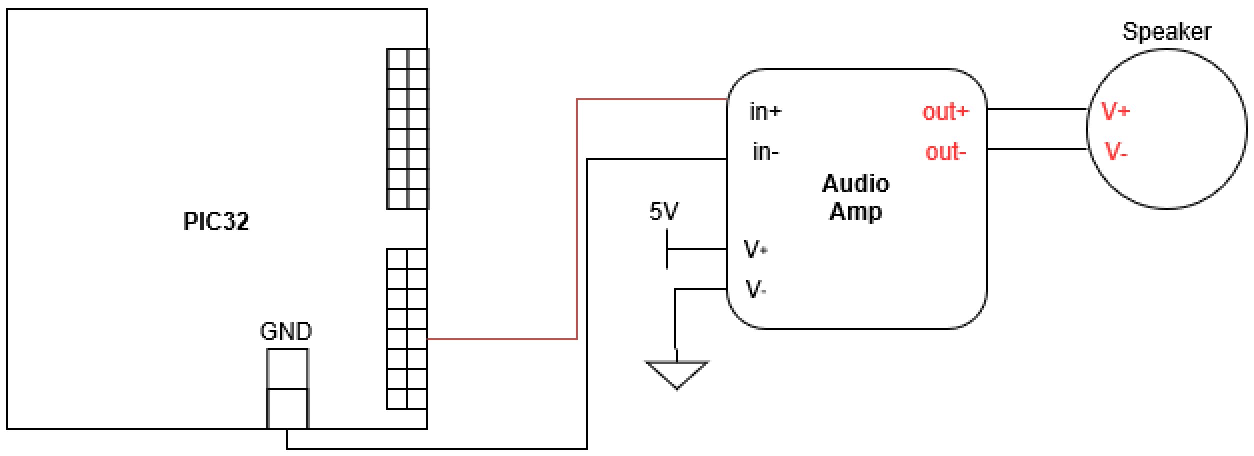
Use the ToneGeneration Library to create a pwm frequency to play out through the speaker. The PIC32 generates a pwm through pin 3 on the board but it is low voltage which needs to be amplified by an audio amplifier. Consult the included diagram for wiring the speaker circuit to the PIC32.

The ToneGeneration Library has an initialize function as well as functions for setting the frequency and turning the output on. These all need to be used before pin 3 begins emitting a pwm signal. There are defined tones in the ToneGeneration library that can be used.

Pseudo-Code

```
#include ToneGeneration
#include BOARD
main(){
    //Initialize board and ToneGeneration
    ToneGenerationSetFrequency(TONE)
    ToneGenerationOn()
    while(1);
    Return 0;
}
```

Schematic



Part 4: Tone Adjusted via Potentiometer

Now use the potentiometer A/D reading to change the tone of the speaker

Procedure

This part uses the potentiometer digital reading to change the frequency of the speaker. The circuit will be the same as the previous part. PIN 3 on the PIC32 IO Shield is always used to generate the pwm for the frequency output in this lab. This time we will be putting part 2 and 3 together so initialize both the peripherals for the AD library and the ToneGeneration. The potentiometer reading will be directly inputted into the setFrequency() function.

Notice once this program is implemented that the audio is very scratchy. This is because the potentiometer is very jump and switches readings quickly. To combat this, add a minimum change in reading before the frequency is changed. 3 is a good place to start.

Pseudo-Code

```
#include ToneGeneration
#include BOARD
#include AD
main(){
    //Initialize board, ToneGeneration, and AD
    ToneGenerationOn()
    newPot = readADPin(AD_A0);
    while(1){
        newPot = readADPin(AD_A0);
        if(newPot-oldPot > 3){
            oldPot = newPot
            setFrequency(OldPot)
        }
    }
    Return 0;
}
```

Part 5: Basic Filtering of Output

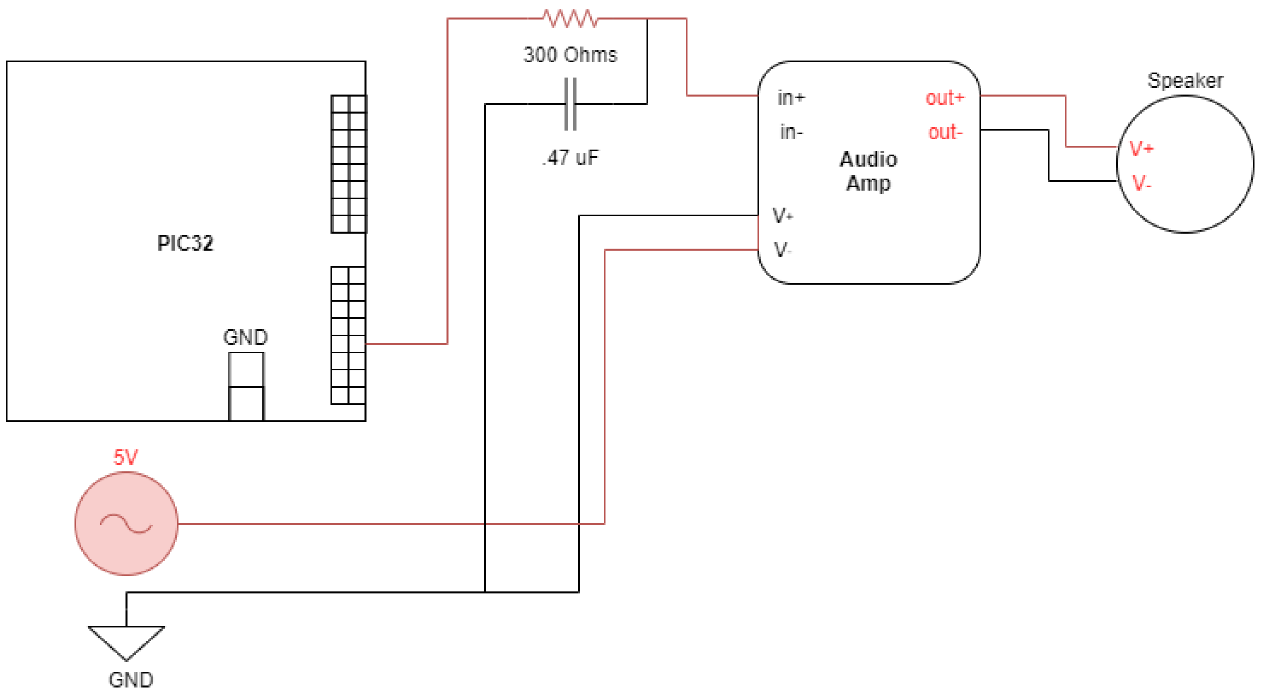
Use a low pass filter using an RC circuit to filter out high frequencies that may be causing some distortion to the speaker output. Use the same program from the last part.

Procedure

Use the same program from part 4 on the board. This time though, wire the signal from the audio amplifier through the RC low pass filter circuit. An example of a low pass filter would be a resistor of 300 ohms and capacitor of .47 uF. This gives a cutoff frequency of 1129 Hz using the equation 1/2piRC.

Notice that the speaker now sounds smoother. This is because higher unwanted frequencies have now been filtered out. Also notice with an oscilloscope that the square wave has smoothed out on the corners because of the capacitors resistance to voltage change.

Schematic



Part 6: Music Board

Build an instrument to play music with your now harmonious assembly of microcontroller, low-pass filter,audio amp, and speaker. The requirements for the instrument are below. Map four tones to the four buttons on the I/O Shield.

Procedure

The circuit will be the same from part 5. We now use all of the work from the previous five parts to make an "instrument" with the board. In actuality, four of the buttons will be harnessed using the BOARD library and the potentiometer will be used as in previous parts.

The code should be similar to part 4 in its use of the potentiometer and the software filtering of the signal. Then use either a switch or series of if else statements to handle the behavior of the buttons, with the BUTTON_STATES defined in BOARD library, producing different signals with an offset provided by the potentiometer. Divide the digital reading of the potentiometer by ten to assure that the frequency is not too high or low to hear.

Pseudo-Code

```
#include ToneGeneration
#include BOARD
#include AD
main(){
    //Initialize board, ToneGeneration, and AD
    newPot = readADPin(AD_A0);
    while(1){
        newPot = readADPin(AD_A0);
        if(newPot-oldPot > 3){
            oldPot = newPot
        }
        switch(BUTTON_STATES)
        case(1000):
        setFrequency(TONE0 + oldPot/10)
        Break;
        case(0100):
        setFrequency(TONE1 + oldPot/10)
        Break;
        ...
    }
    Return 0;
}
```

Conclusion

The lab began introduced the necessity of software AND hardware filter for signals. High grade peripherals are not a guarantee so these techniques are important in creating desired outputs and behavior with systems.