September 1, 2017

Testing of the ANNIE Central Card for a 5-15GSa/s PSEC4 Readout System

John Podczerwinski
Horatio Li

1 Introduction

1.1 The Large Area Picosecond Photo-Detector (LAPPD)

The LAPPD is a 20cm-squared MCP-based photo-detector capable of providing timing resolution under 10psec and a spatial resolution of $700\mu m$. It consists of a photocathode window, two MCP's, and a set of 30 anode strips for readout.

The large area and high degree of timing and spatial resolution make the LAPPD useful for many applications. In the realm of high energy physics, among other applications, LAPPDs would be useful for track reconstruction in large Cherenkov detectors, and for extracting quark information in collider experiments. The LAPPD is also being used in the development of medical imaging techniques. [1]

1.2 The First Generation PSEC4 Readout System

In order to read out fast waveforms, switched capacitor array (SCA) analog memory was developed. SCAs allow for waveforms to be sampled at a high rate, but at the expense of slow readout times. Such technology is well suited for high energy physics, where fast sampling rates allow for precise arrival time measurements, and a bit of dead time is acceptable [8][7].

In order to read out the fast waveforms produced by the LAPPD, new waveform sampling technologies had to be developed. In 2011, University of Chicago graduate student Eric Oberla developed the PSEC4 ASIC, an SCA based, 6 channel, 15GSa/s, 1.5GHz bandwidth waveform digitizing chip [2]. Oberla also designed a front end board called the ACDC, on which 5 of these PSEC4 chips are mounted, giving 30 channels of PSEC4 per board.

In the first generation readout system, the ACDC board is connected via ethernet to a control card (CC), designed by Mircea Bogdan of the University of Chicago. the CC can control up to four slave boards via ethernet, and serves as a communication point between the computer and the electronics setup. [4]

1.3 Commissioning the ANNIE Central Card

In 2015, a new central card known as the ANNIE central card (ACC) was developed by Mircea Bogdan at the University of Chicago for the accelerator neutron neutrino interaction experiment (ANNIE). The ACC performs the same function as the CC, but is able to handle up to 8 slaves and transfer data at faster rate [3].

During the summer, the main goal of our group was to readout 30 channels of PSEC4 using an ACC and ACDC. Such a system was desired as it would be a first step in integrating the superior ACC into ANNIE's data acquisition (DAQ) system. In addition, the ability to readout thirty channels would be useful for the testing of the Margherita 2 LAPPD fabrication system.

2 Overview of the DAQ System

The data acquisition (DAQ) system we worked on over the summer consisted of 5 PSEC4 ASICs mounted on a front end board called the ACDC, and a back end board called the ANNIE Central Card (ACC). All data from and instructions to the DAQ were transfered between the ACC and a computer via USB. See figure 1 for a picture of the DAQ.

The analog to digital converter (ADC) used in this DAQ is the PSEC4 ASIC, an integrated circuit designed by Eric Oberla of the University of Chicago. The PSEC4 has six input channels, samples waveforms at a rate between 5 and 15GSa/s, and has a bandwidth of 1.5GHz. The PSEC4 also has a 256 sample buffer, giving a buffer depth on the order of tens of nanoseconds[2]. See figure 2 for a close up view of a PSEC4 chip.

Another important component of this DAQ is the ACDC. The ACDC is a front end board holding 5 PSEC4 chips, or equivalently, 30 channels of PSEC4. Signals to be digitized are supplied to the PSEC4 chips either via SMA or ribbon cable inputs mounted on the ACDC. On an event, the digitized signals are transfered from the PSEC4 to the RAM of the ACDC's field programmable gate array (FPGA). This data is then transfered from the ACDC's FPGA to the ACC via ethernet.

The back end board of this DAQ is the ACC. In this specific case, the ACC serves to control one ACDC board by sending commands over the LVDS lines. More generally, an ACC can directly control up to 8 ACDC boards. Depending on the firmware installed, it may also control 8 slave ACCs.

The FPGAs on the ACDC and ACC both run on firmware designed by Eric Oberla, although minor modifications had to be made to the ACDC firmware to make it compatible with the 25MHz clock. The DAQ also runs on a set of C++ codes written by Oberla, which process data from the DAQ and send instructions to it.

2.1 Establishing Communication Between the ACC and ACDC

In this DAQ, the ACC and ACDC communicate via 8 LVDS lines. Data from the ACDC is transferred serially to the ACC via 2 of these LVDS lines. Going the other way, data is transferred serially from the ACC to the ACDC over one of the LVDS lines. In order for this communication scheme to work, the LVDS signals themselves must be at the correct levels, and the ACC and ACDC transceivers must operate on

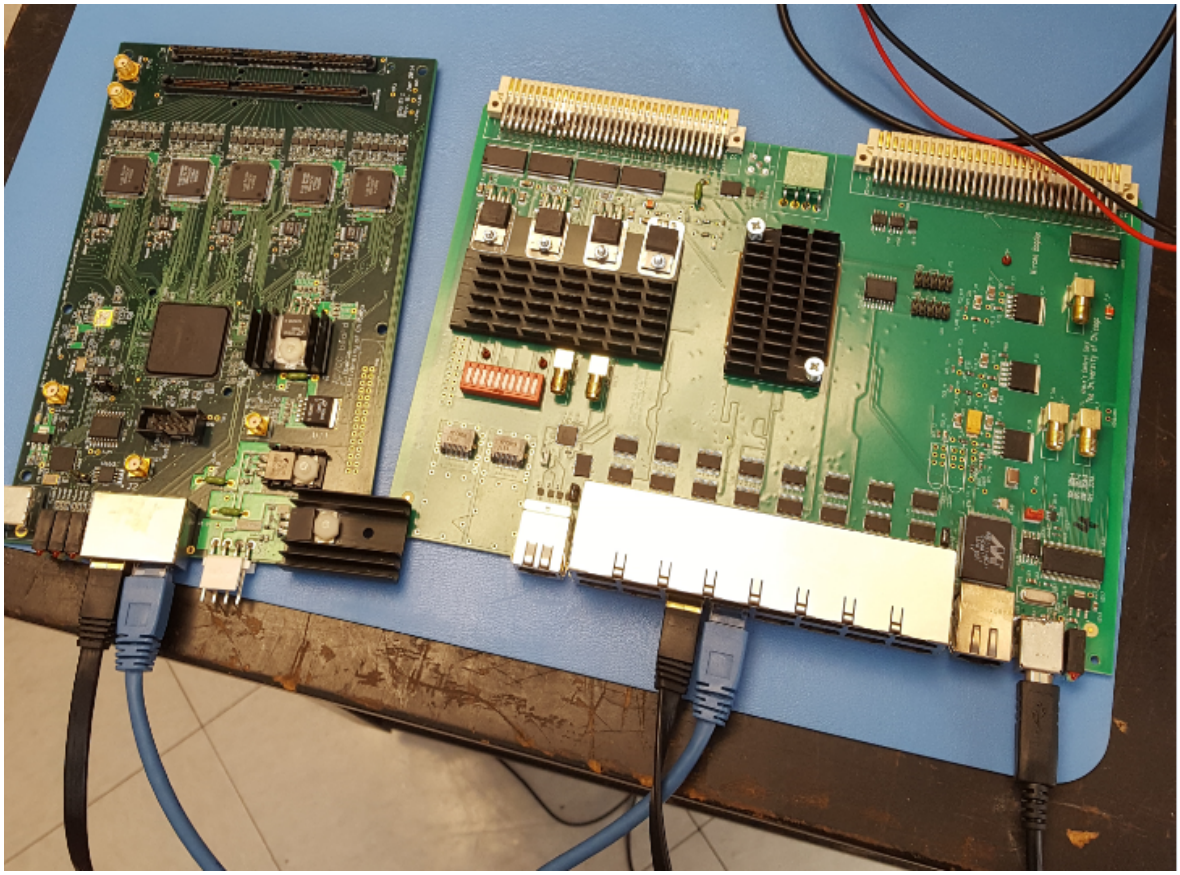

Figure 1: The DAQ we spent the summer working with. On the left is the front end ACDC board with 5 PSEC4 chips mounted on it. On the right is the ACC, with a USB cable running from it to a computer.

Figure 2: A close up view of the PSEC4 chip, the ADC used in the DAQ.

the same clock. Neither of these requirements were satisfied on our DAQ, so we had to do some fixing.

In our DAQ, the FPGAs (and the transceivers) on the ACC and ACDC run on the same 25MHz LVDS clock signal, produced by an oscillator on board the ACC, as seen in figure 3. One output of the oscillator is sent to the FPGA on board the ACC [5], while another output from this oscillator travels over an LVDS line to a clock generator chip on the ACDC. This clock generator chip then generates 5 pairs of clocks. On four of these pairs, one output clock is sent to a PSEC4 chip, while the other is connected to ground. On the last pair, one clock is sent to a PSEC4 chip, while the other goes to the FPGA on board the ACDC [6].

In order for serial communication between the ACC and ACDC to work, we needed the clock generator chip to supply a clock to the FPGA on board the ACDC. Moreover, this clock had to be at the same frequency as and in phase with the clock coming from the ACC. Probing the outputs of the clock generator, we found that it was producing clocks at a variety of different frequencies with no phase relationship to the input.

One cause for this issue was in the settings of the clock generator, which were far from what we needed. The settings on the chip were such that it expected to receive a clock of a different logic type than what what was coming from the ACC. The chip was also set to produce its output clocks using a phase locked loop (PLL), which contains its own oscillator. These factors made it so that the clocks coming out of the generator had no relationship with the inputs, as they were being produced by a PLL without a reference. In order to fix this problem, we made changes to the clock generator control registers through the ACDC firmware. Our changes made it so that the clock generator looked for LVDS signals at its inputs, and operated in a fan-out configuration rather than using the PLL.

After making this change, the clock generator still was not working. In fact, the output clocks were not oscillating at all. This seemed to indicate that the generator chip still was not seeing the clock from the ACC. It turns out that this was because the DC component of the input clock was sitting well below the minimum value accepted by the clock generator. One reason for this problem was that the grounds on the ACC and ACDC were not connected. In order to fix this, we connected the shield of the ACC's RJ45 ports to ground, and connected the grounds of the two power supplies. We also swapped out the ethernet cable that carried the clock, which raised the DC component. These changes raised the DC component of the clock to a good value. Upon probing the clock generator chip, we found it was producing clocks with the same frequency and phase as the input clock.

Once the clock issue was sorted out, we were able to use software developed by Eric Oberla for the old central card to communicate between the ACC and ACDC.

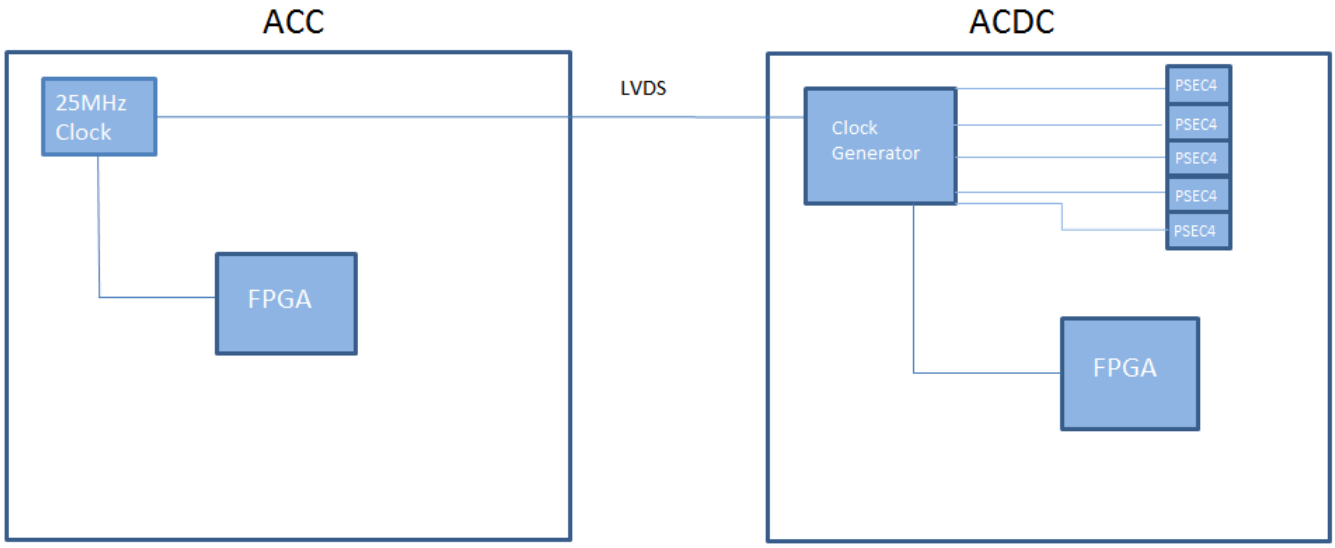

Figure 3: A simplified diagram of the clocking system used on the DAQ.

2.2 Testing Waveform Sampling and Readout

After solving the clock and LVDS communication issues, we began work on testing the ACC/ACDC based DAQ with input signals. In particular, we used a function generator to supply a 120MHz sine wave with 400mV peak to peak amplitude through the SMA calibration input ACDC boards [6], and got mixed results. On 3 of the 4 boards we tested, we found that one PSEC4 chip would sample at the wrong rate, and that another would report noise. these boards are labeled ACDC2, ACDC5, and ACDC10. This behavior can be seen in Figure 4. The chip with the wrong sampling rate is the same across all 3 boards, while the dead chip varies from board to board. On the fourth board tested, which we'll call ACDC11, four of the PSEC4 chips seem to work appropriately, while one is dead.

2.2.1 Bizarre sampling rates

Three out of four of the ACDC boards had one of their PSEC chips sampling at a rate much faster than the rest, as seen in figure 4. It turns out that the root of this problem was in voltage controlled delay line (VCDL) of the offending chip. In operation, the PSEC4 sampling rate is automatically set by the write clock it receives from the clock generator chip as shown in figure 3. When the PSEC4 receives the write clock, the VCDL along with a phase comparator and charge pump act as a delay locked loop (DLL), as in figure 5. This delay locked loop locks the write clock at a one cycle latency, ensuring that sampling occurs correctly [2].

We discovered that the output of the DLL on any offending chip was out of phase

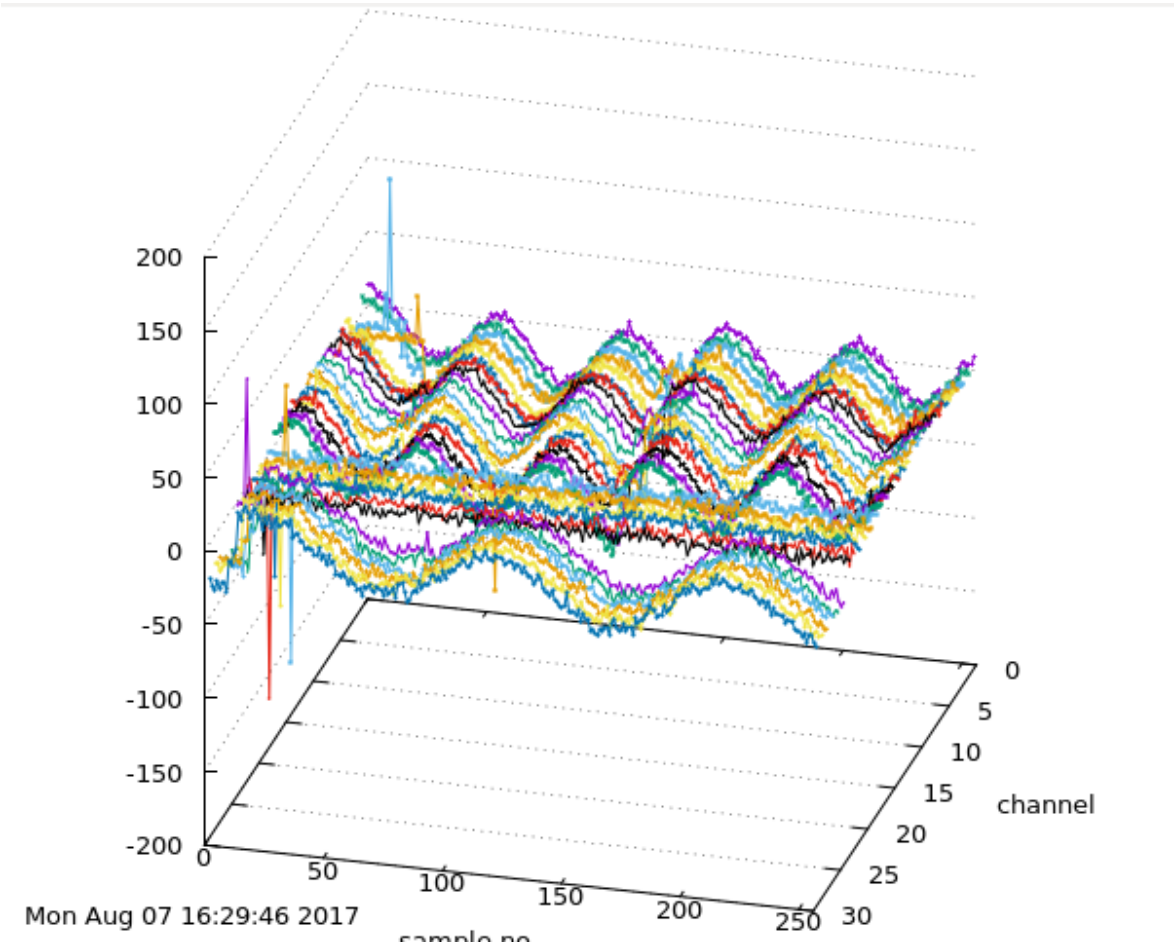

Figure 4: An example of the odd behavior we encountered initially. The x-axis gives the sample number, the y-axis gives the channel number, and the z-axis gives ADC counts.

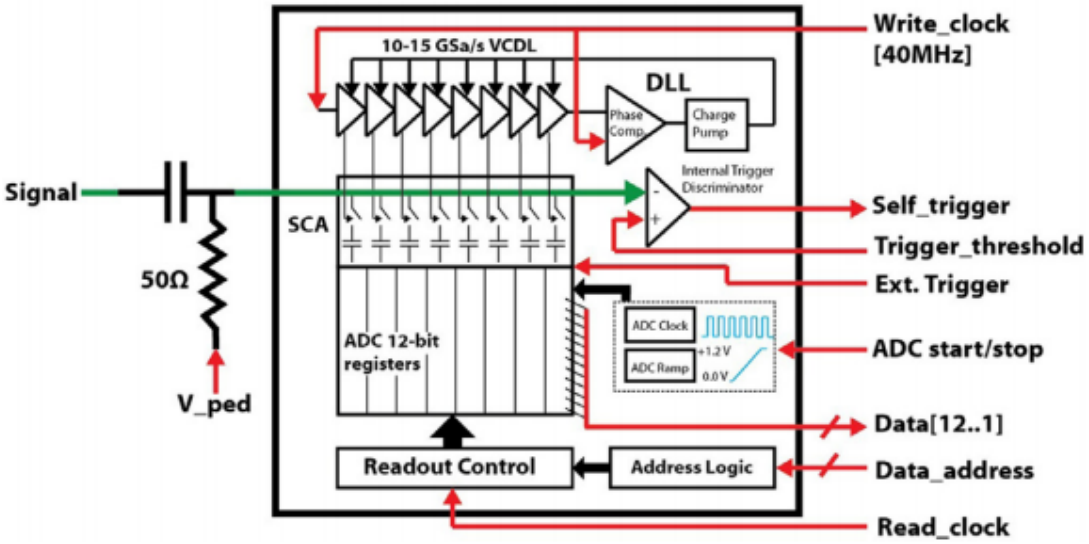

Figure 5: An illustration of what goes on inside of a PSEC4 chip. The sampling rate is set by a DLL, which takes the write clock as a reference. Courtesy of E. Oberla.

with the write clock supplied to it by the clock generator chip. In other words, the DLL was unable to lock onto the write clock we were supplying it with. Since the DLL on the PSEC4 chip controls sampling, this was a likely culprit for the odd rate we saw on our plots.

The only solution found for this issue was cycling power on the ACC several times. When power on the ACC is cycled, the write clock provided to the PSEC4 chip is cycled, giving the PSEC4 several attempts to lock onto the write clock. Often after several attempts, the DLL will lock onto the write clock, and the sampling rate will be fixed, as seen in figure 6. However, this technique is unreliable.

We also observed that these DLL locking issues do not occur when the PSEC4 is supplied with a higher frequency write clock. When supplied with $34MHz$ and $40MHz$ write clocks, the same PSEC4 chips that couldn't lock onto $25MHz$ were able to lock consistently. This indicates that the chips themselves are not broken, but that $25MHz$ may be too slow a write clock for the DLLs to lock onto them reliably.

2.2.2 Dead Channels

We also found that all four boards had one dead PSEC4 chip. When testing with a 25MHz write clock, three of these boards (the same ones with the odd sampling rate), show noise on their dead channels, as seen in figure 4. The dead channels on the fourth board, ACDC11, are not noisy at all. Testing was done on ACDC2 and ACDC10 at the University of Chicago, while ACDC5 was sent to Iowa State.

Probing the DLL output on the noisy chips of ACDC2 and ACDC10 showed problems. On ACDC2, The DLL output would be present, but would disappear as soon as

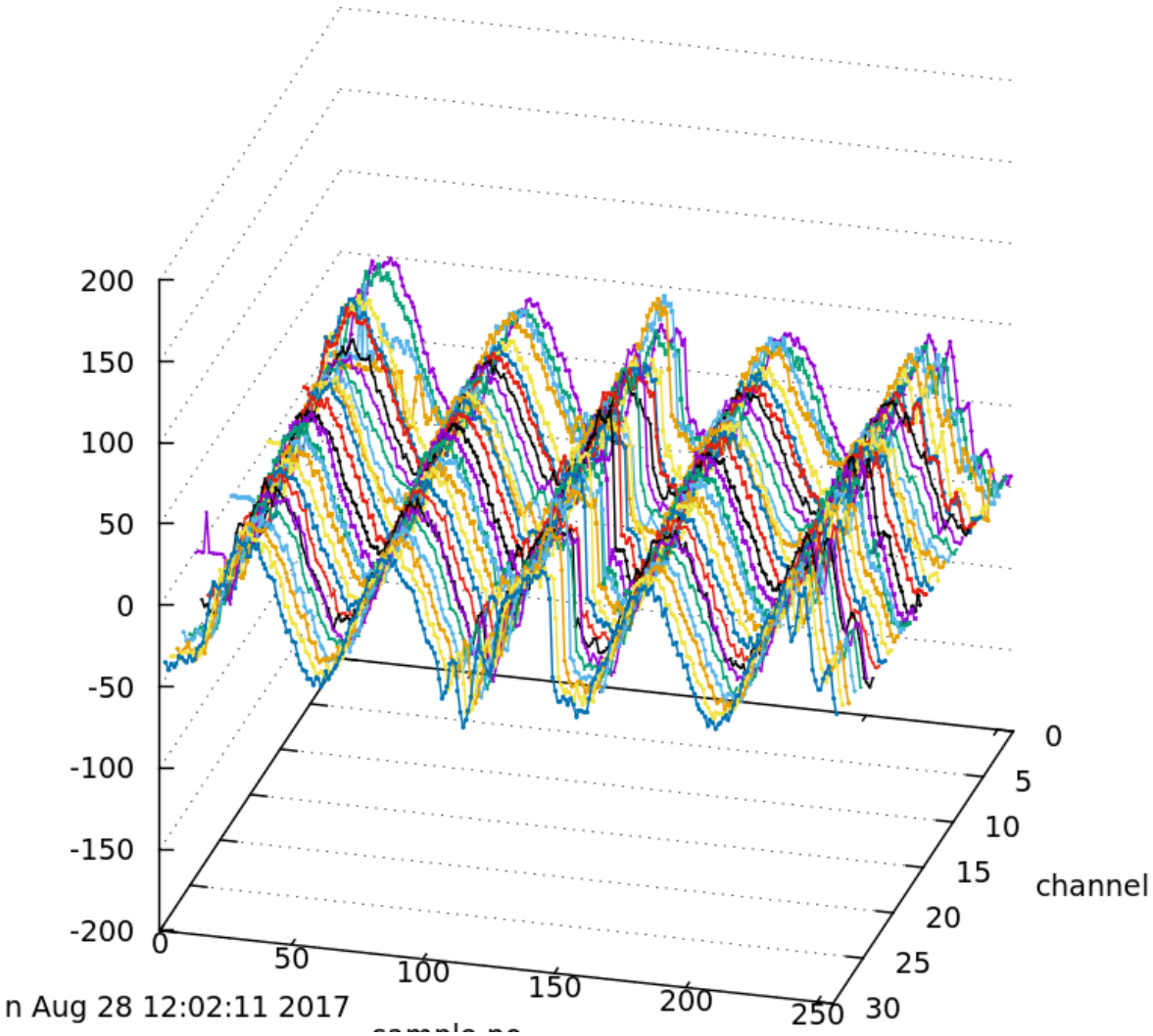

Figure 6: 120MHz sine wave read by an ACDC when the all PSEC4 chips are locked on the write clock. Before cycling ACC power, the sampling rate on channels 30 down to 25 looked like those in figure 4.

we tried to read data from it. The DLL output on ACDC10 was just a flat line with no oscillation whatsoever. This seemed to indicate that the PSEC4 chips might be dead, and so we swapped them out for new ones. On ACDC10, we found that the first new chip was able to lock onto the clock, but produced noisy signals. Investigation showed that it had been soldered on incorrectly. After that, we swapped the PSEC4 chip out again. This second chip was able to lock onto the write clock and work correctly, as seen in figure 6. On ACDC2, the first new PSEC4 chip showed no output on its DLL no matter what frequency clock it was given. We took this to mean that there also soldering mistakes on this chip, and replaced it. The new PSEC4 chip was unable to lock on to 25MHz, but could lock reliably on to a 34MHz write clock. ACDC5, which also showed dead channels, was mailed to Iowa State. At Iowa State, ACDC5 was tested with an old central card (which has a write clock of 40MHz), and it was found to work well.

The fact that ACDC5 worked with a 40MHz write clock, without any hardware modifications whatsoever, indicates that the problem is caused by incompatibility with the write clock rather than broken PSEC4 chips. This is further supported by the fact that all the chips were found to lock perfectly well when supplied with a higher frequency clock. However, since some PSEC4 chips work with 25MHz, it cannot be said that the 25MHz doesn't work at all. This mix of behavior indicates that 25MHz is a threshold value for the PSEC4 chips as they are on the ACDC boards. Since this value is most likely a threshold, there may be a way to make changes to the ACDC boards to get them to lock on 25MHz reliably. In particular, Eric Oberla has suggested that we replace the VCDL control voltage capacitors with larger ones. This would reduce noise and help with locking. If we are unable to find a fix that works, then the 25MHz clock will be replaced by one with a higher frequency. However, moving to a faster clock is undesirable as it shortens the buffer length.

2.3 Software and Firmware

The FPGAs on the ACC and the ACDC both run on firmware written by Eric Oberla. In addition, commands and data are transferred between the DAQ and a computer through a set of C++ codes also written by Eric Oberla. Some time was spent this summer studying these codes, and a few modifications were made for the sake of compatibility with the ACC. This brought the software and firmware to the point where we could communicate with an ACDC through 4 out of 8 of the ACC channels. However, there are still a few aspects of the C++ code and ACC firmware that need to be edited before reading out 8 ACDC boards will work.

The ACC firmware consists of three blocks. One of these blocks handles USB communication between the computer and the FPGA. In particular, this block receives commands from the computer in the form of 32 bit words. The firmware uses 4 bits out of this 32 bit word as a mask, which selects which ACDC boards receive the command. This must be changed to 8 bits in order to read out 8 ACDC boards. The second block handles LVDS communication between the ACC and ACDC, and also processes the data received by the ACDC. The third block handles triggering, as well as timing on

the board. In particular, this block processes triggers, sets the triggering mode, and generates meta data such as timing and event counts for each event.

The ACDC firmware consists of 6 distinct blocks. One of these blocks is dedicated to clocks, taking in two input clocks and generating several clocks of different frequencies using PLLs located on the FPGA. This block also sets the registers on the clock generator chip, and small changes were made to that code to get the settings we needed. It should also be noted that this block sends the clock coming from the ACC to all the other code blocks, meaning that the ACDC mostly runs on the same clock as the ACC. Another block controls the PSEC4 chips and reads data from them. Two of these blocks are dedicated to handling LVDS communication between the ACC and ACDC, as well as decoding instructions sent by the ACC. The other blocks either handle trivial things like controlling LEDs, or are unused.

The entire DAQ is controlled by a set of C++ codes written by Eric Oberla. These codes were intended to be used with the old central card, but are reasonably compatible with the ACC as well after making a few modifications. So far, we have adjusted some parameters to in the definitions file to use 8 rather than 4 front end boards. We also adjusted indexing in the read out code. However, many of the 32 bit words sent out by the C++ are meant for use with 4 front end boards. In order to be able to read out 8 ACDC boards using the ACC, we will have to edit these words.

3 Conclusions

1. The ACC can read out 30 channels of PSEC4 effectively.

2. 25MHz seems to be a threshold value for the write clock of the PSEC4 chip, and there may be a fix to get reliable performance at that frequency.

3. The old central card DAQ software is largely compatible with the ACC. However, some code must be edited to achieve full functionality

4 Acknowledgments

We would like the thank Eric Oberla and Mircea Bogdan for making this project possible, and for all the support and advice they gave. Henry Frisch for giving us this opportunity, and for all his advice along the way. Evan Angelico and Miles Lucas for helping us understand the C++ codes, and for their assistance with troubleshooting throughout the summer. Hayward Melton for lending his knowledge of electronics whenever needed. Mary Heintz for helping us with our various computer problems. Mark Zaskowski for working on many of our boards.

A Setting up an ACDC

1. Go to the electronics shop and get 3 pico-fuses from Mark's area. Two of the fuses should be at least 3A, the others should be at least 1A.
2. Place the $>$ 3A fuse into slot F3 and the other two fuses into slots F2 and F4, shortening the leads as necessary. Make sure to wear a static strap when you handle the board.
3. Find a power supply that can push 3A at 5V.
4. Connect the ACDC board to the power supply. Consult the picture for the proper orientation.
5. Turn on the supply and smell for smoke.
6. Check the current. If there is no amplifier attached, then the current should be a bit under $2A$. If there is one, then it should be a bit under $3A$.
7. If the current is too low, then check and make sure the fuses are installed correctly.
8. Put a jumper at JP1 on the board, as shown in the figure. This enables the local clock.
9. Remove resistor R40 from the board.
10. If slots J4 and J17 are unpopulated, then populate them with female SMA connectors.
11. The ACDC now has all the hardware it needs.

B Putting Firmware on an ACDC

1. Log on to a machine with Quartus installed. All of the computers in the electronics shop have it. If you don't have an account on the e-shop computers, then ask Mary Heintz to set one up for you.
2. Acquire some firmware. If you want to use Eric Oberla's, go to `https://github.com/ejobe/ACDC-firmware`. Note that his firmware was designed with a 40MHz clock in mind. Go to `https://github.com/lappd-daq/ACDC-firmware-25MHz` for a code with 25MHz in mind.
3. Start up Quartus, and go to file $\rightarrow$ open project to open up the firmware.
4. Either press the play button or do ctrl-L to compile the firmware. This should take 3 to 5 minutes.
5. Now that the firmware is compiled, you should decide whether you want to install a .sof or .jic. The .sof file is temporary, meaning that it will be erased when you power down the board. The .jic is permanent, so it will be there until you decide to load on different firmware.

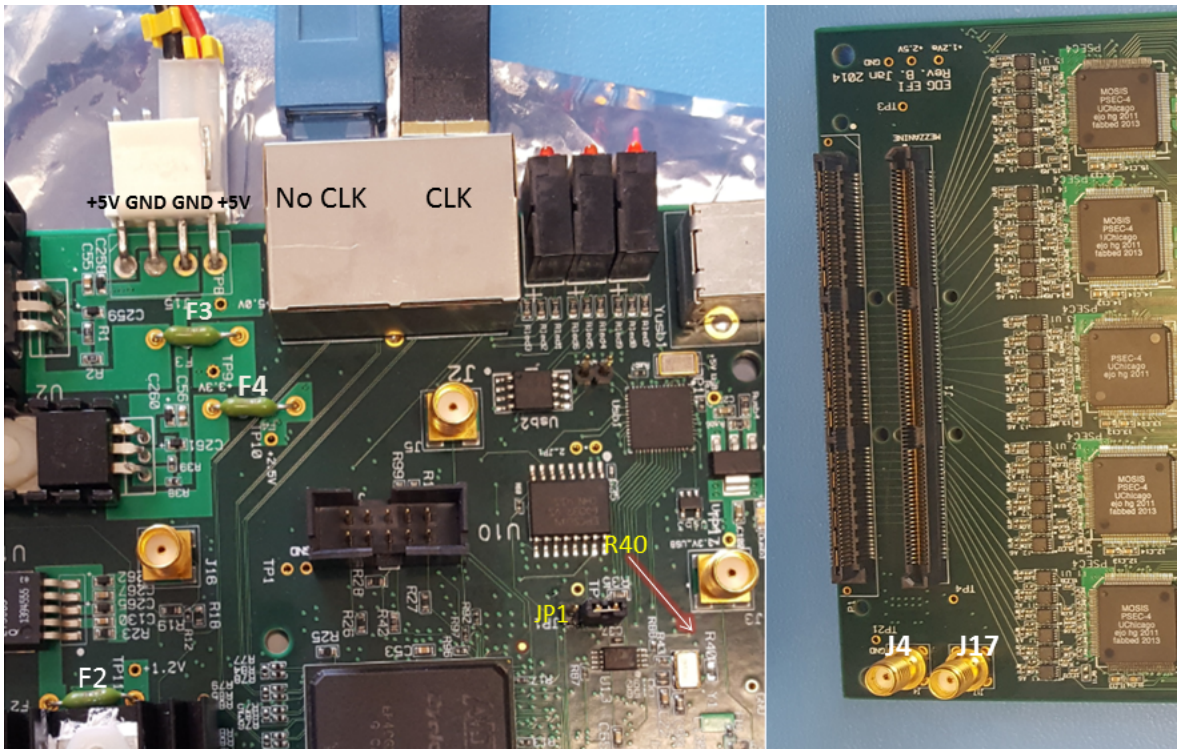

Figure 7: Locations on the ACDC board of all the part mentioned in this recipe. The black cable carries the clock, and the blue does not.

B.1 Installing .sof

1. Find a USB blaster (google "altera USB blaster" for a picture) and connect the USB end to the computer you are using.

2. Put on a static strap and connect the JTAG end of the USB blaster to the 10-pin connector located between the FPGA and the RJ45 ports. If you feel any resistance, then you're putting the connector in backwards.

3. Power on the ACDC board using the "Setting up an ACDC" instructions. The ACDC has to be on in order for the firmware to be installed.

4. In Quartus, go to tools → programmer to open up a programming menu.

5. Click on hardware setup and add the USB blaster.

6. Make sure that the mode on the programmer is set to JTAG.

7. If you don't see your .sof in the files list, go to add file and select your .sof.

8. Make sure that the program/configure box is checked.

9. Hit the start button. Installation should take about 30 seconds.

10. If your installation fails right away, then make sure your JTAG is connected the right way.

11. If you used Eric Oberla's firmware, then you should see a red and green LED blinking in unison.

B.2 Installing .jic

1. In Quartus (I use version 16) go to File → Convert Programming Files to open up the file conversion menu.

2. Under "Programming file type" choose "JTAG indirect configuration file"

3. Under "mode" choose "active serial"

4. Give your file whatever name you feel like.

5. Under "configuration device", choose EPCS64.

6. click on "flash loader" in the file/data area then hit "add device".

7. Under "device family" choose Cyclone IV GX. under "device name", choose EP4CGX110, then hit OK.

8. Click on "SOF Data" in the file/data area, then hit "add file". Choose the .sof file for the firmware you would like to install permanently.

9. Now, click the "generate" button to create your file.

10. Find a USB blaster (google "altera USB blaster" for a picture) and connect the USB end to the computer you are using.

11. Put on a static strap and connect the JTAG end of the USB blaster to the 10-pin connector located between the FPGA and RJ45 ports. If you feel resistance, then you're putting the connector in backwards.

12. Power on the ACDC board using the "Setting up an ACDC" instructions. The ACDC has to be on in order for the firmware to be installed.

13. Go to tools → programmer to open up to the programmer.

14. When the programmer opens, click any files you see and hit delete.

15. Click "add device" and choose you .jic file.

16. Click on the program/configure box.

17. Hit start, installation should a few minutes.

18. If your installation fails automatically, check to see if the JTAG is plugged in backwards.

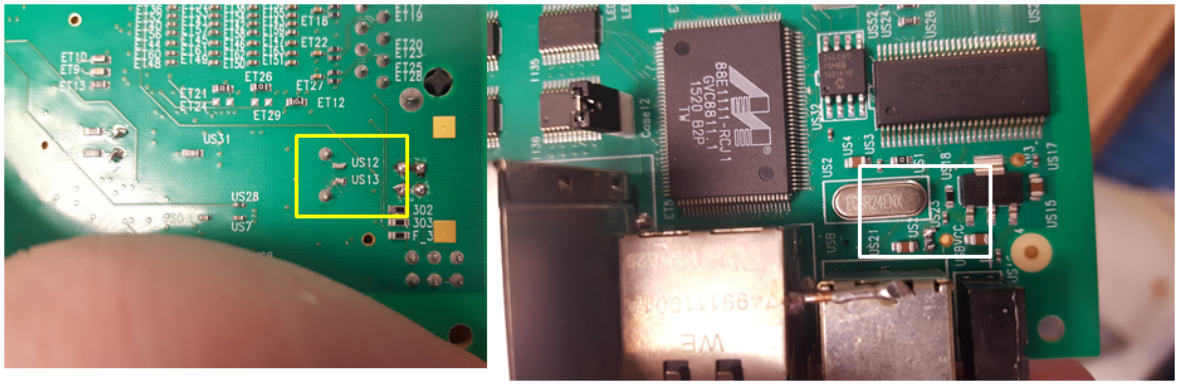

Figure 8: US22 and US18 are inside the yellow box. US12 and US13 are inside the red box, and directly underneath US22 and US18.

19. After installation is complete, cycle power on the ACDC.

20. If you used Eric Oberla's firmware, then you should see a green and red LED blinking in unison.

C Setting up an ACC

1. Check to see if there are resistors at slots US22 and US18 on the ACC board. They are located by the USB port, and are shown in the white box in figure 8. On the board that I have working, US22 is 180Ω and US18 is 115Ω.

2. If these resistors are not on the board, go to Mark's area to get surface mount resistors with appropriate values, then install them. Make sure whoever handles the board wears a static strap.

3. Directly underneath US22 and US18 on the opposite side of the board should be capacitors at US12 and US13.

4. These capacitors should be at 10pF, although they may come as 100nF. If you find they aren't the appropriate value, go to Mark's area to get new surface mount capacitors and install them.

5. Depending on what you use this ACC board for (for instance, reading out ACDC boards), you may also want the RJ45 shield to be connected to ground. To connect this shield to ground, add jumpers to the pins as shown in figure 9.

6. Finally, connect a jumper into slot J1 located by the USB port of the ACC.

7. Your ACC should now have all the hardware it needs.

8. Find a power supply that can push 3A at 5V.

9. Connect the power supply to the ACC as shown in figure 10, and turn on the power.

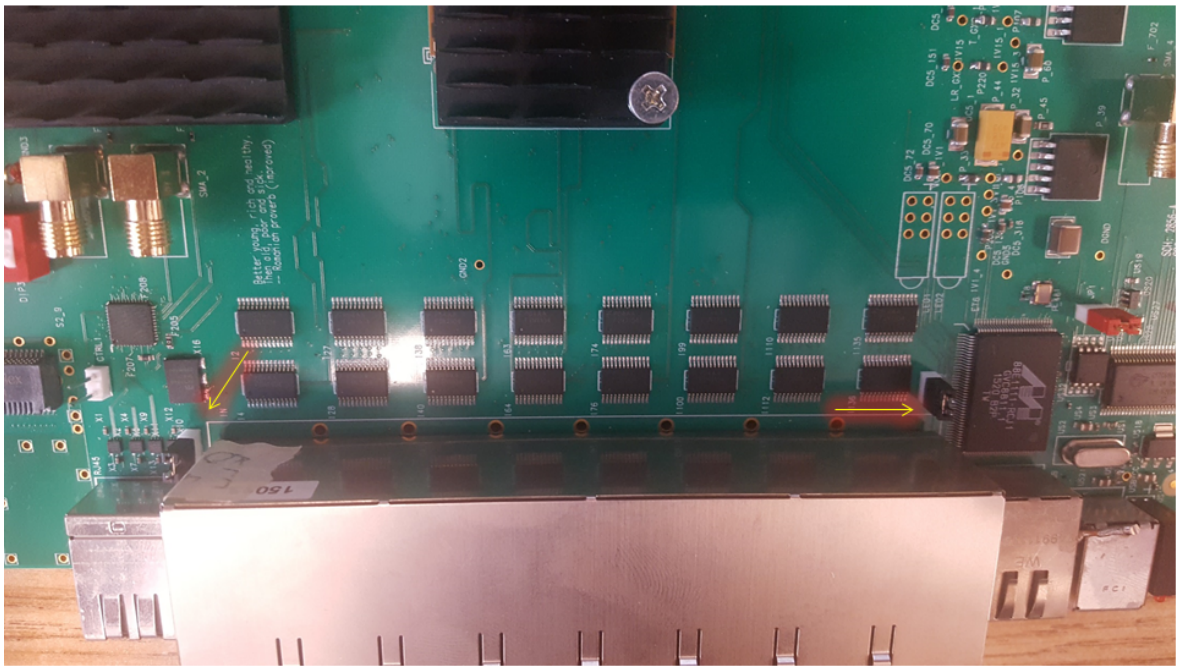

Figure 9: Attach jumpers to these pins on the sides of the RJ45 shield to get a connection with ground.

10. Sniff a few times to make sure there is no smoke coming out.

11. At this point, the ACC should have all the hardware it needs to work.

D Installing Firmware on an ACC

Installing firmware onto the ACC is more or less the same process as installing firmware on the ACDC, so you should read through the ACDC firmware recipe while noting the differences mentioned here.

1. Download the ACC firmware from `https://github.com/lappd-daq/ACC-2xSPF`. Note that This firmware was designed with a 25MHz clock in mind.

2. Connect the JTAG to the pins in the square in figure 10.

3. When you generate the .jic file, there are a few parameters that will be different. These differences can be seen in figure 11. You can find the flash loader under "Arria V".

4. If you installed Eric Oberla's firmware, you should see the red LED blink at about $1Hz$. You should also seeing that the green LED is on, as long as nothing is connected to the USB port.

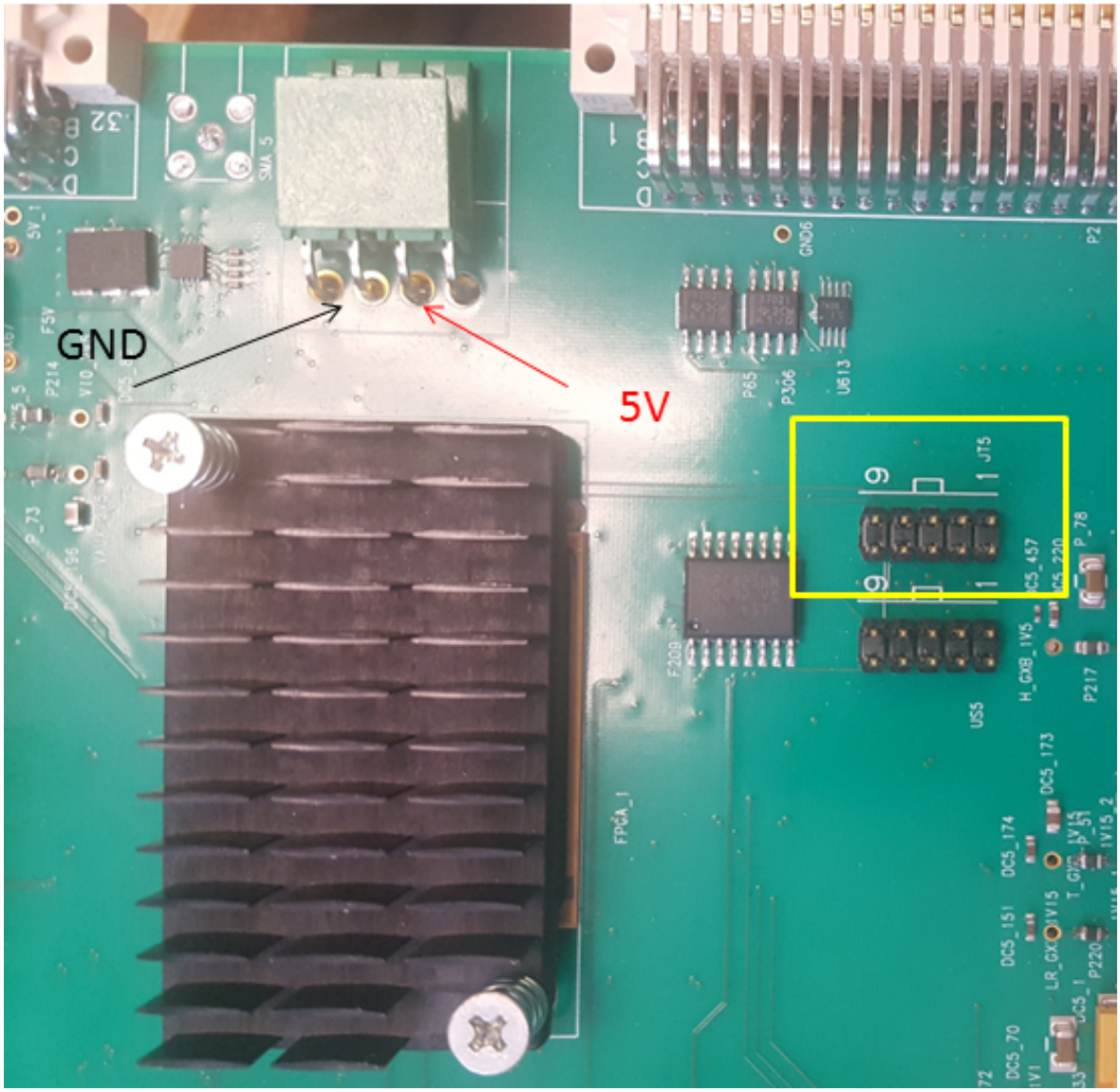

Figure 10: Location of the JTAG port used for flashing firmware onto the ACC, as well as the orientation of the power supply pins.

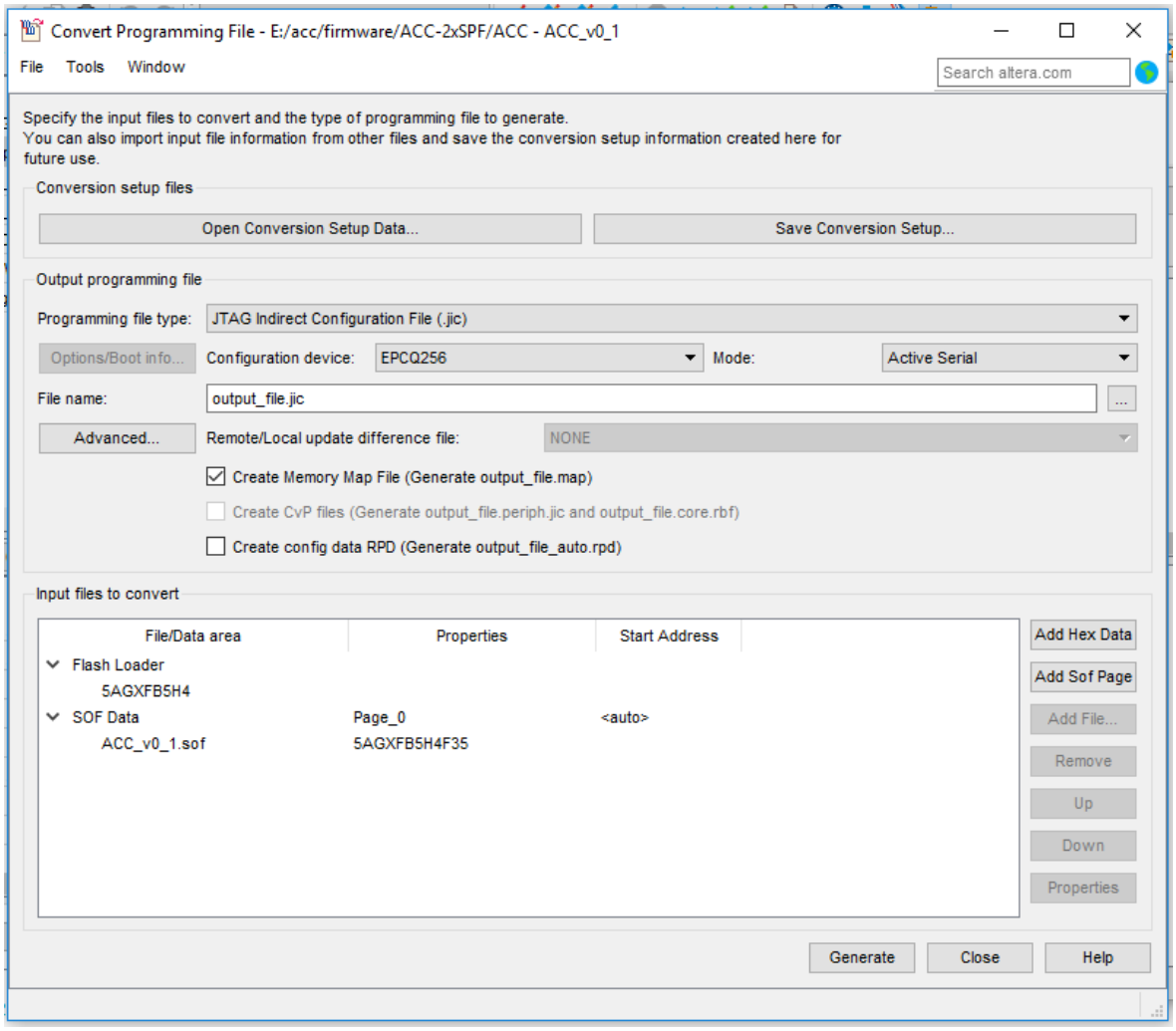

Figure 11: A screenshot of the Convert programming file menu with the proper parameters. Courtesy of J. Eisch.

D.1 Loading .pof onto the ACC

Alternatively, you can also flash your firmware onto the ACC permanently in the form of a .pof file. Eric Oberla's firmware comes with the .pof file, so there is no file conversion necessary for this approach.

1. Go to *tools* → *programmer* to open up the programming menu.
2. Change the mode to active serial.
3. Under "add file", choose ACC_v0_1.pof.
4. Connect the JTAG to the set of ten pens further away from the VME ports.
5. Make sure the program/configure box is checked, and hit start.
6. After the firmware is done flashing on, cycle power.

E Setting up USB communication with an ACC

First, make sure your ACC board has all the hardware it needs as mentioned in the "Setting up an ACC" recipe. Then, make sure it has firmware installed on its FPGA. After that, the first thing you need to do is make sure there is firmware on the USB chip of the ACC board. The next few steps will show you how to do this. Also, if you don't have access to PSEC1, then you can get the firmware and software mentioned here on Eric Oberla's website `http://hep.uchicago.edu/~eric/content/CypUSB/USBinstruct.html` .

E.1 Flashing on USB Chip Firmware

1. Take your ACC board up to the "Frisch Grad Student" office an plug it into the PSEC1 computer.
2. Log on to the PSEC1 computer as root. If you can't do this, then ask Mary Heintz or Evan Angelico for help.
3. Open up a terminal and enter the command *lsusb*
4. If you see a device with the ID 6672:2920, then your board already has the USB chip firmware and you can go on ahead to trying to talk to it. If you see a device with the ID 04b4:8613, then do the following steps.
5. Go to the directory with the libfx2loader software using the command

   ```
   cd /code/libfx2loader-20110913/
   ```

6. Load the firmware onto ram by entering the command

   ```
   ./linux.x86_64/dbg/fx2loader -v 04b4:8613 fw_async_ifclk.iic
   ```

7. Type lsusb. If it worked, then you should see the ID of the ACC is now 6672:2920.
8. Now, load the fx2tools firmware into ram by entering the command
   ```
   ./linux.x86_64/dbg/fx2loader -v 6672:2920 ./firmware/firmware.hex
   ```
9. Do a lsusb. You should see that the ID is now back to 04b4:8613.
10. Now, load the firmware onto eeprom by entering the command
    ```
    ./linux.x86_64/dbg/fx2loader -v 04b4:8613 fw_async_ifclk.iic eeprom
    ```
11. Unplug the USB cable, then plug it back in.
12. Type lsusb. If everything worked out, then you should see the ID of the ACC as 6672:2920.

E.2 Establishing Communication

Opening up USB communication with the ACC requires you use a machine with Linux on it. If you don't have one, then use the PSEC1 or PSEC2 machines.

1. Open up a terminal
2. First, check to see if you have the libusb-dev package by entering the command
   ```
   libusb-config --version
   ```
3. I have version 0.1.12 on my computer, and I believe the PSEC1 and PSEC2 machines have the same.
4. If you don't have libusb-dev, you can install it by entering the command
   ```
   sudo apt-get install libusb-dev
   ```
5. You'll also need the g++ compiler. Get this by entering the command
   ```
   sudo apt-get install g++
   ```
6. Put the followings line in (or create) the file /etc/udev/rules.d/80-uclab.rules
   ```
   SUBSYSTEM=="usb", ATTRS{idVendor}=="6672", MODE="666"
   SUBSYSTEM=="usb_device", ATTRS{idVendor}=="6672", MODE="666"
   ```
7. Update the rules by entering the command
   ```
   sudo udevadm control --reload-rules
   ```
8. Get Eric Oberla's C++ code from `https://github.com/lappd-daq/acdc-daq`.
9. go into the folder and type *make* to compile the code.

10. Hook up your ACC board to power as shown in the "Setting Up an ACC" recipe.

11. Connect your ACC to the computer, and wait for about 20 seconds.

12. After waiting for a while, you should see the green LED on the ACC turn off.

13. While in the folder with the ACDC-DAQ code, enter the command

    ```
    ./bin/setupLVDS
    ```

    If everything is working correctly, you should see the green LED on the ACC toggle on and off quickly.

F Setting up LVDS communication between an ACC and ACDC.

1. Get yourself an ACC board that you know has been setup and has firmware installed. If you have not gone through the USB communication recipe yet, do it before proceeding with this one.

2. If your ACC doesn't have jumpers like in Figure 9, put some on.

3. Get yourself an ACDC board that you know has been setup and has firmware installed.

4. Find two power supplies capable of running these boards. (3A at 5V)

5. Lay out the ACC and ACDC on an anti-static mat.

6. Connect an ethernet cable from any of the CLK ports on the ACC to the RJ45 port on the ACDC closest to the LEDs. Look at figure 12 for a guide of the ports on the ACC.

7. Connect an ethernet cable from the corresponding no clock port to the other port on the ACDC.

8. Connect the grounds of the two supplies together.

9. Power up the ACC and ACDC.

10. Connect a USB cable from the ACC to a computer that you know can talk to it.

11. go to the directory that contains the ACDC-DAQ code and type *./bin/setupLVDS* into the command line. If the LVDS connection is good, then you should see the green LED closest to the RJ45 ports on the ACDC light up.

12. Try entering *./bin/ledEn off* to turn off the LEDs on your board.

13. Type *./bin/ledEn on* to turn them back on.

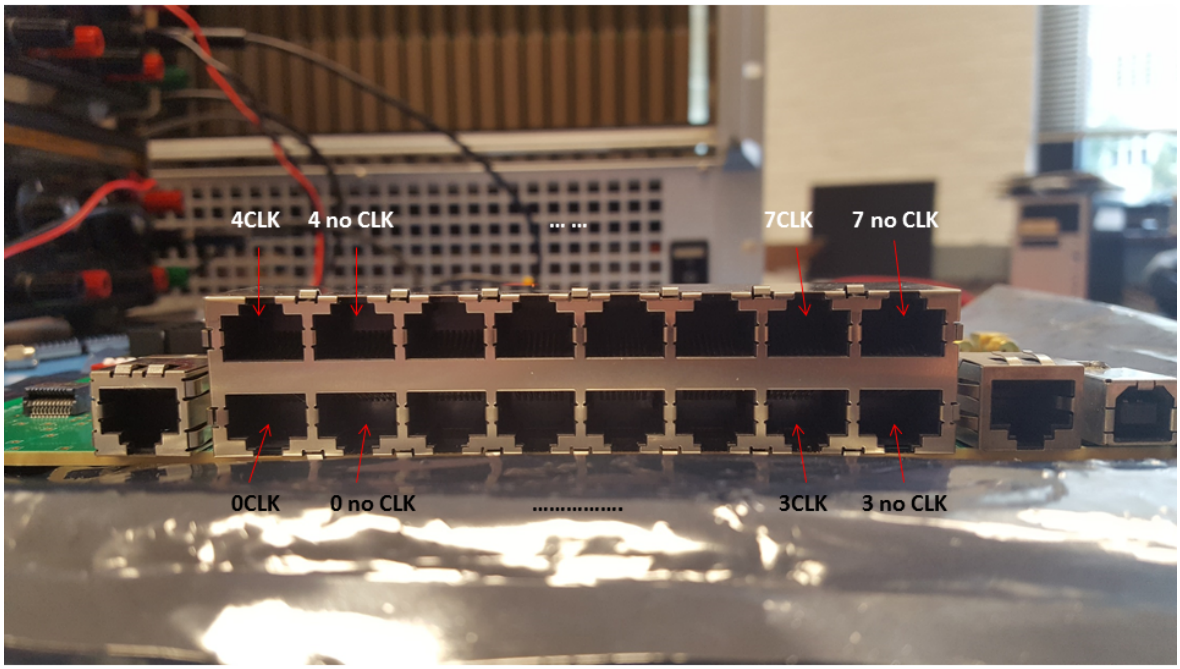

Figure 12: A guide to the 8 channels on the ACC.

14. Type *./bin/readACDC* to get information about your ACDC.

15. If all of these things work, then you have good LVDS communication between the two.

F.1 Troubleshooting

Since getting LVDS communication to work between these boards can be difficult, I added a section with troubleshooting advice. Don't bother reading this if your boards are already talking.

1. Probe resistors R44, R45, R46, R47, and R48 on the ACDC board with a scope. You should see a clock signal on each. If you don't, something is going wrong with the clock generator chip.

2. Use a volt meter to measure the voltage between the grounds on the two boards. This voltage ought to be small, under 20mV. If it isn't then you need to connect the grounds.

3. Use a volt meter to measure the potential difference between resistor R29 of the ACDC board and ground. This should be at least 0.9V in order for the clock generator chip to work. If it is smaller, then try using a CAT7 cable to connect the clock ports. This may raise the value.

4. Double check to make sure you know the frequency of the clock coming from the ACC board.

5. Look at the firmware of the ACC and ACDC to make sure they are setup to work on that clock.

G Reading Function Generator Pulses Using an ACC and ACDC

1. Set up LVDS communication between an ACDC and ACC board.

2. Install gnu plot by typing *sudo apt-get install gnuplot* into the command line.

3. Type *bash config.sh* into the command line to configure the ACDC board.

4. Type *./bin/calEn on 0* into the command line to make the PSEC4 chips receive signals from an SMA connector rather than the ribbon cable.

5. Use a function generator to create a signal of your choosing. The figures in this paper were done using a 120MHz sine wave with 400mV peak to peak amplitude.

6. Connect the function generator output to the SMA connector labeled J4 on the ACDC board.

7. Type *./bin/oScope acdc frames trigger* into the command line, where acdc is the address of the ACDC board, frames is the number of traces you want to plot, and trigger sets the triggering mode. Set trigger = 1 for self triggering and trigger = 0 to send software triggers.

8. If the channels on your plot look like they're all over the place, manually set the pedestal by enter the commands

   ```
   ./bin/setPed num
   ./bin/takePed
   ```

   Where num is an integer between 0 and 4096, with $ped = 1.2V * \frac{num}{4096}$.

References

[1] Frisch, Henry (2016). The LAPPD Pico-second Photo-Detector PowerPoint presentation at MIT Lincoln Labs. http://www.bostonphotonics.org/files/seminars/OSW2016_HFrisch.pdf

[2] Oberla et al. (2013). A 15 GSa/s, 1.5 GHz bandwidth waveform digitizing ASIC http://lappddocs.uchicago.edu/documents/218

[3] Bogdan et al. (2016). A Modular Data Acquisition System using the 10 GSa/s PSEC4 Waveform Recording Chip http://lappddocs.uchicago.edu/documents/288

[4] Bogdan et al. (2014). A Data Acquisition System using the 10 GSa/s PSEC4 Waveform Digitizing ASIC http://lappddocs.uchicago.edu/documents/265/sendit

[5] http://edg.uchicago.edu/~bogdan/AnniesCentralCard/schematics.html

[6] https://hep.uchicago.edu/~eric/?/pcb/acdcrevB/

[7] http://ieeexplore.ieee.org/document/12695/

[8] G. Haller, B. Wooley, A 700 MHz Switched Capacitor Analog Waveform Sampling Circuit, SLAC-PUB-6414, 1993. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.433.9386&rep=rep1&type=pdf