ROACH Engineering Manual

Timothy Madden

Background

Roach 1 and Roach 2

There are two versions of the ROACH board, the 1 and 2. The difference is that whereas ROACH1 is based on a Xilinx V5, ROACH2 is based on Xilinx V6. Also, ROACH2 allows 10GB Ethernet via SPF+ module and optical fiber. At APS, ROACH1 was programmed for the MKID project and replaced with ROACH2. ROACH1 is now used as test bench, and firmware is not up to date for TES work.

The website for the ROACH2 is here

https://casper.berkeley.edu/wiki/ROACH2

The firmware was originally based on the Ben mazin firmware for ROACH1 MKID readout. The Mazin firmware can be found here.

https://github.com/mstrader/MkidDigitalReadout

Mazin's casper library can be found here:

https://github.com/mstrader/mlib_devel

Where source resides

Tarballs of all ROACH Firmware, CASPER libraries, and Software are stored on BOX. These tars were taken from Madden's development directories for both ROACH1 and ROACH2. The easiest way to install the software, firmware, and libs needed for building is to untar both ROACH1 and 2. ROACH1 contains the ROACH1 designs, not needed for ROACH2, but also includes some necessary python modules needed for Roach2.

In BOX look for

All Files/ANL-TES-Internal-Measuremnts/ROACH_TARBALL

The tarballs are the two .tar.gz files.

The easiest way to get latest Argonne FW and build libraries is to untar the tar balls on BOX. Then you will not have to download from git or the casper website.

Firmware

Source firmware resides in

/home/beams0/TMADDEN/ROACH2/RoachFirmPy/Roach2DevelopmentTree

The main firmware file is tesd.slx, which is used for reading out RF multiplexed TES's

A setup firmware file is if_board_setup.slx, which setups up the IF board clocking so the mainfirmware tesd.slx can run correctly at the right clock rate.

The github repo for the APS firmware is

https://github.com/argonnexraydetector/RoachFirmPy

All software and firmware, custom library elements are in the repo.

The easiest way to get latest Argonne FW and build libraries is to untar the tar balls on BOX. Then you will not have to download from git or the casper website.

Python codes

/home/beams0/TMADDEN/ROACH2/RoachFirmPy/Roach2DevelopmentTree/pyfiles

The highest level py file is natAnalGui.py

C++ code

Anritsu Remote control

Code for communication with the Anritsu system is here:

https://github.com/argonnexraydetector/RoachFirmPy/tree/master/vx11/vxi11_1.10

It is installed here:

/home/beams0/TMADDEN/ROACH2/RoachFirmPy/vx11/vxi11_1.10

The executable is: anritsuOsc

The exe is a text-based program that interacts with STDOUT, meaning that starting up the program, you control the Anritsu with text input. Python starts this program as a subprocess and pipes its i/o to this subprocess. In this way, python can control the Anritsu.

QT Data Receiver

A QT program has been written for capturing ROACH data via 10 GB Ethernet. The program written in C++, captures UDP packets from the ROACH, and parses the data to save HDF5 files to disk in real time. The software runs on many threads, and is designed to be expanded to perform real time computation on the ROACH data.

The source code resides:

/home/beams0/TMADDEN/ROACH2/RoachFirmPy/Roach2DevelopmentTree/QT/readRoachStream

It builds to here:

/home/beams0/TMADDEN/ROACH2/RoachFirmPy/Roach2DevelopmentTree/QT/build-testEnet-Desktop-Debug/testEnet

testEnet is the exe file. Yes a terrible name…

Location of ROACH on cook.xray.aps

A local copy of all roach firmware and software is located here:

/localc/roach/RoachFirmPy

To run ROACH you must set an enviromnemtn variable ROACH to the location of the ROACH software/firmware install.

For csh:

setenv ROACH /localc/roach/RoachFirmPy

For bash

export ROACH=/localc/roach/RoachFirmPy

Location of Docs and Files for ROACH

The notebooks are all electronic files and are useful to read to lern how the ROACH system works. Also performance is documented in these books.

Older One note doc. Contains older ROACH data from earlier in the project.

\\Nickel\Detector_Share\aa_Detector Development\ROACH\eNotes\Personal\ROACH

Most recent lab notebooks:

On beams in linux see several LibreOffice word docs (odt)

/home/beams0/TMADDEN/ROACH2/notebooks

Location of Papers/Posters on Firmware

The best way to understand the firmware is to read the ASC 2016 poster and paper. They Are found on BOX at:

The poster is located on BOX at:

Box://AllFiles/Conferences&Talks/ASC_2016/ROACH2/ADC2016_RoachFW_V2.0POSTER.pdf

The Paper in LATEX is in

Box://AllFiles/Conferences&Talks/ASC_2016/ROACH2/ASC2016_ROACG/ASC2016_RoachFW_final.pdf

PDFs of the actial FW drawings are in

Box://AllFiles/Conferences&Talks/ASC_2016/ROACH2/tesd/

A poster was created to document the software for ROACH. It is located at

Nickel:Detector_Share\Tim Madden\Papers\ASC2016_roachposter\ ASC2016_RoachSW_V0.1.pptx

ROACH System Overview

The roach system is a multichannel software defined radio system that both transmits and receives up to 128 simultaneous channels. Each channel corresponds to a SQUID, Resonator, and TES x-ray sensor. On one RF coaxial cable the ROACH can read up to 128 simulteneous channels. There are two parts: The ROACH hardware based on FPGA and Firmware, ADCs/DACs and RF circuits. The second part is the Linux box running python and C++ codes.

The ROACH generates DAC signals to stimulate the resonators over a coax cable by storing complex sinusoids (see Euler's formula) in QDR RAM. There is a RAM for cosines and one for sines, and a DAC for each RAM, allowing generation of cosine and sine, for a complex sinusoid. These two DACs are fed into an IQ mixer to generate single sideband radio in that range of 5GHz. The band of the DACs at baseband are 10MHz to 250MHz. Read about IQ modulation and single sideband radio on Wikipedia.

For rhe receiving end or ROACH, the RF signals are mixed back to baseband with IQ mixer and fed into two DACs, to capture a complex sinusoid. This is IQ demodulation. The captured signal is windowed with a clever way of applying Hamming window (see "Polyphase method" on the Casper ROACH website). The windowed signal is then Fourier transformed with two conconcurrent 512 FFT blocks. We use two FFTs so we can FFT overlapping chunks of time domain data, for better noise performance.

See https://casper.berkeley.edu/wiki/The_Polyphase_Filter_Bank_Technique for theory on the Polyphase windowing method.

For FFT's See https://en.wikipedia.org/wiki/Short-time_Fourier_transform. For better paper see

Welch, Peter. "The use of fast Fourier transform for the estimation of power spectra: a method based on time averaging over short, modified periodograms." *IEEE Transactions on audio and electroacoustics* 15, no. 2 (1967): 70-73.

Allen, Jont B., and Lawrence R. Rabiner. "A unified approach to short-time Fourier analysis and synthesis." *Proceedings of the IEEE* 65.11 (1977): 1558-1564.

The FFt coefficnents undergo a second demodulation, then are sent to a Flux Ramp demodulation algorighm. Finally the data is sent via 10GB to the Linux box.

The Linux box has two connections to the ROACH. The 10GB fiber is for receiving data from ROACH. A 1GB copper connection is for control of ROACH. Python codes run in ipython to control and program the ROACH, control saving of data, and for plotting data. Python provides a QT based GUI for a control user interface. We use anaconda python on APSshare.

The linux box also runs a C++ QT program for receiving data, called by then names "Data Receiver," or "testEnet" or simply "the QT C++ program." This program uses a 10GB interface card (fiber and SPF+) to capture the data stream from ROACH, and save to HDF5 files. Python remote controls the program by open a pipe to its stdin/stdout and sending and receiving text commands. The beauty of QT is that any slot can be called by a text string, becauyse QT implements reflection. The data receiver saves its hdf5 data to disk, and python reads the data to plot.

The linux box also has a network connection to an Anritsu frequency generator, using a C program called anritsuOsc. This is for generating the 5GHz signal to run the mixers in the ROACH. This signal is called "Local Oscillator" or LO. Finally, the linux box uses RS232 connections to control a voltage generator for biasing TES's and for controlling a ramp generator, for modulating the SQUIDs.

All control of the system is ith the Python QT Gui. It is also possible to control the system with python scripts in the ipython shell.

An experimental EPICS interface exists using a soft IOC with soft record support, and pyEpics. An experimental EPUICS V4 interface exists using the EPICS 4 python interface. These EPICS interfaces are only experimental and not used.

The protocol used between ROACH and linux is katcp. The basic idea is that on the ROACH Simulink design you insert a "software register" with a specific name. Then with katcp, from python you can perform a writeInt("softwareregname",100) to write to that register. The Casper library includes this abiulity. The ROACH box runs a katcp server that allows this. It maps the text names to registers on an OPB bus in the FPGA. OPB is a Xilinx bus for connecting hardware to software. AIX is a newer standard that replaces OPB.

We implemented a homegrown katcp in katcpNc.py. The git for the official kapcp is here:

https://github.com/ska-sa/katcp_devel

Setting up QT Development Tools

Here is my recipe for installing QT and QTCreator.

Using:
qt-everywhere-opensource-src-4.8.4.tar
qt-creator-2.7.1-src.tar

0. Untar the above tars.

1. Build QT

cd qt-everywhere-opensource-src-4.8.4

```
./configure --prefix=/local/qt4.8/ \
 -release \
 -shared \
 -no-webkit -no-javascript-jit \
```

-confirm-license

gmake -j8
gmake
gmake install

2. setenv PATH /local/qt4.8/bin:$PATH
   setenv LD_LIBRARY_PATH /local/qt4.8/lib

3. build qt creator
   cd qt-creator-2.7.1-src
   qmake -r
   make -j8
   make

A simpler way to get QT Creator is to download the installer.

/home/beams0/TMADDEN/swWork/QT/qt-creator-linux-x86_64-opensource-2.7.0.bin

After installing QT, run the QTCreator .bin file as an executable. It can install locally without root.

Python runtime environment

Env Variables

Set these environment variables to use ROACH. Put in .tcshrc.

setenv ROACH /home/beams/TMADDEN/ROACH2/RoachFirmPy

setenv PYTHONPATH /home/beams0/TMADDEN/ROACH/lib/python2.7/site-packages:/home/beams/TMADDEN/ROACH/pyqtgraph

Anaconda Python

We use Anaconda Python. Enthought works too with some modifications to the version of QT. Enthought uses pySide, while Anaconda uses pyQT.

In linux make a script called runipython with this line:

```
/APSshare/anaconda/x86_64/bin/ipython –pylab
```

Type ./runipython to run a proper version of python.

Python packages

Special python packages are needed to run ROACH. These are installed in /home/beams0/TMADDEN/ROACH/lib/python2.7/site-packages.

Doing a ls of this dir reveals all the required packages not part of Anaconda python.

bitstring-3.1.2-py2.7.egg-info
bitstring.py
bitstring.pyc
corr
corr-0.5.0-py2.7.egg-info
easy-install.pth
iniparse
iniparse-0.4-py2.7.egg-info
katcp-0.5.1-py2.7.egg
lib
mock-1.0.1-py2.7.egg
site.py
site.pyc
unittest2-0.5.1-py2.7.egg

Setting up MATLAB and Xilinx

Installing Xilinx Tools

Use a Red Hat Linux 6. Linux 7 is incompatible with the Xilinx ISE tools needed for compiling the Virtex 6 FPGA on the ROACH. The ISE tool can be downloaded from the Xilinx website. Because of firewall issues, it is tricky, and may take 4 hours. It is easier to download at home. It takes 4 hours, but it ways works. You download at home, put on a thumb drive, or send to BOX.

The installer is here on Macoupin:

/local/XilinxSetup/ Xilinx_ISE_DS_Lin_14.6_P.68d_3.tar

You untar, then run xsetup. Hit ignore if it tries to goto the Xilinx site. It will still work. We use ISE 14.6 currently.

They should be installed on Macoupin at /local/Xilinx

Installing MATLAB

We use matlab R2012b. You can run it from /usr/local/matlab.R2012b/bin/matlab and hope you get an APS matlab license. Usually if you start it at 8AM you are OK. Preferably you can run from a local install in Macoupin.

To install you download from the matlab website using the detector group login, based on Nino's email and password. You want version 2012b. You download an installer and put it in /local on whatever linux box you are using. The installers are downloaded on Macoupin at /local/matlab_setup. The installers also reside on cook at /localc/matlabsetup/matlab_downloads. You unzip the installer, then run install.

To make the license work, you must rename eno1 to eth0. This should be done by Troy, as it involves root, and messing with the GRUB, or linux bootloader.

Go to: https://www.mathworks.com/login?uri=%2Flicensecenter%2Flicenses
amiceli@aps.anl.gov
utdFg1y0

cook mac 64:51:06:57:bd:5f

To reinstall, as when moving to different computer:

On Cookl:

Cd /local

Installing CASPER Library

The easiest way to get latest Argonne FW and Casper build libraries is to untar the tar balls on BOX. Then you will not have to download from git or the casper website.

The git for CAPSPER is:

https://github.com/casper-astro

Clone the repo: https://github.com/casper-astro

Curently the mlib_dev that works for our FW is on:

/home/beams0/TMADDEN/ROACH2/mlib_devel

It is best to cp the one at /home/beams0/TMADDEN/ROACH2/mlib_devel as it includes the APS designed yellow blocks, though supposedly they are now part of the overall CASPER lib.

Running MATLAB for ROACH

Create the following script in your mlib_devel dir, call it startsg

```
#!/bin/bash
####### User to edit these accordingly ######
export MATLAB_PATH=/local/MATLAB/R2012b
PLATFORM=lin64
export XILINX_PATH=/local/Xilinx/14.6/ISE_DS
export MLIB_DEVEL_PATH=/home/beams0/TMADDEN/ROACH2/mlib_devel
```

export XILINXD_LICENSE_FILE=2100@platinum.aps.anl.gov
#export LM_LICENSE_FILE=2100@platinum.aps.anl.gov

##
source $XILINX_PATH/settings64.sh
export PATH=${PATH}:${XILINX}/ISE/bin/${PLATFORM}:${XILINX_PATH}/ISE/sysgen/bin/${PLATFORM}
export XPS_BASE_PATH=$MLIB_DEVEL_PATH/xps_base
export MATLAB=$MATLAB_PATH

alias sh=bash
alias gmake=make

$MATLAB/bin/matlab

Building the Firmware

All Firmware designs are .slx files in $ROACH/RoachFirmPy/Roach2DevelopmentTree/

1) In /home/beams0/TMADDEN/ROACH2/mlib_devel run startsgCook, on cook.xray.
2) Open tesd.slx, and make it the current window on top.
3) Run $ROACH/RoachFirmPy/Roach2DevelopmentTree/mfiles/makeevents.m. this will define several variables needed for setting up RAMS etc. You will get several plots and an error at the end. Ignore error.
4) In Matlab window, type casper_xps
5) Hit Compile on the casper_xps window.
6) Casper_xps matlab scripts will first construict several ROACH blocks, then run Xilinx system generator to make a Xilinx IP block representing your design. Then a base FPGA project is created automatically with ROACH base design, and your Simulink design included. Finally te Xlinx compiler is run. On cook, compile takes 1 hour. In Macoupin compile takes 5 hours.
7) Once the compile is done, you can check matlab window to make sure it was successful. It usually is.
8) Get the bof file, or the final FPGA firmware that programs the FPGA. Run *$ROACH/RoachFirmPy/Roach2DevelopmentTree/cpbof* This will copy the latest tesd.bof file from its build directory into the *$ROACH/RoachFirmPy/Roach2DevelopmentTree/bestBofFiles* directory and add to the git repo.
9) Edit *$ROACH/RoachFirmPy/Roach2DevelopmentTree/pyfiles/fftAnalyzerR2.py* Change line 903 to reflect the name of your latest FW bof file.

self.mainFW = ROACH_DIR+'/Roach2DevelopmentTree/bestBitFiles/tesd_2017_Jun_21_1708.bof

10) Start the ROACH software and new Firmware will be loaded.

Network Interface Setup

Roach can run on RHEL6 or 7. The device names differ in el7. These instructions concern el6.

In rootsh, to give privilege to anyone to run ifconfig:

```
chmod 4755 /sbin/ifconfig
```

To set up connections so the roach can connect:

In rootsh, run

nm-connection-editor and set up these network conns.

eth0- 1st slot (to right) in the 10GB ethernet card-

Plug the fiber into eth0. Set to address 192.168.1.102

eth1,2,3, other holes in 10GB Enet card

eth4- 164 network

eth5 192 network, set to 192.168.0.203

To see if data is coming over

tcpdump –v –I eth0

The file dataCapture.py, calls ifconfig to set up eth0. So you must set device and ipaddress in this file. This file runs the C++ program that captures ROACH data. You must have privilege to run ifconfig to alter device settings.

The 10GB ipaddress f the roach itself is '192.168.1.11', set in fftAnalyzer2.py, def setupEthernet

The 10GB IPaddress of linux box is set in dataCaptyre.py, in globals:

ten_G_ipaddr = '192.168.1.102'

ten_G_device = 'eth0'

The linux 1GB Ethernet address is set w/ ifconfig and or nm-connection-editor

The roach 1GB Ethernet address is set in katcpNc.py, in init(), as default variable, set up

IP = 192.168.0.70',port=7147

The anritsu box is on the same net as the 1GB roach interface. Its IP is set in anritsu.py, to

self.ANRITSUIP='192.168.0.68'

Simulating the Firmware

Simulating parts of the firmware is most often done, because the whole firmware is so complex to simulate.

Simulating Multichannel FIFO64. This is for testing the channelizer fifos, and FSM for readout the channelizer fifos.

1) Open multififo.slx
2) Run simulation and look at scope.

1) To simulate the firmware it is best to alter tesd.slx to disconnect the ADC outputs from input to the Polyphase/FFT blocks. The purpose is that you need a signal to input into the firmware for a meaningful simulation. Add the simulation sin generator from maddenLibrary.slx and wire into the design.
2) Run Run $ROACH/RoachFirmPy/Roach2DevelopmentTree/mfiles/makeevents.m to generate proper signals and variables into the ROACH. This mfile generates input signals for the ADC, sets up the flux ramp demod and other memories and registers in the firmwarwe, and also generates the expected output of the ROACH flux ramp demod To compare to the ROACH output. Ignore the error message.
3) Hit simulation button on the tesd.slx window.

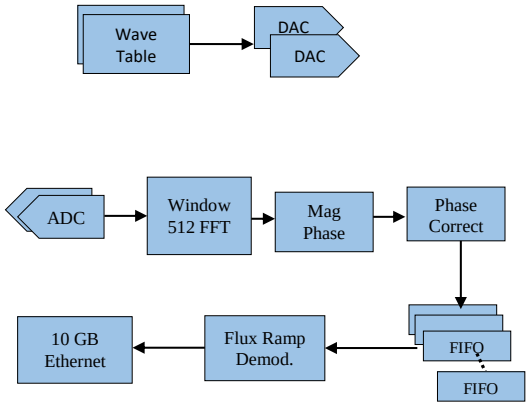
4) Run makeevents.m again. This will take output from ROACH and compare to expected output signals. No error messages this time.

Overview of Firmware

High Level Description of Firmware'

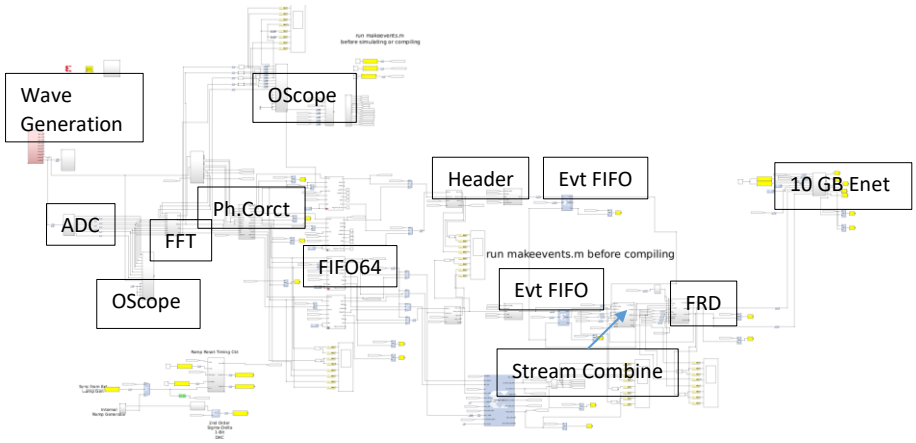
There have been a few changes since the poster and paper were published, so a brief explanation is given here, highlighting differences between current FW and paper description. First read the paper and poster for a description of the firmware.

The firmware, in the figure below, has two sections. The first section generates a summation of many tones, stored in RAM, by playing out the contents of the RAM to an I and Q DAC. The second part captures digitized I and Q signals with an ADC, windows and performs FFTs, channelizes by selecting FFT bins of interest and creating independent streams, and computes flux ramp demodulation. The firmware works with magnitude and phase rather than I and Q, which makes fine modulation simpler.



Top: I and Q generated from signals stored in RAM wave table. Bottom: I and Q captured, FFT performed, converted to magnitude/phase, fine modulation (Phase correct), channelization into separate streams, flux ramp modulation, transfer over 10GB Ethernet.

Here is the whole firmware.



Number of channels

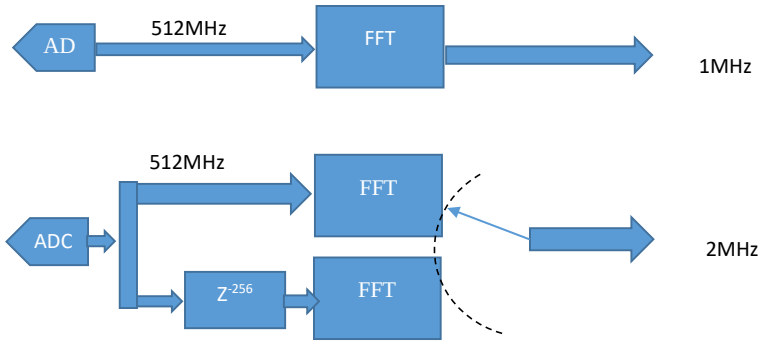
The current ROACH provides 128 channels. Only the lower sideband of LO is supported, that is, coefficients 256 to 512 of a 512 point FFT. The positive side of LO appear in FFT coef. 0 to 255, and are thrown away. Adding a 2nd channelizer is possible (and preferrend) to use the other sideband of LO.

Bacause of FIFO filling problems and time delays in digital state machines, the ROACH will start to drop samples once we use over 110 channels or so. Effective number of channels is around 110.

Polyphse filter and FFT design

When the paper was written the ROACH used one FFt engine to produce a sample rate of 1MHz after the FFT operation. The 1MHz sample rate comes from the fact that the ADCs sample at 512MHz, both I and Q. The I and Q signals are used for real and imaginary parts of a complex FFt. The FFT is 512 samples long, resulting of a data rate of 1MHz after FFT. As the clock rate of the FPGA is 128MHz, four samples are processed at once. The FFT is a block transform, meaning that as 512 samples come in, a single FFT is computed. The next 512 samples contribute to the next FFT. There is no overlap in time domain data for successive FFTs.

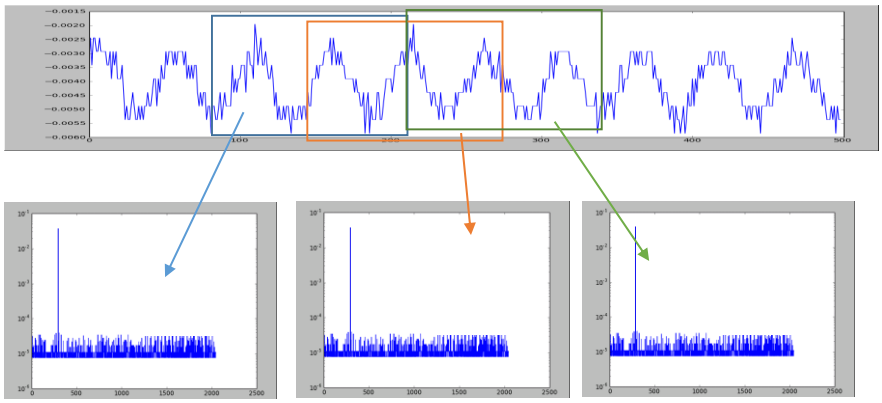
The ROACH currently has 2 FFt engines running in parallel. With two FFTs running concurrently, we allow 256 samples of overlap of time domain data for each successive FFT. This doubles the sample rate, after FFT, to 2MHz. To provide overlap between FFTs a delay line of 256 samples is included. The two 1MHz streams from each FFT are combined by a commutator circuit, or multiplexer, to create a 2MHz stream.



In the ROACH, two FFT engines run with one delayed by 256 samples. This provides overlap in the time domain of FFT computation, and results in a 2MHz sample rate after FFT. AT the time of the ASC paper, the upper design was used, and is in tes.slx. The lower design is current, in tesd.slx.

To show how running two FFTs allows overlap of time domain data see below. Also, running two FFTs with overlap in time domain is necessary to satisfy the Nyquist Criterion. That is, an FFT is actually a filter bank, where each bin is a band pass filter. The band pass filter has a certain bandwidth of nominally

1MHz. For a 1 MHz band width, we need a sample rate of 2MHz. For this reason, two FFTs is better than one. If one FFt is used, the effect is increased noise due to aliasing.



On top is time domain signal. Bottom are 3 successive 512 point FFTs with overlap. Two FFT engines are necessary to do overlapped FFTs.

Details of FFT FW

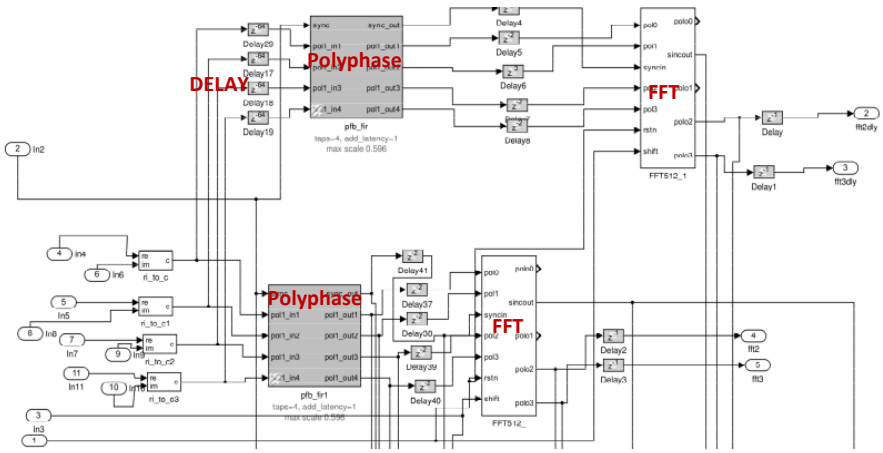
The FFT is designed from Xilinx FFT blocks and aps-designed butterflies. It takes in 4 streams with samples in this order. For samples Xk with k=0,1,2,3,4…

Tap0: k=0,4,8.., Tap1: 1,5,9…. , Tap2: 2,6,10…., Tap3: 3,7,11…

The CAPSER FFT green block, used in most Roach FW has coefficients Fk, with k=0,1,2,3,4…

Its output is is in this order:

Tap0: k=0,4,8.., Tap1: 1,5,9…. , Tap2: 2,6,10…., Tap3: 3,7,11…



Firmware for two Polyphase filters, FFTs, one delayed.

The ANL/Xilinx FFT his this output order:

Tap0: k=0,1,2,3..63, Tap1: 64,65,66,67….127 , Tap2: 128,129,130…383, Tap3: 384,385,396…511

The software interface is roachFFT.py.

Channelizer

The Channelizer is a state machine that picks which FFT coefficients are processed. That is, the FFT outputs its coefficients F_k from 0 to 511, with each F_k representing a different frequency. If we wish to read out one resonator we only need a single stream of F_k, where k corresponds to the resonator frequency of choice. The channelizer maps which F_k to an arbitrary numbering system, or channel number. For example F22 could be mapped to Channel 0, and F37 could be mapped to channel 1. These channels are stored into separate FIFOs to create distinct data streams for each channel. There are 128 channel FIFOs in the ROACH firmware, allowing 128 separate data streams. The 128 FIFOs are implemented as a single dual port memory with a set of 128 address counters. The Channelizer stores a table in BRAM to map FFT coeffieicnts to channels. The BRAM is programmed by the ROACH software to assure the correct set of resonators are read out.

The channel FIFOS are called FIFO64s. The idea is that each FIFO64 is effectively 64 independent FIFOs, one per channel. A read and write address is supplied to tell which FIFO or channel we write and read. The ROACH currently has 128 channels, or 2 FIFO64s. Note: Because the current FW uses 2 FFTs, there are 2 FIFO64s per FFT, or a total of 4 FIFO64s.

See Channelizer.py for software interface.

Note: This FIFO64 design is hard to compile. The 128 counters requires many levels of multiplexers. Though it is a cool design, it is not a good FPGA implementation. A newer FIFO has been designed and is stored in a branch of the git, called newfifo. It is not fully tested. It may be better to go back to the Mazin design using a commutator for the simplest. This is necessary future work.

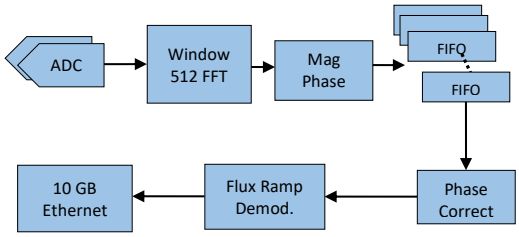
I/Q to Magnitude/Phase

The FFT engine produces complex coefficients at its output in the form if Real and Imaginary numbers. It is beneficial to convert to magnitude and phase using a Xilinx IP CORDIC IP block. Converting to phase/ magnitude allows 1) pulse finding in phase and 2) simpler fine modulation after FFT computation.

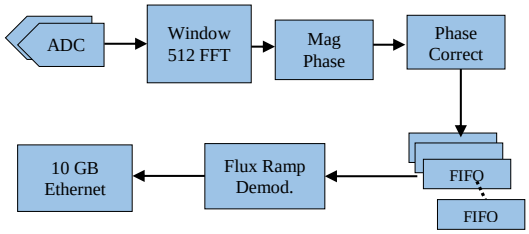
Phase corrector circuit

The diagram below shoes two possible ROACH FW designs. A) was the design published in the ADC2016 paper. B) is the design in the current roach.

a)

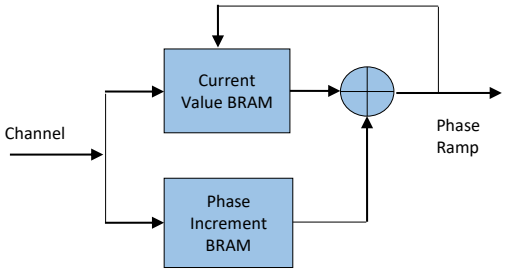


b)



The phase corrector circuit is described in the ASC2016 paper, in terms of its math. The idea is that if the frequency of a resonator does not fall exactly in FFT bin center, signal needs a second modulation to move it to bin center. This can be done with a complex multiply by a complex sinusoid. Another way to do this is to convert to magnitude and phase, and add a ramp to the phase angle. The ROACH firmware uses the second method. A difference in the paper documentation and the current roach firmware is that the order of operations was changed. In a) above the FIFO comes before phase correction. This design was published in ASC2016 and was found to be cumbersome. The current firmware design is b) above in which the phase is corrected before the FIFOs.

Doing the post-FFT modulation using a complex multiply of real and imaginary parts requires a long RAM to store complex sinusoids. Because of many channels, these sinusoid terms are added together in a the same RAM. Therefore many channels, the post-FFT modulation is not done with a single difference frequency, but a plurality of difference frequencies. For this reason, a low pass FIR filter is needed to remove extra images of the signal caused by the plurality of frequencies. The NIST and Mazin design use this method with long RAM of summed sinusoids, complex multiplier and FIR filter. The ANL design differs in this respect. Converting to phase/magnitude first means that the modulation is done with a simple ramp representing phase, or digital counter stored in a small BRAM, simplifying the design. No FIR filter is necessary.



The Phase corrector circuit is controlled by phaseCorrect.py. A phaseCorrect object needs a reference to the roachFFT and sramLut objects, because it must know which FFT bins are being read out, as well as the exact frequencies being generated in order to correct the phase properly.

Synchronization of ROACH computation to Ramp generator

The firmware performs flux ramp demodulation on the FFT coefficients. There are three important settings to allow synchronization of ROACH to ramp generator.

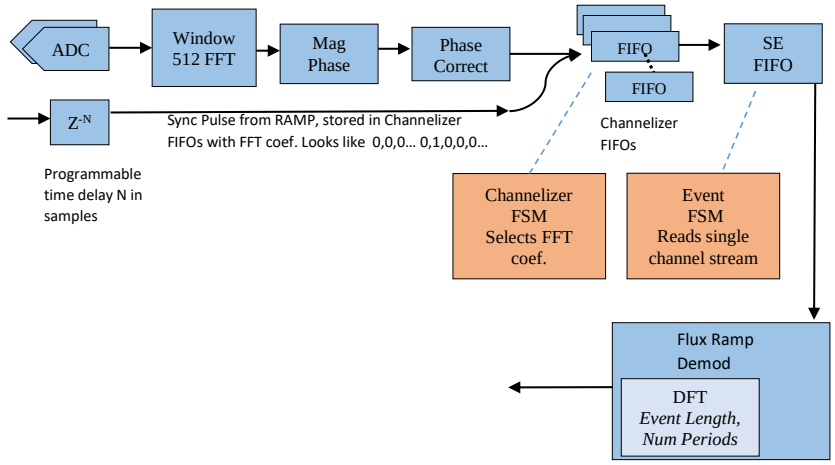
1) Ramp sync pulse delay
2) Integration Time
3) Number of periods

The sync pulse from the ramp is connected to a digital input to the ROACH. This pulse is delayed by some time period, using a digital shift register implemented in RAM. The purpose of the delay is to account for group delay of ramp signal through RF hardware, SQUID, resonator, digitizers and FFT computation. It is desired to line up pulse in the FPGA with corresponding FFT coefficients to the beginning of the useful part ramp signal as seen through the SQUID. The idea is that the delay can be set such that the pulse internal to FPGA is aligned just after the glitch caused by ramp reset. In this way, data that is part of the glitch can be thrown away.

In practice this internal pulse, or sync bit, is combined with the FFT coefficient data stored in the channelizer FIFO. *The Event State Machine*, defined in fifoFSMPh3.m, reads the channelizer FIFO throwing away data until it sees a sync bit set to 1. This corresponds to throwing away glitch data. On seeing a 1, the state machine then begins saving data to a second fifo, called the *Single Event FIFO*, or SE-FIFO. It will write N samples, corresponding to the integration time, to this SE-FIFO. The data in the SE-FIFO is then read out by the flux ramp demodulator to compute one sample of flux ramp demod phase. Integration time is typically set to around 100 to 200 samples.

The third number, Number of Periods, is used to program a DFT (Discrete Fourier Xform) in the Flux Ramp Demodulator circuit,FRD circuit. If for example one ramp causes 3 SQUID rotations, then we set then DFT to expect three periods of cosine.

We use the word *Event* to define a segment of FFT data from a single channel used to compute FRD. The Event could be 200 samples long for a 10kHz ramp frequency and 2MHz sample rate after FFT. The Event phase data will have a few rotations of the SQUID used for FRD computation.



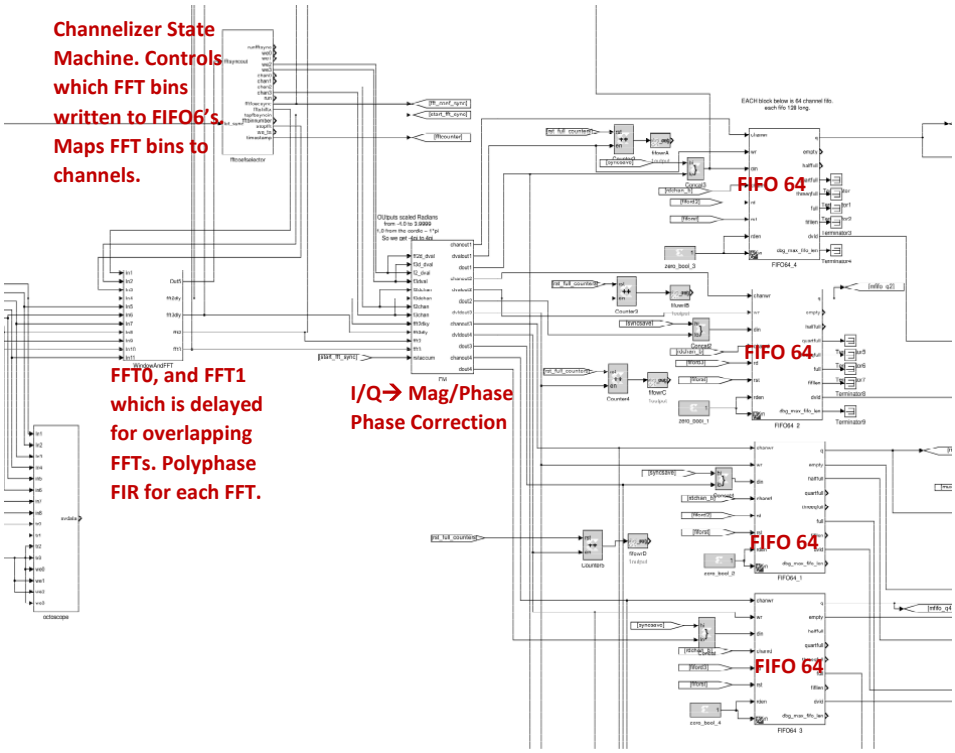
Detailed view of part of ROACH firmware showing state machines, sync pulse stored with FFT coef., single Event Fifo, and Flux ramp demod internal DFT.

Channel Numbering Scheme

To map FFT bin to channels there are several steps. There is a *leg* number of 0 or 1. FFT bins 0 -255 end up on leg 0, bins 256-512 map to leg 1. Leg 0 corresponds to RF frequencies above LO, or the right side band. Leg 1 corresponds to the left sideband, or RF frequencies lower than LO. In the ASC2016 paper and poster Leg 0 and Leg 1 are implemented. This FW is tes.slx. For the current 2MHz dual FFT firmware, leg 0 was sacrificed, and is not implemented. A consequence is that the current FW can only read out one sideband of LO, the lower sideband, and FFT bin s 256-511. A leg has one FRD circuit. Log 0 outputs data into the upper 32 bits of the 10GB Ethernet 64 bit bus. Leg1 uses the lower 32 bis of the 64 bit bus.

In addition to leg number, we have FIFO address of 0 to 64. Each leg has 2*64 channel FIFOs, called FIFO64's , for 128 channels per leg. The channelizer FSM has a table that maps FFT bin to leg, which FIFO 64, and the address, or which FIFO in each FIFO64.

The software roachFFT.py has maps or py dicts, that map bin to channel, leg, and FIFO64 address. roachFFT prgrms the lookup table on ROACH FW that has these mappings.



Firmware schematic. FFT is really 2 FFTs, one delayed to get 2MHz sample rate. Top two FIFO64s give 128 channels to delayed FFT, bottom two FIFO64s give 120 channels to non-delayed FFT.

Channelizer State Machine

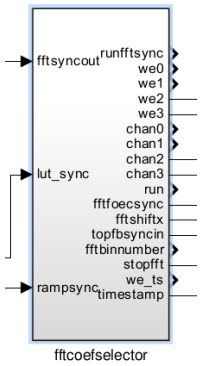
There are several functions of the Channelizer state machine. The design is based on the Ben Mazin firmware, and is written as a set of counters, multiplexers and a RAM for mapping FFT bin number with channel number. Channel number is which channel FIFO the bin gets stored to.

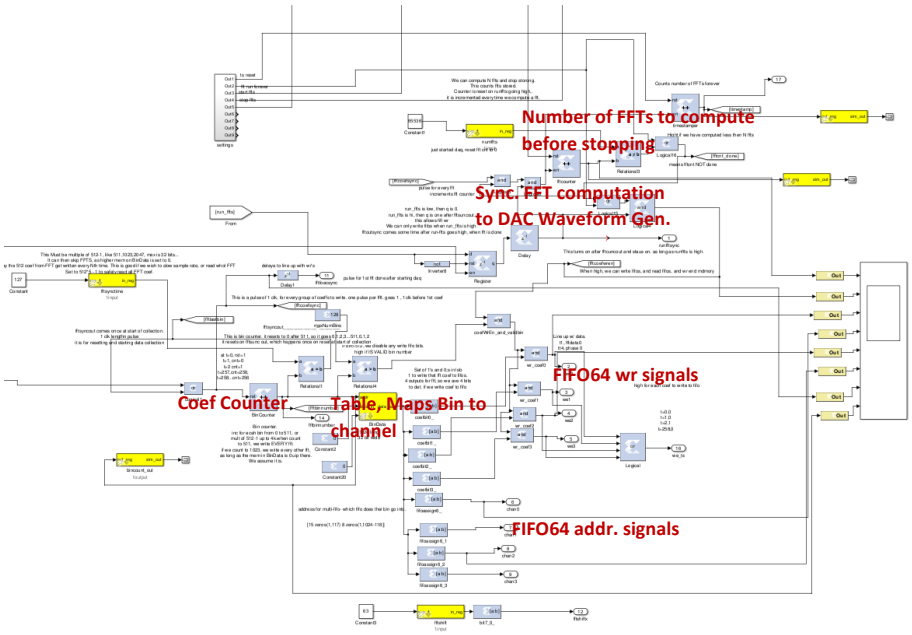
The functions of the state machine are listed:

1) Associate FFT bins with specific channels.
2) Generate FIFO64 write signals to store correct coefficients into correct channels.
3) generate FIFO64 channel signal, which determines which FIFO is written based on channel number.

4) Run FFTs forever, or run N number of FFTs and stop. For frequency sweep of resonator, we run 1 FFT, increase LO, run 1 FFT, increase LO, etc…
5) Synchronize the starting of FFTs with the DAC waveform generator. This provides a needed phase reference between FFT analysis and signal synthesis.
6) Attempt to sync ramp sync signal to FFT. This does not work and is unused.
7) Generate FFT shift signal. This sets signal gain of FFT. Does not seem to work.

A diagram of state machine is below showing inputs and outputs.

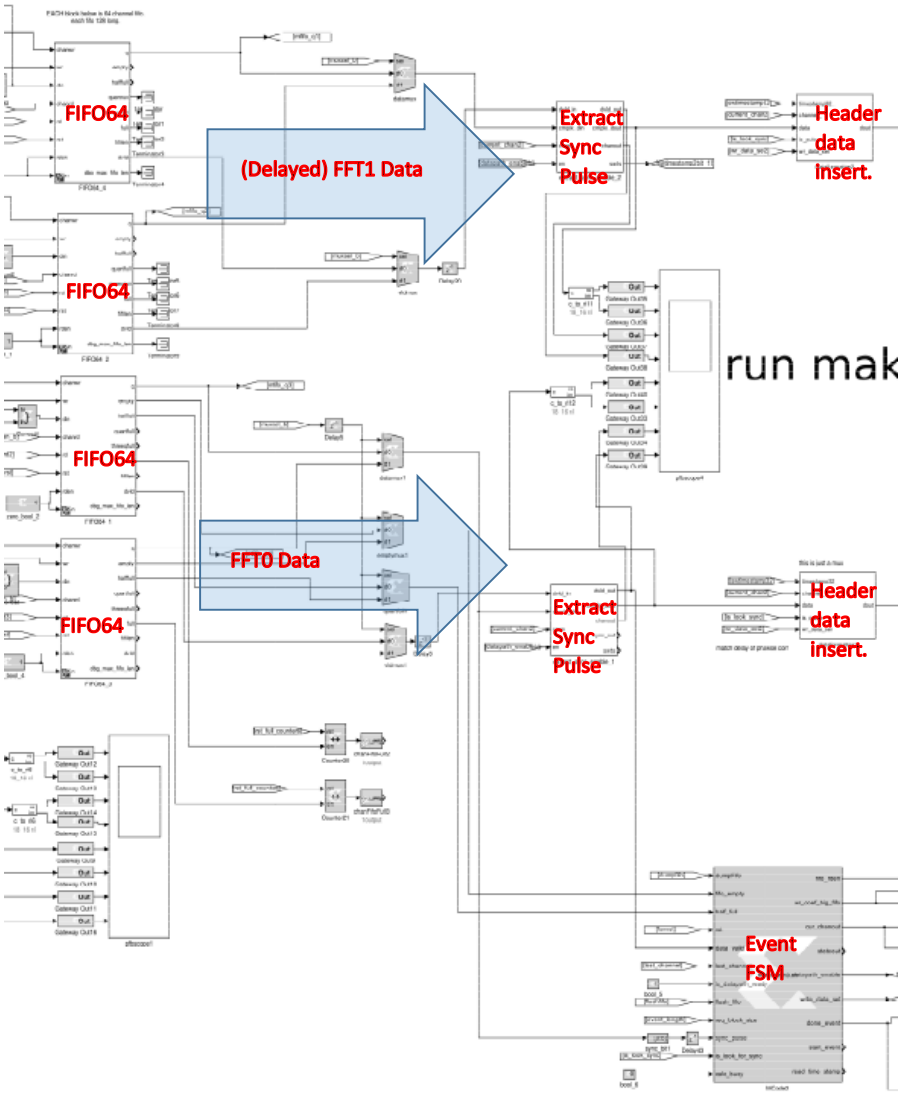




Internal view of channelizer state machine: Counter for FFT bins, so we know which bin is currently coming out of FFT block. A table to map bin number to channel number. Channel number is address signal to FIFO64, and tells which of the 64 fifos are written. We have 2 FIFO64s per FFT block for a total of 128 channels. Also there is logic to start/stop FFTs synchronized to the waveform generator to assure consistent phase reference.

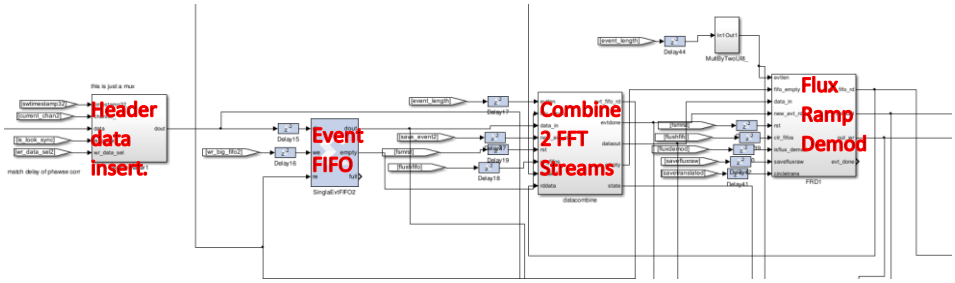
Data Stream after Channel FIFO64s

Below is a view of the firmware showing output of FIFO64s being sent to more firmware blocks that extract the sync pulse and add header data. The Event FIFO FSM controls reading the FIFO64s, inserting header data, and writing to the Event FIFO.



Below is a diagram of the data flow starting at header data insertion. The channel data with header is written to the Event FIFO, controlled by Event FSM. The Data Combiner, reads both Event FIFOs and

combines the streams into 1 stream at twice the sample rate. This stream is sent to the flux ramp demodulator.

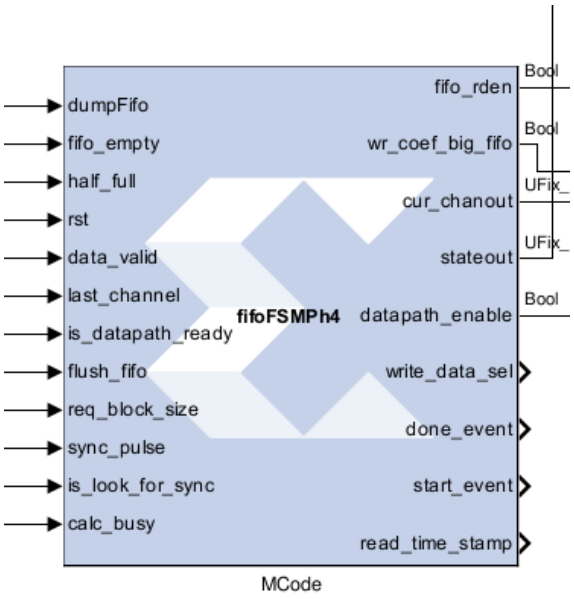


Header Data Insert, Event FIFO

After the FIFO64, which are read out so the stream is from ONE channel at a time, header data is inserted. The header includes 0xaaaa, to mark the stat of an “event” or chunk of channel FFT coef. data for one channel. Also included is a “timestamp”, programmable by python as an uint32, so software can associate some numerical code with event. For example when taking a TES voltage sweep, the “timestamp” is programmed to the TES bias voltage in mV. Perhaps the “timestamp” originally intended in the firmware to give a time, but now is a user-defined number, should have a better name like “user stamp.” Also a but is included in the header data telling of we are in open or closed loop mode, that is if ROACH is sync’ed to the external ramp generator.

After the header information is inserted into the data stream chunk, making and “event,” the event is stored into the Event FIFO.

Event FIFO Machine



The mfile fifoFSMPh3.m is the code that defined the FSM, like a VHDL case statement. Matlab converts this into VHDL FSM. *(Note: the diagram above has fifoFSMPh4, with is not correct. We use fifoFSMPh3.m).*

The FSM reads out the channelizer fifo64s. it stores through channels to readout. If there is data, or enough data, it will readout a chunk of data, of length req-block_size, typically around 100 samples. All samples will be for one channel, or ONE FFT bin, or one TES/Resonator. The FSM also controls logic for inserting header info into the data, like 0xaaaa as a marker for new event (where and event is 100 or so sample sof data from one channel), and channel number. The FSM also handles sync to the ramp generator. The sync pulses are stored in fifo along with data as an extra bit. When FSM is in sync mode, when is_look for sync is 1, it reads the channel fifo throws away data until it finds a sync=1. Then it saves req_block_size samples to output.

FSM inputs

dumpFifo	Bool, 1 to readout out FIFO64 for sending data to output and readout of channels

Fifo+_empty	1 if no data in that channel. Used for clearing fifos.
Half_full	At least ½ full. This is designed to be enough data for 1 event, so data will be read out for channel.
Rst	Reset state machine
Data_valid	Data out fifo output is valid. Use it. Of low, not valid fifo read.
Last_cahnnel	Scan channels 0 to last_channel (up to 128) for looking for fifos with channel data in them.
Is-datapath_ready	1 if we can write to event fifo. 0 to wait.
Flush_fifo	Read out all channel fifo data for all channels from 0 to last_channel. Throw data away. Clears all fifo 64s.
Req_block_size	Number of samples in 1 event. One event is one flux ramp demod calc. typically around 100 samples.
Sync_plse	Sync pulse read out from fifo64.from external ramp generator
Is_look for sync	1 for closed loop, where we read out 1 event on sync pulse from ramp. 0 to read out 1 event, throwing away no data.
Calc_busy	Downstream calculations are busy. Is this used?

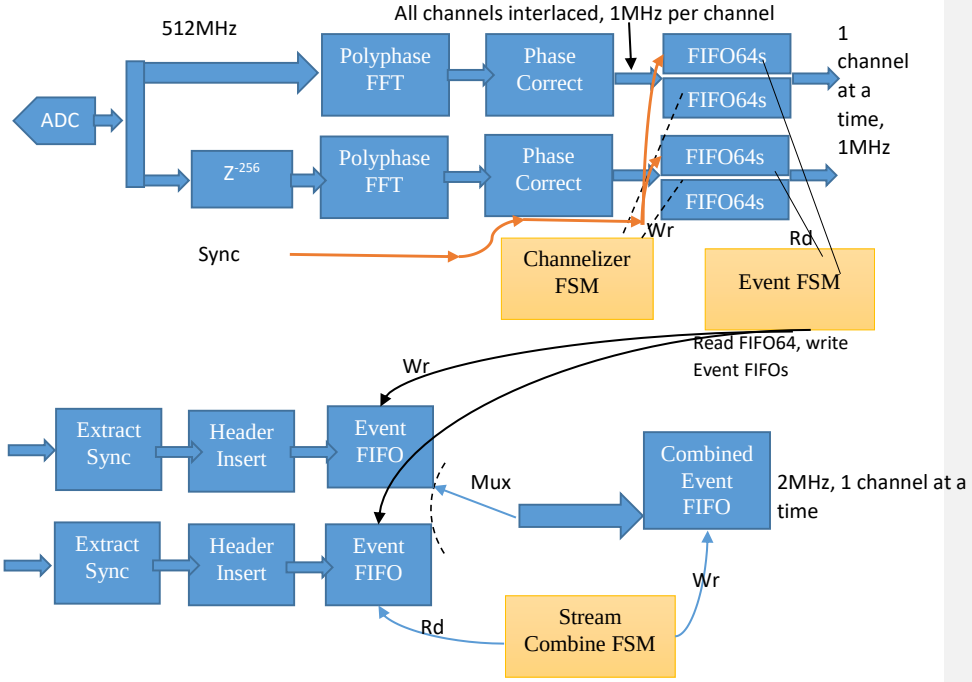
Event FSM outputs

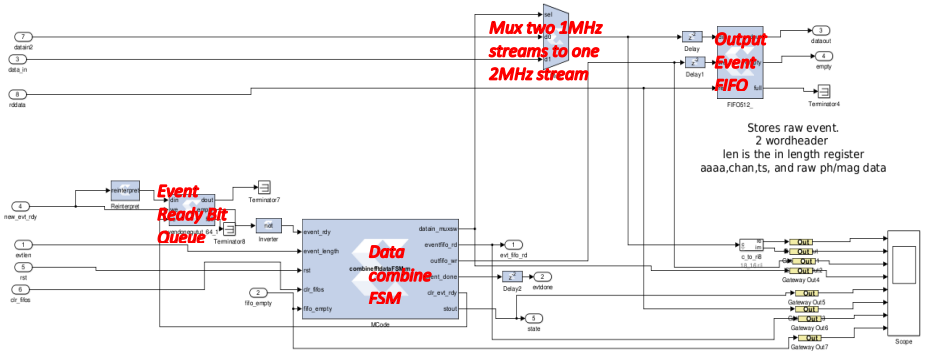
Fifo_readen	bool. 1 to enable reading of channelizer fifo64. 0 to not read.
Write big coef fifo	1 to write data from channelizer fifo64 to event fifo.
Curchan_out	Int7 0 to 127. Which channel fifo we are reading.
State out	Current state of machine, for debugging.
Datapath enable	1 to enable writing to downstream datapath.
Write_data_sel	Mux control signal to write header data or fifo data to event fifo. 2 bits.
Done_event	1 for 1 clock when done writing 1 chunk of data from 1 channel, and associated header data. This signal is sent to downstream hardware to tell that that a complete event is available for processing. Flux ramp demod gets this signal.
Start-event	1 for 1 clock at start of writing of new event. That is header and fifl data.
Read time stamp	1 to read timestamp into the event header data.

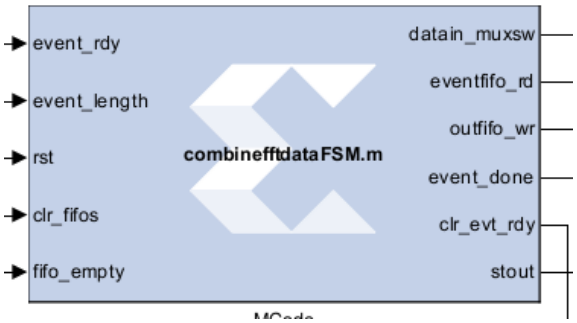
Data Combiner

The data combiner is a state machine that multiplexes data from two Event Fifos, corresponding to the two FFT processing engines; the first FFT and the second time-delayed FFT. Below is a more detailed diagram of the data stream from ADCs through FFT, Phase Correct, FIFO64, Sync extract, Header insert, Event FIFO, to data combiner.

Each FFT outputs coefficients at a sample rate of 1MHz per channel, with all channels interlaced into a 512MHz stream. After the FIFO64s, we read out 1 channel at a time, with a 1MHz sample rate. The data combiner combines the two FFT streams, each at 1MHz, into a single stream at 2MHz sample rate.







The Data Combine State machine

The data combine state machine controls the process of combing two 1MHz streams to one 2MHz stream. It controls reading the upstream Event Fifos from both FFT streams, parsing the header data in the streams, and storing a combined event in the output event FIFO. When the upstream Event FSM is done writing an event to the Event FIFOs (one per FFT stream), it stores a bit in the Event Ready bit Queue. The Data combine FSM gets this bit from the queue, and it knows a complete event is ready to be read from the upstream Event fifos. The FSM then reads the upstream FIFOs, and interlaces the data into one stream, and stores it to the Output Event FIFO. When the event is completely written, the FSM sets the done bit high, telling the next stage in the data stream that data is ready.

Data Combiner FSM inputs

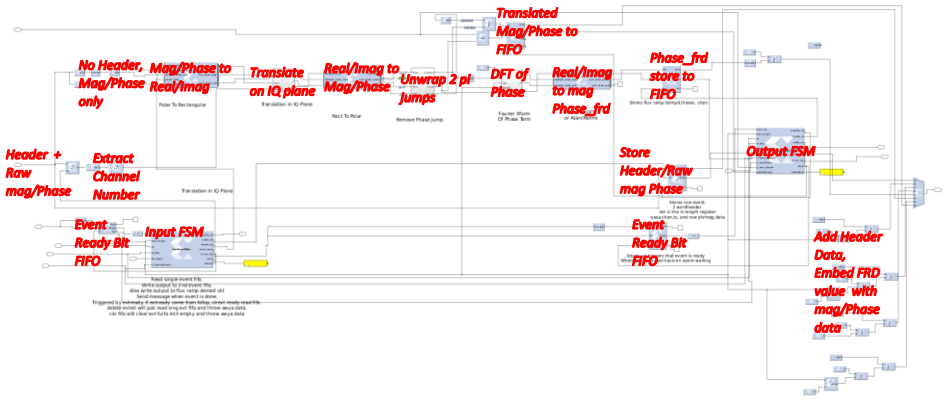
Event rdy	Bool. 1 if upstream data fifos have complete event ready to be read.
Event length	Uint8. Number of mag/phase pairs in event.
Rst	Bool 1 to reset the fsm
Clr_fifos	Bool. 1 to clear the output fifo
Fifo_empty	Bool. 1 means upstream data fifo is empty

Data Combiner FSM outputs

Datain_muxsw	Bool. Control mux that switchs which input data stream we read from.
Event fifo rd	Bool 1 to read input upstream event fifo
Output fifo wr	Bool 1 to write to output combined event fifo.
Event done	Bool. 1 to signify that FSM is done writing a complete combined stream event to output fifo. Ready for processing by next stage of data path.
Clr evt rdy	Rad out bit from input event bit queue.
Stout	Debugging. Which state the FSM is in.

Flux Ramp Demodulation

The flux ramp demodulator is shown below.



The data coming into the flux ramp demodulator is a header of 2 32 bit words, followed by mag and phase data of 16 bits each. The length of the mag/phase data is set by an input wire, called event length,

in 32 bit words. Here is the format of the data. The mag/phase data is sampled at 2MHz, and all corresponds to ONE signal channel. The FRD will process an event of one channel, then an event of the next channel and so on. Each event has up to 255 mag/phase samples.

Data format coming into Flux Ramp Demodulator.

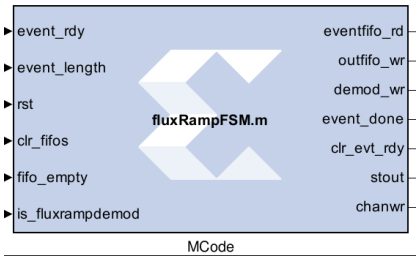
userTimestampMSW_0xaaaa	User timestamp msw is to 16bits of user timestamp. Lower 16 bits is a marker of 0xaaaa
85_UserTimeStampLSW_isPulse_channel	From bit 31 downto 0, we have decimal 85 of 7 bits. Then we have 16bit usertimesample LSW. Then we have a single bit, meaning we are in sync-to-ramp generator mode if 1. Then we have 8 but channel number.
Magnitude_Phase $_0$	We have upper 16 bits of magnitude, Uint16_16. It is a real number wher decmal point is to the RIGHT of but 16. Therefore the range of magnitude is from 0.0 to .999999. The lower 16 bits is Phase in int16_13. Bit 15 is the sign bit. Decimal point is to RIGHT of bit 13. Therefore the values range from -4pi to +3.9999pi.
Magnitude_phase $_1$	
…	
Magnitude_phase $_{Evt\_len-1}$	We have Event_Length mag/phase pairs. Event_len is a wire going into flux ramp demod module, programmed from python.

FRD Input State machine

The purpose of the FRD input FSM is to control the reading Event data from the upstream combined event FIFO, before the FRD circuit in the Firmware, and to control the Flux ramp demodulator. There is a bit event queue that has a 1 bit stored to it when Event Fifo contains a complete event, and is ready to be processed by the FRD. When the bit appears in the queue, the FSM clears that bit and initiates a FRD computation, reading channel data from upstream event FIFO into the FRD circuit. The reason for a event bit queue is similar to a computer GUI: when you hit the mouse 20 times, those mouse clicks are queued for eventual processing. The bit queue prevents the FRD circuit from missing events as they can be queued. The FSM does:

1) Take event bit off of event queue, to read Event FIFO when upstream event FIFO has a complete event stored
2) As upstream event fifo is read, extract mag/phase data to send to flux ramp demod computation, with no header. The FSM writes the correct number of words to the FRD.
3) Extract channel number to send to I/Q translation.
4) Send header and mag/phase data ahead to a storage FIFO. The correct number of words are written to send complete event data.
5) Store a data ready bit to outpt event ready fifo, telling next Output FSM that event was sent to FRD computation, and stored into storage FIFO.

The FRD input FSM is below



Inputs to the FSM

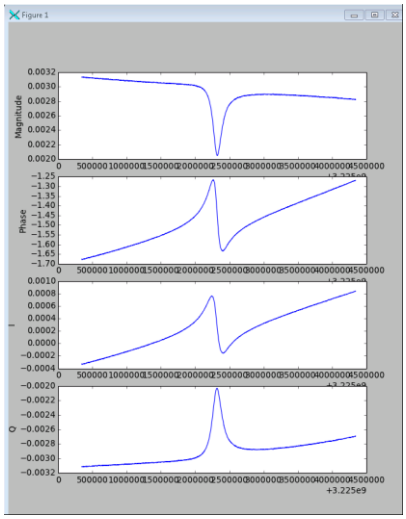
Event_rdy	Bool. 1 means event is read to read from event fifo into FRD.
Event_length	Uint8. Event length. Number of mag/phase pairs in event.
Rst	Bool. Reset FSM
Clr_fifos	Bool Clear the output fifo, FRD fifos.
Fifo_empty	1 if Event FIFO, that inputs to FRD circuit, is empty.
Is_fluxrampdemod	Bool. 1 if we do frd calc. 0 of we do NOT do frd calc, but only send mag/phase data down the data stream raw.

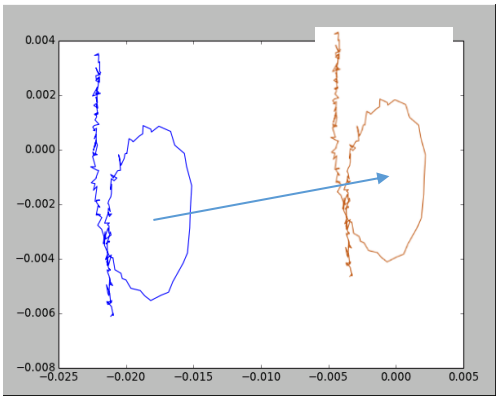
Outputs of the FSM

Eventfifo_rd	Bool. 1 to read event fifo data into FRD.
Outfifo_wr	Bool. 1 to write event fifo data to FRD output fifo.
Demod_wr	Bool. 1 to write event data to FRD calc. circuits.
Event_done	Bool. 1 to write to downstream that event has b een written and processed by FRD.
Clr evt_rdy	Bool. 1 to acknowledge it got an event bit ready from upstream event read bit queue. Clears this bit in event bit queue.
Stout	Debugging. Output of current state
Chan_wr	Bool. 1 to write channel number to a register. Channel number is neede by I/Q circle translator so we use the correct IQ circle center or channel.

FRD: IQ Circle Translation

When a resonator is swept by a network analyzer to find the frequency response, both the phase and magnitude is measured. And example sweep is below:

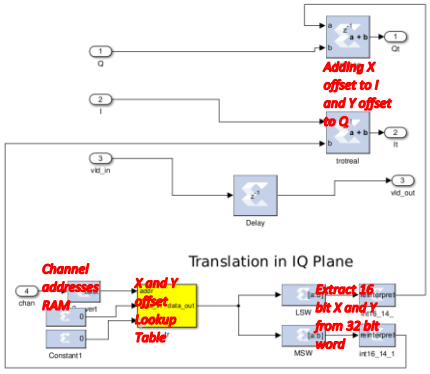




ON left wE have from top to botton, Mag, Phase, I and Q. WE can plot I vs Q, or Real vs Imaginary. This is on the right above. The Red circle is a translated version. The Translator circuit in the FRD does the following:

1) Convert Polar to IQ.
2) Add and X and Y offset to all data to translate the data to origin.
3) Convert IQ back to Polar.

The translator must store X and Y offsets for every channel. For this reason the channel must be known, and mst be used to reference a look up table to find the X and Y offsets so the data is corre tly translated.



Above is the firmware for the translation of the IQ data to origin. The lookup table, a RAM addressed by channel, stores the X and Y offsets. These offsets are added to I and Q.

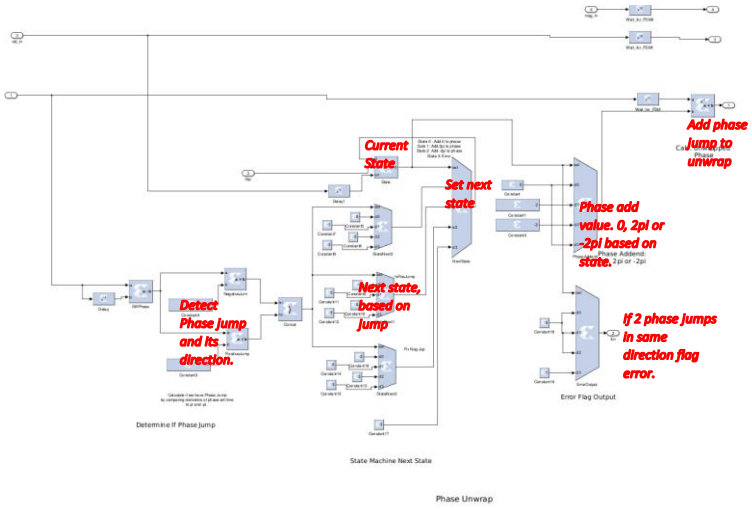
Phase Unwrap Circuit

Phase unwrapping is the removal of jumps in phse from –pi to +pi or +pi to –pi. This is done by adding +/- 2pi the the phase to remove the jump. The circuit must have an element that detects phase jumps, which is done with a subtraction of successive phase terms in time, and comparing to some number, above which meqans a phase jump. IN the roach firmware, phase is not in exact radians. Rather phase is measured in radians/pi. Therefore, 2.0pi is represented as 2.0. The circuit must be a state machine in 3 states:

1) Add 0 to all phase numbers.
2) Add +2pi to all phase numbers.
3) Add -2pi to all p[hse numbers.

If we get a phse jump, and are in state 1) we then jump to state 2 or 3 depending on the direction of the phase jump. The result is the the phase jumps are removed. A potential problem is that we get 2 jumps in the same direction, and need to add 4pi to the phase. This impossible because phase is stored in range from -2 to +2, so only one jump is possible in the same direction.

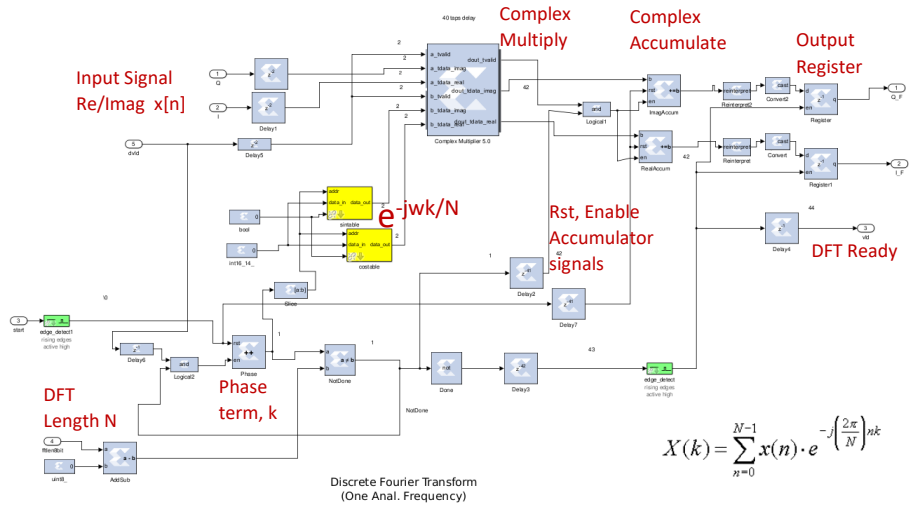
Below is the circuit for phase unwrapping.



Discrete Fourier Transform block

The Discrete Fourier Transform is computed on the phase term of the translated resonator stream data. The idea is that when the SQUID is stimulated with a ramp, the phase term will look like a few cycles of cosine over the length of the event, about 100 -200 samples. When the TES catches an x-ray, the phase of the cosine will change based on the height of the pulse. The DFT will measure the phase of the cosine. The circuit is a brute force method of computing a DFT, for one frequency only. RAMs store the sin/cos coefficients. A complex multiplier multiplies the input signal by the coefficients, and an accumulator adds up the result. A counter addresses the coefficients and sets the done flag when the result is ready.

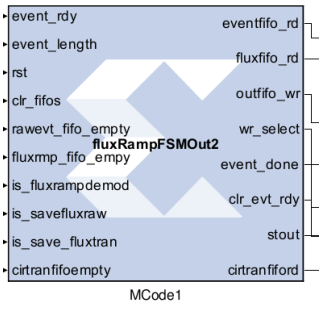
The output of the FFT is a rectangular representation of the magnitude and phase of the cosine from the SQUID. A CORDIC block converts the mag/phase, and only the phase term is kept. The phase term is analogous to the TES pulse height, or the TEs pulse signal magnitude with respect to time. This phase term, output fro the CORDIC block after the DFT, is the final flux ramp demodulated signal. Because there will be periodic phase jumps from –pi to pi, the QT C++ data receiver program performs real time phase unwrapping on the FRD signal in the linux box.



FRD Outpit State Machine

The FRD output state machine performs the following:

1) Read result of the FRD computation from a FIFO and output to downstream circuits.
2) Add header data to events: 0x5555 and length of event. Encapsulats the FRD result, and raw event data, or translated coefficient data.
3) Writes different kind of output event data: FRD only, Raw data only, FRD+Raw, FRD + translated data. Sets a flag in the header denoting what kind of event it is.



Inputs to FSM

Event rdy	Bool, Event data ready from input frd FSM
Event length	Uint 8 Length of event data in samples
Clr fifos	Bool, 1 to clear fifos
Rawevt fifo empty	Bool, 1 if Raw event fifo m
Fluxrmp fifo empy	Flux ramp demod data fifo empty
Is fluxrampdemod	1 if we are saving FRD numbers
Is save flux raw	1 if we are saving raw event data
Is save flux tran	1 if we are saving translated fft coef.
cirtranfifoempty	1 if translated fft coef fifo is empty

Commented [MTJ1]:

Eventfifo rd	Bool. 1 to read from upstream event fifo
Flux fifo	Bool. 1 to read from FRD data fifo
Outfifo wr	Bool 1. 1 to write to output
Wr select	Uint3. Control mux to write header numbers, FRD result, raw or translated FFT coef. Data.
Event done	Bool. 1 when we are done writing a complete output event with FRD and header data.
Clr evt rdy	Bool. 1 ack we received event ready, and clear bit fifo.
Stou	Debugging. FSM state.
Cirtranfiford	Bool 1 to read translated data fifo.

10GB Ethernet

Dataformat Sent over 10GB Ethernet

A complexity of the data format is that the UDP packet length is independent of the chunks of roach data transmitted to a linux system. Therefore there are multiple levels of protocol in teh data format.

Roach has three levels of protocol.

ROACH Data level 1

First there is the standard UDP protocol, using packet sizes of approx. 1500 bytes, set in the FW by a FSM called XXXXX. A packetizer FSM in the FW firsts sends zzzzz's, one 64 but word, the a packet number. The data fifo is read out containing roach data, that could be anything, and independent of the udp packets. Finally a XXXXX is insert3ed at end of the packet.

IN the linux system receiving the packets, there are two levels of parsing. First, packets are decoded and zzzz's are removed. Then underlying data is sent to a software queue for further parsing.

Format of UDP packets from ROACH. Packet count is simple counter from 1…2^32-1, that counts packets. 10Gb Ethernet has a 64 bit wide data bus. The UDP packet is 180 64 bit words long, or 180*8=1440 bytes.

0	0xffff_ffff_ffff_ffff
1	0xffff_ffff_ffff_ffff
2	Packetcount32_packetcount32
3	ROACH DATA Level2
4..178	ROACH DATA level2
179	0xffff_ffff_ffff_ffff

Roach Data Level 2

Roach Data comes in a different protocol sitting on top of the above UDP packets. Roach data packets are different in length to the UDP packets, and will overlap or continue between UDP packets. There are several different types of ROACH packets depending on if we return FRD data, raw coeff. Data, or both.

ROACH data is in 32 bit words.

0	0x5555_uint8a_uint8b	A=event length, B=eventtype
1..event_length	ROACH data level3	

Roach Data Level3

There are several types of Roach data level3, defined by event type above.

Type 0,1

0	Uint16_0xaaaa	Upper bits =- timestamp MSW
1	Uint1_Uint8	

2…eventlen-2	int16_int16	Magnitude is upper 16, Phase is lower 16

Type 2

0	Uint8_uint24	Flux ramp dempd phase lower 24 bit
1	Uint16_0xaaaa	Upper bits =- timestamp MSW
2	Uint1_Uint8	Channel, timestamp, is sync mode.

Type 3

0	Uint8_uint24	Flux ramp dempd phase lower 24 bit
1	Uint16_0xaaaa	Upper bits =- timestamp MSW
2	Uint1_Uint8	Channel, timestamp, is sync mode.
3…eventlen+3	int16_int16	Magnitude is upper 16, Phase is lower 16. Raw coeff data.

Type 4

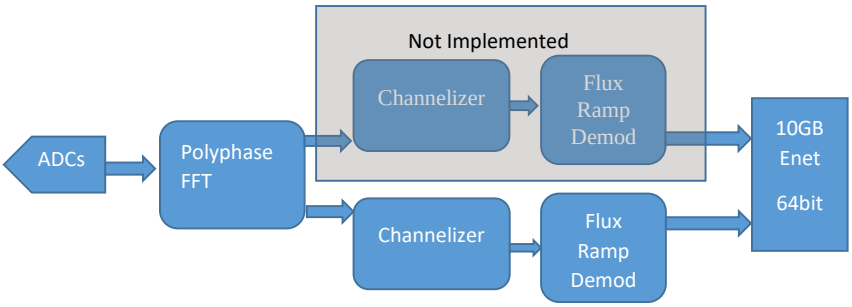
0	Uint8_uint24	Flux ramp dempd phase lower 24 bit
1	Uint16_0xaaaa	Upper bits =- timestamp MSW
2	Uint1_Uint8	Channel, timestamp, is sync mode.

3…eventlen+2	int16_int16	Magnitude is upper 16, Phase is lower 16. Translated coeff. Data.

These formats are defined in firmware, roachParser.cpp, roachParser::parseStream(void), and in dataExtract.py.

64-bit words in 10GB Ethernet

10GB Ethernet has the feature that it is decoded as a 64 bit parallel bus. IN the ROACH. These 64 bit words are split into two independent 32 bit streams, allowing two independent channelizers and flux ramp demodulator systems, thus making ROACH more parallel. Currently, only one flux ramp demodulator and channelizer is implemented in ROACH allowing 128 channels. In this way, only 32 bits in the 10GB Ethernet is currently used, with the unused bytes filled with z characters. The parser that run on linux splits all UDP packets into two independent 32 but wide data streams that are sent to two separate ROACH parser threads and function independently. Because only one half of the data stream is used, only one parser is currently doing anything.



The ROACH is designed to allow two parallel channelizers and Flux Ramp Demodulators. Currently only one of each is iomplemented, using only half of the 10GB bandwidth.

Oscilloscope

For debugging purposes raw ADC data or raw FFT coefficients can be captured to RAM and plotted by the python software.

Programming IF board

Clocking and LO

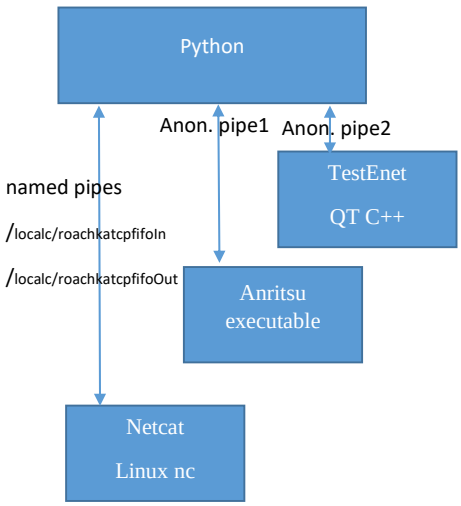
The ROACH IF Board has oscillators built-in for LO (~5GHz) and System Clock (512MHz). These oscillators are PLL-based ICs on the IF board, and are currently broken, due to connection to external hardware with improper grounding. *Because these chips are broken and need to be replaced, we currently use an Anritsu source for LO, and an external 512 MHz clock for system clock.* The local oscillators were damaged by not grounding the ROACH system to common ground, while connecting a 10MHz reference to the front panel on the ROACH. The switching power supply causes large voltage swings (20V) on the ROACH chassis if the chassis is not grounded. These swings damaged the PLL oscillators.

To use the internal oscillators for LO and System clock, the chips could be replaced. To use internal oscillators for LO and System Clock, an external 10MHz reference frequency must be supplied. It is possible to use one internal oscillator and one external oscillator for LO and/or System clock. That is, each internal oscillator is independent of the other, and can be chosen by programming the IF board with a serial data stream, from ROACH FPGA, controlled by python QT GUI running ROACH.

The LO is currently supplied by an Anritsu source at LO frequency of around 5GHz, and at -5dBm. Do not increase output power of Anritsu above -5dBm or chip U35, a PSA-12 amplifier, will be damaged and need to be replaced. The Anritsu should ber connected to a common 10MHz reference frequency.

The system clock is supplied by a XXX source at 512MHz at -5dBm. Do not increase power above -5dBm. The 512MHz directly clocks the ADCs and DACs in the ROACH system at that sample rate, and also is dived down to 128MHz to clock the ROACH FPGA. The sustem clock should be connected to a common 10MHz reference frequency.

Overview of Software



Data Receiver

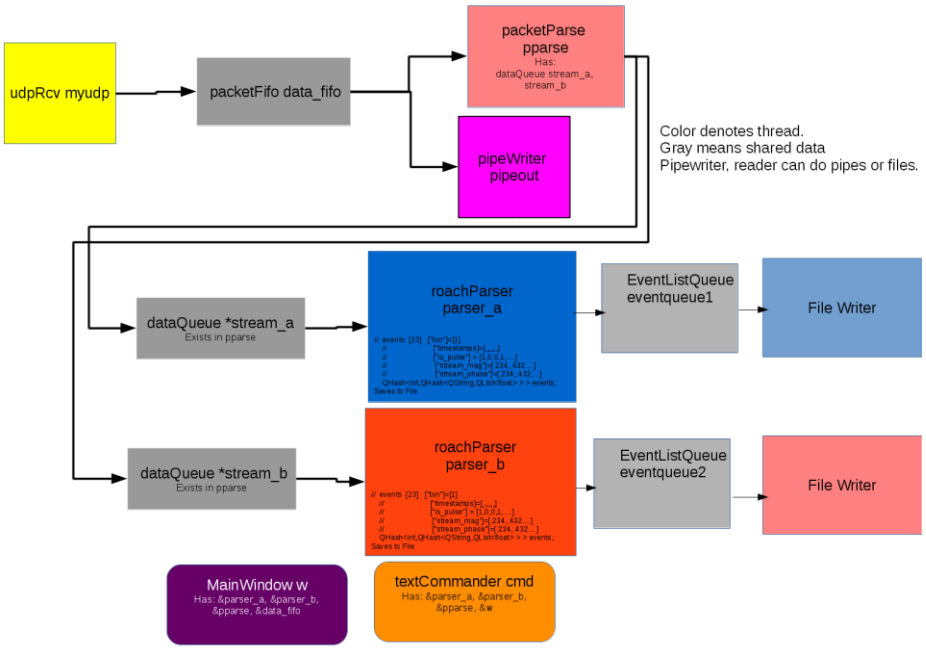
QT C++ is used for the Data receiver from ROACH. The data receiver receives data from RACH via 10GB Ethernet UDP. It stores the packets in a Queue. Other threads parse the UDP packets into separate data streams corresponding to channels on the TES's read by the ROACH. A final thread writes the data to HDF5 files in real time. The Receiver program can be controlled by python via connecting python to the Receiver stdin and stdout, using the python subprocess module. Python sends ASCII text to and from the Receiver through the pipe opened by the subprocess module. Data is transferred from the Receiver to python by saving the data disk from C++ and reading into python. For lack of a better name, the app exe name of the Data Receiver is testEnet. Sorry for the bad name. The window that opens is named "Mainwindow." Sorry again.

Source code is in RoachFirmPy/Roach2DevelopmentTree/QT/readRoachStream

The QT project is testEnet.pro. Use QTCreator to open the project and build. Binaries get put into RoachFirmPy/Roach2DevelopmentTree/QT/ build-testEnet-Desktop-Debug. The exe is testEnet in build-testEnet-Desktop-Debug.

A block diagram of the Data Receiver is shown. A udpRcv object captures UDP packets from the 10GB Ethernet card and places the data into a packetFifo. The packets are then parsed, a first level parsing. Because 10Gb Ethernet provides 64 bit works, the words are split into 32 bit words and sent to two

different queues, or dataQueue objects. The idea is that ROACH has two simultaneous data streams (though as of this writing, one stream used, meaning that half of the 10Gb Ethernet data is just place holders. In a future FW build, this will be fixed). I this way two FFT channels can be processed in FW at once, and passed to the 10GB Ethernet as the high or low order 32 bit word.



The DataReveiver is controlled by python using

Python Files

All py files are loaded into python with execfile rather than import. The reason for this is that

1) Import puts all py files in a new name space of the module name unless you do from XXX import.

2) If you are debugging, importing a file a 2nd time to fix a bug does not necessary import everything and overwrite old code in python memory space. Execfile always overwrites old code when we rerun execfile.

To start the software:

Run ipython in anaoconda

Execfile('natAnalGui.py')

Main()

You will see gui.

To run w/ epics interface

Mainepics()

A soft IOc comes up. All eopics db are created on the flu w/ python and all records are soft records. Python does all the work of monitoring pvs and writing to pvs. The IOC does nothgin but host PVs that do nothing but provide interface to epics MEDM.

katcpNc.py

This file is an APS-written implementation of the katcp protocol used by ROACH. ROACH has a katcp server on its power pc that makes a tcp connection between ROACH and linux box. Katcp is a mixed text and binary format for transmitting commands and data between roach and linux. All registers and control are done w/ katcp. Corr is the casper implementation. We implemented our own because corr at the time did not support uploading the fpga programming files. It now does, but we still use katcp. It is all python, and easy to import.

The APS version of katcp uses netcat (linux nc command) as a network program that opens the TCP connection. Python sends and receives data to nc over named pipes on the file system. The pipes are hard-coded into katcpNc.py with this code:

```
def startNc(self):

    try:
       os.system('mkfifo /localc/roachkatcpfifoOut')
       os.system('mkfifo /localc/roachkatcpfifoIn')

       os.system('sleep 9999999 > /localc/roachkatcpfifoIn &')
       os.system('sleep 9999999 > /localc/roachkatcpfifoIn &')
    except:
       pass
    os.system('nc %s %d > /localc/roachkatcpfifoIn < /localc/roachkatcpfifoOut &'%(self.ip,self.port))
    self.In_pipe = open('/localc/roachkatcpfifoIn','r');
    self.Out_pipe = open('/localc/roachkatcpfifoOut','w');
    print self.In_pipe.readline()
    print self.In_pipe.readline()
```

Netcat or nc is started as a separate process taking to the named pipes to get to python. Two linux sleep 99999 commands connect to write to the pipes to make sure pipes are always left open and do not close. This allows python or nc to start and stop without pipes closing.

Example code using katcpNC

```
execfile("katcpNc.py")
roach2=katcpNc()
roach2.startNc()

roach2.fpgaStatus()  #see fpga status
roach2.help()
#to send fpga prog file
roach2.sendBof("/home/oxygen26/TMADDEN/ROACH2/projcts/bestBitFiles/tengbtest_2015_Mar_31_1055.bof")
#list and read all registers on roach.
roach2.listReg()
roach2.readAllReg()
roach2.closeFiles() #close connection to roach
```

Setting up the 10Gb ethernet on roach box is done w/ katcp. The mac address and ip address is set.

Agt33250A.py, sim928.py

These py files open a serial port to talk to the sim928 voltage source and Agilent 33250A waveform generator.

When talking to sim928, the system 1st connects to sim900. Then you must manually call

Sim.connport(8) to connect to sim928, which is in slot 8 of sim900. The GUI in natAnalgui should do this for you but it does not.

Both sim928 and agt33250A have classes for actual connection to the external hardware via RS232, and NULL classes that do nothing. The NULL class allows the software to function when no external unit is present. You need write access to the serial ports, which are

/dev/ttyS0 for sim928, and /dev/ttyUSB0 for AGT.

Anritsu.py

Antirtu.py is a python module that runs an external C++ program as a subprocess, and interacts with the C++ program's stdin and stdout via an unnamed pipe opend by the python subprocess module. Text is sent through these pipes to the C++ program, called vx11/vxi11_1.10/anritsuOsc. anritsuOsc opens a telnet connection to the anritsu oscillator via a private Ethernet network. The anritsu supplies the 5GHz Local Oscillaotr or LO signal to operate the mixers in the ROACH IF board. Because anritsuOsc is a subprocess to the pytghon process, hitting Ctrl-C in the python shell will shut down the anritsuOsc program.

dataCapture.py

This module controls the Data Receiver C++ program called testEnet. testEnet runs as a subprocess to python, and python controls it by sending and receiving ASCII text commands via an unnamed pipe to the stdin/stdout of testEnt. testEnet has a thread that listens to the command pipe. Because QT has reflection, that is, any slot in QT can be called with a text string, any slot in testEnt can be called by sending a text string to testEnet over stdin. Hitting Ctrl-C in puython kills teetEnet.

testEnet captures data from the ROACH via 10Gb Ethernet. The ROACH data is based on channels, one per TES in the detector. Events are chunks of data from each channel. The event data are organized into structured dictionaries, like python dictionaries, but in QT C++. Each channel has its own data stream and is organized into structured data. This structured data is streamed real time to HDf5 files as ROACH data is collected. Because there are two parsers that parse roach data, two hdf5 files are streamed at once, one called xxx_A and one called xxxx_B.

when python programs testEnet to capture data toa file, the data is stored in a ../temp/xxx_A.hdf file and B file. Python then renames the file to the choice of the user. Because only one parser currently gets any data, only the A file is saved.

Python can only get ROACH data from files. For puython to plot a resonator sweep:

1) Set up testEnet to capture data to hdf5 temp file.
2) Make roach sweep the resonator, and steram to testent, which saves the temp file

3) Python reads the temp file from disk and plots.

dataExtract.py

A python parser for debugging roach raw binary data. Use nc to capture to binary file. Python to parse and plot. Also can create roach binary data fir simulation.

Top of py file has examples on how to use.

Best use4s for this file:

1) Decode raw binary roach data captured w/ nc. Plots FRD and compares FPGA FRD calcs to software FRD calcs. Tests to see if FPGA is working properly.

2) Get roach iq data from hdf file. Compute raw roach data stream from iq data, to simulate roach hardware. Use nc to send this raw data to the QT C++ data receiver program, to test data receiver.

To capture roach raw binary stream to binary file:

nc -ul 192.168.1.102 50000 > myfile.bin

To plot FRD data and see if its correct:

```
execfile('dataExtract.py')
dd=dataExtract(None,None)
fn = 'myfile.bin'
magphs = dd.readBinFile(fn,whichstream=1,blocksize=65536,foffset=0)
events = dd.extractEvents(magphs,is_pause=False)
dd.plotEvents2D(fignum=5, chans=-1,data=events,stevent = 0, nevents=1000)
```

hdfSerdes.py

In python it is possible to use the pickle module to save a python object to a file, called a pickle file, whcu can then be read again and restore the object,. hdfSerdes.py allows the saving of a python object to an hdf5 file. It works by writing groups and datasets and attribut4s. Each group or dataset has an attribute “type.” Type is a string naming a python type. For saving a python list, a dataset with the type attribute “list”, is written as an array. Dictionaries are groups that have groups in them, with each rgroup being a dictionary key, with type like “key_int” or “key_string.” Not all python types are

supported but the most used ones are. Also python classes can be saved, as long as the class definition is in the memory name space. These classes can be restored as objects, if the class resides in memory.

hdfSerdes is used for saving and reading hdf5 data by the ROACH software.

Example code is in the top of hdfSerses.py.

roach_calibration.py

This file has calibration data for roach. At the top of the file are magic numerical constants that are typed in and saved. At the bottom is code that computes more numbers based on the numbers on top. This file is loaded and used for plotting and taking roach data. Below are example numbers for calubraion. They include the boltage outputs of the DAC, voltage input level of ADCs, and various menu setting s for flux ramp demodulation that show up in the Gui.

```
cal_ivstream_voltagestep_delay=1000

flux_ramp_cal_settings = {
   '10k_3V': {'frddly':111, 'freq':10000, 'volts':3.0, 'periods':3.0 , 'frdlen':153},
   '40k_2V': {'frddly':34, 'freq':40000, 'volts':2.0, 'periods':2.0 , 'frdlen':40},
   '40k_3V': {'frddly':34, 'freq':40000, 'volts':3.0, 'periods':3.0 , 'frdlen':39}
   }

# preferred tone power at resonator dbm
tone_power_at_resonator_dbm = -85.0

#hard codeed, watts output for full scale sinewave.
dac_watts_sinewave = 0.00198
#max amp 0p in counts outptu from dac
dac_max_counts_0p = 32766

#12 buit adc 0-p
adc_max_counts_0p=2046
#sine wave fill scale, W
adc_watts_sinewave = 0.00427

#mixer input desired powerin dbm
in_mixer_desired_power_dbm = 1.0
```

analysis.py

this file has several computation for analyzing roach hdf5 file data. These scripts are run by hand when looking a data and analyzing. Not really part of roach software, but useful for understand data from ROACH.

Debug.py

This file are debugging routines used when developing and debugging ROACH FW and software. Often these functions are eventually added to the normal roach software gui. Used for development of software and firmware.

Fitters.py

Fitters is for fitting curves to resonator sweeps. When a resonator has a frequency sweep, Lornenz fits, max velocity fits can be run to find exact resonator center. This code was adapted from Tim Cecil matlab code. It was mostly used for MKID detector development, and has since fallen out of use.

The fitters still work.

fluxRampGen.py

The firmware was an experimental signal-delta based flux ramp generator, to obviate the need for the external ramp generator. This is not used. One function setChannelizerFifoSync, is still used in the roach code, for setup up sync mode in data collection. This is closed loop mode, where roach is synced to external ramp generator.

if_board.py

This code is for programming the if_board, the MUSIC board. The board has 2 PLL based oscillators, 3 attenuators, and several RF switches. There are object/classes for rf settings, atten settings and osc settings. The if_board object has references to these settings. Calling if_board.progIfBoard sends all the settings over a serial stream (via ribbon cable) to the if board.

The board has two oscillators, or PLLs on board, that can function as the LO (or mixer source) and the 512MHz FPGA ADC clock. A class clkGenSetting is filled in with function setFreq. The if_board obhect has references to both oscillators:

If_board.s, the fpga clock

if_board.Lo, the LO clock

You create the if_board object, which creates the clkGenSetting objects. Then you call if_board.s.setFreq and of_board,.LO.setFreqm then if_baord.progIfBoard()

Likewise, there is an rfSwitchSetting object you can set up. For setting the rf switches. Finally, there are 3 attenuators. The settings are in attenSettins. Set up this object and then program the if board.

Example code exists at top of file.

mkidMeasurepy

this code is complex measurement code for using and calibrating the ROACH.

There are two classes. measSpecs is a group iof settings to store data to define a measurement, and to store results of measurement. Mkidmeasure is a class that contains measurement functions for controlling roach.

Measurements that are included:

1) Power sweep of resonators.
2) Voltage sweep of TES for one or several resonators at once. Def voltSweepRes, def voltSweepFast
3) Frequency sweep of groups of resonators. Def sweepResonators
4) Managing programming the Translator circuit. The translator circuit is the firmware that allows the roach to center the IQ circle on the origin. Circle centers per channel, are stored in a RAM table on the roach. mkdMeasure has code for measuring these iq circles, programming the ROACH, and saving the cal data.
5) Extracting IQ sweep data from raw roach data, getIQSweepFromRaw.
6) Extracting Resonators from a sweep of ,any resonator frequencies: getMKIDsFromMultiSweep
7) Calculating the transmission line delay between roach and cryostat.
8) Plotting functions.

mkidDac.py

The DAC on the ROACH accepts serial download to set up its PLOL. It is not generally ised. DAC and be reset and sync'ed. For controlling and setting up DAc on RAOCH.

phaseCorrect.py

The phase correct firmware is a final modulation after the FFt operation. Because the sourced frequencies for the DAC, matched to resonator frequencies, do not land exactly in the FFT bin centers, the phase corrector modulates the signals to land in FFT bin center. When the signal in FFT bin center, the phase term becomes a constant in time, if there is no SQUID signal. The phase correct circuit adds a ramp signal to the phases of the FFT coefficients to make the phase term constant in time. This is done

with an accumulator, a table of phase increments and current accumulator values per channel. The python script calculates these increments base on source frequency, FFT bin center frequencies. Also the module programs the ROACH phase correct circuits.

Qdr.py

This module, adapted from CASPER group code, is used to calibrate the QDR chips on the ROACH. Calibration is done to set IDELAYS and ODELAYS on the FPGA to account for physical wire or trace lengths between FPGA and QDR chips. Uncalibrated QDR chips will produce data errors. Tue QDR chips are RAMs that store wave form data that is sourced the the DACs.

resView.py

This module is a program for opening and navigating hdf5 files with resonator sweep and noise data. It was written for the MKID research at APS, and not used anymore. Hdf5 files can be navigated like a linux file system. Resonator data can be analyzed and plotted and viewed.

To use the module:

>>Execfile('resView.py')

>>hdfshell()

Now you type text commands."man" or "help" gives the help screen.

Resonator.py

This module has 2 classes:

1) MKID-
    a. A class representing a single resonator.
    b. Has list of resonatorData objects in reslist[]
2) ResonatorData
    a. A class representing data on a resonator, a single frequency sweep, data from running fits, and 0 to N noise traces.
    b. plotFreq() is a useful function on a resonatorData object to see the resonator and see data.

There are global functions and variables:

MKID_list=[]

Global variable, a list of MKID objects. This is the list of resonators in the cryostat.

Resonator_freqs=[]

Global list of resonator center frequencies.

Several global functions for loading the AMKID data to and from disk.

See the py for docs.

Important functions:

pyListToMkidList() : LOADS MKID_list or [MKID,….] to a python text file containing only the center frequencies.

mlist2Pylist()

Save MKID_list to text py file.

For MKID R/D we used hdf5 to save resonator data. For TES we just save the resonator frequencies with above functions.For saving hdf5 data, full power sweep data for resonators:

Save MKID data:

def mkidSaveData(filename):

Load MKID data

def mkidLoadData(filename):

roachFFT.py

Programs the channelizer state machine to perform the following:

1) Determine which FFT coefficients are saved to channel FIFOS.
2) Map FFT bin number to channel number, FIFO number, bin frequency, source frequency.
3) Program the above maps into the channelizer state machine RAM table so ROACH saves correct FFT bins into correct FIFOS

4) Starts/Stops FFTs in sync with waveform sourcing so FFT analysis has phase reference to wave form source.
5) Sets number of FFTs to compute, for 1 FFT, then stop, as in taking a frequency sweep, to taking FFTs forever, as in streaming noise or pulse data.
6) Picks which FFT bins to capture based on waveform source frequencies.

roachMatlab.py

Used only on ROACH1. It replaces katcpNc, or the interface from py codes to the ROACH katcp server, and replace it with an interface to matlab. One can run the py scripts as if controlling the roach. A file will be written so the MATLAB firmware Simulink simulator can simulate the FPGA operation given the setup from the py firmware. Was not updated for Roach2.

roachScope.py

the roach can store data to RAM to be read back by linux python. The idea is to have oscilloscopes in the ROACH FW to checkl internal operation of FW in real time without using Xilinx tools. To see raw ADC captured data:

```
fa.adcscope = roachScope(roach, 'octoscope')

fa.adcscopetrig()
fa.adcplot()
Freq plot
fa.adcplot(IQ='IQF')
yscale('log')
Time domain I or Q
fa.adcplot(IQ='I')

fa.adcplot(IQ='Q')
```

One can see raw FFT coefficnets as well or other roach data. See FW.

```
fa.fftscope = roachScope(self.roach2, 'octoscope1')
fa.fftscopetrig()
fa.fftplot('Re')
fa.fftscopetrig()
fa.fftplot('M')
fa.fftplot('Re')
```

sramLut.py

Program the QDR SRAM or statuc ram wave table for sourcing thru DAC. setLutFreqs([freqs in Hz], [amps in counts]) sets the frequency table. Also setLutSize(int) sets the length of the LUT.

One can add pulses or phase modulation into the waveform for debugging, to pretend that there is a crupstat present:

These class variabes can be chaned in py shell

```
#
    # for add test pulses to waveform
    #
    #len in samples
    self.test_pulse_len=256
    #in degrees
    self.test_pulse_amp =20.0

    #true of false

    self.is_test_pulse = False

    #
    # FM modulation of the freq tones.
    #

    #do we use FM?
    self.is_mod_freq = False
    # amplitude of phase modulation in radians
    self.mod_amp=0.1
    #period of modultion in samples.
    self.mod_period=12000.0
    #use a integration time of 190 samples, 4.0 periods
```

fftAnalzer R2

fftAnalyzerR2 is a high level python file to control the ROACH. If you wish to control ROACH with py scripts, use this file. All high level functions are in it, or are in contained objects. Also, data from ROACH gets stored in this object. The global object in the python space is fa. So in python, fa, is a reference to the fftAnayser object, representing the ROACH board high level software interface. Lots of example code is at top of the file in the commetns.

Sweep Data

The data from a sweep is stored in fa.iq_data, as a list [array_I, array_Q]. The sweep frequencies are in fa.freqs_sweep, as baseband freqs. The LO freq is fa.carrier_freq.

Raw Roach data, noise data.

The raw RAOCH data is in iq_data_raw. To get raw data:

```
#set LO to 5GHz
fa.setCarrier(5e9)
# set BB freqs, where RF freqs are LO – BB freqs, for 4990MHz, 4980MHz. DAC amplituds in counts are 10000. Max amp is 32768.
fa.sourceCapture(  [10e6,20e6],10000)
#after a time, stop capture
fa.stopCapture()
#get iq data.
mydata = fa.getIQ()
#fa.iqdata_raw is same as mydata.
```

```
fa.iqdata_raw
Out[22]:
{192: {'bin': array([ 502.,  502.,  502., ...,  502.,  502.,  502.], dtype=float32),
  'event_len': array([ 100.,  100.,  100., ...,  100.,  100.,  100.], dtype=float32),
  'event_type': array([ 1.,  1.,  1., ...,  1.,  1.,  1.], dtype=float32),
  'flux_ramp_phase': array([-1., -1., -1., ..., -1., -1., -1.], dtype=float32),
  'flux_ramp_phase_unwrap': array([-1., -1., -1., ..., -1., -1., -1.], dtype=float32),
  'is_pulse': array([ 0.,  0.,  0., ...,  0.,  0.,  0.], dtype=float32),
  'phase_delay': array([ 0.], dtype=float32),
  'stream_mag': array([ 0.00067139,  0.0007019 ,  0.00061035, ...,  0.00018311,
         0.00019836,  0.00018311], dtype=float32),
  'stream_phase': array([ 0.20248546,  0.21782526,  0.12540293, ..., -0.08283496,
        -0.29644176, -0.24160197], dtype=float32),
  'timestamp': array([ 2.,  0.,  0., ...,  0.,  0.,  0.], dtype=float32)}}
```

The raw data is in stream_mag and stream_phase. FRD data is in 'flux_ramp_phase_unwrap' if FRD is running.

To get useful data, including metadata, ROACH settings,

aa=fa.getUsefulData()

Get only metadata.

aa=fa.getUsefulMetaData()

List all roach registers and valies

```
roach.readAllReg()
```

To do a sweep we set center and span frequencies in Hz. We set number of sweep points, and dBm power at resonator in cryostat. This is not power out of roach, but accounts for calibration data all the way to the resonator.

```
fa.sweep(span_Hz = 300e6, center_Hz=5000e6,pts = 2048, pwr_at_res = -85)
```

To convert iqdata to mag/polar and plot magnitude.

```
ipq = fit.Rect2Polar(fa.iqdata)
plot(fa.freqs_sweep,iqp[0])
```

To capture noise for some period of time:

```
fa.captureNoise(rffreqs=[5000e6,5010e6],timesec = 10.0,pwr_at_res=-85)
clf()
plot(fa.iqdata_raw['stream_phase'])
plot(fa.iqdata_raw[192]['stream_phase'])
plot(fa.iqdata_raw[192]['stream_phase'])
```

You can also capture to a file:

```
fa.captureNoise(rffreqs=[5000e6,5010e6],timesec = 10.0,pwr_at_res=-85,fname='/path/myfile.h5')
```

Links to data receiver program and anritsu box.

The objects fa.an and fa.capture are the interfaces to the Anritsu and data captureing program. To restart these links, if you hit Ctrl-C (for hitting Ctrl-C kills both programs!!!):

```
fa.an.shut()
fa.an = anritsu()
fa.capture.shut()
fa.capture=dataCapture()
```

to setup 10GB Ethernet.

```
fa.setupEthernet()
```

To set Ramp generator and TES voltage source:

```
fa.setRampGenerator( is_on, freq, voltage):

fa. setTESBias(is_on,vbias):
```

To setup ROACH fo IQ circle translation:

The roach must be programmed so it can translate IQ circles to the origin. A series of sweeps is done. The user sets the desired RF freqs, then roach does the sweeps and sets up the translation in the FW.

```
Fa.sweepProgTranslators(
    rffreqs=[5100e6, 5110e6],
    pwr_at_res = -85)
```

Voltage Sweep of TES

To perform a voltage sweep with ROACH, put list of voltages desired as an list.

```
fa.voltSweep(
    vlist=array(10.0,0.0,-0.01).tolist(),
    rffreq=[5100e6],
    fnames='mydata.h5')
```

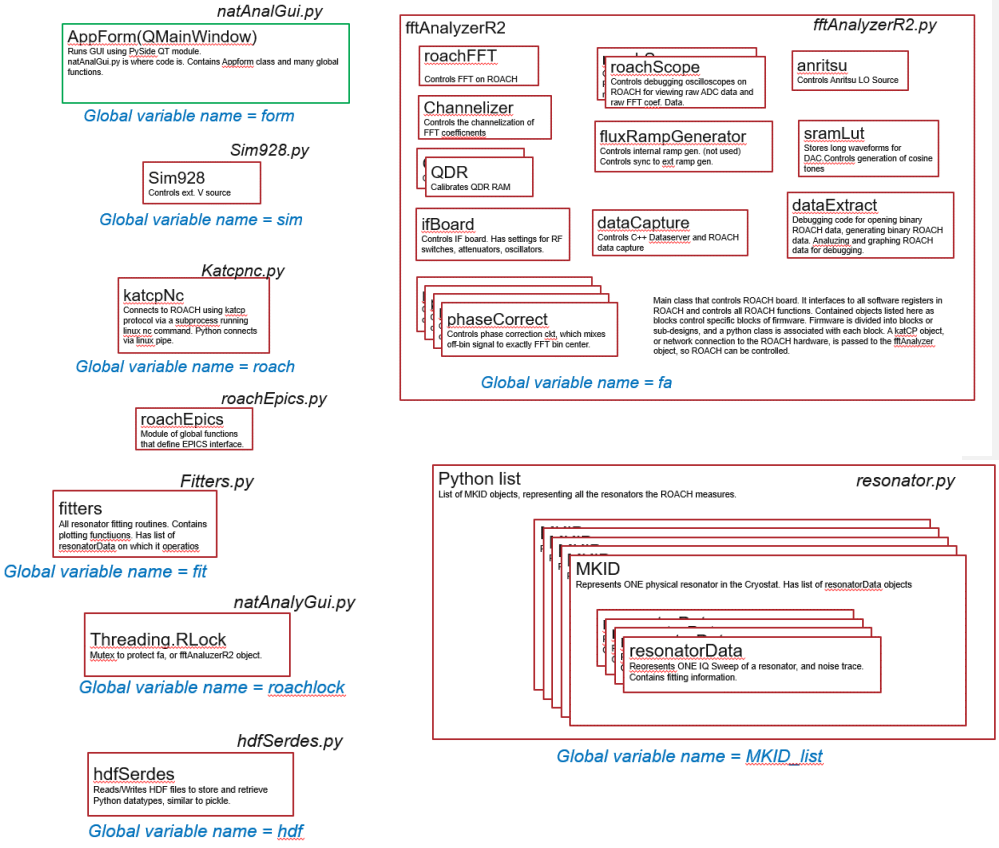
natanaluzergui.py

natAnalyzerGui.py is the highest level py module in the roach software. It loads all other modules on execution, and keeps references to the fftanalyzer object, fa. Also fa is called na, so na = fa in python code. This module implements the GUI using PyQT. Major parts of the file are:

1) Global functions that run various ROACH measurements on a separate thread. All called def runXXXXX(). Long roach operations operate on separate thread, so GUI thread and/or py shell is usable during ROACH operation.
2) Global functions setupEverything, shutdownEverything to turn on/off the roach and set it up.
3) Execfile() of all needed py modules for ROACH.
4) class AppForm(QMainWindow): object, called form as a global variable. This is the GUI.
5) Ability to exit the GUI QT thread from a button, kills App.exec(). Makes Py shell available. Def gui() reenters the QT loop.
6) Form.message_timer=QTimer(). Timer that truggers def form.readMessageQueue() every 500ms. This is the GUI monitor, so gui can update when roach does something.

7) A Queue called gloval message_queue=Queue.Queue() passes messages between ROACH runtime thread and the GUI. The def runXXXX rfunctions put py dicts on the Queue, and form. readMessageQueue() takles the dict off the Quee and makes GUI do something.
8) A mutex called roachlock, which assures only one thread accesses the roach at once is defined on this file.
9) A set of signals/slots, so when Gui button his hit, a slot in class AppForm(QMainWindow) is executed. This slot grabs roachlock if it needs to access roach Sw interface. Often te slot will call thread.start_new_thread(runXXXX,()) so one of the global def runXXXX funtions runs the roach.
10) Has a main() function to start GUI. Has a mainEpics() method to start ROACH using an EPICS V3 interface. This is not well tested.
11) Has many plot functions to plot on the QT GUI.
12) Def create_main_frame() is the function that draws the GUI. And all widgets. All gui is programmatically created, and QTDesigner is not used.
13) Defines a sim928 object so we can talk to the voltage generator for TES bias.
14) Defines callbacks to GUI can respond to when RIACH does something.
15) Defines the IP address of ROACH of 192.168.0.70 in global temp_ip.
16) has is_epics_running global bool, so epics can be used.
17) Interacts with two message Queues for epics, defined in roachEpics.py, if epics is running. Messages are py dicts with string keys sent and received between epics thread in python, and the roach readmessageQueue thread, triggered by form.message_timer.

Python Software Objects


natAnalGui.py
AppForm(QMainWindow)
Runs GUI using PySide QT module.
natAnalGui.py is where code is. Contains Appform class and many global functions.
Global variable name = form
Sim928.py
Sim928
Controls ext. V source
Global variable name = sim
Katcpnc.py
katcpNc
Connects to ROACH using katcp protocol via a subprocess running linux nc command. Python connects via linux pipe.
Global variable name = roach
roachEpics.py
roachEpics
Module of global functions that define EPICS interface.
Fitters.py
fitters
All resonator fitting routines. Contains plotting functiuons. Has list of resonatorData on which it operatios
Global variable name = fit
natAnalyGui.py
Threading.RLock
Mutex to protect fa, or fftAnaluzerR2 object.
Global variable name = roachlock
hdfSerdes.py
hdfSerdes
Reads/Writes HDF files to store and retrieve Python datatypes, similar to pickle.
Global variable name = hdf
fftAnalyzerR2
fftAnalyzerR2.py
roachFFT
Controls FFT on ROACH
Channelizer
Controls the channelization of FFT coefficnents
QDR
Calibrates QDR RAM
ifBoard
Controls IF board. Has settings for RF switches, attenuators, oscillators.
roachScope
Controls debugging oscilloscopes on ROACH for viewing raw ADC data and raw FFT coef. Data.
fluxRampGenerator
Controls internal ramp gen. (not used)
Controls sync to ext ramp gen.
dataCapture
Controls C++ Dataserver and ROACH data capture
anritsu
Controls Anritsu LO Source
sramLut
Stores long waveforms for DAC.Controls generation of cosine tones
dataExtract
Debugging code for opening binary ROACH data, generating binary ROACH data. Analuzing and graphing ROACH data for debugging.
phaseCorrect
Controls phase correction ckt, which mixes off-bin signal to exactly FFT bin center.
Main class that controls ROACH board. It interfaces to all software registers in ROACH and controls all ROACH functions. Contained objects listed here as blocks control specific blocks of firmware. Firmware is divided into blocks or sub-designs, and a python class is associated with each block. A katCP object, or network connection to the ROACH hardware, is passed to the fftAnalyzer object, so ROACH can be controlled.
Global variable name = fa
Python list
List of MKID objects, representing all the resonators the ROACH measures.
resonator.py
MKID
Represents ONE physical resonator in the Cryostat. Has list of resonatorData objects
resonatorData
Reqresents ONE IQ Sweep of a resonator, and noise trace. Contains fitting information.
Global variable name = MKID_list

Running ROACH on cook.xray

Network Setup

There are two 1GB Ethernet and one 10GB Ethernet connections required:

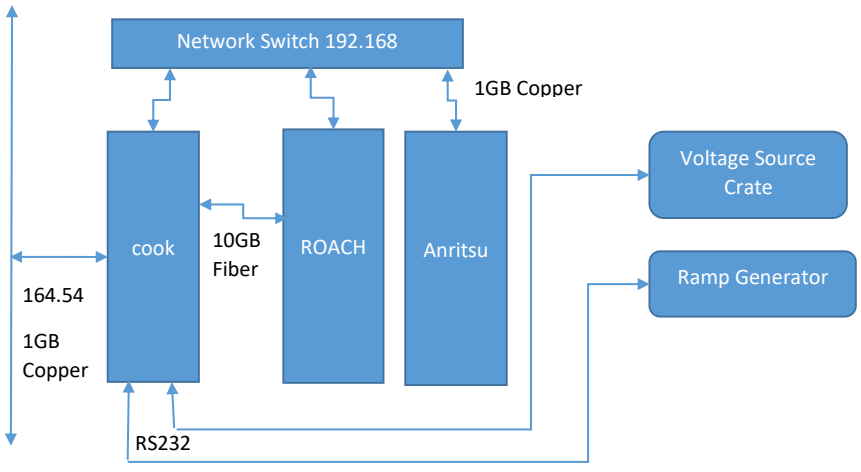
1) The 164.54 network at 1GB copper, should be device eno1, 164.54.101.177
2) A local network 192.168.0 to a network switch. Anritsu and ROACH board connect.
3) 10GB Ethernet fiber directly to ROACH SFP connector

Need pics

Need proper ip address

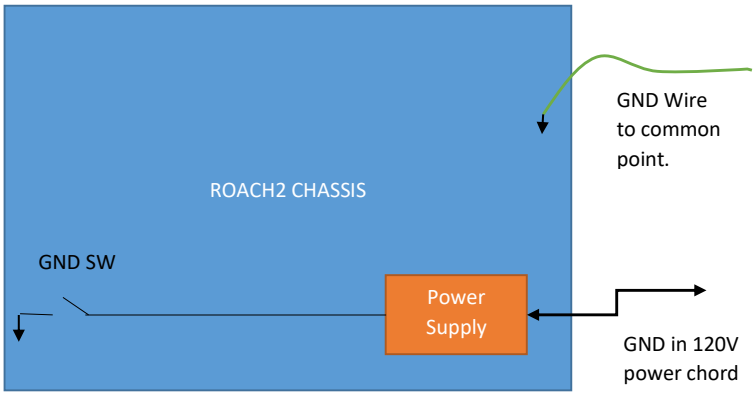
There are two serial RS232 connections:

1) On COM port on back of cook, connect null modem cable from cook to the rack where voltage source is plugged in.
2) Using USB com adapbter, connect with null modem to the Ramp generator.



Power Supply and Ground

The ROACH2 board is powered by a switching power supply that has its own ground from its three conductor power cord. This power supply has been altered (by Jon Baldwin at APS) to disconnect the power supply ground from the DC ground at the power supply output. A switch can connect and disconnect the ROACH ground to the Power supply ground. A second ground wire is connected to the ROACH chassis to allow all hardware in the TES system to have a common ground point. Usually the ROACH ground switch is open, disconnected power supply ground from ROACH electronics, and the ground wire is connected to common ground.



Preferred operation is to open GND SW, connect GND wire to common. If no wire to common, then SW must be closed or IF board could be damaged. If SW is closed and wire to common is present, noise will be increased due to potential ground loop.

If there is no common ground wire and GND SW is open, then the ROACH chassis floats. Because the chassis is powered by a switching power supply, noise from the power supply causes a 300kHz 20V signal to appear on the chassis. Grounding the chassis shunts this signal to ground. *The 20V signal can damage electronics on the IF Board if the IF board is connected to external grounded equipment, because the IF board electronics effectively becomes the ground path. Therefore it is essential that the ROACH chassis be grounded.*

RF Connections

Software Setup

In your root directory set up the following:

1) In .tcshrc add *setenv ROACH /localc/roach/RoachFirmPy*
2) Save this script as runipython
   cd /localc/roach/RoachFirmPy/Roach2DevelopmentTree/pyfiles
   /APSshare/anaconda/x86_64/bin/ipython –pylab
3) Save this script as runhdfview
   #!/bin/sh

   export LD_LIBRARY_PATH=/home/beams0/TMADDEN/swWork/hdf5/hdf5-1.8.12-linux-x86_64-shared/lib
   export LD_LIBRARY_PATH={$LD_LIBRARY_PATH}:/home/beams0/TMADDEN/swWork/boost/boost155Install/lib
   export HDF5_PLUGIN_PATH=/home/beams0/TMADDEN/swWork/hdf5/HDF5Plugin-master

   /home/beams0/TMADDEN/swWork/hdf5/hdf-java-2.10/HDF-JAVA-2.10.0-Linux/usr/bin/hdfview.sh
4) You may need permission to use the serial ports. As rootsh (*sudo rootsh*)

   chmod 777 /dev/ttyS0

   chmod 777 /dev/ttyUSB0

5) You may need to log onto cook at its terminal to enable network connections. At top right there are pull down menus to enable these. It is already setup, but you may need to step through menus on cook linux GUI to set up network connections. Meny on top right setting button.
6)

Bugs, Problems, and how to deal with them

Killing processes when Roach crashes

List your processes to get a list with ps –u. The processes in yellow should be killed.

See below:

```
tmadden@cook pyfiles]$ ps -u
Warning: bad syntax, perhaps a bogus '-'? See /usr/share/doc/procps-3.2.8/FAQ
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
tmadden   20798  0.0  0.0 112668   316 pts/0    Ss   Aug11   0:00 -tcsh
tmadden   20861  0.0  0.0 122728   368 pts/0    S+   Aug11   0:01 screen
tmadden   20863  0.0  0.0 112844  1504 pts/1    Ss   Aug11   0:00 -bin/tcsh
tmadden   20907  0.0  0.0 112488  1128 pts/3    Ss+  Aug11   0:00 -bin/tcsh
tmadden   21365  0.0  0.0 108172  1328 pts/4    S+   13:08   0:00 /bin/bash startsgCook
tmadden   21380  2.5  5.8 6474128 942152 pts/4  Sl+  13:08   3:29 /localc/MATLAB/R2012b/bin/glnxa64/MATLAB
tmadden   21497  0.0  0.0  15044  1108 pts/2    Ss+  13:08   0:00 /localc/MATLAB/R2012b/bin/glnxa64/matlab_helper /dev/pts/2 yes
tmadden   21508  0.0  0.0 108168  1288 pts/2    S    13:08   0:00 /bin/sh /localc/Xilinx/14.6/ISE_DS/ISE/sysgen/bin/lin64/sysgensockgui 3141 0
tmadden   21513  0.0  0.0 238272 12008 pts/2    S    13:08   0:00 /localc/Xilinx/14.6/ISE_DS/ISE/sysgen/bin/lin64/sysgensockgui.bin 3141 0
tmadden   21521  0.0  0.0 108168  1300 pts/4    S+   13:08   0:00 /bin/sh /localc/Xilinx/14.6/ISE_DS/ISE/sysgen/bin/lin64/TclProxyServer
SharedMemory sgproxy21380_599b21a7
tmadden   21526  0.0  0.5 400104 83560 pts/4    Sl+  13:08   0:05 /localc/Xilinx/14.6/ISE_DS/ISE/sysgen/bin/lin64/TclProxyServer.bin
SharedMemory sgproxy21380_599b21a7
tmadden   21633  0.0  0.1 6185788 22440 pts/4   Sl+  13:11   0:03 /localc/MATLAB/R2012b/sys/java/jre/glnxa64/jre/bin/java -cp
/localc/Xilinx/14.6/ISE_DS/ISE/sysgen/bin/sysgen.jar:/localc/Xilinx/1
tmadden   21897  0.0  0.0 112820  1116 pts/4    Ss   Aug11   0:00 -bin/tcsh
tmadden   22418  1.0  0.0 109968  1100 pts/1    R+   15:25   0:00 ps -u
tmadden  136228  0.0  0.0 102972   572 pts/1    S    Aug17   0:00 sleep 9999999
tmadden  136230  0.0  0.0 102972   548 pts/1    S    Aug17   0:00 sleep 9999999
tmadden  136232  0.0  0.0  14048   480 pts/1    S    Aug17   0:04 nc 192.168.0.70 7147
tmadden  136330  1.1 59.0 11146368 9574428 pts/1 Dl  Aug17  70:18
/home/beams/TMADDEN/ROACH2/RoachFirmPy/Roach2DevelopmentTree/QT/build-testEnet-Desktop-Debug/testEnet
tmadden  136411  0.0  0.0  16052  1276 pts/1    S    Aug17   0:00 /home/beams/TMADDEN/ROACH2/RoachFirmPy/vx11/vxi11_1.10/anritsuOsc
192.168.0.68
tmadden  136644  0.0  0.0 112500  1720 pts/6    Ss   Aug17   0:00 -bin/tcsh
tmadden  136829  0.0  0.0 140904  6336 pts/6    S+   Aug17   0:08 vim katcpNc.py
[tmadden@cook pyfiles]$ kill 136228 136230 136232 136330 136411
[tmadden@cook pyfiles]$
```

Anritsu Hangup Problem

Sometimes the Anritsu crashes and needs restarting. You will see a window on the anrotsu box telling of error. Turn off and on. We have had problems w/ resetting anrotsu putting too much power into th IF board. Perhaps remove the Rf cable when restarting anritsu.

Logging onto roach when someone else has it.

If two users tru to connect to ROACH the software will crash. Only one anritsu process can run. Also only one process can get access to the UDP port to capture ROACH data. The roach itself can acceot many connections at once however.

Update Roach SW and the temp directory

Roach will save files to a temp directory when taking data. When we update the software from git,

Cd /localc/roach/RoachFirmPy
git pull https://github.com/argonnexraydetector/RoachFirmPy.git

To install software we do:
Cd /localc/roach
git clone https://github.com/argonnexraydetector/RoachFirmPy.git

Now we must add the temp dir
Cd RoachFirmPy
Mkdir temp

Now we must tive write access
Chmod –R 777 RoachFirmPy
This allows everyone to write/read the roach folders…

Calibration of QDR

Sometimes QDR cal fails. Do it again from shell like below:

fa.calQDR()
Please WAIT- calibrating QDRs
Calibrating QDR
QDR calibration failed.
Calibrating QDR

Roachlock

If the GUI is unresponsive,
There is a mutex in py code to get access to riach interface. Only 1 thread con control roach.
Sometimes it gets locked by main thread.
Type
roachlock
do roachlock.release()

Recent bug

The noise is finishing up in py before the parsing is done. Why?
Parsing goes on after packet q len is 0. Why?

Eveb with 1 resonator, frd onlym, it parses forever, after nothing on the packet queue.

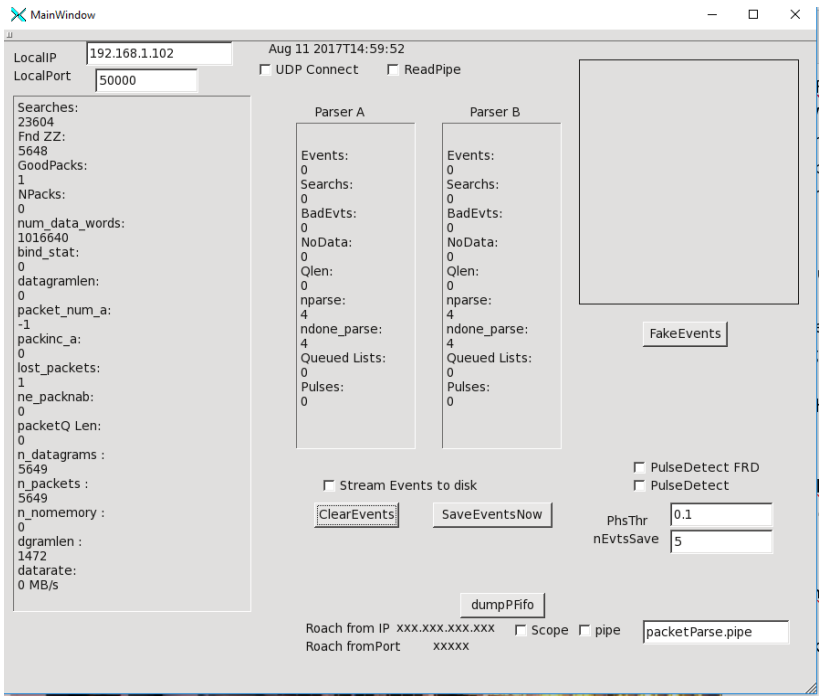
ShutRoach bug

If some software is off, then shut roach will error, and not shut anything.

Debugging the QT program

Make fake events to test parsing.

Hit fake events.
You will see events eventyually apper in ParserB
You then CkearEvents or SaveEventsNOW



Bug: hitting clear Events during data capture will crash QT program.

To stop capture without stopping roach.

Cliock UDP connect, go get check. If it was alrday taking data, it will not make new socket, but just put on the checkl. Python makes the QT connect to socket in background, to capture data, but check wont chow up. Hit the UDP connect again to erase check. Now the sock gets deleted, and QT will not capture data. You can use nc to capture data to disk from roach now.

Debugging data Stream

Duymp raw streem to disk. Put in a filename with path in packetParse.pipe window
Check pipe
Now make rriach take data in usual way. A copy of RAW datastream is saved to disk, without loss, because there is a Q to hold in mem if disk is slow.

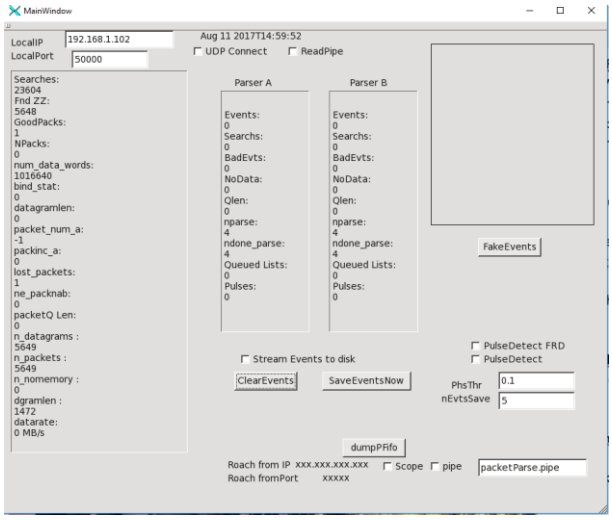
Now you can execute the testEnet program, qt program

Hith UDP connecto to 192.168.1.102,port 50000. It will listen there.
Now you can send the saved stream to that udp port:
nc -u 192.168.1.102 50000 < rawdatastream.bin

Now you can see of the software works.

testEnet buttons:
Scope- a cheap scope not work well. Has bugs
Readpipe- not work. The idea was to read data from lnux pipe. Not work. We just read from udp socket.
Pulsedetect- find pulses and saveonly pilses in raw phase. Set phase threshold above noise, set *nEvetnsSave*, for length of data after pulse trigger. Set to 10 or so.
PulseDetectFRD=- find pulses in frd signal. Preferred operation for pulse mearysmrnet. th
FakeEvents- put fake evetns into parser for debugging the SW. if not roach available.
StreamEvents to disk- in real time sve data to hdf5 as we stream. Python does this by remote control.
ClearEvents- cleawr parser data.
Save EventsNow- save parser data to dosk, of not streaming to dosk
dump P fifo- dum,ps udp packet fifo to clear.

Panels on the testEnet window



See panels 1,2,3,4 from left to right on figure. 4 has nothing it. It is not used.
Panel 1 counts raw udp packets.
The udp packets have a packet number embedded in them, so we can count if there are lost packets.
See Searches: it will search on 1st st of data after starting roach. But never should sercah agbain. Searches should stay a constant.
Lost packets should be a constant, should not be increasing. It counts if the packet number is skipped. Npackets and
PacketQ len will go up, then down. We put raw udp pacets ona Q.

If we see searches uncreasing, and or lost packets increasing, we are using data on the 10GB link.

Panel 2, and 3 are for each parser that runs in tandem. Parser on panel 2 gets no data so nothing happens. Parser in panel 3 gets all data.

If you see searches we are losig data. It looks for 0x5555's in the ata. If we are not losing udp ackets but searching in the panel 3, then we are overflowing fifos on roach.

Dgramlen is 1472.

```
# run flux ramp demod, return raw fft coef. 200MB/sec
fa.chanzer.setFluxRampDemod(
   is_demod=1,
   is_incl_raw_trans=1,
   evt_len=100,
   num_cycles=2)
#source 12 resonators
fa.sourceCapture([10e6,20e6,30e6,40e6,50e6,60e6,70e6,80e6,90e6,100e6,110e6,111e6],300)
#now you watch the QT C++ program gui and look for problems
#after a time you stop capture
fa.stopCapture()
#QT gui will still be parsing for awhil until packet Q isempay
#Now clear full counters- they inc when fifios on ROACH fill up and lose data
fa.chanzer.clearFull()
#print the full counters
fa.chanzer.checkFull()
#take 12 chanes
fa.sourceCapture([10e6,20e6,30e6,40e6,50e6,60e6,70e6,80e6,90e6,100e6,110e6,111e6],300)
#now we chek fir fifos filling several times.
fa.chanzer.checkFull()
fa.chanzer.checkFull()
fa.chanzer.checkFull()
#stop capture
fa.stopCapture()
#read all sw regs on roach
```

```
roach.readAllReg()

# run flux ramp demod, return no raw data.much slower data rate. <10MB/sec
fa.chanzer.setFluxRampDemod(
   is_demod=1,
   is_incl_raw_trans=0,
   evt_len=100,
   num_cycles=2)
```

Get a fresh QT data receiver

fa.capture.shut()
fa.capture = dataCapture()

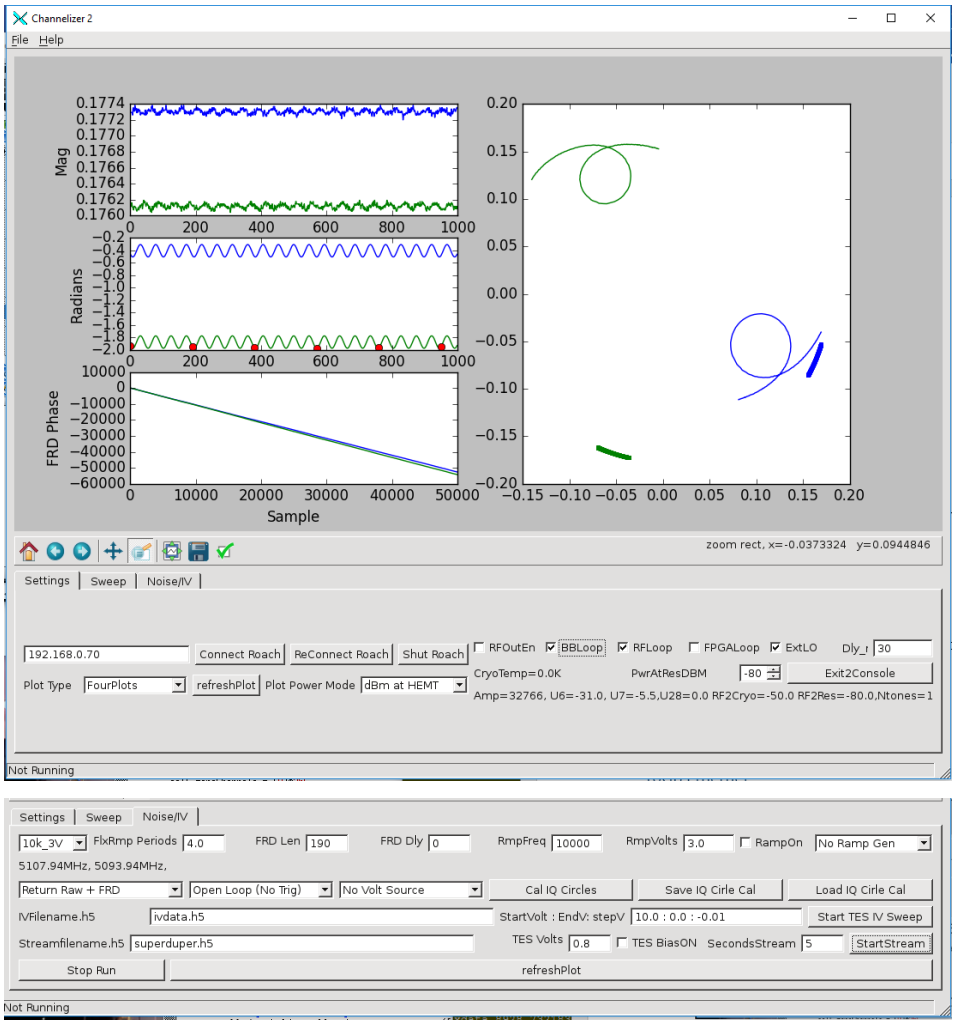
Get a fresh connection to Anritsu

fa.an.shut()
fa.an=anritsu()

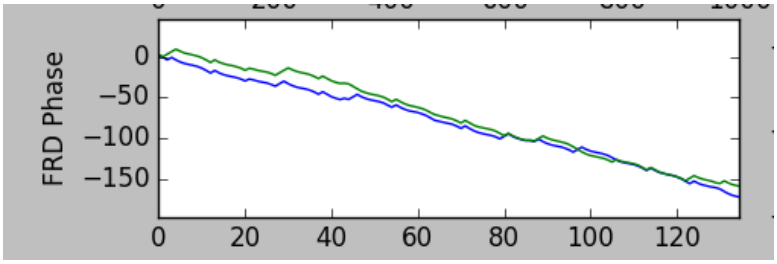
Testing flux ramop demod w/ no crystat

In python

1) In roach gui, load in any resonator cal, so you can take nose
2) Use baseband loopback or rf loopback.
3) In python shell type this following to make the waveform source contain phase modulation like we have a SQUID and resonator.
    a. fa.sram.is_mod_freq=1
    b. fa.sram.mod_amp=0.1
    c. fa.sram.mod_period=12000
    d. fa.sram.setLutSize(1024*1024)
4) Set fluxramp periods to 4.0, FRD Len to 190, FRDDaly to 0, use Return Raw Data, FRD. Openloop mode.
5) Now take nosie for 5 sec, return FRD and raw data, NON sync mode but open loop.



Settings for simulating a SQUID when no cryostat is available.



Closeup of FRD phase

Closing UDP socket on the data Receiver, testEnet

The data receiver runs a thread that waits for a packet on the UDP socket, then puts the packet onto a Queue when one comes in. If we wish to close the socket, this loop is still running, and blocking on waiting for a packet, so the socket cant close. So to close the socket we use Gui, or remote control from python over text socket to disconnect. ON gui this means you check UDP connect, then unc check. The socket will close upon receiving the next udp packet. So closing the socket: 1) tell program to close socket 2) send a udp packet to unblock the thread. Then socket closes.

Problem with Xilinx tools in Cook

The Xilinx tools ISE 14.6 should be reinstalled in cook. The simulator does not work since cook linux was updated to el7, than back to el6. It should be reinstalled.

Sync mode problem

If we are in sync mode, meaning we sync the roach data with the voltage generator we NEED to throw away some of the data because of the way the fifoPhFSM3.m state machine works. It should be fixed so we can be in sync mode without throwing away data. Perhaps sync to 1st pulse, then run unsync'ed for ever. This will work if the ramp generator and roach are on same 10MHz clock, which they should be anyway.

Short Noise Problem

If you take too short a noise trace, the FIFOS in the ROACH and Queues in softwarwe will not flush enough data to start writing the HDF file. Data will be stuck in the Queues and never saved. For FRD data only, take at least 10s of data. For FRD+raw coefficient data, take at least 5sec of data. The problem manifests itself when the plot never appears, and the py shell shows a message along the lines of

“cannot open file. “ The py shell opens the hdf5 file from disk for plotting. To fix this, add code to flush Queues in the testEnet program. It is mostly a software problem.

Sim928 problems

Permission problems can happen with new users to the ROACH. The user must be a member of the serial port group. Troy knows how to do this. The other problem is that the sim928 is in a sim900 rack. When connecting to the sim928, you may actually connect to the rack itself. If you see sim900 appear on the GUI at bottom when opening the sim928, then you have the problem. To fix:

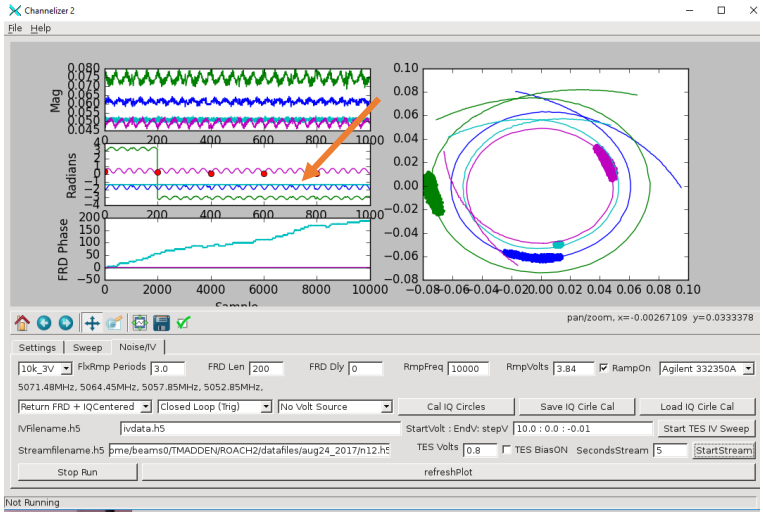
1) Exit to py shell so you can type.
2) Type sim.connport(8)

Now you should be able to use the sim928.

Missing Data problem

If we are in trigger mode, and tryu to save 200 samples, we can run out of data in fifos if several channels?

Missing data here.



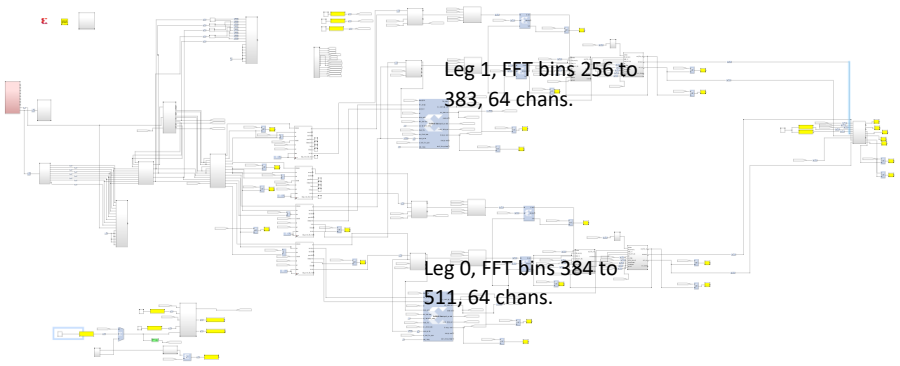
To fix, grap less than 200 samples. We have a problem that we have too short a fifo. And have trouple meeting timing for longer fifos. Problem…

The root of the problem is this:

1) Channel fifos (FIFO64) are written at 1MHz per FFT. For 2 FFTs we write two FIFO64s at once, at 1MHz each.
2) For 50 channels, we write each FIFO64 at 50MHz on average. Because clock is at 128MHz we are fine. For 110 channels, we write at 110MHz.
3) We read each FIFO64 (one per FFT) at close to 128MHz, say around 110MHz, because there are wasted cucles in added header data. So each Single event fifo is written at 110MHz or so. This means we can handle up to 110 channels.
4) Each Single Evnet fifo is read out at 50MHz, minus wasted cycles for the FSM. The reason for the slow down, is that we muxtwo SEFifos into one combined event FIFO, to go from two 1MHz streams to one 2MHz stream. This means we can only deal with

betwwen 45 and 50 channels! This is a major bug, and was introduced by adding the 2nd FFT, after the ASC paper was published. This needs to be fixed. The single FFT FW can do over 110 channels without problems.

There is a branch of the git called twolegs. It doubles the through put by adding a 2nd FRD circuit. Python needs to be updated to make it work as of now. The FW compiles OK. Needs testing. It should allow around 100 channels. It uses the whole 10GB enet link. Both halves of the 64 bit word. The Fw is tesdd.slx



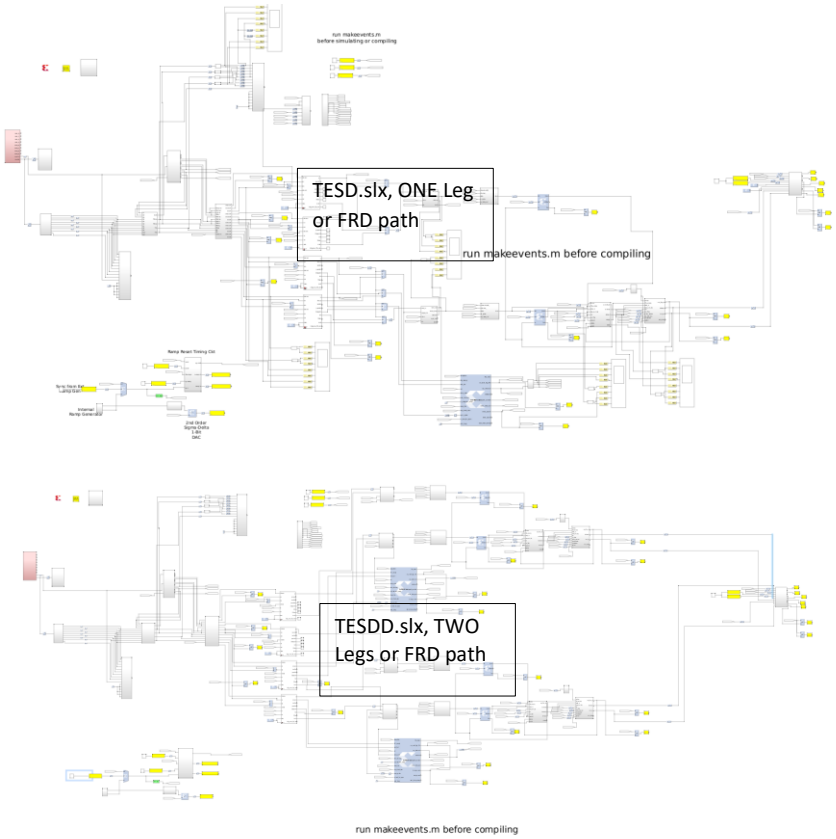
run makeevents.m before compiling

Future work

Double up FRD data path

To make the system run more channels, a 2nd leg must be added. The firmware for this is tesdd.slx, in the github branch twolegs. This is not tested as of Aug 2017. Py scripts need to be updated for this. Below is a pic of tesd.slx, and tesdd.slx, for comparison. Tesdd.slx uses all 64 bits of the 10GB enet, while tesd.slx uses only 32. IN tesdd, both parsers in testEnet run and two hdf files are saved, that must be added together. There is software work needed to do this,

in the py scripts. Also work is needed in py scripts for controlling both legs. Start in fftanalyzerR2.py and work your way down. Make sure all SW registers are programmed in 2nd leg.



Speed up State machines for reading FIFOs

The state machines that read out data fifos can be improved by eliminating states to make them run faster, allowing more channels. Also the header data should not be inserted into the daa stream until the end of the data stream because it adds clock cycles that should be used for

reading out data from the FIFOs. Header data can be inserted into a separate FIFO in parallel with the data, and combined at the end.

Improve FIFO64

The current FIFO 64 has 64 counters for both reading and writing, for a total of 128 counters. This is excessive, especially when it is known that we read the FIFO in channel order, all data rates are the same for all channels, and channels are written in a known order with known rate. Only four counters are needed. This should be redesigned.

A version of the FIFO exists using BRAMs to store the counters, which should be efficient. This almost works, but not fully debuygged. This design exists in the github in the branch newfifo. See tesd.slx in that branch, and multififo.slx for simuilation.

Increase Precision of FRD output, and FFT outputs

The FFTs output data at 18bits, for real and im. Coef. Before FRD it is truncated to 16 bits. This should be fixed. Also the FRD outputs a result of 16 bits. This is easy to increase to 24 bits.

Preliminary design exists in Tesdprec.slx, for FFT output precision.
It is easy o improve the FRD output precision. All you need do is alter the DFT block in the FRD block to not truncate to 16 bits. This is easy. Then the QT C++ program should be updated. This has been tried, but not debugged.

Port to Virtex 7

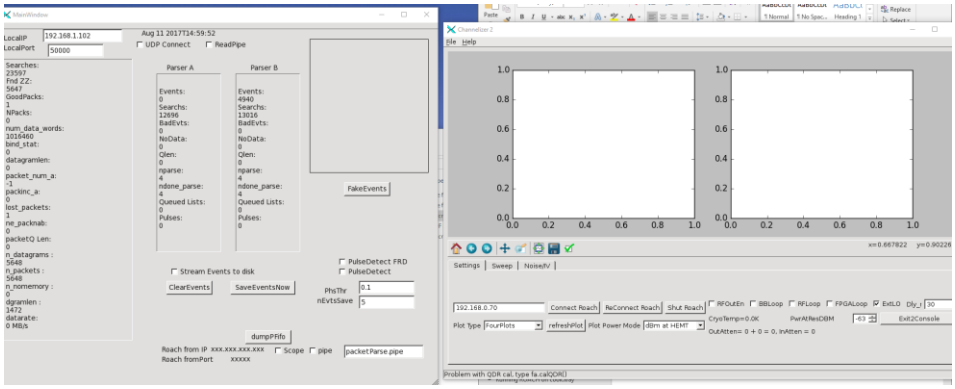
To port the firmware to the FNAL board, or some 4DSP board follow these steps.

1) Redraw portions of the design not using CASPER green blocks. This means using Xilinx FFTs for the FFT, and designing your own polyphase filter. Also the yellow blocks have to go, and be redesigned.
2) For software registers, design a demux so address, data, and wr/rd bits can convert an address/data to a set of software registers. The address/data will then run on AXI. The clock for these registers should match the data sample rate clock.
3) Set up vivado to run with Matlab. This is done by putting a matlab exe in the PATH var. rRun sysgen in the Vivado binary directory. Sysgen will find the matlab and run it. It will be Vivado aware. Use Vivado 2015 and matlab 2015, it works.
4) There is a file, tesv.slx in the ROACH2 dir. It is a design of polyphaser filter with FFT. It compiles as an IP block.

5) You run system generator in matlab/sysgen to make an IP block.
6) You then open a Vivado project, and add this iP block to the IP library or path.
7) IN vovado you put the new IP in the block diagram and compuile w/ vivado as usual.
8) For Vivado, you will need to implement 10GB Ethernet. Use the FASPAX design. For software registers, FASPAX has a AXI set of SW registers for a Zynq chip. The FNAL board is not Zynq, so you need a solution for SW registers from the soft core processor.

Software User Manual

1) Cd to your home directory
2) ./runipython
3) In python: execfile('natAnalGui.py')
4) main()
5) The GUI should open
6) Hit "Connect ROACH"



ON the left is the data receiver program, written in QT C++, executable is testEnet. ON the right is the QT python GUI.

The GUI has three tabs:

1) Setup- for changing settings on ROACH
2) Sweep- for frequency sweep of resonators.
3) Noise IV- for taking noise, pulse data, IV curves.

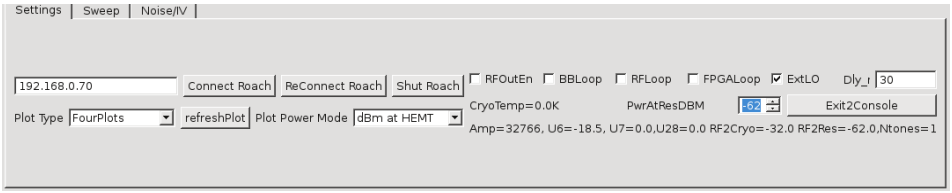
Setup Tab

What each button does on setup tabL

- Connect Roach- open connection to the roach box. Cal QDRs, start up testEnet data receiver. Start up connection to Anritsu.
- ReConnect Roach. Same as Connect except no cal QDR. It is faster, and for debugging only.
- Shut Roach- Shut down connection to roach, shut Anritsu connection, and testEnet.
- PlotType- what kind of plot is displayed.
- Refresh Plot- redraw plot.

- Plot Power Mode- What dBm is displayed: dBm at Resonator in cryostat, dBm at input of Roach RFIn, dBm at mixer input, or raw magnitude of FFt coefficients.
- Check boxes:
    - RfOutEn- check for enable IF board LO oscillator. Do not use of you have anritsu
    - BBLoop- base band loopback on IF board. For debugging.
    - RF Loop- rf loopback from out mixer to in mixer. For debugging.
    - FPGA loop- loopback from waveform source to ADc capture, internal to FOPGA. For debugging.
    - ExtLO- enable external LO input.
- Delay- Type time delay of cryostat cabling in ns. Typically 30ns.

- PwrAtResDBM- Set the dBm power delivered to the resonator in cryostat. Roach_calibrarion.py is used to calculate DAC signal level, and all IF board attenuators to deliver requested power. The power is printed on GUI at various signal points.
- Exit2Console: end GUI thread from QT, and enter ipython console. Generally GUI works still event when console is used. Sometimes opening and saving file windows do not work. Tyope gui() in console to rerun the GUI QT thread.



Settings tab of ROACH software.

To debug the software:

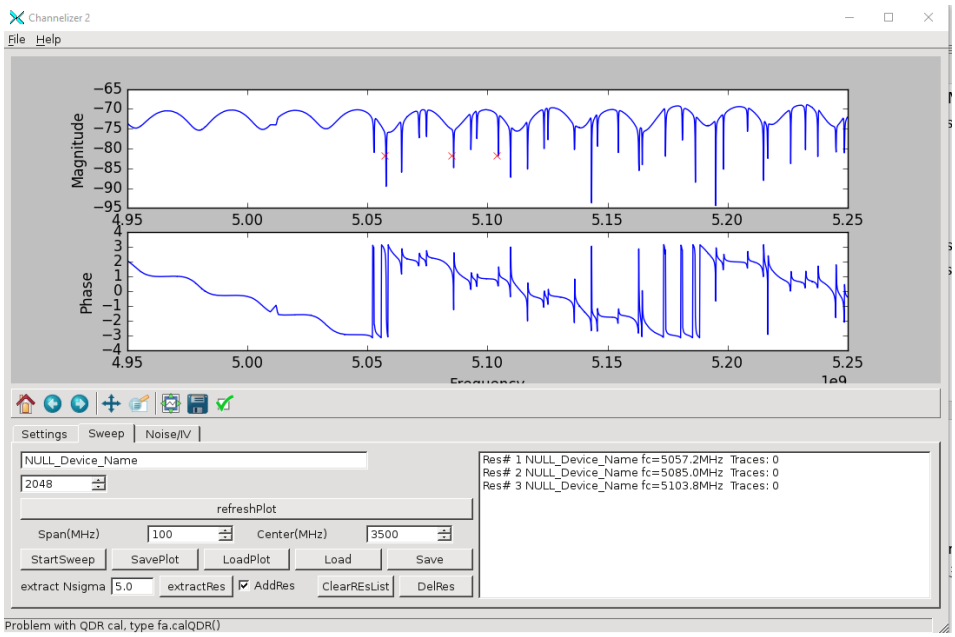
1) *Open natAnalGui.py in a text editor.*
2) *Pick a button name on GUI, like Connect Roach.*
3) *Find "Connect Roach" in the py file.*
4) *Trace to what function are called in the code.*
5) *Work your way down through function calls in py files until you find problem.*
6) *Exit2Console allows copy pasting code into ipython shell to run code as scripts.*

Sweep Tab

The controls on Sweep tab are:

- NullDevice Name- You can name the TEs chip in the cryostat, to be saved in hdf5 files. This is deprecated and a hold-over from MKID days.
- 2048- Number of frequency points to use in a sweep.
- refreshPlot- replot the plot.
- Span- width of sweep in MHz
- Center- center of sweep in MHz.
- StartSweep- begin sweep.
- SavePlot- save sweep just taken as h5 file. Save as a *.h5 file
- Load Plot- load h5 file from older sweep. Load an h5 file.
- Load- load list of resonator freqs from py file. Load a py file that looks like this:
    - *resonator_freqs = [5093.93,5107.94]*
- Save- save list of resonator center freqs to text file. Global variable in py shell resonator_freqs will be created. Save as a *.py file.
- extractRes, extract Nsigma- The software will find resonator centers based on IQ velocity. extractNsigma tells how sensitive the algorithm is. Smaller numbers will find "more" resonators. *There seems to be a bug in this SW here.*
- AddRes- check this then click mouse on magnitude plot. It will put "x" at each resonator and add to the list of res.
- ClearREList- delete all resonators from list.
- DelRes- select res on list, hit Delete to remove.

The software has the concept of a list of resonator center frequencies that are used for all measurements . If you want to use only resonator for noise measurements, then only have that res in the list.



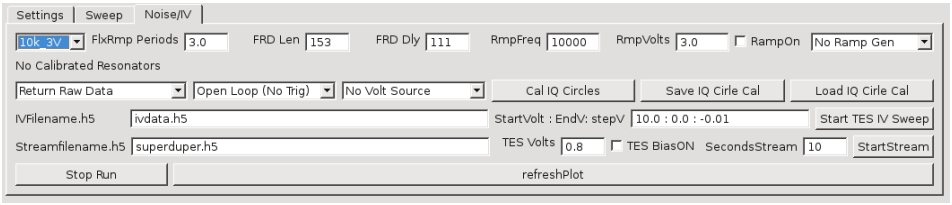
To hand type a list of resonators make a python file like. Frequency in MHz.

resonator_freqs = [5093.93,5107.94]

Save as myres.py. Now Load that file from Sweep tab.

Noise IV Tab

Below is the noise tab.

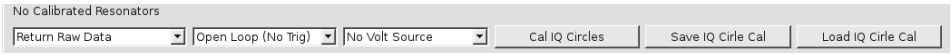


Below we will describe sections of the tab's controls.

Flux Ramp Demod control

Selecting a pulldown on the left will Set the other numbers in the row. The simplest method is to Pull down "No Ramp Gen" to Agilent, to control the Agilent ramp gen. then when we pull down "10V 3V" all the settins will be set correctly: Flux ramp periods, Frd Len, Frd Dly, Ramp Freq, and Ramp volts. Select Ramp On to turn on the Ramp outpt. All settings can be set manually.

- FlxRampPeriods- Number of periods of SQUID oscillations over which to take DFT.
- FRD Len- number of samples in the SQUID oscillations. Max is 510.
- FrdDly- time delay for Sync pulse to Roach. Aligns the glitch. You want to put glitch at END of the segment of data.
- RmpFreq- Sets freq of the Agilent Gen.
- Ramp Volts- Sets voltage on the Gen.
- Ramp OPN- turns on the output of Agilent.
- NoRampGen- Pull down to connect to Agilent.



- Return Raw Data- Pull down to tell ROACH what data to send back. Choices are
  - Return Raw Data- Raw Phase/Mag FFT coeff. No FRD
  - Raw+FRD- Raw FFT coef, FRD calculation.
  - FRD only- Only FRD calc. This is preferred mode of taking data. Lowest data rate.

    - FRD+Trans Data- FRD calculation returned, and IQ translated FFT coef. That is FFT coef are translated in IQ so IQ circle is at origin.
- OpenLoop/Closed Loop- Tell Roach to NOT respond to Suync pulse on Agilent, Or trigger on sync pulse.
- NoVoltSource- Pull down to connecto to Sim 928.
- Cal IQ Circles- When we have a list of resonator frequencys, we must set up ROACH FW and SW to source thise frequencies, and correctly trqnslate the IQ circles for FRD. Hit this button to set up roach before taking any nosie or TES data. Note, that chaning the attenuators on Setup tab requires doing this step again.
- Save IQ …: Save the IQ calibration to hdf.
- Load IQ= load older IQ cal from hdf.

TES Voltage Sweep

We can wave a TES voltage sweep with ROACH. Here are the contr;s

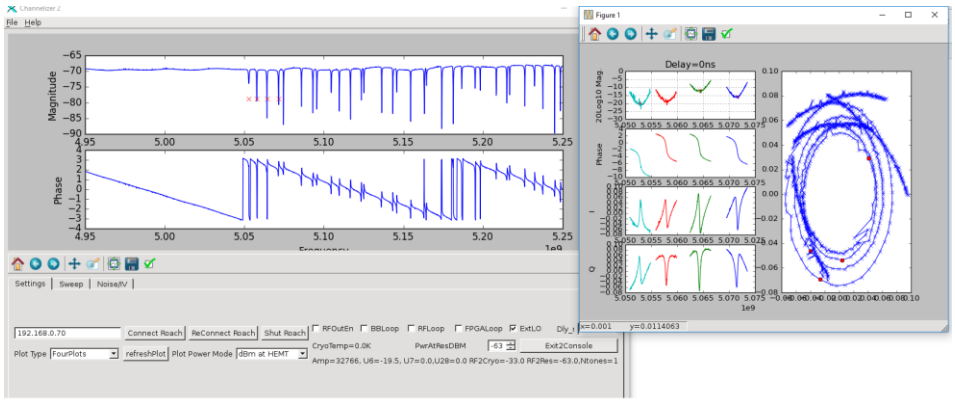
- IVFilename- type in /path/path/name.h5 for your filename.
- StartV/EndV/StepV- we use python list indexing. WE put start voltage, use 10V, then :, then end voltage, use 0.0, then colon, then step voltage, use -0.01.
- Start TES Sweep- this does the IV sweep. You can hit Stop Run to interru[t it.

TES Noise or Pulse measurement:

- To measure noise, put in filename in Streamfilename, like /path/path/name.h5
- Set TES bias in Volts. You need to be connected to sim928.
- Hit TES bias ON if you want to bias TES.
- Put in seconds of data to take. 10s is a good number. Too short and the thing won't work.
- Hit startStream to take the noise. Hit stopRun to interrupt the measurement.

The StopRun buttons will Stop the noise or TES sweep (also the Freq sweep in the sweep plot too). Refresh will replot the plot.

TES measurement Flow



1) Goto Setting tab
2) Connect roach
3) Assure qdr cal'ed. Recal of we need to
    a. Exit to shell
    b. fa.qdrCal()

4) Set attenuation
5) You can type the resonator frequencies in a py file in a text editor. Make a python list: resonator_freqs=[5000,5010….]; #Put freqs in MHz ina py list. Call resonator_freqs. If you make this file, Hit LOAD and proceed to step 11. OR you can NOT type in the list and proceed to step 7, to FIND the resonators.
6) Goto sweep tab
7) Do sweel
8) Find resonators, save py file Save under sweep tab
9) Save sweep file(save plot)
10) Goto Noise tab
11) NoRampGen-> change to Agilent.Assure that "agilent: appears at bottom of GUI.
12) Set NOVoltSource→ sim928. Assure that "sim928" appears at bottom of GUI.
13) If sim920 shows up, goto Settings tab, exit to Console. Type sim.connport(8). Type gui(). Goto Noise tab, Set sim920 back to NoVoltSource, repeat step 12.
14) Cal IQ Circles
15) Save cal.
16) Set return FRD+IQCentered
17) Leave Openloop, no V source.

18) Set filename for noise, set TES bias and select if you want TES bias on. Set stream time in sec, Use at least 10s. Hit startStream. You will see data eventually plot on screen.
19) For closed Loop mode Select closed loop.
20) Set 10k 3V to the settings you want, for ramp generator. Check RAMP on.
21) Set filename in streamfilename, and hit startStream.
22) Set return FRD only.
23) Set streamfilename, tesvolts, teson, and hit startStream.
24) SetIV filanem, set voltage range like 10:0.0:-0.01. Hit StartTESIvSweep.
25) For pulse measurements: On MainWindow (testEnet window), select PulseDetect FRD. Set PhThr to 0.1, EvtsSave 10.
26) On pyGUI, Noise tab, do step 20-23. You should only capture pulses and not all the useless noise.