|  |
| --- |
| University of Idaho |
| Project 4 |
| ECE 443 |

|  |
| --- |
| Donahue, Ryan (dona1865@vandals.uidaho.edu)  10-9-2018 |

## Introduction

This lab explored a new yet familiar protocol system management bus, or SMBus. This communication protocol was invented for battery management systems. It carries useful information about device manufacturers, model/part number and different errors between master, slaves and hosts. The protocol is based on Phillips I2C, but has some key differences: Minimum clock frequency of 10 KHz, low-time clock out, specific command protocols and NACKs to indicate invalid commands or data.

## Objective

To explore the protocol, I implemented a system to measure object temperature. To do this, I used a Melexis MLX90614 Infrared Thermometer. Temperature data was retrieved from the device’s internal RAM, and displayed on the LCD in °F. A control flow diagram for the system is shown in figure 1.

### 

Figure : System Control Flow Diagram

## Hardware Verification

Figure 2 shows the data transfer while reading from the IR sensor’s internal RAM much like figure 8 in the products datasheet.

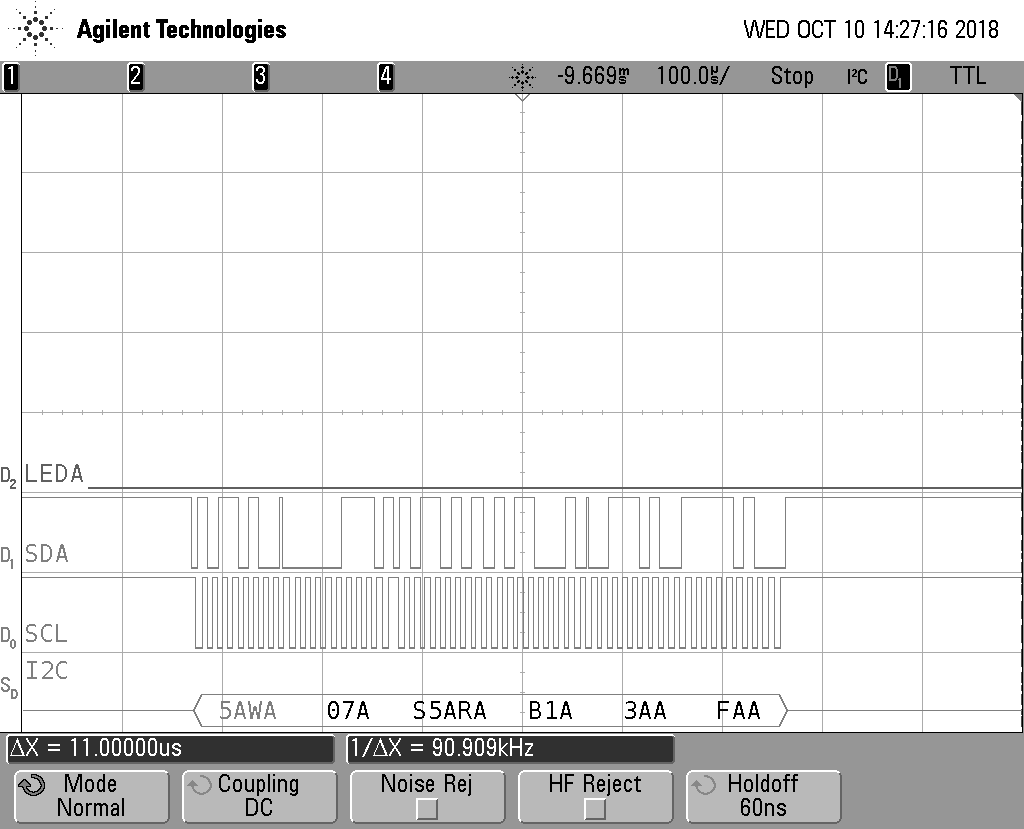


Figure 2: SMBus IR Sensor Read Transaction

## Tracealyzer Verification

I was unable to achieve a 3 millisecond blink rate for LED A with the task at null priority. I attempted to solve this issue by yielding LCD write and SMBus read tasks, but this led to variability in IR sensor read and LCD write task’s execution rates. If I raised the task’s priority to level 1, the average LED toggle was right at 3 milliseconds, but the project specifications were violated. I am not entirely sure how to get around this tradeoff. However, I was able to use Tracealyzer to troubleshoot the problem.

### Idle Priority Blink Task

The view in figure 3 clearly shows the equal execution rate of the four tasks. Due to higher priority events (red-3, yellow-2, green-1) requiring about 3 milliseconds execution time, the 3 ms time requirement has already passed, LEDs are set immediately on servicing, and roughly 2 milliseconds elapses while polling the core timer. Then the red task starts the IR sensor read/write cycle again. The result was every cycle taking 5 milliseconds.

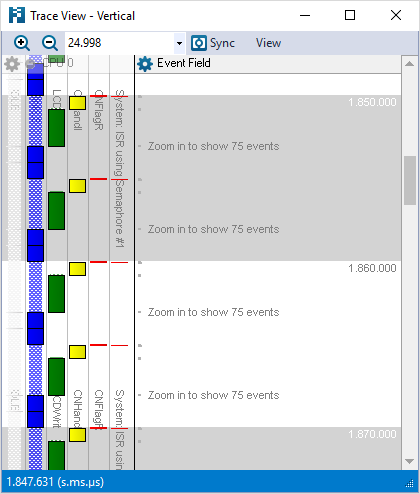


Figure 3: Idle Priority Trace View

### Priority One Blink Task

By raising the priority level of blinking task to match LCD writes, the result was much different. Figure 4 shows the timing in trace view. Now we see execution of the blinking task executes before LCD writes, because it has been in the ready state longer. By checking the core timer four times in a 10 millisecond interval, it is very likely that an accurate 3 millisecond toggle would happen. Figure 5 verifies this assumption.

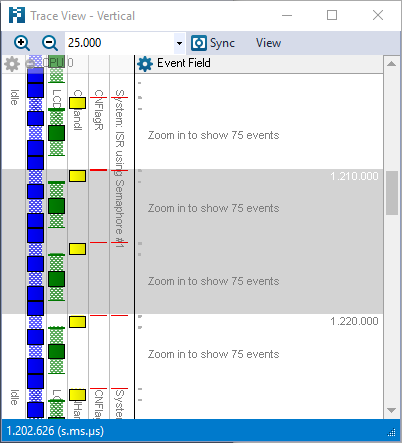


Figure 4: Priority 1 Trace View

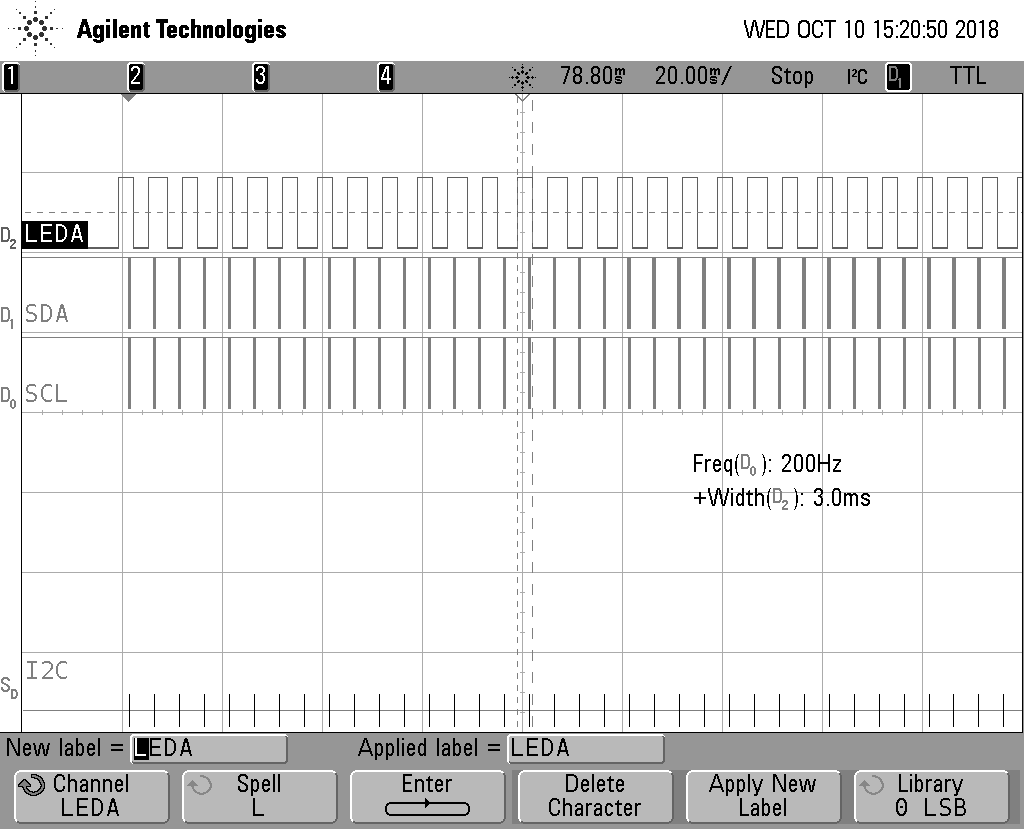


Figure 5: 3 Millisecond Blink Rate

### Additional Verification

I thought a pretty interesting tool was the communication flow tool. It basically created the CFD I drew for figure 1.

## 

Figure 6: Communication Flow

As discussed, program execution rate was very consistent. Figure 7 shows the CPU load.

## 

Figure 7: CPU Load

## Conclusion

What a fun project. I enjoyed actually using the Tracealyzer tools features to diagnose timing. If I were making this project without strict priority requirements, I would implement the modification I justified above. I am still pretty baffled about Doxygen and LaTeX, despite making a report with all my functions appearing. For future projects I would like to investigate other configurations, and further customize my report.