Lab 5 Report: Digital Audio

1 Introduction

In this lab, I used the PIC32 on the μ Mudd32 to play a song that was loaded onto the PIC's flash memory. Two song segments were programmed into the PIC: Fur Elise and Reluctant Hero (from the anime, Attack on Titan). The PIC generated a square wave and drove the speaker through a LM380 audio amplifier.

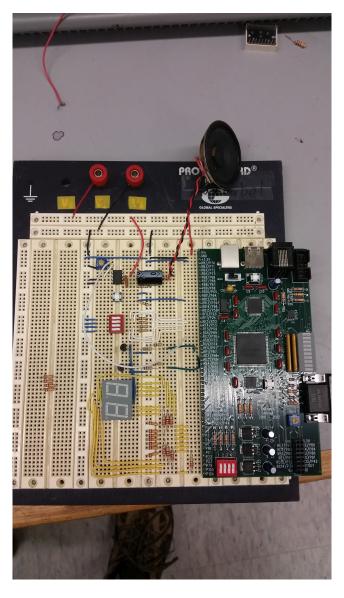


Figure 1: Complete development board with audio buffer / amplifier at the top of the breadboard.

2 Design and Testing Methodology

2.1 Generating Waves

Ideally, the speaker will be driven by a sine wave so that the output tone will be a clean note. However, generating a sine wave is difficult. It can be done through a DAC (digital-to-analog converter) but this is an extra IC that I would need to interface with. Instead, a square wave with the desired frequency can be used to drive the speaker. This is easy to implement and only requires toggling between 1 and 0 on an output pin.

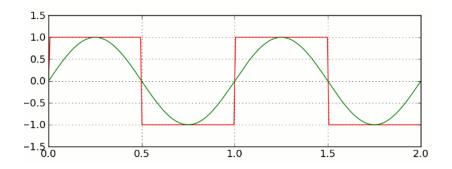


Figure 2: Square wave approximation of a sine wave. Source: http://upload.wikimedia.org/wikipedia/commons/f/f8/SquareWave.gif

2.2 Algorithm

The music playing algorithm is as follows:

- 1. Setup timer 1
- 2. Setup timer 2
- 3. Load first period and duration from flash memory. Parse
- 4. While the note duration has not been met, play the note
- 5. Repeat from step 3 with new note.

Here is some pseudocode / C-code that describes the method in slightly more detail.

```
1
   main(){
2
        // setup timers
                                            // x = don't care
3
        T1CON = 11xx_xxxx_xx11_xxx0;
        T2CON = 11xx_xxxx_x010_xx0x;
4
5
        // setup outputs
6
        TRISG = 1111_{1111_{0111_{1111}}};
7
                                            // make port 7 an output
8
        // start playing the song
9
        int i = 0;
10
        while (true) {
11
12
            int word = \_notes[i];
13
```

```
// check for end of song
14
            if (word = 0)
15
16
                 break;
17
            }
18
            int halfPeriod = word[15:8];
19
            int duration = word[7:0];
20
21
            // play for duration
22
23
            TMR1 = 0
24
            TMR2 = 0
            while (TMR1 < duration) {
25
26
                 //toggle according to the period
                 if (TMR2 >= halfPeriod){
27
                     //toggle output
28
                     PORTG[7] = PORTG[7];
29
30
                     TMR2 = 0;
31
32
             //move to next note
33
34
             i++;
35
36
   }
```

2.3 Timing

The clock on the μ Mudd32 operates at 40Hz. The prescalar for the PIC's peripheral clock was set to 1:8. This means that the peripheral clock will run at $\frac{40MHz}{8} = 5MHz$.

Timer 1's prescalar is set to 1:256. This makes this timer run at $\frac{5MHz}{256} = 19.5kHz$ or a period of $5.12\mu s$. This timer is used to keep track of the duration of each note. Let each count of timer 1 be τ_0 . The duration, τ_0 , of each type of note is given in Figure 3.

Timer 2's prescalar is set to 1:4. This allows the timer to run at $\frac{5MHz}{4} = 1.25MHz$ or a period of $0.8\mu s$. Timer 2 is used to keep track of the period of each note. Let each count of timer 1 be τ_1 . The periods of each note, in units of τ_1 , are given in Figure 3. Let's consider A4 to check the timing. This has a frequency of 440Hz and a listed period of $0x58C\tau_1$. Converting the period to seconds yields $1420(0.8\mu s) = 1.136ms$. This corresponds to a frequency of 880Hz. Note that this is double the real frequency of the note. Also note that to generate a square wave with frequency F, the pin needs to be toggled at a frequency of 2F - the pin needs to be pulled to 1 and then to 0 for each period. Thus, Figure 3 lists the toggling period of the pin, not the signaling frequency.

2.4 Using a Speaker

The output pins of the PIC32 can only output 25mA (from PIC32 datasheet). However, the speaker draws much more. A 2W speaker operating at 5V would need to draw $\frac{2W}{5V} = 400mA$. This would surely burn out an I/O pin on the PIC32. So, I used an audio amplifier to drive the speaker. The lab recommends using the LM386. From the LM386 datasheet, I found a recommended circuit (see Figure 4).

Note	Frequency (Hz)	Period (in units of τ_1)		
A3	220	0xB18		
A sharp / B flat	233.1	0xA79		
B3	246.9	0x9E2		
C3 (middle C)	261.6	0x954		
C sharp / D flat	277.2	0x8CE		
D3	293.7	0x850		
D sharp / E flat	311.1	0x7D8		
E3	329.6	0x768		
F3	349.2	0x6FD		
F sharp / G flat	370.0	0x699		
G3	392.0	0x63A		
G sharp / A flat	415.3	0x5E0		
A4	440	0x58C		
A sharp / B flat	466.2	0x53C		
B4	493.9	0x4F1		
C4	523.3	0x4AA		
C sharp / D flat	554.4	0x467		
D4	587.3	0x428		
D sharp / E flat	622.2	0x3EC		
E4	659.2	0x3B4		
F4	698.4	0x37E		
F sharp / G flat	740.9	0x34C		
G4	784.0	0x31D		
G sharp / A flat	830.6	0x2F0		
A5	880	0x2C6		

The duration depends on an arbitrary choice of tempo (speed at which the piece is played). If a whole note is chosen to be ½ second long, other notes follow accordingly:

Duration	Seconds	Units of τ ₀	
Sixteenth	0.03125	0x0262	
Eighth	0.0625	0x04C4	
Quarter	0.125	0x0989	
Half	0.25	0x1312	
Whole	0.5	0x2625	

Figure 3: Mapping from note frequency and duration to periods of timers 1 and 2.

However, the lab was out of LM386 ICs so I used an LM380 amplifier. The LM380 is similar in function to the LM386 but has different pin layouts. Using the pinout for the LM380 from Figure 5, I replicated the circuit presented in Figure 4.

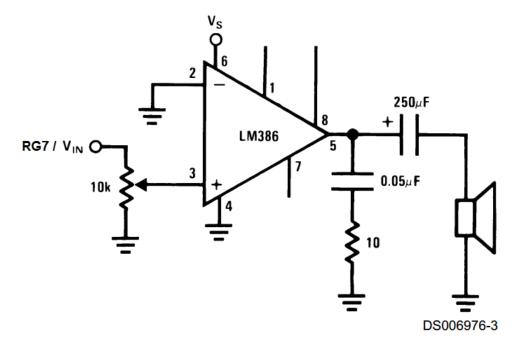


Figure 4: Basic amplifier configuration.

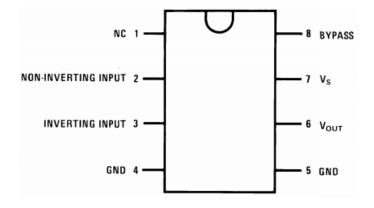


Figure 5: Basic amplifier configuration.

3 Technical Documentation

3.1 MIPS Code - Playing a Song from Flash Memory

```
/* This code plays music! The music is loaded into flash.
3 Author: Sherman Lam
   Email: slam@g.hmc.edu
4
   Date: 10-9-2014
 6
   */
7
8
   # REGISTER USE
9
10
  # t0 = register address
11 # t1 = register value
12 \# t2 = masking operations
13 \# t3 = note period
14 \# t4 = duration
15 \# t5 = address of note
16 # t6 = algebra intermediate values
17 # s0 = PORTG address
18 \# s1 = PORTG output
19
20
21
   # load a header file for pin / register definitions
   #include <P32xxxx.h>
24
25 # Define main function
26
   . global main
27
28 # Compiler instructions
   .text # store the code in the main program section of RAM
   .set noreorder # do not let the compiler reorganize your code
31
32
   # Main program
33
   .ent main # Start function block
34
   main:
35
36
       # setup timer 1 (for 1 unit of signal duration = 51.2 us)
37
       # T1CON:
38
       #
            bit 15 = 1
                            \rightarrow on
39
       #
                14 = 1
                            -> freeze on debug exception
40
                5-4 = 11
                            -> 1:256 prescaler
41
                1 = 0
                            -> use peripheral clock
42
                t0, T1CON
                                # load address of T1CON
        la
                t1, 0(t0)
                                # get the value of T1CON
43
44
                t1, t1, 0xFFFD # use a mask to set the 0 bit
        andi
                t1, t1, 0xC030 # use a mask to set the 1 bits
45
        ori
46
                t1, 0(t0)
                                # load new value to T1CON
47
48
       # setup timer 2 (for half signal period = 0.8us)
49
       # T2CON:
50
       #
            bit 15 = 1
                            -> on
51
       #
                14 = 1
                            -> freeze on debug exception
52
       #
                6-4 = 010
                            -> 1:256 prescaler
                            -> use peripheral clock
53
       #
                1
                    = 0
54
                t0, T2CON
                                # load address of T2CON
```

```
# get the value of T2CON
55
         lw
                 t1, 0(t0)
                 t1, t1, 0xFFAD
56
                                   # use a mask to set the 0 bits
         andi
                  t1, t1, 0xC020
57
         ori
                                   # use a mask to set the 1 bits
58
         sw
                  t1, 0(t0)
                                   # load new value to T2CON
59
 60
         # init some variables
61
                 t5, \_notes
                                   \# starting address of _notes
         la.
62
63
         # init output
                 t0, TRISG
64
         la
                                   # load address of TRISG
                                   # get value of TRISB
 65
                 t1, 0(t0)
 66
                 t1, t1, 0xFF7F
                                   # set RG7 as output
         andi
 67
                 t1, 0(t0)
                                   # store to TRISG
         sw
 68
                 s0, PORTG
                                   # load address of PORTG
         la.
                 zero, 0(s0)
                                  \# start at 0
69
         sw
 70
71
         # start main loop for playing song
72
73
74
             # read the note and duration. If there is no note, end
75
             lw
                      t6, 0(t5)
                                              # load notes and duration into algebra register
76
                      t6, zero, end
                                             # branch if there are no more notes
             beq
 77
             nop
                      t2, 0xFFFF0000
 78
             lί
                                              # mask for duration
 79
                      t4\ , \quad t6\ , \quad t2
                                              # mask off to get duration
             and
 80
             srl
                      t4\;,\;\;t4\;,\;\;16
                                             # shift to lower 16 bits to get duration
                      t3, t6, 0xFFFF
                                              # mask off the note
81
             andi
                      t6, 0x2
82
             l i
                                              # load 2 into t6
83
             mul
                      t4, t4, t6
                                              # multiply duration by 2
 84
 85
             # reset timer 1
                                         # get address of timer 1
 86
             la
                      t0, TMR1
 87
                      zero, 0(t0)
                                         # reset timer to 0
             sw
 88
             # reset timer 2
 89
                      t0, TMR2
 90
             la
                                       # get address of timer 2
 91
                      zero, 0(t0)
                                         # reset timer to 0
 92
             # poll the timer until the duration has been met
93
94
             dur:
                          t0, TMR1
                                           # get address of timer 1
95
                  la.
96
                 lw
                          t1, 0(t0)
                                           # poll timer 1
97
                          t1, t4, next
                                           # check if timer 1 has reached duration
98
                 nop
99
100
                 # while the duration is being held, keep playing the notes
101
                  note:
                              t0, TMR2
102
                      lа
                                                # get address of timer 2
103
                      lw
                              t1, 0(t0)
                                                # poll timer 2
                              t6, t1, t3
                                                # check if time is less than the period
104
                      slt
105
                      bne
                              t6, zero, dur
                                                # if not time to toggle speaker, check duration
106
                      nop
107
                      # toggle!
108
                              t6, 0xFFFF
                                                \# load -1 into t6
                      l i
109
                                                \# xor with -1. This inverts all the bits
                              s1, s1, t6
                      xor
                                                # use mask to only get RG7
110
                      andi
                              s1, s1, 0x0080
111
                              s0, PORTG
                                                # load address of PORTG
                      la
112
                              s1, 0(s0)
                                                # write the value to PORTG.
                      sw
113
                      lа
                              t0, TMR2
                                                # get address of timer 2
```

```
zero, 0(t0)
114
                                                 # reset timer 2
                      sw
115
                                                 # check duration
                      j
                               dur
116
                      nop
117
118
             # figure out next offset
119
             next:
120
                           t0, TMR1
                                                 # get address of timer 1
                  la
121
                           zero, 0(t0)
                                                 # reset timer 1
                  sw
                           t5, t5, 4
122
                  addi
                                                 # get next note's address
123
                  j
                           play
                                                 # play the next note
124
                  nop
125
126
         end:
127
             nop
128
                                            # jump to return address.
             jr
                      ra
129
130
131
    # end the function
    end main
```

3.1.1 Loading Fur Elise to Flash

This piece of code was provided in the lab and plays Fur Elise. The values are loaded into flash memory as a word.

```
1
   # Song notes
 2
        .section .rodata # Store this information in FLASH instead of RAM
 3
 4
        .HWORD 0x3b4, 0x989 \# Data
        .HWORD 0x3ec, 0x989
5
 6
        .HWORD 0x3b4, 0x989
        .HWORD 0x3ec, 0x989
7
        .HWORD 0x3b4, 0x989
 8
        .HWORD 0x4f1, 0x989
9
        .HWORD 0x428, 0x989
10
        .HWORD 0x4aa, 0x989
11
12
        .HWORD 0x58c, 0x1312
13
        .HWORD 0 \times 0000, 0 \times 989
14
        .HWORD 0x954, 0x989
        .HWORD 0x768, 0x989
15
16
        .HWORD 0x58c, 0x989
17
        .HWORD 0x4f1, 0x1312
        .HWORD 0x000, 0x989
18
19
        .HWORD 0x768, 0x989
        .HWORD 0x5e0, 0x989
20
        .HWORD 0x4f1, 0x989
21
22
        .HWORD 0x4aa, 0x1312
        .HWORD 0x000, 0x989
23
24
        .HWORD 0x768, 0x989
25
        .HWORD 0x3b4, 0x989
26
        .HWORD 0x3ec, 0x989
27
        .HWORD 0x3b4, 0x989
28
        .HWORD 0x3ec, 0x989
29
        .HWORD 0x3b4, 0x989
30
        .HWORD 0x4f1, 0x989
31
        .HWORD 0x428, 0x989
32
        .HWORD 0x4aa, 0x989
        .HWORD 0x58c, 0x1312
33
        .HWORD 0 \times 0000, 0 \times 989
34
        .HWORD 0x954, 0x989
35
```

```
36
        .HWORD 0x768, 0x989
37
        .HWORD 0x58c, 0x989
        .HWORD 0x4f1, 0x1312
38
39
        .HWORD 0x000, 0x989
40
        .HWORD 0x768, 0x989
41
        .HWORD 0x4aa, 0x989
42
        .HWORD 0x4f1, 0x989
43
        .HWORD 0x58c, 0x1312
        .HWORD 0x000, 0x989
44
        .HWORD 0x4f1, 0x989
45
        .HWORD 0x4aa, 0x989
46
47
        .HWORD 0x428, 0x989
48
        .HWORD 0x3b4, 0x1c9c
49
        .HWORD 0x63a, 0x989
50
        .HWORD 0x37e, 0x989
        .HWORD 0x3b4, 0x989
51
52
        .HWORD 0x428, 0x1c9c
        .HWORD 0x6fd, 0x989
53
54
        .HWORD 0x3b4, 0x989
55
        .HWORD 0x428, 0x989
56
        .HWORD 0x4aa, 0x1c9c
        .HWORD 0x768, 0x989
57
        .HWORD 0x428, 0x989
58
59
        .HWORD 0x4aa, 0x989
60
        .HWORD 0x4f1, 0x1312
61
        .HWORD 0x000, 0x989
62
        .HWORD 0x768, 0x989
63
        .HWORD 0x3b4, 0x989
64
        .HWORD 0x000, 0x989
65
        .HWORD 0x000, 0x989
66
        .HWORD 0x3b4, 0x989
67
        .HWORD 0x1da, 0x989
68
        .HWORD 0x000, 0x989
69
        .HWORD 0x000, 0x989
70
        .HWORD 0x3ec, 0x989
        .HWORD 0x3b4, 0x989
71
        .HWORD 0 \times 000, 0 \times 989
72
        .HWORD 0x000, 0x989
73
74
        .HWORD 0x3ec, 0x989
        .HWORD 0x3b4, 0x989
75
76
        .HWORD 0x3ec, 0x989
77
        .HWORD 0x3b4, 0x989
        .HWORD 0x3ec, 0x989
78
79
        .HWORD 0x3b4, 0x989
80
        .HWORD 0x4f1, 0x989
81
        .HWORD 0x428, 0x989
82
        .HWORD 0x4aa, 0x989
83
        .HWORD 0x58c, 0x1312
        .HWORD 0x000, 0x989
84
        .HWORD 0x954, 0x989
85
        .HWORD 0x768, 0x989
86
87
        .HWORD 0x58c, 0x989
88
        .HWORD 0x4f1, 0x1312
89
        .HWORD 0x000, 0x989
90
        .HWORD 0x768, 0x989
91
        .HWORD 0x5e0, 0x989
92
        .HWORD 0x4f1, 0x989
93
        .HWORD 0x4aa, 0x1312
94
        .HWORD 0x000, 0x989
```

```
95
          .HWORD 0x768, 0x989
96
          .HWORD 0x3b4, 0x989
          .HWORD 0x3ec, 0x989
97
 98
          .HWORD 0x3b4, 0x989
99
          .HWORD 0x3ec, 0x989
          .HWORD 0x3b4, 0x989
100
          .HWORD 0x4f1, 0x989
101
          .HWORD 0x428, 0x989
102
103
          .HWORD 0x4aa, 0x989
          .HWORD 0x58c, 0x1312
104
          .HWORD 0x000, 0x989
105
          .HWORD 0x954, 0x989
106
107
          .HWORD 0x768, 0x989
          .HWORD 0x58c, 0x989
108
          .HWORD 0 \times 4 f1, 0 \times 1312
109
          .HWORD 0x000, 0x989
110
          .HWORD 0x768, 0x989
111
112
          .HWORD 0x4aa, 0x989
          .HWORD 0x4f1, 0x989
113
          .HWORD 0x58c, 0x2625
114
115
          .HWORD 0 \times 000, 0 \times 000
```

3.1.2 Loading Reluctant Heroes (from Attack on Titan anime) to Flash Memory

This piece of code loads notes and durations for the first 9 measures of "Reluctant Heroes." Below, each block of notes belongs to its own measure.

```
1
   # Song notes
        .section .rodata \# Store this information in FLASH instead of RAM
 2
3
    _notes:
                                  # D4
        .HWORD 0x428, 0x989
4
        .HWORD 0x467, 0x989
                                  # C4#
 5
        .HWORD 0x428, 0x989
                                  # D4
 6
 7
        .HWORD 0x3b4, 0x989
                                  # E4
 8
                                  # F4#
9
        .HWORD 0x34c, 0x1312
10
        .HWORD 0x4f1, 0x989
                                  # B4
        .HWORD 0x428, 0x1312
                                  # D4
11
12
        .HWORD 0x4f1, 0x989
                                  # B4
13
        .HWORD 0x428, 0x989
                                  # D4
                                  # E4
        .HWORD 0x3b4, 0x989
14
15
16
        .HWORD 0x34c, 0x1312
                                  # F4#
17
        .HWORD 0x4f1, 0x989
                                  # B4
        .HWORD 0x428, 0x1312
18
                                  # D4
        .HWORD 0x428, 0x989
19
                                  # D4
20
        .HWORD 0x428, 0x989
                                  # D4
                                  # A5
21
        .HWORD 0x2c6, 0x1312
22
                                  # G4
23
        .HWORD 0x31d, 0x1312
        .HWORD 0x31d, 0x1312
                                  # G4
24
                                  # F4#
25
        .HWORD 0x34c, 0x1312
        .HWORD 0x34c, 0x1312
26
                                  # F4#
27
        .HWORD 0x3b4, 0x1312
28
                                  # E4
29
        .HWORD 0x428, 0x1312
                                  # D4
        .HWORD 0x4f1, 0x989
                                  # B4
30
```

```
31
        .HWORD 0x428, 0x989
                                  # D4
32
                                  # E4
        .HWORD 0x3b4, 0x989
33
                                  # F4#
34
        .HWORD 0x34c, 0x1312
35
        .HWORD 0x4f1, 0x989
                                  # B4
        .HWORD 0x428, 0x1312
                                  # D4
36
37
        .HWORD 0x4f1, 0x989
                                  # B4
                                  # D4
        .HWORD 0x428, 0x989
38
        .HWORD 0x3b4, 0x989
39
                                  # E4
40
        .HWORD 0x34c, 0x1312
                                  # F4#
41
        .HWORD 0x4f1 , 0x989 .HWORD 0x428 , 0x1312
                                  # B4
42
43
                                  # D4
44
        .HWORD 0x428, 0x989
                                  # D4
        .HWORD 0x467, 0x989
                                  # C4#
45
        .HWORD 0x428, 0x1312
                                  # D4
46
47
        .HWORD 0x3b4, 0x1312
                                  # E4
48
                                  # D4
49
        .HWORD 0x428, 0x1312
50
        .HWORD 0x428, 0x989
                                  # D4
        .HWORD 0x467, 0x989
                                  # C4#
51
        .HWORD 0x428, 0x1312
52
                                  # D4
53
        .HWORD 0x34c, 0x1312
                                  # F4#
54
        .HWORD 0x3b4, 0x1312
55
                                  # E4
56
57
        .HWORD 0x0, 0x0
                                  # stop
```

3.2 Timer 1 and Timer 2 Registers

The following tables were obtained from the PIC32 datasheet. They indicate the function of each part of the control registers for Timer 1 and Timer 2. Refer to the datasheet for more indepth descriptions of each part. The important ones used in this lab are ON and TCKPS. ON turns the timer on and TCKPS sets the prescalar of each clock.

Register 14-1:	T1CON: Typ	e A Timer Co	ntrol Register				
r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
_	_	_	_	_	_	_	_
bit 31							bit 24
r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
_	-	_	-	_	_	_	_
bit 23							bit 16
R/W-0	R/W-0	R/W-0	R/W-0	R-0	r-0	r-0	r-0
ON ⁽¹⁾	FRZ ⁽²⁾	SIDL	TWDIS	TWIP	_	_	_
bit 15							bit 8
R/W-0	r-0	R/W-0	R/W-0	r-0	R/W-0	R/W-0	r-0
TGATE	_	TCKPS	S<1:0>	_	TSYNC	TCS	_
bit 7							bit 0
Legend:							
R = Readable bit		W = Writable bit		P = Programmable bit		r = Reserved	bit
U = Unimpleme	nted bit	-n = Bit Value at POR: ('0', '1', x = Unknown)					

Figure 6: Control Register for T1CON.

Register 14-2:	TxCON: Typ	e B Timer Co	ntrol Register	r			
r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
_	_	_	_	_	_	_	_
bit 31							bit 24
r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
_	-	_	-	_	_	_	-
bit 23							bit 16
R/W-0	R/W-0	R/W-0	r-0	r-0	r-0	r-0	r-0
ON ⁽¹⁾	FRZ ⁽²⁾	SIDL ⁽⁴⁾	_	_	_	_	_
bit 15							bit 8
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	r-0	R/W-0	r-0
TGATE		TCKPS<2:0>		T32 ⁽³⁾	_	TCS	_
bit 7							bit 0
Legend:							
R = Readable bit		W = Writable bit		P = Programmable bit		r = Reserved	bit
U = Unimplemented bit -n = Bit Value at POR: ('0', '1', x = Unknown)							

Figure 7: Control Register for T2CON.

4 Results and Discussion

Music playing works! Since the tempo was arbitrary chosen when 3 was created, Fur Elise was too fast. So, I doubled the periods to slow down the song to an appropriate pace.

I also converted the first several measures of "Reluctant Hero" into units of τ_0 and τ_1 . This also plays as expected.

In addition, I was unable to set the peripheral clock through modifying the control register OSCCON directly through assembly code. Rather, they must be set through the configuration bits. This is because the configuration bits take priority and prevent the assembly code from changing OSCCON.

5 Conclusion

5.1 Time Spent

Programming, Simulating 6 hrs

Writing Report 3hrs

Total Time Spent 9hrs

5.2 Suggestions for lab

No suggestions for the lab.