ARAFE Build and Test Manual

Brian Clark [*1] and Patrick Allison [†1]

[1]Department of Physics & Center for Cosmology and Astroparticle Physics (CCAPP), The Ohio State University

June 24, 2017

Abstract

In this document we will detail the procuedure for testing and debugging the ARA Advanced Front End (ARAFE). This includes the Power and Control (PC) boards and their associated DC-to-DC converters, as well as the Radio Frequency (RF) boards. We will describe how to test them as they are built to ensure they are functioning properly, give example oscilloscope and network analyzer traces of boards which are working correctly, as well as offer practical debugging advice as the boards are built. Finally, we detail a procedure for how the RF board s-parameter files should be measured and named, and also describe where they are stored in the data warehouse, which will be of use to data analyzers and simulators.

Contents

1 General Comments **3**
- 1.1 Introduction . . . 3
- 1.2 Powering and Communicating with the Tester Boards . . . 3
- 1.3 Powering and Communicating with the PC Boards . . . 3
- 1.4 Powering and Communicating with the RF Boards . . . 5

2 ARAFE Power and Control (PC) Board **7**
- 2.1 Overview . . . 7
- 2.2 Programming the PC Board . . . 7
- 2.3 5V DCDC Converter Testing . . . 8
- 2.4 12V DCDC Converter Testing . . . 11
- 2.5 Microcontroller Testing . . . 11
- 2.6 Thermal Cycling and Programming the Serial Number in Flash Memory . . . 13

3 ARAFE Radio Frequency (RF) Board **15**
- 3.1 Construction and Initial Testing . . . 15
- 3.2 Attenuator Testing . . . 16
- 3.3 Thermal Cycle and RF Caging . . . 19

[*]clark.2668@osu.edu
[†]allison.122@osu.edu

4 RF Characterization **22**
4.1 About the S-Parameters 22
4.1.1 File Storage 22
4.1.2 File Labelling Conventions 22
4.2 Taking the S-Parameters 23
4.3 Measurements at OSU 23
4.3.1 Procedure 23
4.3.2 Data Processing 24

A Appendix: OSU Code to Take S-Parameters **25**

B Appendix: Code to Merge CSV Signal Files **29**

C Appendix: Code to Merge CSV Trigger Files **30**

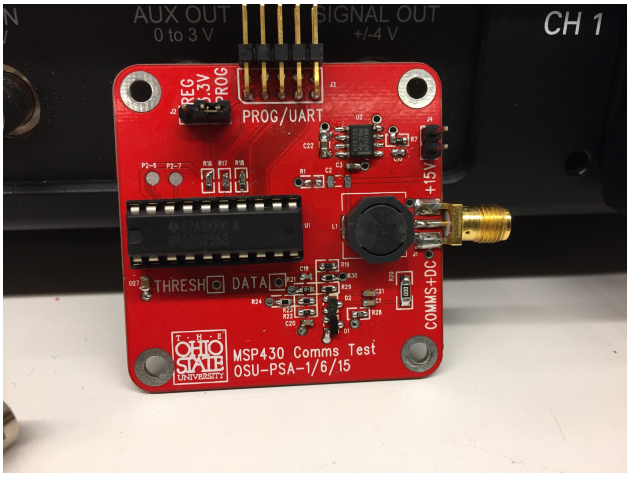

Figure 1: A photo of the MSP 430 Comms Tester. This board serves as a master to the PC slaves.

1 General Comments

1.1 Introduction

A completed ARAFE module, or ARAFE quad, consists of one power and control (PC) board and one radio frequency (RF) board. Each RF board contains four channels of signal conditioning, with tunable attenuation for signal and trigger paths. As such, there are four quads per station for all sixteen antennas. The four quads are controlled by one ARAFE master board. Therefore, a complete ARAFE system consists of four PC boards, four RF boards, and 1 master board.

1.2 Powering and Communicating with the Tester Boards

Testing of the PC and RF board should be done with the ARAFE slave tester, which is the custom red "MSP430 Comms Test" board in figure 1. The communications tester provides power to the quad, and also provides communications-over-power, as detailed here and originally here. So, the comms-tester powers and talks to the ARAFE quads. Communication to the comms-tester is provided by a custom built USB-to-UART converter, the blue board in figure 2. The comms-tester and USB-to-UART bords can be plugged in to one another as in figure 3.

Power is delivered to the comms tester by providing a 15V rail and GND to J4. Power is provided from the comms-tester to the quads by the SMA output, or J1. Communication from laptop to the USB-to-UART bridge is provided by the mini-USB port, or J1. All of this can be seen in the functional diagram in the top of figure 3.

1.3 Powering and Communicating with the PC Boards

The PC boards are powered by +15V and GND, delivered to J1. The two left pins are power, and the two right pins are GND. A photo of a correct powered board can be viewed

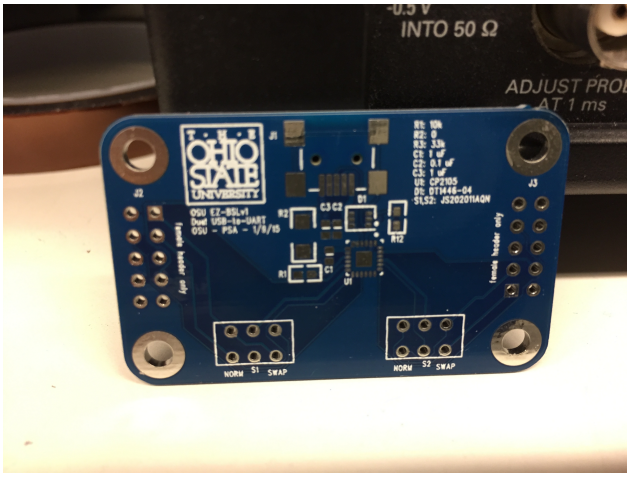

Figure 2: A photo of the USB-to-UART converter. This board serves as a USB interface between a computer (like a laptop) and the MSP430 Comms Tester.

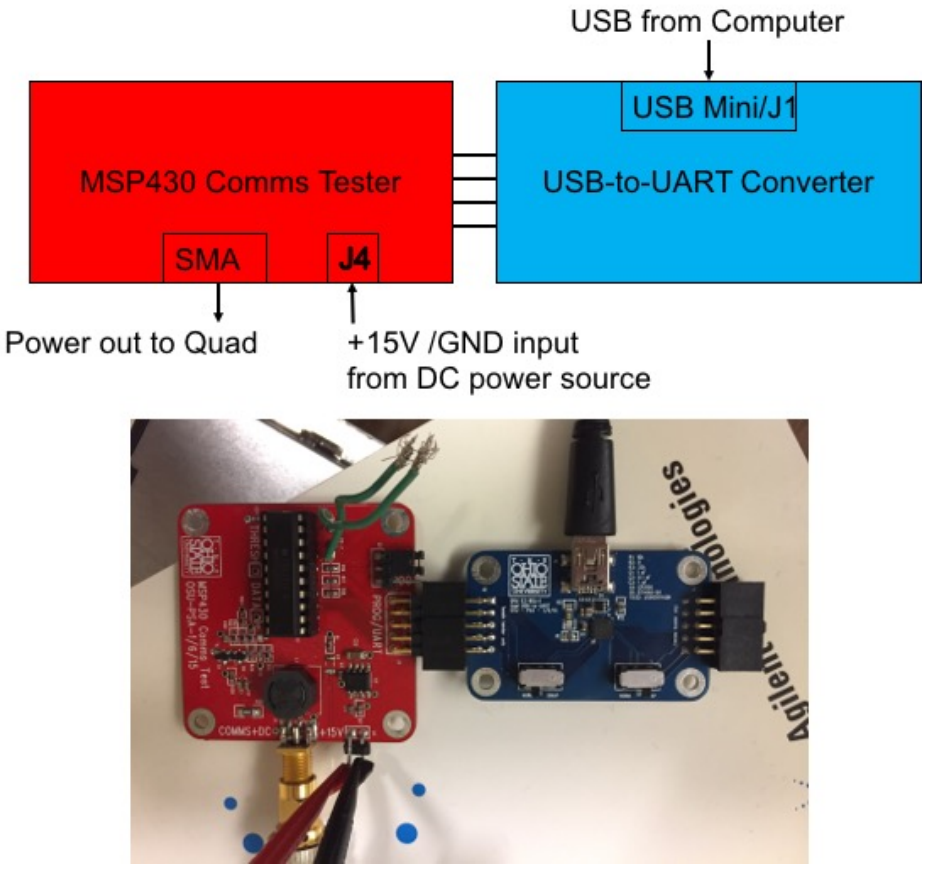

Figure 3: A photo of how to connect the MSP430 Comms Tester to the USB-to-UART converter, along with a functional diagram.

in figure 4. The board can be powered either by the comms-tester described in section 1.2, or can be powered directly by a DC power source.

Since the PC board contains a microcontroller (more on that later), it is equipped with two powering modes, controlled by the jumper J2. The orientation of the jumper J2 controls whether board is in "program" mode, for programming the microcontroller via TI Code-Composer studio/ MSP430 FET Pro LITE, or in "external power" mode, for testing with external power. When the left and middle pins of J2 are connected, the board is configured for external power. When the right and middle pins of J2 are connected, the board is configured for programming. The general procedure for powering the PC boards externally is the following:

1. Jumper together the leftmost pins of J2 on the PC board (the jumper setting for "powered externally").

2. Plug the USB-to-UART converter into the comms-tester. Wire the 15V intput and GND from a DC power source to the comms-tester, but do not turn the supply on yet. A photo of all devices plugged in and ready for use is given in figure 5.

3. From comms tester SMA output, deliver +15V to one of the two left pins of J1 on the PC board, and GND to one of the two right pins of J1 on the PC board. This is most easily done by splitting the SMA output to two micrograbbers.

4. Turn the power supply on. With only the PC board as a load (including DC-DC convereters) it should draw about 20 mA. (The photo in figure 5 shows larger current draw because it was powering an alternative set-up at the time.)

The PC board is reverse bias protected, so if the board does not power up, the polarity of the input is the first thing to check. A photo of the board correctly connected is in figure 4.

1.4 Powering and Communicating with the RF Boards

Powering the RF board must be done carefully. Preferably, you should power the RF board only with the PC board, which will protect the RF board from any under/over-voltage issues, as well as reverse bias voltage. If you need to directly power an RF channel, be sure to get the polarity right, and to not apply more than +5V. Further, because the RF board is 31 mil FR4 (to keep traces thin), and because the header pins engage strongly between the PC and RF board, you should avoid plugging the PC board directly into the RF board during testing. Plying the two apart is difficult, and results in excessive flexing of the RF board. A better solution is to get a breadboard, plug in the male PC board, and wire "arduino type" jumper wires (eg: https://www.adafruit.com/product/758) to the female RF board. This narrower wires do not engage the female headers fully, and are much easier to remove, and even more so, can be removed without flexing the RF board. A photo of Brian Clark's test set-up in the Ohio State lab is visible in figure 6.

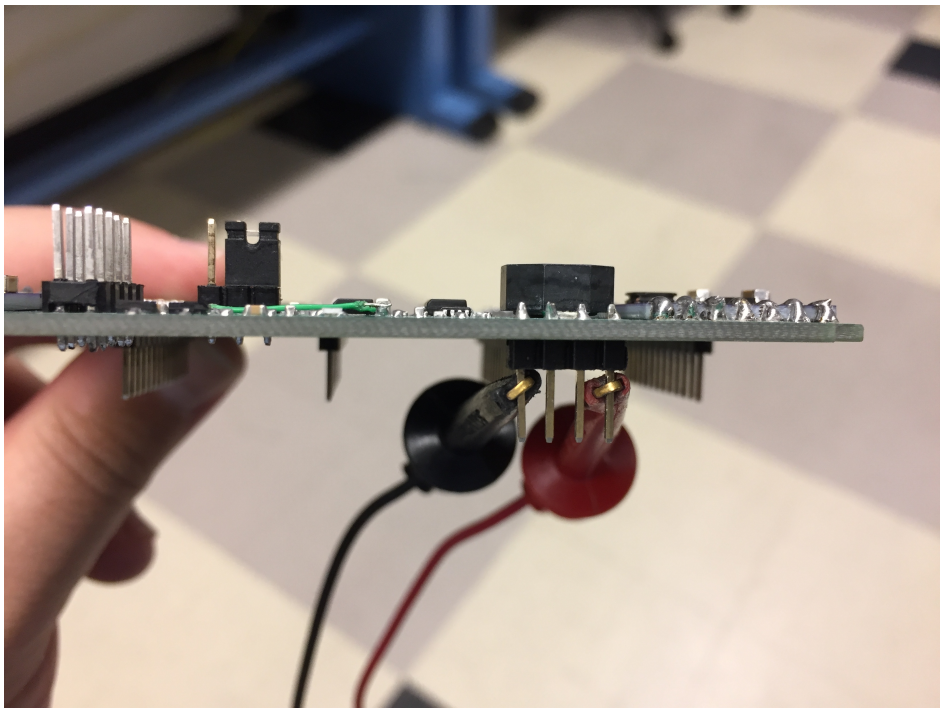

Figure 4: A photo of a correctly powered PC board. Red is +15V and black is GND. In the upper left, you can see the left-most pins of J2 have been jumpered together for "external power" mode. Note that because we are looking at the board edge on from the top, "left" and "right" are mirrored relative to the text.

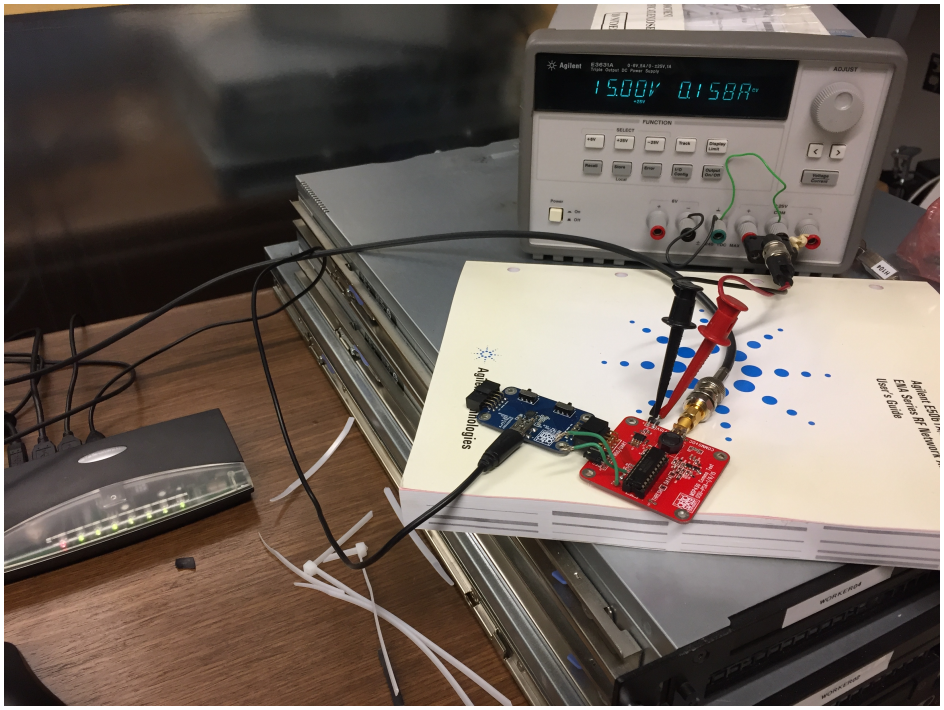

Figure 5: A photo of the complete power supply, USB-to-UART, and MSP430 comms tester ready for testing.

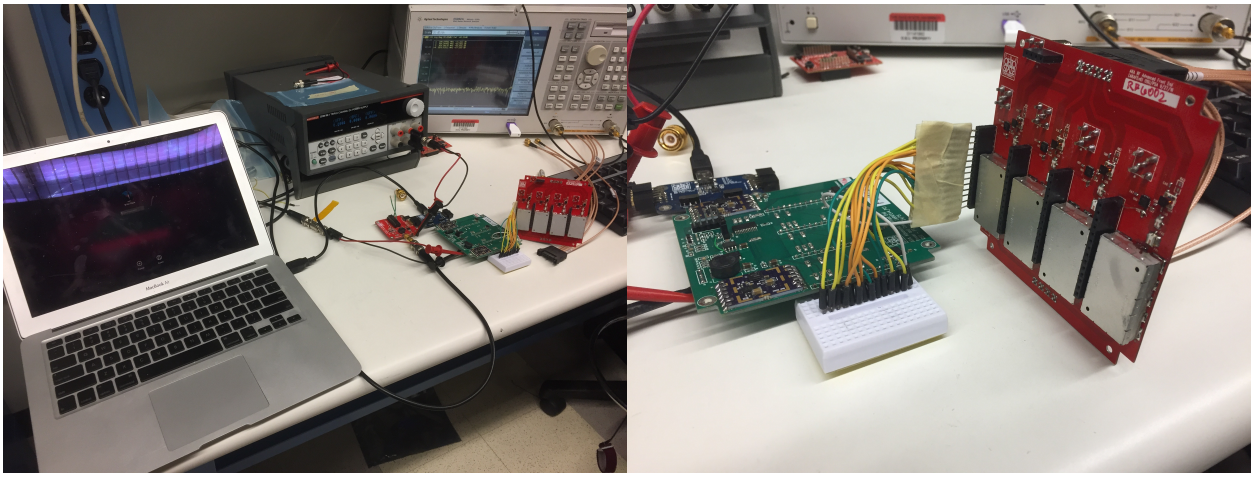

Figure 6: Figure of ARAFE RF testing setup at OSU.

2 ARAFE Power and Control (PC) Board

2.1 Overview

The Power and Control (PC) board provides power to the RF board as well as controls power distribution downhole. The board is equipped with under-over voltage protection, reverse bias protection, and its central brain is a MSP430G2153 microcontroller. The board contains two custom, high efficiency DCDC converters; one converter steps down the 15V from the ASPS DAQ board to 5V, and the other to 12V. The 5V converter powers the RFSA variable attenuators and amplifiers for the RF board, while the 12V converter provides downhole power to the optical zonus via bias-tee. To vet a PC board, the board should be powered, and it's 5V converter, 12V converter, and microcontroller should be tested.

2.2 Programming the PC Board

Programming the PC board is done via Code Composer Studio or CCS (http://www.ti.com/tool/ccstudi
The code repository for the PC board is called "arafe_slave_2" and is located in the ARA DAQ Github. It can be loaded into CCS using CCS's Git capability: File → Import → Git → Projects from Git.

To compile and upload the code to the microcontroller, you will use the 10-pin header on the board. If you have the TI USB FET Programmer you can use that directly. Plug the FET programmer into your computer, and hit Run → Debug, which should upload the code to the microcontroller. If you do not have the programmer, you can use almost any of TI's development boards as programmers also. That is explained here and here You will jumper between the development board TEST, RESET, GND, and 3.3V to the corresponding pins on the PC board. Photos of how to do this are visible in figure 7 and figure 8.

Because the PC board/slave firmware uses the microcontroller information memory, CCS's default programming routine has to be altered. The memory management has to be changed from "Erase Main Memory Only" to "Erase Main and Information Memory." This can be done by doing Run → Debug Configurations → Target → MSP430x → Erase

9	7-TEST	5-GND	3-BSL_RX_UART_TX	1-BSL_TX_UART_RX
10	8-VCC	6-VCC	4-RST	2

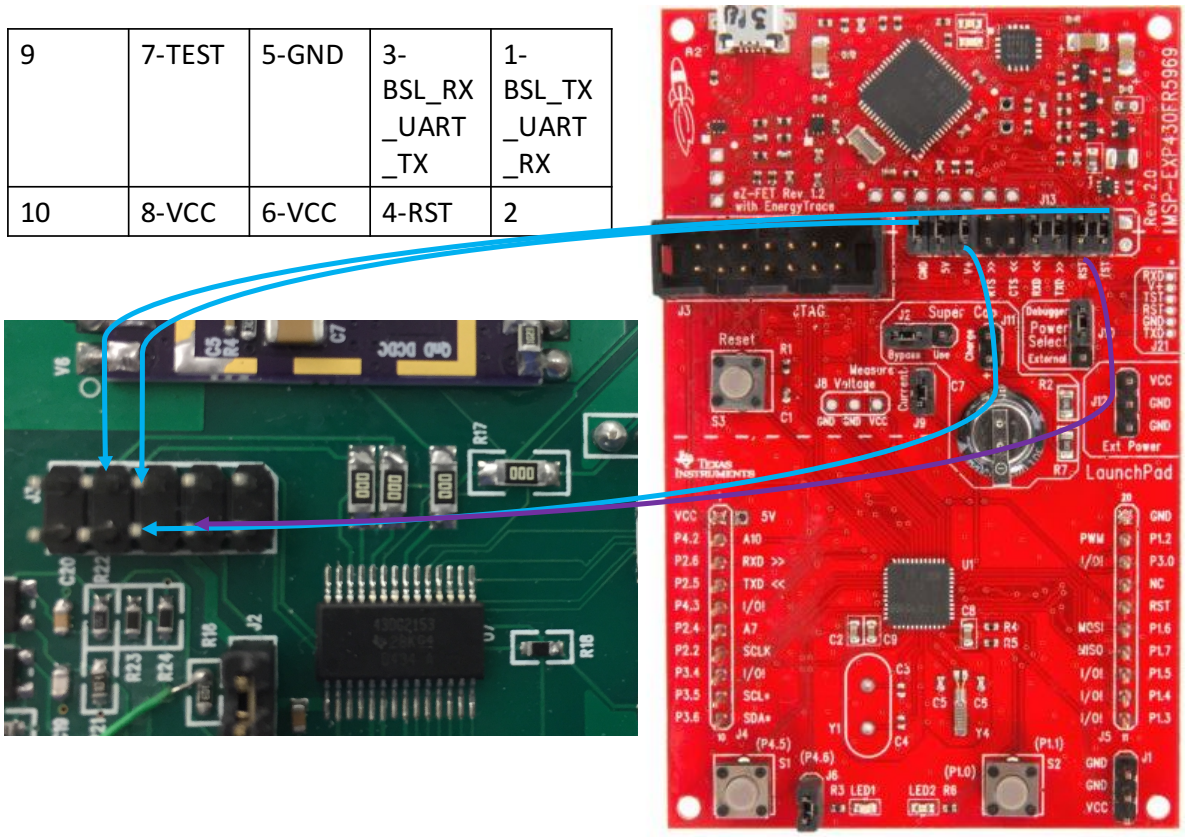

Figure 7: A diagram of the wire jumpering required to use a TI development board as a programmer.

Options and select "Erase main and information memory." If when you try and program the board you get the following error "MSP430: File Loader: Verification failed: Values at address 0x1000 do not match Please verify target memory and memory map." This is the thing to correct.

2.3 5V DCDC Converter Testing

- Test the output of the 5V DCDC converter with a multimeter. It should register +5V. The output of the DCDC converter is labeled "Vout" in figure 11.
- Using a multimeter, probe pin 3, 8, and 14, on the jumpers, ensuring they are all +5V to begin.
- Issue the command "`5v 0`" to the slave tester command line. This will turn off the 5V converter; check with the multimeter that this is true. The current drawn on the power supply should decrease. Issue the comamnd "`5v 1`" to re-enable the 5V converter, and ensure the multimeter registers +5V, and check that the current draw on the power supply increases again.

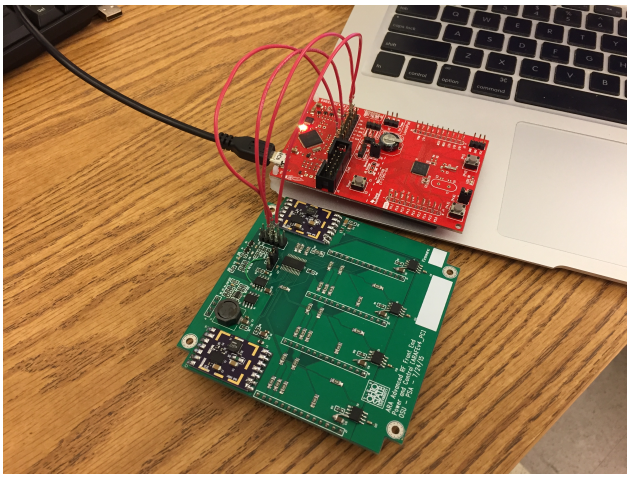

Figure 8: A picture of programming the PC board with a TI development board at OSU.

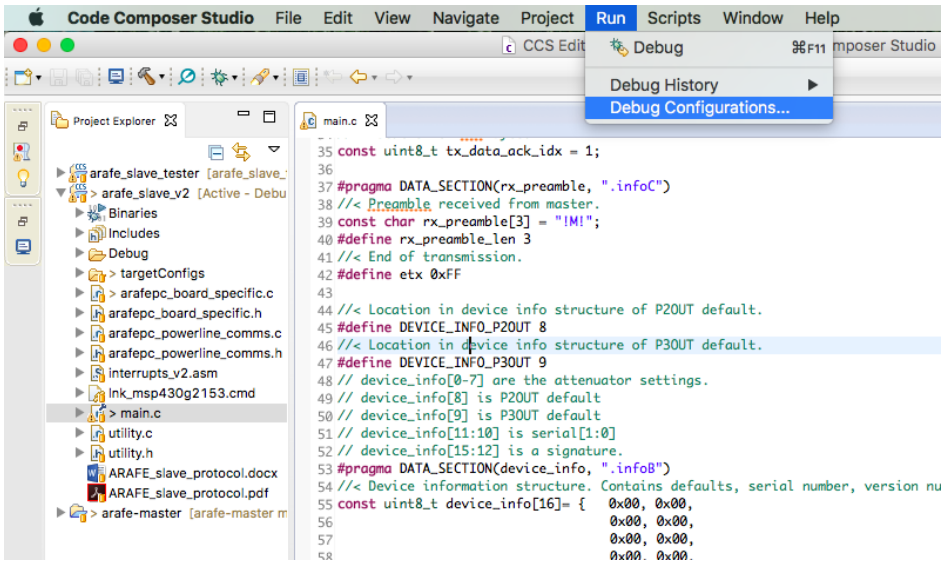

Figure 9: How to navigate to the debug configuration in CCS.

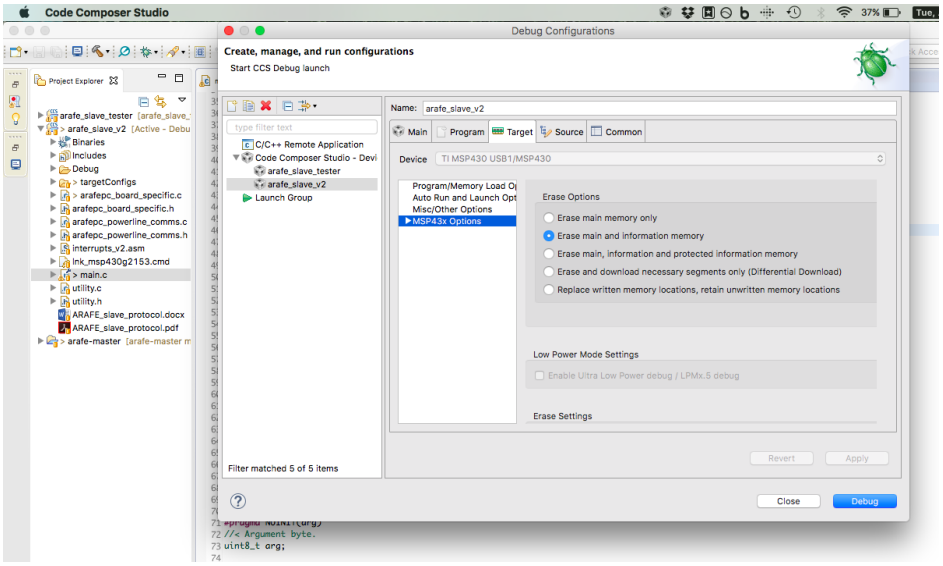

Figure 10: How to navigate to change the memory management in CCS.

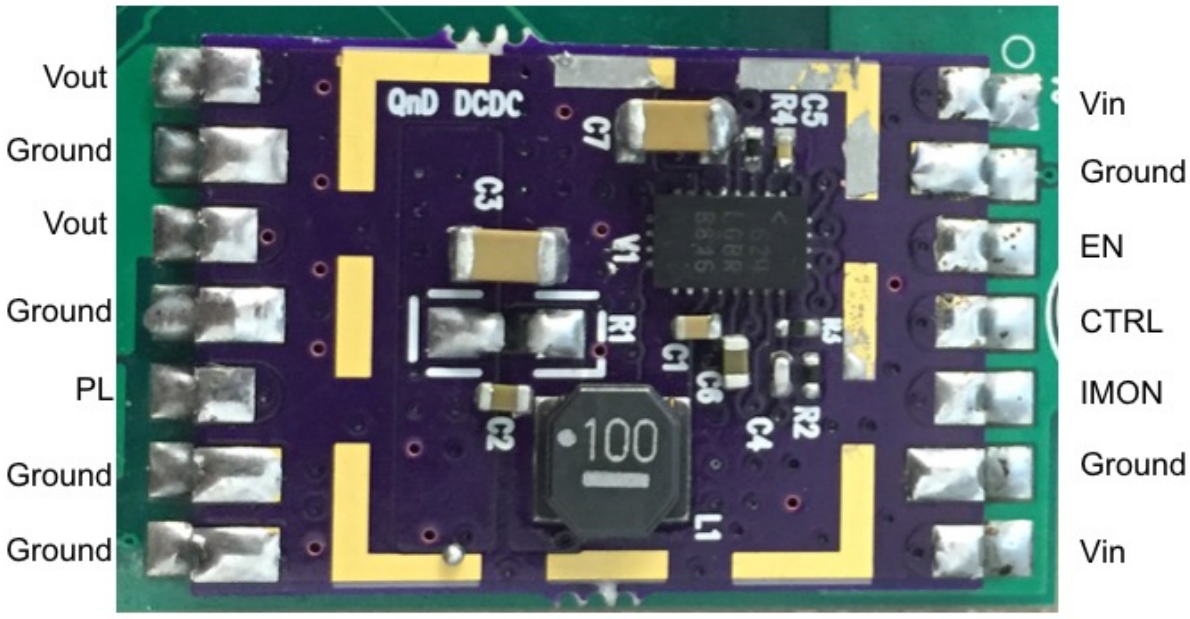

Figure 11: Photo of the DCDC converter with labeled inputs and outputs.

2.4 12V DCDC Converter Testing

- Test the output of the 12V DCDC converter with a multimeter. It should register +12V.
- Issue the command "`12v 0`" to the slave tester command line. This will turn off the 12V converter; check with the multimeter that this is true. The current drawn on the power supply should decrease. Issue the comamnd "`12v 1`" to re-enable the 12V converter, and ensure the multimeter registers +12V, and check that the current draw on the power supply increases again.
- Using a multimeter, probe pin 1 of J4-J7, ensuring they are all grounded to begin.
- Issue the command "`control 0 [0-3]`" to the slave tester via command line. This will enable the individual 12V lines. Probe each line (0-3) in turn with a multimeter to ensure the voltage rises to +12V. There should also be an increase in current drawn from the power supply.

2.5 Microcontroller Testing

The microcontroller job is to program the RFSA3713 variable attenuators by toggling three analog lines: CLK (clock), DATA/SI (data), and LE (latch enable). The uC also allows for some current and voltage monitoring. CLK and DATA are mutual to all attenuators on all channels. Which attenuator will "listen" is controlled by setting the LE pin. We want to test if the uC is issuing necessary commands, to the right attenautor, by probing each of these three pins with an oscillosope, and looking for the analog bits in the oscilloscope traces. The pin mapping between the various inputs of the attenuators, various on board jumpers, and the pin output of the uC is detailed in table 1.

- Prepare an oscilloscope with two probes.
- Issue a command to control either the "signal" attenuator or the "trigger" attenuator. The command for the signal attenuator is "`sig [channel] [setting] = sig [0-3] [0-127]`" while the command for the trigger attenuator is "`trig [channel] [setting] = trig [0-3] [0-127]`".
- Check the CLK pins (J7 and J13) for every channel. You should see 16 clock pulses on every pin, no matter the command or channel to which the command was issued. We advise setting the trigger to this pin.
- Probe the two data pins J5 and J11 for every channel. The height of the transmitted pulse should be roughy half that of the +5V line.
  - Should see something like the SI line on the RFSA timing diagram 12. A picture of the oscilloscope trace is in figure 14.
  - There should be a high bit at the end of the bit train (A7) regardless of the command issued.

Att Pin	Atten Function	Jumper Pin	uC Pin	uC Function
Ch 0				
12	Ch 0, Sig LE	JX-6	19	Ch 0, Signal LE
12	Ch 0, Trig LE	JX-12	18	Ch 0, Trigger LE
13	Ch 0, Sig and Trig Clock	JX-7 (SIG) & JX-13 (TRG)	27	Global SPI Clock
14	Ch 0, Sig and Trig Data	JX-5 (SIG) & JX-11 (TRG)	26	Gobal SPI Data
Ch 1				
12	Ch 1, Sig LE	JX-6	16	Ch 1, Signal LE
12	Ch 1, Trig LE	JX-12	17	Ch 1, Trigger LE
13	Ch 1, Sig and Trig Clock	JX-7 (SIG) & JX-13 (TRG)	27	Global SPI Clock
14	Ch 1, Sig and Trig Data	JX-5 (SIG) & JX-11 (TRG)	26	Gobal SPI Data
Ch 2				
12	Ch 2, Sig LE	JX-6	13	Chan 2, Signal LE
12	Ch 2, Trig LE	JX-12	12	Chan 2, Trigger LE
13	Ch2, Sig and Trig Clock	JX-7 (SIG) & JX-13 (TRG)	27	Global SPI Clock
14	Ch2, Sig and Trig Data	JX-5 (SIG) & JX-11 (TRG)	26	Gobal SPI Data
Ch 3				
12	Ch 3, Sig LE	JX-6	10	Chan 3, Signal LE
12	Ch 3, Trig LE	JX-12	9	Chan 3, Trigger LE
13	Ch 3, Sig and Trig Clock	JX-7 (SIG) & JX-13 (TRG)	27	Global SPI Clock
14	Ch 3, Sig and Trig Data	JX-5 (SIG) & JX-11 (TRG)	26	Gobal SPI Data

Table 1: Details of the connectivity between the RFSA pins, the jumpers on the PC/RF board, and the uC.

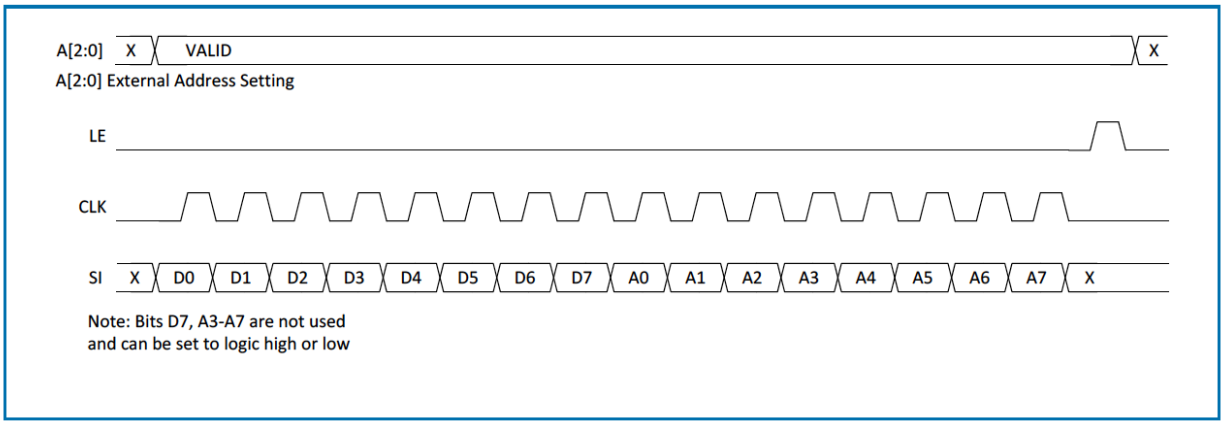

Figure 12: The bit timing diagram for programming RFSA3713 variable attenuator. From [1].

  - Bits D7-A6 should all be low, regardless of the command issued.
  - The first bits D0-D6 should be a combination of low and high, corresponding to the command issued. Note that the seven bits are exactly the number required to control the range of the RFSA; all 0's for setting zero, or minimal attenuation, and all 1's for setting 127, or maximal attenuation.
  - One should see the same command on every data pin, no matter the command or channel to which the command was issued.

- Check the LE pins (J6 for signal and J12 for trigger) for every channel, both signal and trigger. This is where the differences between the "sig" and "trig" commands, as well as the differences between channels, is important.

  - You should see one bit after the last clock bit (see timing diagram 12). A picture of the oscilloscope trace is in figure 13.
  - This bit should only be present in the channel to which the command was issued. That is, if you issue "sig 1" you should only see a bit on the signal attenuator pin of channel 1 (header pin JX-6 of channel 1), and no other. The firmware has already been debugged, so the only way this should not work is if a pin is shorted incorrectly.

2.6 Thermal Cycling and Programming the Serial Number in Flash Memory

Each board should be thermal cycled, and the microcontroller test in section 2.5 should be repeated for all channels to check for broken soldering connections. Once it has passed it's thermal cyclce, the uC needs to be assigned the board's serial number. The uC for each

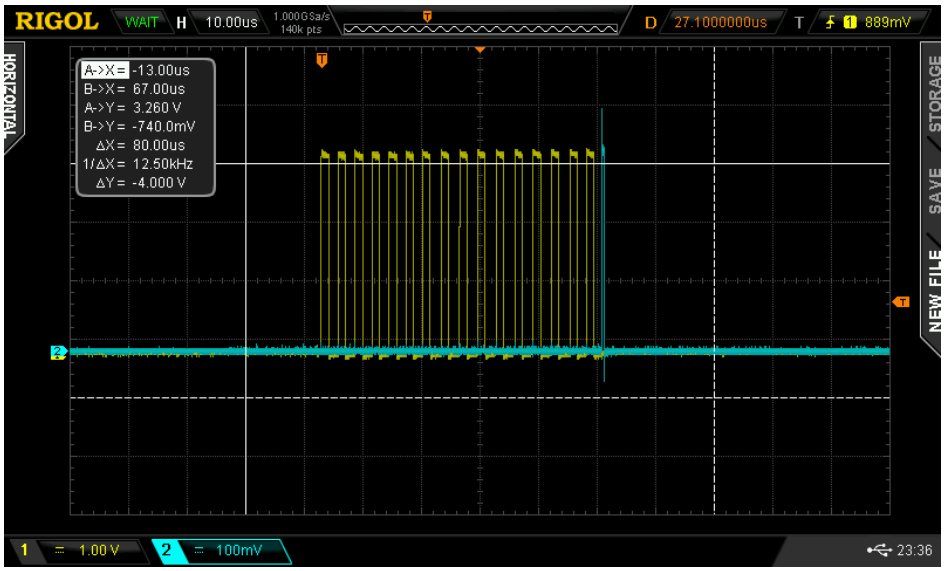

Figure 13: A photo of the uC clock (yellow) and latch enable (blue) pulses on an oscilloscope.

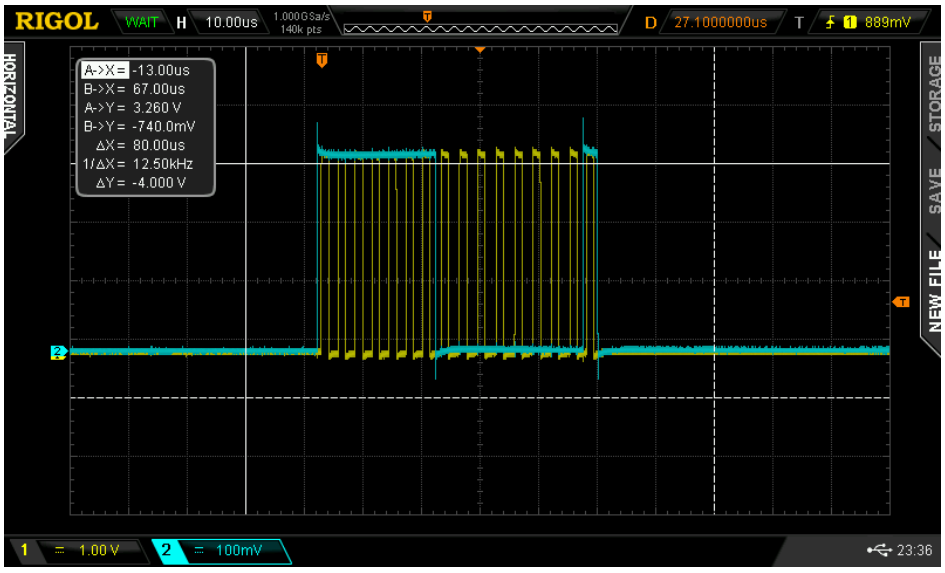

Figure 14: A photo of the uC clock (yellow) and data (blue) pulses on an oscilloscope.

PC board has onboard flash memory to store a permanent serial number. This means that the serial number can be recovered after deployment if need be. At the end of testing (post thermal cycle), a serial number should be written on the board, and flashed into memory. In revision four of the boards, for example, a PC board might have the serial number PC4002, where the "PC" identifies it as a power and control board, the "4000" identifies it as board revision 4, and the 2 is the board specific identifier. In memory, device info 10 holds the most significant byte (MSB) of the serial number, and device info 11 holds the least significant byte (LSB) of the serial number. We want the numbering to start at 4000, which is 0x0FA0 in binary. In the following example, we will program the serial number for board 4002, or 0x0FA2. So:

- We set the MSB to 0x0F, which in decimal is 15. So we will issue the command "`write 10 15`". Now, we have the total value as 0x0F00, which is 3840 in decimal.
- Now, we set the LSB to 0xA2, which is decimal 162. So we will issue the command "`write 11 162`". To summarize, what we have done is program the serial number to be 0x0FA2, which is 0xF000 (3840 in decimal) + 0x00A0 (160 in decmial) + 0x0002 (2 in decimal), or 4002 total.
- Issue the command "`flash`" which will store the serial number in the flash memory of the uC.
- Issue the dump command "`dump`" which will write out the content of the flash information memory to the terminal. Check that the third row, right two columns reflect the serial number you just programmed.

3 ARAFE Radio Frequency (RF) Board

3.1 Construction and Initial Testing

Testing of the RF board should be done channel by channel. After attaching surface mount components, testing of the RF channel requires installation of

- an iso-rate adapter
- a header through pin
- an SMA.

Here is some advice on building the boards

- Test channel 3 first, and channel 0 last. This allows you to power up the channels, while still having full access to the surface mount components.
- Even without power, the S11, S22 trigger path, S22 signal path, and S21 signal path all have distinct gain patters. They can be observed in figure 15,16,17, and18. Therefore, you can actually check many of the necessary connections on the board without powering it on.

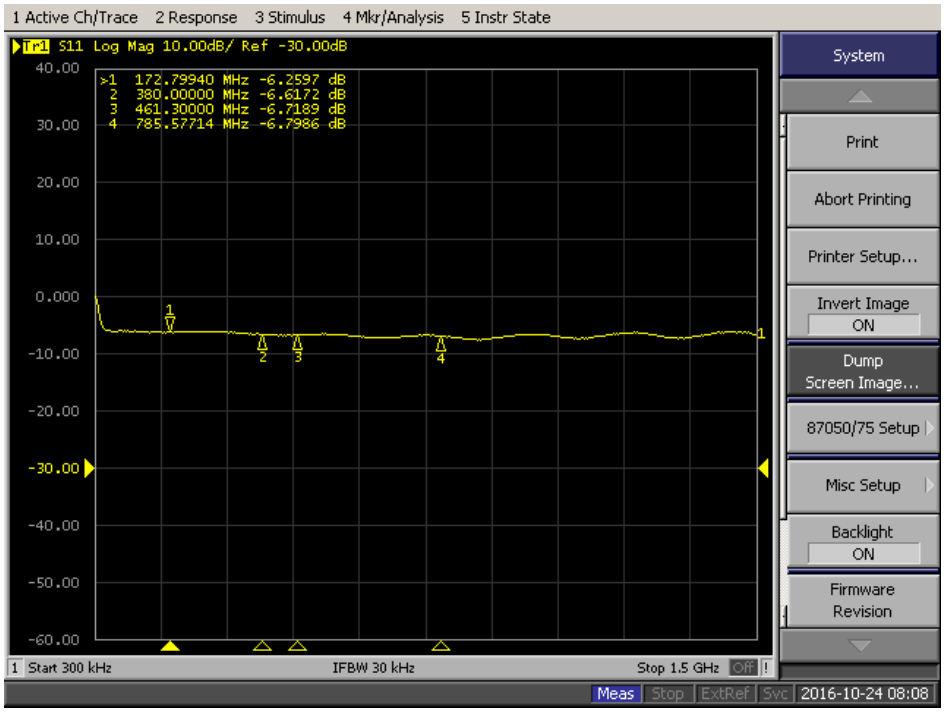

Figure 15: Ideal S11.

Once powered:

- The S21 signal gain should be smooth across the band up to ±2dB as in figure 19. All network analyzer traces should be from channel 2 of RF4003 (this is here for OSU's documentation sake).
- The S21 trigger path will only be open once powered (this is because in most couplers the couple path is DC shorted to ground). The S21 gain should be as in figure 20. Note:
  - The trigger path is attenuated an additional 10 dB, consistent with the choice of coupler.
  - There are "ripples" in the response that are likely due to reflections off the coupler, and potentially the SMA. Given that the couple path is routed to the trigger, and the time integrator functions over ˜5 ns, this reflection should not affect the triggering effectiveness.

3.2 Attenuator Testing

Now that we know the channels are working, we need to verify we can set the variable attenuators. The attenuators have a setting between 0 and 127, with each digit for a 0.25 dB increment of the 31 dB total attenuation possible with the RFSA 3713. The procedure is the following:

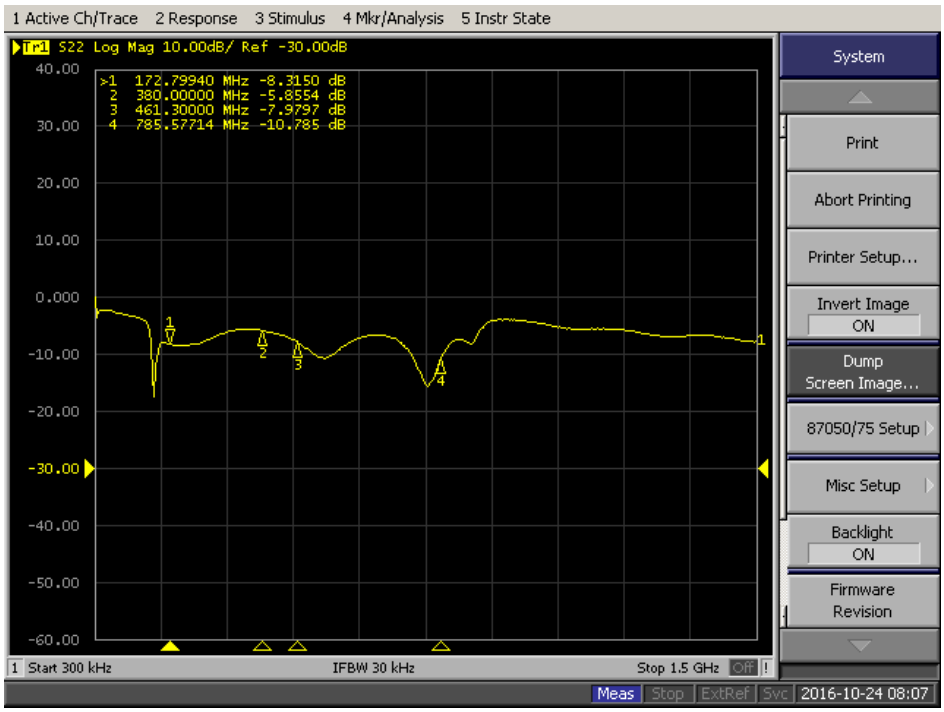

Figure 16: Ideal Signal Path S22.

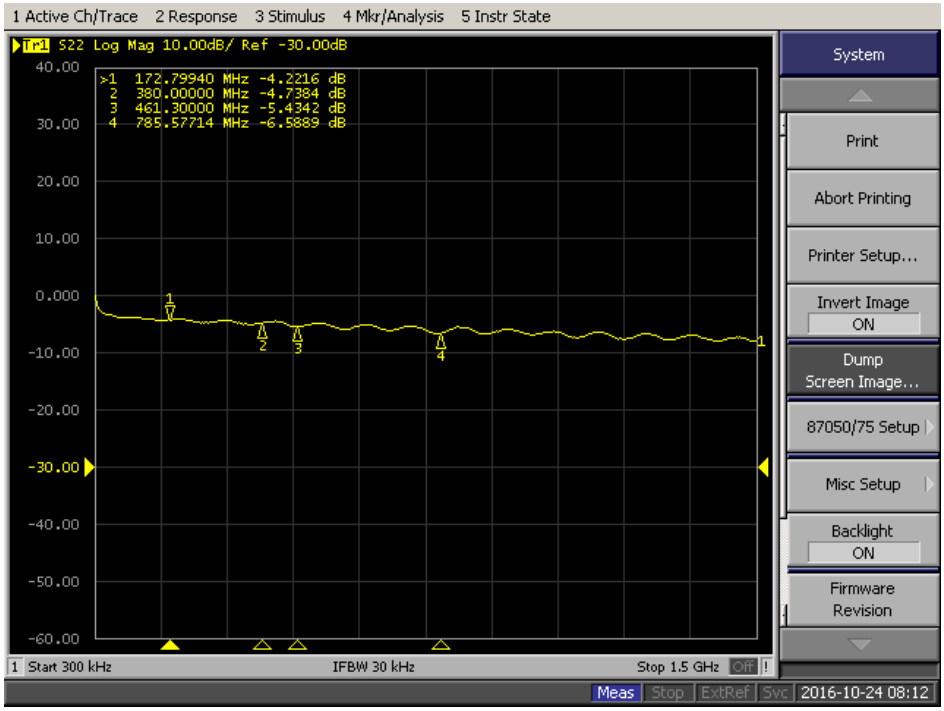

Figure 17: Ideal Trigger Path S22.

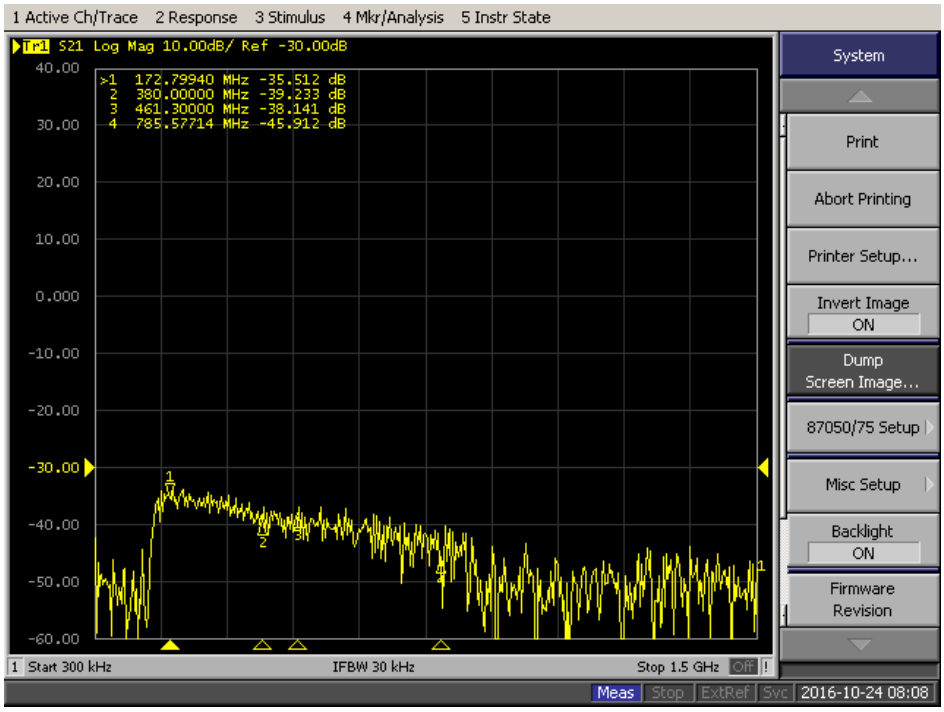

Figure 18: Ideal non-powered signal S21. Note that even with the power off, the system response can be observed in the gain.

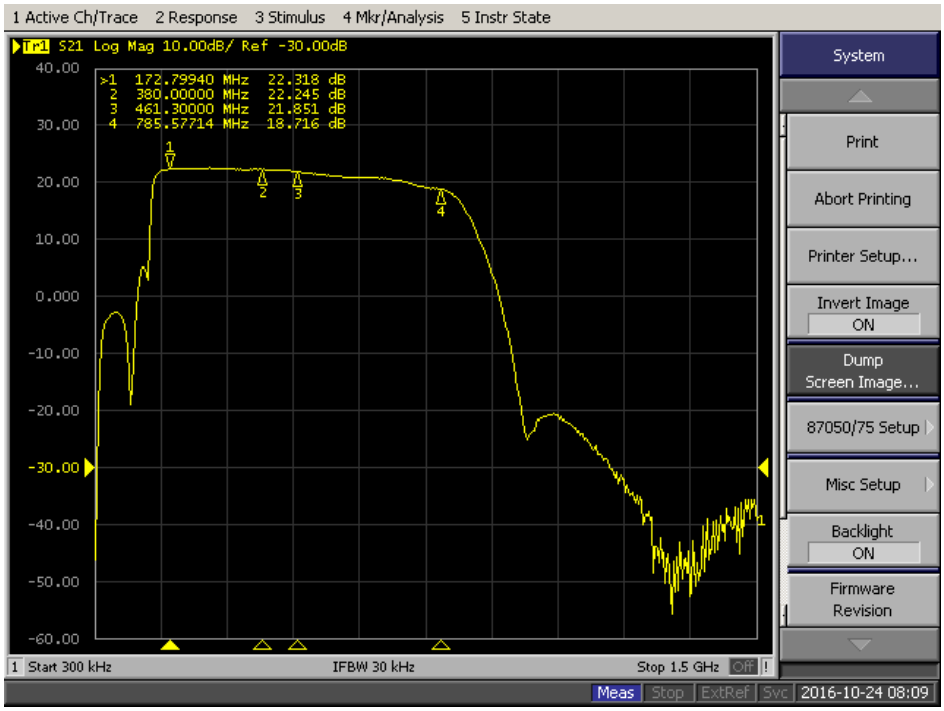

Figure 19: Ideal powered signal S21.

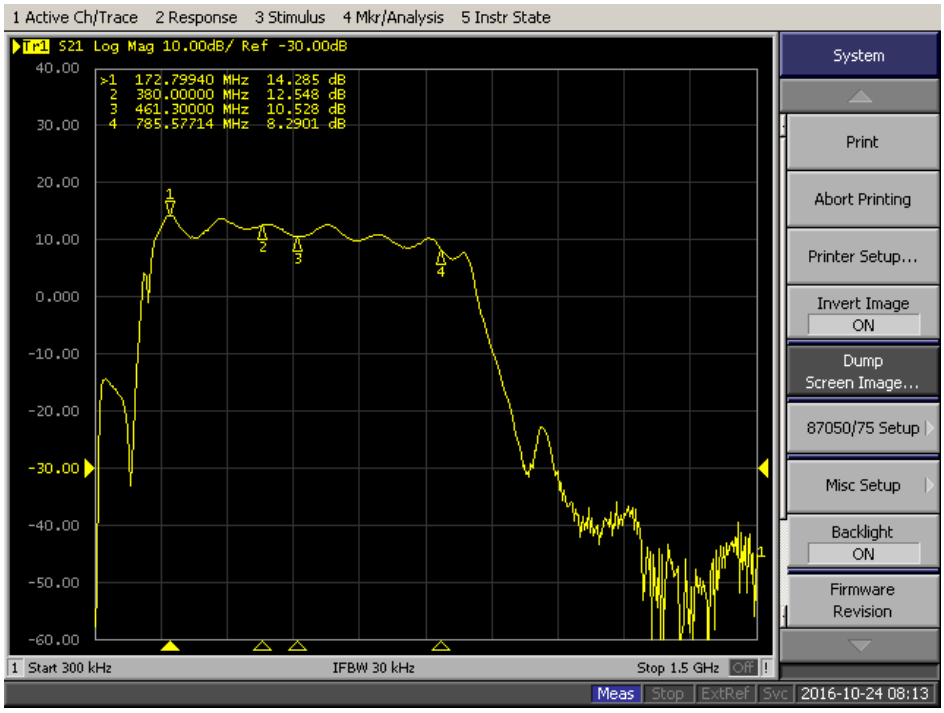

Figure 20: Ideal powered trigger S21.

- Power a channel
- Set the signal attenuation with the command: "`sig [channel] [setting] = sig [0-3] [0-127]`" Play around with various value of attenuation, and you should see the curve move up and down, as in figure 21.
- Set the trigger attenuation with the command: "`trig [channel] [setting] = trig [0-3] [0-127]`". Play around with various value of attenuation, and you should see the curve move up and down, as in figure 21.
- In both cases, make sure that setting the attenuator to "0" results in minimal attenuation, and that setting the attenuator to "127" results in a reduction in the gain by 30 to 31 dB.

Plotted system responses for channel 3 signal and trigger paths. This was done for RF4001 (this is here for OSU's documentation sake).

3.3 Thermal Cycle and RF Caging

Once all RF channels are working, you should thermal cycle the boards, and repeat section 3.2 to ensure no solder joints have broken in thermal test. If joints break, repair them. Once the boards have passed their thermal test, you should attach their RF cages, and give them a serial number on their silk screen. Attaching cages is easiest if you solder the corners first from the outside, and solder the inner pads near the headers last by poking

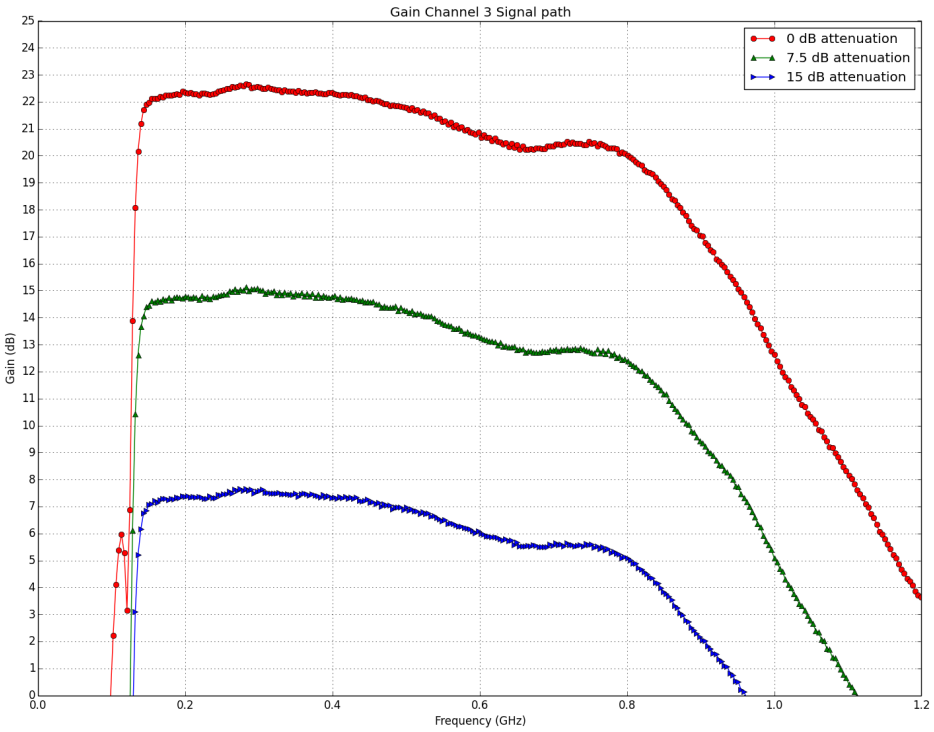

Figure 21: Example of a signal path gain pattern in an ARAFE at several different attenuation values. Note that this plot was made with a LFCN-800+ low pass filter. This was later replaced witha LFCN-630+ filter that has a sharper cutoff above 850 MHz. So the curve you observe should fall away more than 20 dB above ˜850 MHz, as in figure 22. A sign that you have chosen the wrong filter is if the gain does not fall off fast enough.

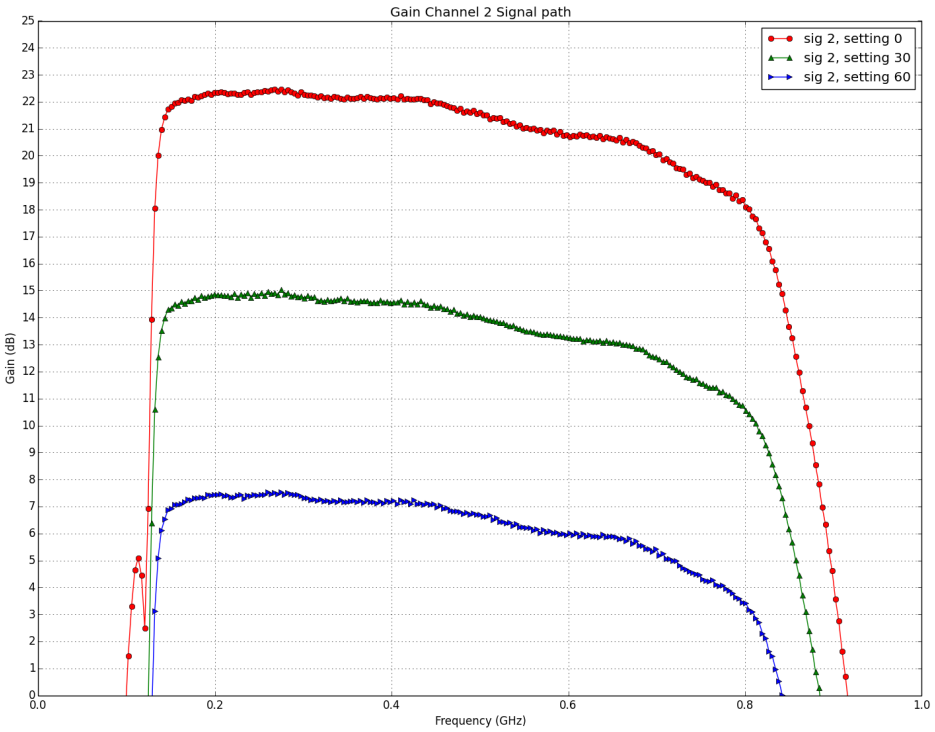

Figure 22: This is an example of the signal path gain with a correctly chose filter, the LFCN-630+.

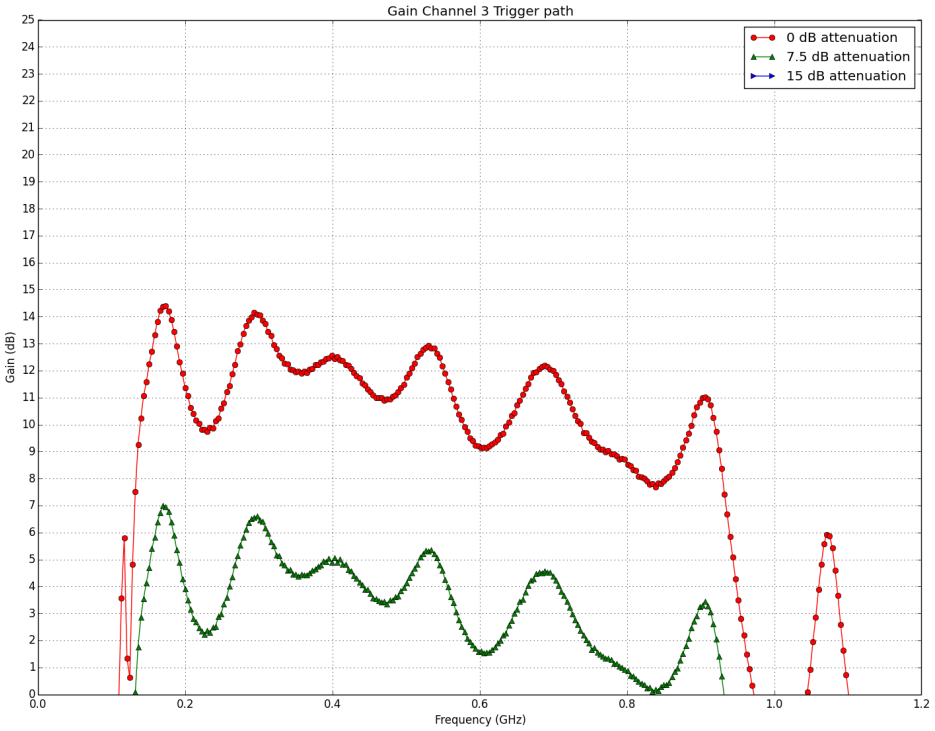

Figure 23: Example of a trigger path gain pattern in an ARAFE at several different attenuation values.

the iron through the open cage. In all cases, you can use solder paste instead of a wire of solder if you like.

4 RF Characterization

4.1 About the S-Parameters

The RF boards must be fully characterized before final integration into the DAQ. This consists of recording the complex response (phase and magnitude) of each channel of the RF board, at a variety of temperatures and attenuator settings. We should record full 2-port s-parameter ("s2p") files in an industry standard fashion, so they can be directly imported into simulation software like Qucs studio.

4.1.1 File Storage

The RF characterization files are stored in the ARA Data Warehouse at the University of Wisconsin. For ARA collaborators with Cobalt access, you can find the files at "`/data/wipac/ARA/pre-deployment/ARAFE`".

There are two sub-directories. "PROCESSED" and "RAW". The RAW folder serves strictly as a back-up for the raw data from the Vector Network Analyzer. Analyzers and simulators should preferably use the PROCESSED data which contains the S2P touchstone files.

4.1.2 File Labelling Conventions

The full s-parameter files are uploaded as tar.gz files specifying the RF board, the channel and the temperature. So the naming convention is "ARAFExxxxSIGnnnn.zip" and "ARAFExxxxTRGnnnn.zip". "xxxx" specifies the board, "SIG" specifies the signal vs trigger path, and "nnnn" specifies the temperature. "nnnn" can be N20, N10, P00, P10, P20, or P30 for -20C, -10C, +0C, +20C, or +30C.

The labelling convention in the PROCESSED files is:
"`ARAFExxxx_SIGy_SIGzzz_TRGvvv_Tnnn.s2p`".

Where "xxxx" is the name of the ARAFE quad number, so ARAFE60002 is for RF board 6002. "y" is the channel; so 0 means RF channel 0, 3 is RF channel 3, etc. "zzz" is the setting of the signal attenuator from 0 to 127. "vvv" is the trigger attenuator setting from 0 to 127. "nnn" designated the temperature; so P00 means positive 0C, etc.

The labelling convention in the RAW files (from the OSU Vector Network Analyzer) is
"`ARAFExxxx_SIGy_SIGzzz_TRGvvv_Tnnn_SmmmGDL.CSV`",
"`ARAFExxxx_SIGy_SIGzzz_TRGvvv_Tnnn_SmmmMAG.CSV`",
"`ARAFExxxx_SIGy_SIGzzz_TRGvvv_Tnnn_SmmmPHS.CSV`".

Where "xxxx" is the name of the ARAFE quad number, so ARAFE60002 is for RF board 6002. "y" is the channel; so 0 means RF channel 0, 3 is RF channel 3, etc. "zzz" is the setting of the signal attenuator from 0 to 127. "vvv" is the trigger attenuator setting from 0 to 127. "nnn" designated the temperature; so P00 means positive 0C, etc. "mmm"

designates which s-parameter, so S21, S11, etc. "GDL" designates the file as a group delay file, "MAG" designated it as a log-magnitude file, and "PHS" designates it as a phase file.

These are merged into the PROCESSED S2P files described above by the procedure described in section 4.3.2.

4.2 Taking the S-Parameters

The procedure for characterization is as follows:

1. Put the quad in a thermal chamber, and let it settle at that temperature for 2 hours. We recommend six temperatures, -20C, -10C, 0C, 10C, 20C, and 30C.

2. After settling at that temperature, take the complex 2-port s-parameter file for every channel. This means you should take the magnitude and phase for S11, S22, S12, and S21.

   (a) For every signal channel, you should take the s-parameter file for the following attenuation values: 0, 8, 16, 24, 32, 40, 48, 56, 64, 72, 80, 88, 96, 104, 112, 120.

   (b) For every trigger channel, you should loop over the following signal settings: 0,16,32,48,64,80,96. For each signal setting, you should loop over the following trigger settings: 0, 8, 16, 24, 32, 40, 48, 56, 64, 72, 80, 88, 96, 104, 112, 120, and for each one, record the s-parameter file.

Note that following the above instructions, it means that for each signal path, there will be 16 s2p files. For each trigger path, there will be 112 files. This means that for each temperature, there will 128×4 (channels) $= 512$ files, and therefore, 512×6 (temperatures) $= 3,072$ files in total.

4.3 Measurements at OSU

4.3.1 Procedure

At OSU, we handle this by utilizing the Astroparticle Lab Libraries (`https://github.com/osu-particle-astrophysics/astroparticlelab-libraries`, contact OSU for access) which are a set of python wrapper classes that use the pyVISA, Modbus, and serial libraries. At OSU, the execution of the measurement code can be done by executing `sudo python meas_arafe_sparameter.py` on the Cherenkov server (which you can access through the Caesar server). It is listed in appendix A.

The code has four important control functions for measurement devices in the Beatty-Connolly lab:

- Control of the Watlow F4 thermal chamber via Modbus.
- Control of the Agilent E5062A Vector Network Analyzer (VNA) via pyVISA/GPIB
- Control of the Agilent 34970 RF Multiplexer via pyVISA/GPIB
- Control of the ARAFE slave tester via serial

A few important notes. In order to take full s2p files with the VNA, you would normally use the included "Touchstone" VBA program that comes with the VNA. However, the E5062A can only call a VBA file from the GUI interface. So, to automate the taking of the s-parameters, we take the gain and phase separately, and merge them into s2p files afterward. This merging described in section 4.3.2.

In order to take all four RF channels (sig or trig) at once, we use the Agilent 34970 RF mutiplexer. To remove the effects of this device on the S2P measurement, we perform a two-port calibration on the VNA.

Saving all of the associated files for all temperatures and channels at all of the attenuations described in section 4.2 (including their group delays) will consume approximately 3.5GB. Because the hard drive storage on the VNA is small, we save the files to a flash drive that has sufficient storage space.

4.3.2 Data Processing

To merge the RAW CSV files into the compact S2P files, we use the python programs listed in appendix and . They are executed by running `python sig_merge_csv_to_s2p.py` and `python trig_merge_csv_to_s2p.py`. The code uses the Pandas libraries to merge the files, and are self documenting.

References

[1] RFSA3713 Variable Attenuator Datasheet. http://www.rfmd.com/store/downloads/dl/file/id/30109/rf

A Appendix: OSU Code to Take S-Parameters

```
#!/usr/bin/python

import serial
import sys
import time
import io
import datetime
from astroparticlelab import watlowf4
from astroparticlelab import E5062A
from astroparticlelab import DA34970

TherCham = watlowf4.WatlowF4('/dev/ttyS0', 1) #make a thermal chamber
Muxer = DA34970.DA34970A() #make a multiplexer
NWAnl = E5062A.E5062A() #make a network analyzer

port1 = '/dev/ttyECOM'
port0 = '/dev/ttySCOM'
ser=serial.Serial()

settle_time = 60*60*1.5 #two hour settle time
measurement_time = 60*10 #wait time between setting the att and making measurement
wait_a_little_longer = 60*20 #if the chamber hasn't settled yet, we'll wait this additional time

#SigOrTrg= "SIG" #sig or trig measurement
SigOrTrg= "TRG"
ARAFE = 6010 #which ARAFE RF board are we measuring


def main():

        #define our temperature points, in an order that calls for oscillations between highs and lows
        TempPoints = [-20,30,-10,20,0,10]
        #TempPoints = [30,-10,20,0,10]
        #TempPoints=[20] #done temp points
        #define the channels we want to loop over
        Channels = [0,1,2,3]
        #define the attenuator settings we want to loop over
        #AttenuatorSettings = [0,10,20,30,60,90,120]
        #AttenuatorSettings=[0,9]
        #AttenuatorSettings=[0,4,8,12,16,20,24,28,32,36,40,44,48,52,56,60,64,68,72,76,84,88,92,96,100,104,108,112,116,120,1
        AttenuatorSettings=[0,8,16,24,32,40,48,56,64,72,80,88,96,104,112,120]
        AttenuatorSettings2=[0,16,32,48,64,80,96]
        #AttenuatorSettings2=[0,16,32]
        #AttenuatorSettings2=[120]
        #take a measurement every 1 dB is basically what the above is doing

        #make a directory for this quad, signal vs trig

        #now, iterate over temperatures
        for temp in TempPoints:


                stringtemp = "tbdtemp"
                if temp <0 :
                        stringtemp = "N%d" %abs(temp)
                if temp ==0 :
                        stringtemp = "P0%d" %temp
                if temp>0:
                        stringtemp = "P%d" %temp

                directory = "G:\ARAFE%d%s%s" %(ARAFE, SigOrTrg,stringtemp)
                NWAnl.make_dir(directory)

                #start this thermal run by resetting all of the attenuators
                for chan in Channels:
                        set_attenuator("SIG",chan,0)
                        set_attenuator("TRG",chan,0)

                print "On temperature point %d" %temp

                #ready the thermal chamber
                ready_chamber(temp)

                #in the case of the signal attenuator, we just need to measure that one
                if SigOrTrg=="SIG":
                        for setting in AttenuatorSettings:
                                print "On attenuator setting %d" %setting
                                #set all of the channels
                                for chan in Channels:
                                        print "setting channel %d signal attenuator to %d" %(chan, setting)
                                        set_attenuator("SIG",chan,setting)
                                time.sleep(60)
                                for chan in Channels:
                                        print "Working on sparams for signal chan %d" %chan
                                        collect_sparameters(directory, SigOrTrg,chan, setting,0, temp)
                                        #fill in a zero for the trigger att, because we don't care its setting
                elif SigOrTrg=="TRG":
                        #first loop over signal attenuator settings
```

```
                        for setting in AttenuatorSettings2: #define a different number of points for the sig att; too many
                                for chan in Channels:
                                        print "setting channel %d signal attenuator to %d" %(chan, setting)
                                        set_attenuator("SIG",chan,setting)
                                #second loop over trigger attenuator settings
                                for setting2 in AttenuatorSettings:
                                        print "working on sig att setting %d and trig att setting %d" %(setting, setting2)
                                        for chan2 in Channels:
                                                print "setting channel %d trigger attenuator to %d" %(chan2, setting2)
                                                set_attenuator("TRG", chan2, setting2)
                                        time.sleep(20)
                                        for chan2 in Channels:
                                                print "Working on sparams for trigger chan %d" %chan2
                                                collect_sparameters(directory, SigOrTrg,chan2, setting, setting2,temp)

        print "Done!"


def ready_chamber(temp):
        #for every new temperature point, purge the chamber
        #45 seconds should do it
        TherCham.open_GN2()
        time.sleep(45)
        TherCham.close_GN2()

        time.sleep(5)
        #now, set our temperature point
        print "Setting thermal chamber to %d" %temp
        TherCham.set_sp(temp)
        time.sleep(5)

        #and wait for it to come to temperature
        #if(temp!=30):
        time.sleep(settle_time) #just this once, skip this step for -20

        #check to see if we've come to temperature
        #if not, wait another 20 minutes
        temp_meas = TherCham.get_sp()
        time.sleep(5)
        if temp_meas > temp+2 or temp_meas < temp-2:
                print "Temperature is %d but goal is %d, so wait a little longer." %(temp_meas, temp)
                time.sleep(wait_a_little_longer)

        #now, we just gotta move on


def collect_sparameters(directory, SigOrTrg,chan, att_sig, att_trg, temp):

        print "Collecting s parameters on channel %d" %chan

        #pick which RF multiplexer to close
        rf_in = 0
        rf_out = 1

        if chan==0:
                rf_in = 211
                rf_out = 221
        if chan==1:
                rf_in = 212
                rf_out = 222
        if chan==2:
                rf_in = 213
                rf_out = 223
        if chan==3:
                rf_in = 214
                rf_out = 224

        #close that channel
        print "close r in muxer to %s" %rf_in
        Muxer.set_RFMUX_close(rf_in)
        time.sleep(2)
        print "close r in muxer to %s" %rf_in
        Muxer.set_RFMUX_close(rf_out)
        time.sleep(2)

        Parameters = ["S11", "S22", "S12", "S21"]
        for param in Parameters:
                get_files(directory, SigOrTrg,chan, att_sig, att_trg, temp, param)


def get_files(directory, SigOrTrg,chan, att_sig, att_trg, temp, param):
        #select the s parameter we are working with
        NWAnl.set_sparam(1,1,param)
        time.sleep(1)
        SaveName = "default.csv" #default filename

        #set up the signal attenuator string
        stringatt_sig = "tbdatt_sig"
        if(att_sig<10):
                if att_sig >0:  stringatt_sig = "SIG00%d" %att_sig
                else: stringatt_sig = "SIG000"
```

```
        elif att_sig<100:
                stringatt_sig = "SIG0%d" %att_sig
        elif att_sig >=100:
                stringatt_sig = "SIG%d" %att_sig

        #set up the trigger attenuator string
        stringatt_trg = "tbdatt_trig"
        if(att_trg<10):
                if att_trg >0:  stringatt_trg  = "TRG00%d" %att_trg
                else: stringatt_trg  = "TRG000"
        elif att_trg<100:
                stringatt_trg = "TRG0%d" %att_trg
        elif att_trg >=100:
                stringatt_trg = "TRG%d" %att_trg

        stringtemp = "tbdtemp"
        if temp <0 :
                stringtemp = "N%d" %abs(temp)
        if temp ==0 :
                stringtemp = "P0%d" %temp
        if temp>0:
                stringtemp = "P%d" %temp

        stringsigtrig = "tbdsigtrig"
        if SigOrTrg == "SIG":
                stringsigtrig = "SIG%d" %chan
        if SigOrTrg == "TRG":
                stringsigtrig = "TRG%d" %chan


        #get the logmag parameter
        NWAnl.set_trace_type(1,"MLOG") #put the logmag "MLOG" option on screen
        SaveName = "%s/ARAFE%d_%s_%s_%s_T%s_%sMag" %(directory,ARAFE,stringsigtrig,stringatt_sig,stringatt_trg,stringtemp,ps
        NWAnl.save_trace_csv(SaveName)  #save the csv file
        time.sleep(1)

        #repeat for the phase
        NWAnl.set_trace_type(1,"PHAS") #put the phase "PHAS" option on screen
        SaveName = "%s/ARAFE%d_%s_%s_%s_T%s_%sPhs" %(directory,ARAFE,stringsigtrig,stringatt_sig,stringatt_trg,stringtemp,ps
        NWAnl.save_trace_csv(SaveName)  #save the csv file
        time.sleep(1)

        #repeat for the group delay
        NWAnl.set_trace_type(1,"GDEL") #put the group delay "GDEL" option on screen
        SaveName = "%s/ARAFE%d_%s_%s_%s_T%s_%sGdl" %(directory,ARAFE,stringsigtrig,stringatt_sig,stringatt_trg,stringtemp,ps
        NWAnl.save_trace_csv(SaveName)  #save the csv file
        time.sleep(1)



def set_attenuator(path, att, setting): #whether sig/trig, attenuation, and its setting
        dt_time = datetime.datetime.now()
        print "%s: Setting %s %d to %d" %(datetime.datetime.now(), path, att, setting)
        cmd="fail"
        if path == "SIG":
                cmd = "sig %d %d \n\r" %(att, setting)
        if path == "TRG":
                cmd = "trig %d %d \n\r" % (att, setting)
        check_connection()
        ser.write(cmd)
        line = ser.readline()
        while len(line) >0:
                print line.rstrip()
                line = ser.readline()
        time.sleep(1)


def check_connection():
        #port0='/dev/ttyUSB0'
        #port1='/dev/ttyUSB1'
        #ser.baudrate=9600
        #ser.timeout=1
        try:
                ser.port=port0
                #try to open the port
                ser.open()
                print "Opening USB successful"
                #if opening will fail, it's either going to fail because the port is already open or because we need to swi
        except:
                if ser.isOpen():
                        print "USB port is open, proceed"
                else:
                        print "Either not port zero, or the old  port has closed, open a new one"
                        ser.port=port1
                        ser.open()

        #fine, let's be a real stickler here
        #first, just assign thte port above
        #then, close it, set up the way we want it to talk, and re-open it
        #so dumb, so so dumb
        ser.close()
        ser.baudrate=9600
```

```
        ser.timeout=1
        ser.open()
        ser.setDTR(0)
        print "using serial line %s" %ser.port

        #To be more verbose, we have to verify that the port actually exists before we can do anything to it
        #The cleanest way to do that is to just open it, but that would initiative it with the wrong baudrate, etc
        #So, first we open it, then close it, then set the baudrates and such, and then finally re-open it
        #See what I mean about dumb?


main()
```

B Appendix: Code to Merge CSV Signal Files

```
# -*- coding: utf-8 -*-
import pandas as pd
#Need the pandas library

Board = "6003"
SigOrTrig = "SIG"

def main():

        #get the first file of interest

        Channels = [0,1,2,3] #what channels?
        #AttenuationSettings =  ["010","020","030","060","090","120"] #what attenuation settings?
        #Temperatures = ["N10", "P00", "P10", "P20", "P30"] #what temperatures?
        AttenuationSettingsSig =  ["000","008","016","024","032","040","048","056","064","072","080","088","096","104","112"
        att_trig="000"
        Temperatures = ["P30"] #what temperatures?

        for chan in Channels:
                for att_sig in AttenuationSettingsSig:
                        for temp in Temperatures:
                                #print "SOURCE/ARAFE%s_%s%d_SIG%s_TRG%s_T%s_S11MAG.CSV"%(Board, SigOrTrig, chan, att_sig, att
                                S11MagFile = "SOURCE/ARAFE%s_%s%d_SIG%s_TRG%s_T%s_S11MAG.CSV"%(Board, SigOrTrig, chan, att_si
                                S11PhaseFile = "SOURCE/ARAFE%s_%s%d_SIG%s_TRG%s_T%s_S11PHS.CSV"%(Board, SigOrTrig, chan, att
                                S11Mag = pd.read_csv(S11MagFile,skiprows=3) #skip the first two rows
                                S11Mag.drop(S11Mag.columns[2], axis=1,inplace=True) #drop the third column
                                S11Phs = pd.read_csv(S11PhaseFile,skiprows=3)
                                S11Phs.drop(S11Phs.columns[2], axis=1,inplace=True)
                                S11Mag.columns=['Frequency','|S11|']
                                S11Phs.columns=['Frequency','<S11']

                                S12MagFile = "SOURCE/ARAFE%s_%s%d_SIG%s_TRG%s_T%s_S12MAG.CSV" %(Board, SigOrTrig, chan, att_s
                                S12PhaseFile = "SOURCE/ARAFE%s_%s%d_SIG%s_TRG%s_T%s_S12PHS.CSV" %(Board, SigOrTrig, chan, at
                                S12Mag = pd.read_csv(S12MagFile,skiprows=3) #skip the first two rows
                                S12Mag.drop(S12Mag.columns[2], axis=1,inplace=True) #drop the third column
                                S12Phs = pd.read_csv(S12PhaseFile,skiprows=3)
                                S12Phs.drop(S12Phs.columns[2], axis=1,inplace=True)
                                S12Mag.columns=['Frequency','|S12|']
                                S12Phs.columns=['Frequency','<S12']

                                S21MagFile = "SOURCE/ARAFE%s_%s%d_SIG%s_TRG%s_T%s_S21MAG.CSV" %(Board, SigOrTrig, chan, att_s
                                S21PhaseFile = "SOURCE/ARAFE%s_%s%d_SIG%s_TRG%s_T%s_S21PHS.CSV" %(Board, SigOrTrig, chan, at
                                S21Mag = pd.read_csv(S21MagFile,skiprows=3) #skip the first two rows
                                S21Mag.drop(S21Mag.columns[2], axis=1,inplace=True) #drop the third column
                                S21Phs = pd.read_csv(S21PhaseFile,skiprows=3)
                                S21Phs.drop(S21Phs.columns[2], axis=1,inplace=True)
                                S21Mag.columns=['Frequency','|S21|']
                                S21Phs.columns=['Frequency','<S21']

                                S22MagFile = "SOURCE/ARAFE%s_%s%d_SIG%s_TRG%s_T%s_S22MAG.CSV" %(Board, SigOrTrig, chan, att_s
                                S22PhaseFile = "SOURCE/ARAFE%s_%s%d_SIG%s_TRG%s_T%s_S22PHS.CSV" %(Board, SigOrTrig, chan, at
                                S22Mag = pd.read_csv(S22MagFile,skiprows=3) #skip the first two rows
                                S22Mag.drop(S22Mag.columns[2], axis=1,inplace=True) #drop the third column
                                S22Phs = pd.read_csv(S22PhaseFile,skiprows=3)
                                S22Phs.drop(S22Phs.columns[2], axis=1,inplace=True)
                                S22Mag.columns=['Frequency','|S22|']
                                S22Phs.columns=['Frequency','<S22']

                                merged = S11Mag.merge(S11Phs, on='Frequency').merge(S21Mag, on='Frequency').merge(S21Phs, o
                                merged.rename(columns={'Frequency':'#Frequency'}, inplace=True)

                                OutputName = "S2P/ARAFE%s_%s%d_SIG%s_TRG%s_T%s.s2p" %(Board, SigOrTrig, chan, att_sig, att_tri
                                merged.to_csv(OutputName,index=False, sep='\t')

#actually execute the main function
main()
```

C Appendix: Code to Merge CSV Trigger Files

```
# -*- coding: utf-8 -*-
import pandas as pd
#Need the pandas library

Board = "6003"
SigOrTrig = "TRG"

def main():

        #get the first file of interest

        Channels = [0,1,2,3] #what channels?
        #AttenuationSettings =  ["010","020","030","060","090","120"] #what attenuation settings?
        #Temperatures = ["N10", "P00", "P10", "P20", "P30"] #what temperatures?
        AttenuationSettingsTrig =  ["000","008","016","024","032","040","048","056","064","072","080","088","096","104","11
        AttenuationSettingsSig =  ["000","016","032","048","064","080","096"] #what attenuation settings?
        Temperatures = ["P30"] #what temperatures?

        for chan in Channels:
                for att_trig in AttenuationSettingsTrig:
                        for att_sig in AttenuationSettingsSig:
                                for temp in Temperatures:
                                        S11MagFile = "SOURCE/ARAFE%s_%s%d_SIG%s_TRG%s_T%s_S11MAG.CSV"%(Board, SigOrTrig, cha
                                        S11PhaseFile = "SOURCE/ARAFE%s_%s%d_SIG%s_TRG%s_T%s_S11PHS.CSV"%(Board, SigOrTrig,
                                        S11Mag = pd.read_csv(S11MagFile,skiprows=3) #skip the first two rows
                                        S11Mag.drop(S11Mag.columns[2], axis=1,inplace=True) #drop the third column
                                        S11Phs = pd.read_csv(S11PhaseFile,skiprows=3)
                                        S11Phs.drop(S11Phs.columns[2], axis=1,inplace=True)
                                        S11Mag.columns=['Frequency','|S11|']
                                        S11Phs.columns=['Frequency','<S11']

                                        S12MagFile = "SOURCE/ARAFE%s_%s%d_SIG%s_TRG%s_T%s_S12MAG.CSV" %(Board, SigOrTrig, ch
                                        S12PhaseFile = "SOURCE/ARAFE%s_%s%d_SIG%s_TRG%s_T%s_S12PHS.CSV" %(Board, SigOrTrig,
                                        S12Mag = pd.read_csv(S12MagFile,skiprows=3) #skip the first two rows
                                        S12Mag.drop(S12Mag.columns[2], axis=1,inplace=True) #drop the third column
                                        S12Phs = pd.read_csv(S12PhaseFile,skiprows=3)
                                        S12Phs.drop(S12Phs.columns[2], axis=1,inplace=True)
                                        S12Mag.columns=['Frequency','|S12|']
                                        S12Phs.columns=['Frequency','<S12']

                                        S21MagFile = "SOURCE/ARAFE%s_%s%d_SIG%s_TRG%s_T%s_S21MAG.CSV" %(Board, SigOrTrig, ch
                                        S21PhaseFile = "SOURCE/ARAFE%s_%s%d_SIG%s_TRG%s_T%s_S21PHS.CSV" %(Board, SigOrTrig,
                                        S21Mag = pd.read_csv(S21MagFile,skiprows=3) #skip the first two rows
                                        S21Mag.drop(S21Mag.columns[2], axis=1,inplace=True) #drop the third column
                                        S21Phs = pd.read_csv(S21PhaseFile,skiprows=3)
                                        S21Phs.drop(S21Phs.columns[2], axis=1,inplace=True)
                                        S21Mag.columns=['Frequency','|S21|']
                                        S21Phs.columns=['Frequency','<S21']

                                        S22MagFile = "SOURCE/ARAFE%s_%s%d_SIG%s_TRG%s_T%s_S22MAG.CSV" %(Board, SigOrTrig, ch
                                        S22PhaseFile = "SOURCE/ARAFE%s_%s%d_SIG%s_TRG%s_T%s_S22PHS.CSV" %(Board, SigOrTrig,
                                        S22Mag = pd.read_csv(S22MagFile,skiprows=3) #skip the first two rows
                                        S22Mag.drop(S22Mag.columns[2], axis=1,inplace=True) #drop the third column
                                        S22Phs = pd.read_csv(S22PhaseFile,skiprows=3)
                                        S22Phs.drop(S22Phs.columns[2], axis=1,inplace=True)
                                        S22Mag.columns=['Frequency','|S22|']
                                        S22Phs.columns=['Frequency','<S22']

                                        merged = S11Mag.merge(S11Phs, on='Frequency').merge(S21Mag, on='Frequency').merge(S
                                        merged.rename(columns={'Frequency':'#Frequency'}, inplace=True)

                                        OutputName = "S2P/ARAFE%s_%s%d_SIG%s_TRG%s_T%s.s2p" %(Board, SigOrTrig, chan,att_sig
                                        merged.to_csv(OutputName,index=False, sep='\t')

#actually execute the main function
main()
```