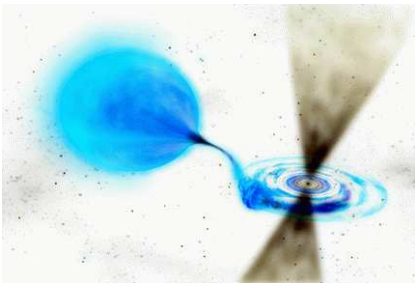

CICADA Note #3

Event Capture α Project Review

Glen Langston, Randy McCullough, John Ford, Brandon Rumberg, Patrick Brandt

NRAO Green Bank

July 19, 2007

Abstract

The Event Capture α IBOB/ADC based data acquisition system has been completed. We present tests of the Event Capture system with IF test sources and astronomical observations using the 43m telescope. This document presents an overview of the FPGA system design, FPGA and Linux side "C" software, and explains the methods for using this design to capture short duration, $< 6\mu-$second, transient events in the input IF signals.

We review features and limitations of the design and point out potential improvements in future designs.

Change Record

Revision	Date	Author	Sections/Pages Affected
	Remarks		
1.0	2007-July-19	G. Langston	All
	Initial version for the project close-out		
1.1	2007-July-19	G. Langston	All
	Add Brandon and Patrick's descriptions of the design and software.		

Background

This document describes the Event Capture α, FPGA based, data acquisition system. The Event Capture α project plan was described in CICADA memo 2, by Langston, McCullough, Ford, Rumberg and Brandt.

This project was conceived in early 2007, and implementation began in May 2007 by Rumberg and Brandt. The project was specifically intended to be a learning project use of the U.C. Berkeley hardware for data acquisition. For this reason, the project scope was limited to acquiring any astronomical data, time tagging the data with accuracy of a few $\mu-$seconds and writing the data to a Linux-based file system. No constraint was placed on the system for down time between events nor was use of the high speed ethernet links required.

The hardware configuration for the Event Capture α design includes one Berkeley ADC board and one Berkeley IBOB board. There are four inputs to the ADC board.

Clock The ADC board takes a high speed clock (100 to 1000 MHz) as input for controlling the data sample rate. The IBOB board clock will be one quarter of the input ADC clock rate.

1 PPS The absolute time reference is based on a GPS referenced 1 Pulse per second synchronization pulse. This pulse is used to measure the offset between the IBOB board internal clock counters and absolute time.

I Channel The astronomical signals are provided to the IBOB in the IF frequency range 500 to 1000 MHz. The input signals are bandwidth limited to avoid aliasing of the input signals.

Q Channel Similar to I Channel.

Summary of FPGA design

Each polarization is sampled at 4 times the system clock speed. So the ADC presents 4 samples for each polarization, each clock cycle. These samples are 8 bits wide, two's complement signed, with the binary point at bit 7. The data follows two paths. One path is through the "pre-block". The "pre-block" has three stages. First it removes a user defined DC bias from the signal. (Perhaps in a future version of this project, this DC bias will be automatically detected and removed.) After the bias has been removed the samples are squared and then accumulated over a user defined number of clock cycles. The output of the "pre-block" is compared to a user defined noise threshold. If the noise threshold is exceeded then an event occurred signal is sent to the "ram-control" block. While data is propagating through the path described above, it also makes its way through a second path. The second path has two stages. The first stage concatenates the 4 parallel 8 bit sample' s into one 32 bit number.

This concatenation is done to store the data in the 32 bit wide RAM locations. After concatenation the data goes into a buffer. The length of the buffer is set to half of the window length. At the time an event is detected, the data coming out of the buffer is the beginning of the event window. When the event occurred signal (described above) arrives at the "ram-control" block, the write enable is asserted and the address counter begins counting. Additional logic in the "ram-control" block keeps the system from capturing a new event until the last event has been read from memory. A future improvement of this project will likely be to fix it so the system will continue to capture new events. This could be done with bigger output RAM blocks and more RAM control logic that would write new event data to higher memory locations. The depth of the buffers is the number of clock cycles that pass between assertion of the buffer's write pin and the buffer's read pin. This timing is handled by the "window-time" block.

A 64 bit counter runs non-stop when the system starts. This counter is used for time correlation. When an event comes the counter value is latched. When a sync pulse comes the counter value is latched.

Linux Software Functionality

This section describes the Linux software associated with the EventCap profile for the IBOB.

IBOB Configuration

Configuring the event cap profile once it has been loaded onto the IBOB is accomplished through the *ecconf* program. *ecconf* is a simple routine that connects via the serial port to the IBOB and sets the registers specified by the user.

Example 1: EventCap configuration

```
Usage:  ecconf [option <value>] ...

acclen <value> Sets the length of the sample accumulator.

bias <value> Set the bias parameter for both polarizations.

bias_i <value> Set the bias parameter for polarization i separately.

bias_q <value> Set the bias parameter for polarization q separately.

noise <value> Set noise threshold for both polarizations.

noise_i <value> Set noise threshold for polarization i separately.

noise_q <value> Set noise threshold for polarization q separately.
```

```
sim <value> Set simulation mode on (1) or off (0).

start Starts event captures.

stop Stops event captures.

window <value> Set the window size (# of data points) to capture.
```

The *acclen* parameter determines how many samples are averaged for threshold testing. A smaller number of samples will increase the sensitivity to small transients (but also to variations in noise).

The *bias* values are 8 bit numbers (with 7 bits of dedicated fractional precision) ranging from -1 to 1. Bias values are used to correct for DC offsets in the incoming signal for detection purposes only. No modification of any kind is done to the data captured (except digitization at the ADC, of course).

The *noise* values are 32 bit numbers (with 14 bits of dedicated fractional precision) that can range from 0 to 262,144.9999. A noise value closer to zero increases the probability of detecting an event (or noise, depending on signal strength). Typical values used with EventCap on the 43m IF observations were between 1 and 10 for these registers.

The *sim* register determines the input of the EventCap design. In simulation mode, the data is read from a set of input BRAMs rather than the ADC, and mock signals or events can be loaded into these RAMs for testing purposes.

The *start* and *stop* registers begin or end threshold detection and capturing of events.

The *window* value sets the number of data points collected per event per polarization.

Note: Currently the window must be set before the start command is given or the event data may become erroneous or improperly centered. Setting the window while start is enabled will do nothing, but if start is disabled and re-enabled, the data will be incorrect (see examples 2, 3, and 4).

Example 2: The good

```
$ ecconf start noise 2 sim 0 window 600
regwrite noise_i 32768 # ecconf displays the commands issued in
regwrite noise_q 32768 # the Tinyshell environment that map to
regwrite window_time 75 # the arguments given by the user
regwrite sim 0
regwrite start 1 # This is ok
```

Example 3: The bad

```
$ ecconf noise 2 sim 0 start
regwrite noise_i 32768
regwrite noise_q 32768
regwrite sim 0
regwrite start 1

$ ecconf window 600 # This won't actually change the output
regwrite window_time 75 # unless you stop and start again
```

Example 4: The ugly

```
$ ecconf noise 2 sim 0 start
regwrite noise_i 32768
regwrite noise_q 32768
regwrite sim 0
regwrite start 1

$ ecconf stop
regwrite start 0

$ ecconf window 600 # This will uncenter the data!
regwrite window_time 75
```

Getting Some Data

Once the IBOB is configured, it is ready to gather event data to transfer off of the Linux machine. This is done using another program, *pickit*. PickIt is a polling program that periodically checks the IBOB's state to see if event data is available. If a transient has been captured, PickIt transfers several timing parameters and the voltage measurements from both polarizations into a data file on the Linux machine via the serial link.

Example 5: PickIt

```
$ pickit -help

Usage:  pickit [-b] [-p] [-s <time>] [-u <time>]

-b Obtain background plots every minute (not precise).

-p Plot output file using GnuPlot.

(Persistent windows, spammy with many events)

-s <time> Set the sleep time between polls.  (seconds)

-u <time> Set the sleep time between polls.  (microseconds)

-help Display this usage information.
```

The *-b* option for pickit enables capturing of "background plots" which contain only ambient noise. These plots can be used to see what kind of RFI may be present in actual transients to differentiate between the interference and true signals.

The *-p* option simply plots the incoming signals using GnuPlot. Note that this uses persistent windows, and frequent periodic sources or high levels of RFI will cause your screen to be coated with graphs.

The *-s* and *-u* options can be used to increase the poll period and decrease system load if the events are expected to be farther apart in time.

The unmodified cycle time of PickIt depends primarily on the number of data points the EventCap profile is acquiring (see above regarding *window* in ecconf and the *window_time* register). At 4800 data points per polarization (6μs of data with an 800MHz ADC clock/200MHz IBOB clock), the cycle time is approximately 1.5 seconds when using the serial link.

Flashing Lights Definition

Table 1: Interpreting the LEDs

	Far Left	Left	Right	Far Right
Top	*flash:* Use to for a quick verification of the correct clock speed.	*event_flash:* Lights when an event has occurred and is waiting to be read.	*thresh_sat_i:* The brightness of this light shows how close polarization I is to the threshold.	*thresh_sat_q:* The brightness of this light shows how close polarization Q is to the threshold.
Bottom	*flash (sync):* Flashes with the sync pulse.		*adc_sat_i:* The brightness of this light shows how close polarization I is to saturating the ADC.	*adc_sat_q:* The brightness of this light shows how close polarization Q is to saturating the ADC.

Project Closeout

The project will be closed out at the end of initial deployment of the system connected to the 43m IF chain. The project team must demonstrate collection of real data from the 43m telescope in both polarizations. The data must be plotted in intensity versus time order and a fourier transformed in four blocks. The blocks are: **A,B** data in each polarization obtained before the event and **C,D** data in each polarization obtained during the event. The event end will be determined by visual inspection of the time series.

REFERENCES

Hankins, T. H.; Ekers, R. D.; O'Sullivan, J. D., (1996), *A search for lunar radio Cerenkov emission from high-energy neutrinos*, *MNRAS*, **283**, 1072.

Langston, G., McCullough, R., Ford, J., Rumberg, B., Brandt, P. (2007) *Event capture α Project Plan NRAO* `http://wiki.nrao.edu/CICADA`

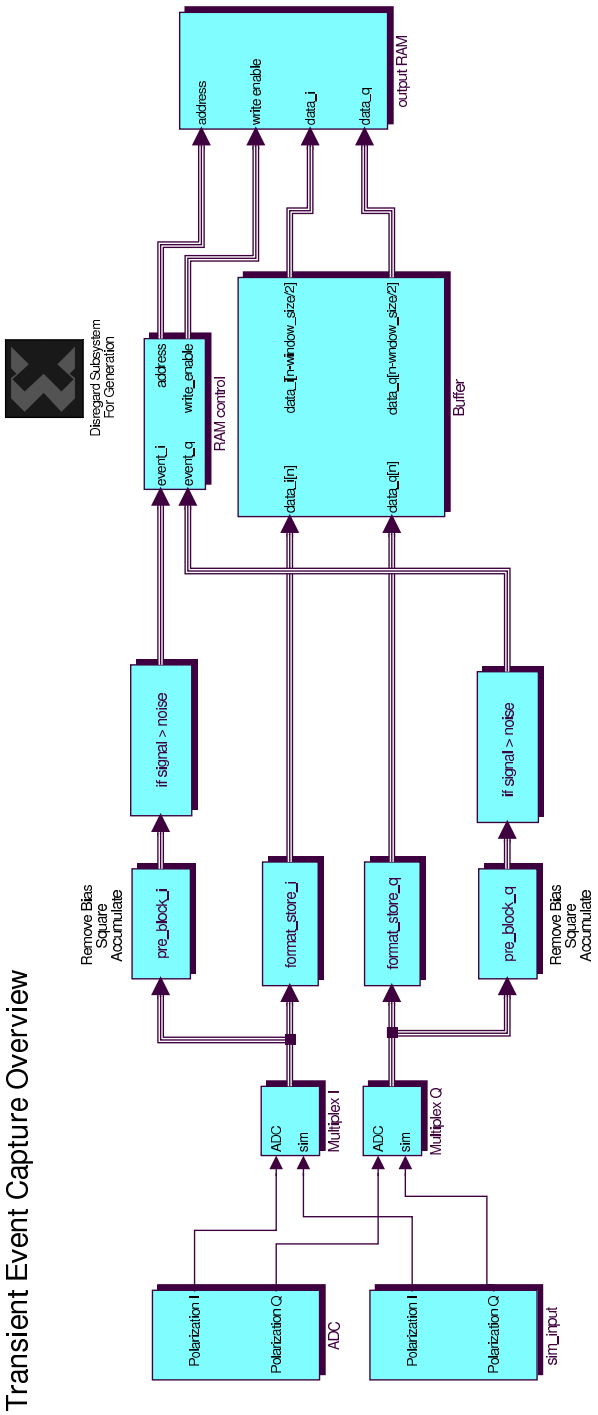

Fig. 1.— Overview design diagram for Event Capture α

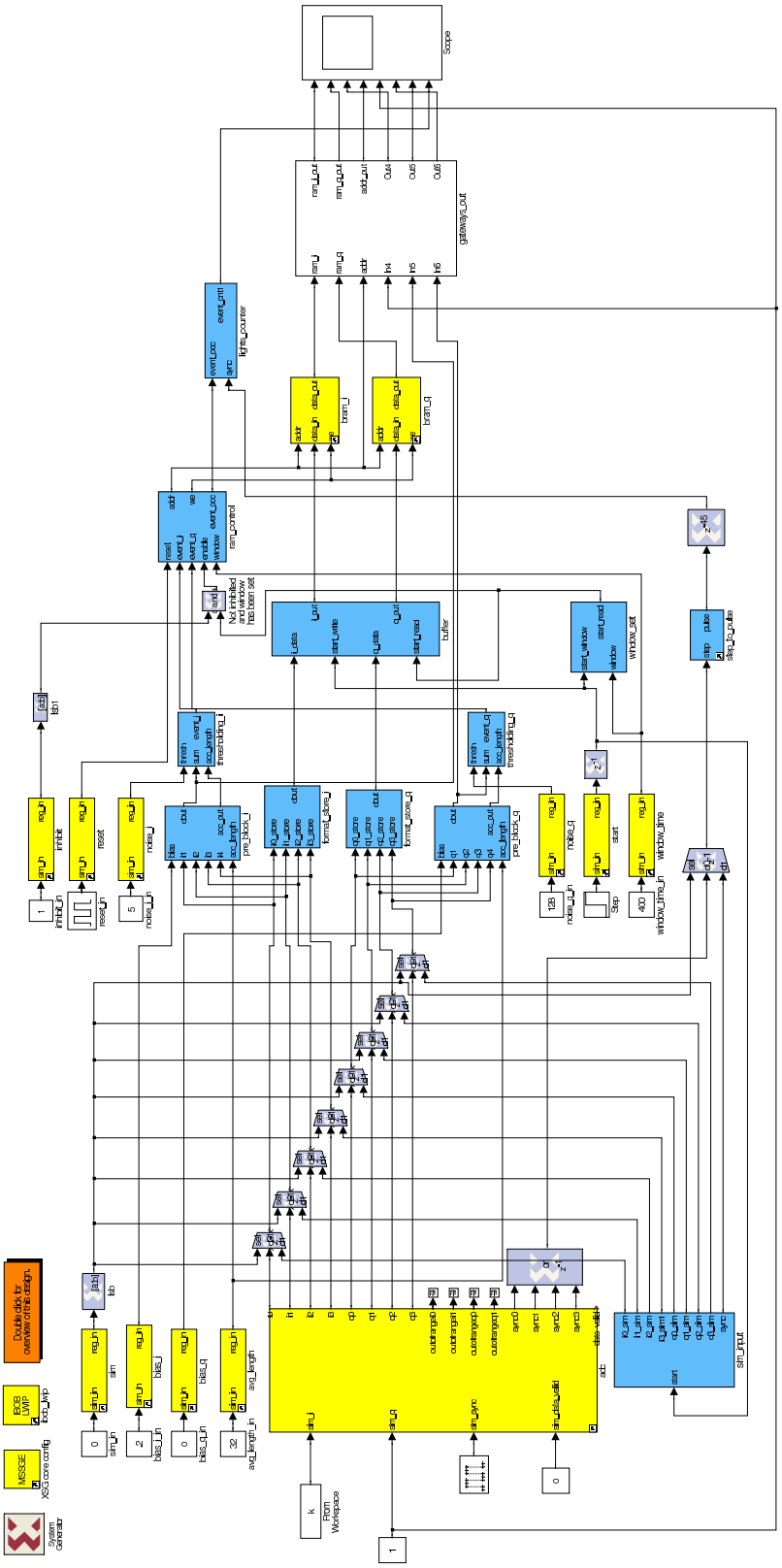

Fig. 2.— Top level design diagram for Event Capture α

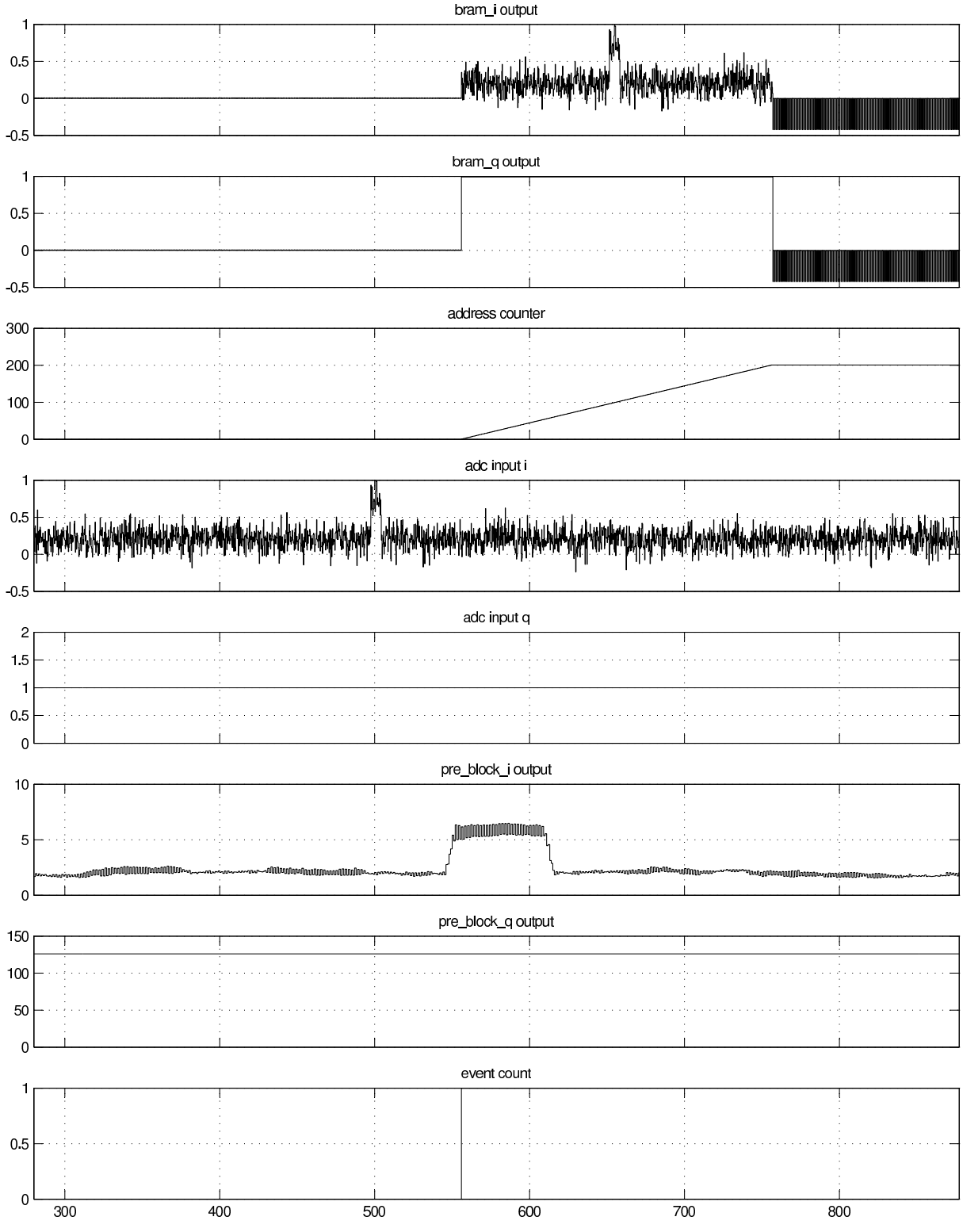

Fig. 3.— Simulation design diagram for Event Capture α

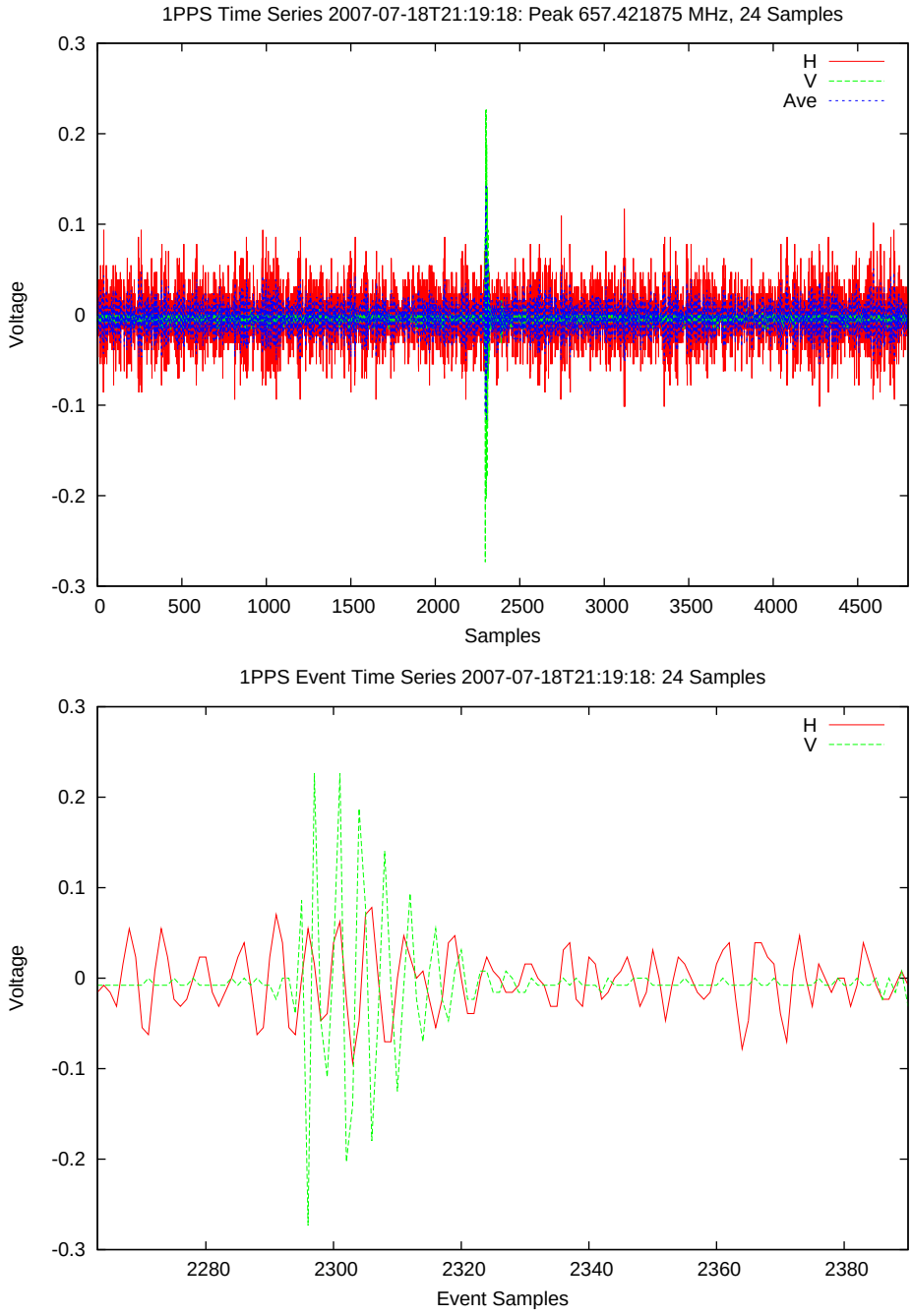

Fig. 4.— **Top:** Plot of voltage versus time of the 1 PPS signal input to the Q input of the ADC when running the Event Capture α design. The I channel of input to the ADC was a band width limited noise source. **Bottom:** Same data as above, except zoomed in on the capture of the one PPS event.

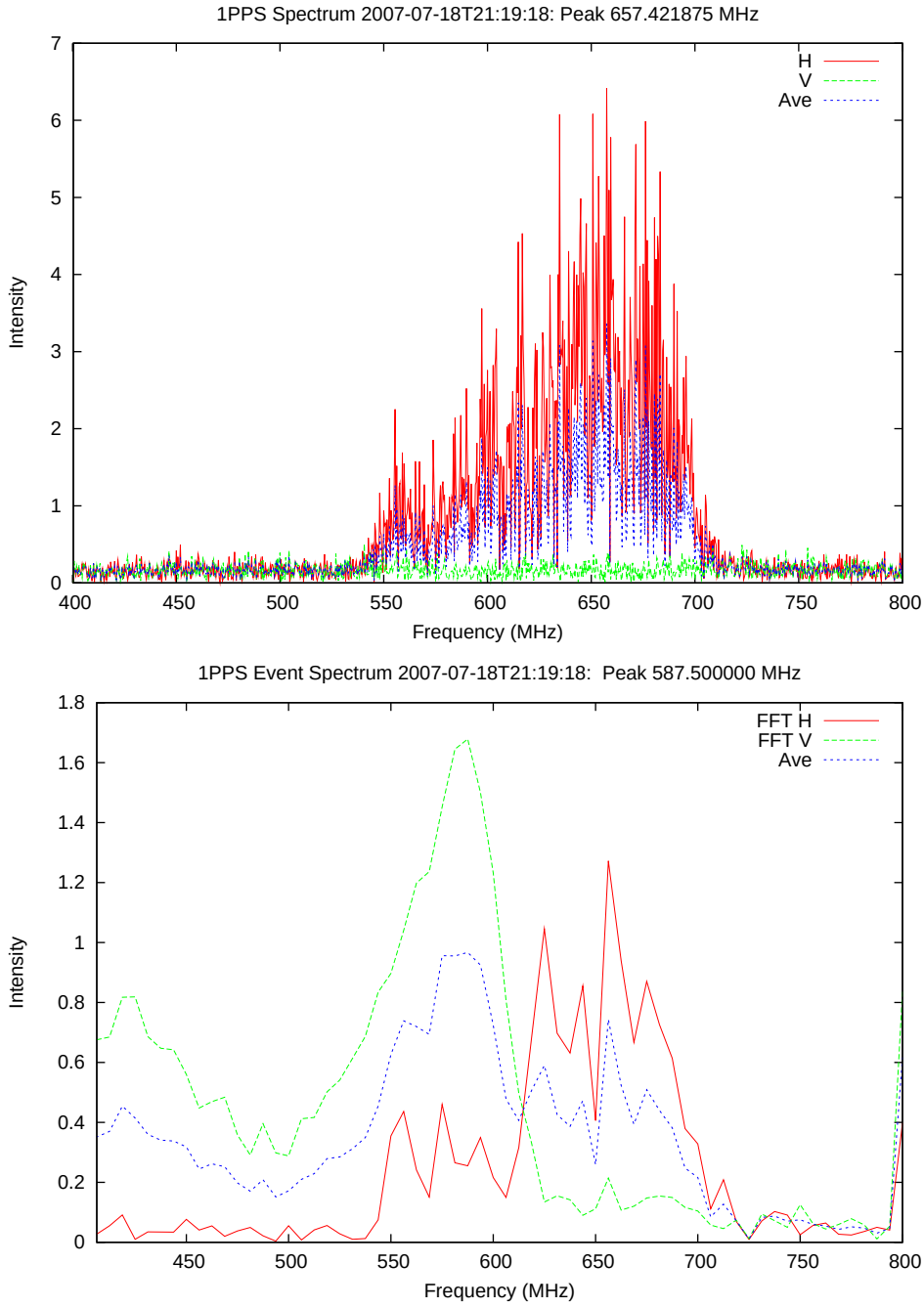

Fig. 5.— **Top:** Plot of intensity versus frequency for the first half of the data, before the event. This spectrum excludes the event and allows deterimination of the spectral charactaristics of the event. **Bottom:** Plot of intensity versus frequency for 128 channels at the time of the transient event. This spectrum is the Fourier transform of the time series of the event data, which is shown in the bottom of the previous plot.

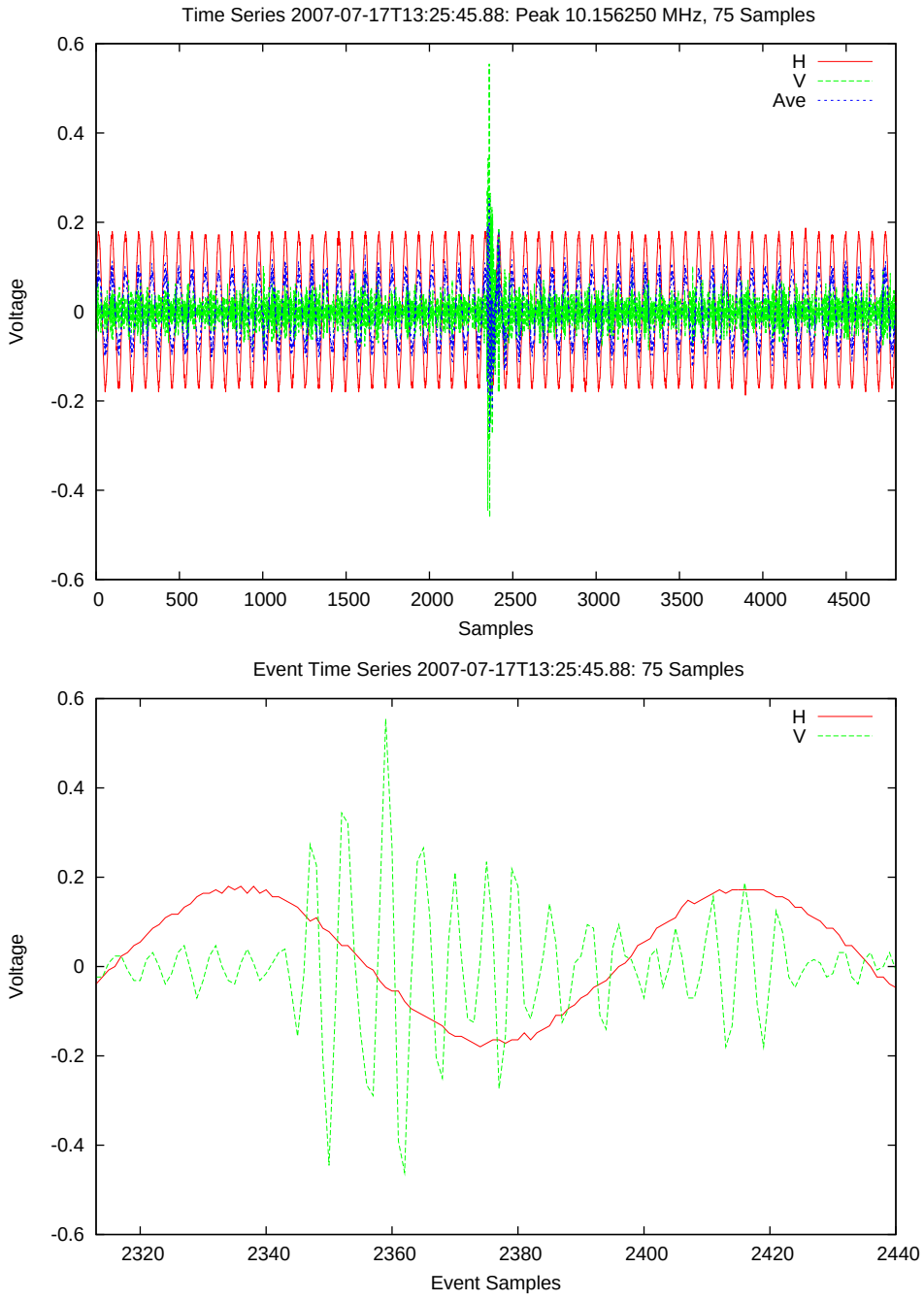

Fig. 6.— **Top:** Plot of voltage versus time of the 10 MHz tone input to the I/H channel of the ADC. The Q/V channel was connected to a noise source. **Bottom:** Same data as above, except zoomed in on the capture of the one PPS event.

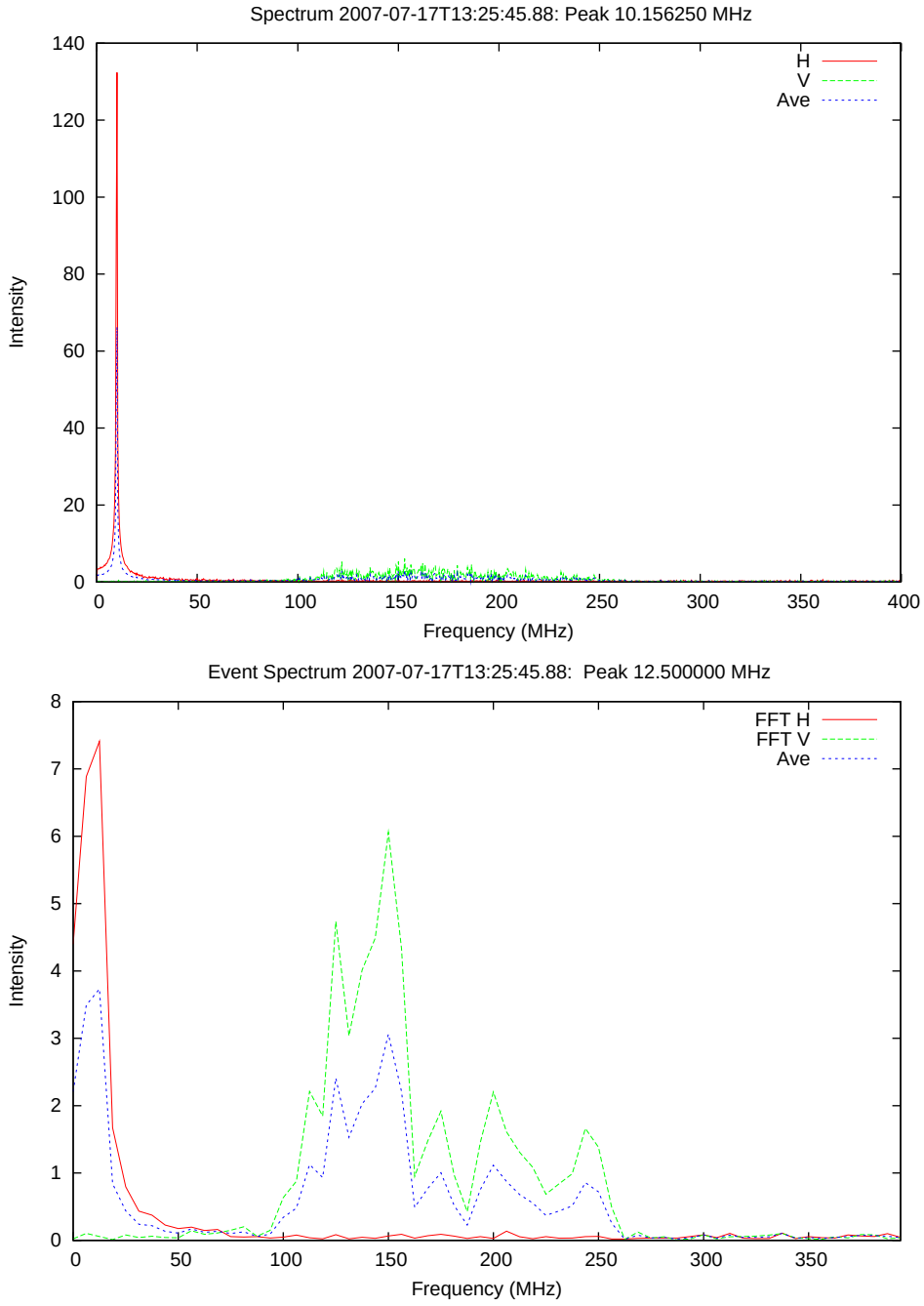

Fig. 7.— **Top:** Plot of intensity versus frequency for the first half of the data, before the event. This spectrum excludes the event and allows deterimination of the spectral charactaristics of the event. **Bottom:** Plot of intensity versus frequency for 128 channels at the time of the transient event. This spectrum is the Fourier transform of the time series of the event data, which is shown in the bottom of the previous plot.

Analysis Program: `eventFft`

The analysis of the captured events is outside the original specification of the Event Capture project. However, in order to determine proper functionality of the design, some analysis was required. Glen Langston wrote a program to plot the dual polarization time sequences and also the Fourier Transform of these sequences.

The program runs in the linux environment and optionally produces plots of the data. `eventFft` always summaries the properties of the captured events. `eventFft` is a C program. Plotting is done by executing external calls to the Linux `gnuplot` program.

The program has internal help information, printed when the program is called without arguments. Ie:

```
->  eventFft
 eventFft: summarize and plot a series of captured events
 eventFft: usage
 eventFft [-a] [-b <freqMHz>] [-e <freqMHz>] [-c <clockMHz>] [-z]
     <eventFileName1> [.. <eventFileNameN>]
 Where
 -a       optionally plot all plots
 -b <freqMHz> optionally set the begin frequency
 -e <freqMHz> optionally set the end frequency
 -f       optionally plot the event spectrum
 -g       optionally plot the event time series
 -i <inDir>   optionally set input directory
 -j       optionally jump byte order of four input data values
 -n <nSamples> optionally set event fft length (ie 16,32,64 or ...)
 -o <inDir>   optionally set output directory (default /tmp)
 -s       optionally plot the bandpass spectrum before event
 -t       optionally plot the entire time series
 -z       optionally compute the average spectrum
           ----- Glen Langston, 2007 July 18
```