Artyom Martirosyan
Id 1682917
6/8/2021

Final Project Report

Introduction:

System Description:

In this lab the student will be implementing components from what they have learned throughout the previous labs(Uart, DMAs, interrupts, SAR ADCetc. etc.) in order to create a 2 channel oscilloscope. The project will take in user input from a terminal(using a UART) and service those inputs. While handling terminal input the microcontroller will be using 4 channels from the SAR ADC in order to read in the voltage from 2 potentiometers(for scrolling) as well as 2 dmas(for reading in the actual input channels for displaying the waves). This will teach the students the importance of creating clean and simple software in order to be able to handle several components simultaneously as the 2 dmas will be transferring data into ping pong buffers which will generate constant interrupts therefore the software needs to function as quickly as possible in order to process the data prior to the next interrupt in order to prevent any data loss. The student will also be using several external hardware components such as potentiometers for x scrolling the two waveforms as well as the TFT display which is a 320x240 pixel display. The student will also need to implement scaling(both x and y scaling) for the displays(affects sampling as well as height of waves via scaling them in order for them to fit the screen). The student will also need to implement triggering which essentially is used to have the wave display starting at a certain point rather than in free mode. The oscilloscope will have a start and stop function for starting displaying/processing the data as well as a free mode and trigger mode for the starting points for displaying the channels.

Description of features:

The final project oscilloscope consists of several features such as displaying two separate waveforms, this is done via using two separate dmas as well as interrupts. It also consists of a command parset which the user can use to configure/change settings on the oscilloscope such as changing the x and y scaling as well as starting or stopping the wave(stops updating the screen and reading in data). The terminal also has two wave modes, a free run mode and a trigger mode. There are also several smaller additional features on the display itself such as displaying the frequency of each wave as well as displaying the x and y scale. Another feature is the wave scrolling feature which is done by rotating the potentiometers which will scroll the waveforms up and down.

Section A) Schematics:

In terms of internal hardware schematics the board uses a SAR ADC with 4 input channels(scan rate of 250,000sps): the first and third channels are used for reading in the signal, these are outputted to 2 separate dmas(one for each channel) which are connected to two separate interrupts. The additional 2 channels are used for the potentiometer readings which will be used for scrolling the signals. There is also a graphings lcd component used for handling the TFT display as well as a UART which will be used for the terminal input therefore it is configured with both a tx and rx fifo. The uart will use pins p5[1] and p5[0] as these are the pins allocated for usb interface with the microcontroller. The SAR ADC will have 4 pins coming into it: p10[6] and p10[4] for reading in data for the 2 waves as well as p10[3] and p10[2] for the input readings from the potentiometers. Below are the external schematics, block designs and internal schematics for the project:

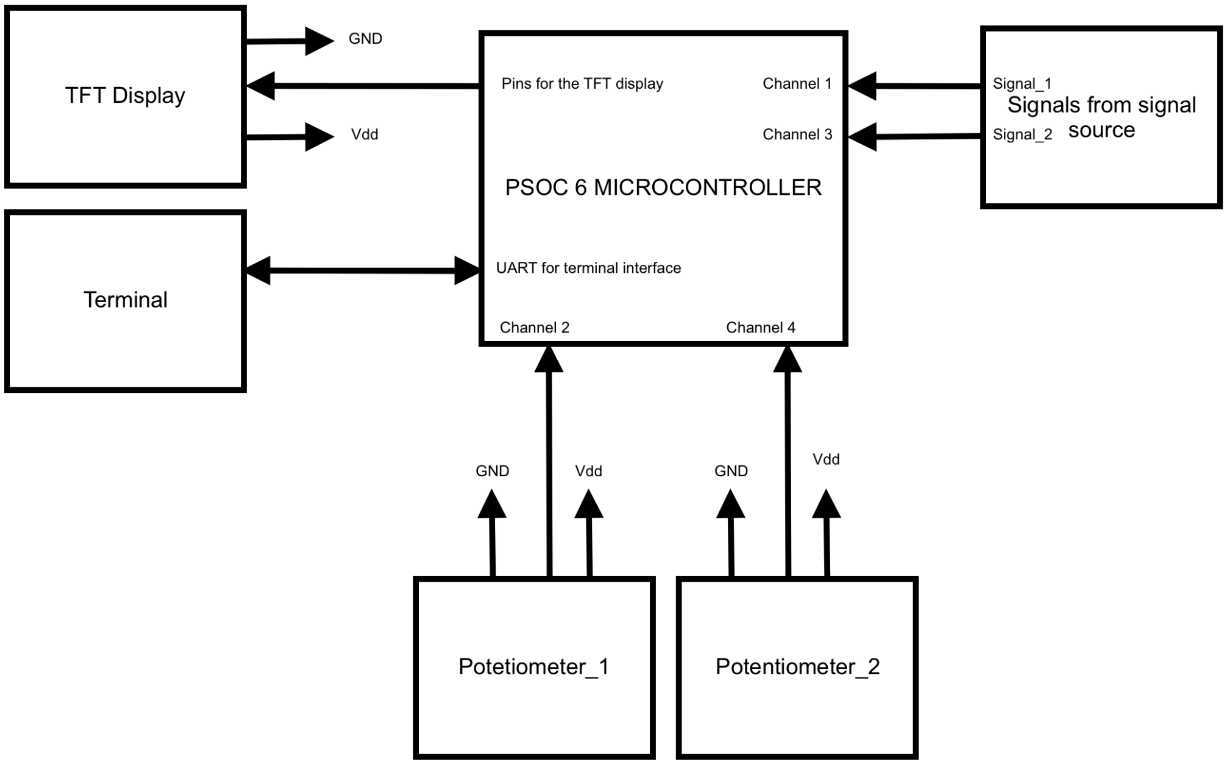


*Figure 1 above shows the block diagram for the project, note that the TFT display has multiple(20-30+) wires coming from it.

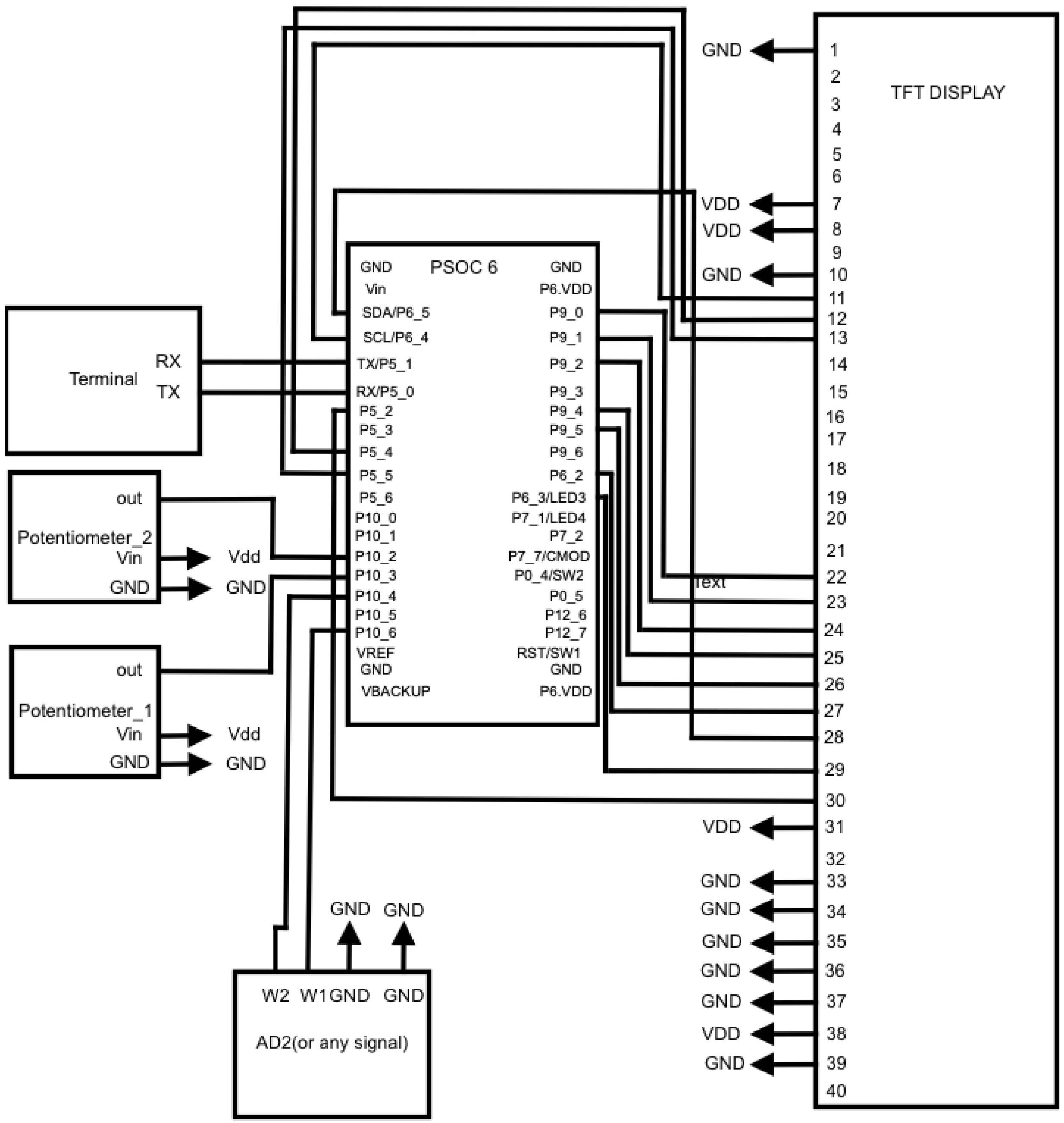


*Figure 2 above shows the external schematic for the final project, as you can see there are many wires coming from the TFT display so a recommended strategy is to use longer wires for connecting the TFT to the microcontroller. Another side note is that the W2 does not need to be attached to ground, but it is highly recommended to prevent any noise on the waveforms.

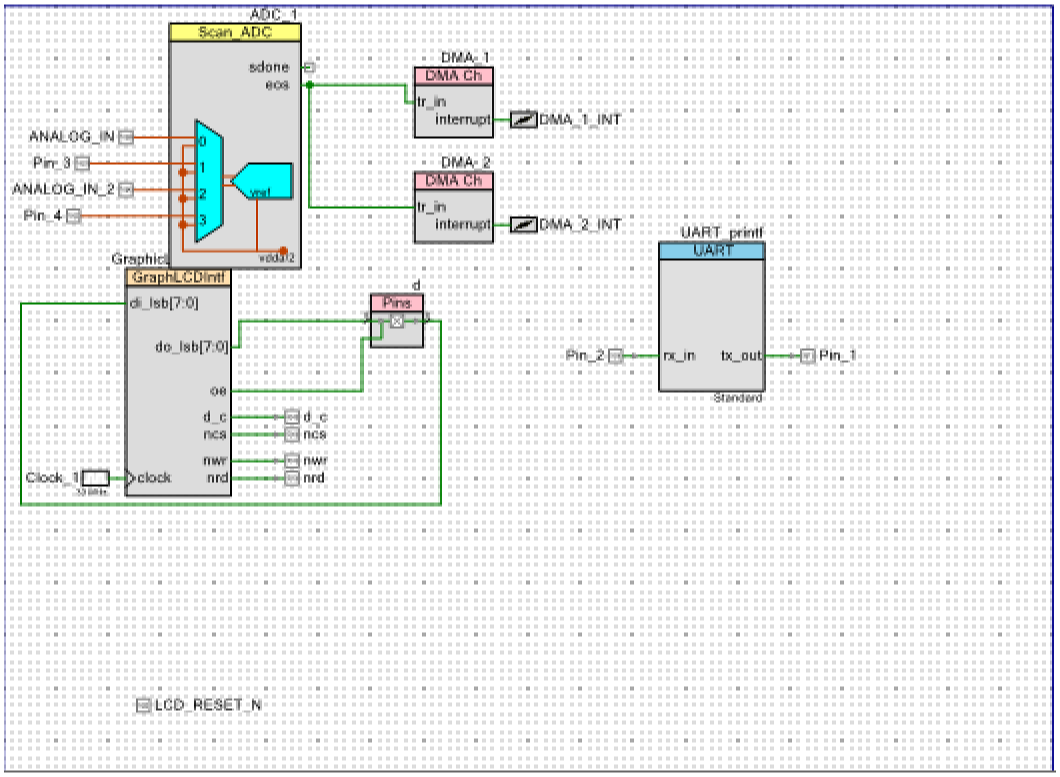
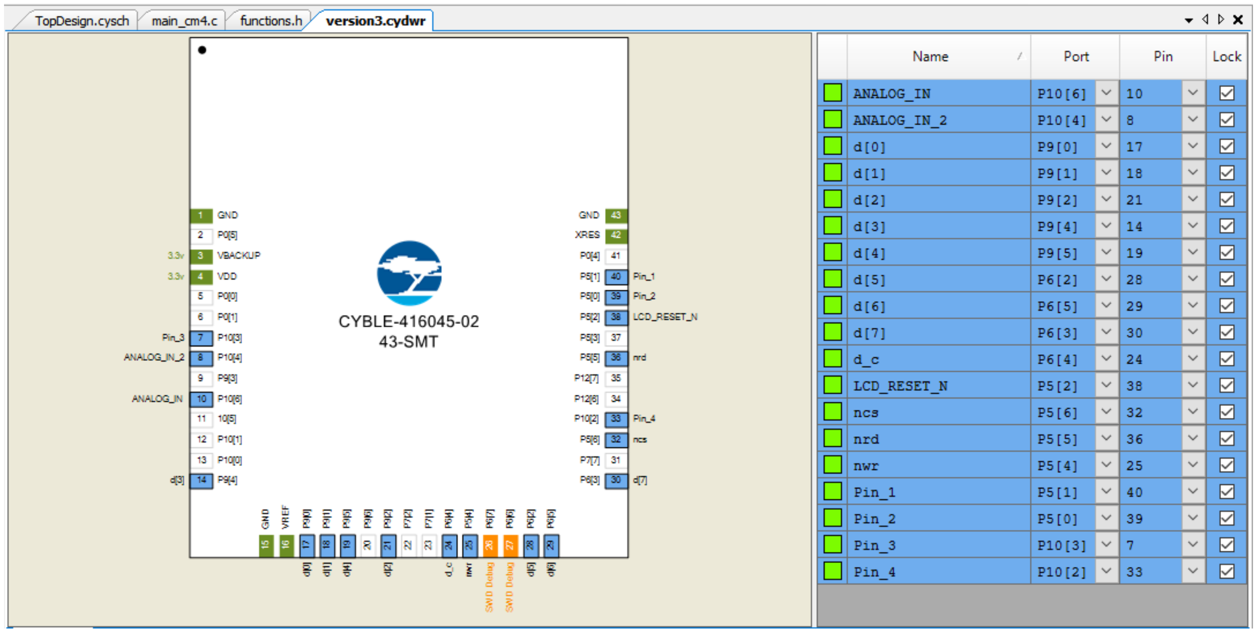


*Figure 3 above shows the internal hardware schematic of the psoc microcontroller.



	Name	Port	Pin	Lock
🟩	ANALOG_IN	P10[6]	10	✓
🟩	ANALOG_IN_2	P10[4]	8	✓
🟩	d[0]	P9[0]	17	✓
🟩	d[1]	P9[1]	18	✓
🟩	d[2]	P9[2]	21	✓
🟩	d[3]	P9[4]	14	✓
🟩	d[4]	P9[5]	19	✓
🟩	d[5]	P6[2]	28	✓
🟩	d[6]	P6[5]	29	✓
🟩	d[7]	P6[3]	30	✓
🟩	d_c	P6[4]	24	✓
🟩	LCD_RESET_N	P5[2]	38	✓
🟩	ncs	P5[6]	32	✓
🟩	nrd	P5[5]	36	✓
🟩	nwr	P5[4]	25	✓
🟩	Pin_1	P5[1]	40	✓
🟩	Pin_2	P5[0]	39	✓
🟩	Pin_3	P10[3]	7	✓
🟩	Pin_4	P10[2]	33	✓

*Figure 4 above shows the pin layout of the microcontroller.

Section B) Design overview:

Hardware Design:

There were several crucial decisions which needed to be accounted for when designing the hardware connections, one major one being the TFT display as those pins were predetermined therefore it could not be changed or modified therefore I needed to work around it making sure that my other external components had easily accessible connection points. Fortunately the corner by the p10[] area had multiple unused pins which were enough to satisfy my hardware requirements. Another constraint was the number of wires which were provided as unfortunately the wires are not of the highest quality therefore many turned out to be faulty, the issue being that there were a limited number of longer wires therefore designing the schematic prior to connecting the pins was crucial in order to prevent any damages or having to start over due to limited number of connectors. In terms of internal hardware the design consisted of one 4 channel ADC which had a scan rate of 250,000 Scans per second for each channel. Two of the channels were simple readers for reading in the value from the potentiometer, the remaining 2 channels were used for constantly reading in signals, these readings would be sent to two dmas which are attached to interrupts which will trigger once the dmas have completed filling up the buffers. There is also a graphics controller used to communicate with the TFT display as well as a uart which is used for communication with the terminal, this uart is configured in both tx and rx mode meaning that it will both read and send data to. The DMAs will convert the data from word to halfword and each will have 2 channels(one for each of their ping pong buffers). The DMAs will trigger interrupts on completion of each channel and the channels are chained to each other. In terms of the UARTs baud rate it simply needs to match the baud rate of the terminal.

Software Design:

There were several main constraints when designing the software aspect of the project, one important one being timing as there is oh so much time before the dmas have overwritten the data inside of the ping pong buffers meaning that the software needs to reach quickly in order to prevent any data loss. Another constraint is the stop feature as one start/stop is called. We need to completely stop processing data and display the previous waveform constantly. Another constraint was x scaling as in order to have enough x scaled values the program would need to service roughly 40,000 bytes of data which was not possible in one large buffer as well as triggering prior to y scaling. Personally I broke this down into two separate tasks: the dma data readings and the display aspect. For the DMA readings I had 2 ISRs(one for each channel) each ISR would essentially be transferring data from 2 ping pong buffers(size 1024) into one larger buffer(size 4096) which is only going to hold the x scaled values. The ISR would first calculate the pixels per sample point(Ps), this is essentially the number of pixels per division divided by the x scale times 4. This results in 3 possible outcomes Ps Ps being > 1 meaning that the sampling rate is too slow for the screen therefore we need to sample every Ps element(for example every 5th element). The next outcome is Ps = 1 meaning that the sampling rate is fine and we can transfer every individual element. The final outcome is Ps <1 meaning that the sampling rate is too fast and we need to grab every 1/Ps element. The Isr will start off by checking if the large buffer is being processed elsewhere as if it is we do not want to overwrite the data just yet, if it is not the ISR will then determine which ping pong buffer is full and start transferring the data into the large buffer. One important note is that there needs to be an index of the large buffer and an index for the ping pong buffers so that we do not pass the range of the buffers. In the scenarios where Ps does not equal 1 you will need to calculate the next index for the next buffer as it cannot simply start off at 0 again it must continue where the last buffer left off. Below is a visual block diagram of one of the ISRs showing an overview of the logic:

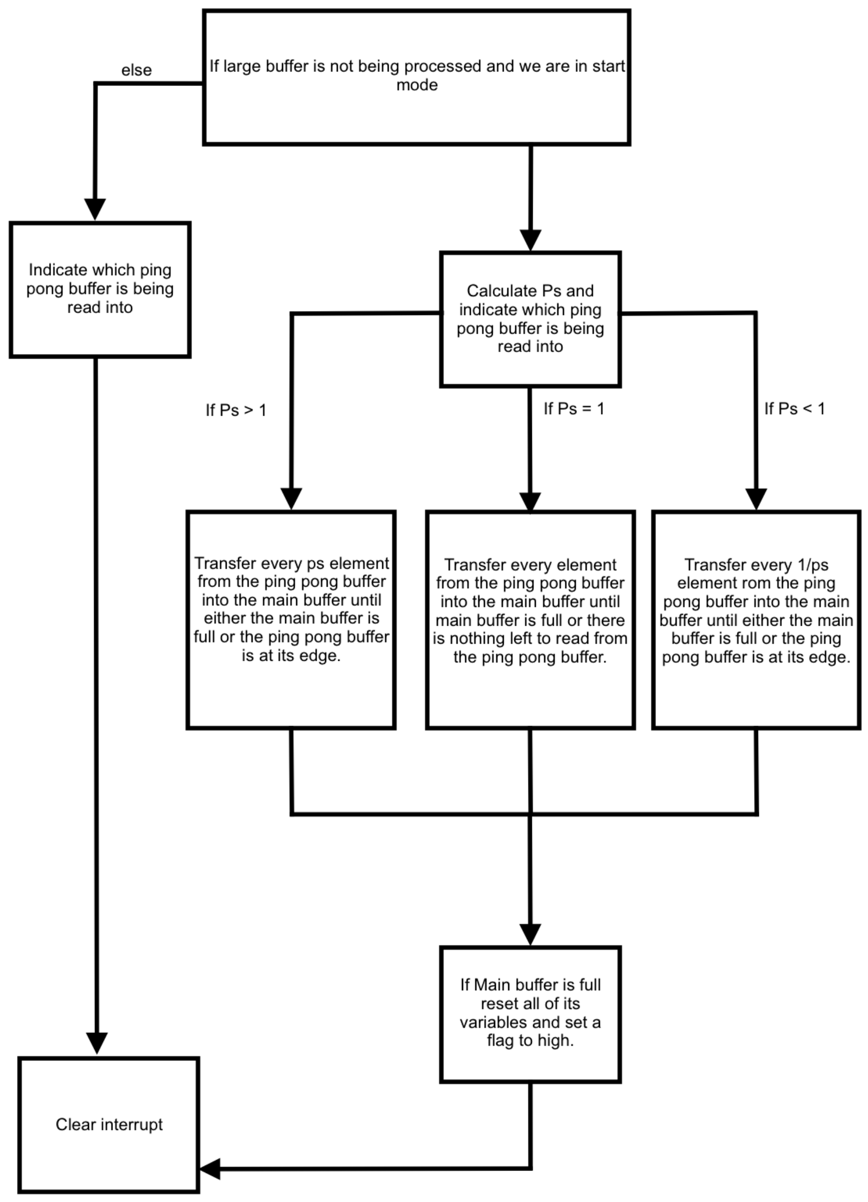

If large buffer is not being processed and we are in start mode
else
Indicate which ping pong buffer is being read into
Calculate Ps and indicate which ping pong buffer is being read into
If Ps > 1
If Ps = 1
If Ps < 1
Transfer every ps element from the ping pong buffer into the main buffer until either the main buffer is full or the ping pong buffer is at its edge.
Transfer every element from the ping pong buffer into the main buffer until main buffer is full or there is nothing left to read from the ping pong buffer.
Transfer every 1/ps element rom the ping pong buffer into the main buffer until either the main buffer is full or the ping pong buffer is at its edge.
If Main buffer is full reset all of its variables and set a flag to high.
Clear interrupt

*Figure 5 above shows the design for the ISRs, these ISRs are used to service the DMAs.

The main function will simply start and initialize the different hardware components(UART, DMAs, GUI) and draw the background of the TFT screen. From there it will enter the main loop which will do the following: first the loop will check which mode the oscilloscope display is in, if it is in start mode the oscilloscope will then check if both buffers are full. If both buffers are full then we will then get the potentiometer readings for both channels and convert them to fit into the screens limits. Since the adc reading for the potentiometer can vary from 0 to 0xfff we will need to scale that to fit into the 240 size of the TFT screen via dividing 0xfff by 26 and adding 5(to stay inside of grid) this allows for both channels to be able to move freely across the screen while keeping the waveform still inside of the screen(for a standard y scale of 1000v). From here we will call a function which will handle y scaling. This function will also handle the triggering(if we are in trigger mode). It is essentially one for loop which will first check ig we are triggering if so we will convert the index value to mv and compare it to the trigger level in order to determine if this is the trigger index(if positive slope looking for index where array[index] < trigger level and array[index + 1] > trigger level, will do vice versa for negative slope). After doing the trigger check we will convert the value to the y scale value by simply multiplying the value by py(number of pixels per y division) / sy(y scale) * 3300(mv range) * array[index] / 4096) we can do all of the logic for both channels in the one array, note I used global variables for indicating the mode as well as the trigger slope and the trigger level. From here the main loop will then call find frequency which is another helper function which will calculate the frequency. In order to do this it will find the middle voltage via finding the highest and lowest voltage and taking the difference and dividing by 2, after finding the middle voltage it will find the points in the array where the waves have crossed the middle function(in a positive slope). Once I find the first crossing point the second loop continues from that point until it locates the second one, from there I take the difference in order to get the period. To attain the frequency I do frequency_2 = (1000000 / (period * (x_scale_1/32)))/10, as in order to find the frequency it is 1/period for MHZ, then I need to account for the x scaling as this data has already been x scaled. After finding the frequency my main loop will then update the frequency and display the scaling, in order to do this I placed the values in front of a rectangle allowing for me to update the values without writing on top of the old data(via filling in the rectangle then writing the new value). After this the drawing of the waves come in, in order to draw a wave we must draw 320 elements from the now x and y scaled draw buffer, but before we can do that we must erase the old waves via drawing over them in the background color and redrawing the grid(I also had several small buffers for recording the previous values which needed to be erased). The method for drawing is the draw function which takes the inputs x1, y1 x2, y2. Fortunately the x values are always constant(from one edge of the grid to the other), but the y scaled values vary therefore I would use buffer[i + trigger_index] + potentiometer value for the y values. This satisfies the triggering(if any, 0 if no triggering) and adds the potentiometer reading which was needed for scrolling. Once I have drawn the line for one channel I need to change colors to draw the next line and constantly alternate the color while drawing the lines(as well as recording the values which are being drawn for erasing and drawing in the stopped state). One important detail is the array style since we are plotting y and y + 1 you will need to also add the final index(309) outside of the drawing loop(or else you will potentially have an overflow error) from here I reset the dma flags allowing for them to transfer more data into the large buffers. The main loop also had an else statement for if we were in the start or stopped state, if we were in the stopped state the main loop would simply record the potentiometer values and display the previous waves constantly, this allows for scrolling on the same waveform. After these statements we reach the end of the loop where we have a 1000 iteration loop which will do a 1us delay and check for any terminal inputs. In order to check for terminal input our helper function will check if the rx fifo is full, if so it will read in one byte at a time and store it into a buffer until reaching an end of line character/new line character. Once it has reached this it will call a new function which will handle understanding what the command is if it is a start or stop, or an x scaling change, or a y scaling change, or changing trigger settings. One important note is that this will also handle conditions such as you are not allowed to change trigger settings while in the start state as well as checking that the trigger setting is valid. Another note on the trigger level is the values as we are drawing on an display where the first pixel(0,0) is on the top left corner meaning that all of our trigger results will be flipped(0 being looked at as 3000 and 3000 being looked at as zero) therefore we can simply flip all of the possible conditions by doing 3000 - wanted trigger lever. Note: default trigger level is 1000mv which would be calculated as 3000 - 1000 or 2000mv. Below is the large scale schematic of the design:

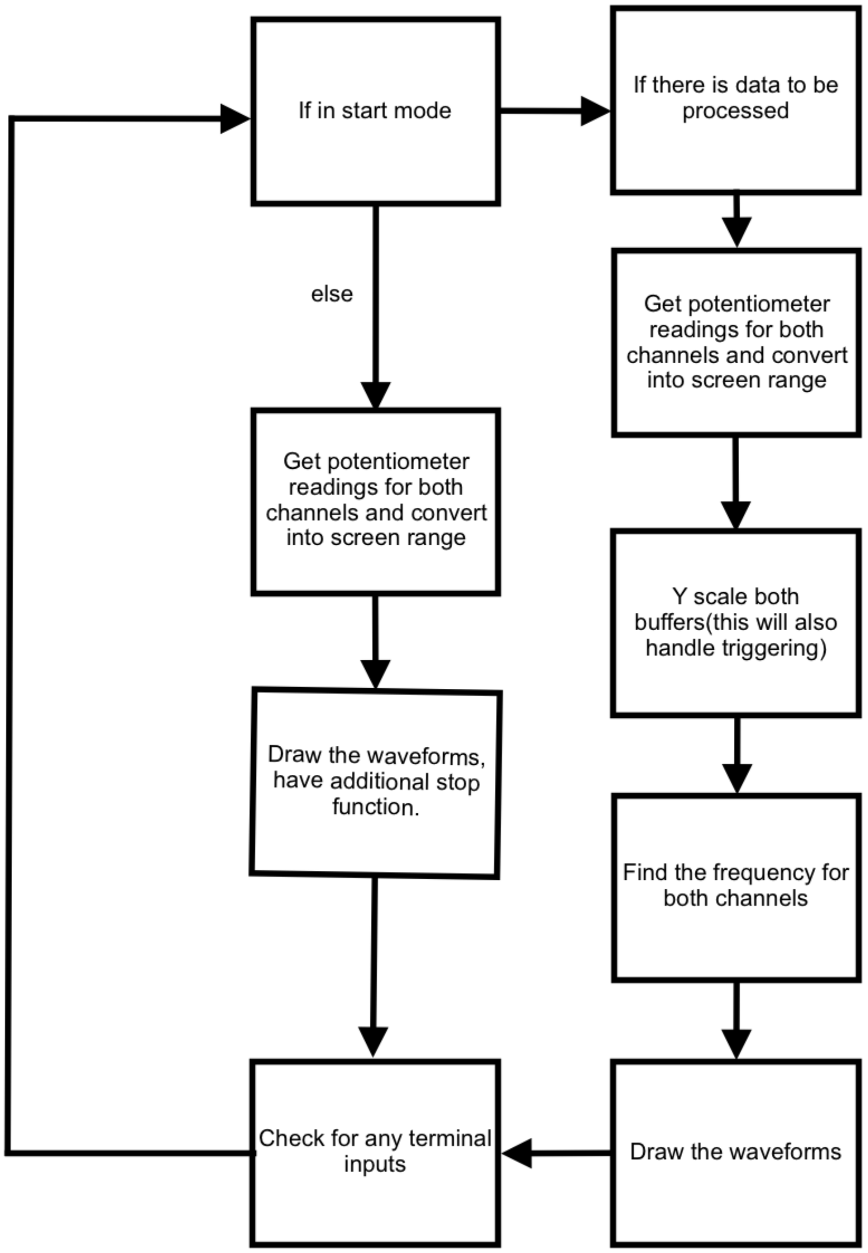

If in start mode
If there is data to be processed
else
Get potentiometer readings for both channels and convert into screen range
Get potentiometer readings for both channels and convert into screen range
Y scale both buffers(this will also handle triggering)
Draw the waveforms, have additional stop function.
Find the frequency for both channels
Check for any terminal inputs
Draw the waveforms

*Figure 6 above shows the flow of the main loop inside of the program.
In terms of buffers there were multiple buffers used in this design: to start there were the 4 ping pong buffers(2 for each dma) which were size 1024. These buffers were used to

fill the 2 large buffers(size 4096) with x scaled values. From here there were 4 smaller buffers of size 310 for deleting the old plot and storing the repeated plot in case we entered the stop state as well as a 50 byte buffer for handling the terminal reading. I personally selected a large buffer of size 4096 in order to prevent any issues with calculating the frequency as the frequency would nearly always be guaranteed to be inside of these buffers due to the size(we would at least have 2 passes of the middle in the positive direction). In terms of communication to prevent any data corruption/loss I have several global flags indicating to the main function that the large buffers are full, once those are full the ISR sets a flag indicating that the data is ready to be processed and it will not be allowed to add new data from the dmas until this data is done being processed by the main loop. While implementing this design I ran across a multitude of errors: to start I had originally done by x scaling in a separate function from the isr, this became an issue as if the x scaling was a very large value(10000) then my large buffer would not have enough values to display the entire waveform. A solution to that was simply doing all of the x scaling inside of the ISR. Another issue was the consistency of the waveforms this was caused due to the fact that the ping pong buffers would alternate while they were running, but once the ISR stopped running and transferring data I would stop alternating the ping pong buffers resulting in some skewed waveforms, fortunately this was another easy fix via just making sure to always alternate the ping pong buffer in any scenario. Another error I had experienced was using the provided draw function. This function would take an array and draw the expected function, unfortunately this function was hanging and causing a lot of errors therefore I ended up creating a helper function which would simply draw the lines from point to point. In terms of hardware one issue I experienced was due to the lower quality of the wires provided which resulted in me frying and damaging one of my psoc microcontrollers, a solution to this was to measure the voltage coming from each wire in order to make sure that none of the wires were faulty or causing any issues. Fortunately I had a second microcontroller and this wasn't such a large setback. Another issue was using an isr to handle terminal inputs, I noticed that the usage of an interrupt for that would have potentially added risk to missing interrupts for the two dmas therefore I simply added the servicing to the main loop in order to prevent any hindrance to the two dma data transfers which have the highest priority in terms of execution.

Section C) Testing:

In terms of testing I started off small by simply wiring up the TFT display and making sure that it was connected properly, from there I implemented the first dma and attempted to x and y scale and display one waveform, upon successful displaying I redesigned my x scaling and retested it. From here I created a new project and completely implemented the terminal interface and then added that to my main design. After that I tested it and added triggering and I implemented the start and stop function. My final addition to the first channel was the potentiometer as well as displaying the frequency and scaling settings. From here I added my second channel and essentially added all of its implementation to the first channel's logic(I could do all of the y scaling and drawing inside of the same loop).

Section D) Conclusion:

In terms of results I was able to mostly display the frequencies properly, as well as display both channels properly(unfortunately my ramp up wave was being displayed in reverse, but this would have been a relatively simple fix had I noticed it sooner via simply mirroring it). Another issue I had not noticed until it was too late was the fact that my x scaling is slightly off in some of the higher frequencies, I believe that this has something to do with my isr logic(I used a ceiling function which was not necessary). Some additional enhancements could have been adding a zoom function(one easier then playing with the scaling) maybe via using additional potentiometers. Another additional enhancement could have been adding a crosshair which could have been moved around the board to get an idea of the voltages, which is something used frequently on normal oscilloscopes, this would have required additional potentiometers and some optimization to the software to support an additional item being drawn.

Conclusion/Reflection:

Ultimately this lab was a good way to introduce the student to a real world application of the different components which we have learned throughout the quarter. It also taught the student how to design a project while considering future components and attempting to satisfy any constraints other hardware blocks may provide when implementing them into the main design. I personally believe that the aspect of giving the students minimal guidance was also a good idea as it allowed for them to have full creativity in implementing their designs and gave them an opportunity to learn from their mistakes as by not having a sound design before starting implementation many students would end up having to reimplement and redesign finished components in order to satisfy other components needs. The hardware aspect of this project was relatively simple to implement, but became another issue due to the quality of the materials provided as it became very difficult to test each wire in order to confirm that voltage was being passed through it.