PROGRAMMING THE CORNELL CMS LEVEL1 TRIGGER FPGA

\\samba\accuser\cesrdaq\hardware\CMS\_Trigger\Docs\FPGA\_Programming\_v1.docx

This manual will be updated routinely. Old versions will be saved in the same area.

7/1/2013 3:09 PM

Contents

[To Do 1](#_Toc359588415)

[Revision History 1](#_Toc359588416)

[V1.1 1](#_Toc359588417)

[NOTES 2](#_Toc359588418)

[DATA FORMAT 2](#_Toc359588419)

[ADDRESSING 2](#_Toc359588420)

[ADDRESS ENCODING: STUB PROCESSING 2](#_Toc359588421)

[ADDRESS ENCODING: TRACKLET MATCHING 3](#_Toc359588422)

[ADDRESS ENCODING: 3rd segment 3](#_Toc359588423)

[ADDRESS ENCODING: 4th segment 3](#_Toc359588424)

[STUB PROCESSING:OVERVIEW 4](#_Toc359588425)

[STUB PROCESSING: DATA STRUCTURES 4](#_Toc359588426)

[Output of “append physical coordinates” and input to “search for tracklets” 4](#_Toc359588427)

[STUB PROCESSING: PROGRAMMABLE MEMORY SPACE 5](#_Toc359588428)

[STUB PROCESSING: PROGRAMMABLE REGISTER SPACE 6](#_Toc359588429)

# To Do

1. Add programming info related to testing. Define addresses for accessing intermediate data buffers and fifos.

# Revision History

## V1.1

1. Initial version for testing basic concepts on the GLIB board.

# NOTES

1. For information related to the physics of this project, refer to the “L1 Track Finding” document by Anders Ryd.

2. Numerical values in this document can be specified in the following formats:

a) Hexadecimal – A ‘c’ style syntax like: 0x12f6 0x0a

b) Binary – A ‘verilog’ style syntax like: 3’b010 5’b01001 6’b10xx10.

It begins with the number of bits, the ‘b, then the bit pattern. ‘x’ means “don’t care”.

c) Signed Decimal – Always preceded by either a plus ‘+’ or a minus ‘-‘

d) Unsigned Decimal - Never preceded by a plus ‘+’ nor a minus ‘-‘

# DATA FORMAT

Data is either *signed* or *unsigned*. Signed data is in 2’s-complement form and unsigned data is straight binary. An 8-bit signed number will have a range from -128 to +127, while an 8-bit unsigned number will have a range from 0 to 255.

Some registers and memory blocks hold a single piece of data, while others hold multiple pieces of data that are packed into a *structure*. The programming instructions for each register or memory block will contain details about whether the data is signed or unsigned, whether it is a single word or a structure, and whether or not sign-extension to a larger word length occurs when reading.

# ADDRESSING

All FPGA registers and memory blocks are mapped in the GLIB “ipb” address space. All accesses involve 32-bit data. The LSB of any address is the address of a 32-bit entity. Access to 8-bit bytes and 16-bit words is not supported.

The “ipb” address space uses 30 bits of the 32-bit range from 0x4000,0000 thru 0x7fff,ffff. The 32-bit address is broken up in different ways in different processing parts of the FPGA.

bit use comment

31:30 always 2’b01 Access user “ipb” space

29:28 segment Access 1 of 4 FPGA code segments

27:0 segment-specific

Registers are sparsely spaced and do not have sequential addresses. There are gaps. However, for high-level coding efficiency, registers can be read or written in contiguous blocks without causing errors. When writing to undefined addresses, the data will simply be discarded. When reading from undefined addresses, the data is undefined.

To aid in programming, a fixed bit mask is defined for all registers, and for the first location of all memory blocks. Something generally needs to be ‘OR’ed with the mask to create a specific address. For instance, anything in the “stub processing” segment needs to have a module number be specified. The mask takes into account all of the unchanging aspects of addressing.

The high-level programmer generally need not be concerned about the fixed bit fields in an address, like “func[19:16]=4’b0100”, “reg\_mem[15]=1’b0”, or “mem\_base[14:12]=3’b000”. They are used internally to the FPGA to simplify addressing. The programmer only needs to use the bit mask.

## ADDRESS ENCODING: STUB PROCESSING

The first segment handles stub data coming from the detector and finds tracklets and track parameters.

bit use comment

31:30 always 2’b01 select user “ipb” space

29:28 always 2’b00 select the ‘stub processing’ segment

27:20 module selects the circuitry associated with 1 of 128 detector modules

19:16 function selects a function within a segment, like ‘tracklet search’ in ‘stub processing’

15:15 reg\_mem selects register space if =1 or memory space if =0

14:0 offset local address of a specific register or memory location

To address a memory table for a specific module, use:

ADR = (mask | (module\_num << 20)) + mem\_loc

To address a register for a specific module, use:

ADR = (mask | (module\_num << 20))

The functions in addr[19:16] are defined in ‘stub\_processing\_top.v’.

## ADDRESS ENCODING: TRACKLET MATCHING

The second segment receives tracklet projection data coming from the stub processing segment and looks for stubs near the projected location.

bit use comment

31:30 always 2’b01 select user “ipb” space

29:28 always 2’b01 select the ‘tracklet matching’ segment

## ADDRESS ENCODING: 3rd segment

The third segment …...

bit use comment

31:30 always 2’b01 select user “ipb” space

29:28 always 2’b10 select the 3rd segment

## ADDRESS ENCODING: 4th segment

The fourth segment …...

bit use comment

31:30 always 2’b01 select user “ipb” space

29:28 always 2’b11 select the 4th segment

# STUB PROCESSING:OVERVIEW

Stub processing is a done in a sequence of functions that starts with receiving stub data from the detector and ends with transmitting track parameters to the tracklet matching segment. The specific functions are:

Receive stub data from the FED

Convert stub indexing

Append physical coordinates

Search for tracklets

Calculate track parameters from tracklets

Calculate track projections to other layers

Transmit track parameters

There is one stub processing block for each inner layer detector module supported by this FPGA. Each block handles one, two, or three outer layer detector modules, depending on where along a rod the inner layer module is located. Inner layer modules nearer the center of the detector where Z=0 will have three outer layer modules. Those further away from Z=0 will generally have two outer layer modules. At the very end of the rod, some inner layer modules will only have one outer layer module.

Generally, the outer layer module at the same Z distance as the inner layer module will be called the ‘a’ module. The next one at a larger Z will be called the ‘b’ module. If one at a smaller Z is required, it is the ‘c’ module. If the inner layer module is at a large enough Z distance so that tracks passing through it will never pass through the outer layer module at the same Z distance, then the first outer layer module that tracks can pass through is the ‘a’ module. It would be followed by the ‘b’ module. Registers and memories related to outer layer modules are specified by appending ‘\_a’, ‘\_b’, or ‘\_c’.

The module numbers are local to each FPGA. A table will be provided to look up the local module number of any module in a sector.

To address something in the stub processing segment for a specific module, use:

ADR = mask | (module\_num << 20)

## STUB PROCESSING: DATA STRUCTURES

This section describes the structure of the data at the input to the stub processing segment, internally between functions, and the final output. The programmer generally does not have access to this data, except when using special versions of the FPGA for testing and development.

### Output of “append physical coordinates” and input to “search for tracklets”

Data is passed in dual-port memories and a fifo.

There is a dual-port memory for the inner layer module, and one for each of the outer layer modules that go along with this inner layer module. The memories are configured as 512 64-bit words. The memory space is divided into buffers. Each buffer is 16 words deep and holds all of the stub data for a single crossing. There are enough buffers to hold the data from 32 crossings. The crossing number is used to select a buffer, and a counter is used to select a stub with a buffer.

The format of the stub address is:

bit use comment

8:4 crossing\_num[4:0] select the buffer for this crossing

3:0 stub[3:0] select a stub for this crossing

The format of the stub data is:

bit use comment

63:48 Z\_idx unsigned logical Z index along a module

47:32 X\_idx unsigned logical X index across a module

31:16 Z\_phy signed physical Z coordinate along a rod in units of 0.0125 cm

15:0 X\_phy signed physical X coordinate across a sector in units of 0.00125 cm

The index is just on a module. The physical coordinates have a more global aspect.

In the Z direction, there are 80 pixels per module, so a 7-bit index is needed. A 16-bit signed number has a range of +/- 32,767. Using a resolution of 0.0125 cm, the physical Z dimension can span +/- 459 cm.

In the X direction, there are 1000 pixels per module, so a 10-bit index is needed. For the physical coordinate, we define X=0 to be on a line that bisects a detector sector, and we assume 15 degree sectors. A 15-bit signed number has a range of +/- 16,383. Using a resolution of 0.00125 cm, the physical X dimension can span +/- 20 cm. At a 1 meter radius, this is +/- 11 degrees.

There may not be any need to pass along the index values for the outer layer modules. For now, they are kept.

The fifo is used to indicate when all of the data for a crossing has been put into a memory buffer, and to pass the number of stubs from each module. We may be receiving for a crossing from one module while data for a previous crossing is still coming in from another module. The fifo is written when all data is in from all modules.

The format of the stub count data is:

bit use comment

31:20 crossing\_num unsigned 12-bit (0 to 4095) value of the crossing number

19:15 c\_cnt unsigned stub count for outer layer module ‘c’ (if ‘c’ exists)

14:10 b\_cnt unsigned stub count for outer layer module ‘b’ (if ‘b’ exists)

9:5 a\_cnt unsigned stub count for outer layer module ‘a’

4:0 in\_cnt unsigned stub count for the inner layer module

Only the 5 LSBs of the crossing number are used to address buffers to search for tracklets, but a larger number is passed along with any tracklets that are found. It may be fewer than 12, however. At 40 MHz, an 8 bit number covers 6.4 usec.

## STUB PROCESSING: PROGRAMMABLE MEMORY SPACE

stub tagging function:

in\_stub\_mem mask=0x40020000

func[19:16]=4’b0010, reg\_mem[15]=1’b0, mem\_base[14:12]=3’b000

mem\_loc = 0..1023

out\_stub\_mem\_a mask=0x40021000

func[19:16]=4’b0010, reg\_mem[15]=1’b0, mem\_base[14:12]=3’b001

mem\_loc = 0..1023

out\_stub\_mem\_b mask=0x40022000

func[19:16]=4’b0010, reg\_mem[15]=1’b0, mem\_base[14:12]=3’b010

mem\_loc = 0..1023

out\_stub\_mem\_c mask=0x40023000

func[19:16]=4’b0010, reg\_mem[15]=1’b0, mem\_base[14:12]=3’b011

mem\_loc = 0..1023

Programmable access to these memories is temporary. It is provided for testing purposes. The description of the memory contents is above in the section “Output of “append physical coordinates” and input to “search for tracklets””. Note that the memories are 32-bit from the programming side and 64-bit on the processing side, so two write operations are needed to insert one stub.

stub\_cnt\_fifo mask=0x40028000

func[19:16]=4’b0010, reg\_mem[15]=1’b1, reg\_loc[2:0]=3’b000

Programmable access to this fifo is temporary. It is provided for testing purposes. The description of the fifo contents is above in the section “Output of “append physical coordinates” and input to “search for tracklets””. This is a write-only register.

tracklet search function:

Window\_X\_LUT mask=0x40040000

func[19:16]=4’b0100, reg\_mem[15]=1’b0, mem\_base[14:12]=3’b000

mem\_loc = 0..1023

Window\_Z\_LUT mask=0x40041000

func[19:16]=4’b0100, reg\_mem[15]=1’b0, mem\_base[14:12]=3’b001

mem\_loc = 0..127

The algorithm for identifying tracklets involves retrieving a stub from the inner layer module, looking up the boundaries of a window that define how far away a stub from an outer layer module can be, then checking all of the outer layer stubs to see if they fall within the window. The search window is always specified as a 'delta' from the current location. For (z), we need to cover the most space at high eta values (eta = 2.5). For (x), we need to cover the most space for low momentum tracks in the outer-most layer. A calculation should be performed, but for now let us assume that a 12-bit signed number is adequate for all cases.

The table entries in the (x) table define CW and CCW boundaries that encompass the region where a low momentum track passing through the inner layer must pass through the outer layer. The table entries in the (z) table define near and far boundaries that encompass the region where a track originating at the IP and passing through the inner layer must pass through the outer layer.

The format of the (x) table data is:

bit use comment

27:16 CCW edge signed 12-bit (+/- 2048) distance from inner layer pixel, units = 0.00125 cm

11:0 CW edge signed 12-bit (+/- 2048) distance from inner layer pixel, units = 0.00125 cm

The format of the (z) table data is:

bit use comment

28:16 far edge signed 13-bit (+/- 4095) distance from inner layer pixel, units = 0.0125 cm

12:0 near edge signed 13-bit (+/- 4095) distance from inner layer pixel, units = 0.0125 cm

The tables are addresses by using the logical index of the pixel on the inner layer, rather than the physical position. The calculations are carried out using the physical positions. The (x) table has 1024 entries and the (z) table has 128.

track parameters function:

Lookup tables to help with the math

## STUB PROCESSING: PROGRAMMABLE REGISTER SPACE

stub tagging function:

None

tracklet search function:

None

track parameters function:

DeltaOverR\_a mask=0x40058000

func[19:16]=4’b0101, reg\_mem[15]=1’b1, reg\_loc[4:0]=5’b00000

DeltaOverR\_b mask=0x40058008

func[19:16]=4’b0101, reg\_mem[15]=1’b1, reg\_loc[4:0]=5’b01000

DeltaOverR\_c mask=0x40058010

func[19:16]=4’b0101, reg\_mem[15]=1’b1, reg\_loc[4:0]=5’b10000

This register is an 18-bit signed number, but it is always positive. It can be written or read. The contents are the result of dividing the distance from the inner layer module to the outer layer module by the radius of the inner layer module.

DeltaOverDX \_a mask=0x40058001

func[19:16]=4’b0101, reg\_mem[15]=1’b1, reg\_loc[4:0]=5’b00001

DeltaOverDX \_b mask=0x40058009

func[19:16]=4’b0101, reg\_mem[15]=1’b1, reg\_loc[4:0]=5’b01001

DeltaOverDX \_c mask=0x40058011

func[19:16]=4’b0101, reg\_mem[15]=1’b1, reg\_loc[4:0]=5’b10001

This register is an 18-bit signed number, but it is always positive. It can be written or read. The contents are the result of dividing the distance from the inner layer module to the outer layer module by the pixel pitch in the X (phi) direction.

TwoOverOuterR \_a mask=0x40058002

func[19:16]=4’b0101, reg\_mem[15]=1’b1, reg\_loc[4:0]=5’b00010

TwoOverOuterR \_b mask=0x4005800a

func[19:16]=4’b0101, reg\_mem[15]=1’b1, reg\_loc[4:0]=5’b01010

TwoOverOuterR \_c mask=0x40058012

func[19:16]=4’b0101, reg\_mem[15]=1’b1, reg\_loc[4:0]=5’b10010

This register is an 18-bit signed number, but it is always positive. It can be written or read. The contents are the result of dividing 2.0 by the product of the radius of the outer layer module times the pixel pitch in the X (phi) direction.

DXOverR \_a mask=0x40058003

func[19:16]=4’b0101, reg\_mem[15]=1’b1, reg\_loc[4:0]=5’b00011

DXOverR \_b mask=0x4005800b

func[19:16]=4’b0101, reg\_mem[15]=1’b1, reg\_loc[4:0]=5’b01011

DXOverR \_c mask=0x40058013

func[19:16]=4’b0101, reg\_mem[15]=1’b1, reg\_loc[4:0]=5’b10011

This register is an 18-bit signed number, but it is always positive. It can be written or read. The contents are the result of dividing the pixel pitch in the X (phi) direction by the radius of the inner layer module.

DXrsq \_a mask=0x40058004

func[19:16]=4’b0101, reg\_mem[15]=1’b1, reg\_loc[4:0]=5’b00100

DXrsq \_b mask=0x4005800c

func[19:16]=4’b0101, reg\_mem[15]=1’b1, reg\_loc[4:0]=5’b01100

DXrsq \_c mask=0x40058014

func[19:16]=4’b0101, reg\_mem[15]=1’b1, reg\_loc[4:0]=5’b10100

This register is an 18-bit signed number, but it is always positive. It can be written or read. The contents are the product of the square of the pixel pitch in the X (phi) direction times the square of the radius of the inner layer module. The final result is divided by 24.

ROverDX \_a mask=0x40058005

func[19:16]=4’b0101, reg\_mem[15]=1’b1, reg\_loc[4:0] =5’b00101

ROverDX \_b mask=0x4005800d

func[19:16]=4’b0101, reg\_mem[15]=1’b1, reg\_loc[4:0] = 5’b01101

ROverDX \_c mask=0x40058015

func[19:16]=4’b0101, reg\_mem[15]=1’b1, reg\_loc[4:0] = 5’b10101

This register is an 18-bit signed number, but it is always positive. It can be written or read. The contents are the result of dividing the radius of the inner layer module by the pixel pitch in the X (phi) direction.