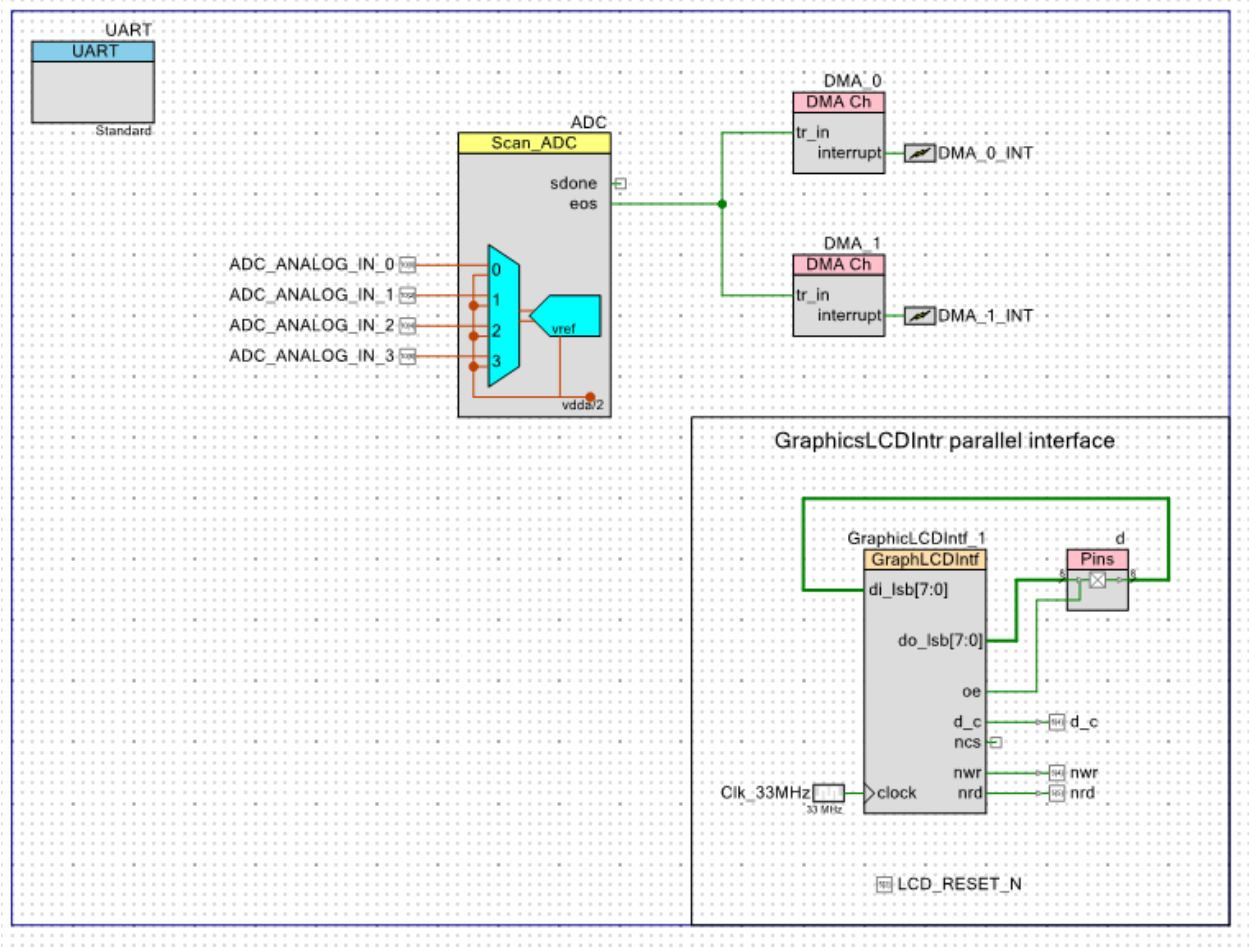
Lillian C. Gwendolyn
CSE121 Final Project Report

Schematics:

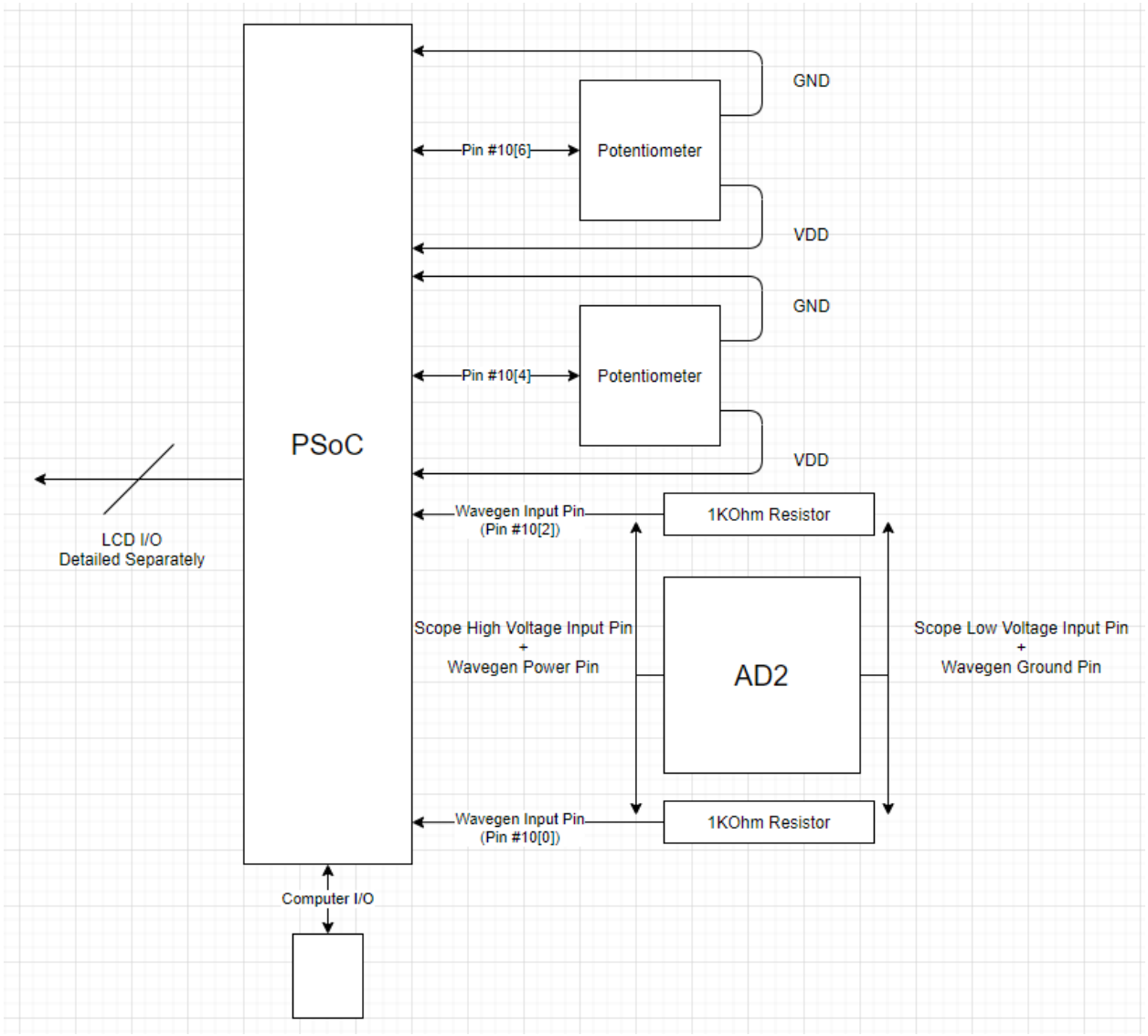
Top level design:



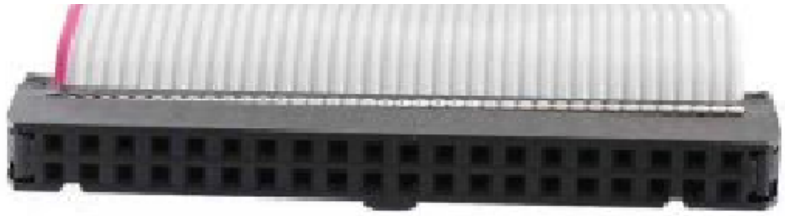
Pinout:

	Name	Port	Pin	Lock
	\UART:rx\	P5[0]	39	☑
	\UART:tx\	P5[1]	40	☑
	ADC_ANALOG_IN_0	P10[0]	13	☑
	ADC_ANALOG_IN_1	P10[2]	33	☑
	ADC_ANALOG_IN_2	P10[4]	8	☑
	ADC_ANALOG_IN_3	P10[6]	10	☑
	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	☑
	nrd	P5[5]	36	☑
	nwr	P5[4]	25	☑

External circuit schematic:



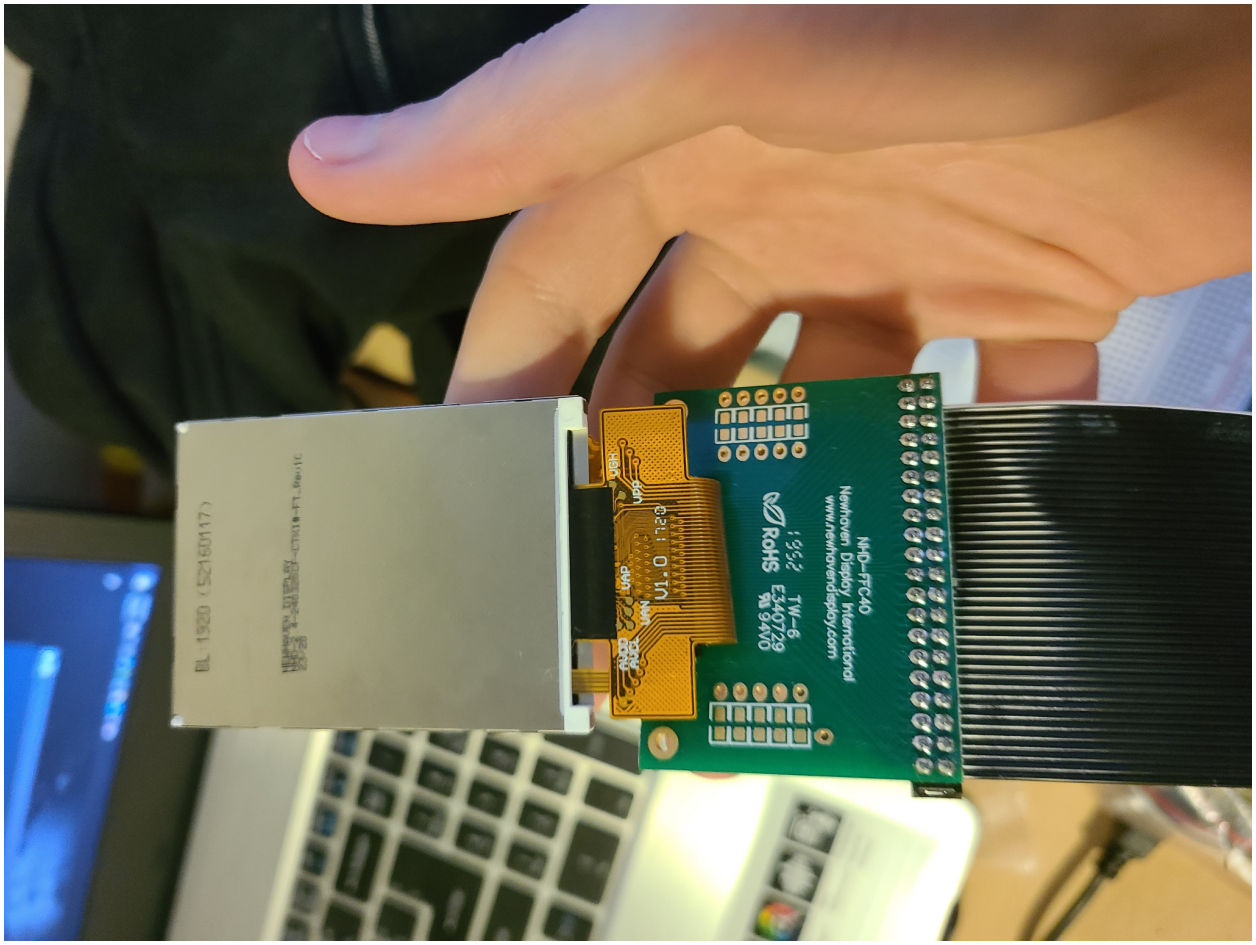
LCD I/O:



from left to right, top to bottom
note - the ribbon seems to be reversed on this image, the off color wire should be on the right side of the pin header

left side being left side, bottom side being side with notches, top side being side with cables
X means not used, P means power (3.3v, plugged into P6.VDD), G means ground (plugged into GND), and otherwise the pins are labeled by which port on the PSoC board they fit into:
top row: X | P | G | G | X | 5[2] | 6[5] | 9[5] | 9[2] | 9[0] | X | X | X | X | 5[4] | G | P | X | X | X
bot row: G | G | G | G | P | 6[3] | 6[2] | 9[4] | 9[1] | X | X | X | X | 5[5] | 6[4] | X | P | X | X | G

with the header plugged into the LCD as such:



Design Overview:

Circuit wise, there was nothing quite special going on outside of the two schematics detailed. It was certainly a lot of wiring but none of it was exceptionally complex.

Hardware wise, there is a little more going on. The UART unit and graphics unit are both pretty set and forget, but the ADC and DMAs have a bit more going on with them. The ADC is set to complete a full scan 250,000 times a second for all 4 outputs - 2 of those outputs are linked to the potentiometers, and the other 2 are linked to the AD2 wave generator channels. From there, the eos output of the ADC links to the trigger inputs of the two DMAs, which then (in code) take their next value from the ADC and wait until retriggered. The DMAs take values from the wave gen ADC output and have a total descriptor capacity of 1024, each with two descriptors to ping-pong back and forth so that one descriptor is always writing to the buffer at a time.

In general this is exactly what was stated by the project designs, except the DMA buffers were expanded to reduce issues with large xscale factors.

Software wise, this is where we get into the important things for this project -

This project had a lot of special features that summed up to essentially emulate an oscilloscope.

You could send a wave to one of the wave input pins and it would appear on the display in real time, even changing the frequency or voltage or shape of the wave would still be mirrored on the lcd display.

The oscilloscope would automatically determine the frequency of the incoming wave and output that to the screen, as well as the current xscale and yscale settings.

You could scroll the wave up and down with the potentiometers.

You could change the xscale and yscale and the waveform would adjust in real time on the display.

You could switch from a free-running mode to a trigger based mode, complete with its own level detection (and a programmable level) at a given slope (also programmable between positive and negative slope) and it would display the wave from any point matching that specific setup, essentially stabilizing the wave on the display.

These settings could be entered through a terminal connected via UART serial connection through a usb connected to both devices.

This UART could also let you stop and start the display at any point, and would handle erroneous input (random strings), out of bounds input (numbers too high or low or uneven for xscale etc.), and input at the incorrect times (changing modes while running). Additionally even

if a lot of those conditions were removed the display would still function fine, they are only limited because they have been asked for in the project guidelines.

And lastly, you could do all of this across two channels at once, with both displayed on the same screen and updating simultaneously.

Now, the actual software had a lot of behind the scenes for each of these features. There is a large number of defines and global variables involved in this, as everything was kept inside the main file. Anyways, going down the list:

The first function at the top handles the UART I/O - this function is run every cycle in the main loop if there is anything for the UART to receive, and if there is then this function takes in all the new input and adds it to a buffer - when the buffer receives a newline character, the function begins to match the input string held in the buffer with all available ‘instructions’ that have been defined globally - if a match is found it will be processed and an associated message will be sent back, if nothing is found it will default to an error message. Sadly there is no support for switch/case statements using strncmp so this was done with if else statements.

The second and third functions are the DMA ISRs, these are pretty simple and essentially every other descriptor completion the DMA will push all of its new data into the buffer that is actually used for drawing. In general other than that data movement these just clear their interrupt and move on.

After that we have the drawing functions. For the most part these were taken from the provided code in the demo and modified to both be a bit more readable and to have less input from the user. ShowStartupScreen shows a screen with the project title and my name on it, drawBackground draws a cyan background, drawGrid draws a grey grid over the background, printScaleSettings and printFrequency draw their respective information in their respective corners of the screen, and plotWave draws a line between all of the provided wave points given to it by other functions.

Next up we have scaleDrawValues - this uses the current xscale and yscale settings to find the y values of every x point that is able to be drawn on the screen, and then pushes this information into the array that is used to draw the specific channel by plotWave. The math used here amounts to the following: scale the data values from the ADC into digital values that map onto the screen, and scale that by the current yscale. The values used from the ADC depends on where the trigger (if any) is determined to start from and how many samples are taken per pixel as decided by the xscale - if this total number gets too large then it loops back around as determined by a modulo of the maximum size of the data buffer.

After this there is a commented out data smoothing function that I implemented according to the professor’s provided algorithm but it made my frequency and trigger finding notably less consistent, surprisingly. It has been left as is but otherwise commented out to show that yes I did implement this.

Next are the functions for finding the frequency and the start of the trigger for each wave - these are rather similar, the frequency function loops through each wave until it has found two passes across the midpoint and counts the number of samples between to be used for a frequency calculation, while the trigger function waits until it finds the first pass in the chosen trigger slope/direction across the chosen trigger level/voltage, and then saves that value to be used by the drawing functions as a starting point for the displayed samples.

Lastly we have the main function - Outside of the for loop is a large number of initializations for the DMAs, the ADC, the UART, the UART buffers, and the LCD. Notably, before the main loop the last things we do are show the startup screen and then turn on the ADC - once the ADC is on the DMAs and everything else begins doing their jobs so initialization would have slowed down if this was done earlier.

The main loop of the main function goes in the following order:

First a delay at the beginning so that this loop repeats ~5x a second.
Second, if there is anything to receive in the UART, go to the UART handling function and deal with that.
Third, if the oscilloscope is in the ‘stopped’ mode then loop back to the beginning of the loop, otherwise proceed to draw things.
Fourth, if it is the first loop through after the display has been restarted then draw the background.
Fifth, save old potentiometer values and take new ones from the respective part of the ADC.
Sixth, save the old draw buffers and use them to draw the background color over the previous waves - this is faster than drawing the background over the whole screen completely and so does not produce a ‘flashing’ effect.
Seventh, find the trigger start values for each channel (if any) and then proceed to turn the values in the data buffers into drawable values in a separate drawing buffer.
Eighth, plot the waves, the grid, the frequency, and the scale.

Testing:

Testing this project was a pretty large process. I ran into a lot of trouble in the first week just trying to get my LCD connected to run the demo we were given, a lot of trial and error involved because we were not provided with a diagram of the full connected setup.

From there once it was working and I could actually begin to implement and test my design on the board, it was another round of trial and error with figuring out why my DMAs and UART could not work - the UART issue was solved first with this helpful video another student sent me: https://www.youtube.com/watch?v=M-DjaxhYr70 - the video is also mentioned in my code where its relevant information was put to use. The DMA issue was solved by a large amount of bug checking and stepping through my program before tracking down the exact error code I was getting, which was fixed by changing my DMA descriptors to go from word to halfword to halfword to halfword.

After that, testing was pretty smooth - I would implement something or make some changes, test it out on the board to make sure everything worked, and went back to programming. I started originally with a lot of code copied from my previous labs and the demo oscilloscope code we were given, modified it all as I went, and added new functions whenever I needed something done that seemed best on its own. For the most part, the actual drawing components were done towards the end of the project, with setting up my trigger being the last part I did - I did this at the same time as the smoothing function using the professor's algorithm, which upon testing actually made things worse, so I stuck to the unsmoothed version and it 'fixed' my frequency and triggering issues.

Conclusion:

All in all this was a pretty good project, I just wish that I had managed to fix my connection issues and solve my DMA and UART issues earlier than I had - if those problems had not come up this would have been much smoother and much less of a grind than it was this time. That being said it was pretty cool to see everything come together and actually work

As far as enhancing my design, I think a better smoothing algorithm could work pretty well, and perhaps a better method for drawing the waves that uses less memory and memory transfers. Similarly I noticed while typing this up that I could redo the way I drew the previous wave and essentially save myself ~600 data transfers every loop of main by just using the current wave as the previous wave instead of transferring into a buffer that is not used anywhere else in the code.

Additionally, if I had the time and energy I could have split the drawing functions into a separate file for readability, this would have required minor modifications to the code however.

Lastly, if you plan on using this LCD again for another assignment for a different course, please provide a diagram of the header on the LCD directly showing which PSoC pin it each port plugs into, as well as detailing the correct orientation of the LCD on its chip and its chip to the ribbon cable. This would have saved me a week of headache and would be much nicer for future reference instead of staring at a not-fully-implemented design clipped on Yuja and comparing it to another student's numbering of the ribbon cable header.

References:

My two references here are the instructor's provided code which was used to base off a lot of my drawing functions, as well as the provided hardware component(s).
Secondly I would like to thank this video: https://www.youtube.com/watch?v=M-DjaxhYr70
for helping me implement my UART I/O.

Other than that this code is my own and I would like to thank the TAs and tutors for working their arses off this quarter and my classmates for pointing out helpful tips.