Programming of a Python GUI to Control Two Zurich HDAWGs and Layout of Three Printed Circuit Boards

Richard McManus
Electrical Engineering
University of Notre Dame
Notre Dame, IN
rmcmanu2@nd.edu

***Abstract* – This report describes the progress made during the Spring 2023 semester in Dr. Snider's research group and focuses specifically on my contributions. The projects described consist of the programming of a python script to control two Zurich HDAWGs and the layout of three distinct printed circuit boards (PCBs) designed for the level shifting of signals between a Virtex-7 VC707 Field Programmable Gate Array (FPGA) and a device under test (DUT).**

I. INTRODUCTION

This semester I had the opportunity to continue working with Dr. Snider's research group. My contributions included programming in python and designing PCBs using Autodesk Eagle. In the following sections, I will discuss the methods and processes applied and the results obtained.

II. Programming of a Python GUI to Control Two Zurich HDAWGs

A. Background

This project required process improvements to streamline the programming of two Zurich HDAWGs for testing a DUT (a microprocessor without interlocked pipelined stages (MIPS) in this project). Currently, the HDAWGs are intended to generate 12 positive trapezoidal waveforms to use as clock signals for the MIPS upon being triggered by a Virtex-7 VC707 FPGA. These 12 signals will then be inverted by a clocking board to generate 24 clock signals: 12 positive clock signals and 12 negative clock signals. A block diagram of this process is shown in Figure 1.

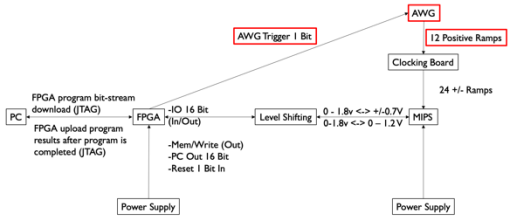

Figure 1. Block Diagram of MIPs Testing (AWGs)

This project built upon Jack Bannon's work in the Spring semester of 2021 to generate trapezoidal waveforms using Zurich's LabOne interface. These waveforms were generated with hardcoded values by cutting and joining rectangular and ramp waveforms. This method is shown in Figure 2.

```
//Waveform 1 (2t,1t,4t,1t,2t) (zeros,ramp,rect,rampdown,zeros)
wave w_flat = rect(4*samples, 1.0); // Generates rectangle wave with constant 1
wave w_trap1 = join(w_zero, w_rise, w_flat, w_fall,w_zero); // combines them into one waveform (8400 samples total)

//Waveform 2 (2.5t,1t,3t,1t,2.5t) (zeros,ramp,rect,rampdown,zeros)
wave w_flat2 = rect(3*samples, 1.0); // Generates rectangle wave with constant 1
wave w_trap2 = join(w_zero2,w_rise, w_flat2, w_fall,w_zero2); // combines them into one waveform

//Waveform 3 (3t,1t,2t,1t,3t) (zeros,ramp,rect,rampdown,zeros)
wave w_flat3 = rect(2*samples, 1.0); // Generates rectangle wave with constant 1
wave w_trap3 = join(w_zero3,w_rise, w_flat3, w_fall,w_zero3); // combines them into one waveform

//Waveform 4 (3.5t,1t,1t,1t,3.5t) (zeros,ramp,rect,rampdown,zeros)
wave w_flat4 = rect(640, 1.0); // Generates rectangle wave with constant 1
wave w_trap4 = join(w_zero4,w_rise, w_flat4, w_fall,w_zero4); // combines them into one waveform
```

Figure 2. Initial Method of Trapezoidal Waveform Generation

During the Fall semester of 2021, a python script was developed through collaboration with Jack Bannon to load in waveforms from a comma separated values (CSV) file and output the waveforms on up to eight channels of a single HDAWG. This script (ReadandCreateWave(1.2).py) ran from the command line and was incapable of adjusting attributes such as frequency and was incapable of synchronizing multiple HDAWGs. An example output of trapezoidal waveforms generated by this python script is shown in Figure 3.

Figure 3. Trapezoidal Waveforms Generated by ReadandCreateWave(1.2).py

The goal of this project was to build upon ReadandCreateWave(1.2).py by developing a graphical user interface (GUI) to load in waveforms from a CSV file and allow the user to specify attributes such as frequency, sample rate, and triggers before generating the waveforms. Such a GUI would enable quick adjustments and expedite the process of programming the HDAWGs. Additionally, the program required the two HDAWGs to be synchronized to allow for use of up to 16 output channels.

B. Converting ReadandCreateWave(1.2).py into a GUI

The initial focus of this project was on incorporating the functional elements of the previously created ReadandCreateWave(1.2).py python script into a GUI to visualize a preview of waveforms before generation. This visualization made use of multiple open-source python libraries as shown in Figure 4. PySimpleGUI was utilized to create the framework of windows and tabs. Matplotlib allowed for waveform preview visualization. An early version of the GUI that loaded in and visualized CSV waveforms is shown in Figure 5.

```
import hdawg
import time
import PySimpleGUI as sg
import os.path
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import
FigureCanvasTkAgg
import numpy as np
from sys import platform
```

Figure 4. Imported Python Libraries

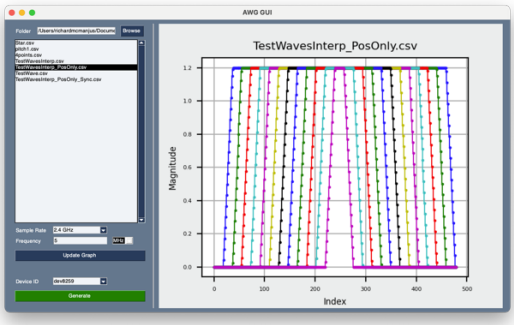

Figure 5. Early Version of Python GUI

This version of the python GUI successfully allowed the user to select a folder containing CSV waveform files, preview waveforms, and generate the waveforms on a single HDAWG.

C. Initial Attempt at Device Synchronization by Splitting External Trigger

An initial attempt at synchronizing multiple HDAWGs was made by configuring the devices such that both devices used the first HDAWG's internal reference clock. In this way, the reference clocks were synchronized. An external trigger signal was split using a BNC T adapter and was connected to the first trigger input of each HDAWG. Both HDAWGs were programmed with a sequence that waited to be triggered before initiating waveform playback. This version of the GUI was capable of programming multiple devices and is shown in Figure 6.

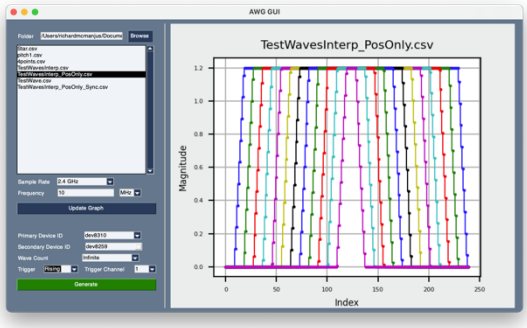

Figure 6. Second Version of Python GUI with Implementation of Multiple Devices

The results obtained upon testing this method of synchronization were inconsistent. Some attempts synchronized while others were nearly 180 degrees out of phase. Due to these inconsistent results, it was assumed that nonsystematic errors existed within the system. Therefore, alternative methods of synchronization were explored.

D. Second Attempt at Device Synchronization using Multi-Device Synchronization

A second attempt at synchronizing multiple HDAWGs was made in accordance with the procedures outlined by Zurich Instrument's HDAWG User Manual. The user manual described a feature of the HDAWGs known as multi-device synchronization (MDS). This feature was intended to allow the user to program a collection of HDAWGs as if they were one single device. To utilize MDS, the HDAWGs' MDS ports needed to be connected in a loop and an external reference clock supplied as shown in Figure 6.

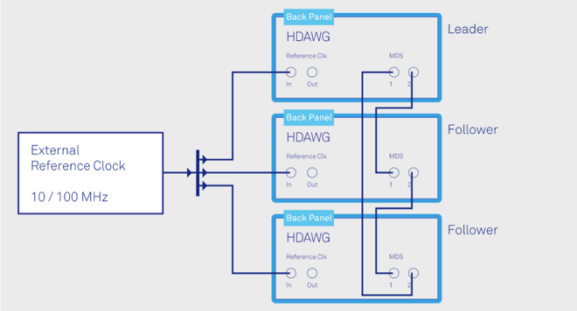

Figure 7. Configuration of HDAWGs for MDS

To incorporate the MDS feature, a MultiDeviceSyncModule had to be initialized using the Zurich LabOne Python API. To initialize such a module and implement it into the previously created GUI, a separate python file, *mds.py*, was created. This file contained numerous functions to initiate an MDS module, generate a LabOne sequence to be to split among the HDAWGs, and run the sequence to generate the waveforms. The outline of *mds.py* is shown in Figure 8.

```
mds.py > ...
1   import textwrap
2   import numpy as np
3   import os
4   import zhinst.utils
5   import time
6
7   def initiate_mds(daq, device_1, device_2):…
36
37  def generate_mds_program(array, mds, awgModule, trigger = '4',
38      trigger_channel = 1, count = 'Infinite'):…
102
103 def run_mds_program(daq, device_1, device_2, awgModule, awg_program):…
```

Figure 8. Outline of mds.py

After the implementation of *mds.py* within the source code of the python GUI, *AWG_GUI.py*, the waveforms generated as expected approximately one out of every ten attempts. The failed attempts caused the HDAWGs to crash requiring manual resetting of the HDAWGs by powering them off and back on again, a process that takes approximately five minutes.

E. Troubleshooting MDS with Zurich Rep

After attempting to correct this issue independently, Zurich Instruments Support was contacted to assist with troubleshooting. They suggested using the LabOne MDS tab to observe if the issue persists. Using the LabOne interface and MDS procedures outlined on page 231 of the HDAWG User Manual, the same error was observed. Approximately 9 of every 10 attempts at initiating a MDS group resulted in the crashing of both the LabOne interface and HDAWG devices. These results were reported back to Zurich Instruments' support representative.

After many emails confirming the procedures for MDS were followed correctly, the Zurich Instrument's Support representative attempted to utilize the MDS feature and observed the same error. The representative internally elevated the issue and learned that this was a known bug and one that would not be corrected moving forward. The representative suggested the implementation of the Zurich Instruments Programmable Quantum System Controller (PQSC), a $10,000 piece of equipment intended to precisely synchronize multiple devices. Zurich Instruments offered to send Dr. Snider's research group a demo unit of the PQSC to test for two weeks. The PQSC is shown in Figure 9.

Figure 9. Zurich Instruments PQSC

F. Third Attempt at Device Synchronization using Delay Compensation

While the demo PQSC unit was being shipped, a third attempt was made to synchronize multiple devices. This method utilized the internal reference clock of one device, called the secondary device, for both HDAWGs. The secondary device was triggered using a marker output of the other device, called the primary device. This configuration is shown in Figure 10.

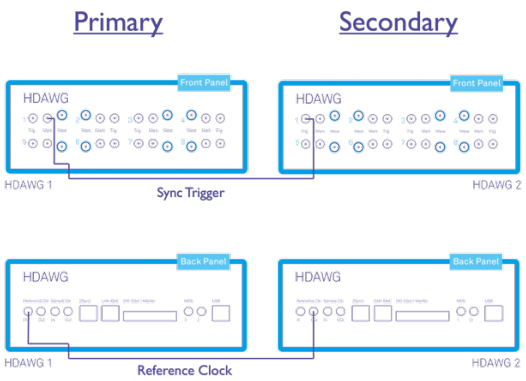

Figure 10. Delay Compensation Configuration

Using this configuration, the secondary device is initiated first and programmed to wait for a trigger from the primary device. Upon initiation of the primary device, a trigger pulse is generated. This trigger enables the waveform playback of the secondary device. This process is shown in Figure 11. As the trigger to output delay of the Zurich HDAWG is near 150 ns, the primary device would initiate waveform playback approximately 150 ns earlier than the secondary device without the implementation of delay compensation.

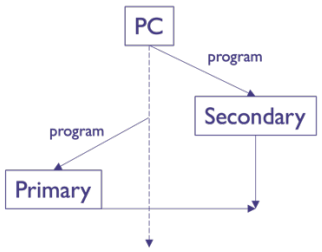

Figure 11. Delay Compensation Timeline

Two methods were implemented to correct the offset between devices. The first, a sequence clock offset, can be thought of as a coarse-tuning knob for synchronization. This method delays the waveform playback of the primary device by discrete values of sequence clock cycles lasting 3.33 ns. This offset is programmed directly into the sequence of the primary device as shown by the function, *wait(46),* in Figure 12.

Primary: dev8310

```
var run = 1;
wave w1 = "wave1";
wave w2 = "wave2";
wave w3 = "wave3";
wave w4 = "wave4";
wave w5 = "wave5";
wave w6 = "wave6";
wave w7 = "wave7";
wave w8 = "wave8";
while(run){
waitDigTrigger(1);
setTrigger(0b0001);
wait(46);
setTrigger(0);
repeat(100){
playWave(1, w1, 2, w2, 3, w3, 4, w4, 5, w5, 6, w6, 7, w7, 8, w8);
}
}
```

Figure 12. Sequence of Primary Device with Sequence Clock Offset

The second method of delay compensation, a sample clock offset, can be thought of as a fine-tuning knob. This method rotates the wave arrays of the secondary device by a discrete number of elements. This corresponds to delaying the secondary device by a discrete number of sample clock cycles or a time delay of the inverse of the sample rate. A sample clock offset effectively shifts the waveforms of the secondary device forward or decreases its phase. A visualization of both methods of delay compensation is shown in Figure 13.

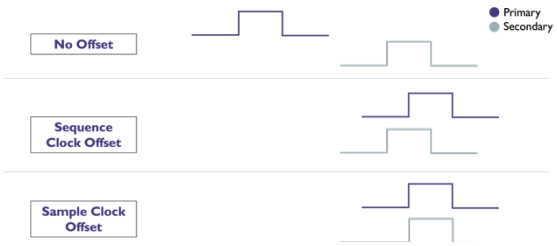

Figure 13. Visualization of Delay Compensation Methods

G. Comparison of Synchronization Methods – Zurich Instruments PQSC vs. Delay Compensation

Upon arrival of the PQSC, the procedures outlined in the PQSC User Manual for synchronization of multiple HDAWGs were followed to set up the device. The user manual outlined a synchronization test to be performed by generating pulses lasting 64 sample clock cycles on two HDAWGs and initiating waveform playback using the PQSC. Figure 14 shows the expected results of this test provided by the PQSC User Manual.

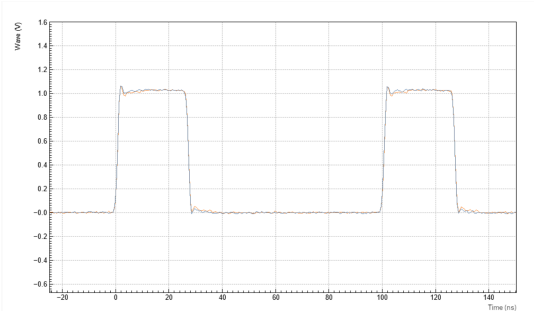

Figure 14. Expected Results of PQSC Synchronization Test

This synchronization test was replicated successfully as shown in Figure 15. However, upon decreasing the time scale to 200 ps per division, a delay of approximately 250 ps between devices was observed. This delay is shown in Figure 16.

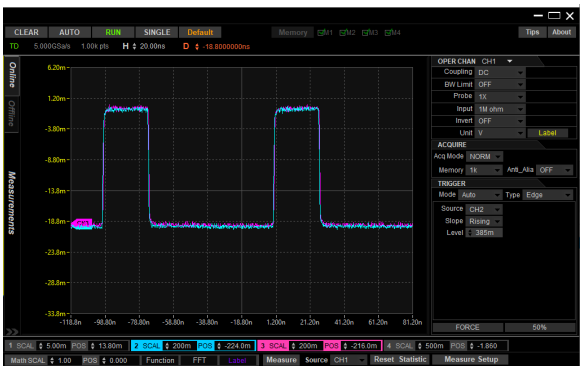

Figure 15. Results of PQSC Synchronization Test Replication (20 ns per division)

Figure 16. Results of PQSC Synchronization Test Replication (200 ps per division)

As the intention of this project was to generate trapezoidal waveforms, a second synchronization test was performed by programming each HDAWG to generate identical trapezoidal waveforms. Similar results were observed for this test and are shown in Figures 17 and 18. The PQSC consistently demonstrated a delay of approximately 250 ps between devices.

Figure 17. Results of PQSC Trapezoidal Waveform Synchronization Test (2 ns per division)

Figure 18. Results of PQSC Trapezoidal Waveform Synchronization Test (200 ps per division)

The same test was conducted using the delay compensation method of synchronization. The results of this test are shown in Figures 19 and 20. The method of delay compensation achieved a delay of approximately 150 ps between devices.

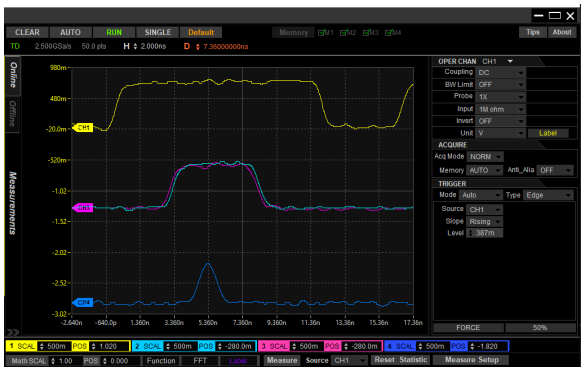

Figure 19. Results of Delay Compensation Trapezoidal Waveform Synchronization Test (2 ns per division)

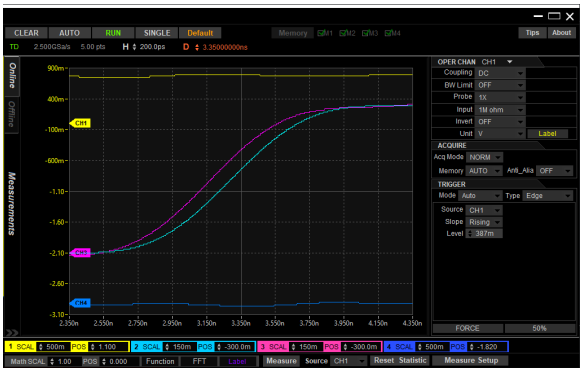

Figure 20. Results of Delay Compensation Trapezoidal Waveform Synchronization Test (200 ps per division)

These results showed that the synchronization achieved using the method of delay compensation outperformed the synchronization of the PQSC. Therefore, the group opted to return the demo PQSC to Zurich Instruments.

The testing of synchronization was continued by comparing the synchronization of two HDAWGs using the method of delay compensation with the internal skew between adjacent channels of the same HDAWG. Channels 1 and 2 of two HDAWGs were programmed with identical trapezoidal waveforms. The configuration of oscilloscope and HDAWG channels is outlined in Table 1.

Table 1. Configuration of Test to Compare Synchronization of Devices with Internal Skew

Scope Ch.	HDAWG #	HDAWG CH.
1	1	1
2	1	2
3	2	1
4	2	2

The delay between devices (Scope Ch. 2 and Ch. 3) fell between 100 and 150 ps. The skew between channels of HDAWG #1 (Scope Ch. 1 and Ch. 2) fell between 200 and 250ps and the skew between channels of HDAWG #2 (Scope Ch. 3 and Ch. 4) fell between 150 and 200ps. These values of skew between channels are near Zurich Instruments' expected value of less than 200 ps. The results of this test are shown in Figures 21 and 22.

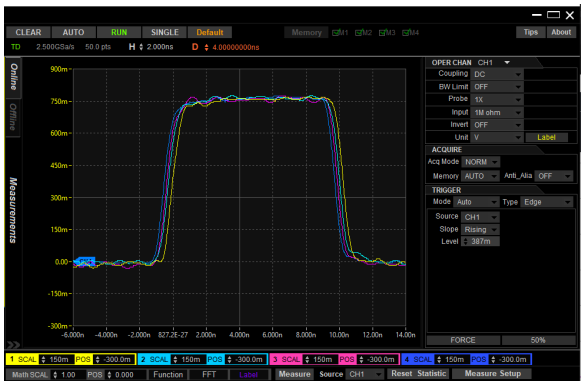

Figure 21. Results of Test to Compare Synchronization of Devices with Internal Skew (2 ns per division)

Figure 22. Results of Test to Compare Synchronization of Devices with Internal Skew (200 ps per division)

The results of this test demonstrated a synchronization between devices that outperformed the skew between adjacent channels of the same HDAWG.

H. Overview of Finalized GUI

With successful synchronization, the GUI was finalized to be user-friendly for programming and debugging. The control panel of the finalized GUI is shown in Figure 23.

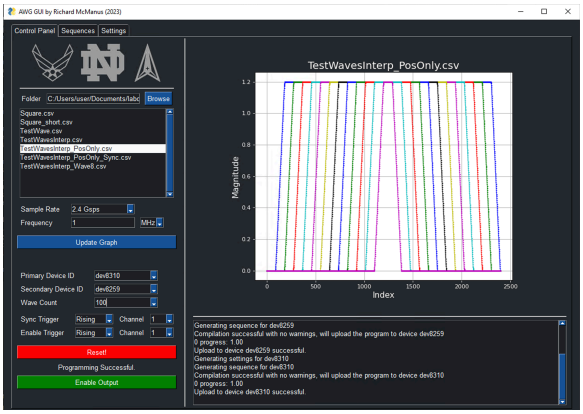

Figure 23. HDAWG GUI Control Panel

From the control panel, a user can select a folder containing waveform CSV files, preview waveforms, and specify sample rate and frequency. A primary device is specified, and a secondary device may optionally be specified if more than eight channels are required. Synchronization trigger channel and type may be specified if using multiple devices. Additionally, an enable trigger channel and type may be specified if an external trigger is required to initiate waveform playback. Finally, the user is able to program the HDAWGs with their specified configuration. Updates are printed to the console and the user will be able to enable the generation of waveforms upon successful programming.

The sequences tab, shown in Figure 24, allows the user to view the sequences generated by their configuration of the control panel and easily debug any runtime errors.

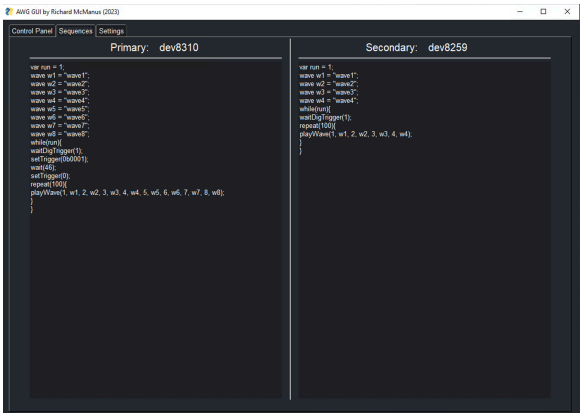

Figure 24. HDAWG GUI Sequences Tab

Finally, the settings tab, shown in Figure 25, allows the user to configure the offsets for the synchronization of two HDAWGs. The default offset values are set to optimize synchronization of the HDAWGs in Dr. Snider's lab; however, sliders are implemented to allow the user to adjust these offsets should other HDAWGs be used, or other sources of latency be introduced.

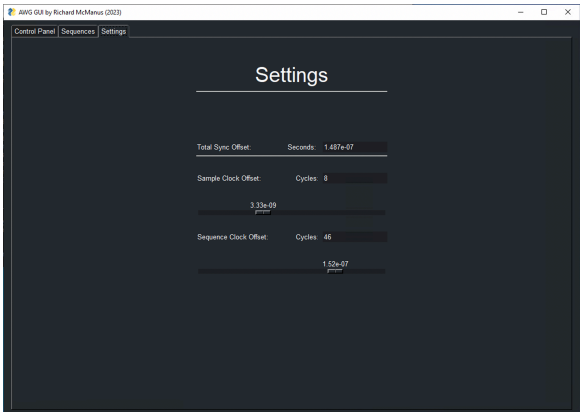

Figure 25. HDAWG GUI Settings Tab

I. Equipment Used

The equipment used in this section is shown in Table 2.

Table 2. Equipment Used to Test Synchronization

Zurich Instruments HDAWG	Arbitrary Waveform Generator	dev8310
Zurich Instruments HDAWG	Arbitrary Waveform Generator	dev8259
Zurich Instruments PQSC	Programmable Quantum System Controller	dev10089
Rigol MSO8204	Oscilloscope	SN# DS8A223400387
BK Precision 9201	DC Power Supply	SN# 449A16132

III. FMC To Card Edge (F2CE) Main Board

A. Background

In addition to the MIPS, the group designed several other chips requiring testing. An 88-pin probe card was chosen to breakout the chip signals for testing. The probe card utilizes an 88-pin card edge connector for input and output (I/O). To interface this card edge connector with the FPGA Mezzanine Card (FMC) connector, a conversion board was required. Additionally, level shifting was required to lower the FPGA's 1.8 V logic to 1.2 V logic for the test chips. Resistor voltage dividers were chosen for this purpose.

Figure 26 shows the design for the second version of the F2CE board that was laid out and manufactured during the Fall semester of 2022. The board was assembled through collaboration with Jon

Cowart in January of 2023 using the equipment provided by the EIH (stencil printer, vacuum pick and place, reflow oven, etc.). Figure 27 shows Jon Cowart using the vacuum pick and place to assemble the board and Figure 28 shows the assembled F2CE V2 board. Figure 29 shows the F2CE V2 board connected via FMC cable to the FPGA and to the card edge connector via 88 manually soldered wires.

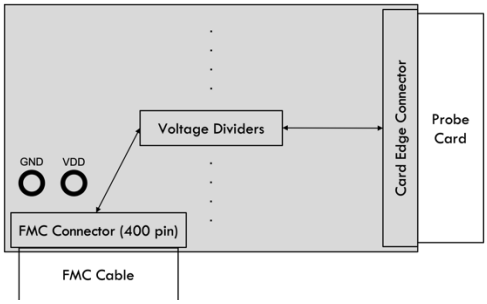

Figure 26. F2CE V2 Board Design

Figure 27. Jon Cowart Using the Vacuum Pick and Place to Assemble the F2CE V2 Board

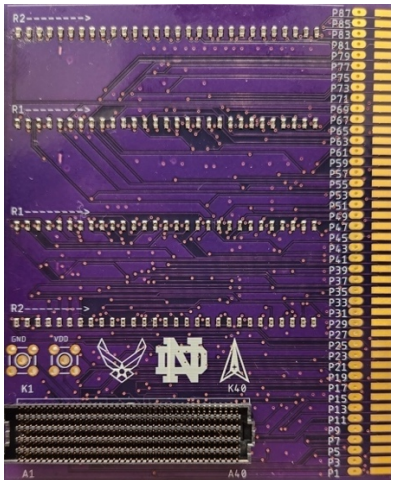

Figure 28. Assembled F2CE V2 Board

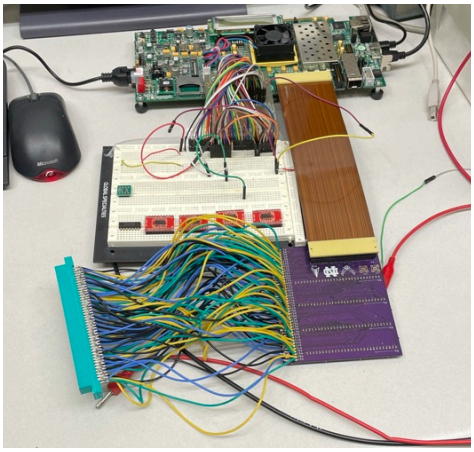

Figure 29. Assembled F2CE V2 Board Connected to FPGA and Card Edge Connector

B. F2CE V3

A design revision was made to the third version of the F2CE board to remove the need for the card edge connector to be connected via 88 manually soldered wires. The new design, shown in Figures 30 and 31, incorporated a main board and a daughter board intended to interface at a right angle using an 88-pin header array.

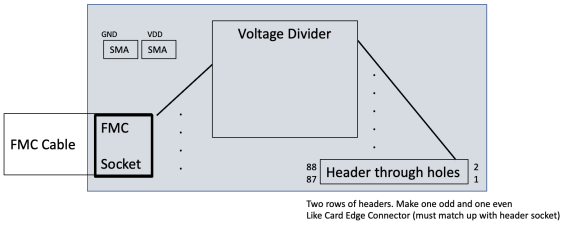

Figure 30. F2CE V3 Main Board Design

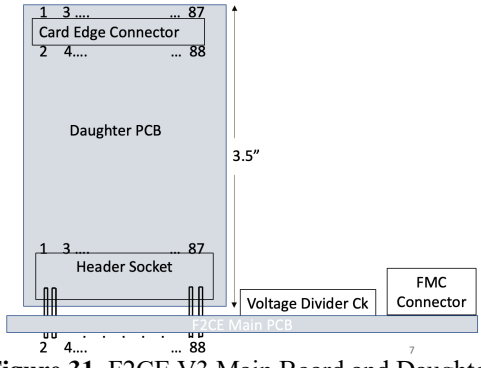

Figure 31. F2CE V3 Main Board and Daughter Board System Assembly

Additionally, the third version of the F2CE board replaced the discrete resistors used in version two with resistor arrays to streamline assembly. The finalized Eagle schematic for the F2CE V3 board is shown in Figure 32 and the corresponding board is shown in Figure 33.

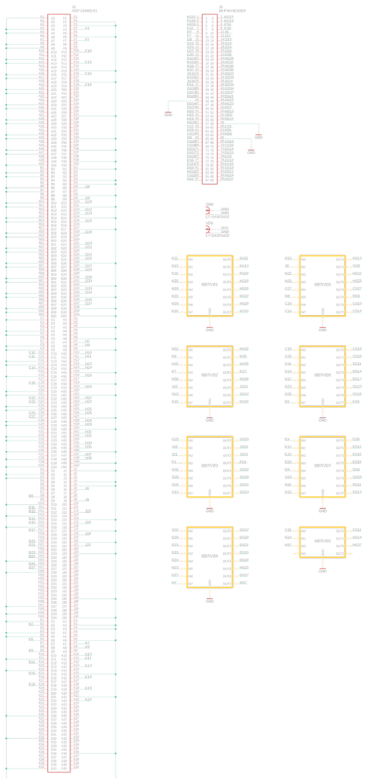

Figure 32. F2CE V3 Eagle Schematic

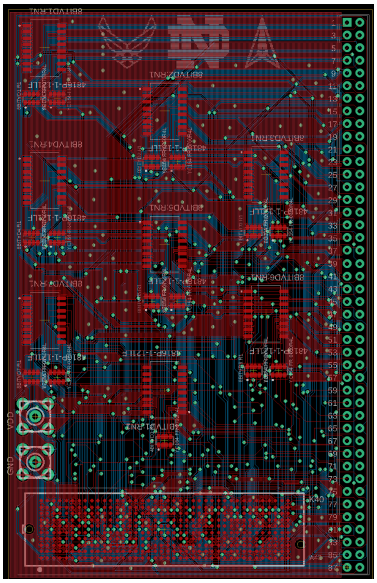

Figure 33. F2CE V3 Eagle Board

C. F2CE V3.1

Soon after completion of the F2CE V3 layout, a second revision was made. It was decided by the group that a voltage regulator should be implemented to ensure a constant 1.2 V would be provided to the DUT regardless of the power supply being used (FPGA or external). The ADP171 adjustable linear regulator was selected for this purpose. The schematic for this voltage regulator is shown in Figure 34 and the equation used to determine the values for R1 and R2 (510 Ω and 360 Ω) is shown in Equation 1.

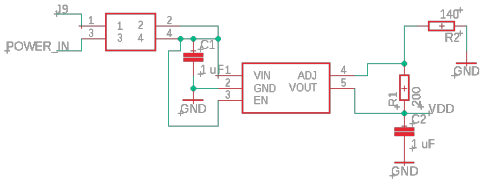

Figure 34. ADP171 Voltage Regulator Eagle Schematic

$$V_{Out} = 0.5\,V(1 + R_1/R_2) + (ADJ_{I-Bias})(R_1)$$

Equation 1. Output Voltage of ADP171 Voltage Regulator

A jumper was included to allow for the selection of power from the FPGA or an external source connected to the subminiature version A (SMA) input labeled “POWER_IN”. The finalized F2CE V3.1 board is shown in Figure 35 and was ordered from Elecrow.

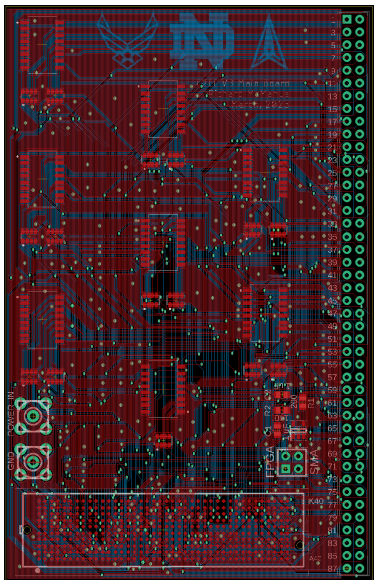

Figure 35. F2CE V3.1 Eagle Board

The board was assembled through collaboration with Ricky Ortiz using the equipment provided by the EIH (stencil printer, vacuum pick and place, reflow oven, etc.). Figure 36 shows Ricky Ortiz applying solder paste to the board. Figure 37 shows the assembled F2CE V3.1 board.

Figure 36. Ricky Ortiz Applying Solder Paste to the F2CE V3.1 Board

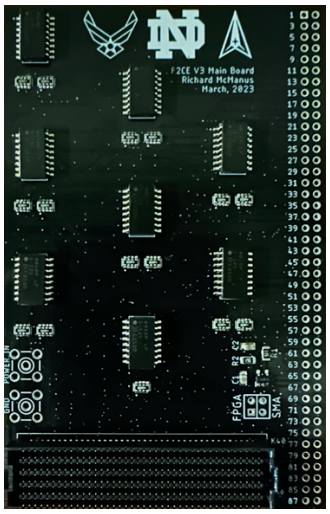

Figure 37. Assembled F2CE V3.1 Board

D. F2CE V4

After completion of the F2CE V3.1 assembly, a third revision was found to be necessary. It was discovered by the group that the Virtex-7 VC707 FPGA required bi-directional signals to be routed to distinct input pins and output pins of the FMC connector to be read from or written to individually. An example of this is shown in Figure 38.

Figure 38. Bi-directional Signal at the FPGA

This discovery required an additional fifteen connections to the FMC connector corresponding to the fifteen bi-directional signals being used. Version four of the F2CE board included these additional fifteen traces. The F2CE V4 board shown in Figure 39 was ordered from Elecrow upon completion.

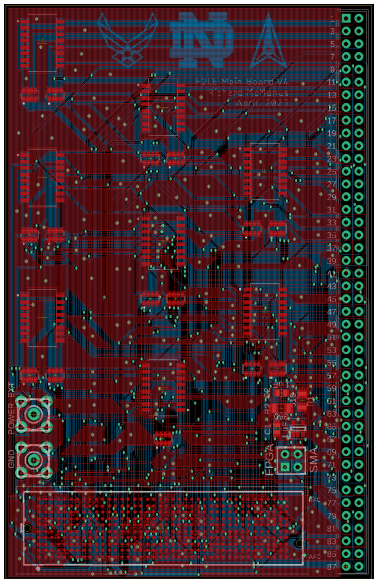

Figure 39. F2CE V4 Eagle Board

IV. F2CE Daughter Board

A. Background

With the two-board design as shown in Figure 31, a daughter board was required to interface the F2CE main board with the 88-pin probe card. AnnahMarie Behn-Link created the initial versions of the Eagle schematic file and board file for the F2CE daughter board. These initial files are shown in Figures 40 and 41.

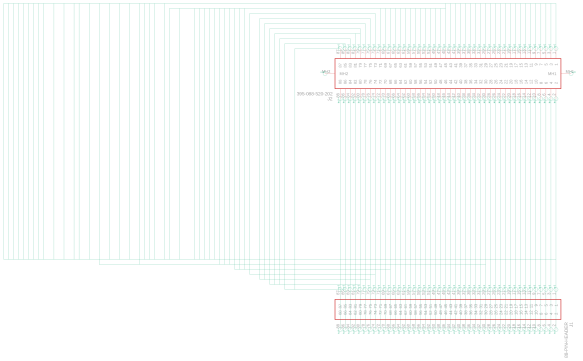

Figure 40. F2CE Daughter Board Schematic

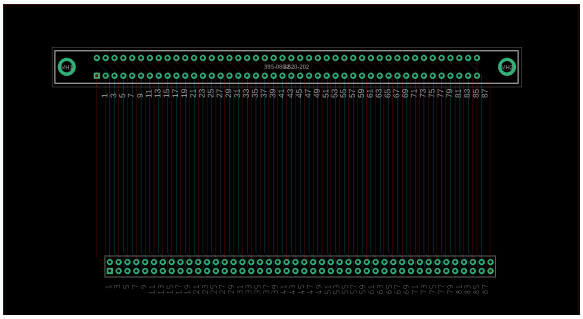

Figure 41. Initial F2CE Daughter Board

B. Final Adjustments

The F2CE Daughter board was finalized before ordering to ensure proper functionality. The footprint of the board was minimized, top and bottom ground pours were added, components were centered, and labels were added. The finalized F2CE daughter board shown in Figure 42 was ordered from Elecrow.

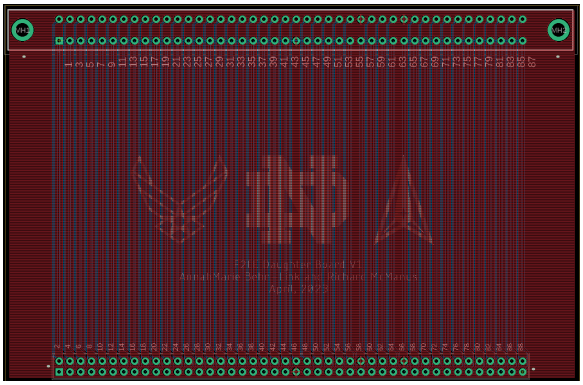

Figure 42. Finalized F2CE Daughter Board

V. Active Level Shifting Board

A. Background

An active level shifting board was required to interface the Virtex-7 VC707 FPGA with an adiabatic MIPS for testing. The 88-pin MIPS requires logic levels of +/-0.7 V while the FPGA utilizes logic levels of 0 V and 1.8 V. Therefore, a level shifting stage shown in the block diagram of Figure 43 was required.

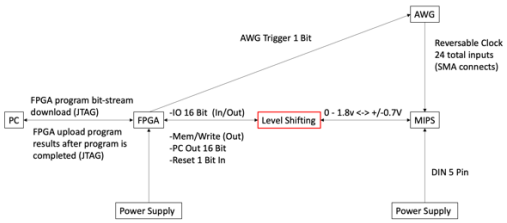

Figure 43. Block Diagram of MIPS Testing (Level Shifting)

During the fall semester of 2022, Dr. Orlov developed an active level shifting circuit shown in Figure 44 utilizing two comparators for each channel and a direction controlling MOSFET. The Eagle schematic shown in Figure 45 was developed to incorporate sixteen copies of the circuit in Figure 44 but an effective method of laying out sixteen identical channels was not discovered.

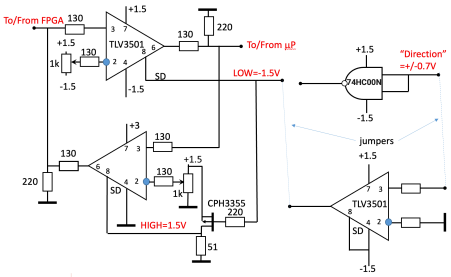

Figure 44. Active Level Shifting Circuit

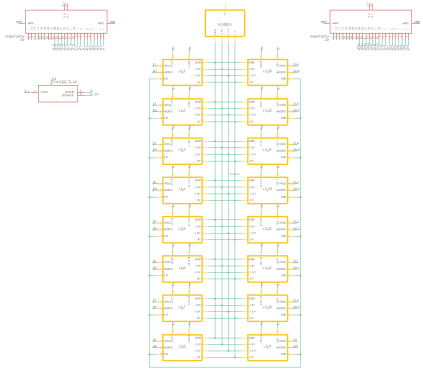

Figure 45. Initial Eagle Schematic for Active Level Shifting Board

B. Initial Layout Attempt - Modules

An initial attempt at laying out the board was made utilizing schematic modules to group the components of each channel. As shown in Figure 46, this would have required the manual placement of each discrete component and was therefore was not a viable option.

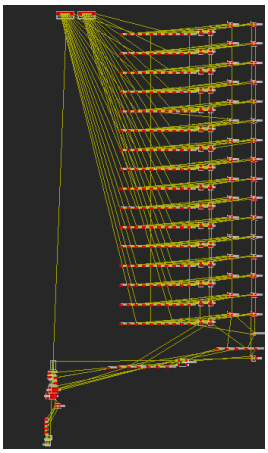

Figure 46. Active Level Shifting Board using Modules

C. Second Layout Attempt – Design Blocks

A second attempt at laying out the board was made upon the discovery of design blocks. Eagle design blocks allow the user to copy and paste already laid out circuitry into a larger board file. The caveat with this method is that a design block cannot be revised after implementation into a larger circuit. In assisting Dr. Orlov, we were successful in laying out an initial version of the sixteen-channel board using sixteen copies of a single-channel design block. The finalized board (LLS_V3_8x2_Inline) shown in Figure 47 was ordered from Elecrow.

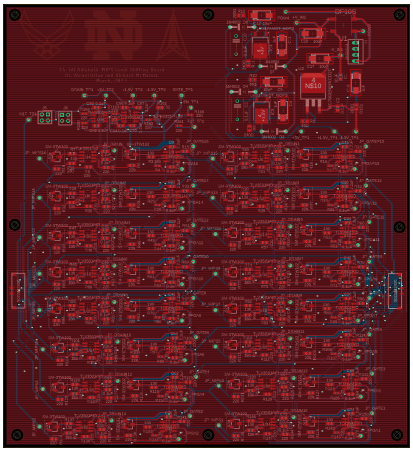

Figure 47. LLS_V3_8x2_Inline Eagle Board

A single channel of the board was assembled through collaboration with Ricky Ortiz using the equipment provided by the EIH (stencil printer, vacuum pick and place, reflow oven, etc.). Figure 48 shows Richard McManus adding components to the board using the vacuum pick and place. Figure 49 shows the assembled LLS_V3_8x2_Inline board.

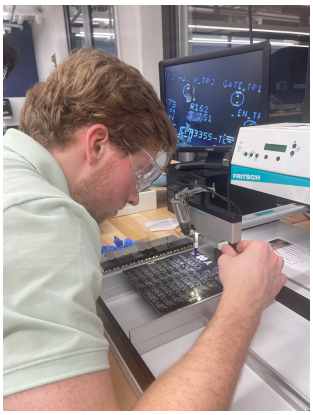

Figure 48. Richard McManus Using the Vacuum Pick and Place to Assemble a Single Channel of the LLS_V3_8x2_Inline Board

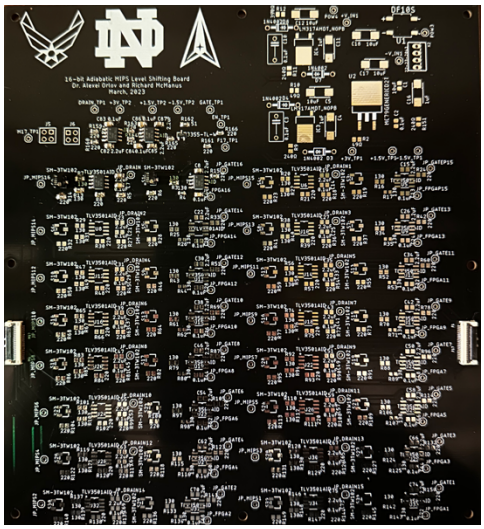

Figure 49. LLS_V3_8x2_Inline Board with One Channel Assembled

D. Revision to Incorporate Bi-directional Design

As described in section III.A, the group discovered a revision was necessary as the Virtex-7 VC707 required bi-directional signals to be routed to distinct input pins and output pins of the FMC connector. Dr. Orlov implemented this revision into the design of the fourth version of the active level shifting board through separate read and write

circuits and an additional 24-pin ribbon cable connector. After final adjustments were made, the board shown in Figure 50 (LLS_V4_8x2_Inline) was ordered from Elecrow.

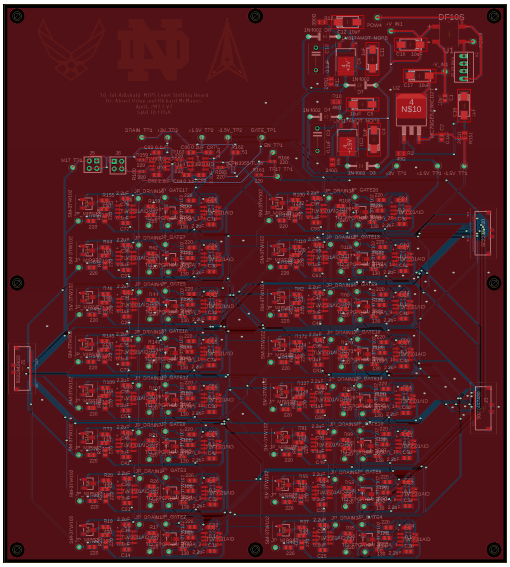

Figure 50. LLS_V4_8x2_Inline Eagle Board

VI. Next Steps

All three boards have been received by Elecrow and assembled by Ricky Ortiz and AnnahMarie Behn-Link. With assembly complete, the functionality of the boards will need to be tested and any necessary revisions will need to be made. Upon successful testing, the boards will be ready for implementation. These boards will be used to interface a DUT with the FPGA while measurements of heat dissipation are made to observe the efficiency of the quasi-adiabatic reversible logic chips produced by the group.

VII. Attachments

1. Folder containing the files necessary to run the GUI shown in Figures 23-25 (*HDAWGs)*

2. Folder containing multiple iterations of both the F2CE Main and Daughter board (*FMC-to-Card-Edge-Level-Shifter)*

3. Folder containing multiple iterations of the Active Level Shifting Board (*16-Bit-Active-Level-Shifting-Board)*

VIII. REFERENCES

ADP171 Datasheet, https://www.mouser.com/datasheet/2/609/ADP170_171-3121698.pdf

Elecrow, https://www.elecrow.com

MatPlotLib Library, https://matplotlib.org

PySimpleGui Library, https://www.pysimplegui.org/en/latest/

Zurich HDAWG User Manual, https://docs.zhinst.com/pdf/ziHDAWG_UserManual.pdf

Zurich LabOne MultiDeviceSyncModule, https://docs.zhinst.com/labone_api/python/autosummary/zhinst.core.MultiDeviceSyncModule.html

Zurich LabOne Programming Manual, https://docs.zhinst.com/pdf/LabOneProgrammingManual.pdf

Zurich PQSC User Manual, https://docs.zhinst.com/pqsc_user_manual/tutorial_synchronization_hdawg.html