Lab 5: Digital Audio

Peter Johnson[1]

E155 Fall 2018 Section 2, October 11, 2018

***Abstract*— This paper describes the circuit design, programming and testing of a Raspberry Pi such that it can run a C program to play a song on an 8-ohm speaker.**

I. Introduction

In this lab, the MuddPi Mark IV utility board was not utilized at all. Instead, the lab was centered on gaining familiarity with the Raspberry Pi and accessing GPIOs and timers through C programming. Once the Pi was running, I wrote C code to access the system timer to set up a counter for use in a delay function. From there I implemented a function to read in a list of notes specifying the pitch in Hz and the duration in ms.

II. Design and Testing Methodology

I approached the design of the system by first getting the code on the Pi running. The first thing I did was to read through the starter code to understand what it did, and how I would probably interact with it. From there, I started by implementing the pinMode, digitalRead and digitalWrite functions we implemented in class. After confirming that those worked, I moved on to implementing a delay function. From the Pi's BCM2835 datasheet I found information on the System Timer register. I learned that there were four compare registers and several counters. To measure an amount of time, the current time (found from one of the free running counters) was added to the amount of time that we wanted to measure and then put into the compare register. The flag of the match bit for that register would then be reset to 0. Each time the counter incremented its value would be compared to the value in the compare register. Once there was a match between the two, the match bit would change to a 1, notifying the system that the amount of time in question had elapsed.

To implement this, I made a pointer to the base address of the System Timer using the same memory mapped method as the GPIO in the starter code. From there, I created pointers to each of the addresses that I would need just by adding offsets to the original pointer. To me, this made it easier to see what I was accessing than trying to subscript the base address of the System Timer. The way it worked is I would dereference the pointer to the lower counter, add the time I wanted to delay for (in microseconds since the clock counted 1 microsecond at a time) and then put that value in the compare register C2. Next, I would write a bit to M2. which was the 3rd bit in the CS register, so I ORed the CS register with 0b100 to only affect that single bit. The last part of the delay function was a while loop which checked that the value of M2 was still 0. As soon as it read 1 it meant the time had elapsed so the function would end. Once I had the delay function I used this to make a playNote function which took in the desired frequency and time in ms. I used the frequency to calculate the period of the signal. I then wrote a for loop to write a 0 for half the period and then a 1 for the other half of the period. The number of loops was determined by dividing the duration by the period. Implementing a song was then just looping through the array of tuples and calling playNote for each of them. To ensure I did not damage the speaker, a simple Op-amp circuit was constructed to increase current flow using an LM386 amplifier.

My testing methodology was simple and just consisted of several iterative trials all in the hardware. The first test was just ensuring my pinMode(), digitalRead(), and digitalWrite() functions worked properly. I did this just by hooking up wires to pins on the Pi pinout. I would then check with a multimeter if the correct value was being produced. Once I had written a delay function I checked that a square wave could be produced at a pin. After that, all my testing was just done listening to the output of the speaker since it was easy for me to tell whether something was correct.

III. Code and Schematics

Refer to attached the code and schematics.

IV. Results and Discussions

All of the prescribed tasks were accomplished: the Raspberry Pi's GPIO pins were accessed through a memory map and can now be set to a mode, written to and read from. The system timer was also accessed through a memory map and in conjunction with the GPIO functionality, a delay function was created. This delay function was then used to play notes. From this ability to play notes, I wrote a function which allows the Pi to play songs. In addition to playing Fur Elise, I wrote an additional short piece for my Pi to play. Once Fur Elise is done, the Pi briefly pauses and then moves into the opening theme of the Good the Bad and the Ugly.

V. Conclusions

The lab was successfully completed. The Raspberry Pi's GPIO and system timer can be accessed, and simple songs can be played on the speaker. I gained experience with reading datasheets for new devices, writing memory mapping functions, and basic C like pointers and arrays. In total I spent 9 hours.

[1]Peter Johnson is with Department of Engineering Harvey Mudd College, Claremont, CA 91711 pjohnson@hmc.edu