Particle Tracking Silicon Microscope

Version 1.0

Author: Forest Martinez-M^C^Kinney

1.0 Purpose

The purpose of this paper is to provide a rough overview of the Particle Tracking Silicon Microscope system under design at the SCIPP, Santa Cruz Institute of Particle Physics as well as provide documentation on the parallel to serial conversion aspect of the data compression and analysis system.

2.0 PTSM Overview

Particle Tracking Silicon Microscope, PTSM, is a detector system for measuring the location and energy of protons, alpha particles and heavy ions after they have traversed a biological sample. The experimental applications of PTSM are targeted at radiobiology for biomedical and space investigations.

The experimental situation calls for a worm, *C*. elegance, to be placed on the surface of the detector. The detector and the worm are then bombarded with protons, Alpha Particles or heavy ions. The particles that traverse the sample deposit some of their energy in the sample before depositing most or all of their remaining energy in the detector. The distribution of energy about the detector allows the worm to be imaged at the cellular level. The correlation of energy deposited in relation to specific cells of the worm will be crucial to the dosimetery, the study of radiation dosages.

The detector system utilizes diode junctions where each diode is laid out as strips on the front and back of the detector forming a right-angled grid. A 'hit' occurs when a particle traverses the detector causing the incident diodes, front and back, to produce a current. If the current is large enough, that specific junction then it will be register as a hit with the readout electronics. A track causes two hits on the detector to be registered, this is known as an event. Events will be reconstructed on the computer by correlating what diodes, also know as channels, have registered a hit and when those hits occurred. Once events are reconstructed from hits, the amount of energy deposited in the detector can be used to reconstruct an image.

The system configuration is being designed to be relatively inexpensive, easy to use, with low development costs. Due to concerns of the overall expenses both in terms hardware and engineering costs, the system configuration is planned to be as seamless as possible. This factor has lead to a number of design decisions. The Particle Microscope Frontend, PMFE, will perform mostly analog functions. The digital aspects are transferred to an FPGA based system to minimize costs associated with complex ASIC design and fabrication. All signaling will be done with, Low Voltage Differential Signaling, LVDS, to minimize digital noise getting to the PMFE. The data will be processed with a Data

Acquisition, DAQ, Card and Lab View software by National Instruments to reduce software/hardware interface issues.

3.0 Particle Tracking Silicon Microscope System

The crucial PTSM system elements are the shown in Figure 1. The MultiChip Module, MCM, is the heart of the dector system. The Translator Board will convert LVDS signals from a Field Programmable Gate Array, FPGA, to CMOS for the DAQ card 6534. The other card, NI 6703, is an Digital to Analog Converter, DAC, which will provide the DC voltages to the MCM. The +/-50 volt source provides the detector biase voltage and will be powered by the computer.

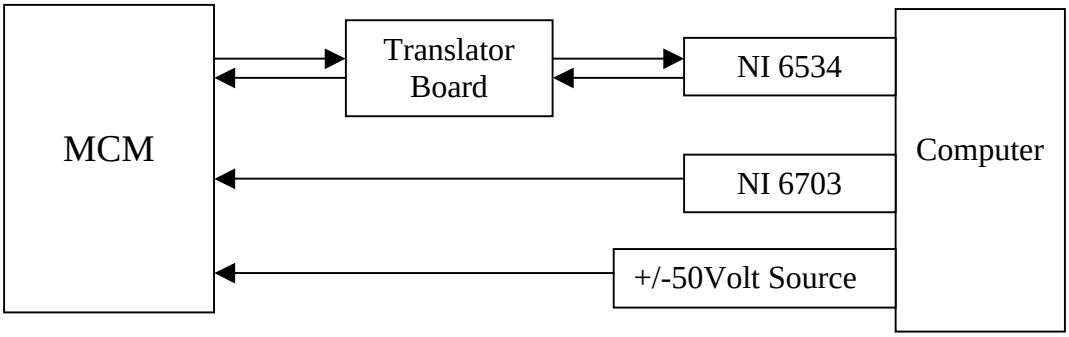

Figure 1.
PTSM System Components

3.1 Multi-Chip Module:

The MCM will be the main detector readout electronics. It will consist of a PCB with a Double Sided Silicon Strip Detector, DSSD, 4 Particle Microscope Frontend chips, 4 PMFE, 1 FPGA, 1 Programmable Read Only Memory, PROM, Chopper circuit, power supply filtering and some required physical features.

Among these features will be strain relief for connectors and ground leads, clamp points for mounting the MCM, epoxy encapsulation for bond out wires between PMFE, DSSD and PCB. The PCB will be machined so that the DSSD can be counter sunk into the board. The system must be designed so that it can be easily set up and dismantled from the radiation source.

3.2 The Detector:
The DSSD chip has diode junctions laid out on both sides of the silicon perpendicular to each other thus yielding an X-Y grid. These junctions will be reverse biased by 100 volts with a fixed high voltage supply. When the proton, alpha particle or ion traverses a junction a group of e^--hole pairs will be formed. They migrate with respect to the depletion region E-field and are seen by the PMFE as a current signal to be processed.

There are data possessing considerations that arise from the physical behavior of the detectors. There will be charge sharing on the neighbor strips for a given side and a single event. The resulting data must be filtered to show which channel the particle actually traversed. Analysis of the charge sharing that occurs between channels allows a spatial resolution less than the channel pitch, Appendix A. There is also an issue known as Ghosts that arises when there are two many hits in comparison to the time resolution of the PMFE.

Ghosts occur when two hits occur at the same time. It is not possible to correlate the proper X-coordinate with the proper Y-coordinate. In the instance below, given both particles traverse the sample at the same time, it is not possible to determine whether the channel coordinates of the hit are (X_1, Y_1) and (X_2, Y_2) or (X_1, Y_2) and (X_2, Y_1). When this occurs the data from these two tracks must be thrown out. The only way to minimize this error is to reduce the amount of data by reducing the particle spill rate.

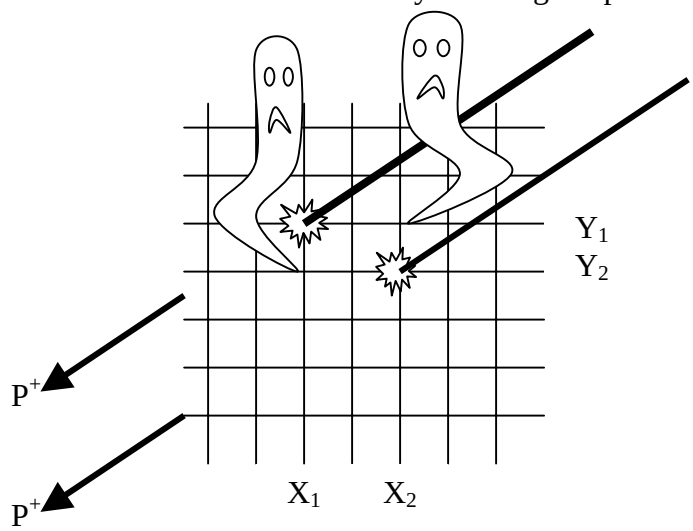

Figure 2.
Ghost hits at (X_1, Y_2) and (X_2, Y_1)
Actual hits at (X_1, Y_1) and (X_2, Y_2)

3.3 Particle Microscope Frontend:

The PMFE is mostly an analog ASIC with minimum digital hardware at the data output. This configuration allows the digital processing to be concentrated on the FPGA and the analog features to be preformed by the ASIC. The functions preformed by the PMFE are current to voltage conversion and amplification, signal shaping, followed by a discriminator stage for a Time Over Threshold, TOT, measurement. There is additional circuitry for calibrating the discriminator with respect to amplifier gain and noise.

The FPGA will provide two clock signals for the PMFE and receive data back on the eight serial lines. The first clock is the Fast Clock used by the PMFE to latch data out Double Data Rate, on its rising and falling edges. The second clock signal, Clk-data, is not a 50% duty cycle clock. Clk-data is used internally by the PMFE to generate an internal reset signal. The result of this configuration is that Clk-Data is also a frame for the 5th clock cycle where no valid data is provided by the PMFE on the serial lines due to the reset.

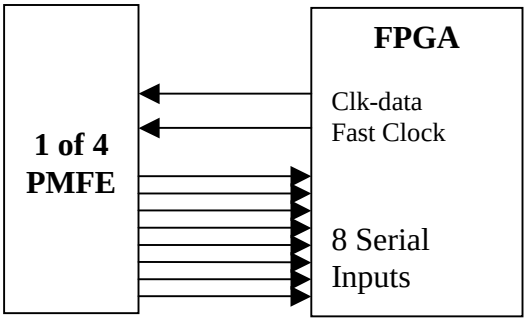

Figure 3.
FPGA to PMFE signal lines

The multifunctional aspects of Clk-data signal require the width of the pulse be variable to accommodate the reset signal. Reset is generated by an AND between the Fast Clock signal and Clk-data and clears the data from the output for one cycle of the Fast Clock. The phase relation of the Fast Clock and Clk-Data signals at the PMFE with respect to them at the FPGA must to be adjusted to for the propagation delay through the PMFE. The data to Fast Clock relation shown in Figure 3 is what the FPGA needs at its input to latch valid data. The PMFE requires the Fast Clock to Clk-Data have a setup time greater 2ns and a hold time of 3ns to meet the reset requirements.

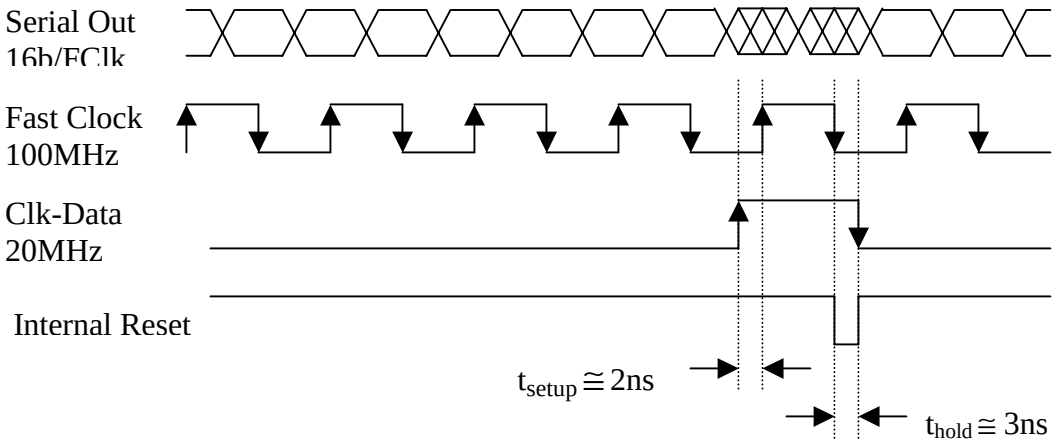

Figure 4.
Crucial PMFE Signals

3.4 Field Programmable Gate Array:

The FPGA fills many requirements in terms have functional and system configuration versatility. Designs can be easily modified and implemented unlike ASIC design where the system configuration must be completely realized before fabrication. The Xilinx VirtexII family of FPGAs has many features that are ideal for this application.

- LVDS
- Digital Clock Manager, DCM
- Block Ram

LVDS is crucial minimize digital noise on the PMFE power Supplies.

The DCM is a very important resource on the FPGA it allows an adjustable phase shift to be applied to a clock at that can be adjusted to 1/256 of the clock period. This will be critical for match the data to the propagation delay of the PMFE. It also provides clock multiplication with limitations on skew [3].

Block Ram is a crucial asset due to the data throughput requirements of the FPGA. In order to realize a system that does not lose data, it is critical to have large enough data buffers to handle all channels and provide a structure that is not overwriting data.

3.5 System Modes of the FPGA:

The FPGA will control all data acquisition and calibration features. There will be one state machine that communicates with the computer and provides the hardware configuration modes of the PTSM system. The state machine will control all handshaking between MCM and the Computer and the three perspective modes of the FPGA.

- Normal Operation
- Calibration
- Testing

3.5.1 Normal Operation

The Normal Operation Mode will be the typical operating configuration. In this mode communication between the MCM and the computer will be continuous. The communications between the FPGA and the computer are crucial for system error detection, system initialization, clock management, output data rate control, and loading system configuration. No matter what mode the FPGA is in, the managerial state controls will be active. Calibration and testing require most Normal Operation hardware; thus they are depicted as subsets in Figure 5.

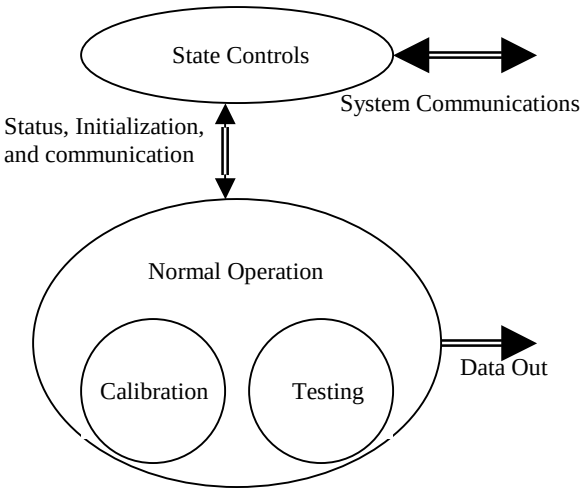

Figure 5.
FPGA Configuration

The state machine will need to have the following features:

- Communication between FPGA and Computer
- System Mode Control
- System Initialization/Reset
- Channel Mask Loading capability
- FPGA to Computer clock Synchronization
- Error Flags
- Wait signals

There will be a known startup sequence and configure of all features or modes. The computer or the FPGA will be able to initiate communications with the other, which allows wait signals or errors to be managed during operation. This will be done through two dedicated serial lines. The system startup up sequence should consist of a power up, establish communications with FPGA, set mode, load channel mask (if needed), and start taking data.

A channel mask is required to reduce the amount of unneeded data taken during an experiment. The system will be used with a biological sample, thus the necessity to "listen" to all channels on the detector will be reduced to those covered by and adjacent to the sample.

The internal FPGA clock and the computer clock must be in sync with respect to the data or no correlations will be possible. The data that is sent to the computer from the FPGA is compressed to a channel number and a start time for the beginning of a hit and later a channel number and stop time for the end of a hit. The computer is responsible for the correlations.

The first order correlation is matching start times with stop times for a given channel. This will require any possible clock roll over errors to be handled. The second order correlation is the channel-to-channel charge sharing which produces yields a particle location with respect to two channels on one of the DSSD. The final correlation is the front and back plains of the DSSD two reconstruct events.

3.5.2 System Calibration:

The calibration of the system is crucial for accurate Time Over Threshold, TOT, to particle energy correlation. Calibrating the system requires a characterization of the Frontend analog electronics. The PMFE uses the differentiating characteristics of capacitors with respect to the rising edge of a pulse to simulate an input current from a hit. The width of the pulse must be longer than the shaping time to keep from seeing the current demands caused by differentiating the falling edge of the pulse.

The National Instruments 6703 DAC card will provide varying amplitude voltage at the control of the FPGA. The MCM will have a Chopper circuit at to provide the step at the control of the FPGA. By stepping though the range of threshold settings, while varying the amplitude and width of the input step, a plot of data efficiency with respect to these variables will show the noise and gain of the amplifier.

3.5.3 Testing:

This Mode is meant to be a system hardware check. The system can be tested after any hardware alterations, changes, or stresses have occurred. Testing will consist of loading known bit patterns into the data pipeline and seeing if they are processed as expected while ignoring any signals from the PMFE. This mode is important for verifying that the system is operating properly when anything has been altered or data does not appear as expected.

3.6 LVDS to CMOS Translator Board

Due to the need to transmit the data from the FPGA to the computer over a cable length of approximately 3 meters at 20MHz, LVDS signaling is required to maintain digital signal integrity and minimize switching noise on the analog power rails. LVDS I/O DAQ cards are scarce. A translator board is required to convert the FPGA LVDS to CMOS for the N.I. DAQ card. The Translator board is being designed to be transparent to FPGA/Computer communications.

3.7 Cabling Between Hardware

One of the major limiting factors of the system is the physical constraint of the hardware configuration. Signals must be routed with LVDS pairs and terminating resistances. The fan out from the PMFE to the DSSD and the FPGA to the DSSD with respect to power consumption, trace dimensions and characteristics limit the possible configurations of the MCM. The cabling between the MCM and the Translator Board must ensure that noise from the TTL side of the translator board doesn't reach the MCM. All these factors are curial to maintain the reliability of the system.

3.8 The data rates

The speed often system is limited by many different hardware factors. There is some ability to tune these factors but they all have their own limits. The most fundamental control of the system data rate is the spill rate of the particle beam. In the optimized case 33% of the time there will be no hits, 33% of the time there will be single hits, and 33% of the time there will be ghost hits (Poisson distribution). For every edge of the Fast clock the PMFE will latch out a set of 8 bits. This yields a bit rate between the 4 PMFE and FPGA of 2Gb/s, Appendix A, which far exceeds the possible bit transition rates of the rest of the system. The FGPAs main function could thus be stated as data compression.

The maximum number of bits for a hit from the PMFE is $TOT_{max} * f_{Clk\text{-}Data}$, Appendix A.2, which is 600 Clk-Data cycles. The FPGA will be reducing the number of bits for a hit down to 64 bits, which amounts to 32 bits for channel number & start time, channel number & stop time. The DAQ card then must read the data out of the FPGA.

The NI 6534 DAQ Card is what limits the spill rate of the particle beam since the system readout electronics can easily overwhelm the DAQ card data acquisition rate. The DAQ card requires 4 reads to get the data from one hit. Thus the DAQ card can read 5Mhits/s. has to be passed out of the DAQ card so the rate may go down even further.

3.9 Data Processing Inside the FPGA:

The data processing inside the FPGA is depicted in Figure 6. The data from the PMFE is read into the FPGA using the Fast and Double Data Rate, DDR. For every Clk-Data period all 64 Channels from each PMFE is read into the FPGA and presented in parallel to the Channel Monitoring block. The Channel Monitoring Block examines all the channels for a low to high or high to low transition and latches out a start time and channel number or stop time and channel number respectively. These times and addresses are stored in small FIFO Channel Buffers, 8 channels to each buffer, as temporary storage. These buffers are then read into the Output Buffer depending on which has the most data in it. These will keep data from being over written but requires special attention to roll over of the internal clock. A 2∗(Fast Clock) and Clk-data is be used internally for processing data. The Clock will reference the CLK-Data since it defines the resolution of the system. The Buffer Controls keep data from being over written and synchronize data transfers.

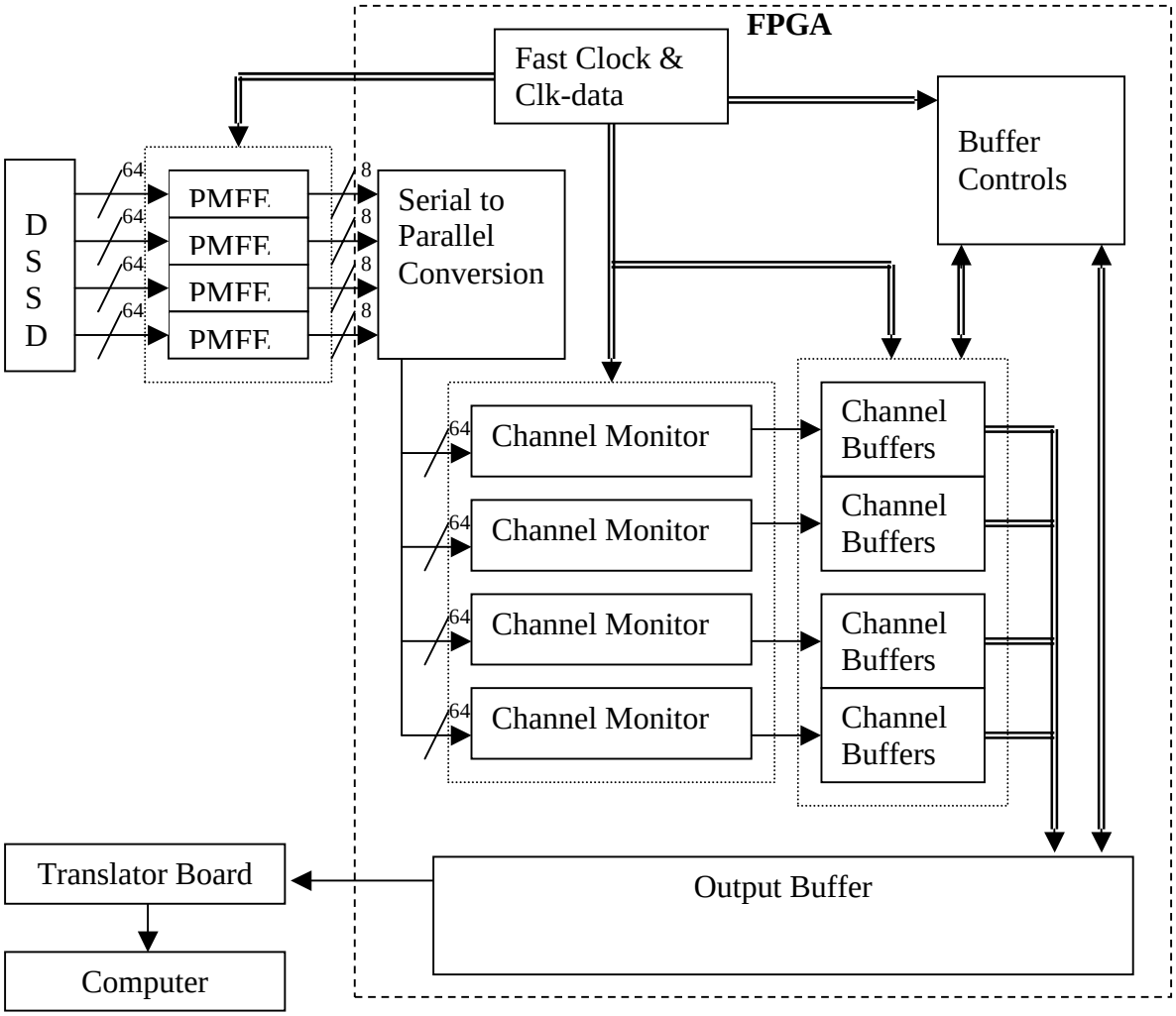

Figure 6.
Data path inside FPGA

4.0 Serial to Parallel Conversion

The Serial to Parallel Conversion Block is written in the Verilog Hardware Design Language. This block will be a modified version of an application note, xapp265, provided by Xilinx for a demo board. All aspects of the design are not fully understood or needed. The application utilizes Block RAM in a dual port configuration to realize a FIFO for buffering data transmitted out of the FPGA with LVDS and received into the FPGA with LVDS. The PTSM system will not require the LVDS transmitter portion of the application note and the structure of the receiver will require modifications to meet the constraints of the data format from the PMFE. The emphasis of this documentation is to familiarize the reader with the structure of the parallel to serial conversion portion of the application note while pointing out some of the modifications needed to both test and focus the design at the PTSM Read Out Controller, PTSM ROC.

4.1 Verilog Terminologies and Structure:

There are some conventions used when discussing Verilog code. Code that describes the functions of a digital hardware block having inputs and outputs is called a module. Verilog C like language with statements with call typical digital hardware available in the software libraries. One module can call instances of the other modules much the same as a routine can call a function in normal programming. The main difference from other programming languages is that modules operate in parallel rather than sequentially just as a real circuit does.

5.0 Xilinx Application Note 265

The serial to parallel Verilog module is provided by Xilinx and described in their application notes as a “High-Speed Serialization and Deserialization (840 Mb/s LVDS).” The module has three different levels. The upper most module, top16a, controls the clock speeds and phase, instantiates the LVDS input and output buffers, and controls when data is written out of the FIFO. The PTSM system only requires the receiver portion, with modifications, therefore that will be the focus.

Each module requires modifications to mesh with our intention of 8x8 to 1X64 bit serial to parallel conversion with no valid data on every fifth clock cycle. These modifications are a mix of adjusting the data widths to meet the FE chip’s 8 serial bit output configuration and extra controls for keeping the invalid fifth clock cycle data from enter the data pipeline. The currently know modifications will be discussed in the Modifications section.

Top-level module, top16a, calls one instance of serdes16b_1to8. This module calls four instances of the lowest level module, serdes4b_1to8 and tiles them together into one module making busses out of all the signals. The resulting reciver is an 8x16 bit to 1X128 bit DDR serial to parallel conversion from data read in via DDR LVDS. For the sake of simplicity top16a will be referred to as ‘Top,’ serdes16b_1to8 will be ‘Middle’ and serdes4b_1to8 will be ‘Bottom.’

6.0 Testing Modifications:

Bottom requires slight modifications to its inputs and the addition of supporting hardware to simulate its functions Figure 7. The module requires two input clocks to operate with DDR. However, the simulation software bases all transitions on a single reference clock. The software only allows the toggling of different with respect to the rising or falling, not both, edges of the reference clock. This constraint makes it impossible to easily follow the data as it is processed. The two input clocks, rxclk and rxclknot, which must be the exact opposites of each other, are replaced with a single input clock, Masterclk, to which rxclk is set equal and rxclknot is the inverted version. This internal assignment allows a single input clock to be distributed in its inverted and original form. When the testing of this block is complete the Masterclk signal must be disabled so that rxclk and rxclknot can be reassigned as inputs.

The additional hardware consists of a clock delay locked loop. Using the slower clock to drive the Masterclk input of Bottom and assigning the reference clock to the input of the clkdll allows the input data to be toggled between the rising and falling edges of the Masterclk input.

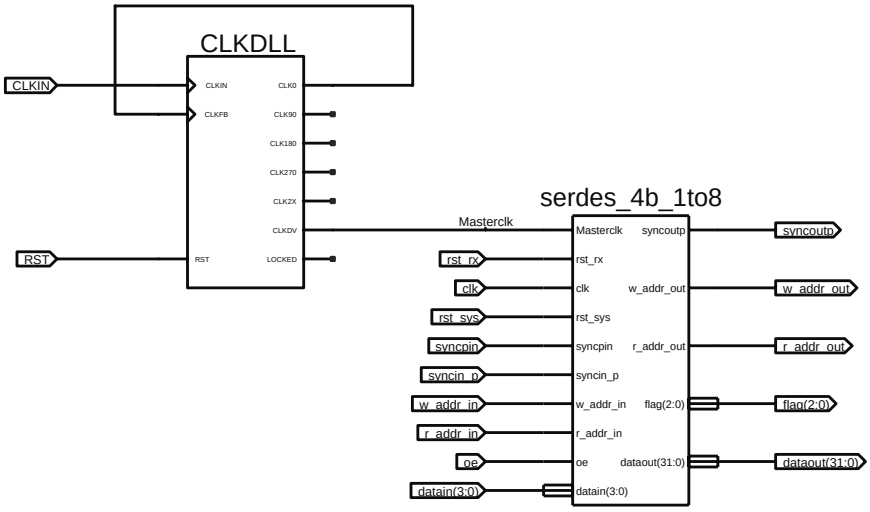

Figure 7.
Additional hardware and changes to Bottom for accurate DDR simulations.

7.0 Different Blocks:
The internal structure of Bottom can be divided into the path that the data follows input ports to output ports and the controls that keep the positive edge data in sync with the negative edge data. Bottom reads the data for the LVDS input Buffers at double data rate and separates the positive edge data from the negative edge data. The respective data converted from serial to parallel then interleaved into the proper order to be inserted into the FIFO buffer. The synchronization of the data is maintained through the control blocks, which keep positive and negative data aligned while also providing synchronization for tiling multiple instances of Bottom.

- Blocks for the data path:
  - Double Data Rate
  - Shift-Register
  - Positive and negative data interleave
  - FIFO Buffer

- Control Signals:
  - Addressing look up tables
  - FIFO flags
  - Positive and Negative edge data controls
  - Enabling and Reset signals

The schematic representations of the different blocks are the actual components that make up the version of the receiver that will be used. The receiver is comprised of three levels of Verilog modules that have only one schematic symbol associated with them. The schematics for each block are a representation Verilog code that describes them but are not the meant to be used as the functional equivalent. The Verilog design includes many constraints on the locations that specific components can be placed on the FPGA.

7.1.1 DDR block:
Double data rate is accomplished by latching data into the data pipeline on rising and falling edges of the master clock. Since flip-flops are rising edge triggered an inverted version of the clock is used to latch the data for the falling edge of the master clock. This method of receiving data increases the speed by a factor of two but also requires new data to be valid at the input on very rising and falling clock edge [2].

The two clocks used are rxclk and rxclknot. The result is that data is latched from the input lines on both the rising fall edges of the Masterclk that they are derived from. The module then processes the positive and negative edge data in parallel during the deserialization portion. Data is processed with respect to their latching clock, master clock, rxclk, for the positive edge data. The inverted master clock rxclknot, is used for processing the data that is acquired on the falling edge of the master clock.

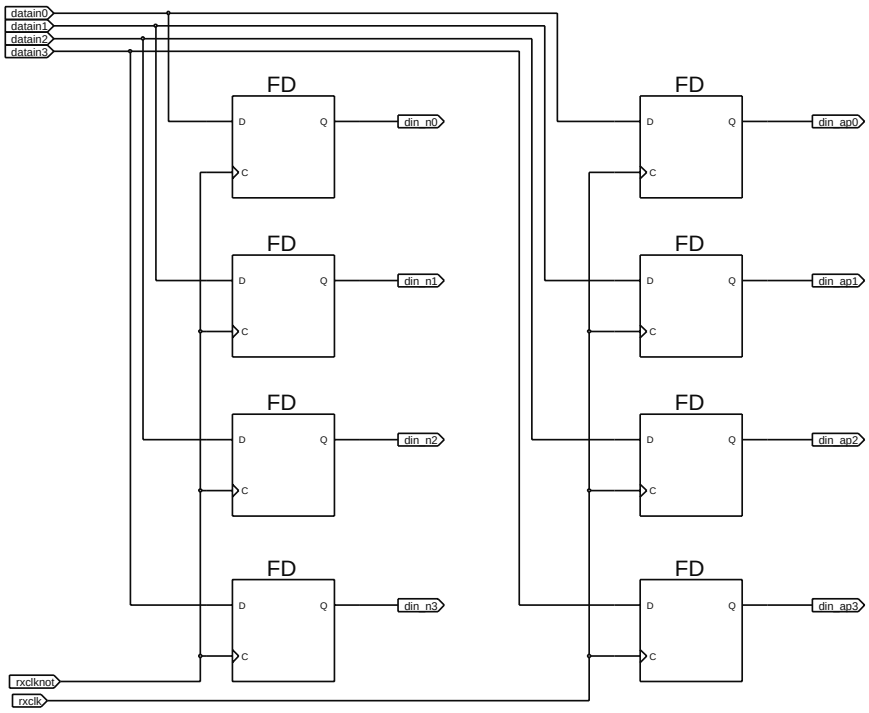

Figure 8
Double Data Rate Block

```
FD fdn0(.C(rxclknot), .D(datain[0]), .Q(din_n[0]))/* synthesis IOB = "TRUE" */; // -ve edge IOB regs
FD fdn1(.C(rxclknot), .D(datain[1]), .Q(din_n[1]))/* synthesis IOB = "TRUE" */;
FD fdn2(.C(rxclknot), .D(datain[2]), .Q(din_n[2]))/* synthesis IOB = "TRUE" */;
FD fdn3(.C(rxclknot), .D(datain[3]), .Q(din_n[3]))/* synthesis IOB = "TRUE" */;

FD fdp0(.C(rxclk), .D(datain[0]), .Q(din_ap[0]))/* synthesis IOB = "TRUE" */; // +ve edge IOB regs
FD fdp1(.C(rxclk), .D(datain[1]), .Q(din_ap[1]))/* synthesis IOB = "TRUE" */;
FD fdp2(.C(rxclk), .D(datain[2]), .Q(din_ap[2]))/* synthesis IOB = "TRUE" */;
FD fdp3(.C(rxclk), .D(datain[3]), .Q(din_ap[3]))/* synthesis IOB = "TRUE" */;
```

7.1.2 Shift Block:

For both positive edge data and negative edge data the serial to parallel conversion is done by latching four parallel bits of data into four 4 bit registers. At the rising edge of each clock the data is presented to the next set of flip-flops. Every three clock cycles there are four sets ready to be latched into the sixteen bit register. On the fourth clock cycle the data from the four registers is presented to the to the 16bit register and the next four bits are latched into the deserialization block.

The sixteen-bit register is intermediate storage for the positive and negative edge data. The data from these two registers is then interleaved at the input of the FIFO to correspond with the order that it arrival order. These two registers are clocked with signals from the CENINT and CEPINT control busses. This allows the 180° phase shift between positive and negative edge data to be ignored at the FIFO input.

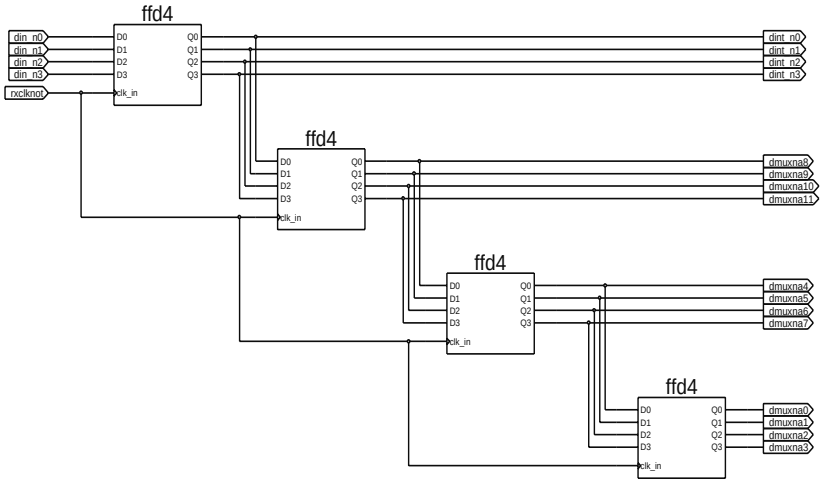

Figure 9
4 bit negative edge data Shift Block

```
FD fds4  (.C(rxclknot), .D(din_n[0]), .Q(dint_n[0]))/* synthesis RLOC = "X0Y4" */;
FD fds5  (.C(rxclknot), .D(din_n[1]), .Q(dint_n[1]))/* synthesis RLOC = "X0Y4" */;
FD fds6  (.C(rxclknot), .D(din_n[2]), .Q(dint_n[2]))/* synthesis RLOC = "X0Y6" */;
FD fds7  (.C(rxclknot), .D(din_n[3]), .Q(dint_n[3]))/* synthesis RLOC = "X0Y6" */;
FD fdce_dn0 (.C(rxclknot), .D(dmuxna[4]),  .Q(dmuxna[0]))/* synthesis RLOC = "X4Y2" */;
FD fdce_dn1 (.C(rxclknot), .D(dmuxna[5]),  .Q(dmuxna[1]))/* synthesis RLOC = "X4Y2" */;
FD fdce_dn2 (.C(rxclknot), .D(dmuxna[6]),  .Q(dmuxna[2]))/* synthesis RLOC = "X4Y3" */;
FD fdce_dn3 (.C(rxclknot), .D(dmuxna[7]),  .Q(dmuxna[3]))/* synthesis RLOC = "X4Y3" */;
FD fdce_dn4 (.C(rxclknot), .D(dmuxna[8]),  .Q(dmuxna[4]))/* synthesis RLOC = "X4Y6" */;
FD fdce_dn5 (.C(rxclknot), .D(dmuxna[9]),  .Q(dmuxna[5]))/* synthesis RLOC = "X4Y6" */;
FD fdce_dn6 (.C(rxclknot), .D(dmuxna[10]), .Q(dmuxna[6]))/* synthesis RLOC = "X4Y7" */;
FD fdce_dn7 (.C(rxclknot), .D(dmuxna[11]), .Q(dmuxna[7]))/* synthesis RLOC = "X4Y7" */;
FD fdce_dn8 (.C(rxclknot), .D(dint_n[0]),  .Q(dmuxna[8]))/* synthesis RLOC = "X5Y4" */;
FD fdce_dn9 (.C(rxclknot), .D(dint_n[1]),  .Q(dmuxna[9]))/* synthesis RLOC = "X5Y4" */;
FD fdce_dn10(.C(rxclknot), .D(dint_n[2]),  .Q(dmuxna[10]))/* synthesis RLOC = "X5Y6" */;
FD fdce_dn11(.C(rxclknot), .D(dint_n[3]),  .Q(dmuxna[11]))/* synthesis RLOC = "X5Y6" */;
```

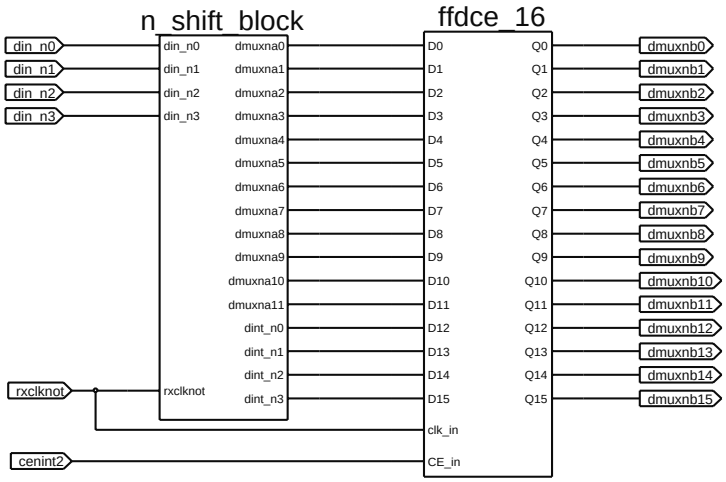

Figure 10
Parallel negative data to be read into FIFO

```
FDE fdce_rn0 (.C(rxclknot), .D(dmuxna[0]),  .CE(cenint[2]), .Q(dmuxnb[0]))/* synthesis RLOC = "X2Y2"
*/;
FDE fdce_rn1 (.C(rxclknot), .D(dmuxna[1]),  .CE(cenint[2]), .Q(dmuxnb[1]))/* synthesis RLOC = "X2Y2"
*/;
FDE fdce_rn2 (.C(rxclknot), .D(dmuxna[2]),  .CE(cenint[2]), .Q(dmuxnb[2]))/* synthesis RLOC = "X2Y3"
*/;
FDE fdce_rn3 (.C(rxclknot), .D(dmuxna[3]),  .CE(cenint[2]), .Q(dmuxnb[3]))/* synthesis RLOC = "X2Y3"
*/;
FDE fdce_rn4 (.C(rxclknot), .D(dmuxna[4]),  .CE(cenint[2]), .Q(dmuxnb[4]))/* synthesis RLOC = "X2Y6"
*/;
FDE fdce_rn5 (.C(rxclknot), .D(dmuxna[5]),  .CE(cenint[2]), .Q(dmuxnb[5]))/* synthesis RLOC = "X2Y6"
*/;
FDE fdce_rn6 (.C(rxclknot), .D(dmuxna[6]),  .CE(cenint[2]), .Q(dmuxnb[6]))/* synthesis RLOC = "X2Y7"
*/;
FDE fdce_rn7 (.C(rxclknot), .D(dmuxna[7]),  .CE(cenint[2]), .Q(dmuxnb[7]))/* synthesis RLOC = "X2Y7"
*/;
FDE fdce_rn8 (.C(rxclknot), .D(dmuxna[8]),  .CE(cenint[2]), .Q(dmuxnb[8]))/* synthesis RLOC = "X3Y2"
*/;
FDE fdce_rn9 (.C(rxclknot), .D(dmuxna[9]),  .CE(cenint[2]), .Q(dmuxnb[9]))/* synthesis RLOC = "X3Y2"
*/;
FDE fdce_rn10(.C(rxclknot), .D(dmuxna[10]), .CE(cenint[2]), .Q(dmuxnb[10]))/* synthesis RLOC =
"X3Y3" */;
FDE fdce_rn11(.C(rxclknot), .D(dmuxna[11]), .CE(cenint[2]), .Q(dmuxnb[11]))/* synthesis RLOC =
"X3Y3" */;
FDE fdce_rn12(.C(rxclknot), .D(dint_n[0]),  .CE(cenint[2]), .Q(dmuxnb[12]))/* synthesis RLOC =
"X3Y6" */;
FDE fdce_rn13(.C(rxclknot), .D(dint_n[1]),  .CE(cenint[2]), .Q(dmuxnb[13]))/* synthesis RLOC =
"X3Y6" */;
FDE fdce_rn14(.C(rxclknot), .D(dint_n[2]),  .CE(cenint[2]), .Q(dmuxnb[14]))/* synthesis RLOC =
"X3Y7" */;
FDE fdce_rn15(.C(rxclknot), .D(dint_n[3]),  .CE(cenint[2]), .Q(dmuxnb[15]))/* synthesis RLOC =
"X3Y7" */;
```

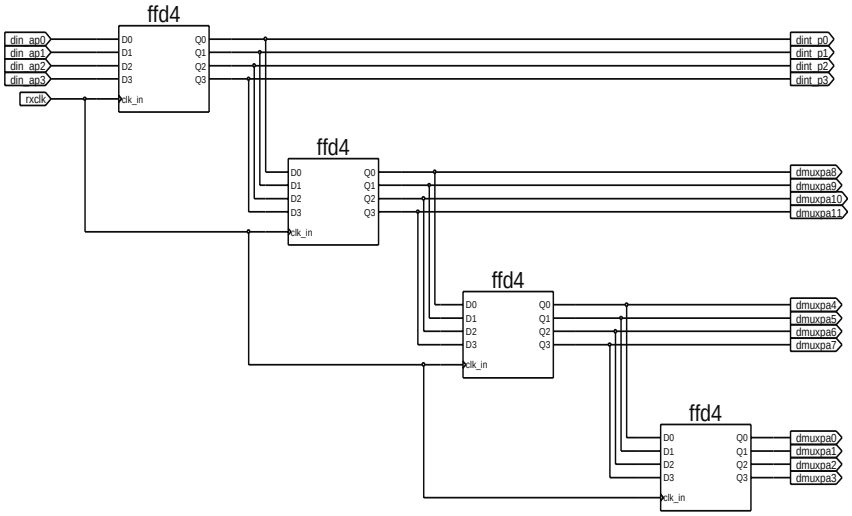

Figure 11
4 bit positive data shift block

```
FD fds0  (.C(rxclk), .D(din_ap[0]), .Q(dint_p[0]))/* synthesis RLOC = "X0Y3" */;
FD fds1  (.C(rxclk), .D(din_ap[1]), .Q(dint_p[1]))/* synthesis RLOC = "X0Y3" */;
FD fds2  (.C(rxclk), .D(din_ap[2]), .Q(dint_p[2]))/* synthesis RLOC = "X0Y5" */;
FD fds3  (.C(rxclk), .D(din_ap[3]), .Q(dint_p[3]))/* synthesis RLOC = "X0Y5" */;
FD fdce_dp0(.C(rxclk), .D(dmuxpa[4]), .Q(dmuxpa[0]))/* synthesis RLOC = "X4Y0" */;
FD fdce_dp1(.C(rxclk), .D(dmuxpa[5]), .Q(dmuxpa[1]))/* synthesis RLOC = "X4Y0" */;
FD fdce_dp2(.C(rxclk), .D(dmuxpa[6]), .Q(dmuxpa[2]))/* synthesis RLOC = "X4Y1" */;
FD fdce_dp3(.C(rxclk), .D(dmuxpa[7]), .Q(dmuxpa[3]))/* synthesis RLOC = "X4Y1" */;
FD fdce_dp4(.C(rxclk), .D(dmuxpa[8]), .Q(dmuxpa[4]))/* synthesis RLOC = "X4Y4" */;
FD fdce_dp5(.C(rxclk), .D(dmuxpa[9]), .Q(dmuxpa[5]))/* synthesis RLOC = "X4Y4" */;
FD fdce_dp6(.C(rxclk), .D(dmuxpa[10]), .Q(dmuxpa[6]))/* synthesis RLOC = "X4Y5" */;
FD fdce_dp7(.C(rxclk), .D(dmuxpa[11]), .Q(dmuxpa[7]))/* synthesis RLOC = "X4Y5" */;
FD fdce_dp8(.C(rxclk), .D(dint_p[0]), .Q(dmuxpa[8]))/* synthesis RLOC = "X5Y3" */;
FD fdce_dp9(.C(rxclk), .D(dint_p[1]), .Q(dmuxpa[9]))/* synthesis RLOC = "X5Y3" */;
FD fdce_dp10(.C(rxclk), .D(dint_p[2]), .Q(dmuxpa[10]))/* synthesis RLOC = "X5Y5" */;
FD fdce_dp11(.C(rxclk), .D(dint_p[3]), .Q(dmuxpa[11]))/* synthesis RLOC = "X5Y5" */;
```

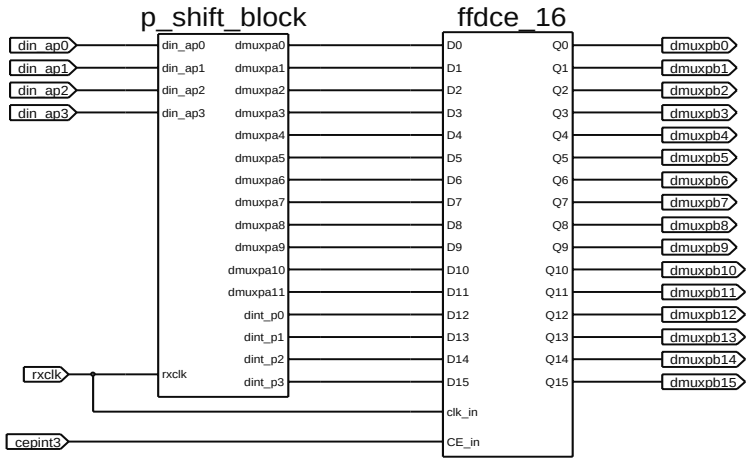

Figure 12
Parallel positive data to be read into FIFO

```
FDE fdce_rp0 (.C(rxclk), .D(dmuxpa[0]),  .CE(cepint[3]), .Q(dmuxpb[0]))/* synthesis RLOC = "X2Y0" */;
FDE fdce_rp1 (.C(rxclk), .D(dmuxpa[1]),  .CE(cepint[3]), .Q(dmuxpb[1]))/* synthesis RLOC = "X2Y0" */;
FDE fdce_rp2 (.C(rxclk), .D(dmuxpa[2]),  .CE(cepint[3]), .Q(dmuxpb[2]))/* synthesis RLOC = "X2Y1" */;
FDE fdce_rp3 (.C(rxclk), .D(dmuxpa[3]),  .CE(cepint[3]), .Q(dmuxpb[3]))/* synthesis RLOC = "X2Y1" */;
FDE fdce_rp4 (.C(rxclk), .D(dmuxpa[4]),  .CE(cepint[3]), .Q(dmuxpb[4]))/* synthesis RLOC = "X2Y4" */;
FDE fdce_rp5 (.C(rxclk), .D(dmuxpa[5]),  .CE(cepint[3]), .Q(dmuxpb[5]))/* synthesis RLOC = "X2Y4" */;
FDE fdce_rp6 (.C(rxclk), .D(dmuxpa[6]),  .CE(cepint[3]), .Q(dmuxpb[6]))/* synthesis RLOC = "X2Y5" */;
FDE fdce_rp7 (.C(rxclk), .D(dmuxpa[7]),  .CE(cepint[3]), .Q(dmuxpb[7]))/* synthesis RLOC = "X2Y5" */;
FDE fdce_rp8 (.C(rxclk), .D(dmuxpa[8]),  .CE(cepint[3]), .Q(dmuxpb[8]))/* synthesis RLOC = "X3Y0" */;
FDE fdce_rp9 (.C(rxclk), .D(dmuxpa[9]),  .CE(cepint[3]), .Q(dmuxpb[9]))/* synthesis RLOC = "X3Y0" */;
FDE fdce_rp10(.C(rxclk), .D(dmuxpa[10]), .CE(cepint[3]), .Q(dmuxpb[10]))/* synthesis RLOC = "X3Y1"
*/;
FDE fdce_rp11(.C(rxclk), .D(dmuxpa[11]), .CE(cepint[3]), .Q(dmuxpb[11]))/* synthesis RLOC = "X3Y1"
*/;
FDE fdce_rp12(.C(rxclk), .D(dint_p[0]),  .CE(cepint[3]), .Q(dmuxpb[12]))/* synthesis RLOC = "X3Y4"
*/;
FDE fdce_rp13(.C(rxclk), .D(dint_p[1]),  .CE(cepint[3]), .Q(dmuxpb[13]))/* synthesis RLOC = "X3Y4"
*/;
FDE fdce_rp14(.C(rxclk), .D(dint_p[2]),  .CE(cepint[3]), .Q(dmuxpb[14]))/* synthesis RLOC = "X3Y5"
*/;
FDE fdce_rp15(.C(rxclk), .D(dint_p[3]),  .CE(cepint[3]), .Q(dmuxpb[15]))/* synthesis RLOC = "X3Y5"
*/;
```

The ordering of the positive and negative data presented to the FIFO is accomplished with a little a shuffle of wires during a bus concatenation.

```
assign ramdatain = {dmuxnb[15:12], dmuxpb[15:12], // bits 31 downs 24
                    dmuxnb[11:8],  dmuxpb[11:8],   // bits 23 downto 16
                    dmuxnb[7:4],   dmuxpb[7:4],    // bits 15 downto 8
                    dmuxnb[3:0],   dmuxpb[3:0]} ;  // bits 7 downto 0
```

7.1.3 FIFO via Block Ram:
The FIF0 buffer is realized using Block Ram. The Xilinx VirtexII has two types of memory available, distributed ram and lock ram. Every CLB has 16bits or ram or there are blocks of the IC that are fabricated as ram, Block Ram. The FIFO is configured with RAMB16_S36_S36 and what is referred to as a "Self Addressing" scheme [5].

The idea behind self-addressing is to use a portion of the ram data bits to store the next write address as well as the data. By doing this, the depth of the FIFO and the number of data bits used completely describe the depth of the FIFO.

The FIFO is 16 slices deep and data is clocked in with the ram_en signal. The ram_en is generated from the cepint[2], cepint[3] and rxclk. The signal is twice the rxclk signal.

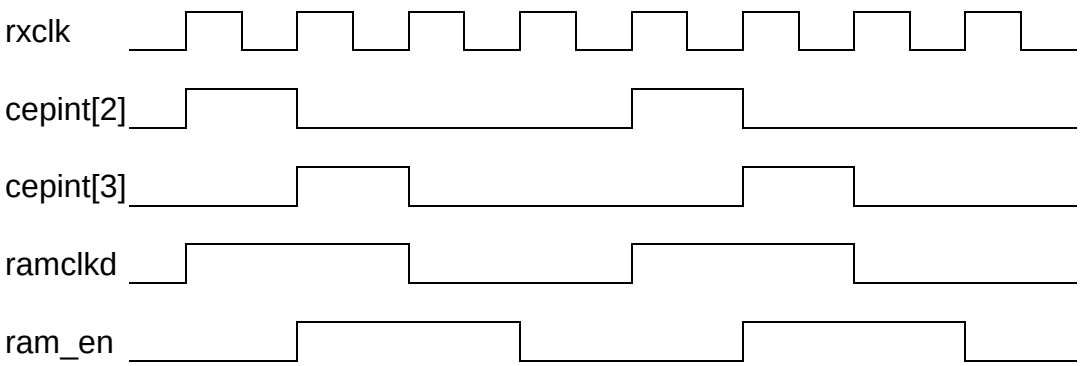

Figure 13
Ram enable signal is created from

The structure of the FIFO requires some consideration of the rate at which data is read from it. If data is read out faster than it is arriving and the buffer empties completely it will go into an unknown state. The unknown state will also be mirrored in flags[0:2]. The FIFO will not break if it fills up and data is not read out. The only consequence is that data will be overwritten.

It is highly unlikely that the ROC system will vary the readout speed of the FIFO. The only consideration that will needed is how full it should be kept during operation and an initialization protocol for the arrival of valid data at its output. There should be an added feature to the FIFO block that tells the next stage of the ROC system when the data is valid. Once an operating depth is decided the FIFO should be read out of as fast as data is written in maintain depth. The

7.2.1 FIFO Addressing:

Look Up Tables, LUTs, are hardwired truth tables. They are used as arbitrary logical function providers for each Slice of each Configurable Logic Block, CLB. LUTs are the most desirable method for producing any logical function since the have the same propagation delay regardless of the function [8]. They produce the next write address for the FIFO that is implemented with ram block [1]. There are five four-bit LUTs in the module. The first four are used to generate the next write address and the fifth is used as a synchronization signal out of the module. When the Grey Code condition of incrementing the address is not satisfied [5].

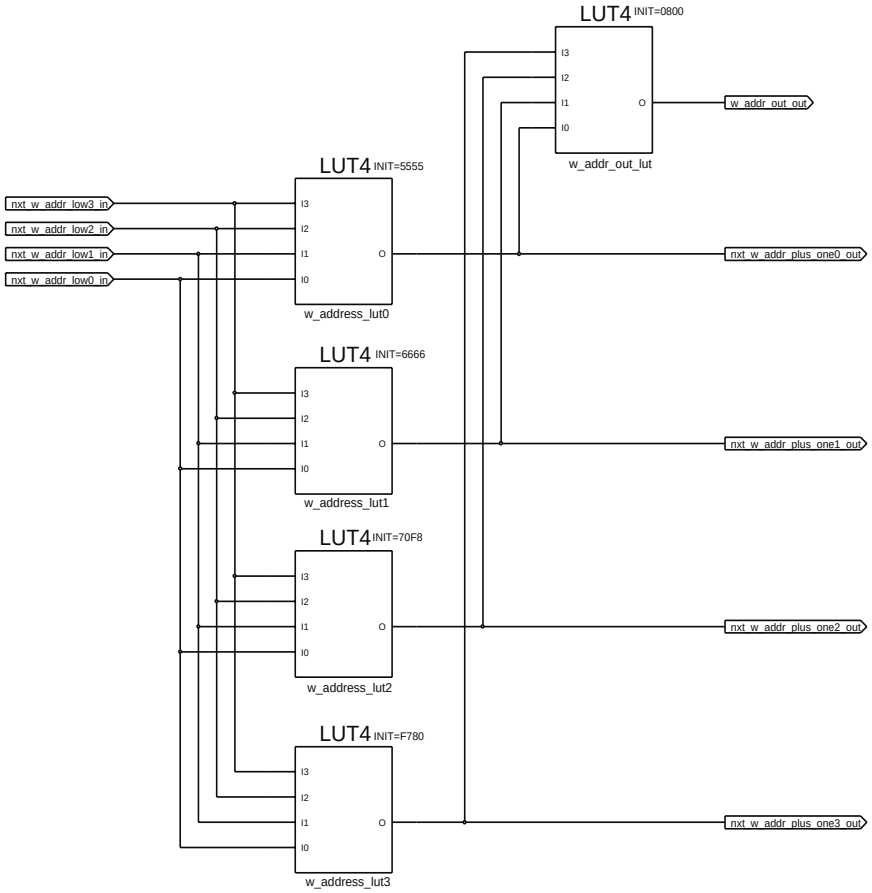

Figure 14
Look Up Tables for generating the FIFO Self Addressing scheme

```
LUT4 w_address_lut0(.I0(nxt_w_addr_low[0]), .I1(nxt_w_addr_low[1]),
                .I2(nxt_w_addr_low[2]), .I3(nxt_w_addr_low[3]), .O(nxt_w_addr_plus_one[0]))
                /*synthesis xc_props = "INIT=5555" rloc = "X1Y6" xc_map = "lut" */ ;
LUT4 w_address_lut1(.I0(nxt_w_addr_low[0]), .I1(nxt_w_addr_low[1]),
                .I2(nxt_w_addr_low[2]), .I3(nxt_w_addr_low[3]), .O(nxt_w_addr_plus_one[1]))
                /*synthesis xc_props = "INIT=6666" rloc = "X1Y6" xc_map = "lut" */ ;
LUT4 w_address_lut2(.I0(nxt_w_addr_low[0]), .I1(nxt_w_addr_low[1]),
                .I2(nxt_w_addr_low[2]), .I3(nxt_w_addr_low[3]), .O(nxt_w_addr_plus_one[2]))
                /*synthesis xc_props = "INIT=70F8" rloc = "X1Y7" xc_map = "lut" */ ;
LUT4 w_address_lut3(.I0(nxt_w_addr_low[0]), .I1(nxt_w_addr_low[1]),
                .I2(nxt_w_addr_low[2]), .I3(nxt_w_addr_low[3]), .O(nxt_w_addr_plus_one[3]))
                /*synthesis xc_props = "INIT=F780" rloc = "X1Y7" xc_map = "lut" */ ;

LUT4 w_addr_out_lut(.I0(nxt_w_addr_low[0]), .I1(nxt_w_addr_low[1]),
                .I2(nxt_w_addr_low[2]), .I3(nxt_w_addr_low[3]), .O(w_addr_out))
                /*synthesis xc_props = "INIT=0800" rloc = "X0Y5" xc_map = "lut" */ ;
```

The table below depicts the correspondence between the nxt_w_addr_low signals and the nxt_w_addr_plus_one. The next address transition table is depicted below. It is read from left to right such that current address is on the left and the next write address is right.

nxt_w_addr_low					nxt_w_addr_plus_one				
	0	1	2	3		0	1	2	3
0	0	0	0	0	1	0	0	0	1
1	0	0	0	1	2	0	0	1	0
2	0	0	1	0	3	0	0	1	1
3	0	0	1	1	4	0	1	0	0
4	0	1	0	0	5	0	1	0	1
5	0	1	0	1	6	0	1	1	0
6	0	1	1	0	7	0	1	1	1
7	0	1	1	1	c	1	1	0	0
8	1	0	0	0	d	1	0	0	1
9	1	0	0	1	e	1	0	1	0
a	1	0	1	0	f	1	0	1	1
b	1	0	1	1	0	0	0	0	0
c	1	1	0	0	d	1	1	0	1
d	1	1	0	1	e	1	1	1	0
e	1	1	1	0	f	1	1	1	1
f	1	1	1	1	8	1	0	0	0

Figure 15
FFIFO addressing table

7.2.2 Flags:

The FIFO module has a set of flags, Flag0-2 that keep track of how full the FIFO is. These flags are derived from the read and write pointers of the FIFO. They can be used as a flag to start reading from the FIFO once it has reached a certain desirable depth. Flag[0] = empty to ½ full, Flag[1] = 1/4 to ¾ full, Flag[2] = ½ to full.

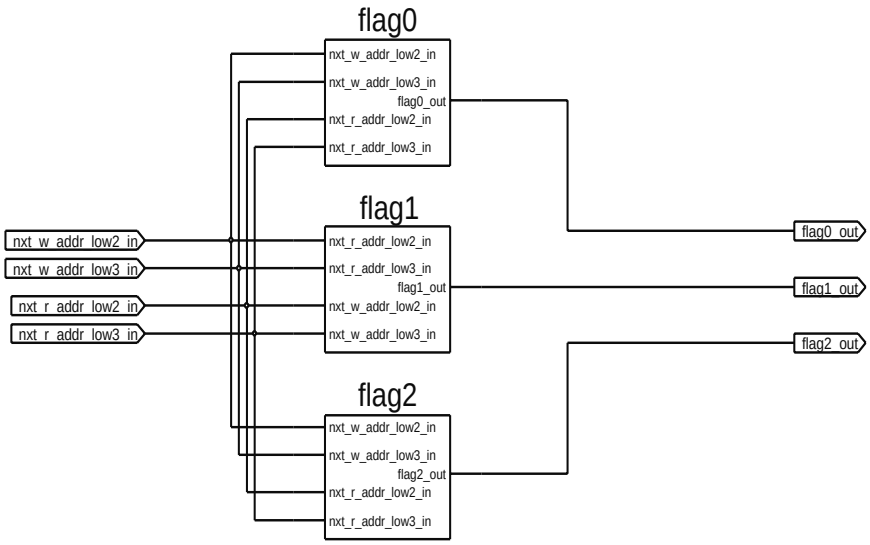

Figure 16
FIFO Flags 0 1 & 2

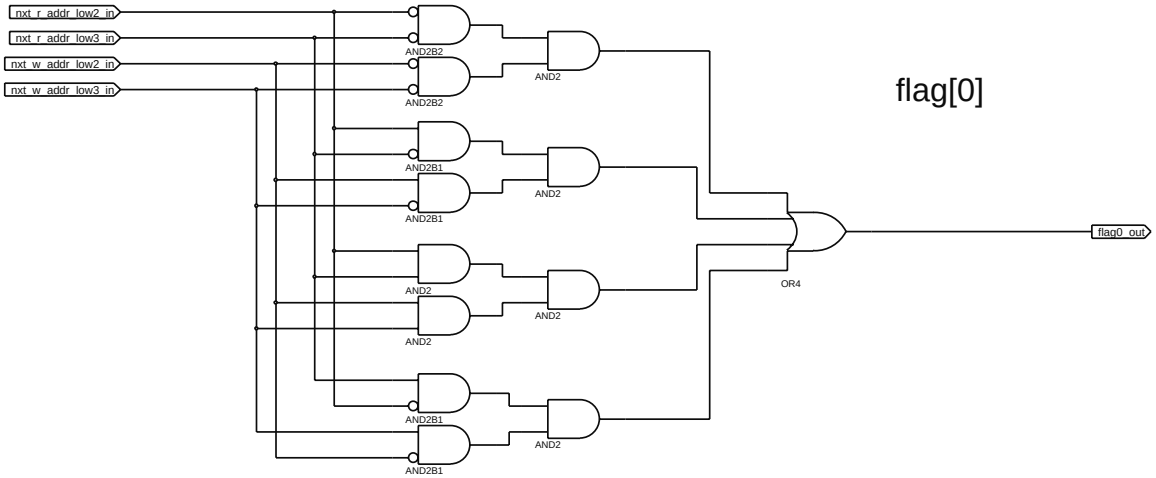

Figure 17
Flag 0 logic

```
assign flag[0] =(((nxt_r_addr_low[3:2] == 2'b00) && (nxt_w_addr_low[3:2] == 2'b00)) |
                ((nxt_r_addr_low[3:2] == 2'b01) && (nxt_w_addr_low[3:2] == 2'b01)) |
                ((nxt_r_addr_low[3:2] == 2'b11) && (nxt_w_addr_low[3:2] == 2'b11)) |
                ((nxt_w_addr_low[3:2] == 2'b10) && (nxt_w_addr_low[3:2] == 2'b10)))
                ? 1'b1 : 1'b0;
```

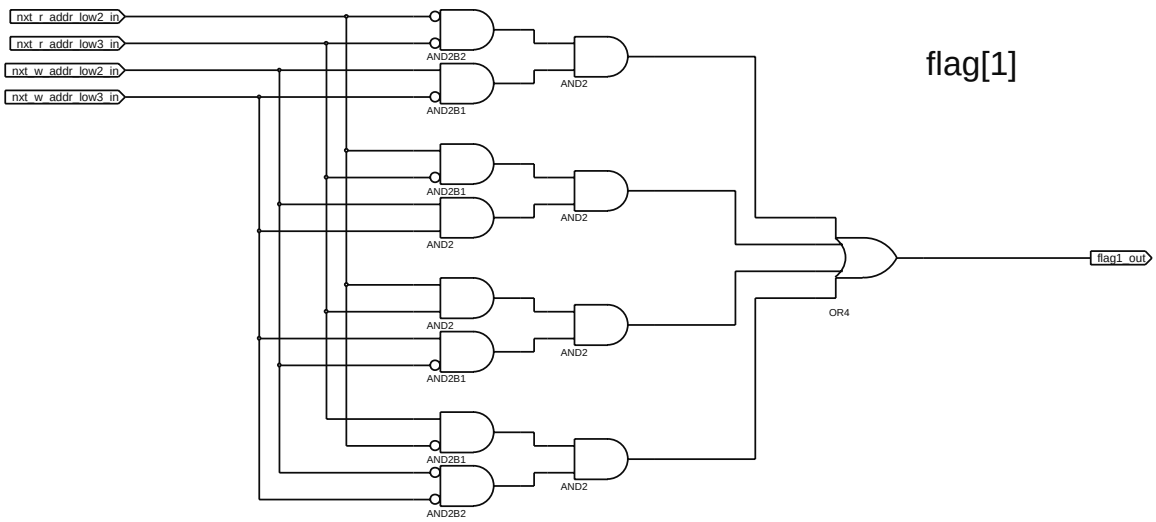

Figure 18
Flag 1 logic

```
assign flag[1] =(((nxt_r_addr_low[3:2] == 2'b00) && (nxt_w_addr_low[3:2] == 2'b01)) |
                ((nxt_r_addr_low[3:2] == 2'b01) && (nxt_w_addr_low[3:2] == 2'b11)) |
                ((nxt_r_addr_low[3:2] == 2'b11) && (nxt_w_addr_low[3:2] == 2'b10)) |
                ((nxt_r_addr_low[3:2] == 2'b10) && (nxt_w_addr_low[3:2] == 2'b00)))
                ? 1'b1 : 1'b0;
```

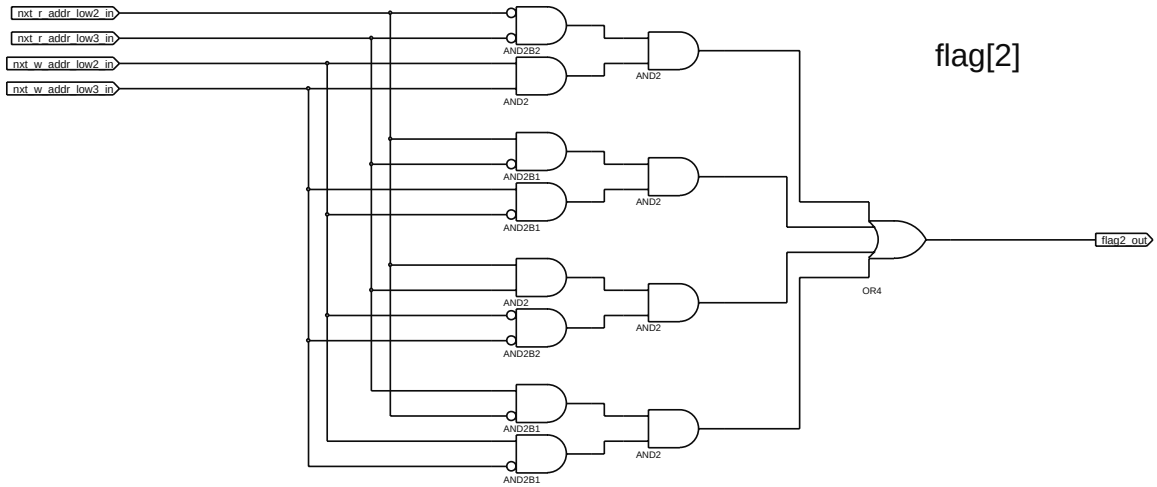

Figure 19
Flag 2 logic

```
assign flag[2] =(((nxt_r_addr_low[3:2] == 2'b00) && (nxt_w_addr_low[3:2] == 2'b11)) |
                ((nxt_r_addr_low[3:2] == 2'b01) && (nxt_w_addr_low[3:2] == 2'b10)) |
                ((nxt_r_addr_low[3:2] == 2'b11) && (nxt_w_addr_low[3:2] == 2'b00)) |
                ((nxt_r_addr_low[3:2] == 2'b10) && (nxt_w_addr_low[3:2] == 2'b01)))
                ? 1'b1 : 1'b0;
```

7.2.3 Synchronization Busses

CEPINT[0:3] and CENINT[0:3] are synchronization busses four bits wide used for synchronizing the FIFO buffer with the deserialized positive and negative edge data once it has been deserialized. The functions of these two blocks are identical except of the input synchronization signals syncp and syncn.. CENINT is dependent upon CEPINT for its initialization. Once the two busses are coordinated; they continuously cycle a high bit along respective wires. CEPINT leads CENINT by 1½ of the master clock cycle. These two busses form hold times for signals in the data path and controls.

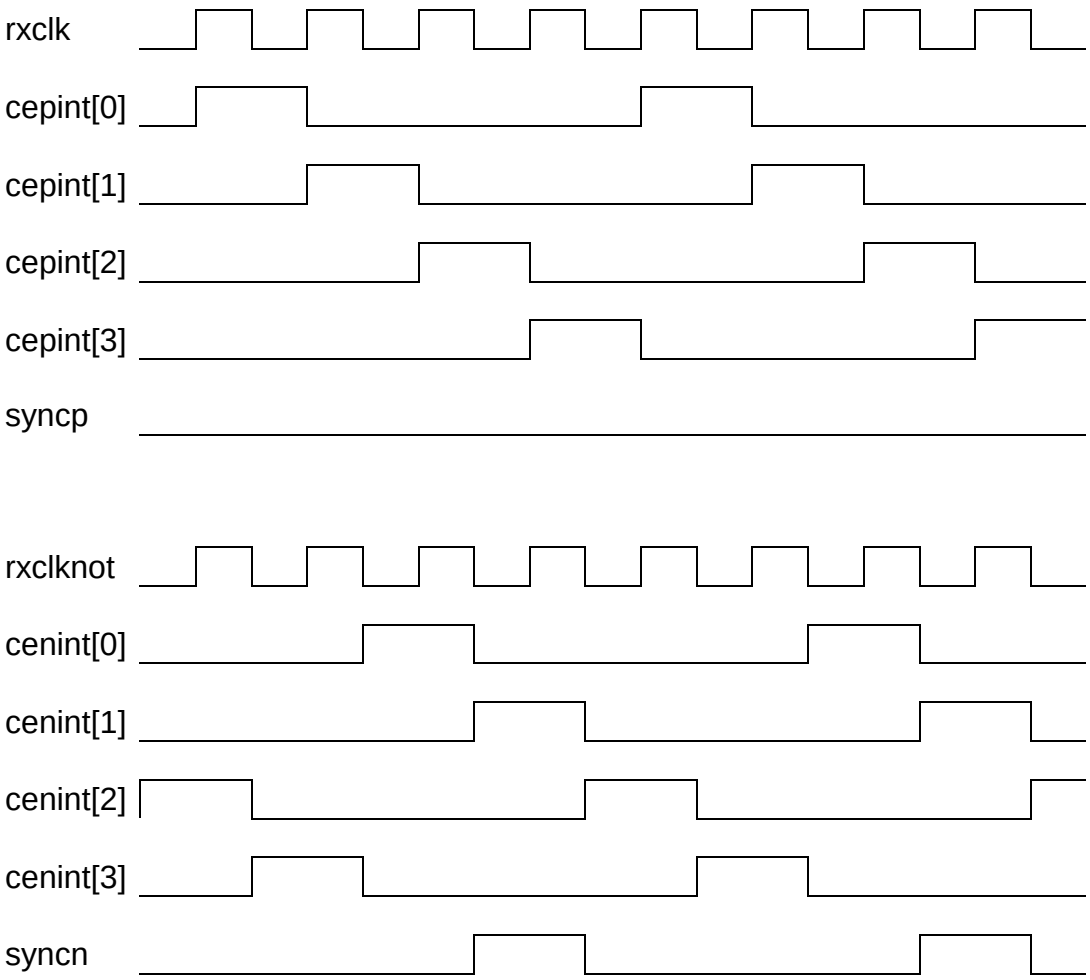

Figure 20
Synchronization buss signals

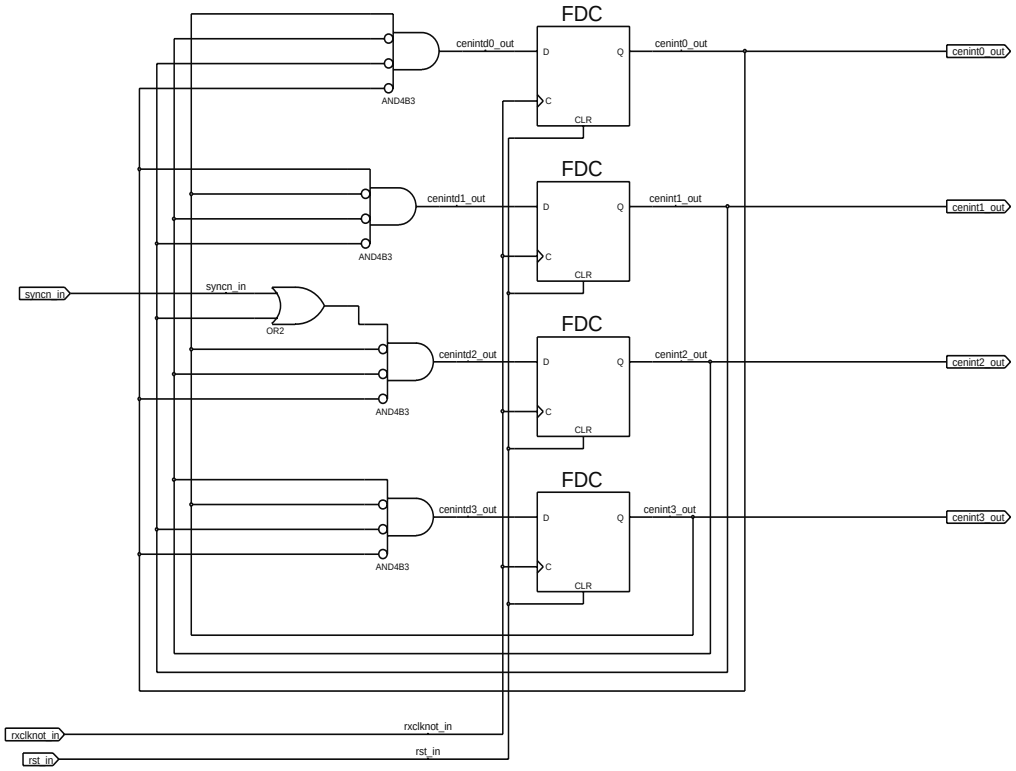

Figure 21
Negative edge data synchronization Bus

```
FDC fd_nint0(.D(cenintd[0]), .C(rxclknot), .CLR(rst), .Q(cenint[0]))/* synthesis RLOC = "X0Y2" */ ;
FDC fd_nint1(.D(cenintd[1]), .C(rxclknot), .CLR(rst), .Q(cenint[1]))/* synthesis RLOC = "X0Y2" */ ;
FDC fd_nint2(.D(cenintd[2]), .C(rxclknot), .CLR(rst), .Q(cenint[2]))/* synthesis RLOC = "X1Y2" */ ;
FDC fd_nint3(.D(cenintd[3]), .C(rxclknot), .CLR(rst), .Q(cenint[3]))/* synthesis RLOC = "X1Y2" */ ;

always @(cepint or syncp)
begin
if (syncp || cepint[1])
        cepintd = 4'b0100 ;
else if (cepint[2])
        cepintd = 4'b1000 ;
else if (cepint[3])
        cepintd = 4'b0001 ;
else
        cepintd = 4'b0010 ;
end //always
```

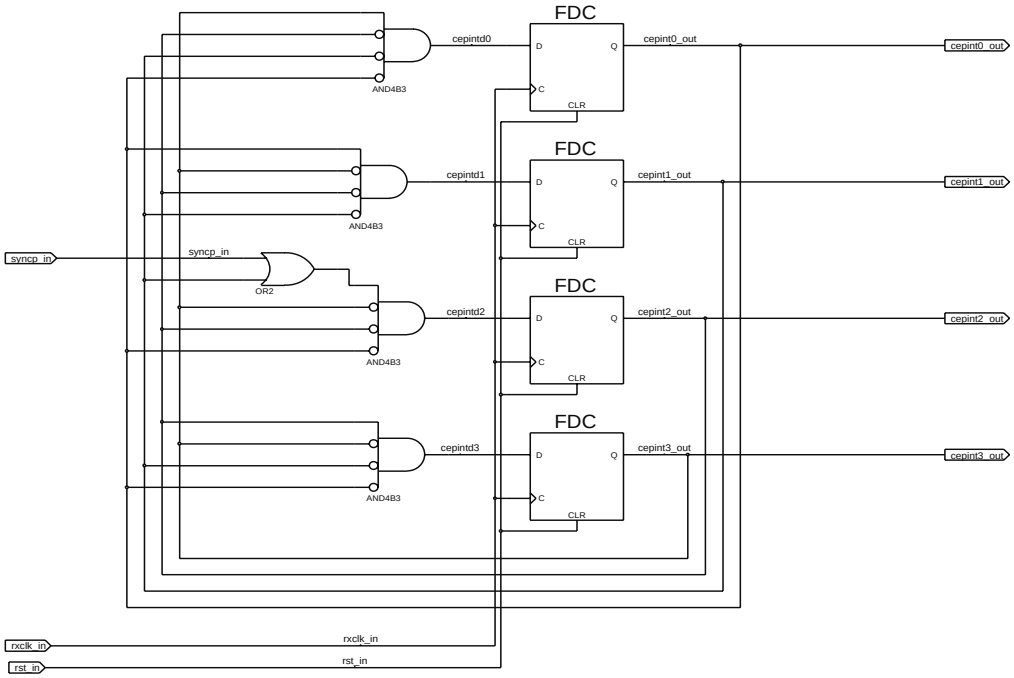

Figure 22
Positive edge data synchronization bus

```
FDC fd_pint0(.D(cepintd[0]), .C(rxclk), .CLR(rst), .Q(cepint[0]))/* synthesis RLOC = "X0Y1" */ ;
FDC fd_pint1(.D(cepintd[1]), .C(rxclk), .CLR(rst), .Q(cepint[1]))/* synthesis RLOC = "X0Y1" */ ;
FDC fd_pint2(.D(cepintd[2]), .C(rxclk), .CLR(rst), .Q(cepint[2]))/* synthesis RLOC = "X1Y1" */ ;
FDC fd_pint3(.D(cepintd[3]), .C(rxclk), .CLR(rst), .Q(cepint[3]))/* synthesis RLOC = "X1Y1" */ ;


always @(cenint or syncn)
begin
if (syncn || cenint[1])
        cenintd = 4'b0100 ;
else if (cenint[2])
        cenintd = 4'b1000 ;
else if (cenint[3])
        cenintd = 4'b0001 ;
else
        cenintd = 4'b0010 ;
end //always
```

7.2.4 Enabling and Reset Signals

The Enabling signals are generated from the synchronization busses since they are the ultimate reference of the system after it has been reset. They provide the ability to apply an external synchronization without reset.

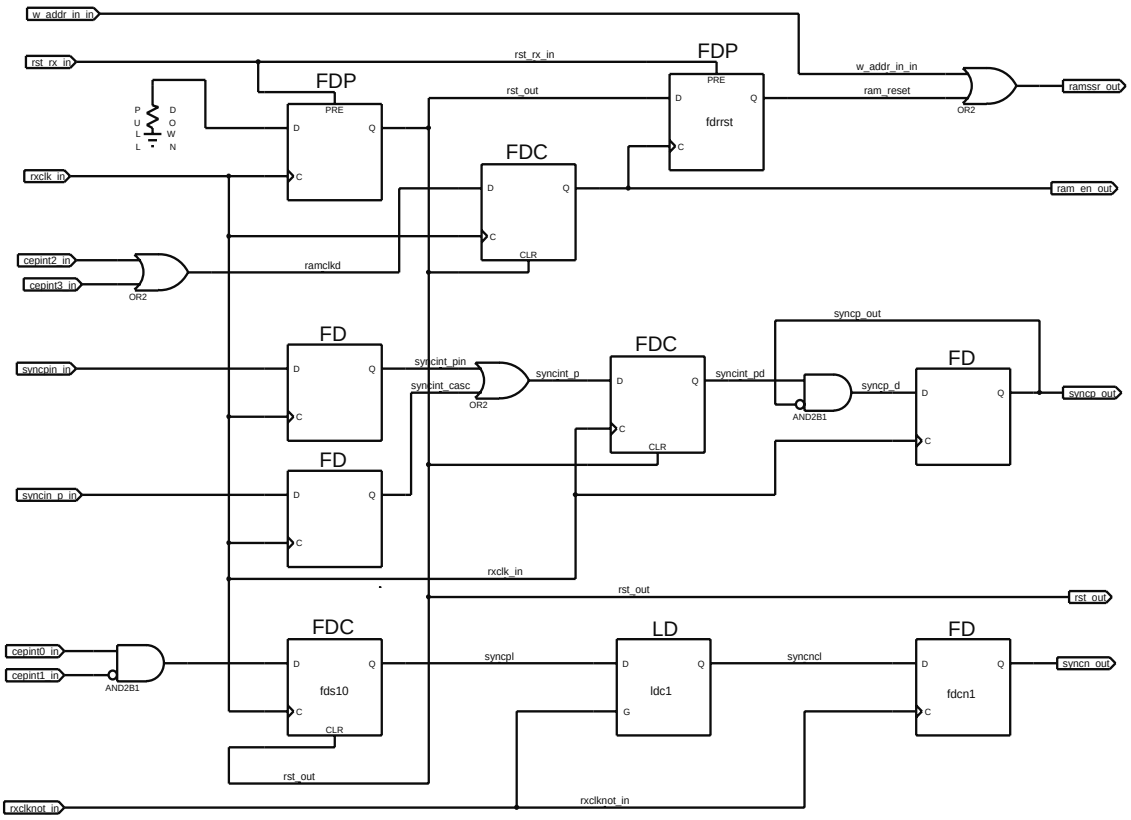

Figure 23
Enable and Reset signals

```
FD fdp4a(.C(rxclk), .D(syncpin), .Q(syncint_pin))/* synthesis IOB = "TRUE" */;
FD fdp4b(.C(rxclk), .D(syncin_p), .Q(syncint_casc))/* synthesis RLOC = "X0Y7" */;
FDC fd_psyo(.D(cepint[1]), .C(rxclk), .CLR(rst), .Q(syncoutp))/* synthesis RLOC = "X1Y0" */ ;
FDC fd_rcb (.D(ramclkd),   .C(rxclk), .CLR(rst), .Q(ram_en))/* synthesis RLOC = "X1Y3" */ ;
FDP fdlrst(.D(low), .C(rxclk),  .PRE(rst_rx), .Q(rst))/* synthesis RLOC = "X1Y7" */ ;
FDP fdrrst(.D(rst), .C(ram_en), .PRE(rst_rx), .Q(ram_reset))/* synthesis RLOC = "X0Y0" */ ;
FDC fds8 (.C(rxclk), .D(syncint_p), .CLR(rst), .Q(syncint_pd))/* synthesis RLOC = "X1Y4" */;
FDC fds9 (.C(rxclk), .D(syncp_d),   .CLR(rst), .Q(syncp))/* synthesis RLOC = "X1Y3" */;
FDC fds10(.C(rxclk), .D(syncpl_d),  .CLR(rst), .Q(syncpl))/* synthesis RLOC = "X1Y4" */;
LD ldc1(.G(rxclknot), .D(syncpl), .Q(syncl))/* synthesis RLOC = "X1Y6" */;
FD fdcn1(.C(rxclknot), .D(syncl), .Q(syncn))/* synthesis RLOC = "X1Y5" */;
assign syncint_p = syncint_pin | syncint_casc ;
assign syncp_d = syncint_pd & ~syncp ;
assign syncpl_d = cepint[0] & ~cepint[1] ;
assign ramclkd = cepint[2] | cepint[3] ;
```

8.0 Modifications

The next phase of serial to parallel design is be tailor the needs of the serial to parallel module to the needs of the PMFE. The PMFE requires two clock signals form the FPGA. The major changes to the Bottom module is to change all the Flip Flops to include clock enable inputs that can be controlled with the Clk-Data signal to not latch data form the PMFE into the FPGA on fifth clock cycle when the data is not valid.

The Middle module is requires few modifications since its main function is to tile four of the lower level modules together. Since the PMFE has 8 channels at its output, this module only needs to be changed to reflect the number of control signals that are and the data width for matching the FE output. Matching the data with of the FE is not a difficult task since it requires only two the lower level modules. The aspect of eliminating the fifth clock cycle data is the main issue and is a constraint of the Bottom module.

The Top module has not been a focus; however, are some changes that will need to be made. The Top-level module is written with the intention that it read in data serially, convert it to parallel data for the FIFO buffer then convert it back to serial data. This data is to be displayed on a demo board with LED's. Thus there are some changes that can be noted now.

- The call to serdes16b_8to1, transmitter, will need to be removed along with any attachments to it since the data out of the FIFO will be processed.
- The display LED's are not necessary for our design and can be removed.
- Any subsequent changes to the data path configuration will need to be proper mapped to higher-level modules.
- Any added controls for the fifth clock cycle will need to be supplied as ports to higher-level modules.

9.0 Conclusion

The Particle Tracking Silicon Microscope system is a constantly evolving system that will one day allow highly accurate dosimetry to be accomplished with a relatively low cost system. This is just an the beginning of the first PTSM and many more innovations on the original design are ahead.

Acronyms
TOT: Time Over Threshold
PTSM: Particle Tracking Silicon Microscope
MCM: Multi-Chip Module
PGA: Field Programmable Gate Array
ROC: Readout Controller
FE: Frontend
DSSD: Double Sided Silicon Detector
CMOS: Complimentary Metal Oxide Semiconductor
DAQ: Data Acquisition
DAC: Digital to Analog Converter
LVDS: Low Voltage Differential Signaling
FPGA Features:
CLB: Configurable Logic Block
DDR: Double Data Rate
LUT: Look Up Table
DCM: Digital Clock Manager

PTSM system:
Hardware
Double Sided Silicon Detector, DSSD
4 Particle Microscope Frontend Chips, PMFE
1 Xilinx VirtexII Field Programmable Gate Array, FPGA
1 Programmable Read Only Memory, PROM
LVDS to CMOS translator Board
National Instruments Data Acquisition, DAQ, Card 6534
National Instruments Digital to Analog Converter, DAC Card 6703
Data Acquisition Computer
Simulation Computer

2.2 Software
National Instruments Lab View
Xilinx Integrated Software Environment, ISE 4.2
Modeltech Modelsim Xilinx Edition, MXE

Reference Documents:

1. High-Speed Data Serialization and Desialization (840Mb/s)
   http://www.xilinx.com/xapp/xapp265.pdf
2. Using Double Data Rate (DDR) I/O
   http://www.xilinx.com/products/virtex/handbook/ug002_ch2_ddr.pdf
3. Using Digital Clock Managers (DCMs)
   http://www.xilinx.com/products/virtex/handbook/ug002_ch2_dcm.pdf
4. Using Block Select Ram Memeory
   http://www.xilinx.com/products/virtex/handbook/ug002_ch2_blockram.pdf
5. Self-Addressing FIFO http://www.xilinx.com/xapp/xapp291.pdf
6. Architecture Virtex-II Overview
   http://www.xilinx.com/products/virtex/handbook/ug002.pdf
7. PTSM System Hartmut Sadrozinski Febuary 4,2002
8. Look Up Tables, LUTs
   http://www.xilinx.com/products/virtex/handbook/ug002.pdf