Integration and Automation Software for Testing of RIME and REASON Instruments

Zachary Luppen  
*University of Iowa*

Craig Kletzing  
*University of Iowa*

Donald Kirchner  
*University of Iowa*

Follow this and additional works at: [https://ir.uiowa.edu/honors_theses](https://ir.uiowa.edu/honors_theses)

Part of the [Space Vehicles Commons](https://ir.uiowa.edu/honors_theses)

Copyright © 2018 Zachary Luppen

Hosted by Iowa Research Online. For more information please contact: lib-ir@uiowa.edu.
INTEGRATION AND AUTOMATION SOFTWARE FOR TESTING OF RIME AND REASON INSTRUMENTS

by

Zachary Luppen
Craig Kletzing
Donald Kirchner

A thesis submitted in partial fulfillment of the requirements for graduation with Honors in the Astronomy

Craig Kletzing
Thesis Mentor

Fall 2018

All requirements for graduation with Honors in the Astronomy have been completed.

Cornelia Lang
Astronomy Honors Advisor

This honors thesis is available at Iowa Research Online: https://ir.uiowa.edu/honors_theses/
Integration and Automation Software for Testing of RIME and REASON Instruments

Luppen, Z., Kletzing, C., Kirchner, D.L.

University of Iowa, Iowa City, IA

Abstract

In collaboration with the NASA Jet Propulsion Laboratory (JPL), the University of Iowa Radio and Plasma Wave Group is building the RIME and REASON radar instruments for the JUICE mission and the Europa Clipper mission, respectively. Our team is in charge of designing and fabricating the transmitter circuit boards consisting of the power supplies, the high-frequency amplifier and the signal conditioner for both missions. I have been working with the group for the past year and a half, and this summer I interned at JPL developing software for the Europa Clipper mission. For the 2018-2019 year, I am continuing this work with the Iowa team to help develop the flight boards for the mission and contribute to the testing phase. I developed several programs and graphical user interfaces (GUIs) to interact with test equipment during my internship. These programs were originally specific to the JPL side of the testing phase, but are extremely useful on the Iowa side as well. Using the code I’ve already developed, I am working with the Iowa team to develop these programs on this end and create further programmatic implementation to help with the testing process. This code will allow the extensive testing phase to proceed with more automation and in a more comprehensive manner.

Introduction

The Europa Clipper spacecraft is an upcoming NASA mission to the moons of Jupiter slated for launch in the early 2020s. It will augment the Jupiter Icy Moons Explorer (JUICE) mission and focus closely on the Galilean moon of Europa. Europa is an icy moon with a very youthful water-ice crust thought to conceal a water ocean beneath. This under-ice ocean may have the ability to harbor life, making it a high-profile target to study. The REASON instrument on this spacecraft will attempt to study this ocean. It is a dual-frequency subsurface radar instrument that will be able to peer down into the icy crust of the moon and characterize its environment. It will use VHF and HF bands to investigate the ice shell, the subsurface ocean, plumes, tides and potential landing sites for a future Europa lander mission. Figure 1 shows a block diagram of the REASON instrument. It is similar in scope to the Mars Express MARSIS instrument, which has been able to examine the surface of Mars as well as the material immediately below. The instrument is being built by the NASA Jet Propulsion Laboratory (JPL), in collaboration with the University of Iowa, the University of Texas, Stellar Scientific, as well as other institutions.

The RIME instrument for JUICE and the REASON instrument for the Europa Clipper are similar in both their design and mission goals, except for mainly focusing on separate Jovian moons. Engineering models of both instruments’ software have been built at the University of Iowa, which I’ve
been involved with since May 2016. We are now in the thermal testing stage of the circuit boards, as well as running test scripts to examine their functionality. The goal of my internship at JPL was to continue this testing process and prepare for the fabrication of the flight boards. I wrote GUIs to interact with our test equipment so that we can operate them from a single computer, as well as write programs that perform numerous systems measurements. The work I did automates the testing process and will make the testing phase for the flight boards (once they are at JPL) go much more quickly.

After returning to Iowa, I realized that this code would be tremendously useful for the Iowa team working on the mission. It could be used on JPL and Iowa sides to speed up the testing phase and to use similar methods throughout the testing process. To fund it, I applied to the Iowa Space Grant Consortium’s (ISGC) Hands-On grant, which will fund up to $5,000 for NASA-related research projects. The grant was awarded to me in September 2018 with funding beginning October 2018 and going through May 2019. Now that the Fall 2018 semester is ending, I am about halfway through the duration of this grant-funded project.

In this document, I will discuss both the work of my internship and fall semester portion funded by the ISGC. This will give clarity to several of the programs written and explain the reasons for their creation.

Internship at the Jet Propulsion Laboratory

This summer, I took part in an internship at the NASA Jet Propulsion Laboratory with mentor Tushar Thrivikraman. Dr. Thrivikraman is one of the main JPL contacts for the University of Iowa team working on RIME and REASON, and he visits the university periodically to check on our progress. Before proceeding with work, Tushar and I summarized the main goals of my experience during the internship. It was titled “Integration, Testing and Automation for the REASON Instrument.” The overarching goal was to

Figure 1. Block diagram of the REASON Power Supply Unit (PSU) and Power Amplifier Unit (PAU).
understand the system engineering process of developing a radar system, in this instance the Europa Clipper REASON instrument. Additionally, I was expected to gain an understanding of how to use electronic test equipment, develop the software necessary to communicate with it, and know the integration and testing needs for the REASON RF subsystem.

Test Equipment and Setup

Performing tests, calibration and analysis of our circuit boards requires several pieces of equipment. For my work, I used an MSO5 Tektronix Oscilloscope [1], a Keysight 33600A Arbitrary Waveform Generator (AWG) [2], an Agilent N6705A DC Power Analyzer [3], an Agilent 34972A Data Acquisition Unit [4] and a computer. Figure 2 shows a pictorial representation of this setup. A Raspberry Pi with a touch screen was eventually used, to replace the laptop entirely. All this equipment was connected to a network set up at the bench to allow communication with the laptop/pi. The software written to operate each of the instruments from the laptop was written in Python 3.6 and utilized the TKinter GUI package to generate programs the user could operate.

Several programs used to interact with test equipment were already written prior to starting the internship. These programs are highly functional and easily facilitate data recording from an oscilloscope or operation of a waveform generator. These programs make use of the internal functions of the equipment and allow the operator to perform almost any operation that could be done via the buttons and knobs on the control panels. While many apps were created, and the essential code needed to connect to each instrument has been written, more work was necessary to optimize the usage of the programs, fix bugs, and create further implementation to make these applications as versatile and useful as possible during testing.

Major Implementations and Programs

The work performed for REASON integration and testing automation culminated in the creation of several programming accomplishments. Each of the main tasks worked on, while different in scope, directly benefitted a user’s ability to perform the necessary testing. Voltage waveform smoothing was altered since the code to smooth waveforms was not adequate for our purposes. A GUI was created to perform power measurements over time. Another program was written to interact with the DC Power Analyzer at the bench. An update was applied to the existing arbitrary waveform generator code to allow it to recreate waveforms that have already been generated and saved separately. And lastly, to perform long-duration tests of these programs, a raspberry pi with a touch screen was hooked up to the equipment and...
the programs run. Each of these accomplishments are further discussed below.

a. Voltage Waveform Smoothing

One of the necessary functions of our testing is to monitor the power input and output of the circuit boards. It’s a standard measurement that, besides measuring the power of the system, can help determine if the right amount of power is being applied. Unfortunately, there are no peak power meters at less than 10MHz readily available, and so a new method of power monitoring needed to be used. It was decided to have the laptop read in a voltage vs. time waveform, determine the number of zero crossings, take the maximum of the absolute value voltage waveform between each crossing, determine the \( V_{\text{rms}} \) from \( V_{\text{peak}} \), and then convert to a power vs. time waveform.

Arbitrary Waveform Generator, zoomed up close to a zero crossing. As this image details, a simple sinusoid is noisy enough to create many zero crossings where only one should be. Figure 4 shows a power waveform converted from this sinusoid and very well illustrates the problem we have. What should be a smooth line is instead a jumble of peaks and troughs, since there are pockets of many zero crossings. The smoothing methods already implemented into the waveform class – NumPy’s built-in hamming, hanning, gaussian and Blackman filters – do not adequately smooth the waveform. The power measurements, since they’re utilizing all the zero crossings, therefore do not display useful information.

The key problem with this method of making measurements is that the calculations make use of every zero-crossing of the waveform. For a noisy waveform, there can be many extraneous crossings. Figure 3 shows a sinusoid generated by the

Figure 3. A single zero-crossing of an arbitrary sinusoid, zoomed in to show detail.

Figure 4. A power waveform based off of a hanning-smoothed voltage waveform.

Figure 5. A representation of the Savitzky-Golay method. Source: Wikimedia Commons.
To solve this issue, a Savitzky-Golay filter was implemented. This method was developed by Abraham Savitzky and Marcel J. E. Golay in the 1960s. [5] This smoothing method performs an \( n \)th polynomial least squares fitting to the data. A window of \( x \) number of data points is specified for a data set, and then for each point and the \( x \) number of points around it, a polynomial of the specified \( n \) degree is fit, and a smoothed waveform is produced. Figure 5 shows a demonstration of this method. [6] Using a 3rd degree polynomial fit, the original noisy voltage waveform is reduced to a greatly smoothed one. An example of this implementation is shown in Figure 6, using the same waveform as in Figure 3.

We can then use the smoothed voltage waveform to count zero crossings and convert to a power over time waveform. Figure 7 shows the original voltage waveform in blue, the old smoothing method in orange and the Savitzky-Golay smoothing method in green. We take this calculation a step further by averaging the values to get a single number: the average power for the given time interval. These calculations are used in the next big task, the oscilloscope power monitor GUI.

**b. Oscilloscope Power Monitor GUI**

With the power calculations working properly, the next step was to create a GUI that can monitor the power over time. This is both a very important aspect of the system to measure and can be entirely automated, so the user doesn’t have to do the measuring themselves. A screen capture of the GUI is shown as Figure 8. The GUI is written in TKinter and provides a live-updating interface to monitor both input and output power. In addition to the plot, a .CSV file is also generated that stores the data line by line.
This method of appending to the .CSV file is very necessary as it prevents the file from crashing and not saving if communications is lost with the oscilloscope. That way, a user will have the data regardless if a crash occurred or the program was force-closed. This application is limited slightly in that a single .CSV file can only support 1,048,576 rows, and further code can be written to close the initial .CSV file and open a new one repeatedly.

c. DC Power Analyzer GUI

One of the pieces of test equipment, the N6705A DC Power Analyzer, lacked a GUI to operate it. In a situation where the test equipment cannot be readily accessed, a GUI solves that problem. The program for this piece of equipment allows the user to connect to any N6705A provided they know the IP address. A screen capture of this GUI is shown as Figure 9. Once connected, the user can then operate any of the 4 channels from the GUI, adjusting their voltages or currents and turning them on or off. Additionally, the GUI features a CSV writing feature that records the voltage and current from each channel at a given interval. This way, a user can monitor the channels and have a log of each’s parameters.

d. AWG Configuration Reloading

The arbitrary waveform generator is an extremely useful tool for making practice waveforms to send through our circuit boards. One of the most common waveform types to test is called a chirp, a sinusoidal waveform that turns on and off at a set interval. An example of a single chirp is shown in Figure 10. One of the GUIs already written, the AWG_Configure.py script, has the capability of loading various chirp waveforms into the oscilloscope. Once in the scope, this waveform can be saved to a laptop (or any device connected to the network) and be read into python. For the

![Figure 10. A single chirp waveform.](image)

![Figure 11. A screen capture showing the AWG Configuration Tool, and the reload tool.](image)
AWG configuration reload script, the idea is that once we have the waveform, saved in a .pkl file format, we want to reload it back into the AWG and play it back again without using the AWG Configure.py script to load it from scratch. This implementation allows the user to repeat some specific waveform (perhaps with a specific frequency or duty cycle) that they had looked at previously and reuse it. This functionality has been added to the original script and can be used via a button pressed in the GUI interface of the AWGConfigure.py script. Figure 11 shows a screen capture of the AWG Configuration Tool and demonstrates how a .pkl file is loaded into the AWG.

e. Raspberry Pi Configuration

One of the issues that came up during my time writing scripts was how to perform long-duration testing. In several cases, we simply wanted to run the applications for several hours or even a day and ensure that they could continue operating over that time. Since I required my laptop off-site to continue working and maintain contact, we needed to come up with a cheap alternative. Our solution was to use a Raspberry Pi. The Pi, which only costs about $35, is a small computer that can be programmed by the user to practically anything they need. After connecting the Pi to use the instrument network and make minor script modifications, we were then able to run our GUIs from the Pi. Because the Pi’s processor is not as good as that in a brand-new laptop, it runs slower. However, by limiting the amount of data sent to the Pi, we can speed up the processing and make the performance comparable to that of a laptop. A photograph of the Raspberry Pi operating the Oscilloscope Power GUI is shown as Figure 13. We also added a touch screen to the Pi, so that the user can control the instruments as one would use an application on a phone or tablet. [7] It’s a very simple way of operating the test equipment, and something that we hope will make the testing process for the REASON instrument proceed much more quickly.

Transitioning the Code to Iowa

I returned to Iowa in early August, fully aware that this code could be implemented on our side of the Europa Clipper mission as well as the Jupiter Icy Moons Explorer mission. Eager to pursue this work, I quickly applied for the Hands-On Grant from the ISGC. I am working under Dr. Donald Kirchner for this project, which is a direct continuation of my work at JPL. There are two goals for the work under this grant. The first is to redevelop programs I produced at JPL for the University of Iowa side, as much as possible without violating the sharing rules. Since this is just implementation of the same programs, it is not discussed in this paper. The second goal is to write new programs that can help in this process and work as programming support of the Plasma Wave group members.

Figure 12. An image of the oscilloscope power monitor GUI (The wavy structure on the screen is due to aliasing).
currently working on the RIME and the REASON missions. This work is now halfway through completion and will proceed until May when the funding period ends.

It is important to note that not every program I developed while interning at JPL can be added immediately to the programs at Iowa. Due to sharing regulations, all of the work I performed is the property of Caltech-JPL and can only be handed over as a complete entity. So while I implemented the Savitzky-Golay filter into the voltage waveform smoothing function, I am not able to use it at Iowa until my JPL mentor Dr. Thrivikraman adds it to the most updated version of the smoothing and provides it to Iowa. The file reading implementation of the AWG_Configure.py script faces a similar situation.

While there are some limitations to this development, there are still many different aspects that I have been able to develop while back at Iowa. Software used to generate bills of materials (BOMs) needed to be implemented on the Iowa side. A GUI needed to be created for another power supply model, as one did not previously exist. [8]

a. BOM-Generating Software

My most laborious task for this grant has been the integration of the software used to generate bills of materials. Since about March 2018, the University of Iowa has been relying on Dr. Thrivikraman to generate BOMs, label sheets and other CSV files used on the missions. This was very time-consuming, as some of these files could go days without being produced. To correct this, I was tasked to work with Dr. Thrivikraman and implement the BOM-generation software at Iowa.

The BOM-generating software works thusly. One of the electrical engineers uses Altium to output a .CSV document of the components on a circuit board. The document lists each part’s name, quantity, reference designator(s), footprint, description, layer and extra comments. From this, the parts are read line-by-line into python code and cross-referenced with a database of parts already created. If the part is found, then the code marks it successful and moves to the next entry. If the part is not found in the database, then the part is flagged as invalid. These parts could’ve come back invalid because of a name error, because they’re to be included in the list but not installed immediately or because that part does not exist in the database. This last reason came up a lot, because despite a massive database of parts there were still a

![Figure 13. A screen-capture of the BOM Generation tool.](image)
couple dozen missing. Once the software was integrated and working on our computers, I then had to run several .CSV files containing the lists of parts for our circuit boards to see which parts weren’t in the database. This was extremely effective and helped knock down the turn-around time of these documents from sometimes days to about an hour. Figure 13 shows a screen capture of the BOM-generation software.

One of the simple but time-consuming issues behind the BOM-generation is fixing certain values in the files. From an engineering standpoint, the terms “RESISTOR” and “RESTHK-FLM” are interchangeable for certain parts. Not for this code. Each cell of the .CSV file read in is assumed to be a single string. Spaces cannot be included in these strings and will output errors if run through the software and the database cannot find them. For example, the term “RESTHK FLM” is incompatible with the code. As these files are made without exact attention to this detail, it is my responsibility to correct these files and make sure they are detected by the database. A BOM without these errors can take less than 5 minutes, but files with errors can slow down the turn-around time. While it would be conducive to have the code check for spaces, this has been attempted and does not appear compatible with the code.

With this BOM-generation software, we now have the capability again of generating our documents. Being without it was a huge waste of time and productivity, and certainly caused delays between March 2018 and September 2018. Now that this code is implemented, we can produce our label sheets and BOMs much more quickly and focus on more complex parts of the mission. Additionally, it is an aspect of the mission of which I have the sole responsibility. I oversee this code and the BOM-generation, as well as an authority on the programming of it in general.

b. E36313A Power Supply GUI

The DC supply GUI that I wrote at JPL forms a basis in tKinter for writing GUIs to interact with test equipment. The code used to generate the GUI in which the program operates does not itself have issues with sharing regulations. In the Department of Physics & Astronomy, we make use of the E36313-A DC Power Supply by Keysight Technologies in multiple spacecraft lab facilities. This supply, a three-channel power supply, is compatible with the DC Supply class written by Tushar. Reanalyzing the DC Power Supply GUI written for the N6705A supply, it was...
apparent that the only work necessary to modify this code for use with the E36313-A was remove the fourth channel from the GUI and the code that interacts with it. Once this change was made, the program now operates nominally. Additionally, a day-long test has been performed to determine that the code does not crash. Figure 15 shows a screen capture of the newly created GUI for the E36313A supply. [9],[10]

c. Configure_AWG Implementations

While I cannot use the form of the waveform generator code that I worked on while an intern at JPL, I can work with the one Dr. Thrivikraman has provided to university and modify it as needed. We generally use the same model at Iowa as at JPL. An engineer on the Iowa team, Bill Schintler, uses this program to operate the waveform generator for his testing purposes with our missions. With Bill, I have solved several errors and implementations needed to allow him to test. These have stemmed from making a different waveform generator model compatible with the class, to implementing further square wave functionality to creating new waveforms for Bill to generate and test with. These implementations are relatively small to other tasks I’ve been working on and are usually straightforward. They have made our test procedures slightly more robust and provide more in-depth analysis to evaluate circuit board performance.

Conclusion

The testing stage of any spacecraft instrument is an enormous task. Since it is going into space, as part of a costly mission, it is very important that the testing be complete and extremely thorough. The process can be long and arduous, and creating anything that can speed up this process is always appreciated. The code that I have written and implemented to interface with the test equipment for the RIME and REASON missions will help expedite this process. A user can now work with power waveforms and perform power monitoring, operate the DC Power Analyzer via a laptop or a Raspberry Pi, and perform further arbitrary waveform generator functions.

Acknowledgements

I would like to acknowledge all of the people who helped to make my internship and work with the Iowa team a fantastic research experience, and one that I plan to repeat next year. Donald Kirchner, Dick Beals, Robert Johnson, Richard Dvorsky, Jeff Dolan, Scott Bounds, Tushar Thrivikraman, Harry Figueroa, Mauricio Sanchez Barbetty, Duane Howard, Jordan Tanabe, Brian Custodero, James Hoffman, Kim Orr, Jessica Snyder, Ed Friesema and the JPL Education Office.
References