CMPE 12L Lab 6 - Winter 2014
Timers and Interrupts

100 Points (40 Report, 60 Work)
Due Date: March 12, 2014 12:30 PM

IMPORTANT: Try to minimize the noise you create when testing your program. You can disconnect the speaker to immediately stop any squealing and annoying other students, TAs, and tutors!

1 Background

In lab 5, you essentially made a delay loop that was a polling loop. Your program iterated until a time was up to create a delay. In this lab, you will use interrupts from a timer to implement similar functionality. This is then used to drive a piezoelectric speaker at certain frequencies to play a tune.

Note: Since this is the last lab we will not accept more than 1 day late. eCommons will not accept submissions after March 13 at 12:30pm.

2 Prerequisites

- Read through this **entire** lab assignment.
- Read Sec14_Timers.pdf in eCommons (specifically pay attention to Type A/Timer 1)
- Read Sec08_Interrupts.pdf in eCommons

3 Tutor/TA Review

Your lab tutor/TA will cover the following items in the first portion of the first lab:

- Audio signal theory: frequency, period, duty cycle
- How to hook up your piezoelectric speaker
- What's required

4 Piezoelectric Speakers

Piezoelectric speakers are frequently used in "cheap" devices like watches. They are inexpensive and are not easily overloaded/destroyed. They have the disadvantage that they have poor audio quality and easily distort, however.

Piezoelectric speakers work by applying a voltage that causes a piezoelectric film to deform. This distortion can cause an audio sound by displacing air much like normal speakers. Note that this behavior can also work the other way by applying a strain to the material and measuring a voltage to measure the strain. This is commonly used as a strain or pressure sensor.

If you apply a square wave voltage to our piezoelectric speaker module, you will get a constant frequency output that you can hear. Not applying a voltage produces no sound output. In order to get different frequencies, we must drive the speaker with a square wave signal of the desired frequency. This is where the timer comes in.

5 Timer Interrupts

In the previous lab assignment, you created a delay function using a delay loop. This method, however, is not very accurate and occupies the processor so that it cannot do other work. In this lab assignment, we will use an on-chip "timer" module that counts clock pulses and causes an interrupt when a specified number have occurred. The frequency of the waveform sent to our speaker will be controlled by this timer and its interrupt.

As a reminder from lab 5, each of the device registers actually has four addresses that perform different functions. The base register can be read or written to change all of the bits at once. Three additional registers are used to modify one bit at a time: clear (offset 4), set (offset 8), and invert (offset 12). The ones in a mask specify which bits to clear, set or invert. So, for example, if you want to set bit 0 of the TRISF register (0xBF886140), you can write a binary mask of 0x1 to address 0xBF886148 (0xBF886140 + 8).

Table 4-7 on page 56 of PIC32_61143H.pdf provides the configuration registers for the timer module: T1CON, TMR1, PR1. These are configurable just like the digital output registers in the last lab using direct write, clear, set and invert register offsets. TMR1 is the current timer count and PR1 is the period register. After TMR1 reaches the count in PR1, it will trigger an interrupt when configured to do so. You should clear TMR1 and set PR1 according to the period of the tone that you want to generate.

T1CON is the timer configuration register. You should use the T1 clock prescaler (T1CKPS[1:0]) to divide the system clock to a slower one so that the counter operates slower. You should configure this to use the maximum prescalar of 1:256 by writing 0b11 to T1CKPS[1:0] (T1CON[5:4]). After you have set the prescalar, you can enable the timer by setting bit the ON configuration bit T1CON[15]. (This should wait until you have interrupts set up too, though.)

Table 4-4 on page 53 of PIC32_61143H.pdf provides the interrupt configuration registers: IFS0/IFS1 and IEC0/IEC1. T1IF is the timer 1 interrupt status flag and T1IE is the timer 1 interrupt enable flag which behave exactly like the LC-3 interrupt configuration bits. These are in registers IFS0 and IEC0, respectively.

The PIC32 also has programmable interrupt priorities in the IPC0-IPC11 registers. T1IP[2:0] which is the IPC1 register is the timer 1 interrupt priority configuration register. You should set

it to 0b100 (4). T1IS[1:0] is the interrupt subpriority which is used if you have more than one interrupt matched to the same priority. We don't care about that.

The detailed discussion of the functionality of the timer registers and interrupts are available in Sec14_Timers.pdf of the PIC32 Reference Manual.

Details on interrupt handlers are in Sec08_Interrupts.pdf. Specifically, section 8.6 talks about the address of the handler. The Timer 1 vector is in Table 7-1 of PIC32_61143H.pdf on page 90. Our PIC32 is using multi-vectored interrupts (Section 8.5) similar to the LC-3 vector table. The EBASE address is set to 0xBFC01000 which is the configurable location of this vector table. The vector offset is computed according to Section 8.6.1 as

```
Computed Offset Vector Address = Vector * Vector Spacing + EBase + 0x200
```

Our vector spacing is set as 32 bytes and timer 1 uses interrupt vector 4. Lucky for you, our linker uses the chipKIT-UNO32-application-32MX320F128L.ld linker script to know where to place this in memory using the following "section" directive and the jump instruction:

```
.section .vector_4,"xaw"
j T1_ISR
```

This will jump to a service routine that you implement called T1_ISR.

6 Assignment

In this lab, you will use interrupts from a timer to implement a music playing program that drives a piezoelectric speaker at specified frequencies for specified durations to play a tune. You are given the main.cpp program that contains the top-level program and iterates over an array of notes and durations to play a melody. You must implement the required subroutines in assembly to complete this program.

6.1 SetupPort()

The setup port procedure in your program should configure the output port to be a digital output just like in lab 5. This is called once from main.cpp to initialize the output port. Instead of the LED for output, you should configure pin 9 on the Uno32 board. Pin 9 is connected to pin 3 of port D of the PIC32 processor. Please refer to PIC32_61143H.pdf to find the correct memory mapped address for this I/O port using your knowledge gained in the last lab.

The piezoelectric speaker should have the red wire connected to pin 9 on header 16 of the Uno32 board. The black wire should be connected to one of the GND pins on header 7. For a description of these, please see page 2 of the Uno32 reference manual previously provided in lab 5.

6.2 SetupTimer()

The setup timer procedure in your program should configure timer 1 for the correct prescalar, interrupt priority, and make sure the count and interrupt status are cleared. This is called once from main.cpp. This procedure should **NOT** start the timer or create an output sound until we actually play a note. That means it should **NOT** enable the timer to start counting. This is because we don't yet know what frequency of note that we want to play.

6.3 PlayNote(int, int, int)

The PlayNote procedure is the main part of the program. The input of your program is also contained in main.cpp. There are three variables: size, melody, and duration. Size specifies the length of the two arrays melody and duration. Melody contains the notes to play and duration is the length of each note in milliseconds. The notes are given symbolic names using define preprocessor directives in C. NOTE_A4, for example, is replaced with the constant frequency 440.

A single "note" as shown in Figure 1 is defined as a tone followed by a duration of silence. All durations are specified in milliseconsd (ms). The silence after a tone fills up an entire full note duration which is the final argument to PlayNote. The silence allows for separation of the notes when playing a tune.

The tone of the note is a square wave with a oscillating high and low signal. The frequency of the oscillation determines the pitch of the note. This frequency is inversely proportional to the period of the note according to:

$$\text{Frequency (Hz)} = \frac{1}{\text{Period (seconds)}} \tag{1}$$

The unit of "Hertz" (Hz) means number per second. A 440Hz signal (called A4 by musicians) will have 440 periods per second and each of these periods is 2.27ms. You need to implement a routine to play all the notes from C3 (131Hz) to B5 (988Hz) as listed in the main.cpp file provided.

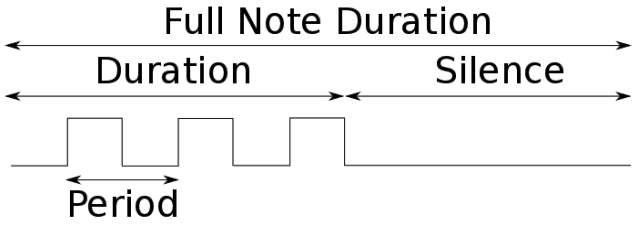

Figure 1: A "note" has an on tone and a silence that fill up a full note duration. The frequency of the note is determined by the inverse of the period of oscillations.

6.3.1 Timer 0 Usage

Timer 0 is already set up for you by the init() function in main.cpp. This timer provides the functionality in the delay() subroutine that you should call to specify the duration of notes. You use the delay function just like in the previous lab by placing the number of ms of delay in register a0 and then jumping-and-linking to the delay subroutine. It will return after the spcified amount of time. **You may NOT use this delay function to implement the different frequency waveforms. You can only use this delay function to specify the duration of a note or silence.**

6.3.2 Timer 1 Setup

Timer 1 should be used to implement the variable frequency tone. Depending on the PlayNote command, you should set up the PR (period) register to cause an interrupt that will trigger each high and low transition. Your interrupt service routine (ISR) should handle these interrupts and toggle the output waveform to create the oscillation and the tone in the speaker.

While calculating the value to put in the PR register, you should consider the following formula:

$$\text{Period} = \frac{f_{clk}}{256} \times \frac{1}{2} \times \frac{1}{\text{Frequency}} \tag{2}$$

where f_{clk} is the PIC32 clock frequency (what was this?), and Frequency is the note frequency to play. (Why do we need the extra $\frac{1}{2}$?) The above conversion will require the use of a division instruction and the special purpose HI and LO registers. You can access these registers using the mflo and mfhi special instructions. Please refer to the instruction reference to see how the divu, mflo and mfhi instructions work. Also, note that most of the above terms are constants and can be precomputed to simplify the math – only the frequency part changes. Also, a frequency of 0Hz should be treated as silence. However, **make sure not to divide by a zero frequency as this will cause an exception and your code will not continue running!!** Given the above information, you should complete the T1_ISR subroutine.

7 Requirements

In your assignment, you must:

- Configure pin 9 (Port D, pin 3) of the Uno32 board as a digital output
- Use Timer 0 (via the provided "delay" subroutine) to control the note/silence durations
- Use Timer 1 and interrupts to control the note frequency of oscillation
- Modify the Timer 1 period to control the frequency of the signal that drives a piezoelectric speaker
- Play the tunes provided in main.cpp that have variable note frequencies and durations
- Only modify main.cpp if you add the extra credit in Section 9

In order to do the above, you must write at least the following subroutines that are declared as globals and are called as C functions from main.cpp:

- SetupPort - no arguments, no return values
- SetupTimer - no arguments, no return values
- PlayNote - 3 integer arguments (frequency in Hz, duration in ms, full note duration in ms), no return values

In addition, you must complete the interrupt handler, T1_ISR.

8 Suggestions

It is suggested that you follow the following steps to get your program working:

- Configure pin 9 (Port D, pin 3) for output similar to lab 5
- Play a single frequency note of 440Hz on pin 9 with the piezoelectric speaker (search the web or look in the forum for a 440Hz reference tone to verify)
- Play configurable multiple frequencies of notes using the define statement notes in main.cpp (test with the scale test)
- Play a variable duration note (test with the other two melodies provided or write your own)
- Play the provided melodies

You should only demonstrate the final step of the whole tunes playing.

In addition to the above steps, please consider these hints:

1. Be careful to disable the timer when modifying its behavior and re-enable it when done. Otherwise, you could get an interrupt while changing the behavior.
2. Please note that other interrupts are already enabled for Timer 0, so do not over-write the configuration bits for other peripherals (e.g. T0IE) when starting Timer 1!
3. Do not divide by zero or you will get an exception.
4. Be sure to use proper subroutine calling conventions. If not, it is easy to over-write registers when calling subroutines or in interrupts. I provided two assembler macros for push/pop that can help with this.
5. Make sure to put nop instructions after all branches and jumps.
6. I found it useful to add some additional subroutines in my assembly code to enable and disable the timer. These were called in my assembly code whenever I wanted to configure a new note or start the silent period after a tone.
7. The provided tunes assumes that a frequency of 0Hz is silence. This is a special case where I turned off the timer 1 completely. This also requires that I do NOT perform a divide by zero or I would get an exception that my program cannot handle. You can implement the silence in any way you chose.

9 Extra Credit

It is required that you complete the rest of the lab first and that you demo the extra credit during your lab check-off. You cannot get extra credit if your assignment is late and you cannot do a second check-off with the extra credit.

You can receive 10 points of extra credit for writing a unique song in main.cpp. This can be any known song, a unique jingle (a Nintendo theme?), or your own creation. If it is your own,

it should have some artistic merit as judged by the TA/tutor during sign-off. If it is not your own, the song must be reasonably accurate. Demonstrate it to your TA/tutor during check-off to receive the extra credit. In your final report, please specify how you got your melody and what it is supposed to be.

10 Lab Submission

Your lab will be submitted via your eCommons account. Please log in to eCommons using your UCSC account and attach the following files to your "Lab 6" assignment submission:

- lab6.s
- lab6_report.pdf

Note that the final report must be submitted in PDF format.

Make sure to confirm that your assignment is SAVED and SUBMITTED before the deadline. You may resubmit your assignment an unlimited number of times up until the due date.

10.1 Check-off

For this lab, as with most labs, you will need to demonstrate your lab when it is finished to the TA or tutor and get it signed off. You will also need to submit your lab files using eCommons.

10.2 Grading template

This is a suggested grading rubrik. It is also a good general guideline before submitting your lab to check off these points.

10.2.1 Requirements

☐ (10 pts) Subroutines

- Are subroutines used?
- Are arguments given to the subroutines in a0..a3? Return values in v0..v1?
- Are caller/callee save conventions properly followed?
- Is the stack properly used to save/restore registers?
- Is there a SetupPort subroutine?
- Is there a SetupTimer subroutine?
- Is there a PlayNote subroutine?

☐ (25 pts) Timers

- Is Timer 0 only used for note durations?
- Is Timer 1 used for the note frequency?

- Is the ISR for timer 1 used?
- Does the ISR save/restore registers?
- Does the ISR get called and re-enable interrupts?

☐ (25 pts) Audio Output

- Can the program play audio on the piezo speaker through pin 9?
- Can the program play the various frequencies of tones?
- Can the program play various duration notes?
- Does a partial note include silent space to make a full note duration?
- Does the scale properly play?
- Does "Mary had a Little Lamb" properly play?
- Does "Shave and a haircut" properly play?

☐ (10 pts) Extra Credit

- Is there an extra melody in main.cpp?

10.2.2 Lab write-up requirements

In the lab write-up, we will be looking for the following things. The lab report is worth 40 points. We do not break down the point values; instead, we will assess the lab report as a whole while looking for the following content in the report.

Along with the usual items, you should answer the following questions:

- How did you go about designing your program?
- How did you convert the frequency to a note duration?
- How did you implement the silent part of a note after a tone?
- What sort of bugs did you encounter while writing your program?
- Why do we need the extra $\frac{1}{2}$ when calculating the prescalar register value?