Sherman Lam
E155
October 13, 2014

Lab 5 Report: Digital Audio

1 Introduction

In this lab, I used the PIC32 on the μMudd32 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.

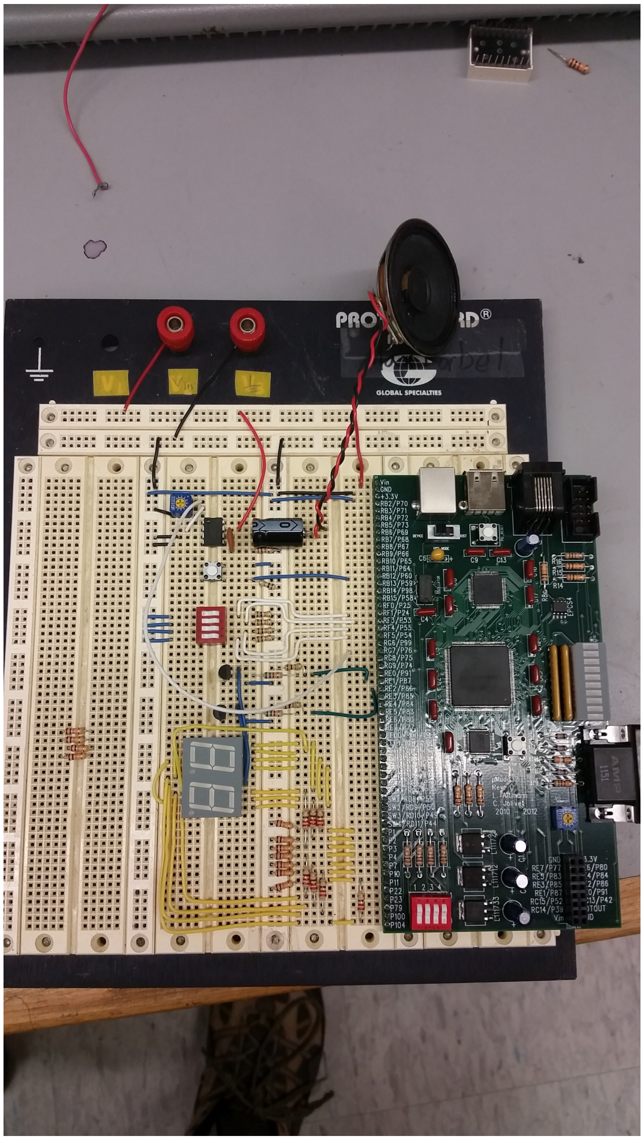


Figure 1: Complete development board with audio buffer / amplifier at the top of the breadboard.

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.

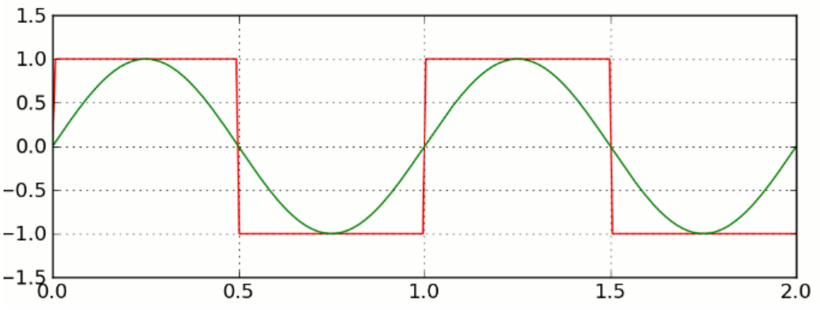


Figure 2: Square wave approximation of a sine wave. Source: http://upload.wikimedia.org/wikipedia/commons/f/f8/SquareWave.gif

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.

```
main(){
    // setup timers
    T1CON = 11xx_xxxx_xx11_xxx0;    // x = don't care
    T2CON = 11xx_xxxx_x010_xx0x;

    // setup outputs
    TRISG = 1111_1111_0111_1111;    // make port 7 an output

    // start playing the song
    int i = 0;
    while(true){
        int word = _notes[i];

        // check for end of song
        if (word == 0){
            break;
        }

        int halfPeriod = word[15:8];
        int duration = word[7:0];

        // play for duration
        TMR1 = 0
        TMR2 = 0
        while(TMR1 < duration){
            //toggle according to the period
            if (TMR2 >= halfPeriod){
                //toggle output
                PORTG[7] = ~PORTG[7];
                TMR2 = 0;
            }
        }
        //move to next note
        i++;
    }
}
```

2.3 Timing

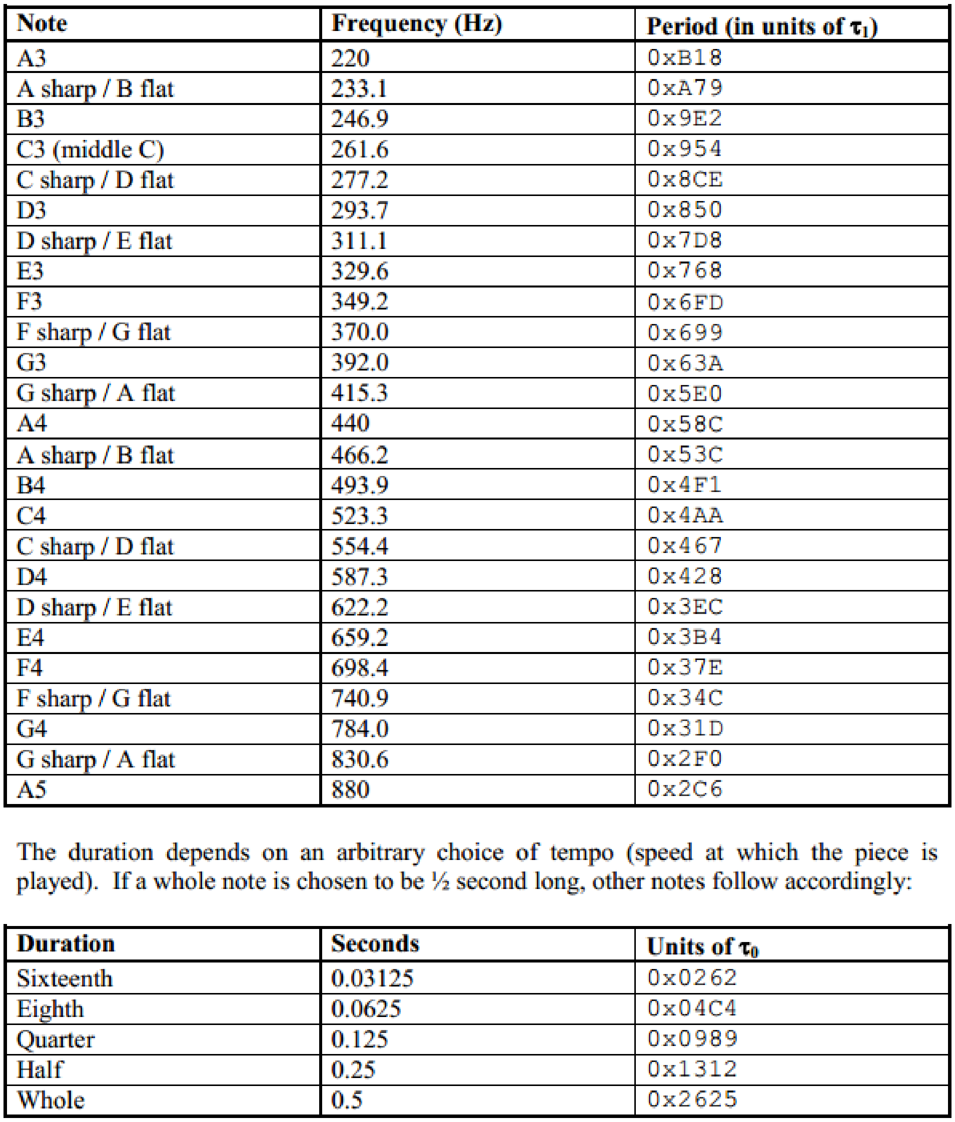
The clock on the μMudd32 operates at 40Hz. The prescalar for the PIC's peripheral clock was set to 1:8. This means that the peripheral clock will run at $\frac{40MHz}{8} = 5MHz$.

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 0x58Cτ_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.

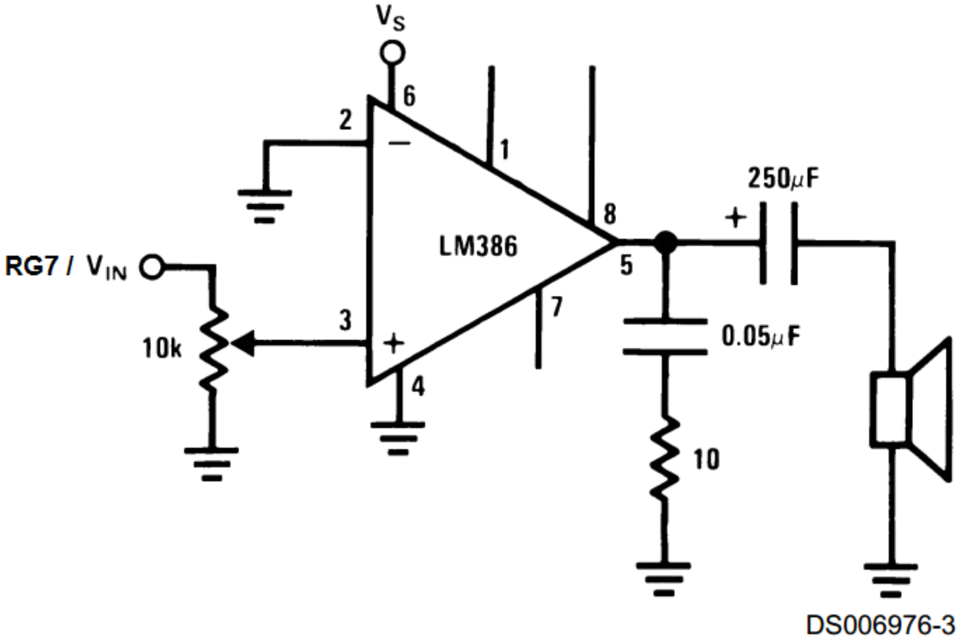


Figure 4: Basic amplifier configuration.

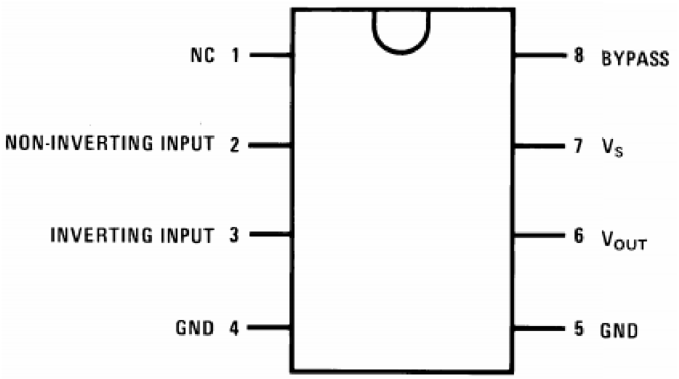


Figure 5: Basic amplifier configuration.

3 Technical Documentation

3.1 MIPS Code - Playing a Song from Flash Memory

```
/* This code plays music! The music is loaded into flash.

Author: Sherman Lam
Email: slam@g.hmc.edu
Date: 10-9-2014
*/


# REGISTER USE
# t0 = register address
# t1 = register value
# t2 = masking operations
# t3 = note period
# t4 = duration
# t5 = address of note
# t6 = algebra intermediate values
# s0 = PORTG address
# s1 = PORTG output



# load a header file for pin / register definitions
#include <P32xxxx.h>

# Define main function
.global main

# Compiler instructions
.text # store the code in the main program section of RAM
.set noreorder # do not let the compiler reorganize your code

# Main program
.ent main # Start function block
main:

    # setup timer 1 (for 1 unit of signal duration = 51.2us)
    # T1CON:
    #   bit 15  = 1     -> on
    #       14  = 1     -> freeze on debug exception
    #       5-4 = 11    -> 1:256 prescaler
    #       1   = 0     -> use peripheral clock
    la      t0, T1CON       # load address of T1CON
    lw      t1, 0(t0)       # get the value of T1CON
    andi    t1, t1, 0xFFFD  # use a mask to set the 0 bit
    ori     t1, t1, 0xC030  # use a mask to set the 1 bits
    sw      t1, 0(t0)       # load new value to T1CON

    # setup timer 2 (for half signal period = 0.8us)
    # T2CON:
    #   bit 15  = 1     -> on
    #       14  = 1     -> freeze on debug exception
    #       6-4 = 010   -> 1:256 prescaler
    #       1   = 0     -> use peripheral clock
    la      t0, T2CON       # load address of T2CON
    lw      t1, 0(t0)       # get the value of T2CON
    andi    t1, t1, 0xFFAD  # use a mask to set the 0 bits
    ori     t1, t1, 0xC020  # use a mask to set the 1 bits
    sw      t1, 0(t0)       # load new value to T2CON

    # init some variables
    la      t5, _notes      # starting address of _notes

    # init output
    la      t0, TRISG       # load address of TRISG
    lw      t1, 0(t0)       # get value of TRISB
    andi    t1, t1, 0xFF7F  # set RG7 as output
    sw      t1, 0(t0)       # store to TRISG
    la      s0, PORTG       # load address of PORTG
    sw      zero, 0(s0)     # start at 0

    # start main loop for playing song
    play:

        # read the note and duration. If there is no note, end
        lw      t6, 0(t5)               # load notes and duration into algebra register
        beq     t6, zero, end           # branch if there are no more notes
        nop
        li      t2, 0xFFFF0000          # mask for duration
        and     t4, t6, t2              # mask off to get duration
        srl     t4, t4, 16              # shift to lower 16bits to get duration
        andi    t3, t6, 0xFFFF          # mask off the note
        li      t6, 0x2                 # load 2 into t6
        mul     t4, t4, t6              # multiply duration by 2

        # reset timer 1
        la      t0, TMR1            # get address of timer 1
        sw      zero, 0(t0)         # reset timer to 0

        # reset timer 2
        la      t0, TMR2          # get address of timer 2
        sw      zero, 0(t0)         # reset timer to 0

        # poll the timer until the duration has been met
        dur:
            la      t0, TMR1        # get address of timer 1
            lw      t1, 0(t0)       # poll timer 1
            beq     t1, t4, next    # check if timer 1 has reached duration
            nop

            # while the duration is being held, keep playing the notes
            note:
                la      t0, TMR2        # get address of timer 2
                lw      t1, 0(t0)       # poll timer 2
                slt     t6, t1, t3      # check if time is less than the period
                bne     t6, zero, dur   # if not time to toggle speaker, check duration
                nop
                # toggle!
                li      t6, 0xFFFF      # load −1 into t6
                xor     s1, s1, t6      # xor with −1. This inverts all the bits
                andi    s1, s1, 0x0080  # use mask to only get RG7
                la      s0, PORTG       # load address of PORTG
                sw      s1, 0(s0)       # write the value to PORTG.
                la      t0, TMR2        # get address of timer 2
                sw      zero, 0(t0)     # reset timer 2
                j       dur             # check duration
                nop

        # figure out next offset
        next:
            la      t0, TMR1            # get address of timer 1
            sw      zero, 0(t0)         # reset timer 1
            addi    t5, t5, 4           # get next note's address
            j       play                # play the next note
            nop

    end:
        nop
        jr      ra                  # jump to return address.


# 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.

```
# Song notes
    .section .rodata  # Store this information in FLASH instead of RAM
_notes:
    .HWORD 0x3b4, 0x989 # Data
    .HWORD 0x3ec, 0x989
    .HWORD 0x3b4, 0x989
    .HWORD 0x3ec, 0x989
    .HWORD 0x3b4, 0x989
    .HWORD 0x4f1, 0x989
    .HWORD 0x428, 0x989
    .HWORD 0x4aa, 0x989
    .HWORD 0x58c, 0x1312
    .HWORD 0x000, 0x989
    .HWORD 0x954, 0x989
    .HWORD 0x768, 0x989
    .HWORD 0x58c, 0x989
    .HWORD 0x4f1, 0x1312
    .HWORD 0x000, 0x989
    .HWORD 0x768, 0x989
    .HWORD 0x5e0, 0x989
    .HWORD 0x4f1, 0x989
    .HWORD 0x4aa, 0x1312
    .HWORD 0x000, 0x989
    .HWORD 0x768, 0x989
    .HWORD 0x3b4, 0x989
    .HWORD 0x3ec, 0x989
    .HWORD 0x3b4, 0x989
    .HWORD 0x3ec, 0x989
    .HWORD 0x3b4, 0x989
    .HWORD 0x4f1, 0x989
    .HWORD 0x428, 0x989
    .HWORD 0x4aa, 0x989
    .HWORD 0x58c, 0x1312
    .HWORD 0x000, 0x989
    .HWORD 0x954, 0x989
```

```
        .HWORD 0x768, 0x989
        .HWORD 0x58c, 0x989
        .HWORD 0x4f1, 0x1312
        .HWORD 0x000, 0x989
        .HWORD 0x768, 0x989
        .HWORD 0x4aa, 0x989
        .HWORD 0x4f1, 0x989
        .HWORD 0x58c, 0x1312
        .HWORD 0x000, 0x989
        .HWORD 0x4f1, 0x989
        .HWORD 0x4aa, 0x989
        .HWORD 0x428, 0x989
        .HWORD 0x3b4, 0x1c9c
        .HWORD 0x63a, 0x989
        .HWORD 0x37e, 0x989
        .HWORD 0x3b4, 0x989
        .HWORD 0x428, 0x1c9c
        .HWORD 0x6fd, 0x989
        .HWORD 0x3b4, 0x989
        .HWORD 0x428, 0x989
        .HWORD 0x4aa, 0x1c9c
        .HWORD 0x768, 0x989
        .HWORD 0x428, 0x989
        .HWORD 0x4aa, 0x989
        .HWORD 0x4f1, 0x1312
        .HWORD 0x000, 0x989
        .HWORD 0x768, 0x989
        .HWORD 0x3b4, 0x989
        .HWORD 0x000, 0x989
        .HWORD 0x000, 0x989
        .HWORD 0x3b4, 0x989
        .HWORD 0x1da, 0x989
        .HWORD 0x000, 0x989
        .HWORD 0x000, 0x989
        .HWORD 0x3ec, 0x989
        .HWORD 0x3b4, 0x989
        .HWORD 0x000, 0x989
        .HWORD 0x000, 0x989
        .HWORD 0x3ec, 0x989
        .HWORD 0x3b4, 0x989
        .HWORD 0x3ec, 0x989
        .HWORD 0x3b4, 0x989
        .HWORD 0x3ec, 0x989
        .HWORD 0x3b4, 0x989
        .HWORD 0x4f1, 0x989
        .HWORD 0x428, 0x989
        .HWORD 0x4aa, 0x989
        .HWORD 0x58c, 0x1312
        .HWORD 0x000, 0x989
        .HWORD 0x954, 0x989
        .HWORD 0x768, 0x989
        .HWORD 0x58c, 0x989
        .HWORD 0x4f1, 0x1312
        .HWORD 0x000, 0x989
        .HWORD 0x768, 0x989
        .HWORD 0x5e0, 0x989
        .HWORD 0x4f1, 0x989
        .HWORD 0x4aa, 0x1312
        .HWORD 0x000, 0x989
```

```
    .HWORD 0x768, 0x989
    .HWORD 0x3b4, 0x989
    .HWORD 0x3ec, 0x989
    .HWORD 0x3b4, 0x989
    .HWORD 0x3ec, 0x989
    .HWORD 0x3b4, 0x989
    .HWORD 0x4f1, 0x989
    .HWORD 0x428, 0x989
    .HWORD 0x4aa, 0x989
    .HWORD 0x58c, 0x1312
    .HWORD 0x000, 0x989
    .HWORD 0x954, 0x989
    .HWORD 0x768, 0x989
    .HWORD 0x58c, 0x989
    .HWORD 0x4f1, 0x1312
    .HWORD 0x000, 0x989
    .HWORD 0x768, 0x989
    .HWORD 0x4aa, 0x989
    .HWORD 0x4f1, 0x989
    .HWORD 0x58c, 0x2625
    .HWORD 0x000, 0x000
```

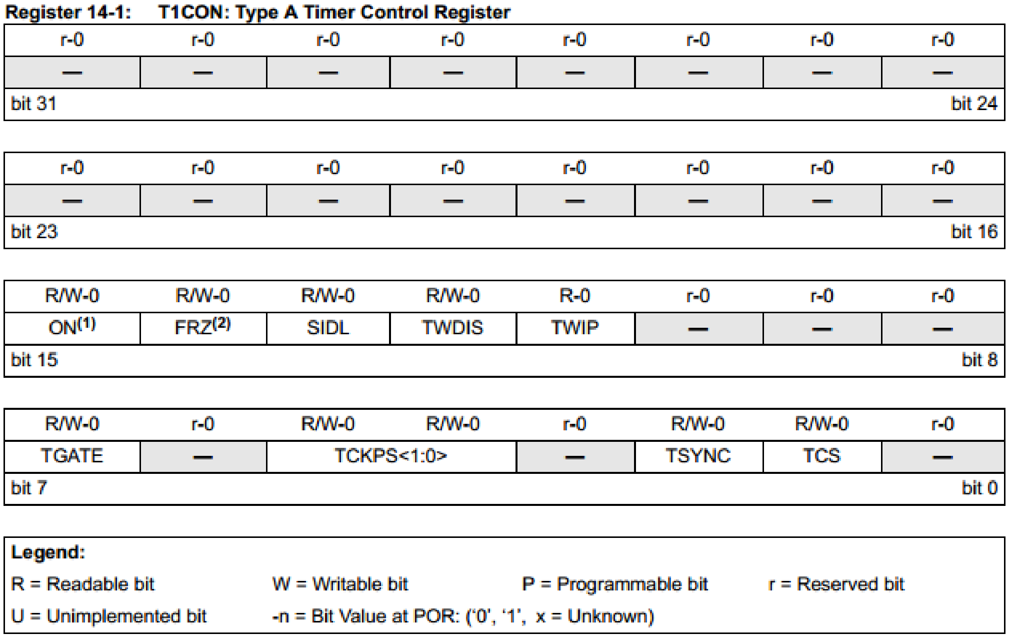
3.1.2 Loading Reluctant Heroes (from Attack on Titan anime) to Flash Memory

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.

```
# Song notes
    .section .rodata  # Store this information in FLASH instead of RAM
_notes:
    .HWORD 0x428, 0x989     # D4
    .HWORD 0x467, 0x989     # C4#
    .HWORD 0x428, 0x989     # D4
    .HWORD 0x3b4, 0x989     # E4

    .HWORD 0x34c, 0x1312    # F4#
    .HWORD 0x4f1, 0x989     # B4
    .HWORD 0x428, 0x1312    # D4
    .HWORD 0x4f1, 0x989     # B4
    .HWORD 0x428, 0x989     # D4
    .HWORD 0x3b4, 0x989     # E4

    .HWORD 0x34c, 0x1312    # F4#
    .HWORD 0x4f1, 0x989     # B4
    .HWORD 0x428, 0x1312    # D4
    .HWORD 0x428, 0x989     # D4
    .HWORD 0x428, 0x989     # D4
    .HWORD 0x2c6, 0x1312    # A5

    .HWORD 0x31d, 0x1312    # G4
    .HWORD 0x31d, 0x1312    # G4
    .HWORD 0x34c, 0x1312    # F4#
    .HWORD 0x34c, 0x1312    # F4#

    .HWORD 0x3b4, 0x1312    # E4
    .HWORD 0x428, 0x1312    # D4
    .HWORD 0x4f1, 0x989     # B4
    .HWORD 0x428, 0x989     # D4
    .HWORD 0x3b4, 0x989     # E4

    .HWORD 0x34c, 0x1312    # F4#
    .HWORD 0x4f1, 0x989     # B4
    .HWORD 0x428, 0x1312    # D4
    .HWORD 0x4f1, 0x989     # B4
    .HWORD 0x428, 0x989     # D4
    .HWORD 0x3b4, 0x989     # E4

    .HWORD 0x34c, 0x1312    # F4#
    .HWORD 0x4f1, 0x989     # B4
    .HWORD 0x428, 0x1312    # D4
    .HWORD 0x428, 0x989     # D4
    .HWORD 0x467, 0x989     # C4#
    .HWORD 0x428, 0x1312    # D4

    .HWORD 0x3b4, 0x1312    # E4
    .HWORD 0x428, 0x1312    # D4
    .HWORD 0x428, 0x989     # D4
    .HWORD 0x467, 0x989     # C4#
    .HWORD 0x428, 0x1312    # D4

    .HWORD 0x34c, 0x1312    # F4#
    .HWORD 0x3b4, 0x1312    # E4

    .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: Type A Timer Control Register

r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
—	—	—	—	—	—	—	—
bit 31							bit 24

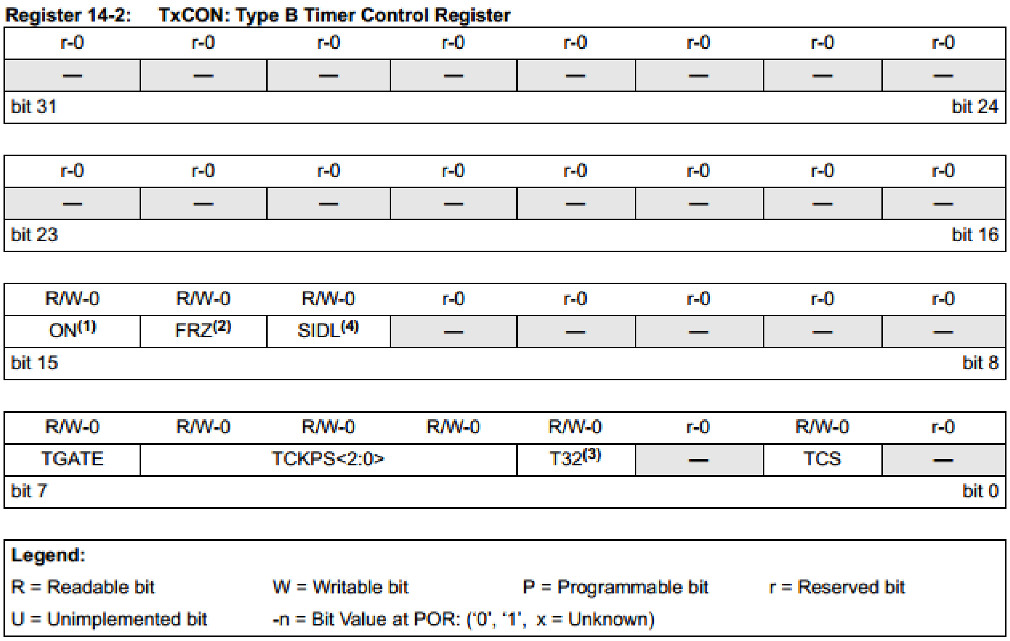
r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
—	—	—	—	—	—	—	—
bit 23							bit 16

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<1:0>		—	TSYNC	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 6: Control Register for T1CON.



Register 14-2: TxCON: Type B Timer Control 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-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.