CSE 121 - Spring 2021
Lab Project - Tinyscope

Donald Shannon
dishanno@ucsc.edu
1527177

Introduction:

Tinyscope is a dual-channel oscilloscope with the ability to change parameters with a console, and graphically depict waves on a display. Waveforms are inputted via two wires attached to the device, the console is accessed via a usb UART protocol, and the display controlled via onboard potentiometer knobs.

Contents:

1. Block Diagram

2. External Device Schematic

3. Internal Microcontroller Schematic

4. Description of Features

5. Command Documentation

6. Hardware Design

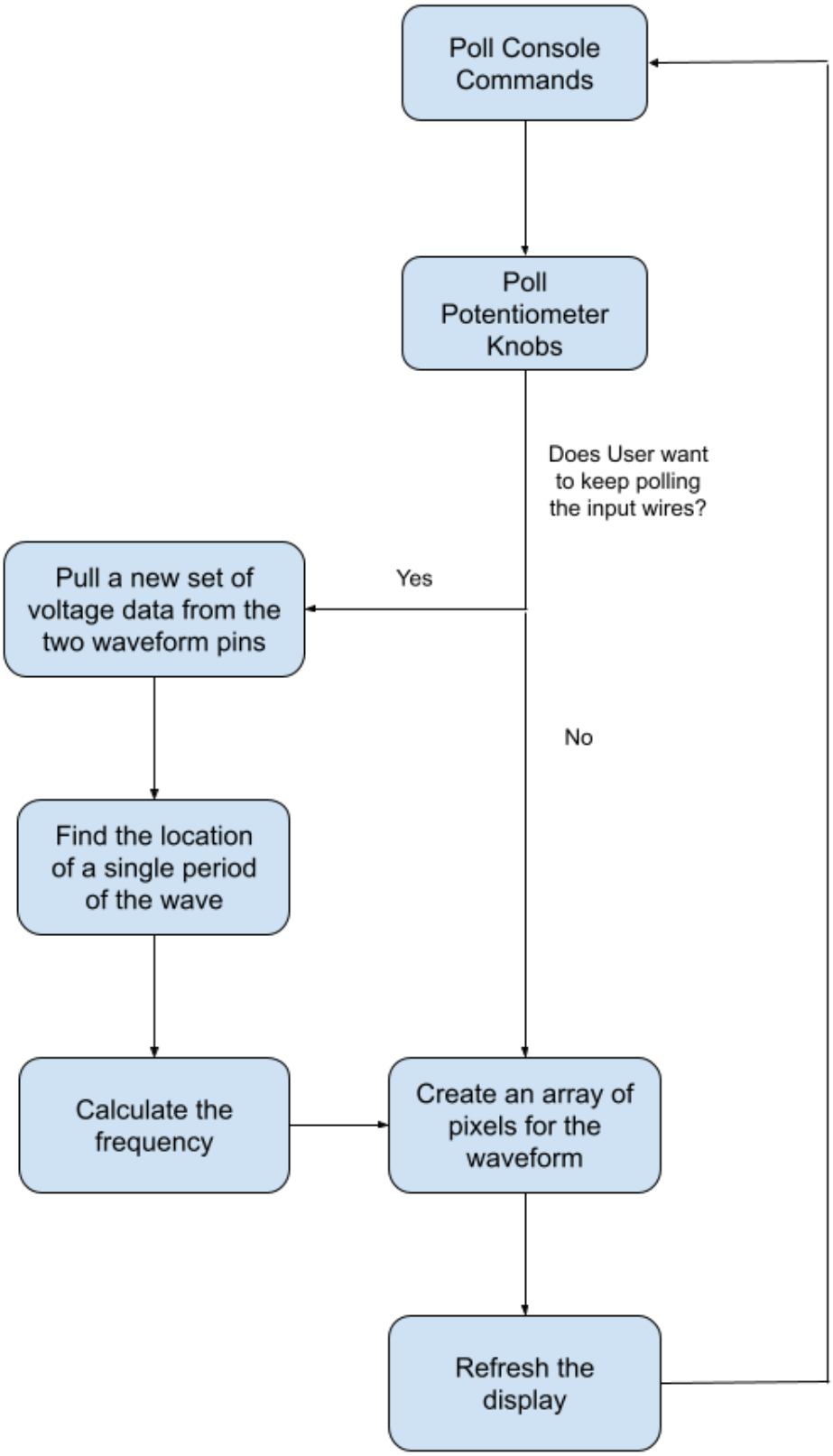
7. Software Design

8. Testing Methodology

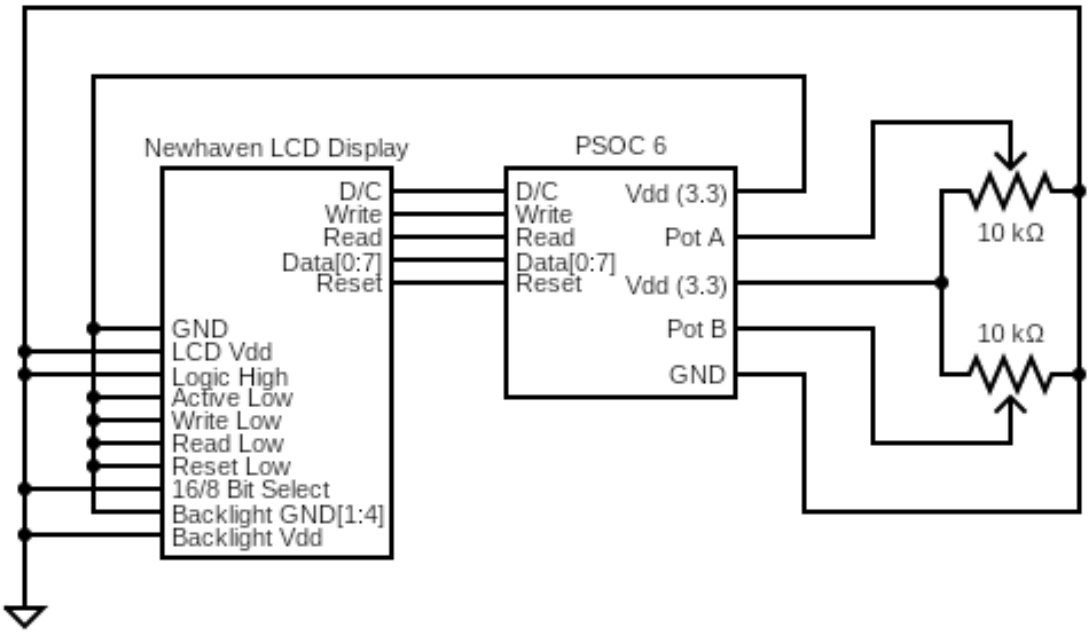
9. Future Additions and Conclusion

10. References

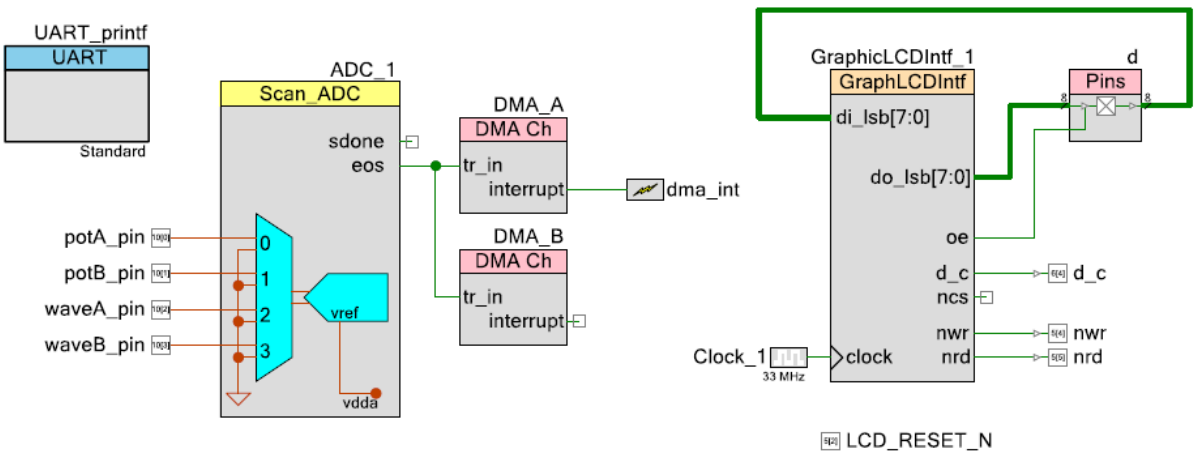
Block Diagram:



External Device Schematics:



Internal Microcontroller Schematic:



Description of Features:

Tinyscope is a dual-channel oscilloscope with the ability to change parameters with a console, and graphically depict waves on a display.

The input for each channel is a blue and orange wire for channel A and B respectively. If any channel is unused, it is recommended to ground that wire to mitigate any noise that could arise. The input voltage for the waveforms can only be from 0 to 3.3 V. Any amount over or under will be read as 0 V. Any type of wave can be detected (i.e. sine, square, triangle, etc.).

The two potentiometers are used to scroll each waveform up and down on the same screen, allowing the user to overlap them for comparison.

The waveforms can be displayed as a free-running signal, or can be aligned with a trigger. The trigger is a voltage level that when the signal crosses, marks the left side of the waveform that will be displayed. This maintains a consistent view of the waveform that remains persistent as continuous waveform data is read. The user can select the channel that the trigger reads from and whether a positive or negative edge for the voltage is used.

The scale of the display for both the voltage in the y-axis, and the time in the x-axis can be changed dynamically as needed.

The display, in addition to depicting the waveforms, also shows the frequency of each wave, using the specified trigger type as the markings for the start and end of a single period. Supported frequencies are from 100 to 1000 Hz, with greatly decreased accuracy when going under or over. Up to a ±1% error can persist in the supported range depending on the frequency used.

To access the command line interface, any usb UART console can be used. PUTTY is recommended. The baud rate for the interface is 115200. The commands are shown in detail in the next section.

Command Documentation:

start

Starts reading from the input pins and displaying the waveforms.

stop

Freezes the current waveform for stable viewing and stops reading from the input pins.

set mode <mode>

free - displays the incoming waves as they come
trigger - sets the waveform position to start at the specified trigger
Can only be set when in a stopped state.

set trigger_level <level>

Sets the voltage level for the trigger point. Takes an integer in millivolts in increments of 100. Can only be set when in a stopped state.

set trigger_slope <slope type>

positive - sets the trigger point for a positive edge
negative - sets the trigger point for a negative edge
Can only be set when in a stopped state.

set trigger_channel <channel>

Chooses the channel that the trigger point is set to. The two channels can be referred to as ‘a’ and ‘b’ or ‘1’ and ‘2’.

set xscale <scale>

Set the horizontal scale for the waveform in microseconds per division. Supports 100, 200, 500, 1000, 2000, 5000, or 10000.

set yscale <scale>

Set the vertical scale for the waveform in millivolts per division. Supports 500, 1000, 1500, or 2000.

Hardware Design:

The cyble-416045-02 PSOC 6 from cypress was used because integrated hardware blocks were essential to the design of Tinyscope.

A UART hardware block was used at an 115200 BAUD rate to communicate via usb to the command terminal. To access the command line interface, any UART supporting console, such as PUTTY, can be used.

Two channels of the built in ADC block were used to read voltages from the two 10 kΩ potentiometers. Each channel was polled at about 250,000 samples per second, although the software only polled them once every display refresh. These 16-bit voltage values were used for positioning the waveforms on the screen.

The waveform input wires were connected to the other two channels of the built in ADC. Also running at 250,000 samples per second, two DMA components were used to directly insert the data into memory using a ping-pong buffer system, saving valuable cpu time.

A NewHaven 320x240 TFT display is used to depict the waves and other information, although the code supports a variable resolution, given enough memory. The data lines for the display were wired to the pins of the controller, along with the various Vdd and GND pins needed. While some components in the display needed 3.1 V for optimal performance, 3.3 V was used as it was close enough to not cause problems, and easily accessible with the PSOC 6. Only the 8-bit controller of the display was used, so the 16/8 bit pin was set high to change it to 8-bit mode.

Software Design:

The firmware for the device was created in a bare-metal format, using the emWin graphics API, and some methods from Anujan Varma's waveform demo to create graphics for the screen.

The display is refreshed once each chunk of data pulled from the ADC has been processed. The design uses just under 99 kb of memory when fully zoomed out (xscale = 10000, yscale = 2000). At the current 150 mHz CM4 core, a maximum refresh rate of 15 Hz can be achieved.

Each display refresh cycle, the command terminal and potentiometers are polled for any user inputs, and the various parameters are set accordingly.

To take data in from the UART, a flag is read which indicates that a new input string is ready from the UART buffer. This is done by scanning the UART at the beginning of each display refresh cycle and pulling all available characters. Once the 'end' character is read, which may take several calls to the scan function over multiple display refreshes, a flag is set to indicate that a string is ready. This string can then be retrieved using a read function, although this string had to be dynamically allocated due to size variations. Theoretically, if the user were to type in 128 characters within one display refresh cycle, data would be lost. This was not viewed as a problem as such data would likely be keyboard spam and would not critically affect the functioning of the device. The user would just have to type in again, but slower.

Because the calculations for drawing on the display take a long time in comparison to reading ADC data, the data was taken in a contiguous chunk each refresh cycle so as to not have data loss or data overwrites whilst said data is being used. Each chunk is large enough so as to store not only a complete cycle of data at lower than 100 Hz, but also to hold enough data to be zoomed out to the maximum allowed zoom while still having enough data points to fill the entire display. This system, while necessary to avoid data conflicts, uses a large amount of memory, because at the 250,000 sample rate, 25,600 16-bit points need to be stored. The memory size of 100 kb for this PSOC 6 was a limiting factor for the max zoom as this firmware only leaves only 1kb to spare when data is stored. This was a major point of contention as much of the code needed to be optimized significantly to fit within the memory limitation. New chunks are only taken if the scope is in 'run' mode, otherwise the same old chunk is used.

When the scope is reading a new chunk, the function waits until the DMA finishes an entire one of the two ping-pong buffers. When this occurs, an interrupt sets a flag to alert the function which buffer is ready to be read. This process continues until the entire chunk is filled out. The current chunk size as max zoom out is equivalent to 100 ping-pong buffers. This data, which would range from 0x0 (for 0V) to 0x7FF (for 3.3 V), not only had to be normalized to actual voltage values, but also had to be checked for over and under flow which would result in data greater than the 0x7FF value in the 16-bit output of the ADC.

The frequency is calculated by finding the period between the first two trigger points of the waveform chunk. To reduce noise, the average of the last 10 frequency values are displayed instead of the current one, although a ±1% error can still persist due to sampling rate limitations.

Using the xscale (time scale) value, data points are taken from the chunk spaced out so only one point aligns with one vertical pixel value. If the scaling is too zoomed in for there to be enough points, the same point is repeated for multiple pixels until a new value is available. The voltage values of this new data set are normalized using the voltage scale (yscale) value to a certain pixel height, plus the additional pixel height from the potentiometer value. The potentiometer can scroll the 0V point of the waveform all the way from the bottom of the screen to the top. The final result will be a 2 x n array where the two points are the x and y pixel coordinates, and the n is the number of horizontal pixels in the display.

Once the data and parameters are set, the display is refreshed by first drawing a blue background, drawing the white grid, drawing the wave, printing the scale settings in the upper left corner, then printing the frequency values in the upper right corner. The waveforms are drawn using multiple lines from each pixel point to the next. For all of these graphics related functions, except the wave drawing, code was borrowed from Anujan Varma's waveform demo. The APIs used for drawing came from the emWin library.

For all encountered critical errors, a message would be printed to the terminal indicating the nature and location of the error, then the program would stall in a while loop. For small errors that would not affect continuing functioning of the code, an error message by itself would suffice. Most errors related to data reading and transferring in the UART protocol would just result in the data being dumped.

Testing Methodology:

Tinyscope was created in a highly modular way, with each major function being tested independently for bugs and functionality before being implemented in the final design. Additionally, new features were tested one by one so as to ensure proper functioning before adding the next. This resulted in no major bugs during the final implementation as bugs were wiped out in the early stages of testing.

As many of the top-level features relied on previous functions, the result was mostly bug free. One major flaw in the final design was memory usage, which would cause crashes at high zoom settings. Once memory usage was optimized in the major memory hog functions, the result was just under the limit and would function correctly.

During the initial stages of development, the emWin spline API was used to draw the waveforms, and it was more aesthetic due to smoothing than creating lines, however this would cause memory leaks even when properly deallocated due to an emWin bug. We were forced to take the hit in appearance and use line drawing, however it was hard to notice the difference to the untrained eye, especially at high zoom out settings and high frequency waveforms.

Future Additions and Conclusion:

For future additions, many features could be added for greater functionality, as well as performance improvements, given enough development time.

In terms of performance improvements, a big one would be to have the DMA interrupt put the values into the data chunk directly, and freeze writing new data whenever the main function was using it to draw the waveforms. Additionally the frequency calculation and the pixel value calculations can be merged into a single loop, saving cpu time. Although only potentially possible, the second CM0 core could be used for drawing the graphics and the CM4 for the heavy calculations, with synchronization flags between to prevent conflicts. This could greatly increase the maximum refresh rate.

Additional features that could be useful include: exporting waveform data to a CSV file for further analysis, more complex pattern detection, automatic scaling, automatic trigger point detection, scrolling horizontally to get voltage values along the waveforms, and reading data for a prespecified timer period. The display also supported touch screen functionality, so that could also be integrated into the interface potentially for zooming and scrolling.

These improvements could bring this scope very close to the features of many commercially available oscilloscopes at a low cost.

This project showed how critical proper data flow is in bare-metal programming so as not to overwrite needed data.

References:

Final_project_demo - Anujan Varma - 2021

emWin - Segger