ECE 477 Lab 3

Christian Knight, Nikko Noble

September 16, 2020

Introduction

The purpose of this lab is to interface with the Intel 8253 timer chip that is present in all PCs. Specifically, the goal is to develop a C routine that a non-superuser can run using the provided program `ioshell` that can measure real time on the timer's unused channel and output the measurements to a `.csv` file. Additionally, a C routine was pursued that uses the timer as a second counter that increments by one every second and prints the elapsed time to the terminal.

The 8253 has three independent timer channels; the first and second channels are used for the PC's real time clock and dynamic memory refresh respectively, where the third channel is generally connected to a speaker internally to generate an audible tone when the PC boots up. The third channel can be accessed and manipulated without disturbing the other two timers.

Part A

This section covers the development of the code that is used to interface with the 8253 timer. The code starts by creating a `.csv` file to log the results to and then proceeds to loop through delay values ranging from 1μs to 2000μs ten time each, using the timer to measure the actual delay. Running the program using `ioshell` allows a non-superuser to use the program since `ioshell` temporarily allows access to the I/O ports to run the desired program.

To run the main program, `ioshell.c` needs to be compiled as well as the main program `partA.c` with `timer_func.c` which holds the required functions.

```
gcc ioshell.c -o ioshell
gcc partA.c timer_func.c -o partA
```

In order to access the I/O ports requred for the timer, both output files `ioshell` and `partA` need to be owned by root.

```
sudo chown root:root ioshell
```

```
sudo chmod +s ioshell
sudo chown root:root partA
sudo chmod +s partA
```

The main program can then executed by passing it as an argument to `ioshell`.

```
./ioshell ./partA
```

Within the main program, variables are initialized first, and then a file `partA.csv` is created, which will contain the logged results. The `.csv` has two columns; one column to hold the expected delay values from 1μs to 2000μs that are given to `usleep()`, and the second column holds the actual time that it takes for `usleep()` to return from each delay using the timer.

To measure elapsed time over a call to `usleep()`, the timer should be set to its maximum value (0xFFFF) before starting and then calling `usleep()` since the timer will count down. Then, the timer should be stopped and the timer value should be subtracted from the starting value of 0xFFFF to determine how many 'tics' occurred during the execution of `usleep()`. The timer runs at 1.19MHz, so the conversion from ticks to elapsed time in μs is ticks/1.19.

It was found experimentally (using the provided `inp` and `outp` programs and `ioshell`) that setting the least significant bit of port 0x61 starts the timer, and clearing stops it. According to the Wikipedia page on the 8253 timer, 0x43 is the control register to write to for configuring the timer mode, and setting the port to include the value 0xB4 sets timer 2 into counting mode 2, in which case it counts down from its current value each cycle. [1]

The program measures the actual delay that `usleep()` gives for each expected value by executing the following sequence:

1. Turn timer off by clearing LSB of port 0x61
2. Set counting mode of timer 2 to 2 (count down) by setting port 0x43 to contain the value 0xB4
3. Initialize counter value to 0xFFFF by setting all bits of port 0x42 twice
4. Turn timer on by setting LSB of port 0x61
5. Call `usleep()` with desired delay
6. Turn timer off
7. Get value of timer from port 0x42
8. Calculate number of timer ticks during `usleep()` execution
9. Calculate time elapsed from ticks
10. Log expected and actual delays to `.csv`

11. Repeat 1-10 ten times

12. Repeat 11 for next delay value

The execution of this sequence results in a total of 20,000 measurements. These results were analyzed in Matlab to determine how well `usleep()` delays over the range of values. Figure 1 shows all of the recorded data for expected delay versus measured delay.

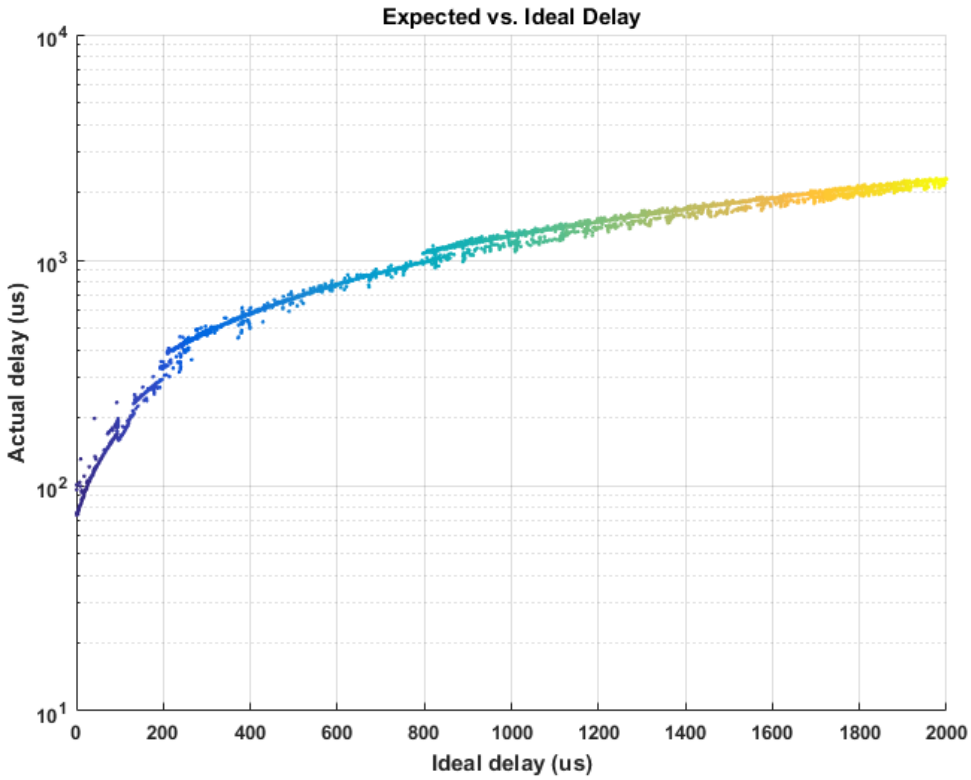

Figure 1: All data points of the ideal versus expected delay from `usleep`

The scatterplot above show a increasing trend for the actual delay versus the ideal delay time.

Next, the average of the actual delay was interpolated over 10 iterations of `usleep`.

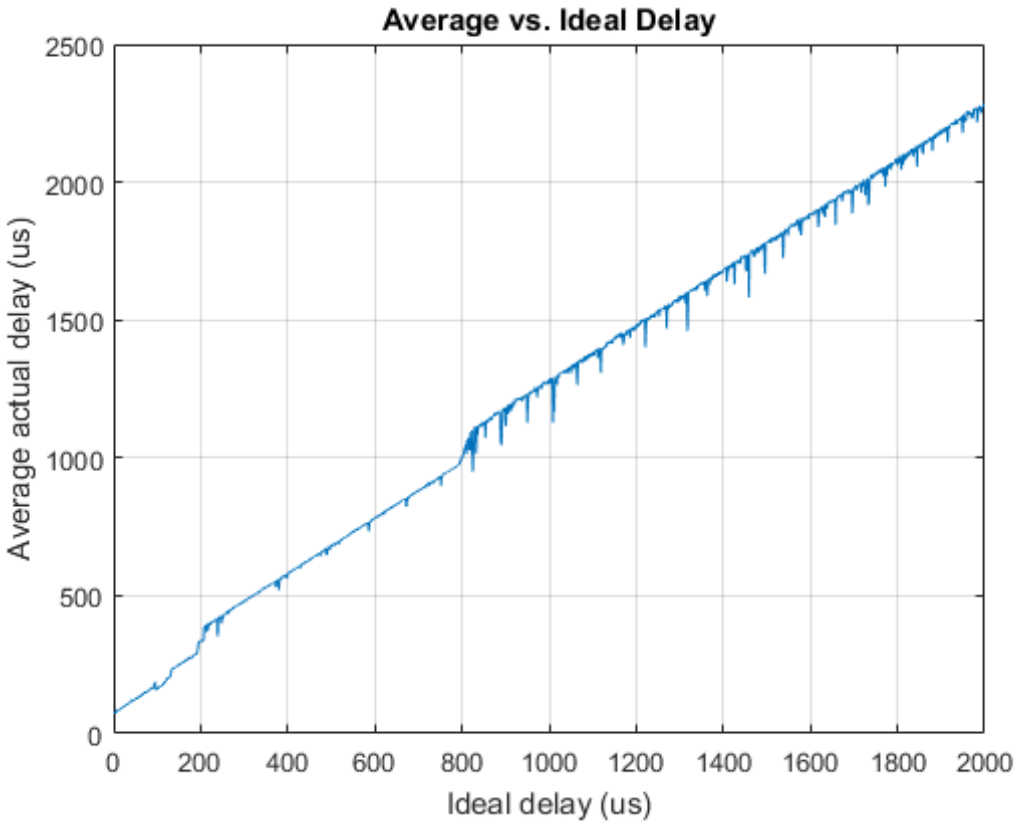

Figure 2: All data points of the ideal versus average expected delay from `usleep`

From the Figure above, the data follows a linear increase in the actual delay versus the ideal delay .

The percent error of the ideal versus actual delay time can be seen in Figure 3.

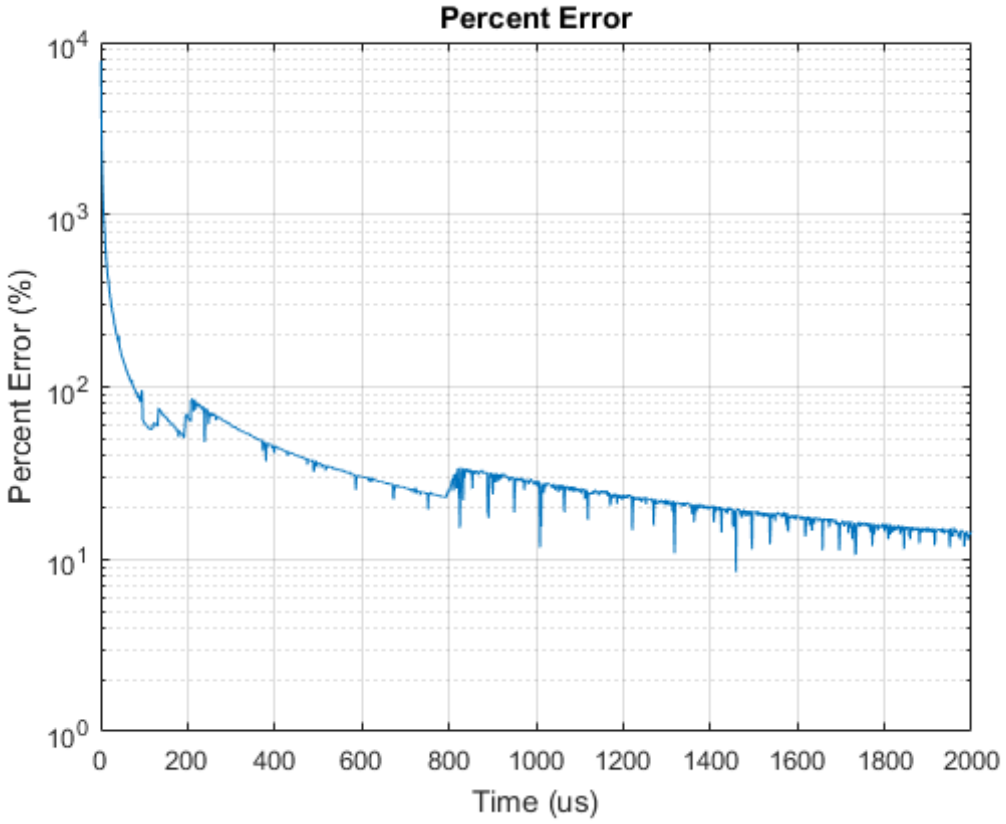

Figure 3: Percent error from ideal versus average expected delay

Figure 3 shows that when calling the `usleep()` function for small delay values, the error is very large due to the overhead associated with calling the function. Since the overhead should be the same regardless of the desired delay, the percent error decreases with increasing delays, so that it is close to about 10% for 2000μs

Part B

For an interesting use of the timer in part B, a program that counts up by one every second and prints to the terminal was desired. The program followed a similar format to that of Part A, but runs in a `while(1)` loop and waits for the timer to complete several ticks.

The program failed to work as expected; it counted much too fast. Since the counter is 16 bits and runs at 1.19MHz, it takes 55ms for the timer to run from its maximum value to minimum value. Therefore, this is the time that it took for the program to count to the terminal. The timer needs to wrap around several times before one second has elapsed. This is most likely the reason that the code did not work as desired.

Conclusion

The programs developed for interface with the Intel 8253 timer chip present in all PCs has been described. Specifically, the goal is to develop a C routine that a non-superuser can run using the provided program `ioshell` that can measure real time on the timer's unused channel and output the measurements to a `.csv` file. A C routine was pursued but not completed that uses the timer as a second counter that increments by one every second and prints the elapsed time to the terminal.

The results from Part A showed how the overhead associated with calling the `usleep()` function affects the accuracy for small delay values. The overhead stays relatively constant for increasing delay values, so the error decreases.

A Part A Source Code

Attached: `partA.c`, `timer_func.c`, `timer_func.h`

B Part B Source Code

Attached: `partB.c`, `timer_func.c`, `timer_func.h`

Sources

[1] Wikipedia. *Intel 8253*. Jan. 20, 2017. URL: https://en.wikipedia.org/wiki/Intel_8253.