Arbitrary Waveform Generator Python Programming and Peltier Module Test Environment

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

***Abstract* – This report describes the progress made during the Fall 2021 semester on two separate projects in Dr. Snyder's Research Group and focuses specifically on my contributions. The first project refers to a python script used to efficiently generate test waveforms with an arbitrary waveform generator (AWG) and the second refers to a test environment comparing the sensitivities of two distinct methods for measuring temperature.**

I. INTRODUCTION

This semester I had the opportunity to join Dr. Snider's research group and contribute to their efforts. My contributions consisted of researching and developing API implementation for Zurich AWGs and assembling electronic circuits to test for effectiveness. In the following sections, I will discuss the processes for these projects and the results obtained.

II. Arbitrary Waveform Generator Python Programming

A. Background

This project was designed to build upon John Bannon's work last semester to generate trapezoidal waveforms from the Zurich AWGs in the lab. Previously, these waveforms were generated with hardcoded values by cutting and joining preset waveform types (i.e., rectangular and ramp waves). This method can be seen in figure 1. The goal of this project was to develop a process to load in and generate waveforms from a table (e.g., .xlsx or .csv file). This would make generating waveforms simpler and more capable of quick adjustments.

```
//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 (6400 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 1. Previous Method of Waveform Generation

B. Familiarization with LabOne APIs

Initially, I attempted to use LabOne's MATLAB API for this project but encountered several issues. Dr. Jorge Medina, a Zurich Technical Sales Associate, suggested using LabOne's Python API instead of MATLAB.

Once using python, my initial efforts focused on running the LabOne example script, *example_awg.py*. This file used hardcoded values to generate waves and output them using the AWG as shown in figures 2 and 3.

```
# Define an array of values that are used to write values for wave w0 to a CSV file in the
# module's data directory
waveform_0 = -1.0 * np.blackman(AWG_N)

# Define an array of values that are used to generate wave w2
waveform_2 = np.sin(np.linspace(0, 2 * np.pi, 96))
```

Figure 2. *example_awg.py* Waveform Generation

```
awg_program = textwrap.dedent(
    """\
    const AWG_N = _c1_;
    wave w0 = "wave0";
    wave w1 = gauss(AWG_N, AWG_N/2, AWG_N/20);
    wave w2 = vect(_w2_);
    wave w3 = zeros(AWG_N);
    while(getUserReg(0) == 0);
    setTrigger(1);
    setTrigger(0);
    playWave(w0);
    playWave(w1);
    playWave(w2);
    playWave(w3);
    """
)
```

Figure 3. *example_awg.py* Multiline String

Figure 2 shows two waves being generated using hardcoded values and standard mathematical

functions. Figure 3 shows the multiline format necessary to successfully define and generate waveforms using the LabOne Servers. LabOne servers use a proprietary programming language so commands must be contained within multiline string to be executed properly.

To successfully compile and run the example python file, three steps needed to be completed. First, imported libraries needed to be pip installed. Second, the line of code in figure 4 needed to be added to update the value of the node.

```
daq.setInt(f"/{device}/awgs/0/userregs/0", 1)
```

Figure 4. Updated Node Value

Third, a while loop was needed to cause the waveforms to be generated indefinitely. This while loop can be seen in figure 5.

```
while True:
    daq.setInt(f"/{device}/awgs/0/single", 1)
    daq.setInt(f"/{device}/awgs/0/enable", 1)
```

Figure 5. While Loop for Indefinite Wave Generation

After these changes were made, the program ran successfully and generated the four predefined waveforms as expected. This output can be seen in Figure 6.

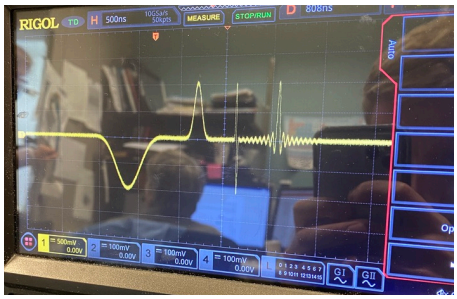

Figure 6. *example_awg.py* Successful Output

C. *Familiarization with LabOne APIs*

After successfully running *example_awg.py,* the python script *ReadandCreateWave,py* was created using example_awg.py as a foundation. This file makes use of the csv and tkinter libraries to read in waveforms from .csv files as shown in figure 7.

```
import os
import time
import textwrap
import numpy as np
import zhinst.utils
from tkinter import filedialog as fd
filename = fd.askopenfilename()
import csv
with open(filename,"r") as file_name:
    array = np.loadtxt(file_name, delimiter=",")
```

Figure 7. *ReadandCreateWave.py* Reading File

As shown in figure 7, tkinter is used to prompt the user for a file and the csv library is used to load that file into an array.

Next, as shown in figure 8, the user is prompted for an integer to specify a sampling rate in the form:

$$Sampling\ Rate = \frac{100\ MHz}{2^n} \qquad \textbf{Eqn. 1}$$

In this form, *n* is user-defined and restricted to integer values between 0 and 12. Figure 8 also shows numerous other settings for four output channels that turn on the outputs, set amplitudes, etc.

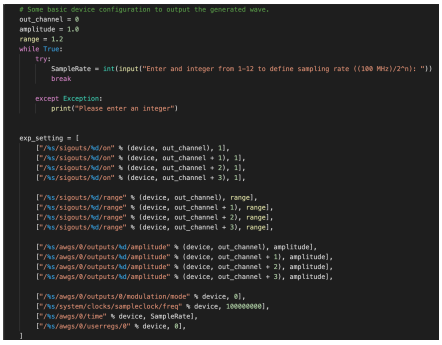

Figure 8. Sampling Rate and AWG Settings

After these settings are specified, the program prompts the user for the column index of the first waveform to be generated. Then, the original array from the input file is broken down into separate vectors, one for each waveform, as shown in Figure 9.

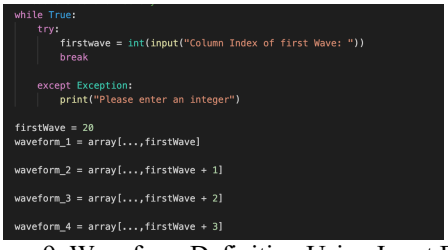

Figure 9. Waveform Definition Using Input File

Lastly, the program creates temporary .csv files for each of these arrays that are then used to generate the waveforms within the LabOne Server. This process can be seen in Figures 10 and 11.

```
# Save waveform data to CSV
csv_file = os.path.join(wave_dir, "wave1.csv")
np.savetxt(csv_file, waveform_1)
csv_file = os.path.join(wave_dir, "wave2.csv")
np.savetxt(csv_file, waveform_2)
csv_file = os.path.join(wave_dir, "wave3.csv")
np.savetxt(csv_file, waveform_3)
csv_file = os.path.join(wave_dir, "wave4.csv")
np.savetxt(csv_file, waveform_4)
```

Figure 10. Temporary .csv Files

```
awg_program = textwrap.dedent(
    """\
    const AWG_N = _c1_;
    wave w1 = "wave1";
    wave w2 = "wave2";
    wave w3 = "wave3";
    wave w4 = "wave4";
    while(getUserReg(0) == 0);
    setTrigger(1);
    setTrigger(0);
    playWave(1,w1, 2, w2, 3, w3, 4 , w4);
    """
)
```

Figure 11. Multiline String Containing .csv Files

After completely writing this program and making it through the inevitable debugging process, the AWG successfully output four waveforms as defined by an input file. This output is visible in the form of pulses on the oscilloscope as shown in figure 12.

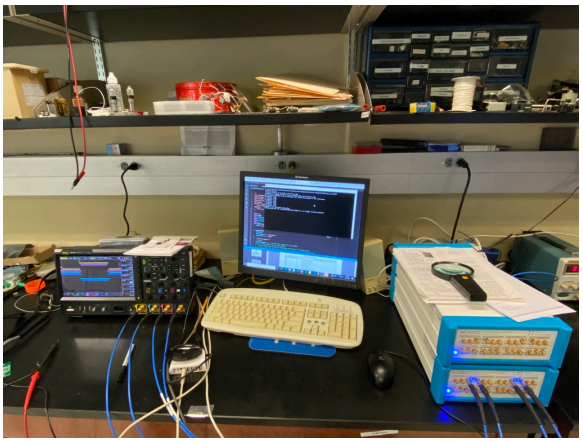

Figure 12. Successful Pulse Generation

To form the desired trapezoidal waveforms, I used Origin and interpolated the waveforms provided by Rene Celis-Cordova using 500 points. The output of these waveforms using ReadandCreateWave.py is shown in Figure 13.

Figure 13. Successful Trapezoidal Waveform Generation

IV. Next Steps

With these waveforms being successfully generated, I began working on the second project of the semester. However, I plan to implement an algorithm to automatically interpolate waveform data allowing the user to select any frequency they desire rather than being limited to the 13 default sampling rates. In addition, I plan to continue Jack Bannan's efforts to sync the two AWGs together making sixteen output channels available.

III. Peltier Module Test Environment

A. Background

The goal of this project was to compare the sensitivity of Peltier modules and thermocouples when measuring temperature. A surface mount resistor powered by an operational amplifier being used as a current source would supply the heat for this test environment.

B. Design and Fabrication

My work on this project began with arranging the circuit boards on an aluminum plate and marking where to drill the holes. Then, Dr. Orlov drilled and tapped holes in the aluminum plate and place the boards on stands as shown in figure 14.

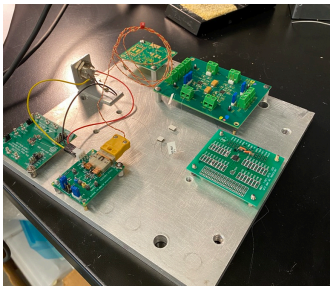

Figure 14. Arranged Boards on Stands

Next, a 5 pin DIN socket was fixed to the aluminum plate. Positive, negative, and ground wires were

soldered to the power supply pins and ran in parallel to the thermocouple amplifier and voltage regulator. This can be seen in figures 15 and 16.

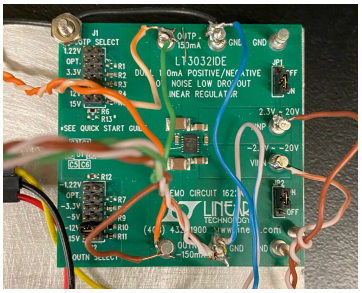

Figure 15. Voltage Regulator

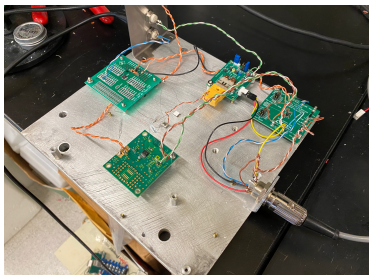

Figure 16. Completed Test Environment

Next, the operational amplifier circuit to be used as a current source was assembled based on Dr. Orlov's circuit diagram shown in figure 17.

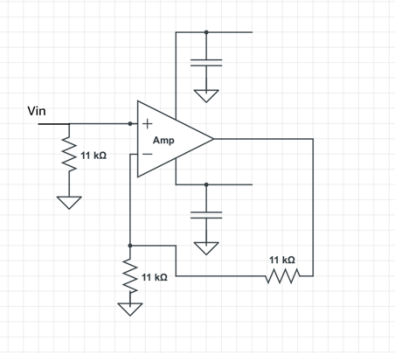

Figure 17. Operation Amplifier Circuit Diagram

The load in this circuit is the 11kΩ in the feedback loop. This design was made to control the power dissipated in the load which can be derived as:

$$Power = \frac{Vin^2}{11\ k\Omega} \qquad \textbf{Eqn. 2}$$

Assembly of this circuit required the soldering of surface mount capacitors and resistors. The amplifier was powered by the voltage regulator and the input was supplied by an external power supply. The fully assembled circuit can be seen in Figure 18.

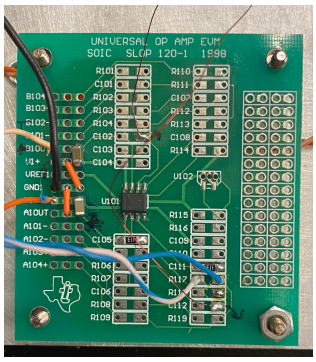

Figure 18. Assembled Operational Amplifier Circuit

The 11 kΩ resistor being used as a heat source was then placed in the feedback loop using fine manganese wire. This resistor was then mounted to the top surface of one of the Peltier modules using crazy glue. The Peltier modules were mounted to the surface of the aluminum plate and the thermocouple junction was glued to the top surface of the same Peltier module as the resistor. This configuration can be seen in figure 19.

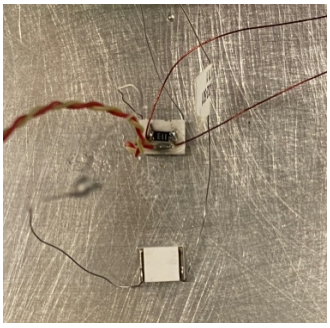

Figure 19. Peltier Module, Thermocouple, and Resistor Configuration

C. Results

Initial data from this project showed the Peltier modules to be significantly more sensitive than the thermocouple. However, the Peltier modules also experienced a longer settling time than the thermocouple based on their increased mass.

REFERENCES

https://docs.zhinst.com/pdf/LabOneProgrammingManual.pdf

https://docs.zhinst.com/pdf/ziHDAWG_UserManual.pdf