Project
CMPE 121, Professor Varma

Stephanie Salazar

14 June 2017

Introduction

This project's goal was to create a dual channel oscilloscope and an eight channel logic analyzer with the PSoC-5 microcontroller and Raspberry Pi 3 in two separate applications. The PSoC-5 should sample the signals to be monitored and the Raspberry Pi will visualize them. This lab uses the UART interface to send commands and stream data.

Block Diagram

0.1 Oscilloscope

Figure 1: PSoC-5 block diagram

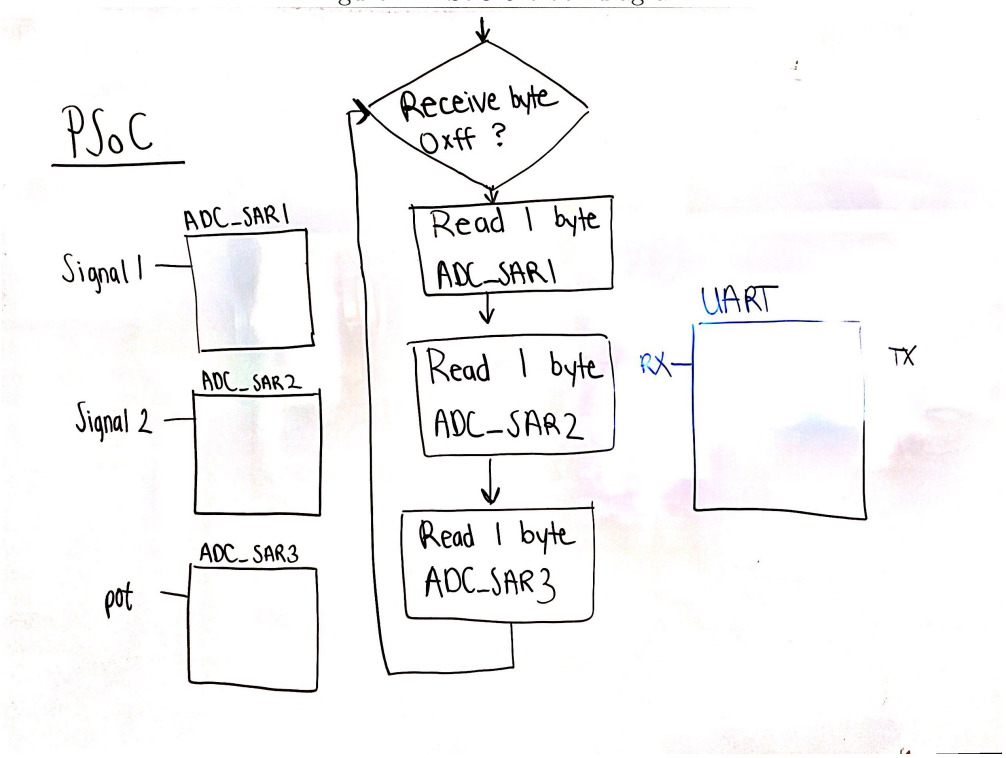
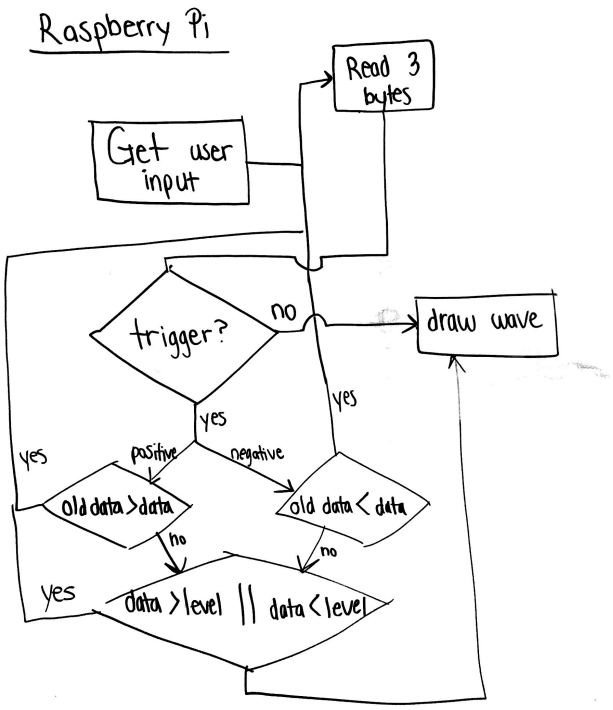
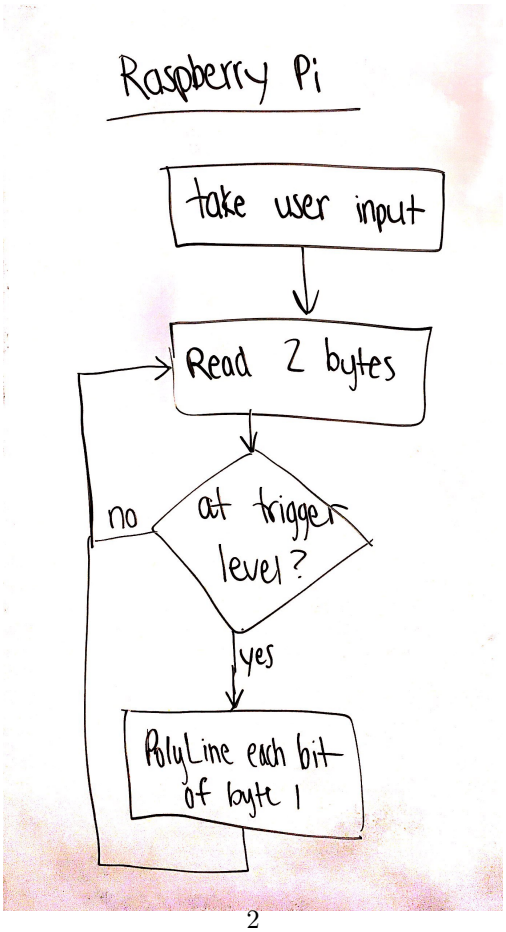


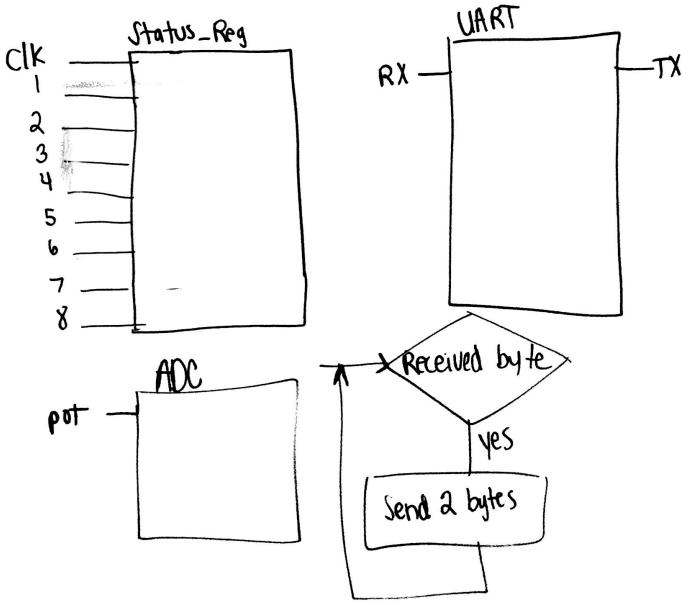
Figure 2: Raspberry Pi 3 block diagram



0.2 Logic Analyzer

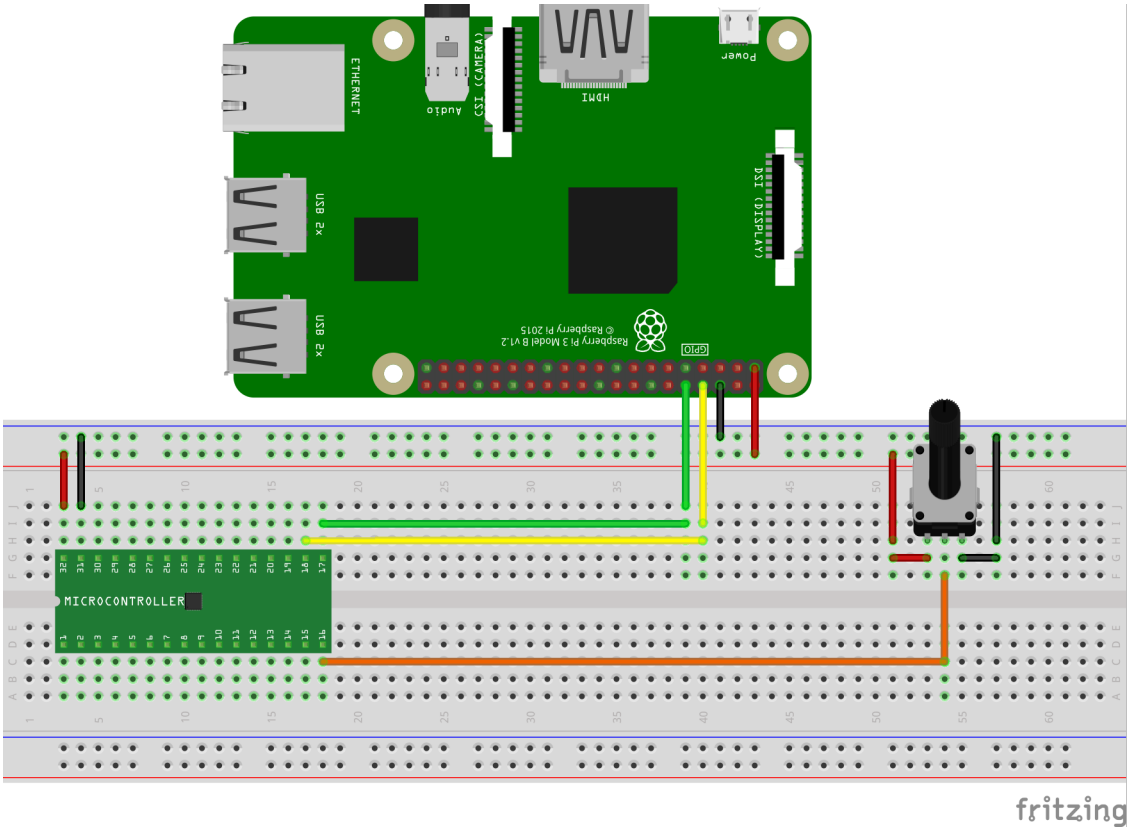


PSoC


Status_Reg
Clk
1
2
3
4
5
6
7
8
UART
RX
TX
ADC
pot
Received byte
yes
Send 2 bytes

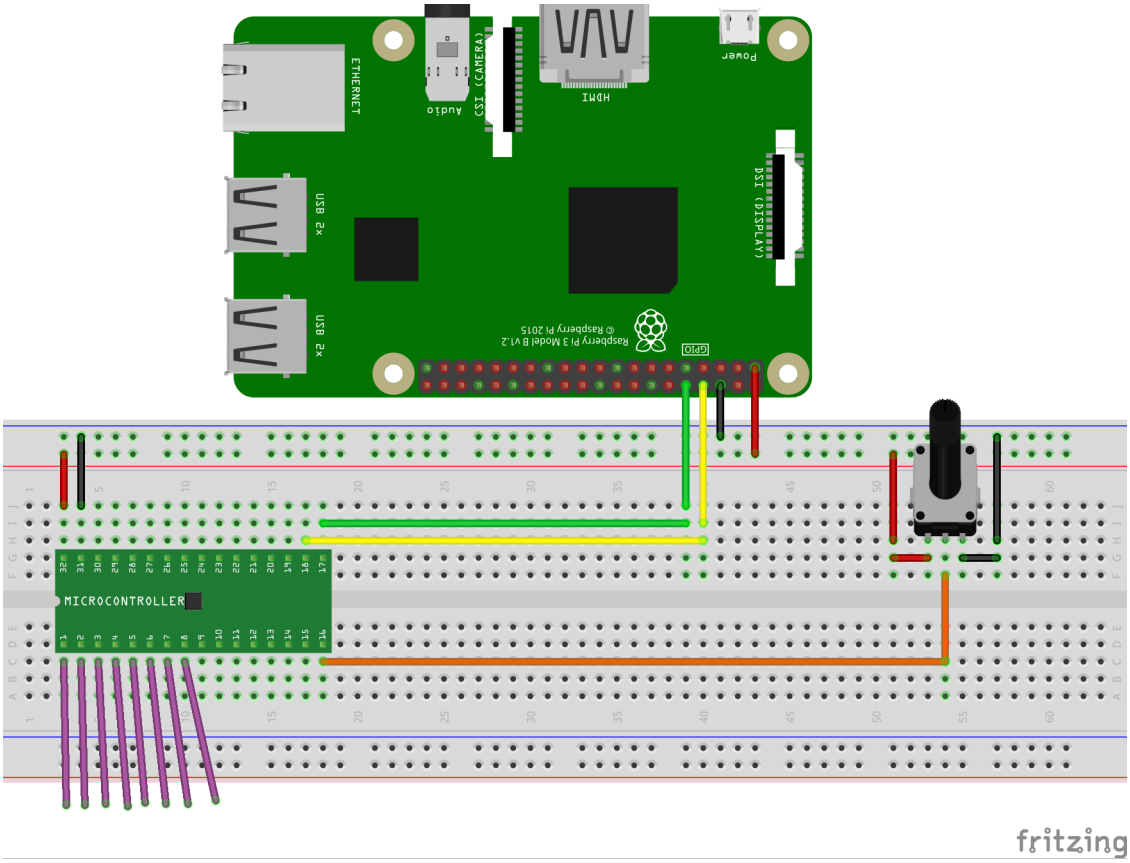
Schematics

0.3 Oscilloscope



Microcontroller to PSoC-5
Pin 32 Power
Pin 31 Ground
Pin 18 Pin12[1] RX
Pin 17 Pin12[1] TX
Pin 16 Pin1[7] Potentiometer

0.4 Logic Analyzer



Microcontroller to PSoC-5
Pin 32 Power
Pin 31 Ground
Pin 18 Pin12[1] RX
Pin 17 Pin12[1] TX
Pin 16 Pin1[7] Potentiometer
Pin 1 - Pin0[0]
Pin 2 - Pin0[1]
Pin 3 - Pin0[2]
Pin 4 - Pin0[3]
Pin 5 - Pin0[4]
Pin 6 - Pin0[5]
Pin 7 - Pin0[6]
Pin 8 - Pin0[7]

Description of Features

Oscilloscope

The oscilloscope is dual channel, meaning it can display two separate waveforms. It is programmed with a PSoC-5 and Raspberry Pi 3. The program takes in user inputs to set

certain features of the oscilloscope. The number of channels can be set to 1 or 2, which allows the user to view 1 or 2 waves. There are two modes- free running and trigger. If trigger is selected, the trigger slope and trigger channel must be selected. This will indicate whether the wave should start at a downward or upward slope depending on either the 1st or 2nd channel. A y-scale can be set to 0.5, 1, 1.5 or 2 which defines the vertical scale of the waveform display in volts per division. An x-scale can be selected from the values 1, 10, 100, 500, 1000, 2000, 5000 or 10000 to define the horizontal scale of the waveform display in microseconds. The oscilloscope starts when the user inputs 'start.'

Logic Analyzer

The logic analyzer can take in up to 8 channels. It utilizes the PSoC-5 and Raspberry Pi to view a signal's binary states. It takes in user inputs such as the number of channels the user wants to use. This can be between 1-8. A trigger condition can then be entered. For this data analyzer, it takes in a binary number and triggers at this state. The trigger condition can be positive or negative where positive direction specifies that the trigger should become active when the trigger condition changes from false to true, and negative direction specifies that the trigger should become active when the condition changes from true to false. A memory depth can be entered, so that the size of the window can be set. The sampling frequency can be taken in which maxes out at the max clock speed. The x-scale can be set as 1, 10, 100, 500, 1000, 2000, 5000 or 10000, and 'run' is entered to begin the logic analyzer.

System Description

The system for this project utilizes Open VG to draw the waves and informatio. Thee PSoC samples the signals to be monitored, and the Raspberry Pi to analyzes them and visualizes them with Open VG. A UART interface is used, so that a signal is read in from the PSoC and transfered to the Raspberry Pi through TX and RX. The Raspberry Pi sends a signal to the PSoC letting it know that it is ready to receive data. Once the PSoC receives this byte, it sends out specific bytes.

For the oscilloscope, the PSoC sends out three bytes. The first two correspond to two separate signals. The third byte is the potentiometer value. When the raspberry pi receives this data, it sends back a byte. Thus, the cycle begins where the raspberry pi and PSoC communicate to each other to stay in sync. The raspberry pi draws channel 1 with the 1st bytes received, and the the second channel with the 2nd bytes received. The third byte is used to add onto the second channel, so that the user may move the signal to match it with the first signal.

The logic analyzer PSoC side sends out two bytes. The first byte contains the 8 bits that correspond to the 8 bits used for each channel of the logic analyzer while the second byte contains the value of the potentiometer. After the raspberry pi receives the two bytes, it sends back a specfic byte, so that the PSoC knows when to send the next two bytes. This works in the same way as the oscilloscope part of the project. Only the amount of bytes sent changes.

Hardware Design

Oscilloscope

For the oscilloscope part of the project, one delta sigma ADC was used to take in values from the potentiometer while two ADC SARs were used to take in the values from a signal. ADC SARs were used because we can only use one ADC delta sigma. A UART was used to transmit and receive the bytes. The TX transmits the bytes that are read from the ADCs and receives a starting byte from the raspberry pi to begin transmitting after each transmit. The UARTs baud rate was set to 115,200 with a stop/start bit and odd parity. This creates a frequency of 10,472.7 bytes/second.

Logic Analyzer

The logic analyzer hardware design for the PSoC utilized a status register to take in 8 different signals to put into one byte. This took 8 pins into the status register. An ADC Delta Sigma was used to read in the potentiometer value. A UART was set up to transmit and receive bytes, sending out two bytes: the status register byte and the potentiometer byte. It would receive a byte from the raspberry pi to begin this transfer each time. The UART was set the same as it was for the oscilloscope with a baud rate of 115,200 with a stop/start bit and odd parity. This creates a frequency of 10,472.7 bytes/second.

Software Design

Oscilloscope

The software side for the Oscilloscope has two parts. For the PSoC, I programmed it to read in a byte and check it against a specific byte that was supposed to be sent from the raspberry pi. Once this value checks out, it begins to send a byte after the ADC SAR is done converting. This is done twice for each channel. It then transmits a third byte reading from the ADC Delta Sigma. This is in a for-loop so that it is always reading the signal and potentiometer and staying synced.
On the Raspberry Pi side, this is where most of the software design comes into play. The program firsts asks for input based on the features of the program. These are taken in with a character array and the command fgets() which then checks if a valid entry was entered. This data is stored in a struct. Once this data is collected, if trigger mode was selected, it enters into if-statements checking whether the user wants a positive or negative slope. In each of these cases, it checks against the level entered to start the signal at a certain point and slope. New data is collected from RX until the right values are reached. If trigger mode is not selected, it skips this part and goes straight to plotting the signals. Plotting the signals is done through Open VG where the line function is used. I take the new data and connect it with the old data to form a line. With small enough increments, this creates a smooth wave. The window is cleared at the end of the for loop, so that a single wave always appears. The user inputs are then displayed on the monitor with text commands from Open VG.

Logic Analyzer

The software side for the Logic Analyzer for the PSoC is simple. It is similar to the oscilloscope software in that it checks for a specific byte from the Raspberry Pi before beginning its transmit of two bytes that are read from ADC SARs. It then begins a loop of checking with the Raspberry Pi before sending data so that it stays synced.
The Raspberry Pi side of the software is also similar to the oscilloscope, except that it is graphing 8 signals rather than 2, and splitting one byte into eight separate bits. User input is asked for again with fgets() and is compared against acceptable results and once the user inputs something valid, it is stored in a struct for later use. Once all user input is given, it checks against the trigger level and keeps reading new data until the trigger level is met. It then draws the signal on the monitor with PolyLine() which takes in a bit of the array that the first byte is stored in. After graphing each bit, the user inputs are displayed with TextMid(). This is in a continuous for-loop, so that the data is always changing with the signal that it is reading, but the user input is only asked for once at the beginning of the code.

Testing

Oscilloscope

To test the oscilloscope, I first connected a wave generator to a pin on the PSoC-5. The signal that appeared on my oscilloscope appeared to only be noise. I then hooked up this signal to an actual oscilloscope that also display noise. This led me to realize that the wave generator had a loose connection. I switched to sending a 256 entry lookup table of a sine wave. Once this appeared on the monitor, I found that my wave was drawing so that the lines were not always connected. I found that I was not resetting my old data correctly. Once I got my signal working with the lookup table, I connected it to a different wave generator which also showed a correct sine wave. When I switched to a square wave, nothing happened. I kept changing the code, until I realized it was just taking a long time for my wave to generate. I found that it took about 45 seconds for my wave to change.

Logic Analyzer

To test out the logic analyzer, I first used a timer to generate a signal and connect each bit to a channel. When the waves were still flat lines, I checked over my wiring again. I found that I had accidentally changed the TX and RX wires. After fixing this, I tried running the program again. I still could not see anything, so I specifically colored each channel to a color. I found that they were all still flat lines, but they were appearing. I figured out that they were only going up by 1 pixel, so I resized the waves. This fixed the problem. I then tried connecting the channels to wave generator, but I could not get a signal. I think that the problem could have been solved by changing the clock on the status register so that it was running at double the frequency.

Conclusion

This lab was a great way to put together everything I have learned about microcontrollers in CMPE 121. I was able to utilize both the Raspberry Pi and PSoC and use skills from previous lab and build from these to create an oscilloscope and logic analyzer.

It took the basics from lab 1 in learning how to set pins to certain values and taking in potentiometer values and using them to change values such as an LED, but for this lab. In lab 2, we learned how to use DMAs, which were a cornerstone in transferring data. Lab 3 helped with learning how to use a UART to transmit and receive data. Lab 4 taught us how to use a USB interface to transfer data, while lab 5 taught us the pros and cons of using UART vs SPI interfaces. This project helped me understand how to transfer data in an effective way, so that I could properly display data in a timely matter with certain constraints.