Brandon Rullamas – brullama

Lab section: 4 TTH TA: Shea Ellerson

Due: 6/5/2013

Lab Partner: None

Title:

Lab 6: Timers and Interrupts

Purpose:

The purpose of this lab is to learn about the Arduino, MIPS assembly language, and PIC32 by creating two programs: creating a program that will take in an argument as a number of milliseconds which will make a LED on the Uno32 board blink on and off, and a program that uses the stack and reverses the order of the characters in a string.

The purpose of this lab is to learn more about the Arduino, MIPS assembly language, and PIC32 by creating a program that plays noises/songs based on a combination of a program in MIPS and a program in C. In order to correctly play the noises/songs, the Arduino uses timers and interrupts.

Procedure:

Because the lab was very similar to the previous lab where the Arduino was made to blink an LED on the board, the process for this lab was very quick as a lot of the concepts were the same. First, the lab pdf was read thoroughly to gain a good understanding of what had to be done to properly complete the lab. Then, the resources that were given to us were looked at by going through each of the slides of the PowerPoints and looking at how each part of the presentation would play a part in the lab. The final part of preparation was to look at the sample code that was provided and then carefully examining that to ensure a thorough understanding of the material.

The process of this program was broken up mainly into four different parts. The first part was to do the setupPort() subroutine. This mostly consisted of things from lab 5, so the code from lab 5 was simply copy/pasted to lab 6 and the values were changed to fit the bits and ports of lab 6. It was then tested using an infinite loop to ensure that the Arduino could play sound. Any miscellaneous information that was needed was found in the lab pdf, and the procedure needed for it was thoroughly explained there.

The next part of the program was to complete the setupTimer() subroutine. By this time, the TA had given us a thorough outline of the algorithm that was needed which made this process much simpler. To set up the timer, the prescalar has to be set, the counter needs to be cleared, the period value needs to be set, the interrupt priority needs to be set to 4, and interrupts need to be enabled. The hardest part about this part of the process was that the outline as well as the pdf said that the prior interrupts needed to be cleared, but when it came to testing as well as asking the TA, this was deemed unnecessary.

The third part of the program was to complete the ISR subroutine. The first part of this subroutine was to push all of the registers used to ensure that the stack was being utilized, and then pin 9 on the Arduino would be toggled. After that, the counter and prior interrupts were cleared. Finally, all of the registers that were pushed at first were then popped. The process for learning what to do for this subroutine involved what the TA provided as well as reading the lab pdf.

The last and final part of the lab was to finish the playNote() subroutine. This involved the most trouble but it was relatively quick in figuring out. First, the period was calculated using the formula that was provided in the pdf which is described in the section, “Algorithms and other data”. After that, the period was set. This involved some trouble as it was eventually found out that to properly set the period port, the entire port would have to be set, instead of using an offset of 4, 8, or 12 to clear, set, or toggle it respectively. After that, the timer was turned on and the delay function from the main.cpp program was called. The timer was then turned off and the delay function was called again. A branch statement is used at the beginning of this subroutine to check if the frequency is 0, and if it is, it branches to the end of the subroutine where it simply returns to where it was called.

Algorithms and other data:

There was one main algorithm that was needed in relation to the music to properly play music. The first one that was noted in the lab is that the Frequency (in hertz) is equal to 1 over the period (in seconds). So, if there is a 440Hz signal, there will be 440 periods per second, meaning that each period is 2.27 milliseconds. The second piece of information was that the period is equal to the frequency of the clock divided by 256, which is halved, which is then multiplied by 1 over the frequency. Other than that, most of the algorithms that were needed for this lab were thoroughly outlined in the lab pdf as well as through any assistance that the TAs and tutors provided.

What went wrong or what were the challenges?

The main problem that occurred with this lab was getting the speaker properly set up to play when it was wanted to and that it didn’t fall out right in the middle of testing. Other than that, this lab was relatively painless and quick to complete, especially in relation to something like lab 4, which was very hard.

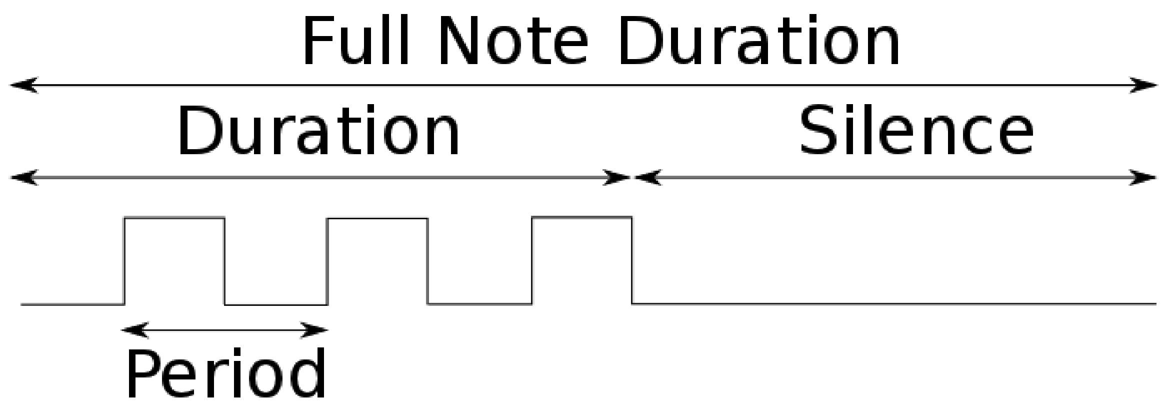
Other information:

How did you go about designing your program?

This program was completed in steps very similar to the way the lab pdf was laid out. The steps are described above. Other than that, the resources that were provided online (the MIPS powerpoint, the lab pdf, and the lab code outline) were used to gain the information needed to properly finish the lab.

How did you convert the frequency to a note duration?

To convert the frequency to a note duration, the inverse of the period of the oscillations was used. The following picture thoroughly describes the system behind this:



How did you implement the silent part of a note after a tone?

To implement the silent part of a note after a tone, a third argument was passed to the playNote subroutine which determined the full note duration. This allows for there to be a silence after the given duration of a note as well as for the separation of the notes when playing a tune.

What sort of bugs did you encounter while writing your program?

The main bug that was encountered while writing the program was that the entire PR1 port had to be set instead of using an offset of 8. After figuring this out, the rest of the program wasn't terribly complicated and there were no major bugs encountered.

What do we need the extra ½ when calculating the prescalar register value?

The extra ½ is needed when calculating the prescalar register value because of the picture above. The duration that the frequency will be playing only takes up half of the full note duration, whereas the other half is silence.

How did you get your melody and what is it supposed to be?

My melody that I added in was the Super Mario Bros. theme song. I got it by looking at sheet music that was available online and then matching the notes with the proper notes that were predefined in main.cpp. I was unable to read sheet music beforehand so it took a bit to learn what notes were what, but after a little practice, it just took time to go through the main portion of the song. In terms of the timing, it was just lots of testing.

Conclusion:

This lab was relatively simple for where it stands in the course as well as in relation to the previous two labs. It wasn’t easy enough to where it could be completed in a day and it still provided a nice challenge, but it wasn’t hard enough to where there was frustration and sadness that was associated with the lab. The lab as a whole was a good learning experience and it was fun to make my own melody that would then be played through the Arduino.

Extra:

The algorithms and tips given by the TAs were helpful. If it had not been for them laying out the processes needed for each subroutine, the program would have probably taken a bit longer to figure out as the lab pdf is much more wordy than the outline given to us by the TAs and it makes it harder to figure out what the lab pdf is trying to say is needed for the lab.