**Software Documentation – Summer Research 2018**

The software in this project can be divided into two parts at a high level:

1. Interface with the Nexys Video FPGA by receiving UDP packets over the ethernet port, decoding the data, analysing with an FFT algorithm, and displaying the results in real time.
2. Interface with the Nexys 4 FPGA by sending parameters through USB to the signal generator hardware to produce signals with specific characteristics

**Part 1**

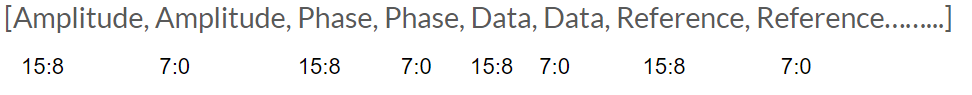
**Receiving Data from the FPGA**

The program receives data from the FPGA by using UDP communication through an ethernet port. The IP address is pre-set to 192.168.1.12 and the port to 5010 to match the hardware parameters. UDP packets are coming in at a high frequency, and in order to make a stable connection, the socket library for python is used.

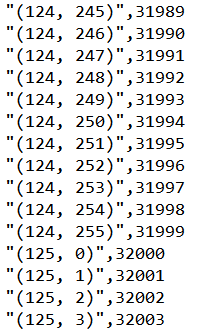
At a time, 500 UDP packets (each of size 1040 bytes) are received, decoded, analysed and plotted. At the end of the cycle the UDP socket is closed and re-opened at the next iteration. The cycle continues indefinitely.

**Decoding Data**

The data received from the FPGA is in an unsuitable format and thus needs to be decoded. From the FPGA, the numbers are coming in as 16 bit binary numbers, but python can only receive 1 byte as an individual point. Therefore, each number is broken up into two halves as shown in the figure below:



Thus, each 1040 byte packet can only store complete information about 130 points. The numbers are read as unsigned values. To decode into meaningful information, each number must first be converted into an 8 bit binary, the two halves must be concatenated, and then it must be read back as a 16 bit number. To optimize performance and reduce computations, all the possible pairs that can be received have been computed and hash tables have been constructed to match a tuple of two values to an output. An extract from the hash table is shown below:



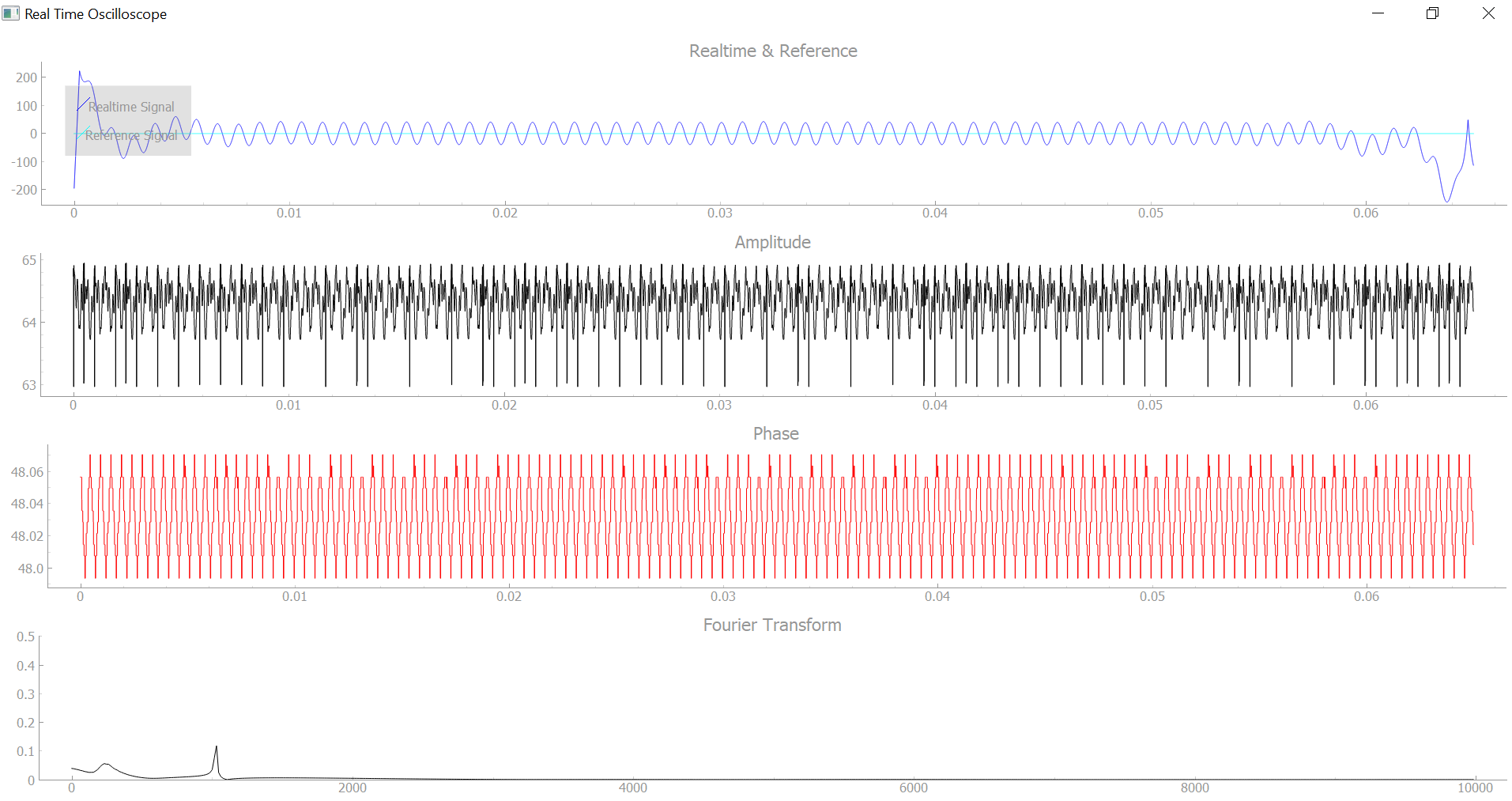
**Signal Processing**

The FPGA gives information about the signal, its amplitude, phase and reference signal (to which it is compared for phase calculations). In python, the incoming data from the decimation filter is smoothened out using a moving average filter, and then the DC offset is removed by using a high pass filter. Finally, an FFT analysis on the real time signal is done, based off the predefined sampling frequency. Thus, the FFT algorithm takes it from time space to frequency space. The absolute value of the fft results is taken and to remove the negative frequencies, only half the domain is analysed. The signal processing is done in batches of 500 packets (65000 points) as they come in.

For the signal processing another python library called ‘scipy’ it utilized. The FFT function, high pass filter, and the moving average filter are all imported from this library.

**Real Time Display**

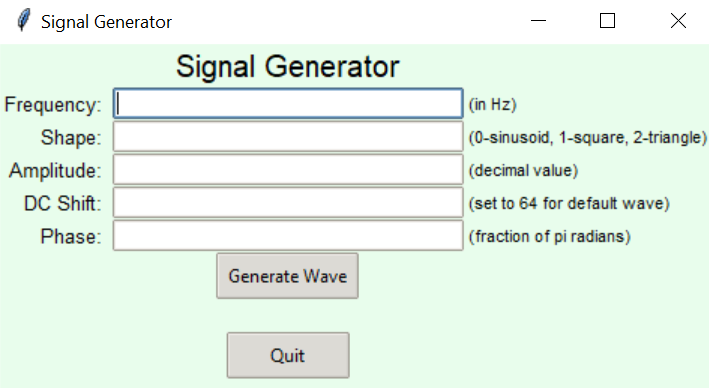
Once all the relevant data from the signal has been decoded and the FFT has been decoded, the data is plotted. Again, this is done in batches of 500 packets (65000 points). On the real time display, all 65000 points are plotted together. This is done with the help of the APIs PyQt5 and pyqtgraph. The display is implemented as a class that instantiates a GUI and has many functions defined within. One of these functions is the update function that updates the plot every time the timer runs out. Currently the timer is set to run out when the previous frame is complete, and all 65000 points are displayed. This functionality gives a frame refresh rate of 2 fps. This refresh rate can be adjusted to be made much higher (30fps - 40fps) if less than 65000 points are plotted [this adjustment must be made in code as indicated by the comments]. But, to be safe, the parameter is set to 65000 points. An example of the real time display can be seen below:



**Part 2**

**Interfacing with the Signal Generator on the Nexys Video**

To interface with the second FPGA, a completely different code is executed. This code creates a custom GUI for the signal generator. For this purpose, a library called ‘tkinter’ is used. The minimal GUI accepts text in the given fields and passes on those parameters to configure the hardware on the FPGA. These parameters are used to generate a specific signal. the signals can be sinusoidal, triangular or square, and the user can configure the DC offset, amplitude and phase. A screenshot of the GUI is presented below:



**Combining Part 1 and Part 2**

Parts 1 and 2 are made to run in parallel because they share no common straits and do not need to share any data structures or I/O data. To achieve this parallel execution to the best level, multiprocessing is used via the python multiprocessing library. This method of parallel execution (as opposed to the more conventional multi-threading technique) is used because it utilizes separate cores of the CPU to truly run the processes in parallel as opposed to dividing its time in micro batches and processing the two processes sequentially. Disadvantages of this include a large memory footprint and the computer heating quickly. The main advantage however, is that the performance of either program isn’t compromised and the refresh rate of 2 times per second can be maintained.

**Next Steps**

From this point onwards, the next steps in terms of software include calibration. The amplitude needs to be calibrated to match the digital values to the voltage of the signal. at this point, the amplitude is meaningless because it changes when we change the decimation rate (on the decimation filter) but it is proportional to the voltage of the signal. Similarly, calibration is needed for the phase. Currently the main signal goes through the decimation filter and the phase and amplitude detection blocks. There are also high pass and low pass filters. In contrast, the reference signal just goes from one fpga to the other. The delay due to hardware blocks is not equal in these two signals, and that may introduce a phase offset due to delay. This offset needs to be identified and accounted for in calculations.

Another improvement in the project would be to integrate the blocks from the two fpgas onto a single fpga, but currently it is not possible to do so without getting a higher end fpga because the resources are too limited for both the projects together. This may be fixed by optimization of the hardware design. Finally, the software can be built upon to a further extent. There can be a mode for automation of signals to keep the phase and amplitude of the signal received from the biological tissue within certain ranges.