Documentation for Event Capture Beta

- Brandon Rumberg
- Patrick Brandt

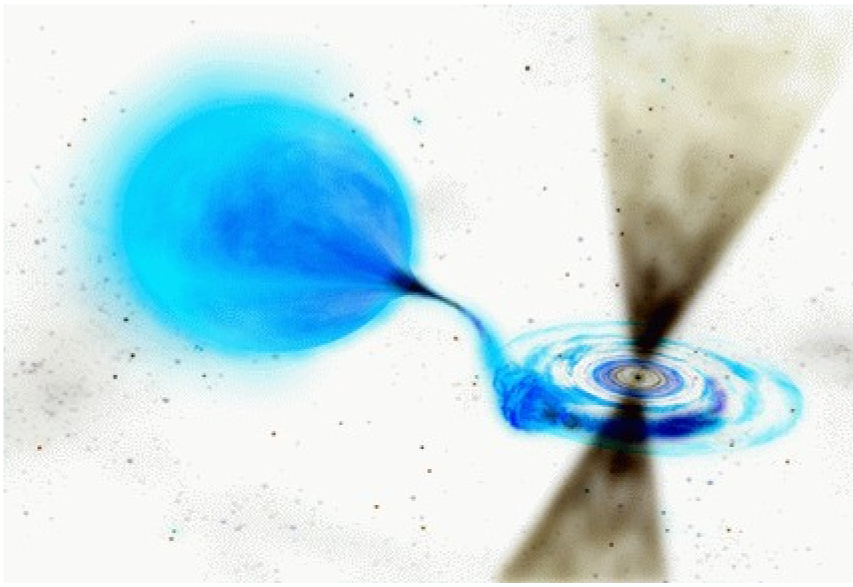

Table of Contents

Illustration Index

About This Document

Event Capture Beta comes in three flavors. First is the single ADC, two input (1GHz sampling) version which is detailed in this document. In addition there exists a 2-ADC, 2-input (2GHz sampling) and a 2-ADC, 4-input (1GHz sampling) version.

All versions of Event Capture Beta have the same functionality so although this document is written for the 1-ADC 2-input mode, it applies to the other modes with minor exceptions.

This document discusses the FPGA portion of the design in a top-down manner. Each subsystem is presented by including a description of the purpose of the subsystem as well as a description of the inputs and outputs.

The purpose of the Event Capture design is to detect short transient events and record them with high time resolution. Motivation and results are described in more detail in notes 2,3, and 4 in the Cicada Notes series.

Top Level

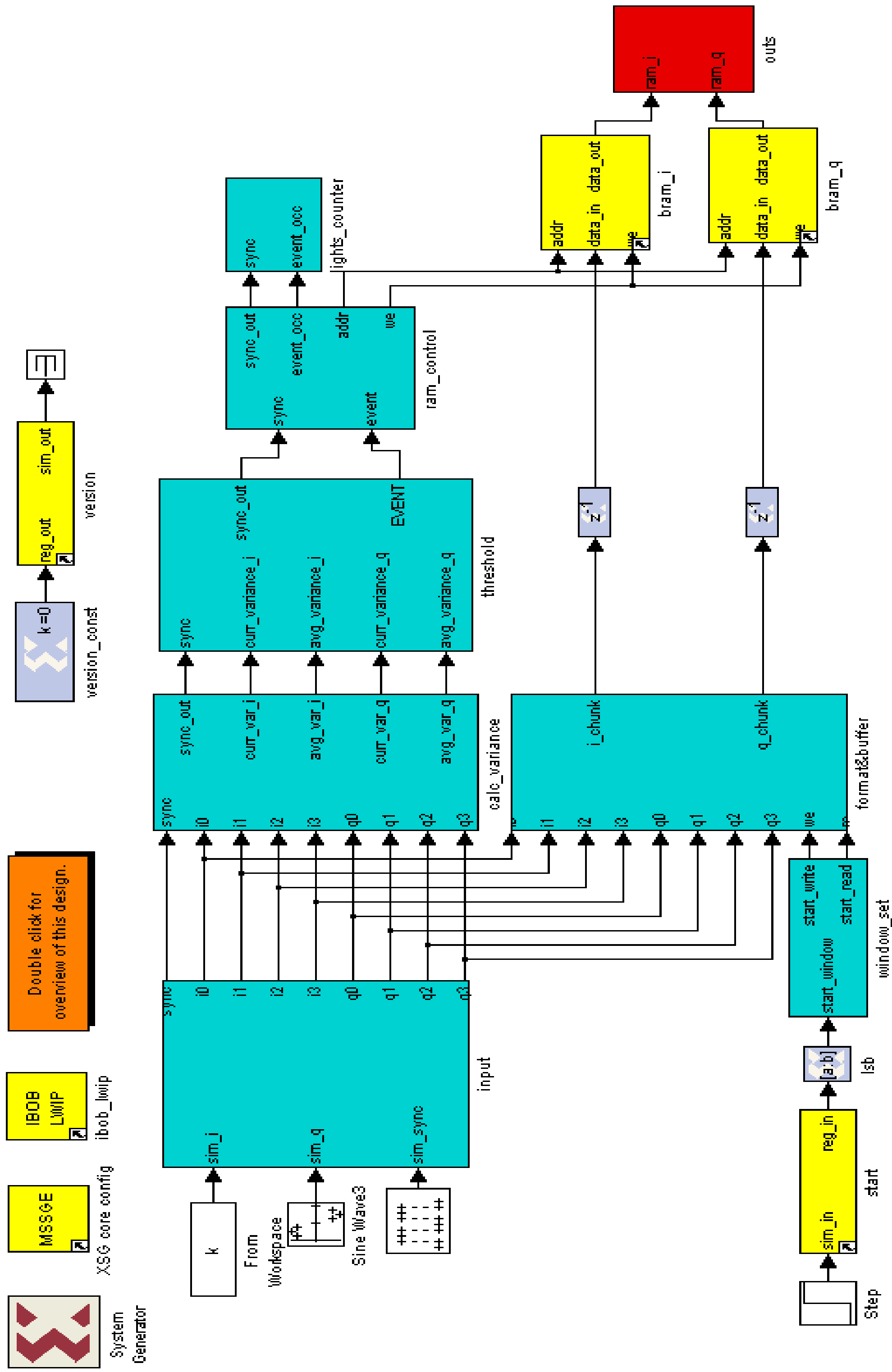

Illustration 1: Event Capture Top Level Diagram

Description:

Event Capture Beta determines when an event has occurred by comparing the current variance (over the past 32 samples) of the input with the average variance over a longer time (user definable). An event is said to occur when the current variance exceeds the average variance scaled by a user defined significance level:

$$Event\ \ occured\ \ if:\ \sigma^2_{current}\ >\ \sigma^2_{average} * significanceFactor$$

Once an event is detected a delayed version of the signal is written into memory.

A window length is defined by the user. The captured event should be centered in the window. The delay for the delayed version of the signal is chosen so the event will be centered and also accounts for the latency in doing the calculations described in the above paragraph:

$$s_{delayed}(t) = s(t + [\frac{windowlength}{2} + latency_{calculation}])$$

Starting at the left of *Illustration 1*, we see that the signals are digitized in the input subsystem after which they follow two paths. The top path is the event detection path. The bottom path delays the signal as described in the above paragraph and concatenates sets of four 8-bit samples to be stored in the 32-bit memory locations.

Each subsystem is described in detail below.

Input Block

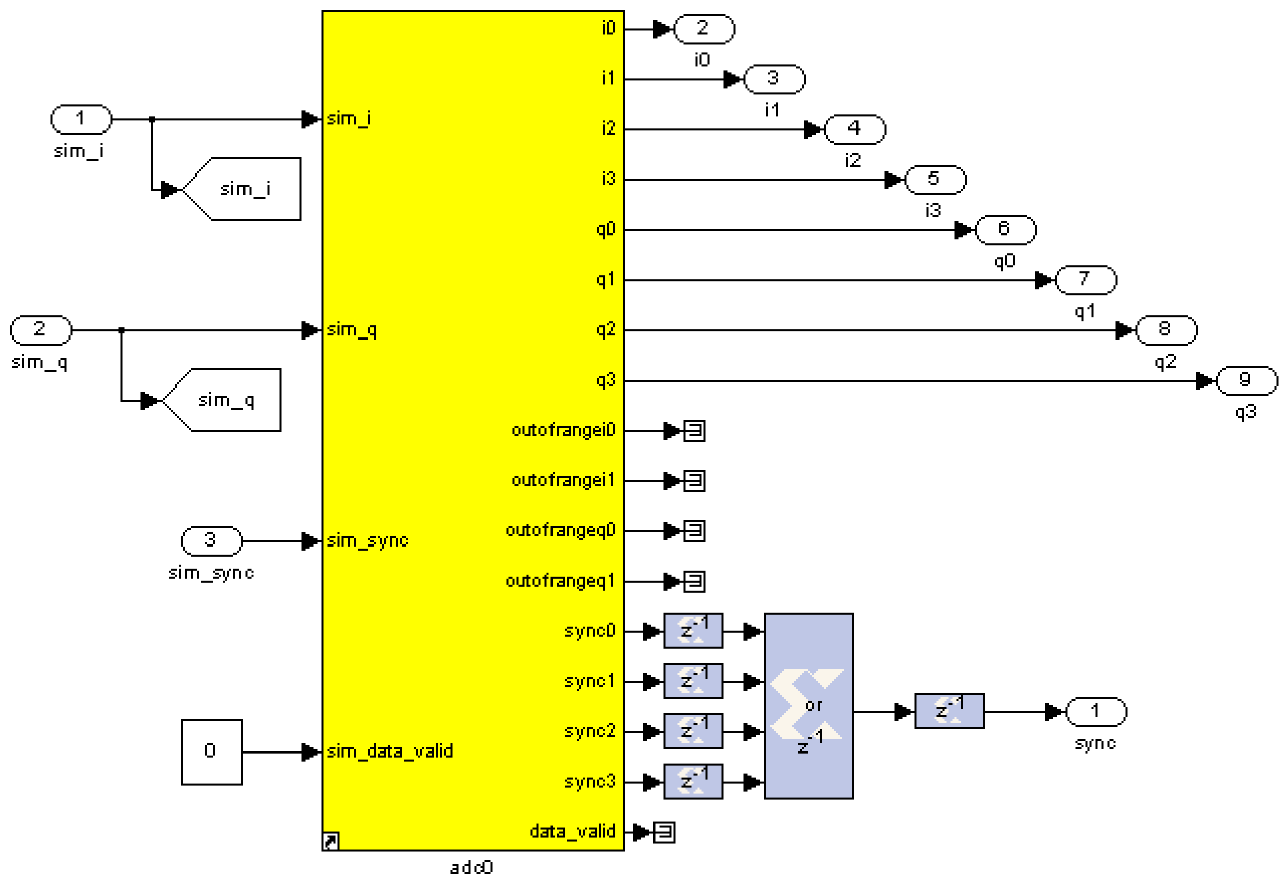

Illustration 2: Input Block Diagram

Description

The *Input* subsystem is responsible for digitizing the incoming signal. Note that the *sync* pulse is down-sampled to the FPGA clock speed.

Inputs

Inputs	Source	Description
sim_i	Top level	Input for polarization I
sim_q	Top level	Input for polarization Q
sim_sync	Top level	1pps input

Outputs

Outputs	**Destination**	**Description**
i0:i3	calc_variance, format&buffer	Sampled input for polarization I
q0:q3	calc_variance, format&buffer	Sampled input for polarization Q
sync	calc_variance	Sampled 1pps

Window Set

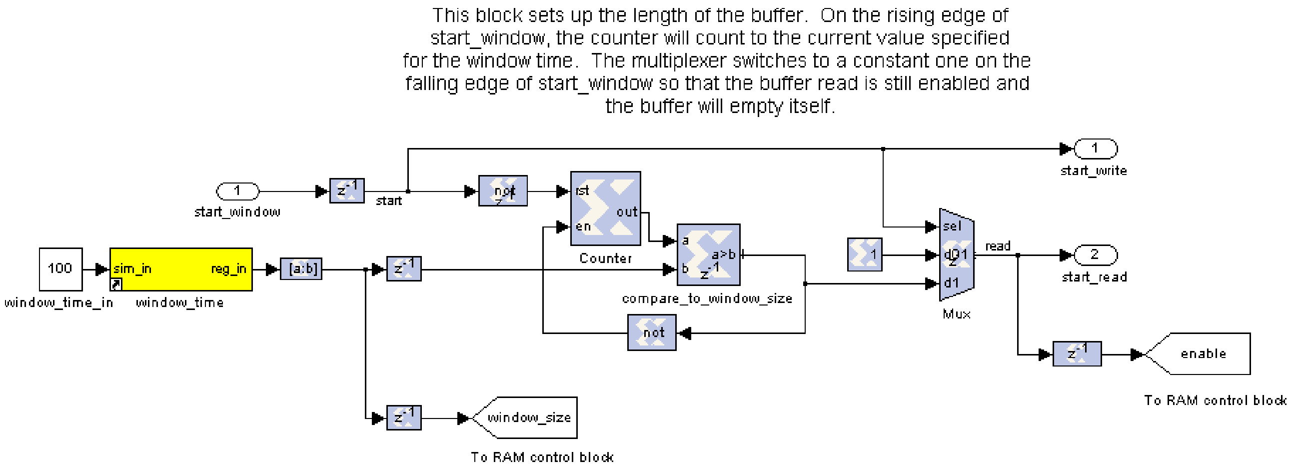

Illustration 3: Window Set Block Diagram

<u>Description</u>

The *Window Set* subsystem initializes the length of the FIFO buffer which is used to delay the signal in order to center the event in the window. On the rising edge of *start_window*, *start_write* is asserted and the counter will count to the value in the *window_time* register. Once this value is reached, *start_read* is asserted.

The multiplexer on the right was added in an attempt to allow the window length to be reset while the system is running. The idea is that the multiplexer switches to a constant value of one on the falling edge of *start_window* so that the buffer read is still enabled and the buffer will empty itself. However this portion of the design does not work so the system must be reset in order to effectively change the window length. This issue is described in Cicada Note #7.

<u>Inputs</u>

Inputs	**Source**	**Description**
window_time	User defined register	The number of clock cycles to capture data on either side of an event
start_window	Top level *start* register	When *start_window* goes high, *start_write* goes high, and the counter counts up to the window size before asserting *start_read*

Outputs

Outputs	**Destination**	**Description**
window_size	RAM control block	Used by address counter to capture the correct number of samples
enable	RAM control block	When asserted, buffer size has been set so it is okay to capture events
start_write	format&buffer block	Begin writing to buffer
start_read	format&buffer block	Begin reading from buffer

Calc Variance

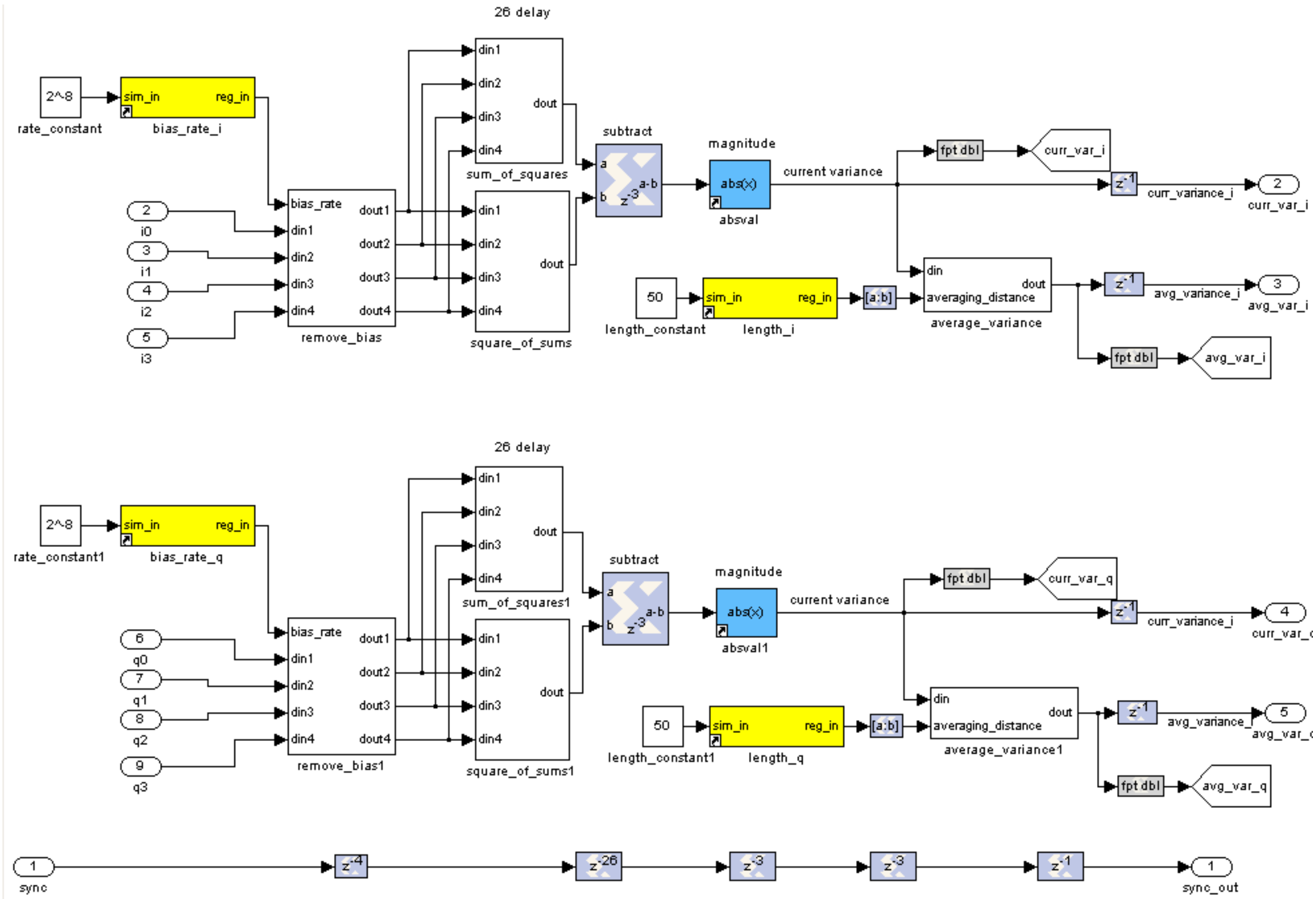

Illustration 4: Calc Variance Block Diagram

<u>Description</u>

The *Calc Variance* subsystem is responsible for calculating both the current and long term variance of the incoming signals. From left to right in Illustration 4, we see that any DC offset is removed from the signal, then the current variance is calculated. The current variance is output from this subsystem and is also fed into the average variance block.

Inputs

Inputs	**Source**	**Description**
i0:i3	ADC	Input for polarization I
q0:q3	ADC	Input for polarization Q
bias_rate_i	User Defined Register	Rate that bias block integrates to the DC offset
bias_rate_q	User Defined Register	Rate that bias block integrates to the DC offset
length_i	User Defined Register	How often samples are used for averaging the variance
length_q	User Defined Register	How often samples are used for averaging the variance

Outputs

Outputs	**Destination**	**Description**
curr_var_i	Threshold	Instantaneous variance of polarization I
avg_var_i	Threshold	Average variance of polarization I
curr_var_q	Threshold	Instantaneous variance of polarization Q
avg_var_q	Threshold	Average variance of polarization Q

Remove Bias

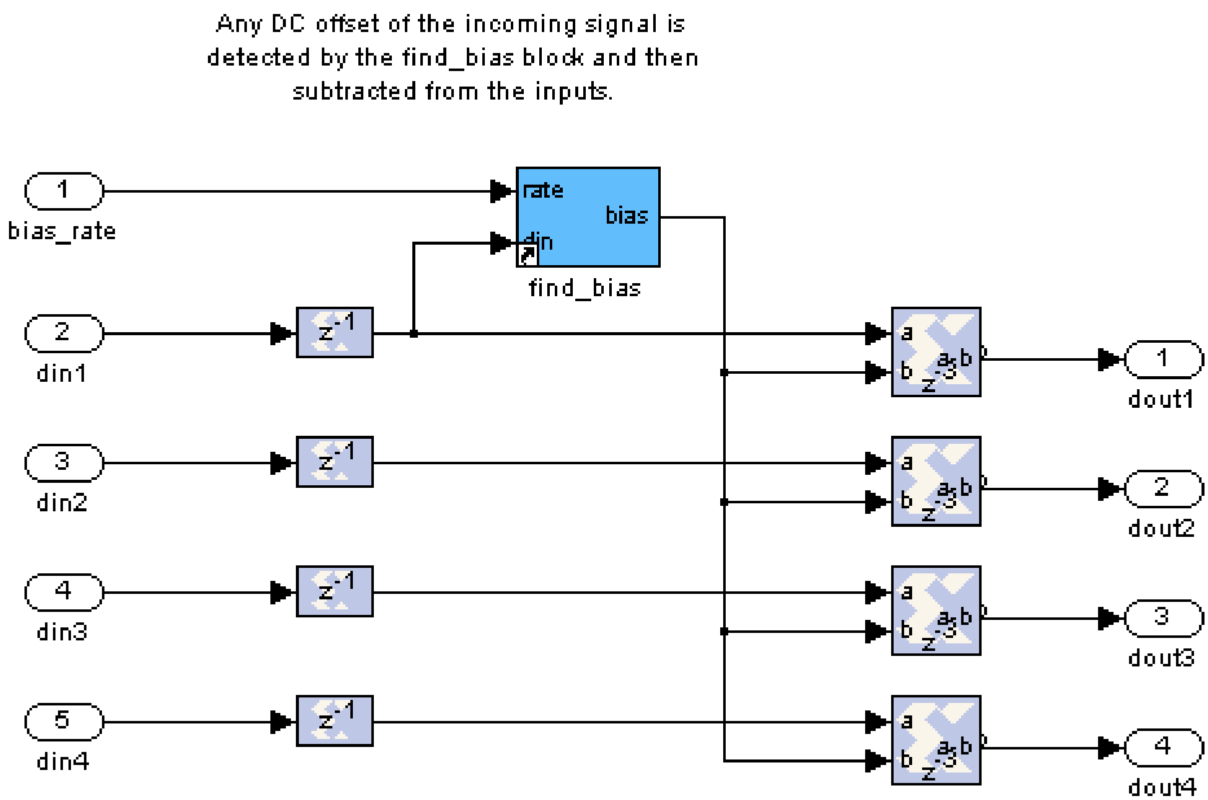

Illustration 5: Remove Bias Block Diagram

<u>Description</u>

Remove Bias does what one would expect from the name. The *find_bias* block integrates to the bias offset of *din1*, this bias is subtracted from each concurrent sample. The algorithm used by *find_bias* is described in the next section.

Find Bias

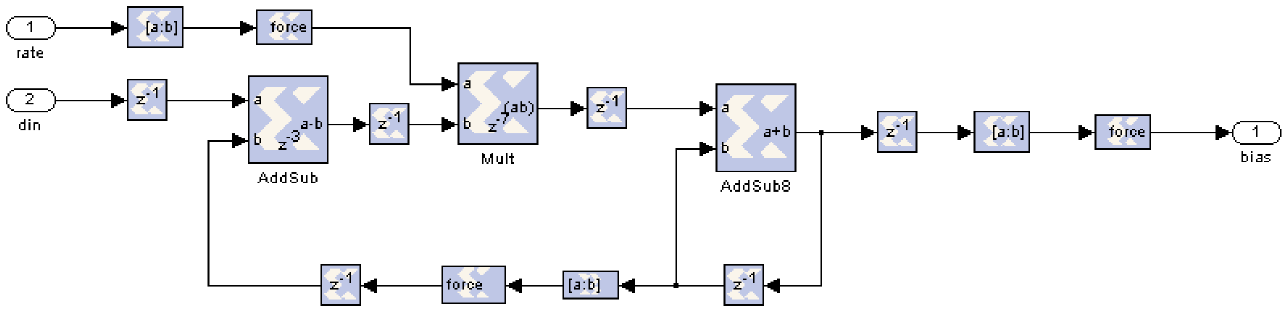

Illustration 6: Find Bias Block Diagram

<u>Description</u>

Find Bias digitally models an integrating filter as described in the Xilinx TechXclusives article Digitally Removing a DC offset. [1]

Sum of Squares

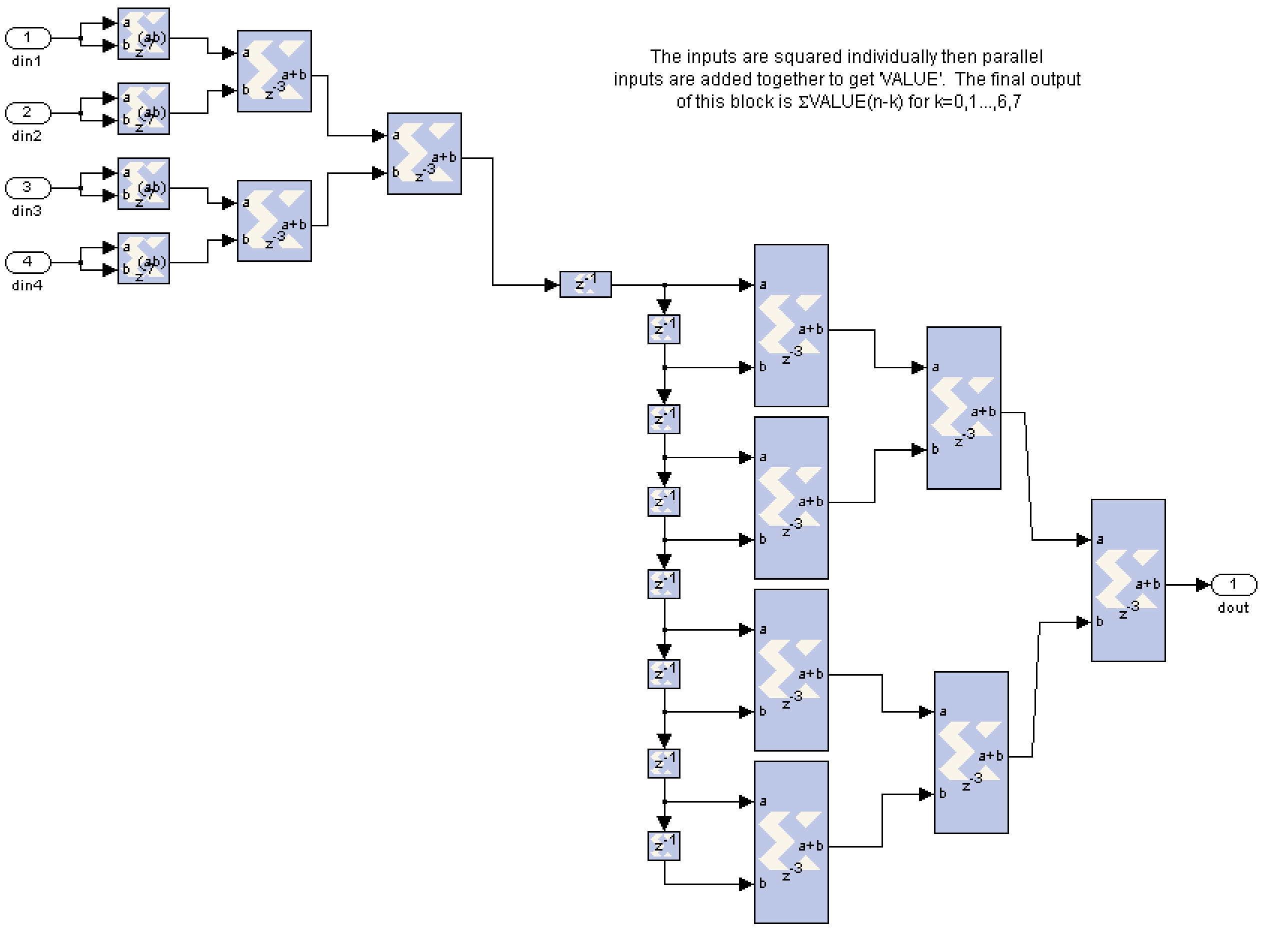

Illustration 7: Sum of Squares Block Diagram

<u>Description</u>

This block sums the squares of the last 32 samples.

Square of Sums

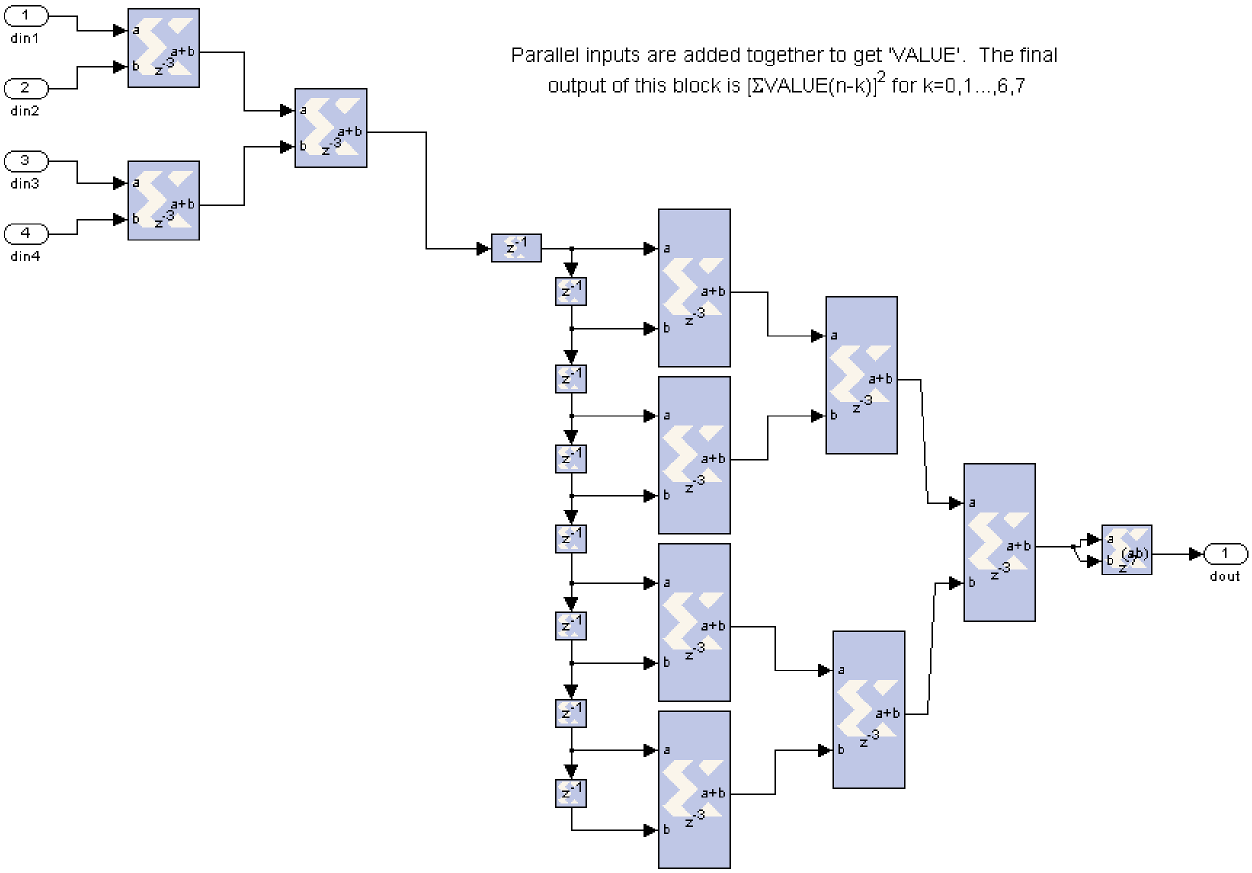

Illustration 8: Square of Sums Block Diagram

Description

This block squares the sum of the last 32 samples.

Average Variance

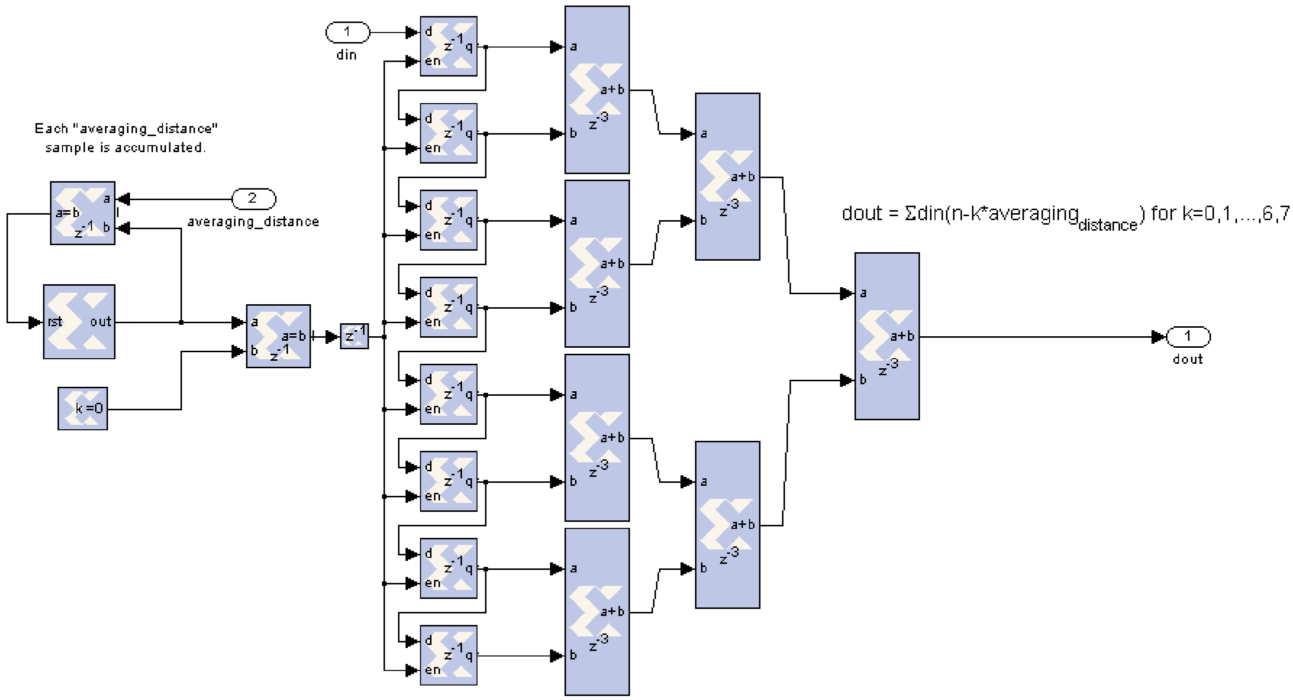

Illustration 9: Average Variance Block Diagram

<u>Description</u>

This block sums the current variance at previous times as represented by this equation:

$$dout[n] = \sum_{k=0}^{7} din[n + k * averagingDistance]$$

Each output is held over *averaging_distance* clock cycles.

Threshold

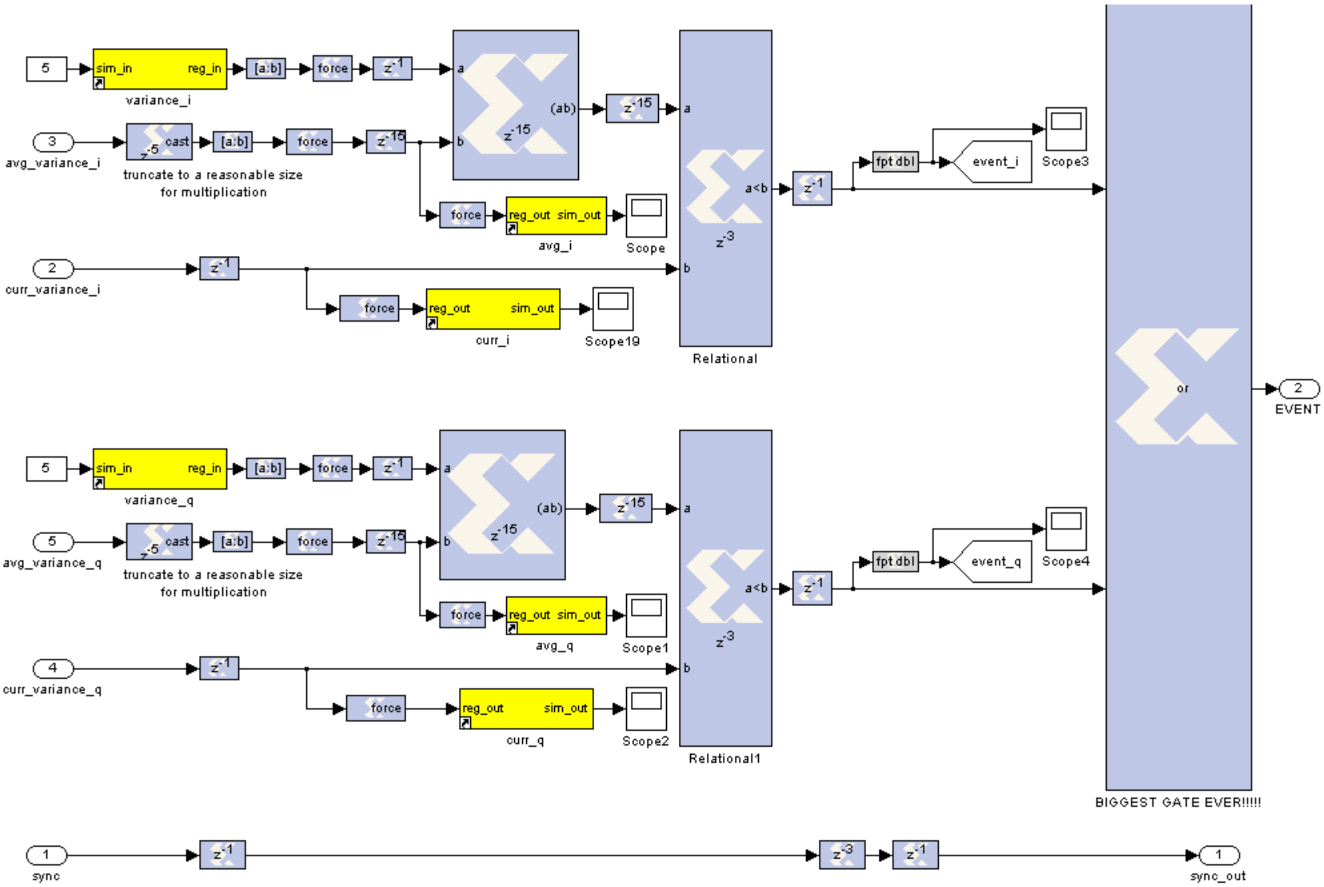

Illustration 10: Threshold Block Diagram

<u>Description</u>

Threshold is responsible for detecting events. The average variance of each channel is multiplied by the user defined significance level in *variance_i* or *variance_q* and compared to the current variance. If the current variance is greater then an event has occurred.

Inputs

Inputs	**Source**	**Description**
avg_variance_i:q	calc_variance	Average variance of respective polarization
curr_variance_i:q	calc_variance	Current variance of respective polarization.
variance_i:q	User defined register	The ratio by which the current variance must exceed the average variance to trigger an event
sync	calc_variance	1pps

Outputs

Outputs	**Destination**	**Description**
avg_i:q	User viewed register	Average variance of respective polarization, viewed as an unsigned number with binary point at zero. (Otherwise number is truncated when run in hardware)
curr_i:q	User viewed register	Current variance of respective polarization, viewed as an unsigned number with binary point at zero. (Otherwise number is truncated when run in hardware)
EVENT	RAM_control block	High when an event is detected in either polarization
event_i:q	top level scopes	High when an event is detected in the respective polarization
sync_out	RAM_control block	1pps

Format&Buffer

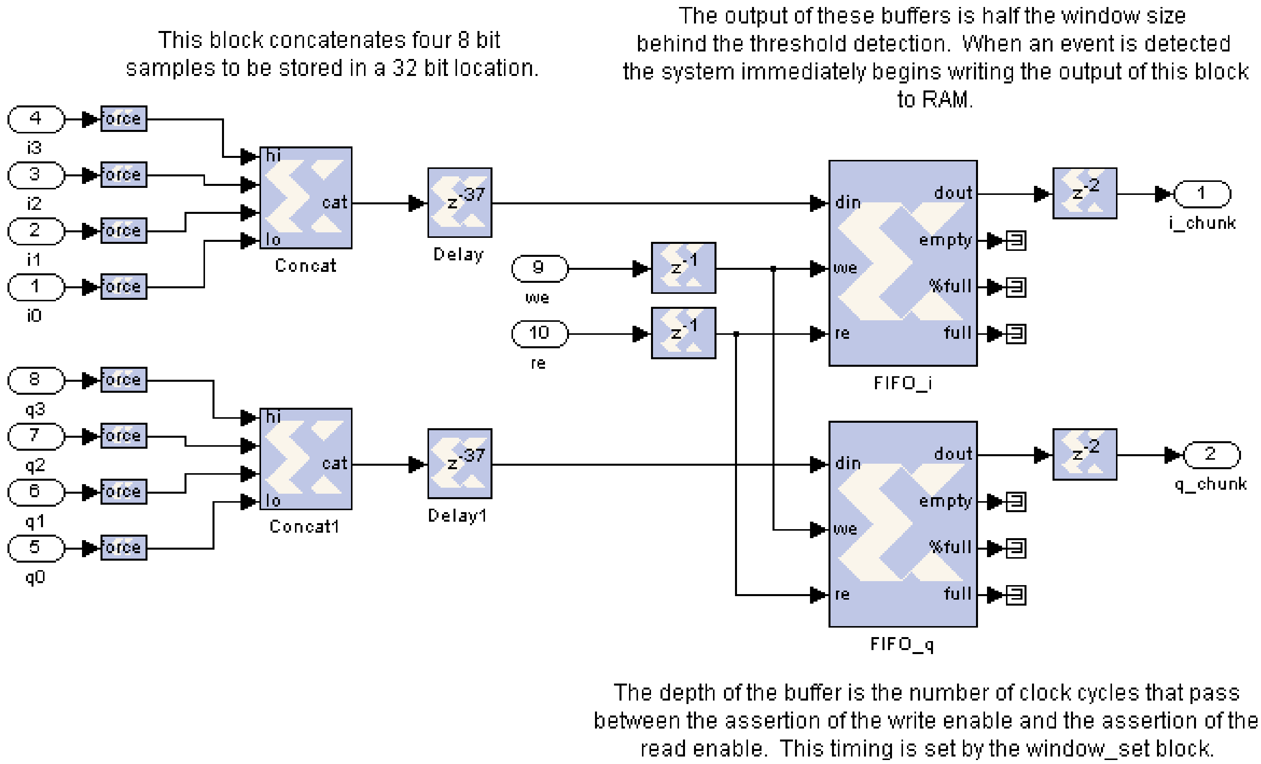

Illustration 11: Format&Buffer Block Diagram

<u>Description</u>

This block concatenates four 8 bit samples to be stored in a 32 bit location. These 32-bit values are then fed into a buffer. The depth of the buffer is the correct length so that the delay caused by running through the buffer is half the window time. The write enable and read enable pins as asserted by the the window_set block to cause the buffer to be the desired depth.

<u>Inputs</u>

Inputs	**Source**	**Description**
i0:i3	Input block	Sampled input for polarization I
q0:q3	Input block	Sampled input for polarization Q
we	window_set block	Begin writing to the buffer
re	window_set block	Begin reading from the buffer

Outputs

Outputs	**Destination**	**Description**
i_chunk	Top level RAM blocks	Sets of four input samples that have been concatenated into one 32 bit word and delayed to be half the window time behind the detection of events
q_chunk	Top level RAM blocks	Sets of four input samples that have been concatenated into one 32 bit word and delayed to be half the window time behind the detection of events

Ram Control

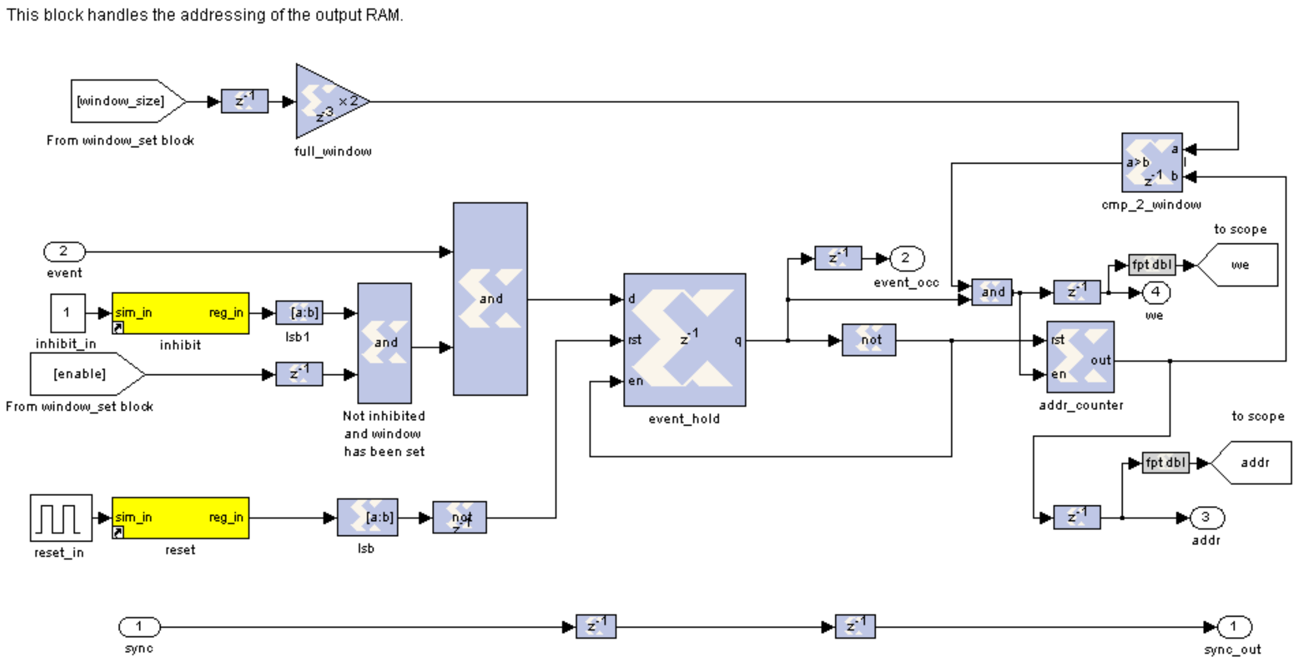

Illustration 12: Ram Control Block Diagram

Description

This block handles the addressing of the output RAM.

Inputs

Inputs	**Source**	**Description**
enable	window_set block	When high, the buffer size has been set so it is okay to capture events
event	threshold	High when an event is detected
inhibit	User defined register	Setting to zero inhibits the capture of events
reset	User defined register	After an event has been captured, this register must be toggled to reset the event_hold register and allow new events to be captured.
sync	threshold	1pps
window_size	window_set block	Number of clock cycles to capture on either side of an event

Outputs

Outputs	**Destination**	**Description**
event_occ	lights_counter block	High when an event has been captured and is ready to be read
we	top level RAM	Write enable for capture RAM block
addr	top level RAM	Addressing for capture RAM blocks
sync_out	lights_counter block	1pps

Lights Counter

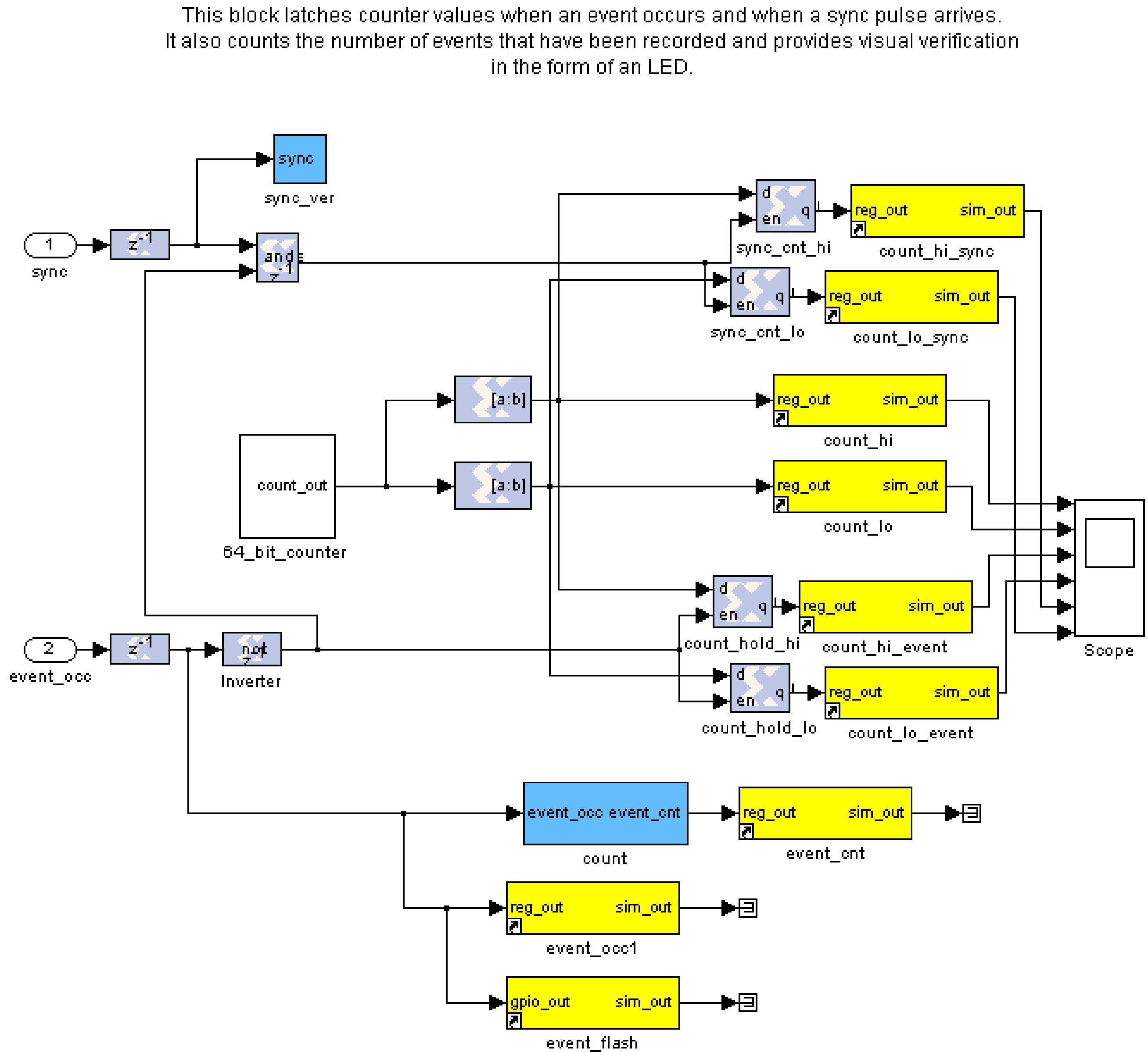

Illustration 13: Lights Counter Block Diagram

<u>Description</u>

This block latches counter values when an event occurs and when a sync pulse arrives. It also counts the number of events that have been recorded and provides visual verification in the form of an LED.

Inputs

Inputs	**Source**	**Description**
event_occ	RAM_control block	High when an event has been captured and is ready to be read
sync	RAM_control block	1pps

Outputs

Outputs	**Destination**	**Description**
count_hi_sync	User read register	Top 32 bits of the system counter at the time of the last sync pulse before an event
count_lo_sync	User read register	Bottom 32 bits of the system counter at the time of the last sync pulse before an event
count_hi	User read register	Top 32 bits of the system counter
count_lo	User read register	Bottom 32 bits of the system counter
count_hi_event	User read register	Top 32 bits of the system counter at the time of an event
count_lo_event	User read register	Bottom 32 bits of the system counter at the time of an event
event_cnt	User read register	Number of events that have been captured since the system powered on
event_occ1	User read register	High when an event has been captured and is ready to be read
event_flash	User read register	LED that lights to show when an event has been captured

References

[1] Chapman, Ken. "Digitally Removing a DC offset." Xilinx TechXclusives.
http://www.xilinx.com/xlnx/xweb/xil_tx_display.jsp?iLanguageID=1&category=&sGlobalNavPick=&sSecondaryNavPick=&multPartNum=1&sTechX_ID=kc_dig_offset

Cicada Note Series
https://wikio.nrao.edu/bin/view/CICADA/CicadaNotes