Lab 6: Internet of Things

Peter Johnson[1]

E155 Fall 2018 Section 2, October 18, 2018

***Abstract*— This paper describes the circuit design, server setup, programming and testing of a Raspberry Pi such that it can control an internet accessible LED and light sensor.**

I. Introduction

In this lab, the MuddPi Mark IV utility board was utilized in conjunction with the Raspberry Pi. Apache2 is run from the Pi allowing an LED on the board to be turned on and off via a website. A photodiode is used to construct a light sensing circuit which is connected to the board's analog to digital converter whose voltage can be retrieved and displayed through the online interface.

II. Design and Testing Methodology

I approached the design of the system by first getting the web server running on the Pi. This mainly consisted of following the instructions to setup Apache2 and get the LEDON.c and LEDOFF.c files running from the ledcontrol HTML page.

Once those portions were running I turned my attention to getting the SPI to run. The first step was setting up the pointers to the registers through memory mapping and then setting the appropriate pinmodes. Next I examined the MCP 3002 ADC datasheet and the Mudd Pi schematic for how to wire up the Pi to the ADC. I also learned what bits I would need to send to initialize the ADC for communication with the SPI. This was X110 1XXX. Where 1 was the start bit, 10 selects channel 0 and single ended mode. The last 1 sets the data to most significant bit first. The building block of SPI usage is being able to send and receive 8-bit packets, so I wrote a function to do that. I do this by setting the transmit value into the FIFO register, a CLK divisor value into the CLK register, and all of the CS bits to 0 (default). To send a message, the Transfer Active bit is set high. When a message is received, the done bit goes high. A problem was the ADC uses 10-bits, so I wrote a function to send two packets and then receive two packets for a total of 16 bits received. From there I tried to use an and to get rid of the upper 6 bits. This was now a value from 0-1023. To convert, I used this calculation where Vref was 3.3V:

$$V_{out} = ADC_{val} * V_{ref} / 1023$$

The most original piece of work that I did was probably in the creation of my html for accessing the ADC. I added another button to the ledcontrol page which would run my ADCREAD executable. To deal with maintaining a web interface, I copied the code from the ledcontrol.html and put it into a print statement in my ADCREAD. So when ADCREAD is called, it displays the same html page with the same functionality. There is just one addition to the page produced by printf-a line to display the value of the voltage variable. This variable is accessible since the printf occurs inside my main function. I did not go to the trouble of trying to make it available on both pages and continuously update.

My testing methodology was simple and just consisted of several iterative trials all in the hardware. The first test was just ensuring my index.html appeared when I navigated to my Pi's ip address. The next test was whether I could turn on LED0 remotely. Once this occured, I knew I could properly run executables in my cgi-bin. The longest testing and debugging process was for the functionality of the SPI. I inserted print statements after every step starting from initializing the SPI, loading a char into the FIFO and then whether the done bit had been raised. Once I knew for sure I was receiving a done bit, I started printing out the value as both an int (converted from the unprocessed short) and then as a processed float value which should correspond to the voltage. At this point I could see that I was reading values and whether they were correct or not since I was using a power supply as the voltage I was reading.

The last task to do was to build the phototransistor circuit. It was simple to find a schematic and then use trial and error to find a resistor value that would produce a sufficiently large voltage swing to take advantage of the entire reference voltage.

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 LED can successfully be turned on from any computer online as long as they have the Pi's ip address and are on Claremont-WPA. The SPI on the Pi was successfully accessed through the same memory mapping process used in previous labs. The SPI is then able to successfully read values from the ADC on the Mudd Pi board. These values are then converted to voltage and displayed online.

V. Conclusions

The lab was successfully completed. I learned how to access a new peripheral- the Raspberry Pi's SPI register and how to interface it with a slave device. I gained experience with reading datasheets for new devices such as the ADC, writing memory mapping functions, and writing both C and HTML. In total I spent 16 hours, it was a long lab.

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