Using Advanced CUDA Libraries – David Pfeiffer

Academic Integrity Statement

The work represented in this document is my own unless otherwise stated.

Overview

The goal for the assignment is to explore how the advanced CUDA libraries can be utilized in the course project in addition to satisfying the assignment requirements of using two libraries. The high level flow of the program will be as follows:

- Generate SPI data that can be used later in the final project
- Use cuRAND to apply some noise to the analog data
- Use cuFFT to process the signals and compare how well the clock information can be gleaned from the transformed analog data, noisy analog data, and digitized data.
- Plot the output and manually analyze

Results

To accomplish this I created a CLI app, whose help text is shown below and best describes its features:

```
dave:~/P/j/adv_libs$ ./assignment.exe --help
Usage: square_wave_fft [--help] [--version] [--square VAR] [--sin VAR] [--noise VAR] [--input VAR...] [--verbose] blocks

Positional arguments:
  blocks         number of blocks to use

Optional arguments:
  -h, --help     shows help message and exits
  -v, --version  prints version information and exits
  --square       Generate a square wave to analyze with the given frequency in MHz
  --sin          Generate a sin wave to analyze with the given frequency in MHz
  -n, --noise    Noise scale to use. (Gaussian from 0-1 will be multiplied by this and added to the signal).
  -i, --input    path(s) to the signal data to be processed [nargs: 0 or more]
  --verbose      Enable verbose output
dave:~/P/j/adv_libs$
```

I then used the following commands to generate some data to analyze (included in a run.sh file):

```
dave:~/P/j/adv_libs$ ./assignment.exe 256 --square 6 > square_fft_6mhz.csv
dave:~/P/j/adv_libs$ ./assignment.exe 256 --square 6 -n 0.1 > square_fft_6mhz.csv
dave:~/P/j/adv_libs$ ./assignment.exe 256 --square 6 -n 0.1 > square_fft_6mhz_0_1_noise.csv
dave:~/P/j/adv_libs$ ./assignment.exe 256 --square 6 -n 1 > square_fft_6mhz_1_0_noise.csv
```

```
dave:~/P/j/adv_libs$ ./assignment.exe 256 --sin 6 > sin_fft_6mhz.csv
dave:~/P/j/adv_libs$ ./assignment.exe 256 --input ../dla_data/3_375mhz_data/analog_0.bin > dla_clk__fft_3_375mhz.csv
dave:~/P/j/adv_libs$
```

To perform the analysis, I started with a simple 6MHz sin wave to prove my usage of the cuFFT library was functioning properly. I struggled getting a C++ plotting library to work in time, so I dumped the data out to a CSV and used a spreadsheet to visualize as show in Figure 1.

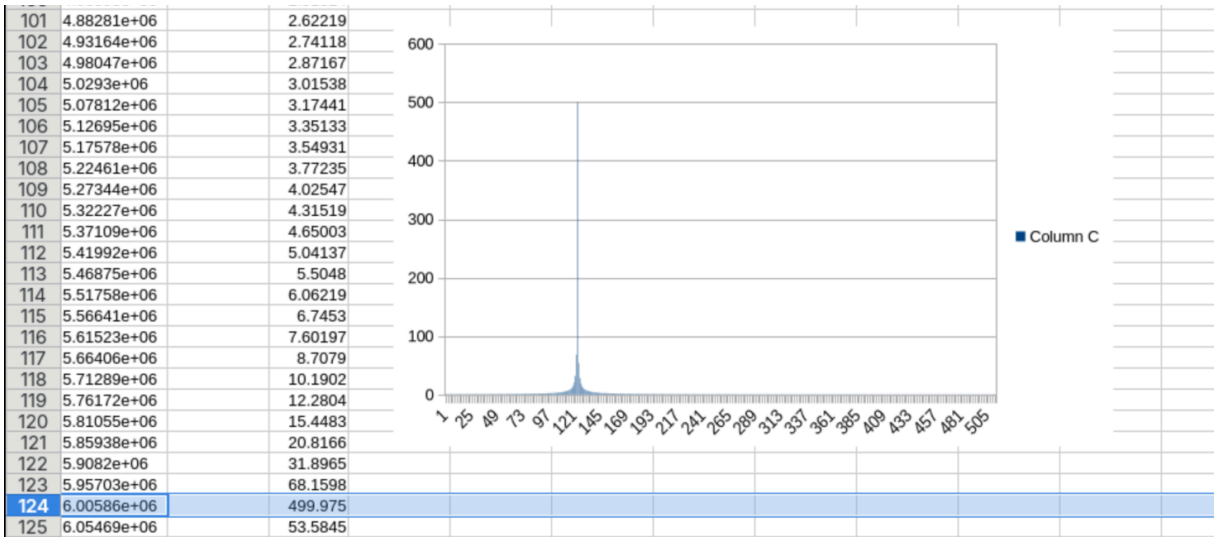

Figure 1: FFT of a 6MHz sin wave. A peak is clearly demostrated a 6MHz.

Next, I attempted data more relevant to my project with a square wave that represent a digital clock and also used cuRAND to inject white noise on top of it. This process led to somewhat satisfying, if expected, results shown in Figure 2.

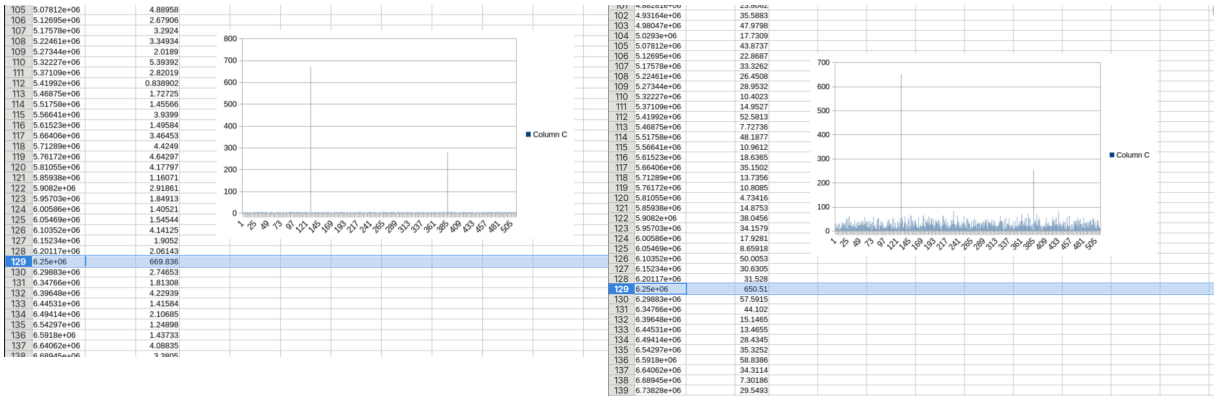

Figure 2: FFT of a 6MHz square wave with and without noise. Peaks are clearly visible at approx. 6MHz and 18MHz as would be expected, with a “noise floor” present in the second sample.

Spurred by the success, I took an MCU development board I had lying around and wrote a program to generate random SPI transfers at 8 different baudrates. With the MCU working, I hooked up a digital logic analyzer, captured transactions at the 8 different buadrates, and dumped the raw data to my PC. With the data laboriously gathered, I wrote a simple parser for the DLA’s file format and tested one of the clock signals shown below in Figure 3 and Figure 4.

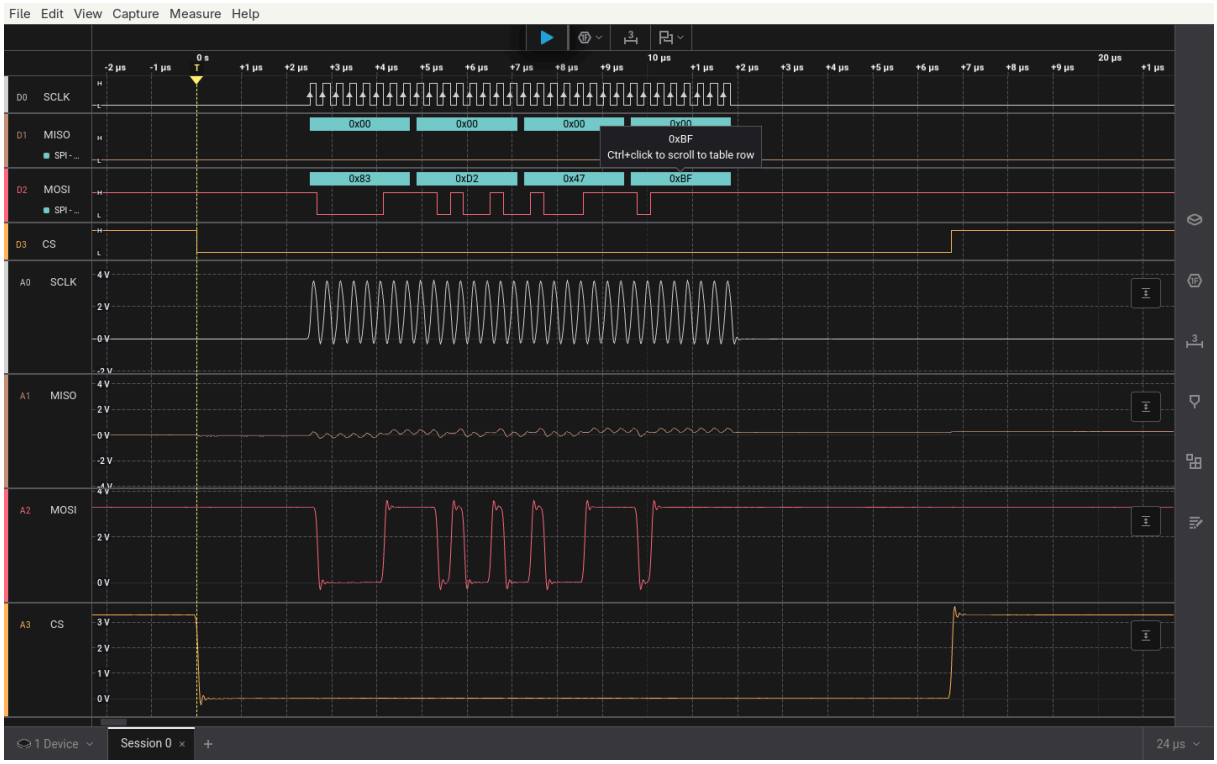

Figure 3: DLA capture of a 3.375MHz SPI transfer. You can see the clock is somewhat "squashed" by the parasitic capacitance at this speed.

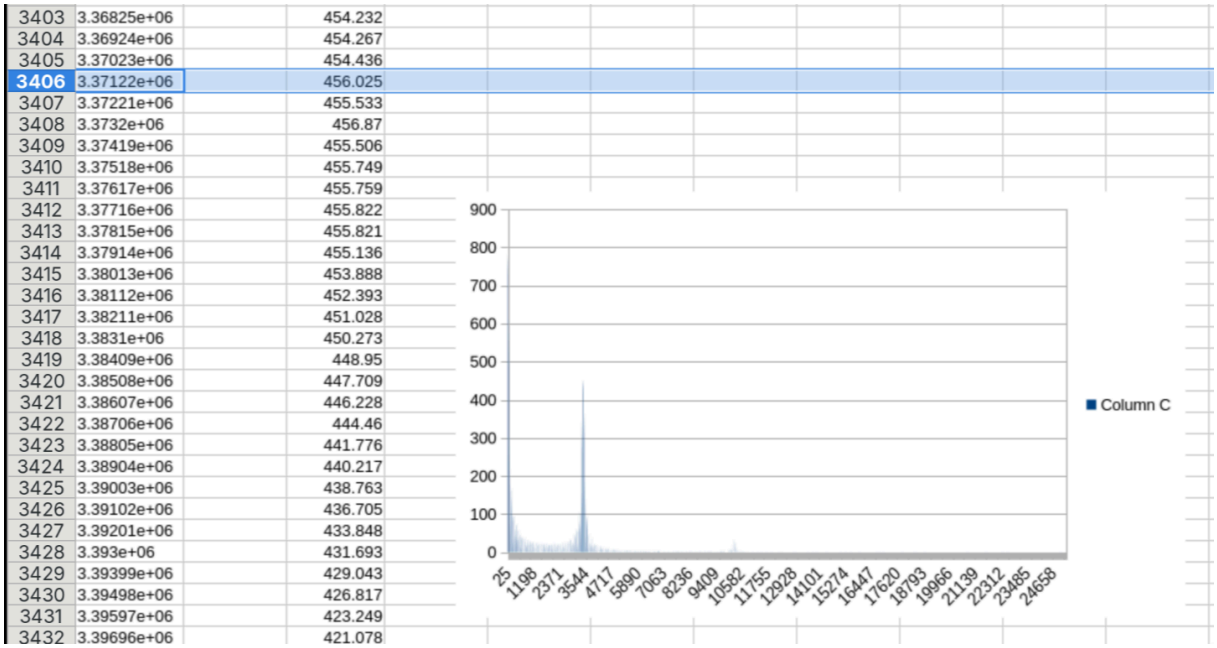

Figure 4: FFT of the SCLK signal from Figure 3. The DC component is clearly visible in the low frequency bins and the clock train resembles the sin wave's FFT more than the square wave.