HIPSR Documentation

Release 1.0

Danny Price

October 30, 2012

CONTENTS

HIPSR is a new reconfigurable digital backend for the Parkes multibeam receiver. HIPSR is capable of running many different firmware modes, so can be used for both high resolution, wide bandwidth spectral line observations, and high time resolution pulsar observations.

Contents:

CHAPTER

ONE

HIPSR USER'S GUIDE

This User Guide gives an overview of the HIPSR system. HIPSR is the next-generation signal processor for the multibeam receiver. If you're reading this, you probably want to know how to use it.

If you're in a hurry, skip down to Observing with HIPSR. Otherwise, continue reading for an introduction to the HIPSR system.

Note: If you're not familiar with the Parkes 64 m telecsope, you should have a read of the telescope user guide.

1.1 Introduction

HIPSR is a new reconfigurable digital backend for the Parkes multibeam receiver. HIPSR is capable of running many different firmware modes, so can be used for both high resolution, wide bandwidth spectral line observations, and high time resolution pulsar observations.

As of writing, there are two main modes of operations supported:

1. **HISPEC** 8192 channel, 400 MHz bandwidth spectrometer for HI observations.
2. **BPSR** high time resolution pulsar modes, used in the HTRU survey.

The documentation here focuses on HISPEC; please head over to [??] for documentation on the pulsar modes.

1.2 Hardware

For an in-depth overview of the HIPSR hardware, head over to the the *Hardware* chapter. To give a quick overview, HIPSR consists of:

- 13x dual-input CASPER iADC digitizer cards,
- 13x CASPER ROACH FPGA digital processing boards for "low-level" signal processing
- A Cisco 10GbE switch for data interconnect
- A CPU/GPU cluster for "high-level" DSP and data storage.

These are located in racks R, S, and T. All of the signal processing nodes in HIPSR are connected with Ethernet; the network architecture is shown in the figure below:

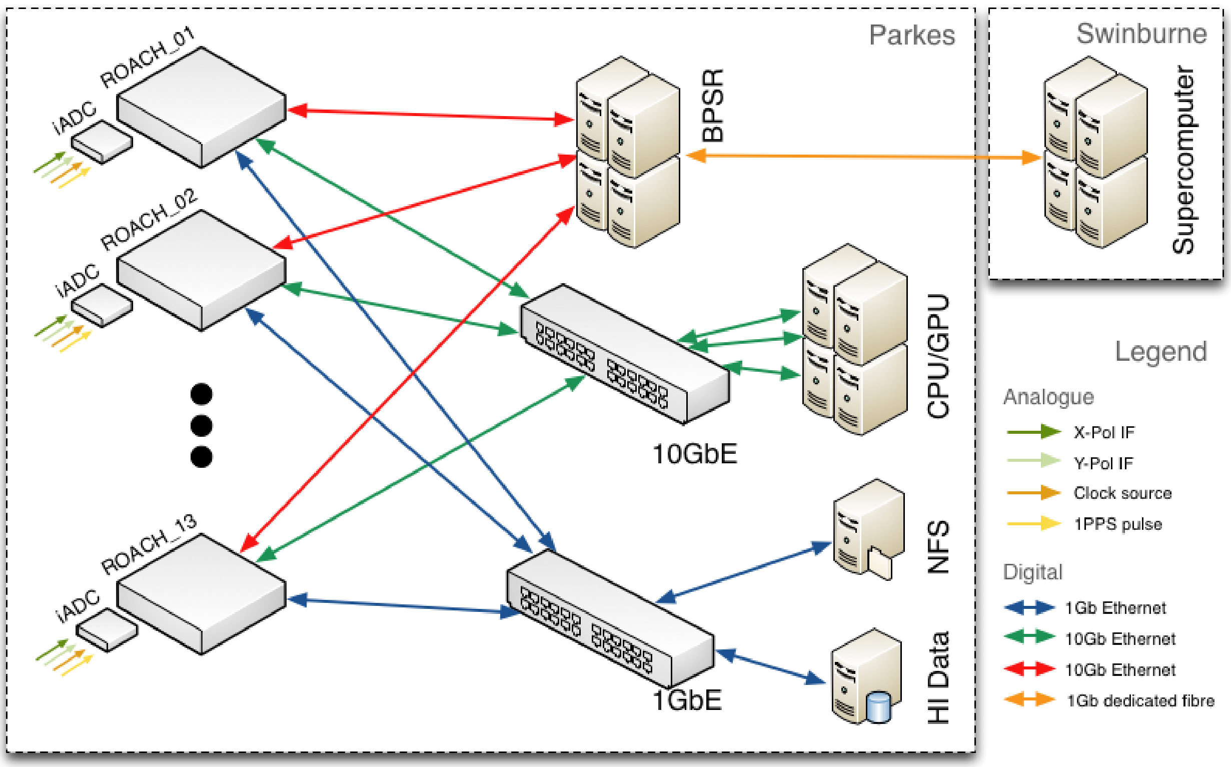


Figure 1.1: *Network architecture for HIPSR.*

Note: The GPU servers are mainly used for pulsar work. All of the signal processing for HISPEC (HI spectrometer) is currently done on the CASPER ROACH boards. Future (lower bandwidth) modes of observation will likely use the GPU servers too.

1.3 Software

If you *really* want to know what's going on, the module listing and API can be found here: *Software*. A quick run-through is shown below.

1.3.1 hipsr-server.py

For HISPEC, all of the signal processing is done on the ROACH boards. The ROACH boards are controlled by a script, *hipsr-server.py*, which collects data from the ROACH boards and metadata (e.g. pointing info) from the telescope control system (TCS). The server script then writes the data to HDF5 files. Note that *hipsr-server.py* does not control telescope pointing, it only collects and collates data/metadata.

1.3.2 hipsr-gui.py

Data from HISPEC can be monitored using the *hipsr-gui.py* script. This script gives shows the bandpass from all 13 beams, and has a beam scope, waterfall plotter and total power monitor.

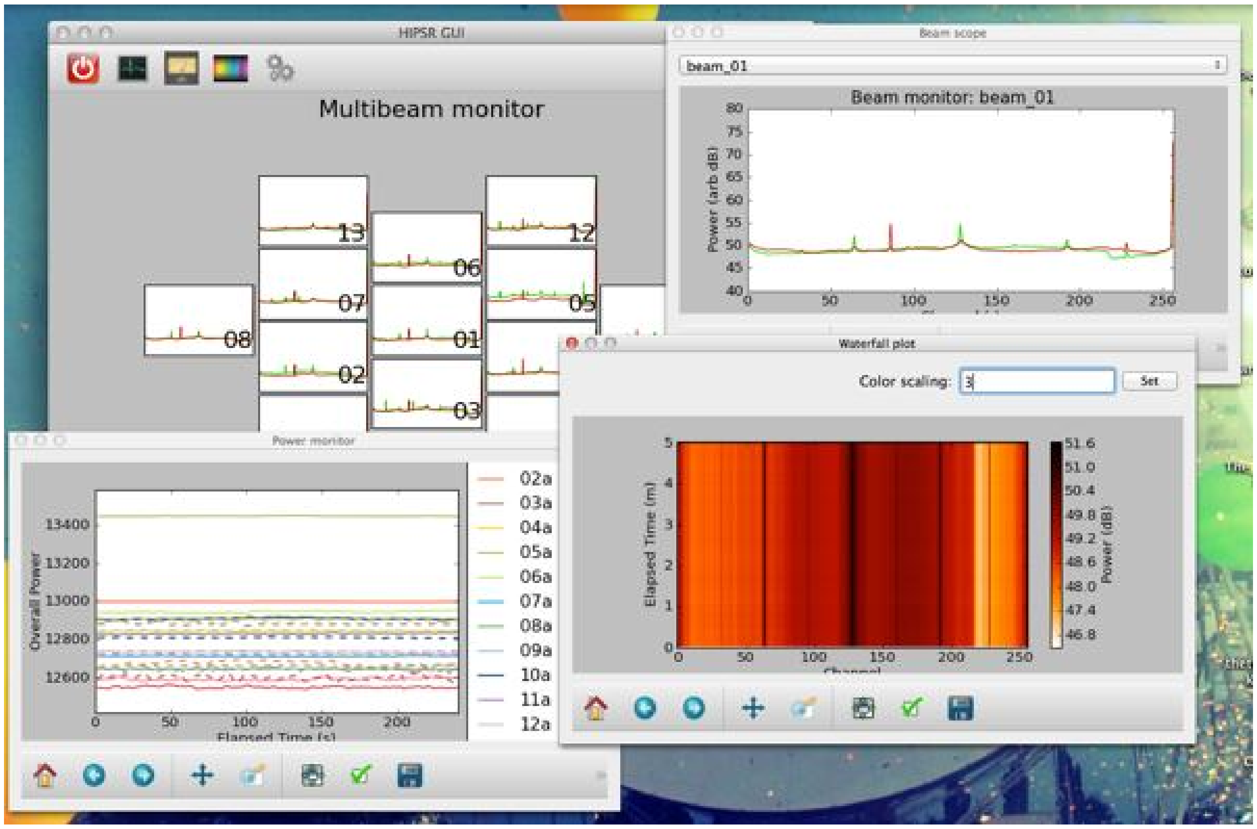


Figure 1.2: *Screenshot of hipsr-gui running on Mac OSX.*

1.3.3 hipsr-writer.py

The *hipsr-writer.py* script converts files from HDF5 to SD-FITS. At the end of your observations, you'll want to run this as most data reduction packages don't support HDF5 yet.

Note: All firmware and software for HIPSR is stored on github: https://github.com/telegraphic/hipsr. It is of course, already installed wherever it needs to be.

1.4 How everything fits together

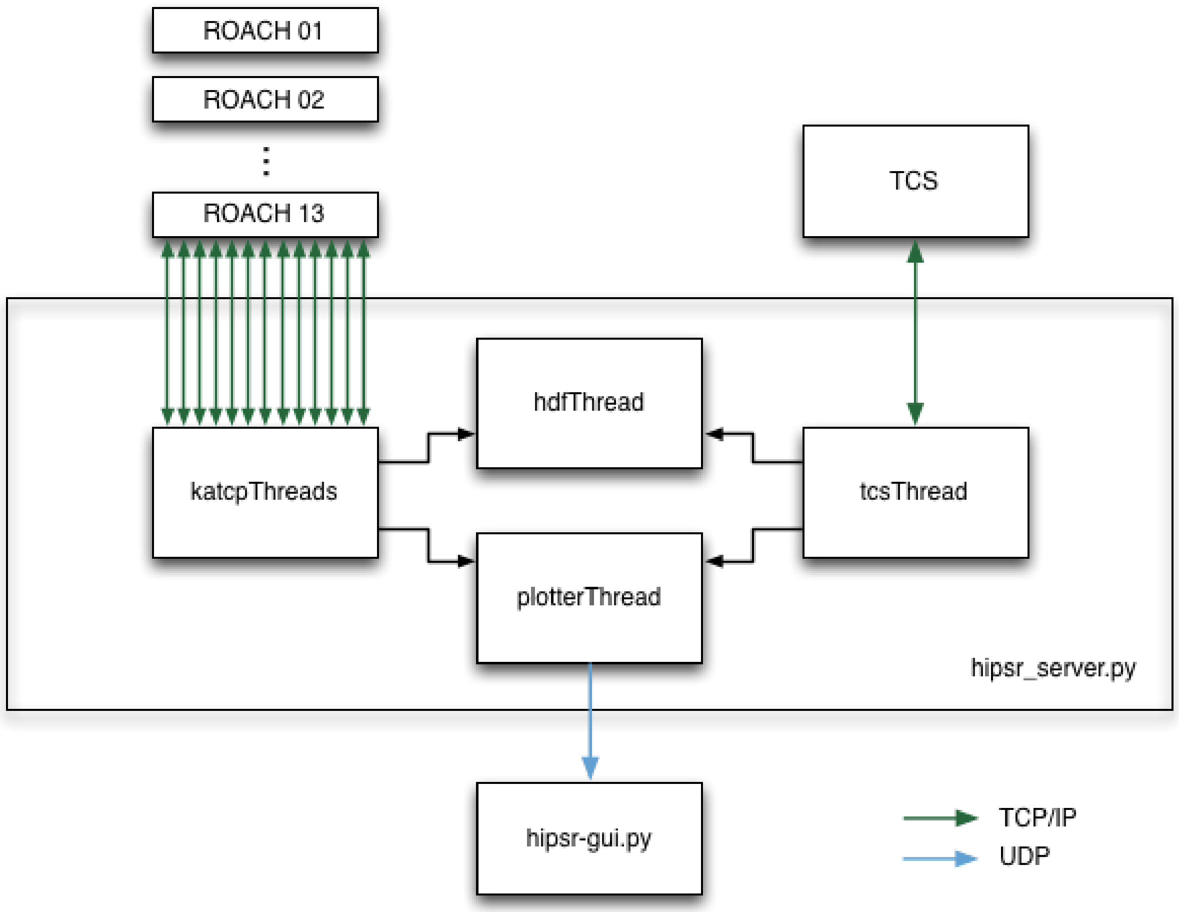


Figure 1.3: *What's really going on.*

The hipsr-server collects, collates and writes data to HDF files. To do this, several threads must be run in parallel. Firstly, a connection to each ROACH board is made using KATCP, a communication protocol which runs over TCP/IP. Each connection runs in a separate thread. In addition to this, a TCP/IP server is set up to communicate with TCS, which sends ASCII command : value pairs with info about telescope setup, observation config and pointing detiails.

Having multiple threads all attempting to write the same HDF file isn't good. So, there's a dedicated HDF thread which has a data input queue, into which the KATCP and TCS threads append data. Finally, so we can see what's going on, TCS and KATCP threads send a subset of data over a UDP connection to the hipsr-gui.py script, which is generally run on a different computer. This UDP connection sends python dictionaries converted into JSON.

1.5 Observing with HIPSR

1.5.1 Starting observations

There are three things that must be started to observe with HIPSR:

1. The telescope control system, TCS
2. The hipsr-server script which runs on hipsr-srv0
3. The hipsr-gui plotter which runs on [?]

To start TCS, get onto one of the Sun machines and type:

```
> tcs
```

Until someone copy and paste's how to use TCS here, you'll have to consult the telescope user guide.

Before you press "go" on TCS, you first need to start the hipsr-server script. To do so, you need to SSH into hipsr-srv0:

```
> ssh [user]@hipsr-srv0(.atnf.csiro.au)
```

Once you've connected, change into directory /data/hipsr/hipsr-server/:

```
> cd /data/hipsr/hipsr-server/
```

Finally, start the server with the command:

```
> ./hipsr_server.py
```

You can optionally pass the argument -p PXXX, where PXXX is your project ID, e.g.:

```
> ./hipsr_server.py -p P641
```

This might save you a few seconds.

1.5.2 Stopping observations

To stop the server, press ctrl + C. The server will close all open files before exiting.

1.5.3 Getting your data

HIPSR data is stored on hipsr-srv0 in /data/hipsr/.

1.6 When things go wrong

Here's a few notes on problems you might run into with HIPSR. For anything to do with the telescope, consult the telescope user guide.

1.6.1 Socket errors

TCS seems to give a random socket error 9: bad file name the first time it connects to the HIPSR server (hipsr-srv0). This doesn't seem to matter.

Sometimes the TCS socket (59012) is held open, and neither TCS or HIPSR will negotiate a new TCP/IP connection. If this happens, you can check the port status by typing:

```
> netstat | grep 59012
```

You can check whether there's another instance of hipsr_server by typing:

```
> ps aux | grep hipsr_server
```

Caution: If there's a rogue instance of hipsr-sever.py running, you may have to kill it. This can be done with the command:

```
> kill [PID]
```

However, take care when doing this that you kill the right thing.

CHAPTER

TWO

HARDWARE

This page details the hardware in use in the HIPSR system. HIPSR consists of digitizer cards; FPGA digital processing boards for “low-level” DSP; a 10GbE switch for data interconnect; and, a CPU/GPU cluster for “high-level” DSP and data storage.

2.1 HIPSR Packetized Architecture

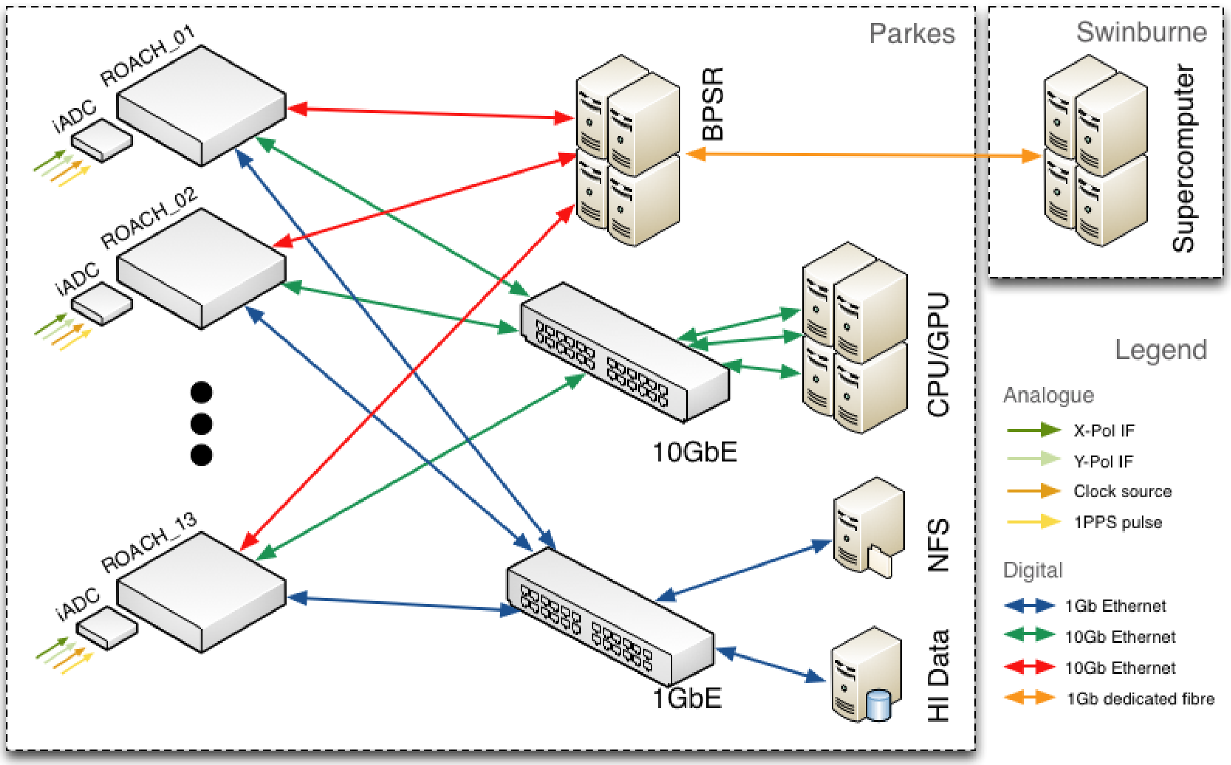


Figure 2.1: *Network architecture for HIPSR.*

The HIPSR system may be described as a multi-node DSP ‘frontend’ connected to a multi-node DSP ‘backend’ by a 10Gb Ethernet (10GbE) switch. In the HIPSR system detailed here, DSP tasks are divided between FPGAs and GPGPUs as appropriate. The FPGA based DSP frontend conducts signal processing tasks such as signal filtering and channelization; the GPGPU based DSP backend performs higher-level, moderate- to low- bandwidth DSP such as running pulsar processing pipelines. A diagram of the system archiecture is shown below.

The inclusion of a switch greatly increases the flexibility of the system as data can be dynamically routed between nodes. It does, however, require all data to be packetized along with metadata that describes packet's data payload. Nevertheless, using a switched 10GbE based packetized architecture instead of defining and commissioning a bespoke backplane has drastically decreased development time: all interconnect is off-the-shelf, industry standard, commodity hardware.

2.2 CASPER ROACH boards

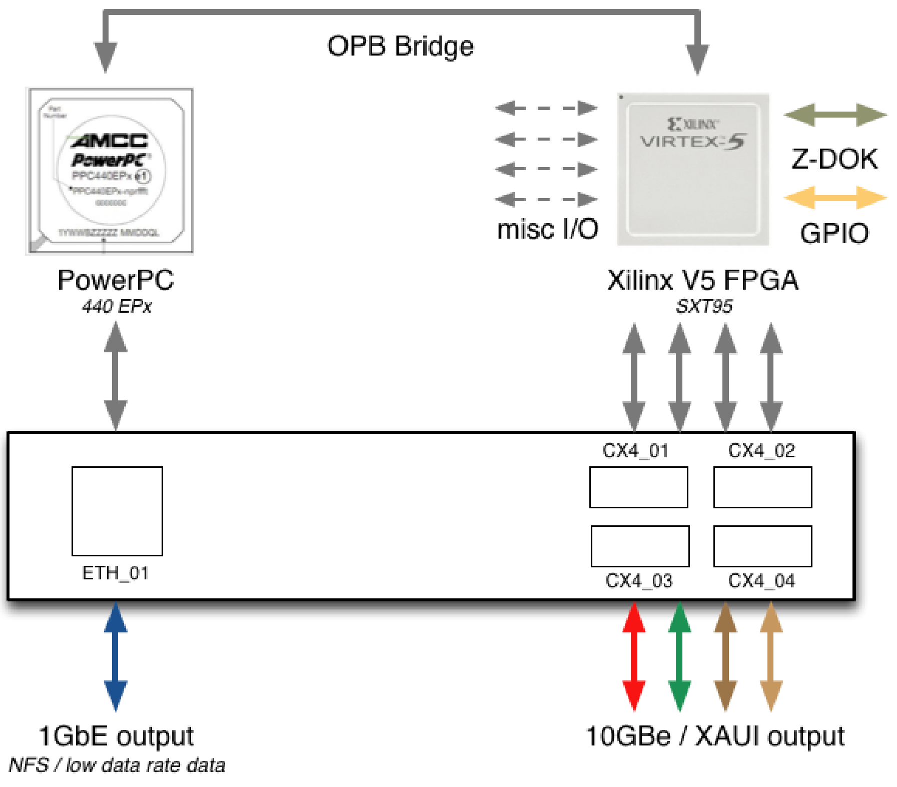


Figure 2.2: *Roach Input/Output*

ROACH (Reconfigurable Open Architecture Computing Hardware) is a standalone FPGA processing board, developed by the Collaboration for Astronomy Signal Processing and Electronics Research.

The centrepiece of ROACH is a Xilinx Virtex 5 FPGA (SX95T). A separate PowerPC runs Linux and is used to control the board (program the FPGA and allow interfacing between the FPGA "software registers/BRAMs/FIFOs" and external devices using Ethernet).

Two quad data rate (QDR) SRAMs provide high-speed, medium-capacity memory (specifically for doing corner-turns), and one DDR2 DIMM provides slower-speed, high-capacity buffer memory for the FPGA. The PowerPC has an independent DDR2 DIMM in order to boot Linux/BORPH.

The two Z-DOK connectors allow ADC, DAC and other interface cards to be attached to the FPGA, in the same manner as the IBOB allowed (with backwards compatibility for the ADC boards used with the IBOB).

Four CX4 connectors provide a total of 40Gbits/sec bandwidth for connecting ROACH boards together, or connecting them to other XAUI/10GbE-capable devices (such as computers with 10GbE NICs and 10GbE switches).

For more detailed descriptions, see:

- https://casper.berkeley.edu/wiki/Main_Page
- https://casper.berkeley.edu/wiki/ROACH
- https://casper.berkeley.edu/wiki/ROACH_Architecture

2.3 iADC digitizer cards

Each HIPSR ROACH board is populated with an iADC digitizer card. These cards are based on the Atmel/e2V AT84AD001B 8-bit Dual 1Gsps ADC:

- http://www.e2v.com/assets/media/files/documents/broadband-data-converters/doc0817I.pdf

More information about the card is available at:

- https://casper.berkeley.edu/wiki/ADC2x1000-8

2.4 Cisco switch

The DSP frontend and backend nodes are connected via a 10Gb Ethernet switch. This Cisco 4900M switch is configured with 24 CX4-type ports, 13 of which are connected to the DSP frontend ROACH boards, and 8 of which are connected to the server nodes. The switch allows bi-directional data flow between nodes. The 10GbE switch also has a dedicated 1GbE fibre link to the Green II supercomputer

at Swinburne University in Melbourne.

All nodes are also connected by 1GbE, through a Cisco 3750 switch. For low bandwidth applications where all of the necessary DSP can be conducted on the FPGA, all data may be read off the 1GbE ports of the ROACH boards.

Item	Description
WS-C4900M	CISCO Base system with 8 X2 ports and 2 half slots
S49MIPBK9-15002SG	CISCO Cisco CAT4900M IOS IP BASE SSH
PWR-C49M-1000AC	CISCO 4900M AC power supply, 1000 watts
PWR-C49M-1000AC/2	CISCO Redundant AC PS for 4900M
CAB-AS3112-C15-AU	CISCO AS-3112 to IEC-C15 8ft Aus
MEM-C4K-FLD128M	CISCO Catalyst 4900M Compact Flash, 128MB Option
4900M-X2-CVR	CISCO X2 cover on 4900M
WS-X4908-10GE=	CISCO 8 port 2:1 10GbE (X2) line card for 4900M series (this is the expansion module - there are two slots in each chassis)

2.5 HIPSR Servers

2.5.1 Server nodes

The GPGPU server nodes are bespoke systems, built to specification by Silicon Graphics Pty. Ltd. Each of the 8 server nodes are comprised of:

- dual 2.66 GHz Intel Xeon six-core CPUs,
- 48 GB DDR3 memory, and
- dual Nvidia Tesla C2070 GPGPUs.

While it is intended that a majority of DSP will be conducted on the GPGPUs, the Intel Xeon CPUs may also be used, if required.

As each server has a single CX4-type 10GbE network interface card (NIC), the input data rate for each server is limited to 10 Gb/s. Given that there are 26 IFs which must be processed by 8 server nodes, the DSP frontend must decrease the data rate of

2 x 400MHz x 8bit = 6.4 Gb/s

from each IF from 6.4Gb/s to about 3.0Gb/s.

2.5.2 hipsr-srv0

In addition to the GPGPU server nodes, there is a server which provides monitor and control for the DSP frontend, and provides data storage. This server has:

- a 2.4 GHz Intel Xeon four-core CPU,
- 24 GB DDR3 memory,
- 5TB RAID HDD

As it is not used for DSP, it does not have a GPGPU. This server runs a DHCP daemon and provides a network file system (NFS) to the ROACH boards.

2.6 Cabling

In March 2012, the HIPSR hardware was installed into RFI shielded racks R and S, located on the second floor of the Parkes 64m telescope building (this room lies underneath the telescope dish). * A set of BNC cables were laid under the floor from the existing multibeam IF distribution panel to BNC feedthroughs installed on the racks. * Another set of cables connect the BNC feedthrough to the SMA input of the iADC cards. * A third set of cables connects the 1PPS input of the iADC to a PPS distribution unit, and a clock signal is distributed to each board by a fourth set of SMA cables.

Each server node and ROACH board is connected to a 1GbE switch by an Ethernet cable (CAT-5E STP). Similarly, each server node and ROACH is connected to a 10GbE switch via CX4 type cables.

CHAPTER

THREE

SOFTWARE

3.1 hipsr_server.py

HIPSR wideband spectrometer server script. This script reprograms and reconfigures the roach boards, creates a data file, and then begins collecting data from TCS and the roach boards.

A seperate thread is created for each roach board, so that reading and writing data can be done in parallel. In addition, there is a thread which acts as a server that listend for TCS messages. To write to the HDF data file, these threads append an I/O requests to a FIFO (Queue.Queue) which is constantly checked by yet another thread.

Created by Danny Price on 2011-10-05. Copyright (c) 2012 The HIPSR collaboration. All rights reserved.

class `hipsr_server.`**`DataFetcher`**`(`*queue*`)`
Thread worker function for retrieving data from roach boards

Methods

`run``()`
Thread run method. Fetch data from roach

class `hipsr_server.`**`Logger`**`(`*filename*, *filepath*`)`
Logger which records the output of stdin to file

Methods

`hipsr_server.`**`createNewFile`**`()`
Closes current file and creates a new one

`hipsr_server.`**`getSpectraThreaded`**`(`*fpgalist*, *queue*`)`
Retrieves spectral data from multiple roach boards

Spawns multiple threads, with each thread retrieving from a single board. A queue is used to block until all threads have completed.

`hipsr_server.`**`hdfWriter`**`()`
Worker thread that writes to HDF files

`hipsr_server.`**`tcsServer`**`()`
A TCP server which listens for TCS commands and parses them

`hipsr_server.`**`toJson`**`(`*npDict*`)`
Converts a dictionary of numpy arrays into a dictionary of lists.

3.2 hipsr_gui.py

This script starts a Qt4 + matplotlib based graphical user interface for monitoring HIPSR's data output.

3.2.1 Requirements

PySide (or PyQt), for Qt4 bindings numpy, matplotlib.

3.2.2 TODO:

- Zoom in on selected area
- Tabbed versions
- Pause & explore plot
- save image
- show which beam is which

class `hipsr_gui.`**`HipsrGui`**

HIPSR GUI class

A Qt4 Widget that uses matplotlib to display data from UDP packets.

Methods

`bufferUDPData``()`

A circular buffer for incoming UDP packets

`createMultiBeamPlot``(`*numchans=256*`)`

Creates 13 subplots in a hexagonal array representing the multibeam feeds

`createOverallPowerPlot``(`*numchans=120*, *beamid='beam_01'*`)`

Creates an overall power vs time plot.

`createSingleBeamPlot``(`*numchans=256*, *beamid='beam_01'*`)`

Creates a single pylab plot for HIPSR data.

`createWaterfallPlot``()`

Creates a single imshow plot for HIPSR data.

`initUI``(`*width=1200*, *height=750*`)`

Initialize the User Interface

Parameters **width: int** :

width of the UI, in pixels. Defaults to 1024px

height: int :

height of the UI, in pixels. Defaults to 768px

`onBeamSelect``(`*beam*`)`

Beam selection combo box actions

`updateAllPlots``()`

Redraw all graphs in GUI

`updateOverallPowerPlot`(*key*, *xx*, *yy*)
: Update power monitor plot.

`updateWaterfallPlot`(*new_data*)
: Updates waterfall plot with new values

`updateWaterfallThreshold`()
: Change the threshold value for the waterfall plot

CHAPTER

FOUR

GATEWARE

4.1 Overview

The MSSGE toolflow (short for Matlab/Simulink/System Generator/EDK) is the platform for FPGA-based CASPER development, which stitches together several design and implementation environments.

It is better known as the bee_xps toolflow, which was developed at the Berkeley Wireless Research Center (BWRC) as a high-level design tool for the BEE and BEE2 platforms. We have extended it to work with all other CASPER boards as well. It provides a graphical Matlab/Simulink design environment using the Xilinx System Generator Toolbox, and abstracts the Xilinx implementation details behind a one-click compile interface.

For installation instructions, see the MSSGE Toolflow page:

- https://casper.berkeley.edu/wiki/MSSGE_Toolflow

4.2 Spectral line modes

4.2.1 HIPSR Wideband Spectrometer

Current gateware: hipsr_16_2011_Oct_01_0824.bof

This is a 400MHz bandwidth, 8192 channel polyphase filterbank. The filterbank is 4-taps, with hamming window coefficients.

The registers that can be used to control the board logic are listed below.

Register	Description
quant_gain	Gain used in quantization, accepts an eight bit number which sets the shift before quantisation. Values under 00001111 shift the data to the left; values over 00001111 shift the data to the right.
mux_sel	Digital noise source or the ADC? A value of 1 selects the digital noise source, and 0 selects the ADC input.
fft_shift	FFT shift, selects which stages of the FFT the bits are shifter to the left. Accepts 12 bits, one bit for each stage in the FFT. For example, 111111111111 shifts every stage, and 010101010101 shifts every second stage.
acc_len	Number of accumulations, in clock cycles. As the FPGA is clocked at 200MHz, and there are 8192 channels, a value of 24414 corresponds to 1 second. The maximum value is 2^18 accumulations, which gives about 10s, although care must be taken to ensure there is no overflow.

Resource utilization summary

The design uses a fair proportion of slices and BRAMs, but there is a fair amount of DSP48E logic left.

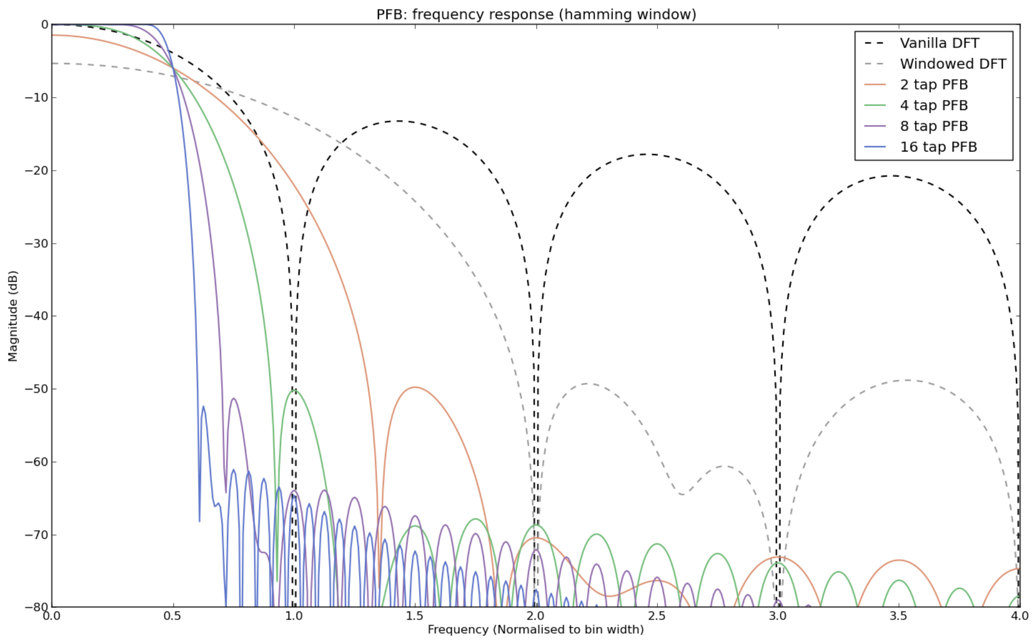


Figure 4.1: *Filterbank response for different numbers of taps. We are using 4-taps in this 8192 channel design.*

Logic distribution and feature utilization		
Number of occupied Slices:	10,305 out of 14,720	70%
Number of LUT Flip Flop pairs used:	36,261	
Number with an unused Flip Flop:	2,830 out of 36,261	7%
Number with an unused LUT:	9,846 out of 36,261	27%
Number of fully used LUT-FF pairs:	23,585 out of 36,261	65%
Number of unique control sets:	167	
Number of slice register sites lost		
to control set restrictions:	229 out of 58,880	1%
Number of BlockRAM/FIFO:	191 out of 244	78%
Number using BlockRAM only:	191	
Total primitives used:		
Number of 36k BlockRAM used:	185	
Number of 18k BlockRAM used:	12	
Total Memory used (KB):	6,876 out of 8,784	78%
Number of DSP48Es:	222 out of 640	34%

4.3 Pulsar modes

The CASPSR firmware is based on the Packetized Astronomy Signal Processor firmware designed by Terry Filiba. The BPSR firmware is based on the Parspec firmware designed by Peter Macmahon and ported to ROACH by Danny Price.

4.3.1 ADC to Ten Gigabit Ethernet

Current gateware: adc_to_10gbe_2011_Oct_18_1156.bof

CASPSR uses a stripped out PASP. It has a more accurate reset, and just turns the ADC input into 10GE packets.

- https://casper.berkeley.edu/wiki/Packetized_Astronomy_Signal_Processor

Register	Description
todo	todo.

Resource utilization summary

This design fitted onto an iBOB, so unsuprisingly isn't pushing the ROACH in any way.

Logic distribution and feature utilization		
Number of occupied Slices:	6,258 out of 14,720	42%
Number of LUT Flip Flop pairs used:	16,637	
Number with an unused Flip Flop:	6,747 out of 16,637	40%
Number with an unused LUT:	3,378 out of 16,637	20%
Number of fully used LUT-FF pairs:	6,512 out of 16,637	39%
Number of unique control sets:	459	
Number of slice register sites lost		
to control set restrictions:	1,041 out of 58,880	1%
Number of BlockRAM/FIFO:	21 out of 244	8%
Number using BlockRAM only:	21	
Total primitives used:		
Number of 36k BlockRAM used:	12	
Number of 18k BlockRAM used:	16	
Total Memory used (KB):	720 out of 8,784	8%

4.3.2 Parkes Spectrometer (Parspec)

Current gateware: parspec_01_2011_Oct_12_1520.bof

Parspec is a 400MHz bandwidth, 1024 channel, 2-tap polyphase filterbank based spectrometer.

- https://casper.berkeley.edu/wiki/Parspec

Register	Description
todo	todo.

Resource utilization summary

Parspec was designed and spec'd to fit onto an iBOB; now it's been ported to ROACH it feels positively roomy.

Logic distribution and feature utilization		
Number of occupied Slices:	6,171 out of 14,720	41%
Number of LUT Flip Flop pairs used:	20,028	
Number with an unused Flip Flop:	3,139 out of 20,028	15%
Number with an unused LUT:	5,881 out of 20,028	29%
Number of fully used LUT-FF pairs:	11,008 out of 20,028	54%
Number of unique control sets:	349	
Number of slice register sites lost		
to control set restrictions:	728 out of 58,880	1%
Number of BlockRAM/FIFO:	85 out of 244	34%
Number using BlockRAM only:	85	
Total primitives used:		
Number of 36k BlockRAM used:	10	
Number of 18k BlockRAM used:	119	
Total Memory used (KB):	2,502 out of 8,784	28%
Number of DSP48Es:	164 out of 640	25%

CHAPTER

FIVE

ROACH SETUP/CONFIG

5.1 IF mappings

I	Q	beam ID	hostname	nickname	Rack	iADC	ADC notes
1	2	beam_01	roach040149	Drake	R	iADC v1.1a	oxford loan
3	4	beam_02	roach040150	Hendrix	R	iADC v1.1a	oxford loan
5	6	beam_03	roach040151	Keenan	R	iADC v1.1a	oxford loan
7	8	beam_04	roach040152	Mackaye	R	iADC v1.1a	oxford loan
9	10	beam_05	roach040154	Reznor	R	iADC v1.1a	oxford loan
11	12	beam_06	roach040155	Barrett	S	iADC v1.1a	BPSR spare
13	14	beam_07	roach040148	Curtis	S	iADC v 1.1b	HIPSR
15	16	beam_08	roach040140	Albarn	S	iADC v 1.1b	HIPSR
17	18	beam_09	roach040158	Willis	S	iADC v1.1a	oxford loan
19	20	beam_10	roach040157	Waits	S	iADC v1.1a	oxford loan
21	22	beam_11	roach040156	Yorke	S	iADC v 1.1b	HIPSR
23	24	beam_12	roach040146	Cobain	S	iADC v1.1a	oxford loan
25	26	beam_13	roach040147	Cohen	S	iADC v1.1a	oxford loan

TODO: Check and update

5.2 IP Address Allocation

hostname	nickname	Eth0 MAC	Eth0 IP	Xport IP
roach040149	Drake	02:00:00:04:01:49	192.168.1.149	192.168.1.49
roach040150	Hendrix	02:00:00:04:01:50	192.168.1.150	192.168.1.50
roach040151	Keenan	02:00:00:04:01:51	192.168.1.151	192.168.1.51
roach040152	Mackaye	02:00:00:04:01:52	192.168.1.152	192.168.1.52
roach040154	Reznor	02:00:00:04:01:54	192.168.1.154	192.168.1.54
roach040155	Barrett	02:00:00:04:01:55	192.168.1.155	192.168.1.55
roach040148	Curtis	02:00:00:04:01:48	192.168.1.148	192.168.1.48
roach040140	Albarn	02:00:00:04:01:40	192.168.1.140	192.168.1.40
roach040158	Willis	02:00:00:04:01:58	192.168.1.158	192.168.1.58
roach040157	Waits	02:00:00:04:01:57	192.168.1.157	192.168.1.57
roach040156	Yorke	02:00:00:04:01:56	192.168.1.156	192.168.1.56
roach040146	Cobain	02:00:00:04:01:46	192.168.1.146	192.168.1.46
roach040147	Cohen	02:00:00:04:01:47	192.168.1.147	192.168.1.47

TODO: Check and update

CHAPTER

SIX

NOISE DIODE CONTROL

To get noise adding radiometer modes working, the noise diode will need to be controlled by HIPSR. To do so, a "master" roach board will control the noise diode, using square wave generator gateware tied to the GPIO output on the ROACH board.

The sections that follow give more detail into how this is/will be achieved.

6.1 Firmware overview

While the firmware is still in development, here's the general idea:

- Sync all boards up using the 1PPS and some sync pulse logic.
- Generate a square wave to control the noise diode
- Take the ADC data, do a small FFT and take the power.
- Demux the FFT data into ON and OFF. This is controlled by the square wave generator.
- Accumulate the ON and OFF spectra
- Store the sum ON-OFF for each channel in a shared BRAM.

Note that while only one board controls the noise diode, all boards require square wave generators. The other option would be to distribute the square wave between boards with cables; this would be pretty horrible.

6.2 Square wave generation

Firstly, we need to generate the signal with which to control the noise diode. This is done with the **sq_wave_gen** block:

6.3 FFT and power

The output of the ADC is changed from 8_7 bits to 18_17 bits, then FFT'd. There are 16 complex channels output. The power (autocorrelation) is taken, then the signal is quantized down to 8 bits, in prep for the vector accumulator.

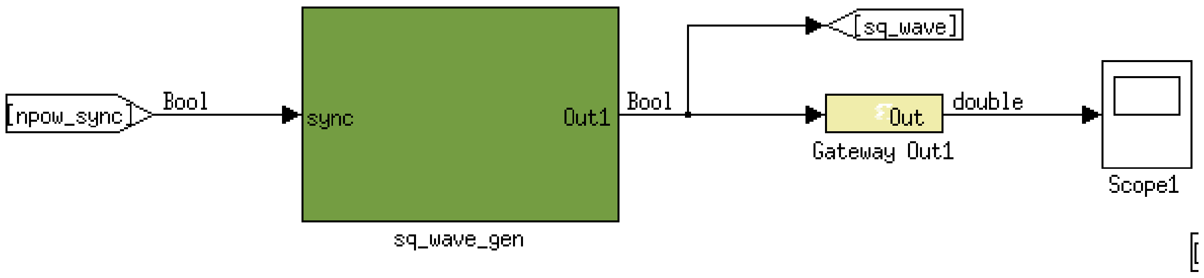


Figure 6.1: The **sq_wave_gen** block generates a boolean signal which cycles between ON (True), and OFF (False).

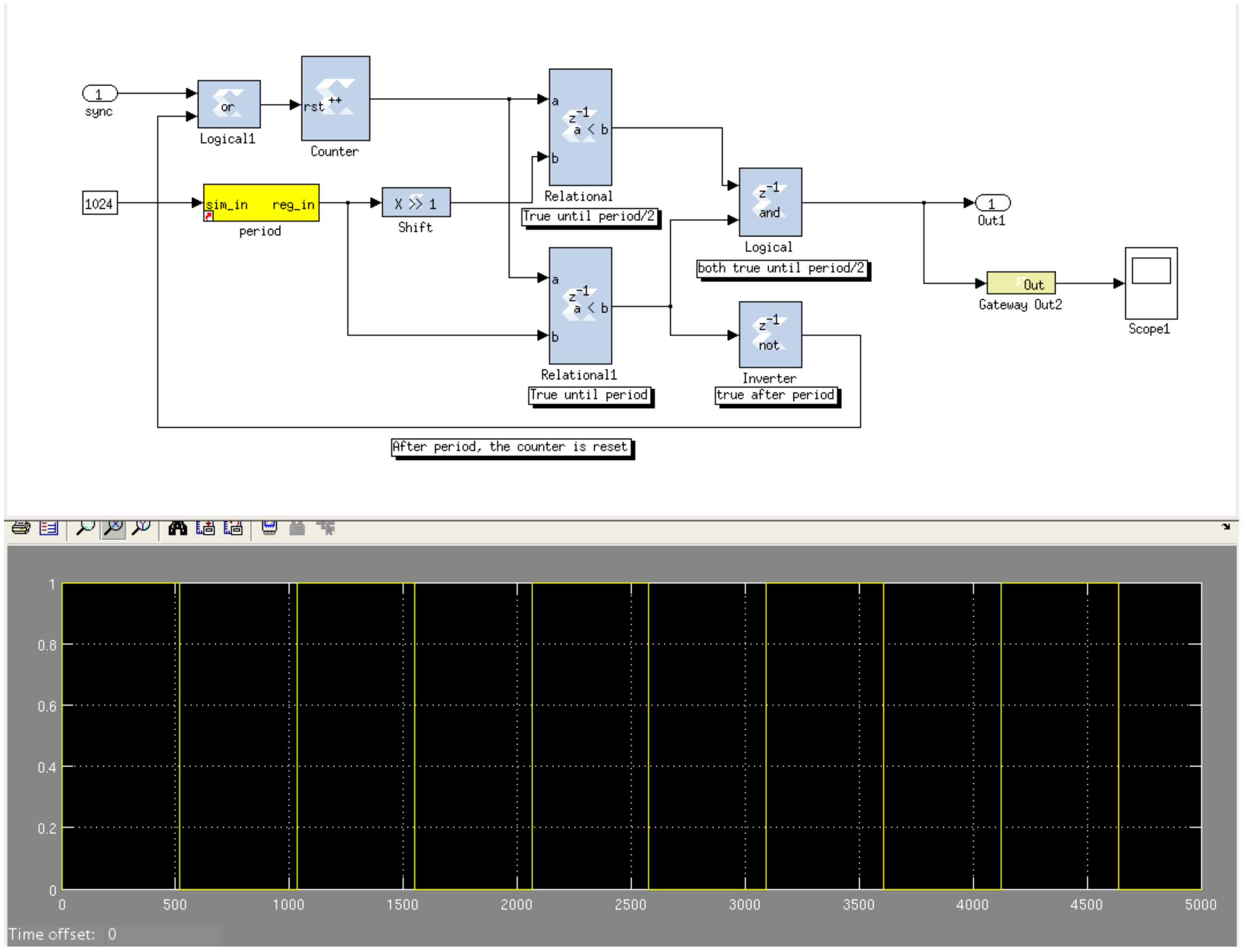


Figure 6.2: The logic inside the **sq_wave_gen** block, and a simulated square wave. This produces a boolean square wave signal of the given period. The period can be controlled by the **period** register.

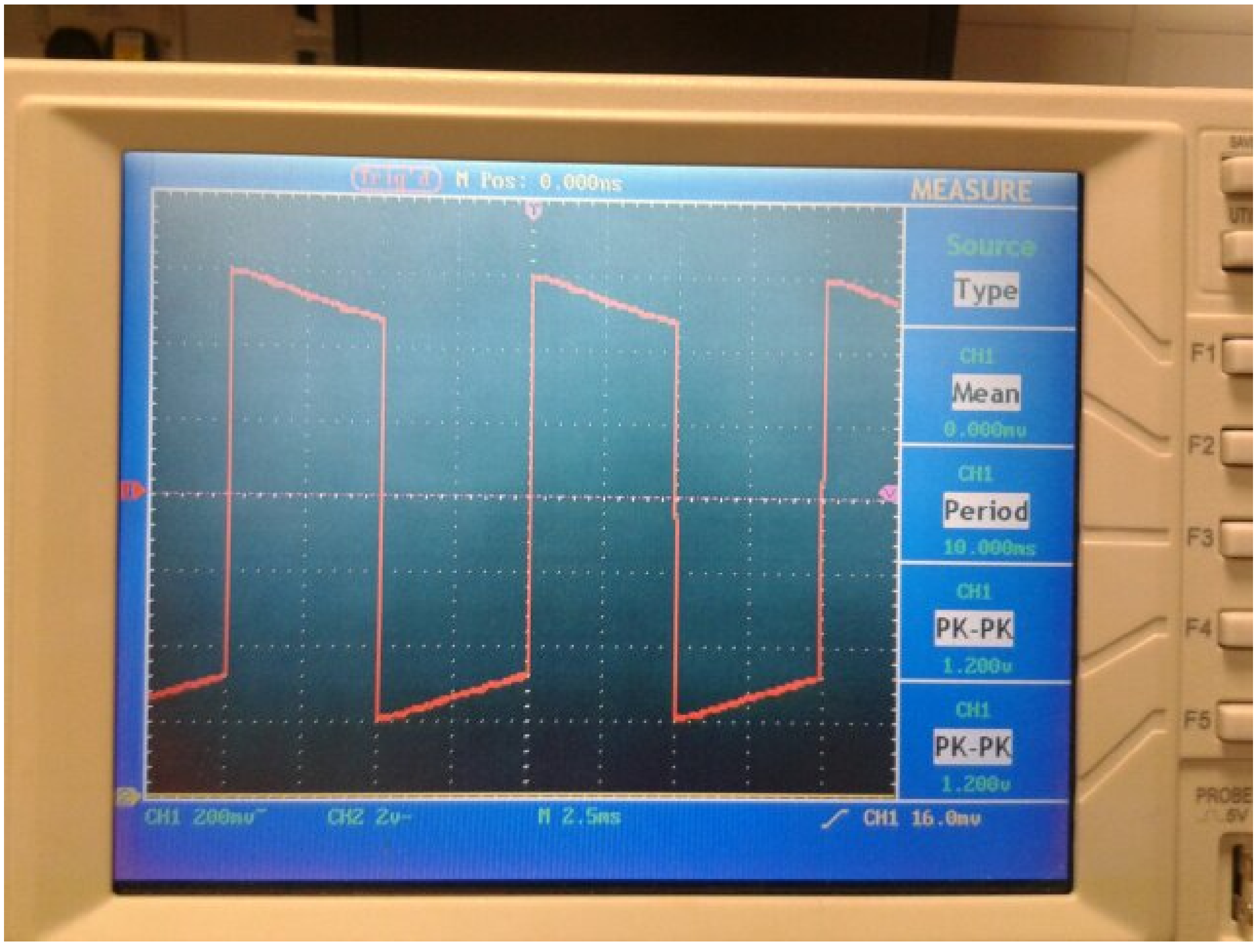
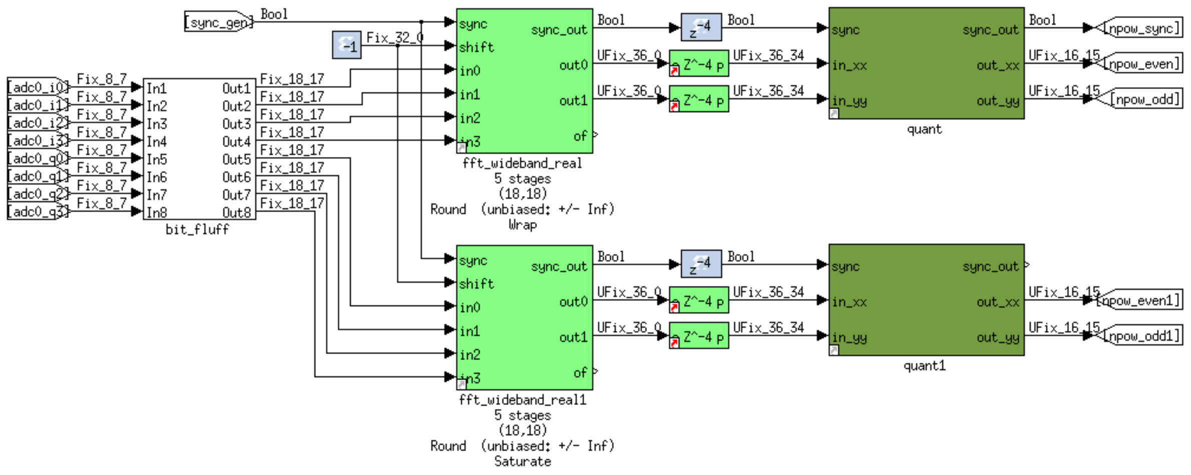


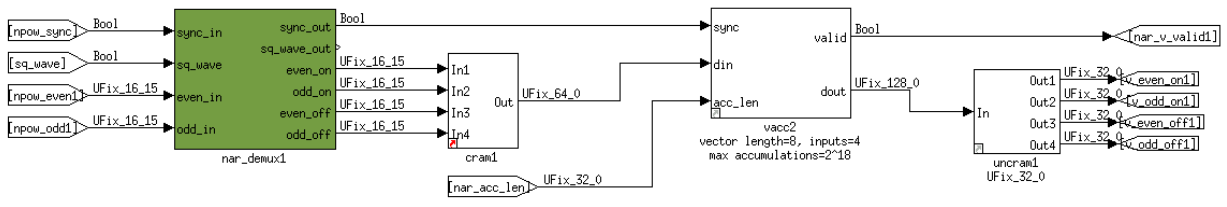
Figure 6.3: The resulting square wave, as measured on an oscilloscope in the lab. The output of **sq_wave_gen** is tied to the GPIO headers. These can run at up to 10 MHz, and output a 3.3V level signal.



6.4 Demux and accumulation

The output of the FFT needs to be separated into ON and OFF, which corresponds to the noise diode being on and off. To do this, the square wave generator is used as an input to a 'demux' block.

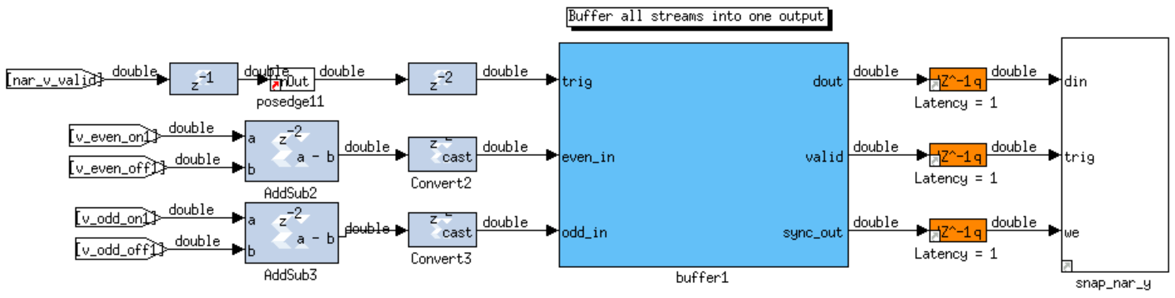
After separating the two signals, they are fed into vector accumulators. As the vectors are short (2x8 channels), bits grow quickly. As such, the input is 8 bit, and the output is 32 bit. This gives 2^24 max accumulations before bit overflows can occur (might need to increase this).



6.5 Data readout

To compute the sum ON - OFF, the outputs of the vector accumulators are fed into an adder block (which is set to minus). This creates a 33 bit signed number (can be -ve), which is recast to 32.

After this, the signal is buffered and then fed into a shared BRAM, which can be accessed via the ROACH's Power PC.



6.6 Testing

TODO.

CONTACTS

Contact Name	Role	Email
Danny Price	Project Manager	danny.price@astro.ox.ac.uk
Lister Staveley-Smith	Project Leader – Phase 1	
Matthew Bailes	Project Leader – Phase 2	
TBD	Project Engineer (software)	
TBD	Project Engineer (hardware)	
Erik Lensson	CSIRO coordinator	
Douglas Bock	CSIRO coordinator	
Brett Preisig	Parkes technical coordinator	
George Hobbs	CSIRO Pulsar Scientist	
Mike Jones	Oxford Coordinator	
Matthew Bailes	Swinburne Coordinator	
Robert Braun	Project CI: HI, noise cancellation	
Steven Tingay	Project CI: Digital systems	
Sascha Schediwy	Project Participant: System testing	
Danny Price	Project Engineer: FPGA gateware	
Willem Van Straten	Project Engineer: CPU/GPU	
Jonathon Kocz	Project Engineer: PSR modes, RFI	
Andrew Jameson	Project Engineer: GPU/GPU	

CHAPTER

EIGHT

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

PYTHON MODULE INDEX

INDEX