EIS Readme, Presented by TAMU ECEN Team 18 for the Lynntech EIS team. For any questions please email quentinboothman@gmail.com

This document will serve as a guide to the various programs and code relevant to the Electrochemical Impedance Spectroscopy project for Lynntech, available through the project github (https://github.com/anguyen721/TEAM18EIS). All breakdowns will be by folder and then by file unless files reference each other.

- Capstone
  - **capstone.cache**, **capstone.hw**, **capstone.ip_user_files**, **capstone.runs. capstone.sims**, **capstone.srcs**, **capstone.xpr**
    - These folders are all part of the "capstone" Vivado fileset, which exists for deployment of the files in **capstone.srcs**
    - **capstone/capstone.srcs/sources_1/new** contains **adc_simple.v**, **adcCode.v**, **sintable.v**, **nco.v**, **fpga_run.v**, **and table.txt**
      - **adc_simple.v**: This file is meant as a very simple on-pc test of the other systems with respect to the "ADC" program, which is supposed to take in voltages from the ADC and write them to an on-board file for later calculations. This file is not synthesizable due to the fwrite function
      - **adcCode.v**: This is a more sophisticated version of **adc_simple.v**, which stores the values to an 8 x 256 bit table before writing every 256 ticks, saving fwrites. It is also not synthesizable due to the fwrite function, but the function could be replaced, making it synthesizable.
      - **sintable.v:** This function takes in a clock and phase input, and outputs an amplitude (o_voltage) periodically, using a table generated elsewhere, so long as reset and control inputs are in the correct positions. This file is synthesizable as it uses the function $readmemh, but because it requires a pregenerated lookup table, it is still not completely FPGA-sufficient.
      - **nco.v**: This function takes in a clock and phase division input and outputs an 8-bit amplitude (o_voltage) contingent on a control input. It uses a numerically-controlled oscillator to generate a phase (to act as input for **sintable.v**) with the fewest possible clock ticks per generated phase, then outputs the amplitude generated by **sintable.v**. It works very simply, by advancing a 32-bit (adjustable width) register by the input value iphase (which determines the ratio to the frequency of the output of **NCO.v** to the input clock frequency. The top 8 bits (adjustable) of this register are used as inputs to **sintable.v**, and o_voltage is set to the output. This was the newest version of the periodic wave

        generation program, and worked well to generate waves on the board.
    - **fpga_run.v:** This is a synthesizable demonstration file designed to show that both the periodic wave generation and wave recording could run on the board at the same time. It calls **nco.v** and **adc_simple.v**.
    - **table.txt:** This is an example lookup table used by **sintable.v**, in the format required for the function $readmemh. It was generated using a labVIEW program that computes 8 bit voltage values using the sine table generation formula. The values are stored as hexadecimals in the table.
- ECEN403SineTable
    - Used to test whether or not the GUI produces a correct sine table to be used by the Numerically controlled oscillator.
    - The program creates its own version of the sine table and runs it against the created table and checks the entire table for correctness.
    - To run the program have the sine table text file in the same file as the program. Make sure that in the code, the file name of the file being open is correct before running the program.
    - The output window displays how many errors there are in the sine table if any.
- LabVIEW GUI
    - Uses LabVIEW to create a GUI for the user to input the dual frequencies, then displays the waveform made by the frequencies. Also accepts the final impedance calculation results and displays them as well as the waveform produced by the output frequencies. Also calculates a Nyquist plot from the final calculation results.
    - The program is contained entirely in the **ControlGio.vi** program.
    - It prompts the user to specify the location to save the output file to and where to read the final impedance results and the sine table for the output from.
    - There is built error checking to make sure the input frequencies are within the acceptable range, as well as whether the save and output file paths have been specified. It uses a constantly running textual system status indicator to keep the user informed.
- Communication
    - All files needed to construct the communication system is found under the Communication folder in the repository.
    - The overall logic in the project final report that details how this program works is a very high level explanation. I found this logic and the original program on an online blog post found here: https://zipcpu.com/blog/2020/04/08/axitb.html. All of the files are completely open source and copyright free, and have been permitted for use in any context.
    - Note: explaining the entirety of the project and the details of my changes would have taken roughly 200 pages so I have linked the blogpost which goes into

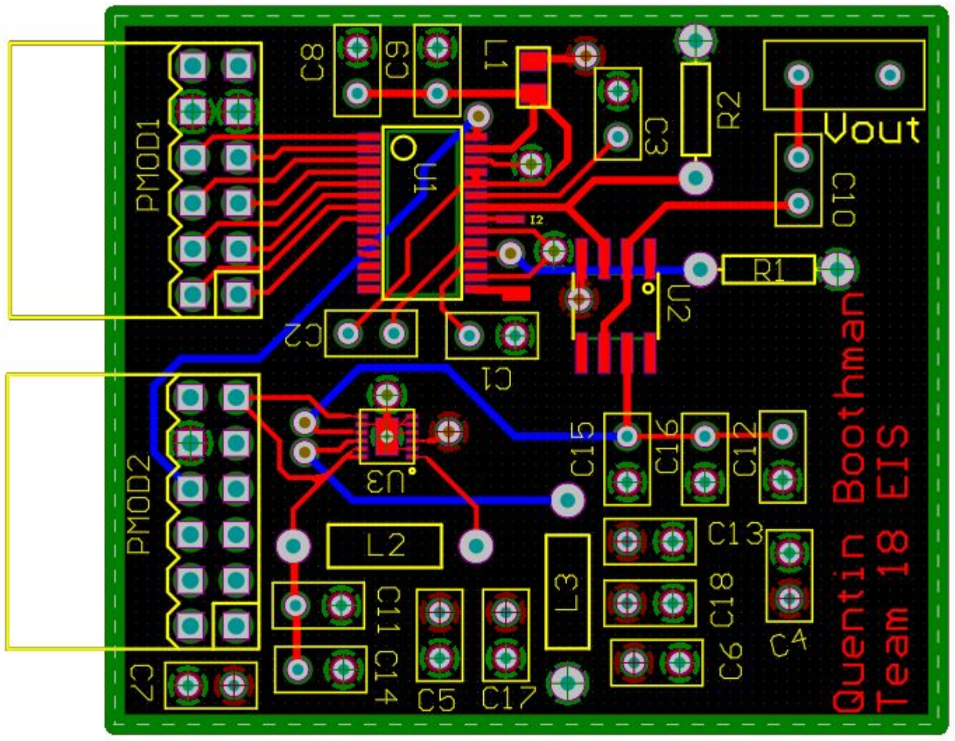
immense detail and is generalized so the solution can easily be adapted to be used with any board, not just the one used for this project. This will prove very helpful in future endeavors as a much better board will surely be used for further development of the EIS device.

- The blogpost details achieving this in simulation, but the files were modified to work with the Zybo Z7 and an SDK was successfully created and ran on the board as indicated in the final report.
- Individual slaves control the basic operations of reading from and writing to memory. Writing is handled by **axis2mm.v** and reading by **aximm2s.v**. A memory to memory transfer slave is also included under **axidma.v** although this wasn't modified to be used with this project as it was not needed for any functions outlined in the scope.
- The slaves are connected together using a bus to simplify communication with the master. This bus is found under **axildouble.v**.
- The bus then needs to connect the axi interconnect which in turn connects to the Master. However, the slaves utilize axi-lite and the axi interconnect uses the full axi interface. So, an axi to axi-lite core is used to link the bus with the axi interconnect. This can be found under **axi2axilite.v**.
- As mentioned in the report, Xininx has their own AXI Interconnect that can be used, found in the IP catalogue. However, this ties the design to Xilinx and cannot be used by FPGAs using Intel's designer. Similarly, Intel has a platform designer to make the interconnect that cannot then be used with Xilinx.
- Note: When using anything other than a Zynq, or even perhaps a Zynq with access to a serial port from the fabric which our FPGA did have, then the easy way to write to anywhere in the address space is to create an AXI master that will accept read/write commands from a UART and write them to the address space. This is what was done. If, however, a Zynq is used and the only serial port access to the device is a console port directly connected to the ARM (PS) within it, then there is another method for writing to the memory space. This was not detailed in the blogpost but I have found that if running Linux, it is possible to either FTP the file to the board and then write a small program to write it to memory, or to use zmodem to move it across the serial port. Zmodem is another open source program that can easily be found online.

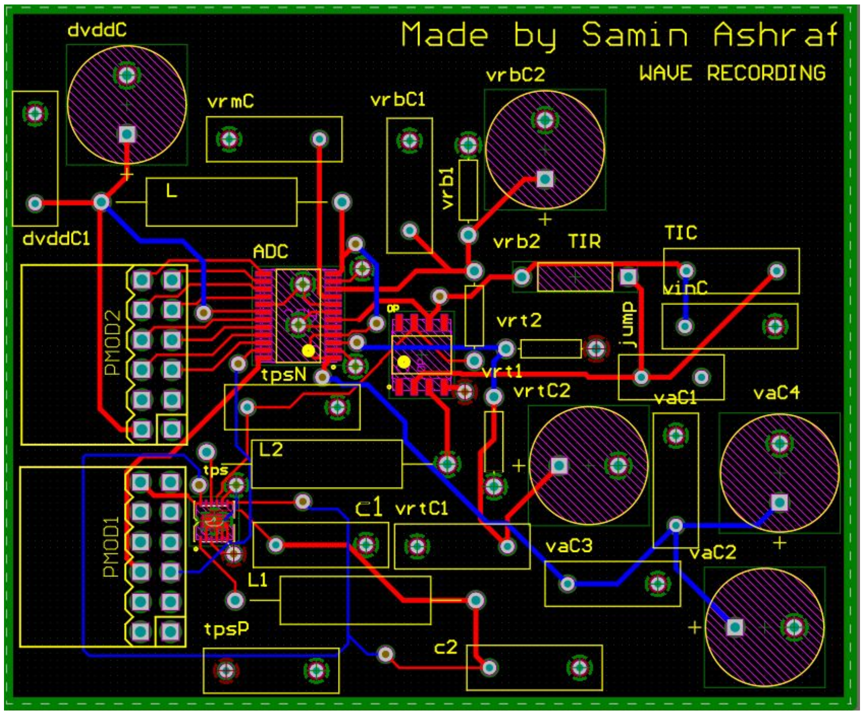
- Impedance Calculations
    - USES FFTW fourier transform library http://www.fftw.org/
    - The program contents are contained in source.cpp
    - Takes in a file that contains data on input and output waves and computes the magnitude of the impedance. Test files are provided in the test_tabbed folder
        - These files are named in the following order scheme "Freq1 Freq 2 Resistance || Capacitor"
            - Frequencies are in MHz unless specified, Resistance is measured in Ohms unless specified

            - Example “19100R32pf’ is 1 Mhz 9 Mhz 100 Ohms in parallel with 32 pf
    - Digital filtering is included in this program under the digital filtering function created
    - The calculation that is used is shown below:
        - $$|Z| = \frac{\frac{V\,out}{V\,in} * Rref}{1 + \frac{frequency}{10^7}}$$
    - The program outputs the magnitude of the impedance of the test circuit
    - N value at the beginning of the program designates how many data points sto use on the fourier transform. N will have to be reduced for smaller frequency tests when going under 1Mhz as there are less data points. Ideal numbers to use are 2^x numbers(2,4,8,....)
    - The program sees higher percent error the more the frequencies increase. This may be due to some type of precision inaccuracies.
- PCB Designs
    - Quentin PCB
        - This is an Altium PCB Design that could be fabricated by any PCB manufacturing service, of the filetype ‘*.pcbdoc’. If another filetype is required, limited filetype conversion ability exists within Altium.
        - This board’s purpose is to connect a DAC, the TI THS6541, to:
            - All auxiliary parts required for its operation
            - The Zybo Z7-10 FPGA via a 2 x 6 pMod connector
            - To an output jumper to be connected to the blood impedance circuits

        - In addition to the file present in the github, a reference image of the board layout is included below.



- Samin PCB
    - Design made on Altium.
    - This board's purpose is to connect a ADC, the TI ADC08100, to:
        - Utilizes a input jumper to connect the blood impedance circuit's current signal
        - Outputs the digital signal to the Zybo Z7-10 FPGA via a 2 x 6 pMod connector
        - All auxiliary parts required for its operation
    - Image below



- XADC
  - **XADC_constraints.xdc:** This is a board constraint file for the Zybo Z7-10 FPGA, designed for the XADC recording fileset. All constraints are commented out, except the following:
    - Each of the LEDS is used, to display the voltage values recorded
    - PMOD Header JA (XADC) is completely uncommented, although only two ports were used, as all 8 ports are needed for the XADC module
  - XADC.v runs the XADC program in conjunction with the board files, recording waveform inputs and displaying them on the on-board LEDs in 4-bit binary form, normalized for a 1V- Amplitude periodic wave