CALIFORNIA STATE UNIVERSITY, NORTHRIDGE

DESIGN AND DEVELOPMENT OF MEMORY MAPPING FOR AN IMAGE DATA ACQUISITION SYSYTEM

A graduate project submitted in partial fulfillment of the requirements

For the degree of Master of Science in

Electrical Engineering

By

Sahar Sadeghi

May 2012

The Graduate Project of Sahar Sadeghi is approved:

Ramin Roosta, Ph.D.	Date
Ali Amini, Ph.D.	Date
Nagi M El Naga, Ph.D., Chair	Date

California State University, Northridge

ACKNOWLEDGMENTS

First and foremost, I offer my sincerest gratitude to my project advisor and my graduate coordinator, Dr. Nagi El Naga, who has supported me throughout my project with his patience and knowledge whilst allowing me to work in my own way. I attribute the level of my Master’s degree to his encouragement and effort and without him this project would not have been completed or written. One simply could not wish for a better or friendlier advisor.

Besides my advisor, I would like to show my gratitude to the rest of my project committee, Dr. Ramin Roosta, and Dr. Ali Amini, for their support and encouragement.

I would also like to thank my other professors at California State University, Northridge, for their support and all they have taught me during my Master’s program.

Last but not the least, I would like to thank my family and friends for their unconditional support throughout all my studies and life.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

DESIGN AND DEVELOPMENT OF MEMORY MAPPING FOR AN IMAGE DATA ACQUISITION SYSYTEM

By

Sahar Sadeghi

Master of Science in Electrical Engineering

Data acquisition systems typically convert analog signals into digital values for processing. In this graduate project, a memory mapping process for an image data acquisition system, which is built out of three levels of arrays of FPGA, has been developed. The system has the capability to sample the outputs of large number of photo-detectors with high resolution and at a very high frequency. The received data is sampled, digitalized and passed through a three levels of FPGAs to a fiber channel switch which provides the final high speed serial output. The array has 256 × 256 of photo detectors. Each 16 detectors of the array are grouped to make a 4× 4 photo detector array module. A sample from all detectors will represent one frame. To form one image, 16 frames are required. Therefore, all the detectors of the array are sampled simultaneously sixteen times to form the 16 frames. The analog signal sample out of each detector is converted into 12-bit digital value. BRAM[1] (16× 49154) is used to store the collected data in each level and define the address of each detector. The time required to sample, digitize and process the collected data through the three levels of FPGA is 0.774ms.

[1] Block Memory RAM

CHAPTER 1: INTRODUCTION

1.1. INTRODUCTION TO DATA ACQUSITION SYSTEMS

Data acquisition system is the process of sampling analog signals that measure the physical conditions and converting the resulting samples into digital numeric values that can be manipulated by a computer [1]. Sending data to a computer is used by various methods, such as: USB, SPI, PCI Express, I2C, Wi-Fi and Ethernet. Depending on the application, the complexity of data acquisition system is different. In some applications such as medical, space and military the accuracy of the system is the most important factor rather than the cost of the system.

Data acquisition systems typically convert analog waveforms into digital values for processing. The components of data acquisition systems include:

- Sensors that convert physical parameters to electrical signals.
- Signal conditioning circuitry to convert sensor signals into a form convertible to digital values.
- Analog-to-digital converters, which convert conditioned sensor signals to digital values [1].

The parameters measured can be shown numerically whereas their relationship can be displayed graphically as a curve on the screen [2].

The storage systems of high speed and large capacity data acquisition is widely used in aerospace, military, telecommunications, medical and other fields. Ultra high speed and high capacity features have become the actual need and developing direction of the data acquisition, and semiconductor devices have become the main storage medium. The various types of memories in FPGA with the advantage of high reliability, low power consumption, long life, high capacity and adapting to the harsh environment, anti-vibration, anti-shock, high and low temperature properties have been developed rapidly [3].

1.2. OBJECTIVE OF THIS PROJECT

The objective of this graduate project is to design and develop a memory mapping for an image data acquisition system which is built of arrays of FPGAs connected to a PC. The system has the capability to samples the outputs of a large number of photo detectors with high resolution at a very high frequency. The received data is digitalized and passed through a series of FPGAs to a fiber channel switch which provides the final high speed serial output. The array has 256 × 256 of photo detectors. Each 16 detectors of the array are grouped to make a 4× 4 photo detector arrays module. Therefore, the array of 256 ×256 detectors becomes an array of 64 × 64 modules which consists of 4096 modules.

A pulse comes to initiate sampling at 1 kHz, and then 16 samples per detector at 12 MHz are sampled. All the detectors of the array are sampled simultaneously to form the 16 frames. The analog signal sample out of each detector is converted into 12-bit digital signal. Data will be grouped into a single data stream and transferred to the processing computer at about 2 Gbyte/sec. After sampling the 16 frames, there would be a significant amount of time to transfer the collected data through the three levels of FPGAs to the processing computer, before initiating the next sampling process.

In this project, one FPGA serves as a master controller to control, the data acquisition system, storing and mapping the data. The top level block diagram of the designing system is shown in Figure 1.1.

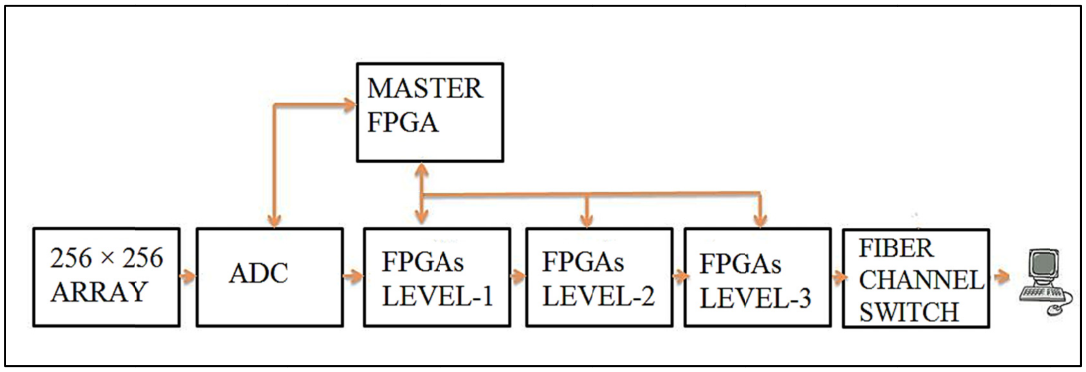


Figure 1.1. Top level block diagram of the designing system.

This project is part of a group effort to design and develop the architecture of the system along with memory mapping of the data. In this project, featuring the design and development of memory mapping process for the image acquisition system are emphasized.

1.3. PROJECT OUTLINE

A complete description of the memory mapping design and development for an image data acquisition system is presented in this project report. This project consists of 7 chapters. Chapter 1 gives the introduction to this project. It describes the importance of the applications of data acquisition system. Moreover, the objective of this project is defined in this chapter. The top level architecture of the image acquisition system and its specifications are presented in Chapter2.

The Analog to Digital Converter is introduced in Chapter 3. This chapter includes features and functional block diagram of the ADC. Chapter 4 is the main chapter of this project, and presents the basic design of the image acquisition system. This chapter provides complete design description for each of the three levels of the architecture. Furthermore, Chapter 5 includes timing and data flow of the sampled data through each level. Chapter 6 explains how to trace and determine the location of the data collected from each detector in the three levels of FPGAs. Chapter 7 includes VHDL modeling of the hardware component required for the system. Finally, the results and conclusions are presented in Chapter 8.

CHAPTER 2: TOP LEVEL ARCHITECTURE

2.1. SPECIFICATION ANALYZES

This chapter describes the specifications for the image acquisition system. The first requirement of an array of 256 × 256 photo detectors spaced 5mm apart is shown in Figure 2.1. The total number of photo detectors is 65536 (256 × 256). The 5mm spacing between detectors results in an array with an approximate area 1.28m × 1.28m.

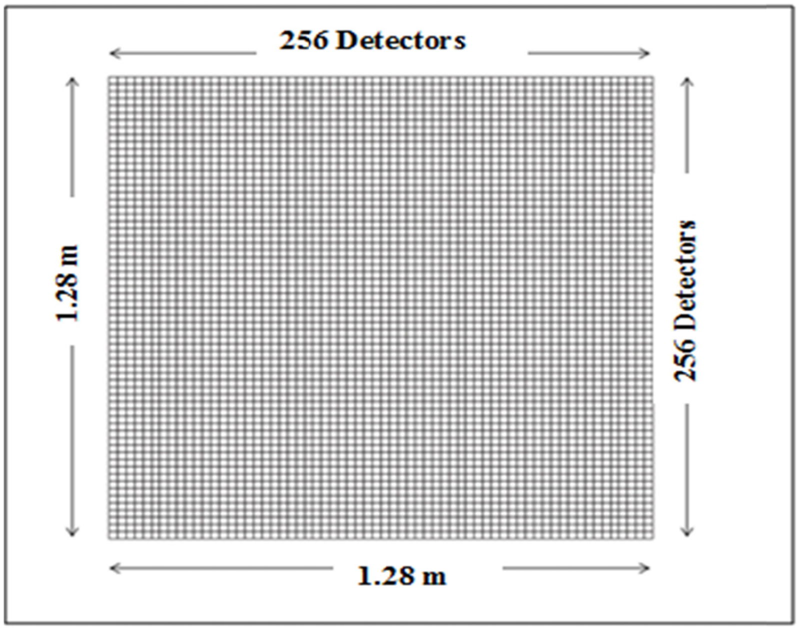


Figure 2.1. Array of 256 × 256 photo detectors spaced at 5mm.

The next specification describes each sixteen detectors of the array are group to make a 4× 4 photo detector arrays module. Hereby, the arrays of detectors is constructed out 4096 of three module arranged in a form of 64 × 64 array as shown in Figure 2.2.

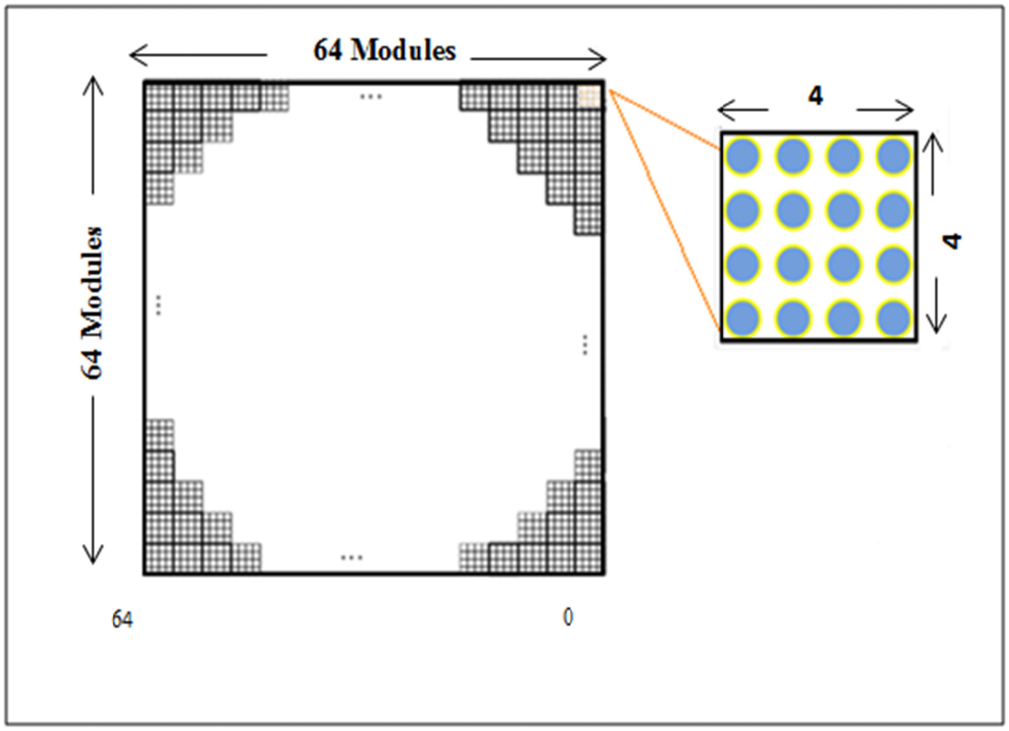


Figure 2.2. The 4 × 4 photo detector array module (64 × 64 = 4096 modules).

The next specification is that a pulse comes to initiate sampling at 1 kHz, and then 16 samples per detector at 12 MHz are sampled. All the detectors of the array are sampled simultaneously to form the 16 frames as shown in the Figure 2.3.

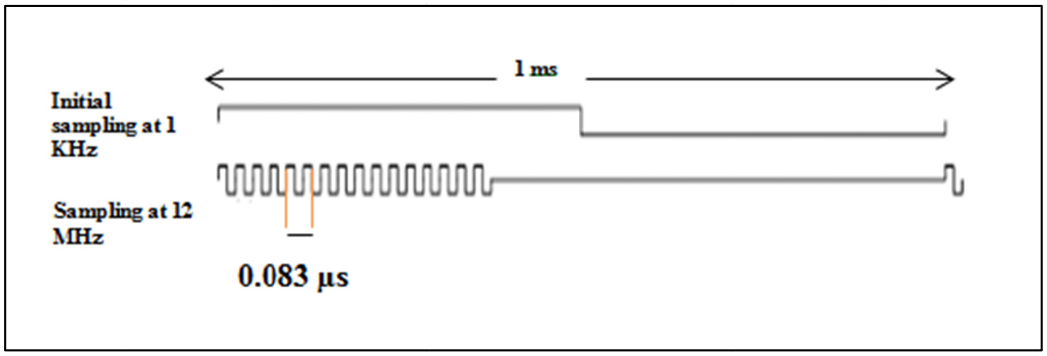


Figure 2.3. The pulse initiates sampling.

The amount of data received per sample pulse is:

$$\overbrace{\underbrace{256 \times 256}_{\text{Total photo detectors}} \times 12}^{\text{One frame}} \times \overset{\text{16 Frames}}{\overset{\downarrow}{16}} = 25{,}582{,}912 \quad \text{bits/pulse}$$

Each photo detector collects 12-bit samples; a total of 12,582,912-bits must be read out before the next sampling pulse is initiated.

2.2. TOP LEVEL BLOCK DIAGRAM ARCHITECTUR

This architecture is constructed by three levels of FPGAs to transmit data from photo detector array to the fiber channel switch. Each 4×4 module is connected to a single Xilinx Spartan-3A FPGA. In the first level of FPGA, the 4096 Spartan3A are used. For the second level of FPGA, the 256 Spartan3A are used. The final stage contains the 16 Xilinx Virtex-6 FPGAs. These powerful FPGAs use PCI Express transfer protocol to output data through the fiber channel switch providing the final serial output. The top level block diagram architecture is shown in Figure 2.4.

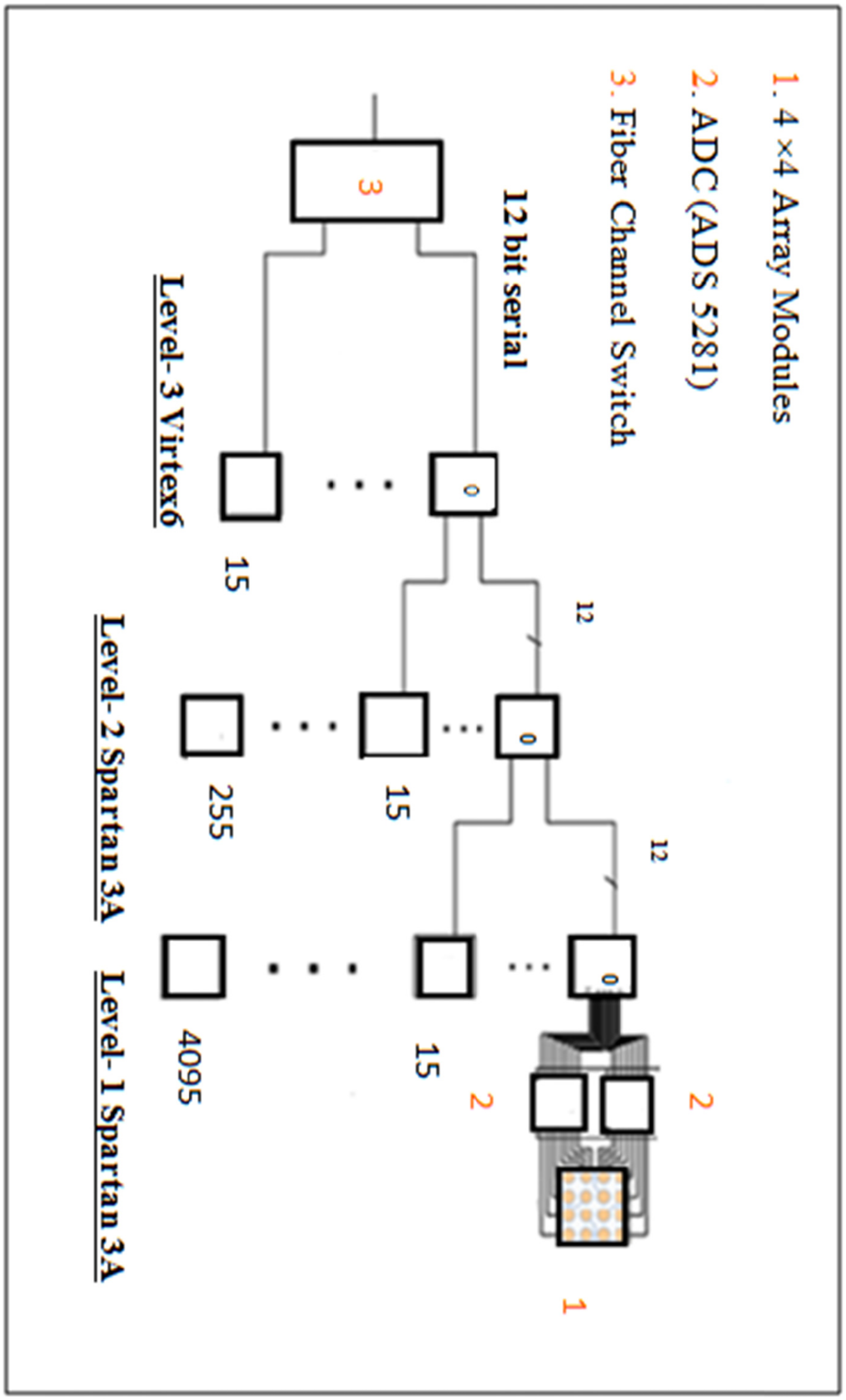


Figure 2.4. Top level blocks diagram architecture.

CHAPTER 3: ANALOG TO DIGITAL CONVERTER

ADS5281 is a 12-bit high- performance, low-power and an octal channel Analog-Digital Converter is selected to be used in this project.

The physical specification of the ADS5281 is:

- Resolution: 12-bits
- Sample rate: 50 msps
- Channel: 8
- Clock inputs: min 10 msps / max 50 msps
- Digital outputs: min 10 MHz / max50 MHz
- Size: 9mm × 9 mm

The ADC uses the SPI serial interface on pin chip select (CS) and serial interface clock (SCLC) along with serial interface data (SDATA) for initialization.

All ADCs are initialized simultaneously. FPGAs will initialize and control the ADCs via the serial interface. Serial peripheral interface (SPI) logic and timing is implemented on the master controller FPGA and interfaced with ADC [4].

Master FPGA will initialize and control the ADCs and slaves FPGAs via the serial interface while all ADCs are initialized simultaneously. Master FPGA send a start signal acquisition, which will activate the CS and BUSY signal; when CS is high, ADC starts sampling; once the sample is over, the CS goes low then it reads out the eight channel data in sequence. The functional block diagram of Analog to Digital Converter ADS- 5281 is shown in Figure 3.1.

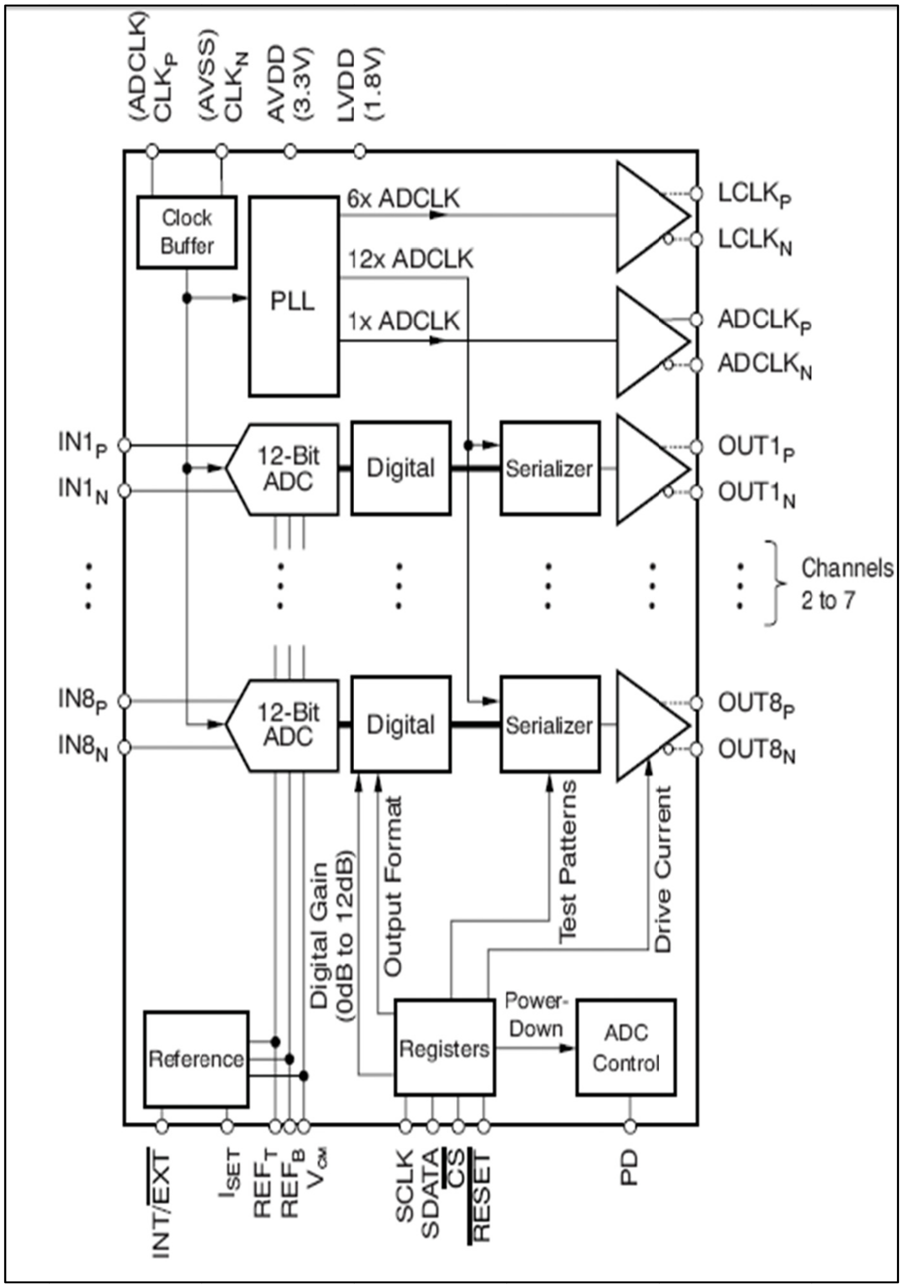


Figure 3.1. Functional Block Diagram.

After the device has been powered up, the following registers must be written (in the exact order listed below) through the serial interface as part of an initialization sequence. [4]

	ADDRESS	DATA (HEX)
Initialization Register1	03	0002
Initialization Register2	01	0010
Initialization Register3	C7	8001
Initialization Register4	de	01C0

Table 1. Initialization Registers.

The ADS528x has a set of internal registers that can be accessed through the serial interface formed by pins CS (chip select, active low), SCLK (serial interface clock), and SDATA (serial interface data). When CS is low, the following actions occur:

- Serial shift of bits into the device is enabled
- SDATA (serial data) is latched at every rising edge of SCLK
- SDATA is loaded into the register at every 24th SCLK rising edge

If the word length exceeds a multiple of 24 bits, the excess bits are ignored. Data can be loaded in multiples of 24-bit words within a single active CS pulse. The first eight bits form the register address and the remaining 16 bits form the register data. The interface can work with SCLK frequencies from 20MHz down to very low speeds (a few hertz) and also with a non-50% SCLK duty cycle. After all registers have been initialized to their default values through a RESET operation, the registered detailed in the table1 must be written. This process must be done after every hardware or software RESET operation in order to reconfigure the device for the best mode of operation [4]. The serial interface timing of ADC 5281 is shown in Figure 3.2.

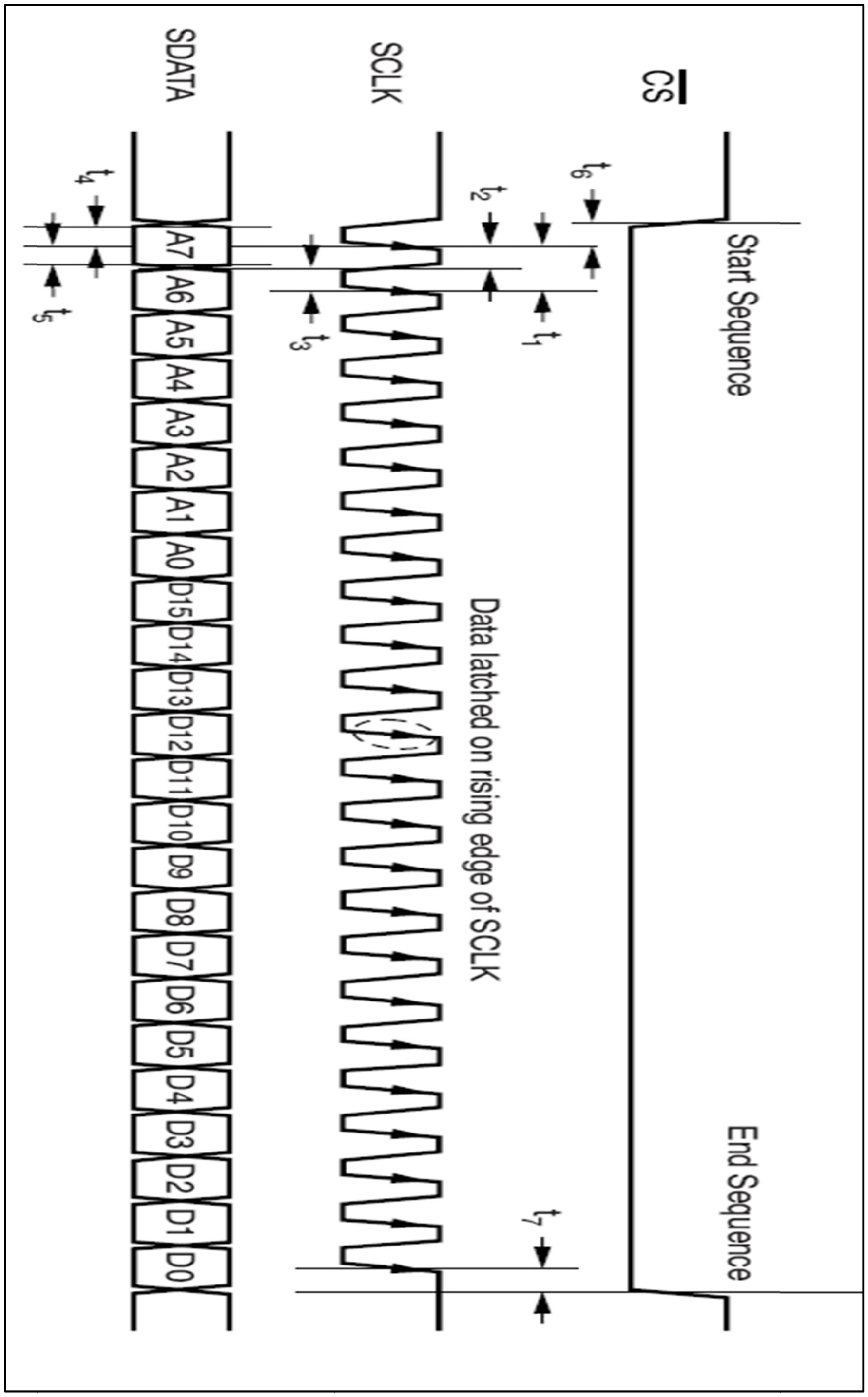


Figure 3.2. Serial Interface Timing. [4]

CHAPTER 4: DESIGN AND DEVELOPMENT OF MEMORY MAPPING

4.1. FROM ADC TO THE FIRST LEVEL OF FPGAS

The presented architecture is designed based on three levels FPGAs; in this design the first level has 4096 Spartan3A - XCS3S1400A and each ADC has a serial output of eight (12- bits) samples to this level.

The Extended Spartan-3A family of Field-Programmable Gate Arrays (FPGAs) solves the design challenges involved in high volume and cost-sensitive electronic applications. With 12 devices ranging from 50,000 to 3.4 million system gates, the extended Spartan-3A family provides a broad range of densities and packaging options, integrating DSP MACs and low total system cost while increasing functionality. The extended Spartan-3A family includes the Spartan-3A device and the higher density Spartan-3A DSP device. In addition, it also includes the nonvolatile Spartan-3AN device, which combines leading-edge FPGA and flash technologies to provide a new evolution in security, protection and functionality ideal for space-critical or secure applications. The extended Spartan-3A family improves system performance and reduces the cost of configuration. These enhancements combined with proven 90 nm process technology, deliver more functionality and bandwidth per dollar. Base on of its exceptionally low cost, the extended Spartan-3A family is ideally suited for a wide range of consumer electronics' applications including broadband access, home networking, display/projection, and digital television equipment. The extended Spartan-3A family is a superior alternative to mask-programmed ASICs. FPGAs avoid the high initial cost, lengthy development cycles, inherent inflexibility of conventional ASICs, and permit field design upgrades. [6]

A sample from all detectors will represent one frame. To form one image, 16 frames are required. Therefore, all the detectors of the array are sampled simultaneously sixteen times to form one image. The analog signal sample out of each detector is converted into 12-bit digital value. Then, the output of each ADC is simultaneously received by level-1 Spartan3A FPGAs. Each Spartan 3A will store 3072 bits (12 “bits per sample” × 16 “samples collected × 16 “ photo detector “) as shown in Figure 4.1.

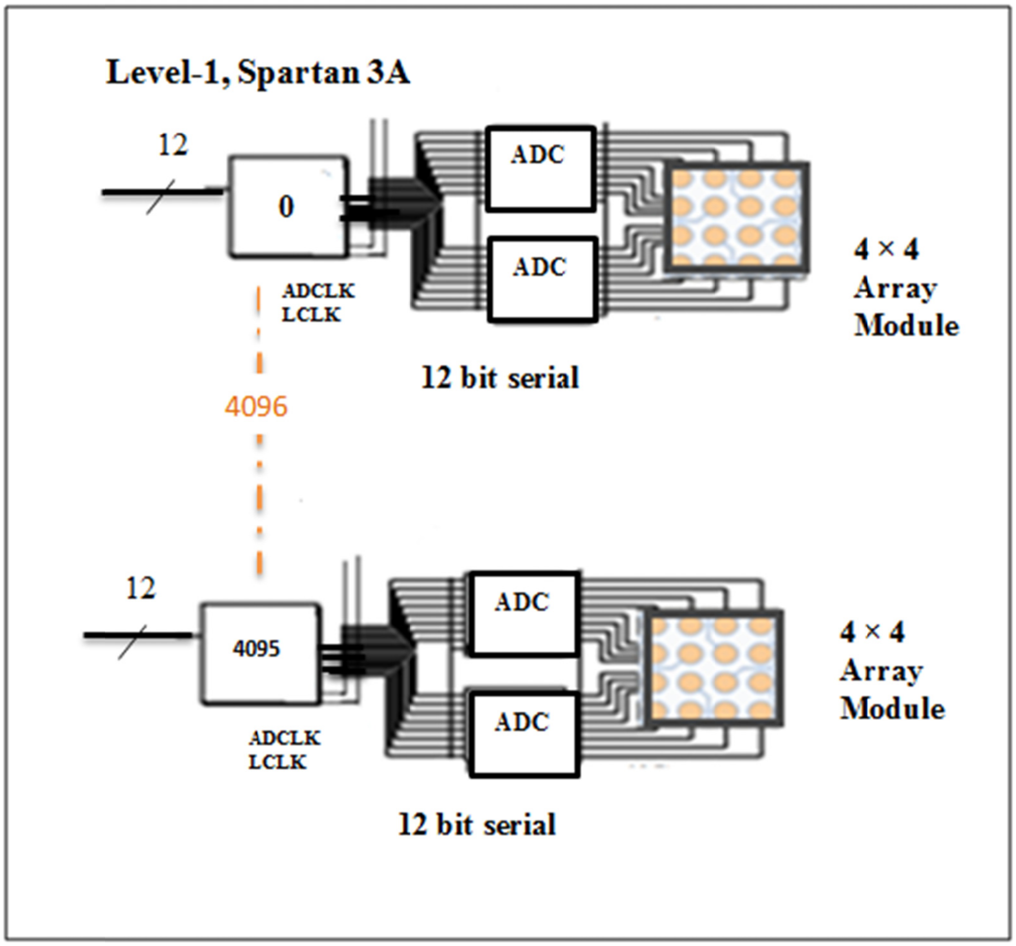


Figure 4.1. Top level-1 block diagram.

The timing calculation per sample pulse and timing sample is calculated as below:

- Data per sample pulse = 12 (bit/sample) × 16 (photo detector) × 16 (samples collected) = 3072
- Required time to sample and digitalize one frame = 16 (samples collected) × 12 (bit /sample) × 83.33ns= 15.936 µs ≅ 16 µs

On the rising edge of Clk_1, the first bit of the 16 photo detectors of all 4096 (4×4) module are read by the FPGA XILINX SPARTAN3A.

It is important to know that the ADC has 12Clk cycle latency for the first sampling and will continuously retrieve 12-bit data with no Clk cycle latency. This amount of delay is required for initialization four registers described in Table-1[5]. Therefore, ADCLK requires a 12Clk cycle for the first sample. The AND gate and 2-to-1 Multiplexer is used to make the first 12Clk cycle delay for ADCLK. When the output of the AND gate becomes 1, line_1 of the multiplexer is selected and the ADCLK will go to the 12-bit delay counter. As a result, the output of this delay is connected to Clk_1.

If line_0 of the multiplexer is selected, the ADCLK will be connected directly to Clk_1 with no delay. When Clk_1 clocks, the first 12-bit detector frame of the 4×4 module is read. On the first rising edge of Clk_1, the 16-to-1 D-Multiplexers (Xilinx D4_16E) select the first frames (frame 1-16) for the 12-bit serial data of the first photo detectors (detectors1-16) of the all 4096 modules.

The selected frames with 12-bit serial data are stored and converted to the 12-bit parallel data by using the Serial-In-Parallel-Out Shift Registers (Xilinx SR16RE).

Figure 4.2 describes the above process for a 4×4 module. Figure 4.2 and Figure 4.4 show only one 4x4 module; note that there is a total of 4096 (4×4) modules.

- Total number of D-Multiplexers for one (4×4) Module is:

  1 (16-to-1D-Multiplexer × 16 (photo detector) = 16

- Total number of D-Multiplexers for level-1 is:

  4096((4×4) Module) × 16(16-to-1 D-Multiplexer) = 65,536

- Total number of SIPO Shift Register for one 4×4 Module is:

  16 (Serial-In-Parallel–Out Shift Register for each detector) ×16 (photo detector) = 256

- Total number of SIPO Shift Register for level-1 is:

  4096((4×4) Module) × 256 (SIPO Shift Register) = 1,048,576

Since the first bit of each frame (frame 0 - 15) is read entirely at the same rising edge of the ADCLK, The 4- bit counterI (Xilinx CB4RE) shows in Figure 4.2 is the only controller for all D-Multiplexers in this level.

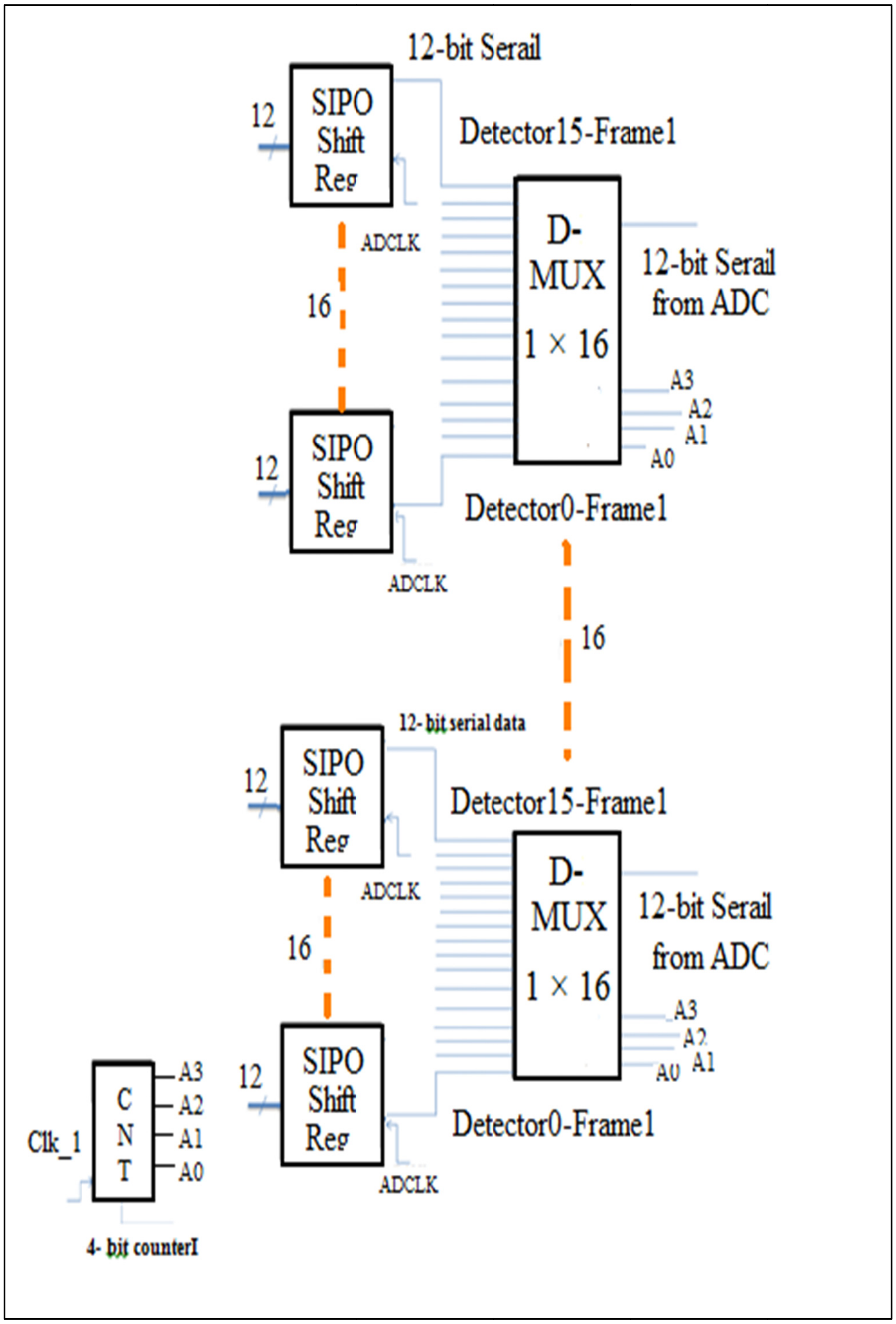


Figure 4.2. 12-bit serial data from D-Multiplexer to the SIPO shift register.

The ADCLK for the FPGAs should be clocked with 12 clock cycle latency for the first sampling. The ADCLK is the frame Clk at a 12 MHz frequency while the LCLK (system Clk) is the stream bit Clk at 144 MHz frequency. The first 4-bit counterI displayed in Figure4.2 is controlled by the Clk under ADCLK and the system CLK.

On the first rising edge of Clk_1, a 12-bit serial data is read by 16-to-1 D-Multiplexers. The counter counts from 0 to 15 and each count takes 12Clk cycles to retrieve the first 12-bit data. At the second count, the second 12-bit data is retrieved and so forth. This process takes 12Clk cycle; 192 Clk cycles are required for a total of sixteen frames. This concept is shown in Figure 4.3.

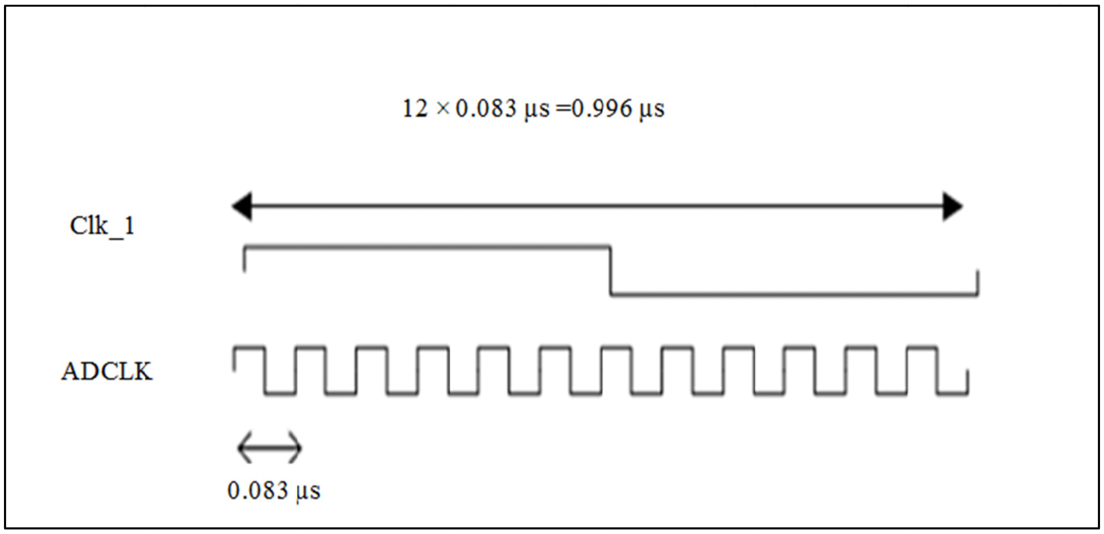


Figure 4.3. The counterI timing analyzer for first frame of detector 1.

The first bit of the first frame will be ready on the output of D-Multiplexer after a 12Clk cycle. The 12-bit Serial-In-Parallel-Out Shift Registers takes a 12Clk cycle to convert the 12-bit serial data to parallel. 24Clk cycle latency is required for the 12-bit serial data to be read from ADC and converted to 12-bit parallel data.

- 12 Clk cycle (SIPO Shift Register) + 12(Clk cycle delay for the first sample) = 24 Clk cycle

In the next step, the 16-to-1 Multiplexers (Xilinx M16_1E) are used to select the 12-bit parallel data from each frame (frame 0-15) in order to send this data to the next step. Figure4.4 shows this process.

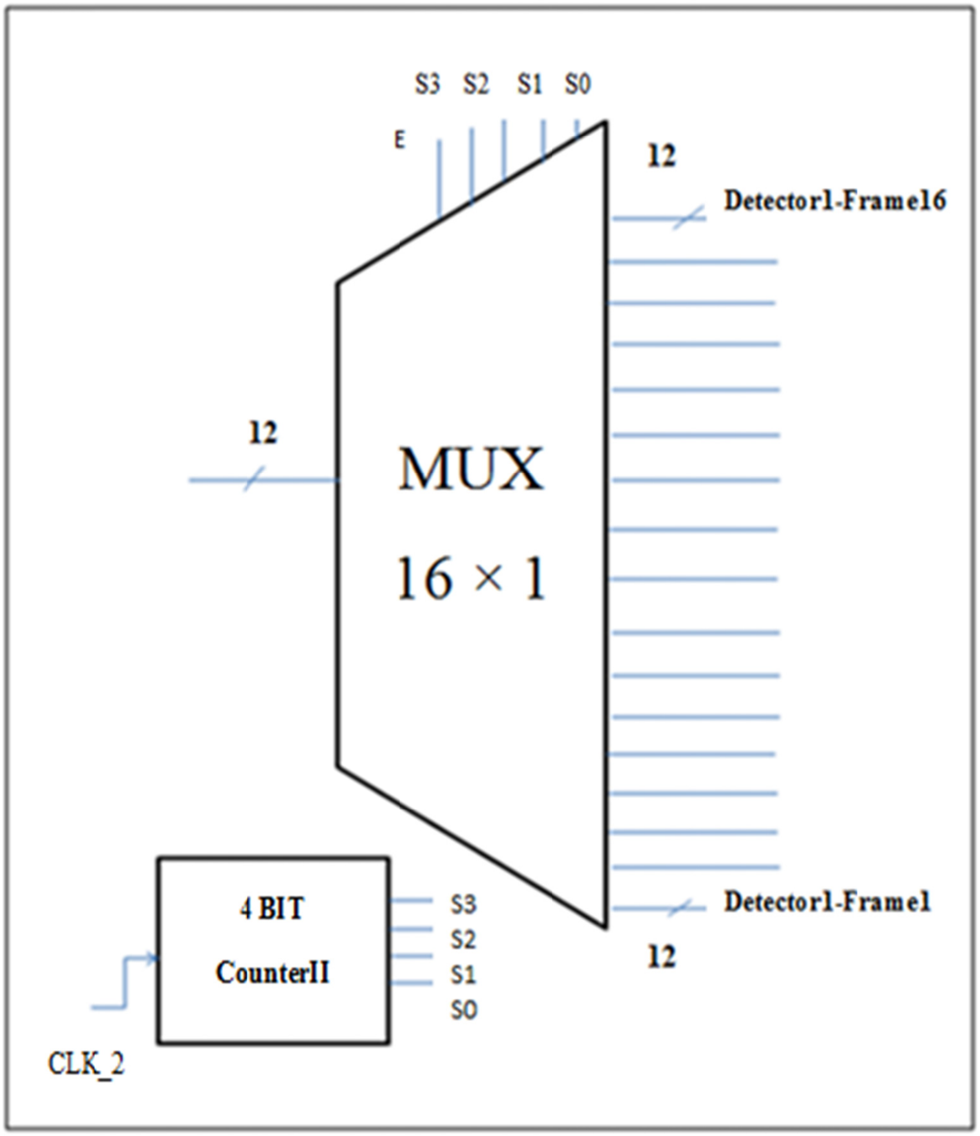


Figure 4.4. The 16–bit Multiplexer selects the frame for one specific detector.

- Total number of 16-to-1 Multiplexers for a 4×4 Module is:

1(16-to-1 Multiplexer × 16 (photo detector)) = 16

- Total number of 16-to-1 Multiplexers for level-1 is:

4096((4×4) Module) × 16(16-to-1 Multiplexer) = 65,536

Since the first bit of each frame (frame 0- 15) is read entirely at the same rising edge of the Clk_2, The 4-bit counterII (Xilinx CB4RE) shown in Figure 4.4 is the only controller for all 16-to-1 Multiplexers in this step.

Figure 4.5 shows that the first 12-bit parallel data will be ready after 24 Clk cycle on the input of multiplexer .Therefore, the Clk_2 for all of these multiplexer should have started after 38 ADCLK. On the rising edge of the 38th ADClk the first frame is selected by the multiplexer and this process should take 16 cycles to read the last frame. Figure 4.5 shows timing analyzes for clk_2 and Figure 4.6 shows all step in Figure 4.2 & 4.

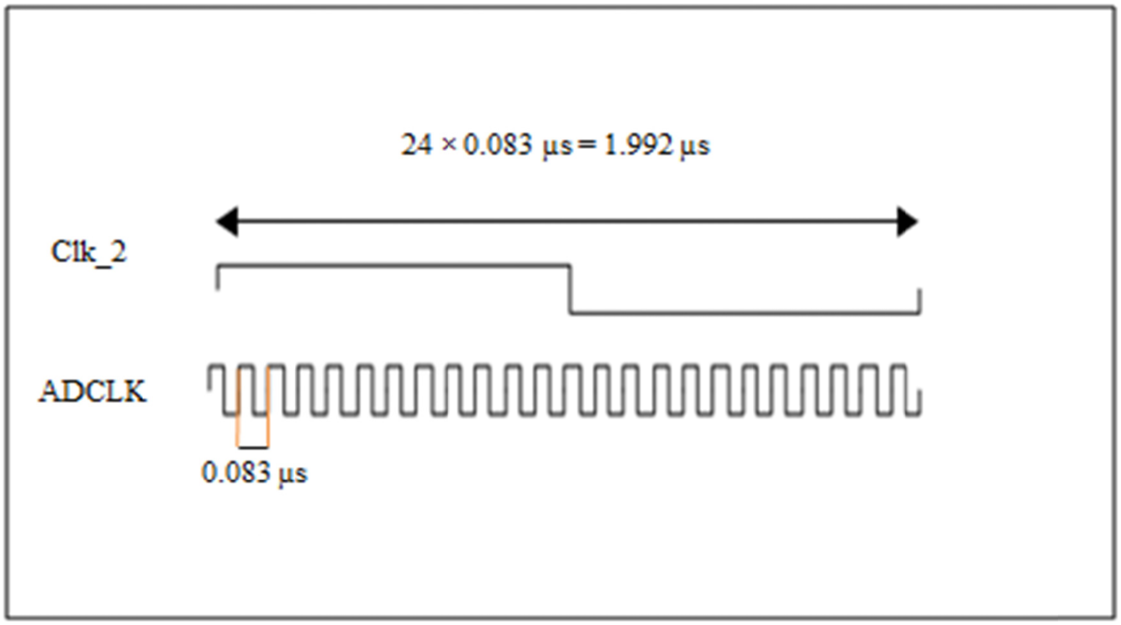


Figure 4.5. Timing analyzes for Clk_2.

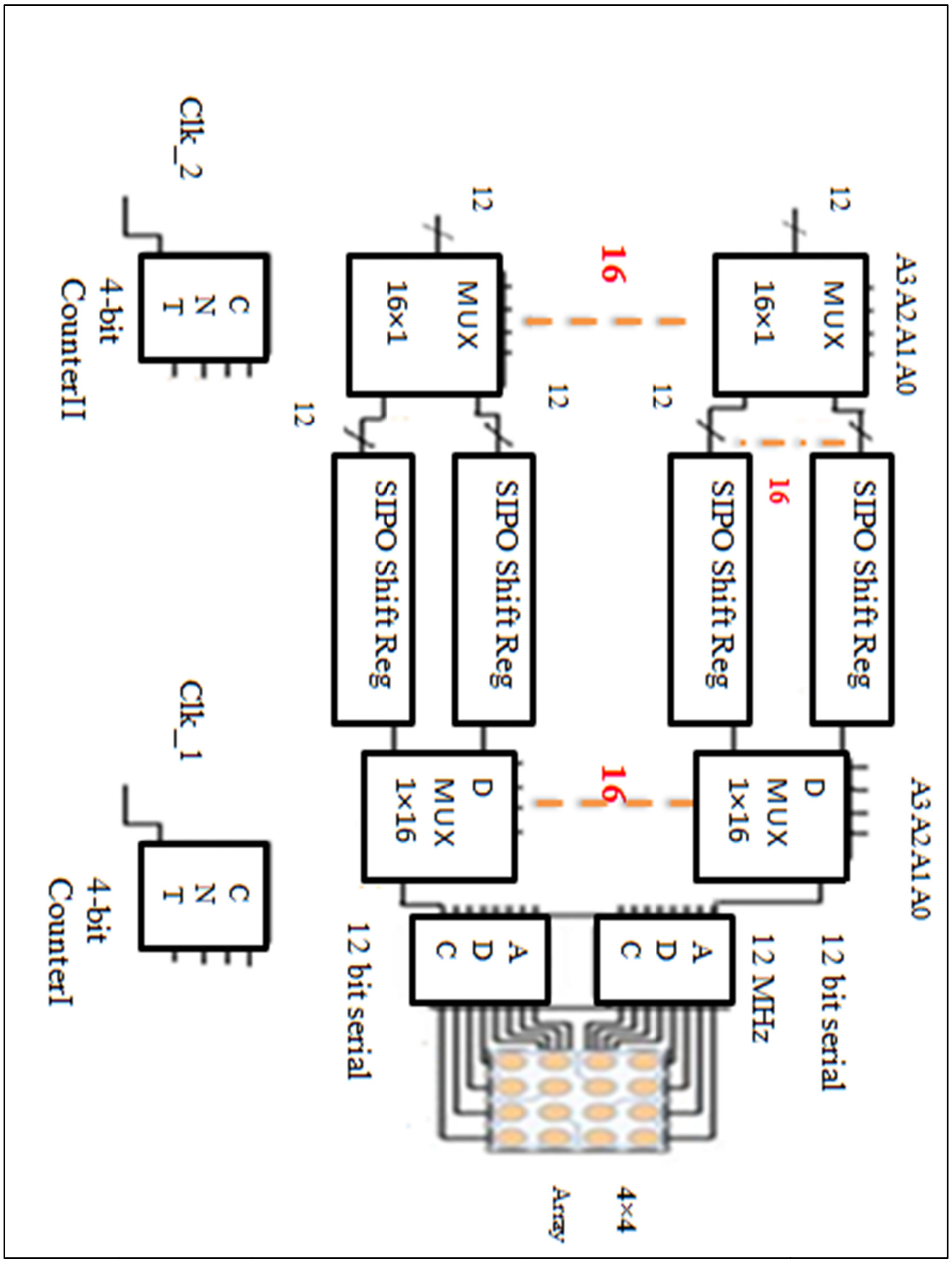


Figure 4.6. Steps in Figure 4.2 & 4.

Since a 4 ×4 array module has 16 detectors, only one 16-to-1 Multiplexer is needed to select the 12-bit parallel data from a specific frame (frame 0 -15) of one of the 16 photo detectors. Figure 4.7 shows this process.

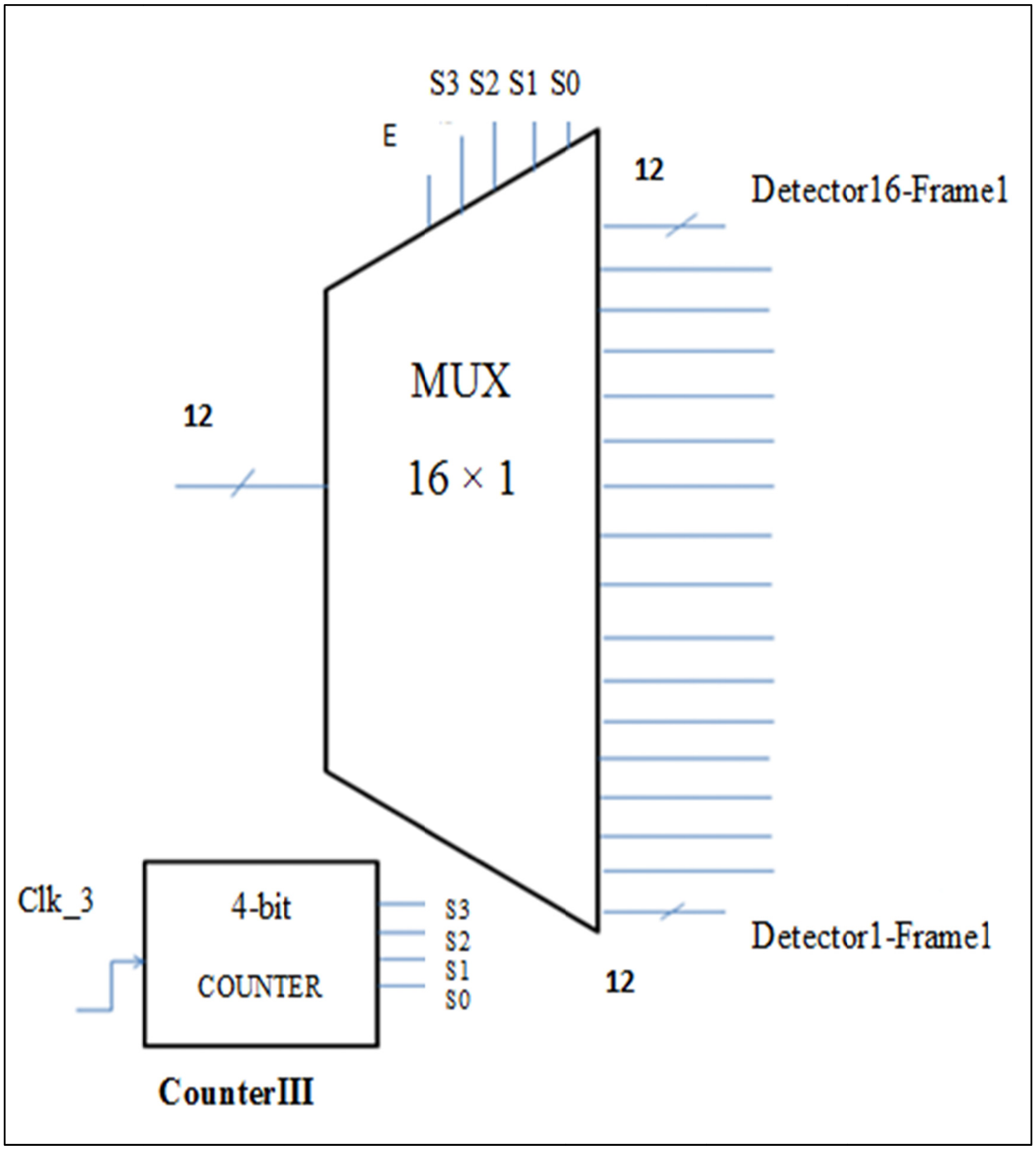


Figure 4.7. One 16-to-1 Multiplexer.

The 12-bit parallel of the first frame will be ready on the input of the multiplexer on the 25th Clk edge. CounterIII clocks after 25 ADCLK, it waits in each count until the last frame of the first detector is selected. This process will be taken for 192 Clk cycles until detector16 of frame16 is selected. Figure 4.8 & 9 show these steps.

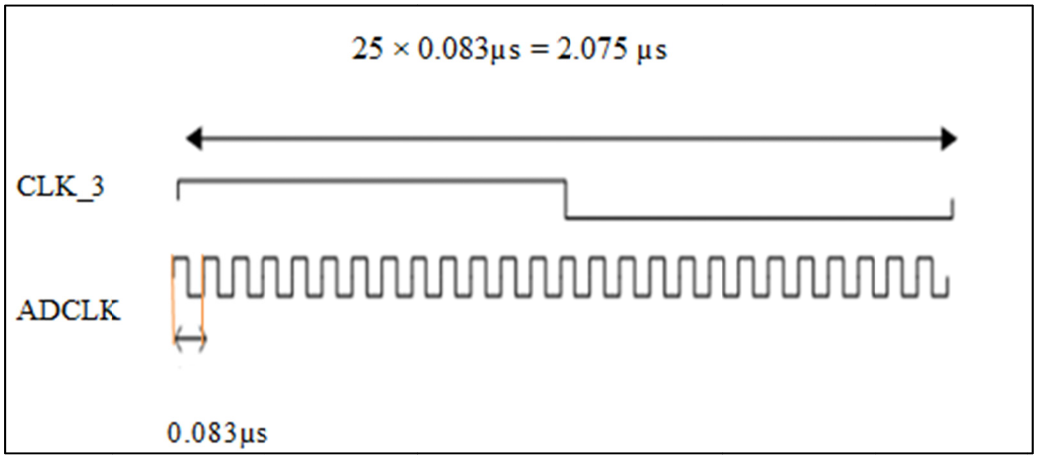


Figure 4.8. Timing analysis CounterIII for first frame of detector 1.

The Block Memory Generator core uses embedded Block Memory primitives in Xilinx FPGAs to extend the functionality and capability of a single primitive to memories of arbitrary widths and depths. Sophisticated algorithms within the Block Memory Generator core produce optimized solutions to provide convenient access to memories for a wide range of configurations. [7]

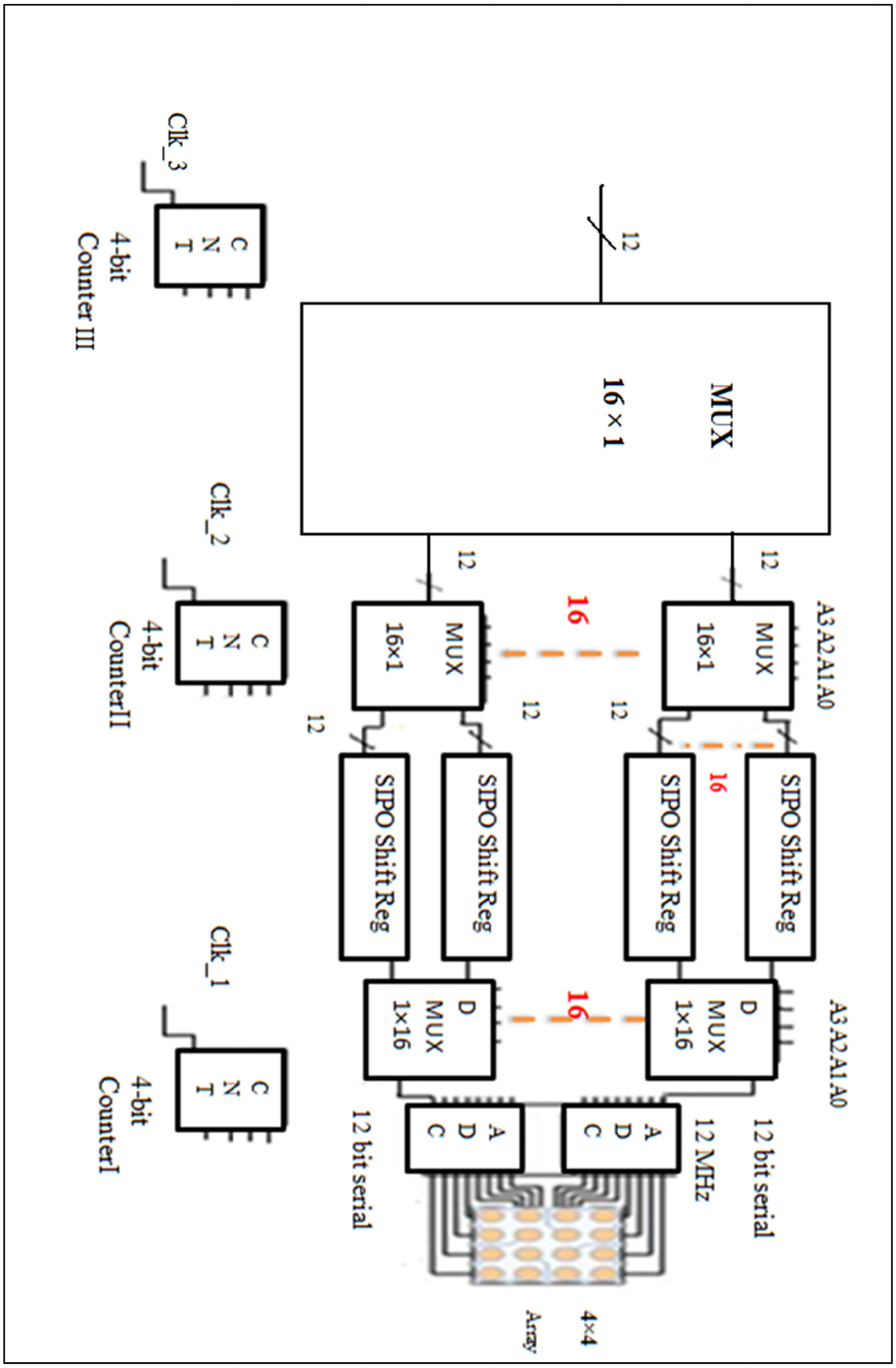


Figure 4.9. Steps in Figure 4.2 & 4 & 7.

The Block Memory Generator has two fully independent ports that access a shared memory space. Both A and B ports have a Write and a Read interface.

In Zynq-7000, 7 series, Virtex-6, Virtex-5 and Virtex-4 FPGA architectures each of the four interfaces can be uniquely configured with a different data width. When not using all four interfaces, the user can selects a simplified memory configuration (for example, a Single-Port Memory or Simple Dual-Port Memory) to reduces FPGA resource utilization.[7]

As the final step of level-1, the Block Memory RAM (192 × 16) ((16 frames × 12-bit parallel data) × 16 photo detectors) is used to store all 16 frames of a 4 × 4 module in the first level.

The Simple-Dual Ram has the Content of addressable memories and FIFOs. Write access to the memory is allowed via port A, and Read access is allowed via port B. The block diagram is shown in Figure 4.10.

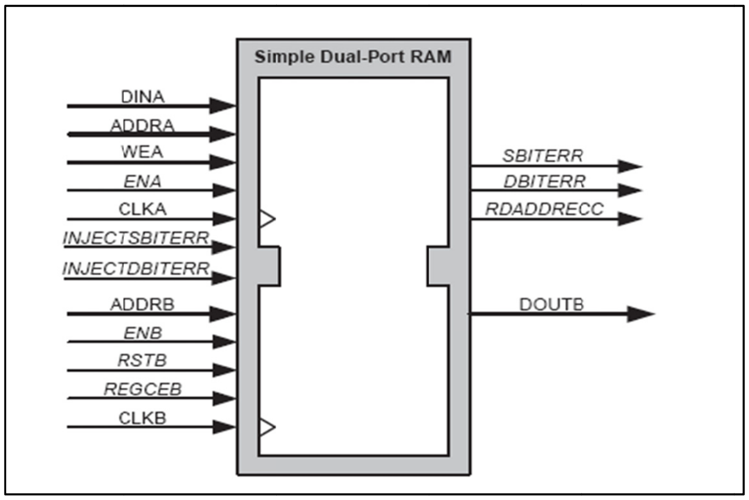


Figure 4.10. Simple Dual-port RAM block diagram.[7]

Each frame is placed on a row of Block Memory Ram. In addition, 12-bit parallel data of each detector is placed on each column. As the result, the address of each detector can easily be defined in the BRAM.

The 8-bit counter for the Write address works based on the Clk_3. At the rising edge of this Clk, the first frame (out of 16 frames) of the first detector (out of 16 detectors) is selected and read by BRAM(192×16) and is placed on the first row - first column. This process is continued until the last frame of the last detector is placed on the last row and column of the BRAM (192×16) as shown in Figure 4.11.

Up to here, a 4×4 module is read and is ready to be sent to the next level. In this design trace and determine the location of the data collected from each detector is easily feasible and each detector has an address defined by BRAM (16×192).

The 8-bit counter for the Read address at the rising edge of ClkB reads 12-bits parallel data one at a time and is incremented by one. Figure 4.11 shows this process.

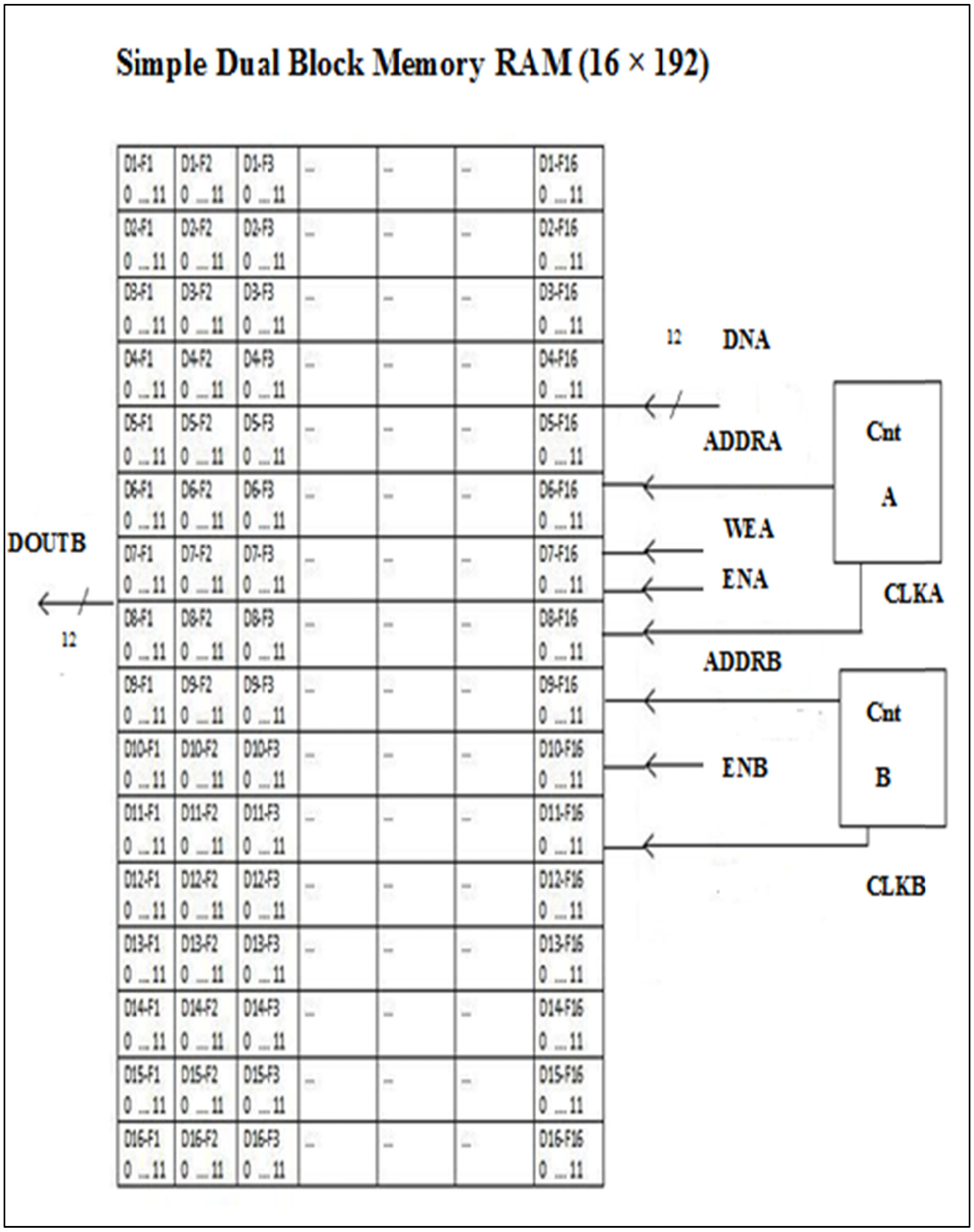


Figure 4.11. BRAM (16 × 192) block diagram.

Timing analysis of the final step shown in Figure 4.12 determines that data is ready to be written after 25 ADCLK cycles. The 12-bit parallel data form the first frame will be written into BRAM on the 26th rising edge of ClkA. Each ClkA takes 16 Clk cycles being that the 16 frame of the first photo detector is written first. Moving forward, the second 12-bit parallel data of the second detector is ready to be written into BRAM and so forth. Therefore, the process for writing 16 frames of 16 detectors into BRAM takes 256 Clk cycle.

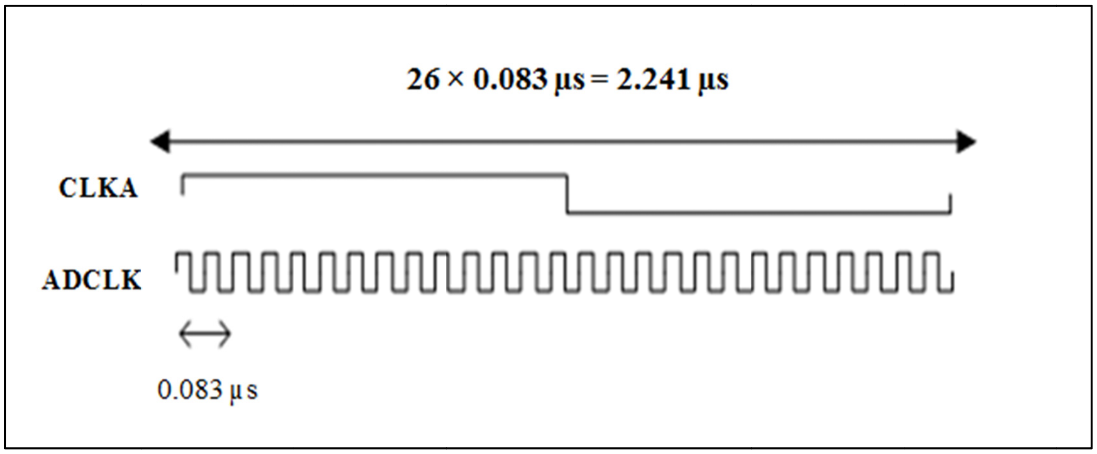


Figure 4.12. CounterA timing analyzes for the first detector and first frame.

Reading the 12-bit parallel data from the BRAM (16×192) is based on the CLKB. This Clk starts after the first row is written. The ClkB should wait for 55 ADCLK before it starts to read the first frame of the first detector and sends it to the next level.

Up to here, we have described the process of reading the 12-bit serial data from ADC and storing it into the FPGAs for the first level. Figure 4.13 shows the entire process as a whole.

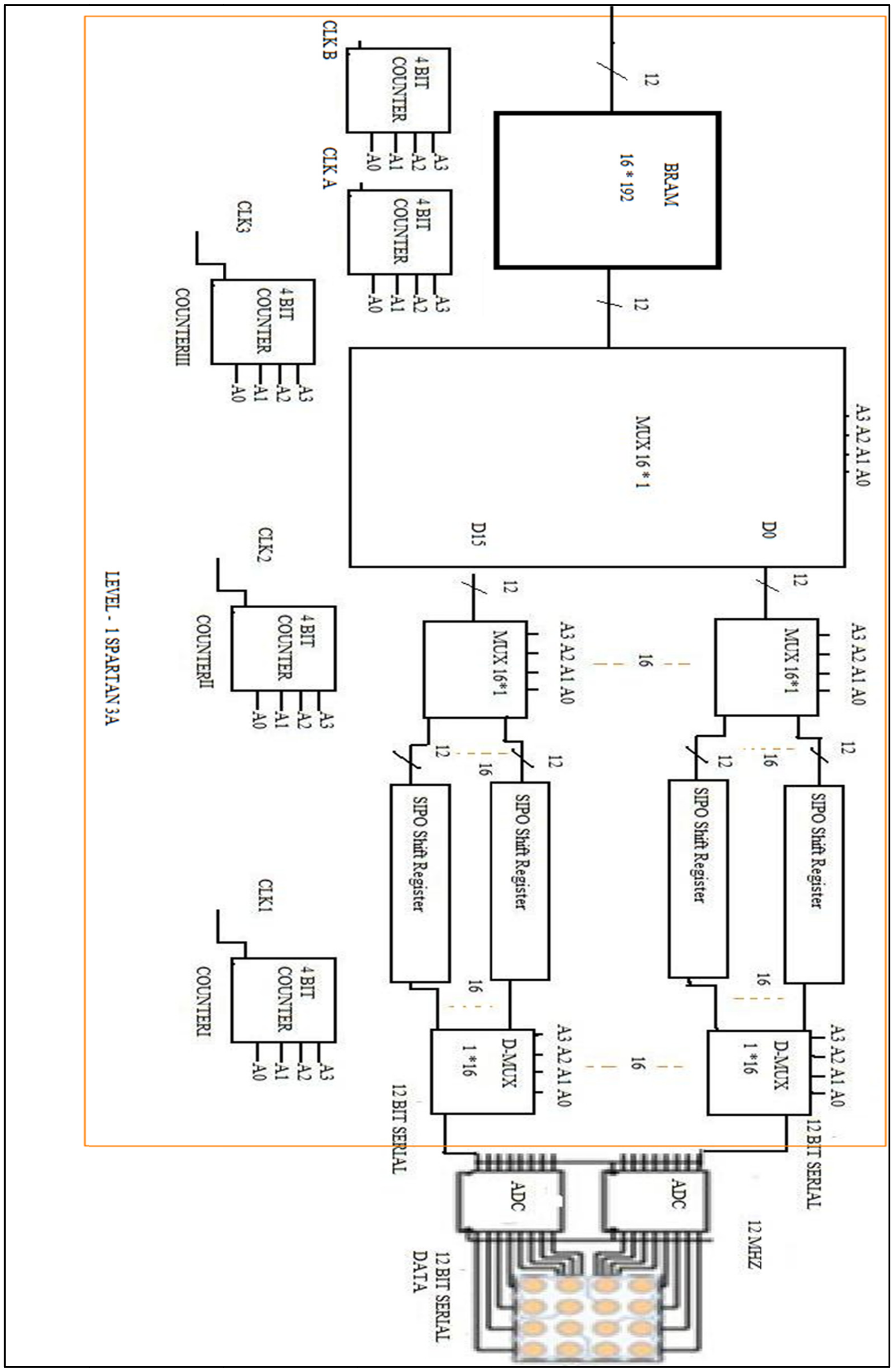


Figure 4.13. Level 1- Block Diagram for a 4×4 module.

4.1.1 TIMING ANALYZES FOR THE FISRT LEVEL

As discussed, in order to meet the specifications, the system has to collect and transmit all data in 1ms interval. The specifications analysis showed that the time the data acquisition system needs to process the collected data is 0.252msec. It takes 0.732 msec to transfer all 12,582,912-bits through a 2 Gbyte/sec serial output and 16 µs for the module to sample and digitalize the incoming signal (1ms – 0.732 ms – 16µs = 0.252). The required time is shown in Figure 4.14.

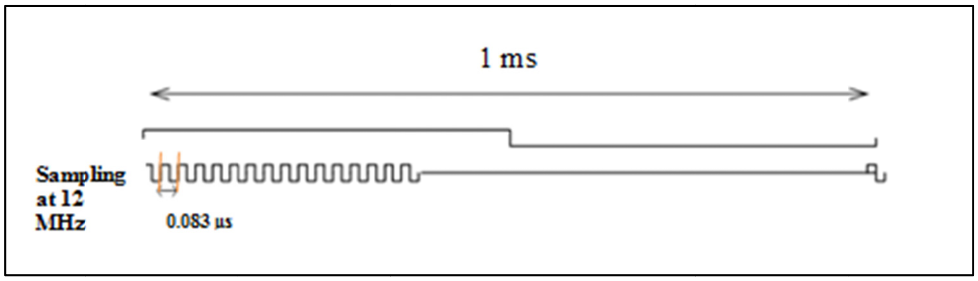


Figure 4.14. Level 1- Required time for a 4×4 module

Based on the calculations in Figure 4.3, the time required for the first detector of the first frame that is shown in Figure 4.15 is as follows:

- Clk_1: the first bit will be ready after 12 ADCLK:
- First frame-first detector: 0.083 µs× 12 = 0.996 µs
- Time to sample and digitalize for one frame :

  16×12 × 0.083 µs = 15.936 µs

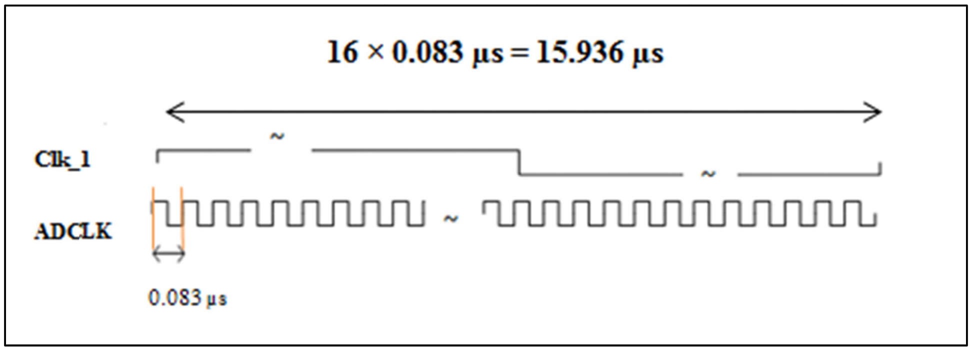


Figure 4.15. Level-1 timing verifications for the Clk_1.

Based on the calculations in Figure 4.5, the time required for the first detector of the first frame that is shown in Figure 4.16 is as follows:

- Clk_2: the first 12 parallel data will be ready after 37 ADCLK:
- First frame-first detector: 0.083 µs× 24 = 1.992 µs
- All 16 frame of one detector: 16 × 1.992 µs = 31.872 µs

To meet the specifications, the system must collect and transmit all 16 frames in 1ms intervals. Since 31.872 µs is less than 500 µs (0.5 ms), the time is acceptable.

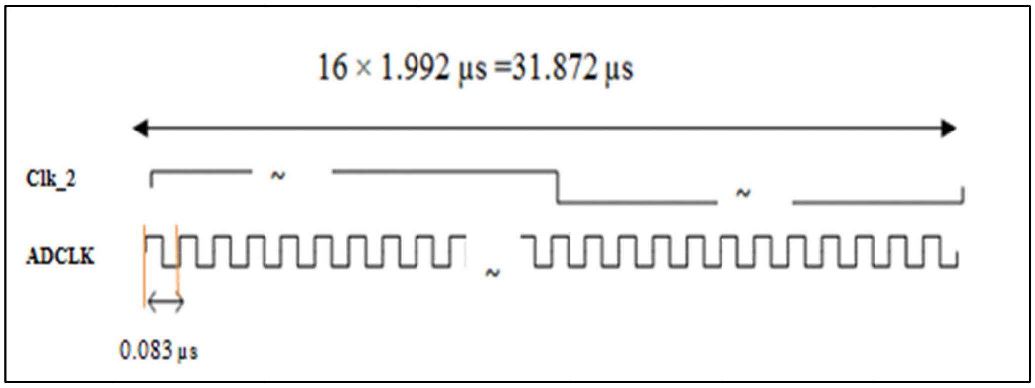


Figure 4.16. Level-1 timing verification for the Clk_2.

Based on the calculations in Figure 4.8, the time required for the first detector of the first frame that is shown in Figure 4.17 is as follows:

- Clk_3: the first 12 parallel data will be ready after 38 ADCLK:
- First frame-first detector: 0.083 µs× 25 = 2.075 µs
- All 16 frame of one detector: 192 × 2.075 µs = 397.44 µs

To meet the specifications, the system must collect and transmit all data in 1ms intervals. Since 397.44 µs is less than 1 ms, the time is acceptable.

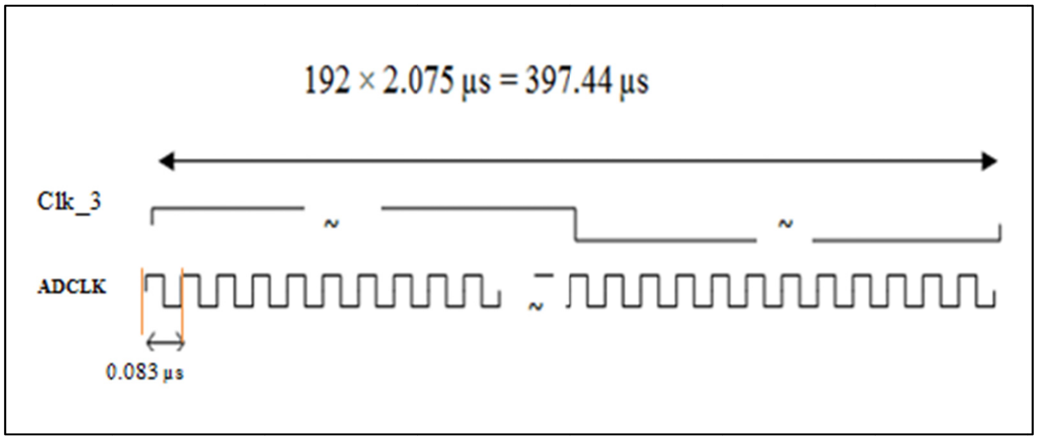


Figure 4.17. Level- 1 timing verification for the third step

Based on the calculations in Figure 4.9, the time required for the first detector of the first frame that is shown in Figure 4.18 is as follows:

- ClkA: the first 12 parallel data will be ready after26 ADCLK:
- First frame-first detector: 0.083 µs× 26= 2.158 µs
- One sample of data: 192 × 2.158 µs = 414.336 µs

To meet the specifications, system should collects and transmits all data in a 1ms intervals. Since 414.336 μs is less than 1 ms the time is acceptable and system has enough time to process and take another sample of data.

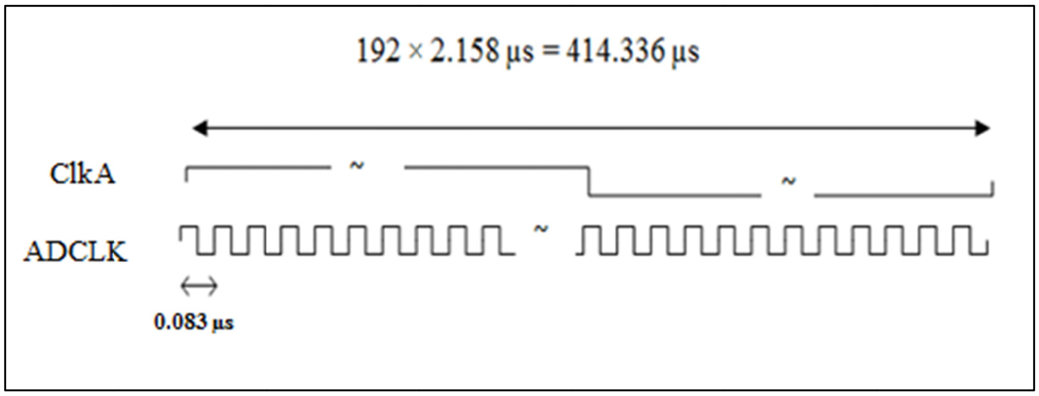


Figure 4.18. Level-1 timing verification for ClkA.

Finally, the time to parallel transmit one frame of data to the second level is:

0.996 μs (first step) +1.992 μs (second step) +2.075 μs (third step) +2.241μs (write to the BRAM) = 7.31μs

The timing analysis for the first level shows the time needed by the data acquisition system to process the collected data is 414.336 μs which is less than 1 ms. Therefore, the data acquisition system has enough time to collect, digitalize, and store 12-bit parallel data into BRAM (16×192) and data is then sent to the next level.

4.2. FROM THE FIRST LEVEL TO THE SECOND LEVEL OF FPGAS

In the architecture described in Figure2.4, the first stage has 4096 Spartan3A, and the second stage has 256 Spartan3A. Each 16 level-1 FPGA transmits 12-bit in parallel to one of the level-2 Spartan-3A FPGAs in this level. Figure 4.19 shows this concept.

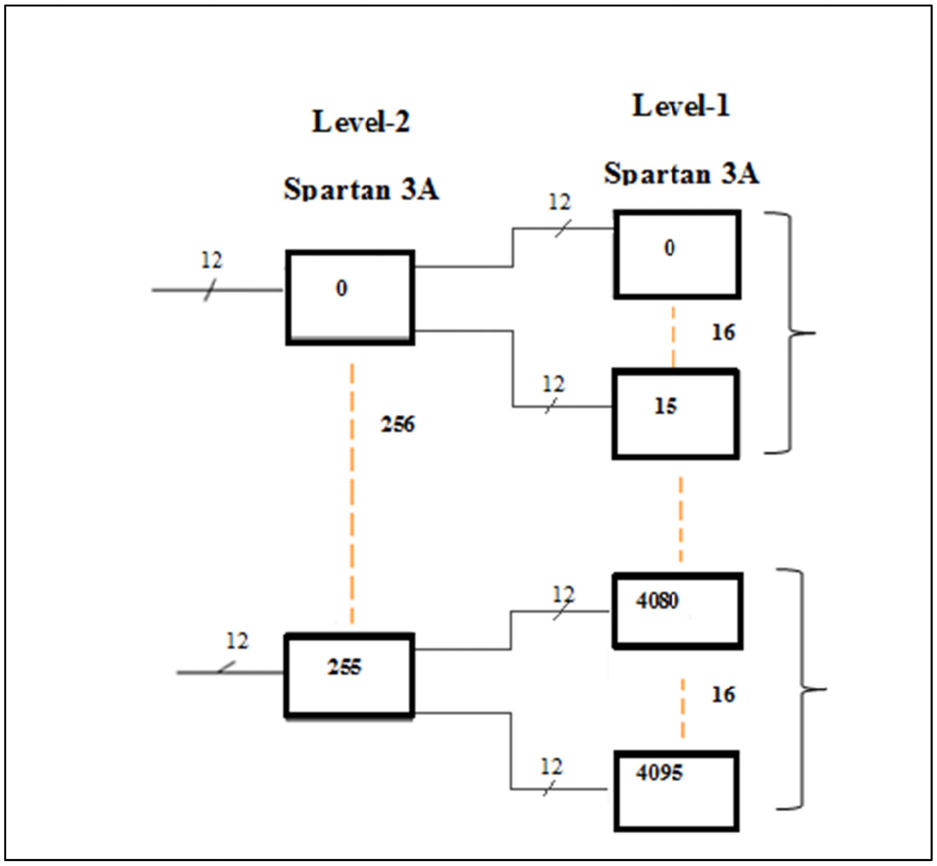


Figure 4.19. Level-1 to level-2 block diagram.

- Total Data Transferred per Sample Pulse for the second level is:
- 16 (level-1 FPGA) × 3072 (16 detectors × 12-bit × 16 frame) = 49,152
- Time to transfer from Level-1 to Level-2 is:

16 (frame) × 16 (photo detectors) × 12-bit per sample × 6.9 ns (144 MHz, CLK) = 21.196μs

The 12-bit parallel output of sixteen FPGAs in level-1 is collected into the one FPGA in level-2. Therefore, the 16-to-1 Multiplexer (Xilinx M16_1E) is used to select a 12-bit parallel data from the 16 level-1 FPGAs.

- Total number of Multiplexers (Xilinx M16_1E) for the second level are:

  1(16-to-1Multiplexer × 256 (the 16th groups of level-1 FPGAs out of 4096 module) = 256

A 4-bit counter is used to control all the 256 16-to-1Multiplexers at the same time. In this design, the first BRAM (192×16) from level-1 is read; then, the second BRAM (192×16) of level-1 is read and so forth. Therefore, finding the address of each detector in this level is easily feasible.

The delay of 256 is needed since each BRAM (192×16) of level-1 must be read completely in each count. The counter counts from 0 to 16 and in each count has 256 Clk cycles delay.

The Block Memory Generator core uses embedded Block Memory primitives in Xilinx FPGAs to extend the functionality and capability of a single primitive to memories of arbitrary widths and depths. Sophisticated algorithms within the Block Memory Generator core produce optimized solutions to provide convenient access to memories for a wide range of configurations. [6]

Block Memory simple Dual port Ram with the write/read width 16 and write/read depth 3072 bits is generated by the IP core generator.

On the first rising edge of the write Clk (ClkB_2), the first detector-first frame is written into BRAM (16 × 3072). First level-1 BRAM (16×192) is placed in first row and so forth, as shown in Figure 4.20. The address width of addressA and AddressB is 12-bit. Figure 4.20 shows the level -2 block diagram.

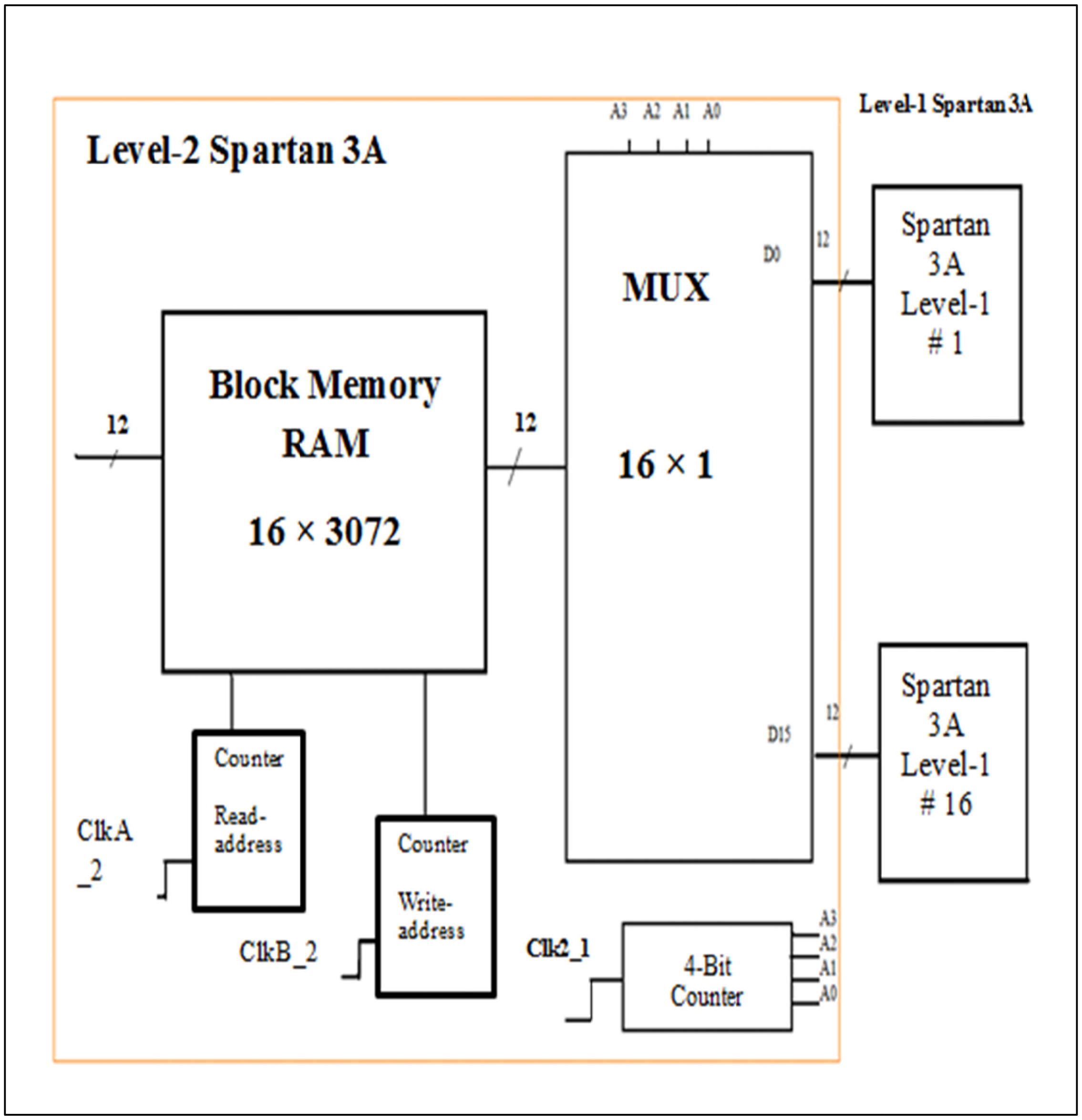


Figure 4.20. Level -2 block diagram.

- The time to parallel transmit one sample of the data to the third level is:

256 (Clk cycle need to send one level-1 BRAM (16×192)) × 6.9 ns (144 MHz) ×16 = 28.44μs

Moreover, the transfer clock speed is limited to the system clock speed of the Spartan 3A FPGAs -144MHz. Figure4.21 shows the complete block diagram for level-2.

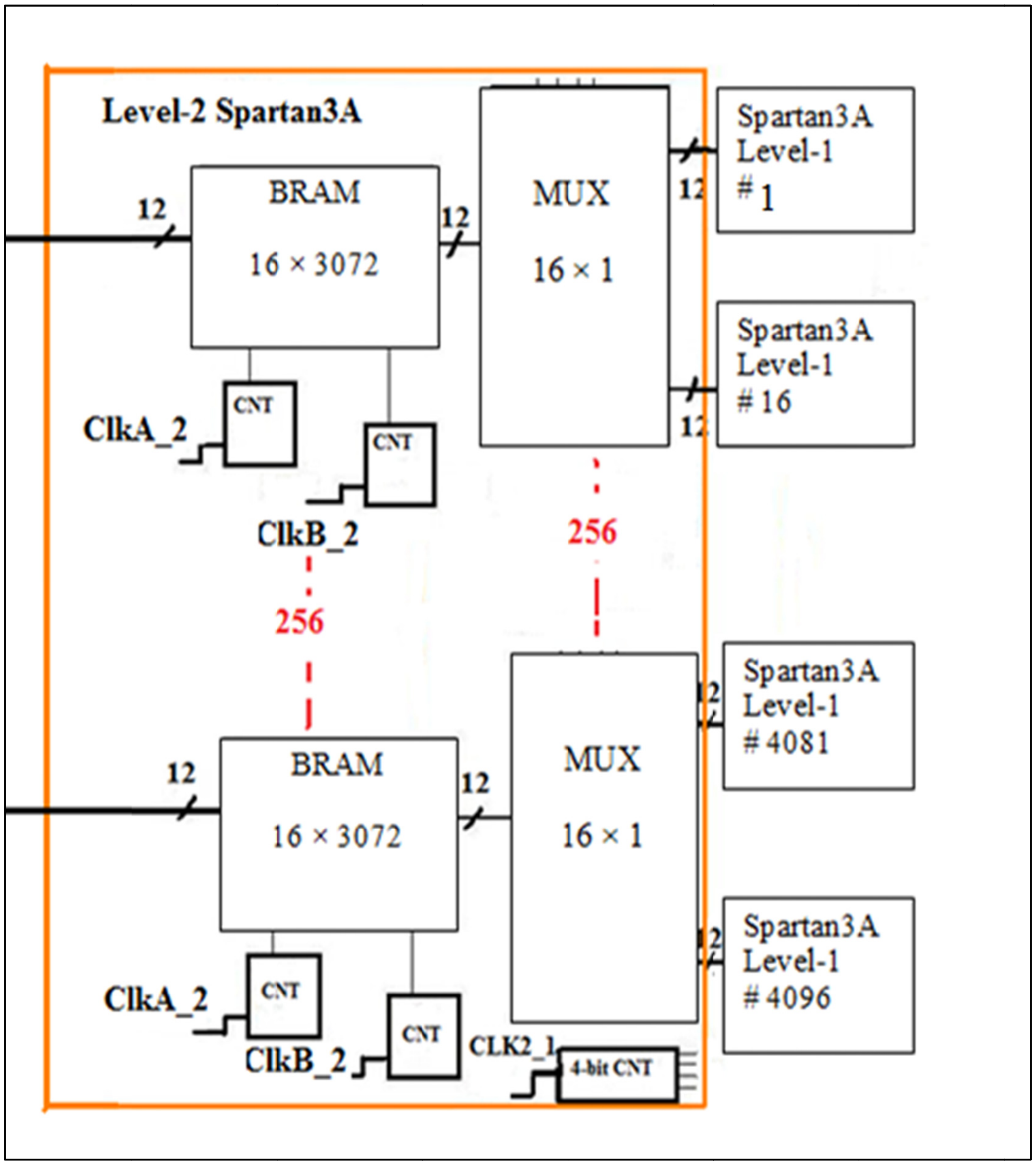


Figure 4.21. Level-2 complete block diagram.

4.3. FROM THE SECOND LEVEL TO THE THIRD LEVEL OF FPGAS

The presented architecture is designed based on three levels FPGAs; in this design the first level has 4096 Spartan3A, the second stage has 256 Spartan3A, and the final level has 16 Virtex6. Each 16 level-2 FPGA transmits 12-bit in parallel to one of the level-3 Virtex6 FPGAs in this level.

Virtex-6 FPGAs are the programmable silicon foundation for Targeted Design Platforms that deliver integrated software and hardware components to enable designers to focus on innovation as soon as their development cycle begins. In addition to the high-performance logic fabric, Virtex-6 FPGAs contain many built-in system-level blocks. These features allow logic designers to build the highest levels of performance and functionality into their FPGA-based systems. Built on a 40 nm state-of-theart copper process technology, Virtex-6 FPGAs are a programmable alternative to custom ASIC technology. Virtex-6 FPGAs offer the best solution for addressing the needs of high-performance logic designers, high-performance DSP designers, and high-performance embedded systems designers with unprecedented logic, DSP, connectivity, and soft microprocessor capabilities. The Virtex6 has the integrated interface blocks for PCI Express® designs. So, the Level-3 is designed by Virtex-6 FPGAs to use PCI Express transfer protocol to pass data forward. [7]

An optional output data pipeline register allows higher clock rates at the cost of an extra cycle of latency. During a write operation, the data output can reflect either the previously stored data, the newly written data, or remain unchanged. DSP applications use many binary multipliers and accumulators, best implemented in dedicated DSP slices. All Virtex-6 FPGAs have many dedicated, full-custom, low-power DSP slices combining high speed with small size, while retaining system design flexibility.

The number of I/O pins varies from 240 to 1200 depending on device and package size. Each I/O pin is configurable and can comply with a large number of standards, using up to 2.5V. The Virtex-6 FPGA Select IO Resources User Guide describes the I/O compatibilities of the various I/O options. With the exception of supply pins and a few dedicated configuration pins, all other package pins have the same I/O capabilities, constrained only by certain banking rules. All I/O pins are organized in banks, with 40 pins per bank. Each bank has one common VCCO output supply-voltage pin, which also powers certain input buffers. Some single-ended input buffers require an externally applied reference voltage (VREF). There are two VREF pins per bank (except configuration bank 0). A single bank can have only one VREF voltage value. An integrated Tri-mode Ethernet MAC (TEMAC) block is easily connected to the FPGA logic, the GTX transceivers, and the Select IO resources. This TEMAC block saves logic resources and design effort. All of the Virtex-6 devices (except the XC6VLX760) have four TEMAC blocks, implementing the link layer of the OSI protocol stack.

The CORE Generator™ software GUI helps to configure flexible interfaces to GTX transceiver or Select IO technology, to the FPGA logic, and to a microprocessor (when required). The TEMAC is designed to the IEEE STD 802.3-2005 specification. 2,500 Mb/s support is also available. [7]

Each 16 level-2 FPGA transmits 12-bit in parallel to one of the level-3 Virtex6 FPGAs in this level. Therefore, 49152 bits of data is transferred in 12-bit parallel. On 192 input pins a total of 786,432 bits are received by each level-3 FPGA.

- Total Data Transferred per Sample Pulse for the third level is:

  16 (level-1 FPGA) × 3072 (16 detectors×12-bit ×16 frame) ×16(level-2 FPGA) = 786,432 bits are received by each level-3 FPGA

Figure 4.22 shows this configuration:

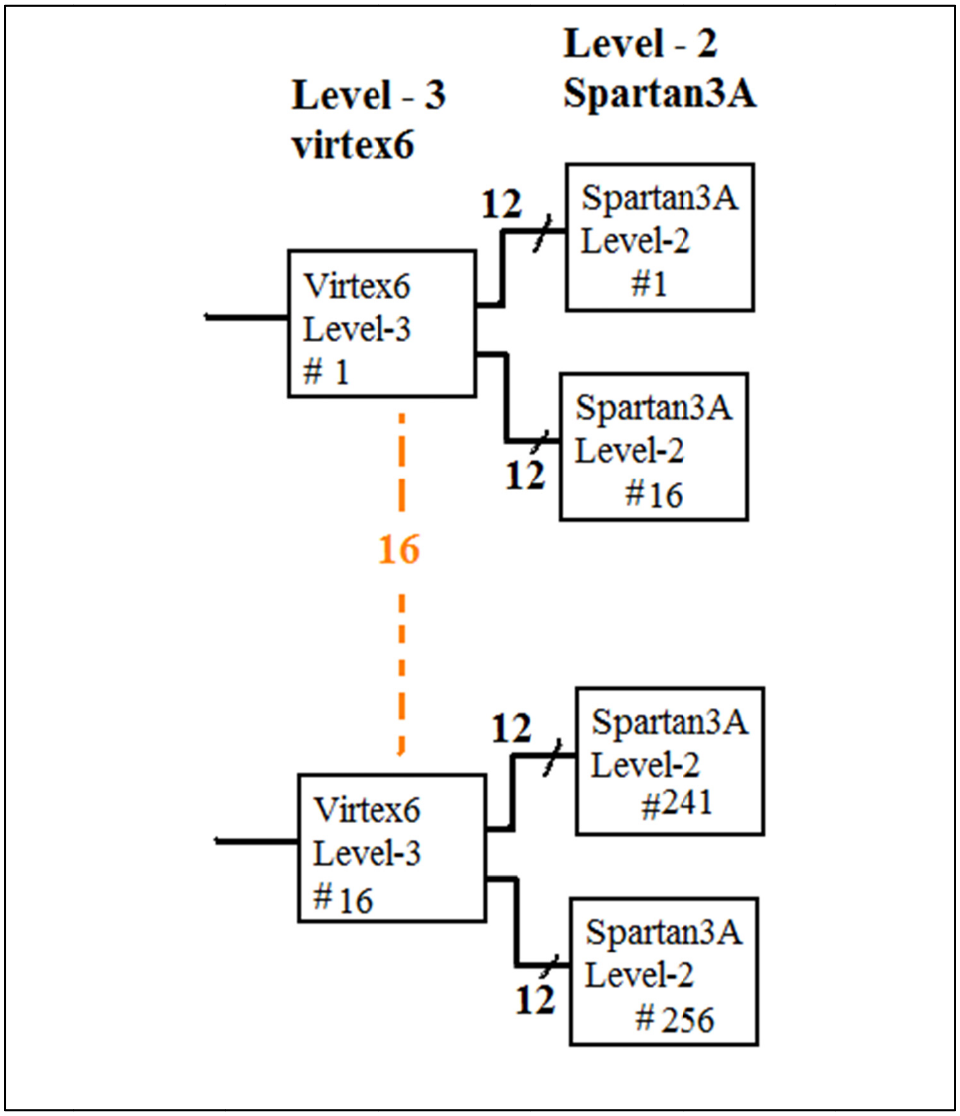


Figure 4.22. Level-2 to level-3 block diagram.

Every Virtex-6 FPGA has between 156 and 1064 dual-port block RAMs, each storing 36 Kbits. Each block RAM has two completely independent ports that share nothing but the stored data. Each memory access, read and write is controlled by the clock. All inputs, data, address, clock enables, and write enables are registered. Nothing happens without a clock. The input address is always clocked, retaining data until the next operation. . [7]

Since the 16 level-2 BRAM (16×3072) is collected and stored in one Virtex6 of level-3, the 16-to-1 Multiplexer is designed to select each level-2 BRAM (16×3072) one at a time and send it to the output. Therefore, the counter controlling all multiplexers in this level has a significant delay in each count in order to get one 4×4 module and finally send it to the output. As a final result, the next 4×4 module is counted and so forth.

The BRAM (16 × 49152) is needed for storing all data bits in the final step. Single Port RAM and block memory resources are:

- 18K BRAM: 1
- 36k BRAM: 22
- Address Width: 16

The output must be 12-bit in serial to transfer all serial data to the fiber channel switch. Output of the multiplexer goes to the one Parallel-In – Serial-Out Shift Register to convert 12- bit parallel data to 12- bit serial data.

- Total number of Multiplexers (Xilinx M16_1E) for the third level is:

  1(16-to-1 Multiplexer ×16 (the 16th groups of level-2 FPGAs out of 256) =16

Figure 4.23 shows the Level-3 complete block diagram.

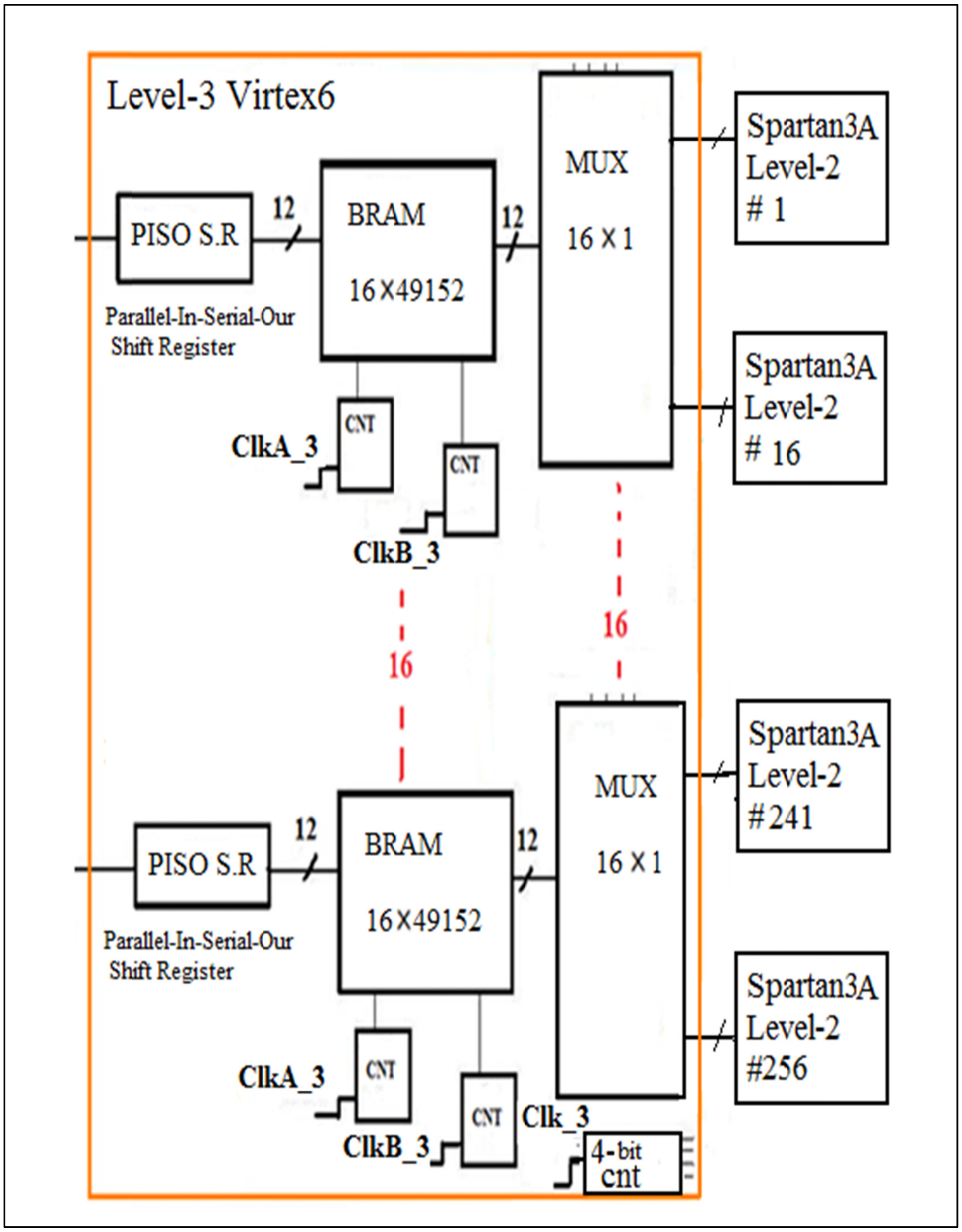


Figure 4.23. Complete block diagram of level-3.

- The time to parallel transmit one sample of data to the third level is:

256 (Clk cycle need to send one level-1 BRAM) × 6.9 μs (144 MHz) ×16 =28.44 μs

Moreover, the transfer clock speed is limited to the system clock speed of the Spartan 3A FPGAs -144MHz.

Each level-3 FPGA uses the PCI Express transfer protocol to produce an optical serial output at a rate of 5Gbits/sec. Two additional Intellectual Property Cores are used to establish the PCI Express protocol – GTX Transceiver and Block RAM. Once data is received and processed by level-3 FPGAs (at approximately 5Gbits/sec), the Fiber Channel Switch takes over to generate the 2GByte/sec transfer rate. Figure 4.24 shows the final steps of this architecture.

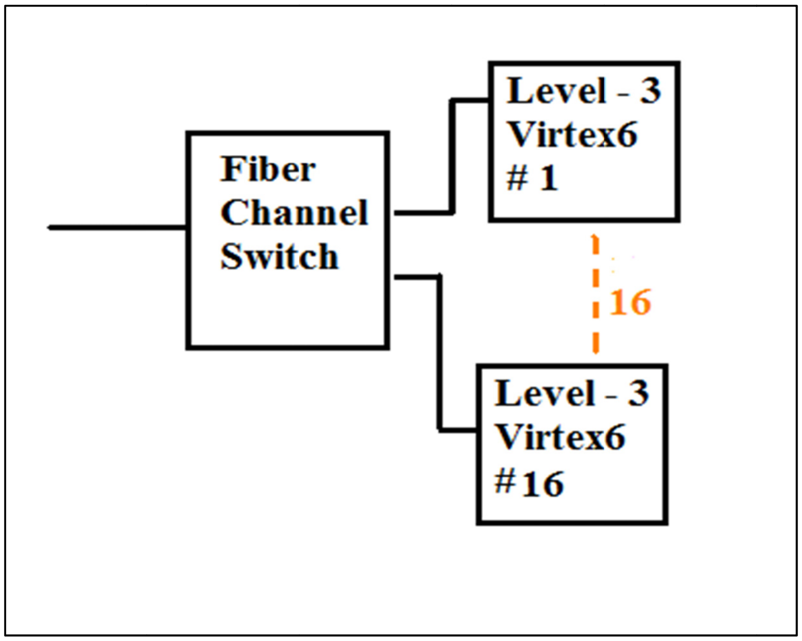


Figure 4.24. Block diagram for level-3 to fiber channel switch.

Figure 4.25 shows the complete block diagram for all three levels of FPGAs.

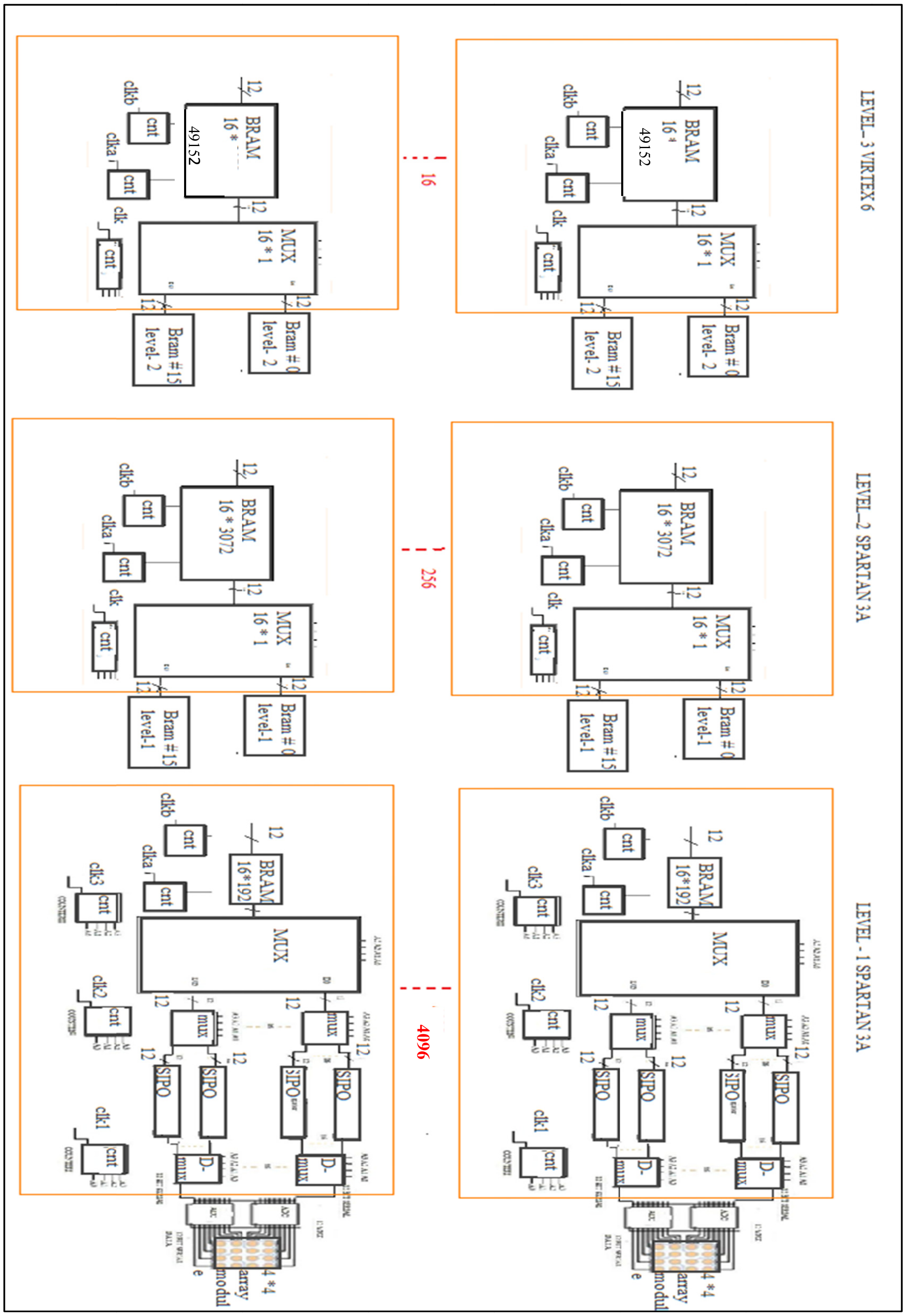
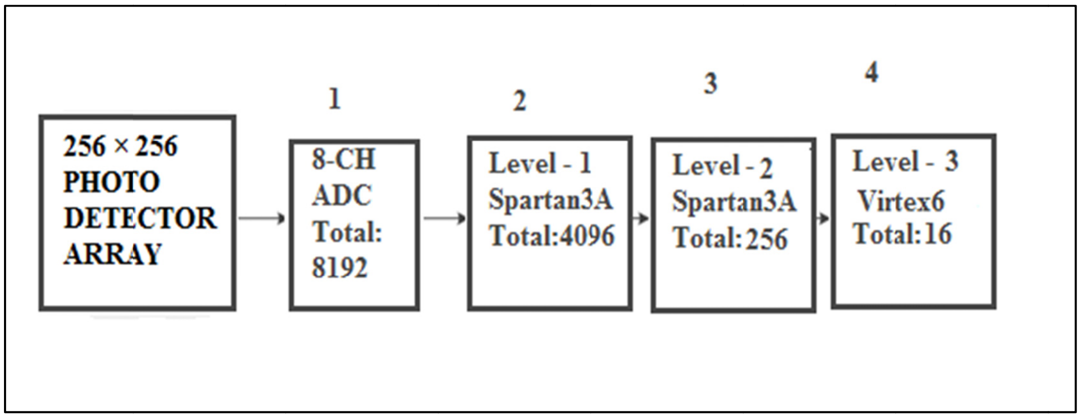


Figure 4.25. Complete block diagram.

CHAPTER 5: TIMING AND DATA FLOWS

In order to meet the specifications, the system should collect and transmit one sample of data in 1ms intervals. Calculations show that time required to sample, digitize and process the collected data (entire 12,582,912-bit) through the three levels of FPGA is 0.774ms. This amount of time is less than 1 ms, which meets the requirement. It also gives the system enough time to process the next sample of data. Figure 5.1 describes the summary of timing analysis for the entire process.



256 × 256 PHOTO DETECTOR ARRAY
1
8-CH ADC Total: 8192
2
Level - 1 Spartan3A Total:4096
3
Level - 2 Spartan3A Total:256
4
Level - 3 Virtex6 Total:16

Figure 5.1. Summary of timing analyzes for all steps.

Summary of timing analysis meets the specifications for all three levels is:

1. ADCCLK input sample per rate is:

   12 MHZ (0.083 µs); 12-bit per sample; 16 sample per pulse

   ADCLK output (LVDS) : 12 MHZ

   LCLK output (LVDS) :144 MHZ

2. Level- 1 :

Amount of data per sample pulse = 12-bit/sample × 16 photodetector = 3072

Required timing sample for level-1 = 16 samples × 12-bit /sample × 83.33 ns (12 Mhz) = 16 µs

- Time to parallel transmit one frame of data to the second level is:

  0.996 µs (first step) +1.992 µs (second step) +2.075 µs (third step) +2.241µs (write to the BRAM) = 7.304 µs

Finally, time to parallel transmit one sample data (received data) to the second level is 414.336 µs. Since the initial pulse comes at 1 kHz and data is sampled sixteen frames at 1ms, this time is acceptable. Therefore, the system has enough time to process data to the next level by the time the next pulse arrives.

3. Level- 2:

Time required for one sample of data in Level- 2 is:

[3072 (12-bit per sample×16 photo detectore×16 frame) +1 (multiplexer) +1 (write to the BRAM)]× 6.9 ns (144 MHz CLK)= 21.2106 µs

4. Level- 3:

Time requirement for one sample of data in Level- 3is:

[3072 (12-bit per sample×16 photo detectore×16 frame) × 16 level-2 FPGA +1(multiplexer) +1 (write to the BRAM)] × 6.9 ns (144 MHz CLK) = 339.16 µs

Total time by image acquisition system for collecting one sample of data and transferring it to the fiber channel switch requires 0.774 µs (414.336 µs + 21.2106 µs + 339.16 µs). This amount of time is less than 1 ms, which meets the initial requirement. Since 0.774 µs is smaller than 1 ms, this design is considered optimal. Therefore, the system has a sufficient amount of time to process the next sample of data.

CHAPTER 6: PLACE OF EACH DETECTOR

In level-1, the simple Dual Block Ram (16×192) is used to store the image, and an 8-bit address is used. The LS four bits of this address defines the place location of each detector in the 4×4 module, and the MS four bits determine the location of module in each frame. Figure 6.1 shows this concept.

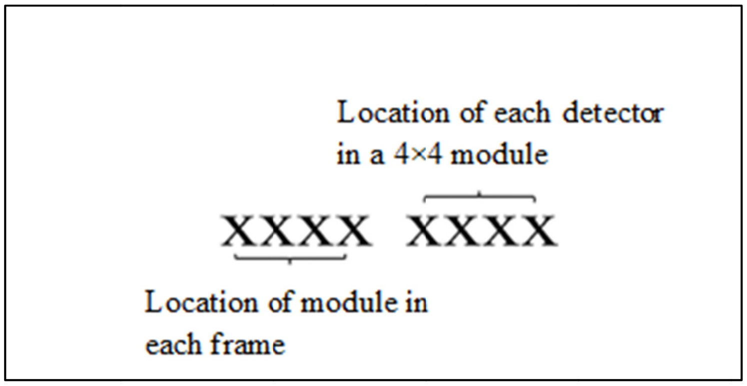

Location of each detector in a 4×4 module
XXXX XXXX
Location of module in each frame

Figure 6.1. Address of photo detector in level-1.

In the seconed level of this architecture, one BRAM (16× 3072) is generated by Block Memory Genrator as shown in Figure 4.20. This BRAM is generated with 12bit width address for reading and writing. This 12-bit address determines the location of detectors in level-1, level-2 and the related frame for each detectors. Figure 6.2 shows this concept.

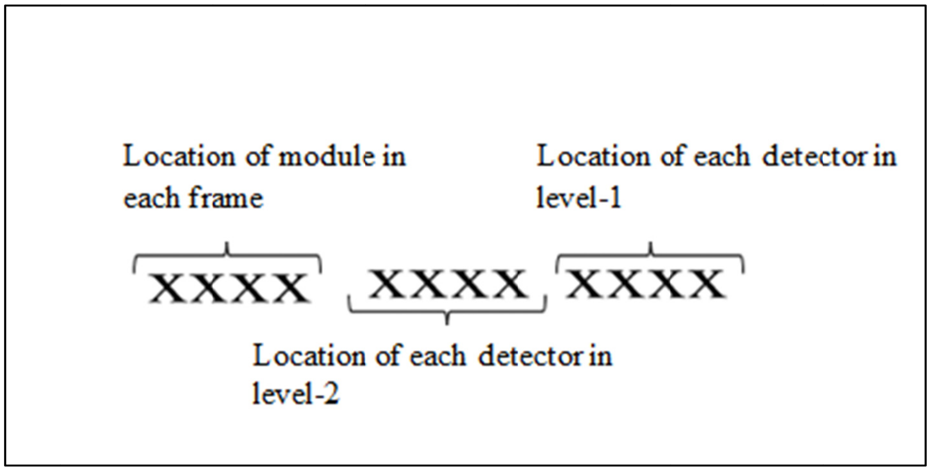

Location of module in each frame
Location of each detector in level-1
XXXX XXXX XXXX
Location of each detector in level-2

Figure 6.2. Address of photo detector in level-2.

Finaly, in the third level, one BRAM (16× 49154) is generated by Block Memory Genrator as shown in Figure 4.23. This BRAM is generated with a 16bit width address for readng and writing. This 16 bit address determines the place of detector in level-1, level-2 , level-3,and the related frames for each detectors. Figure 6.3 shows this concept.

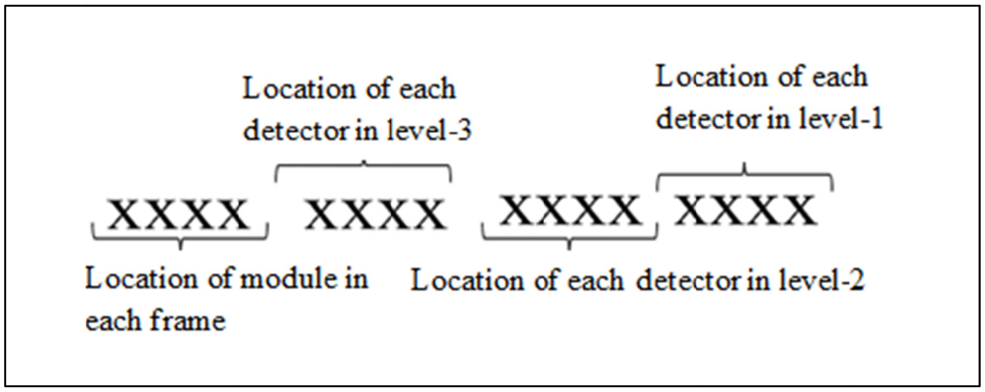

Location of each detector in level-3
Location of each detector in level-1
XXXX XXXX XXXX XXXX
Location of module in each frame
Location of each detector in level-2

Figure 6.3. Address of photo detector in level-3.

Furthermore, Figure 6.4 shows the place of each level of FPGAs and detectors in a 64×64 module .

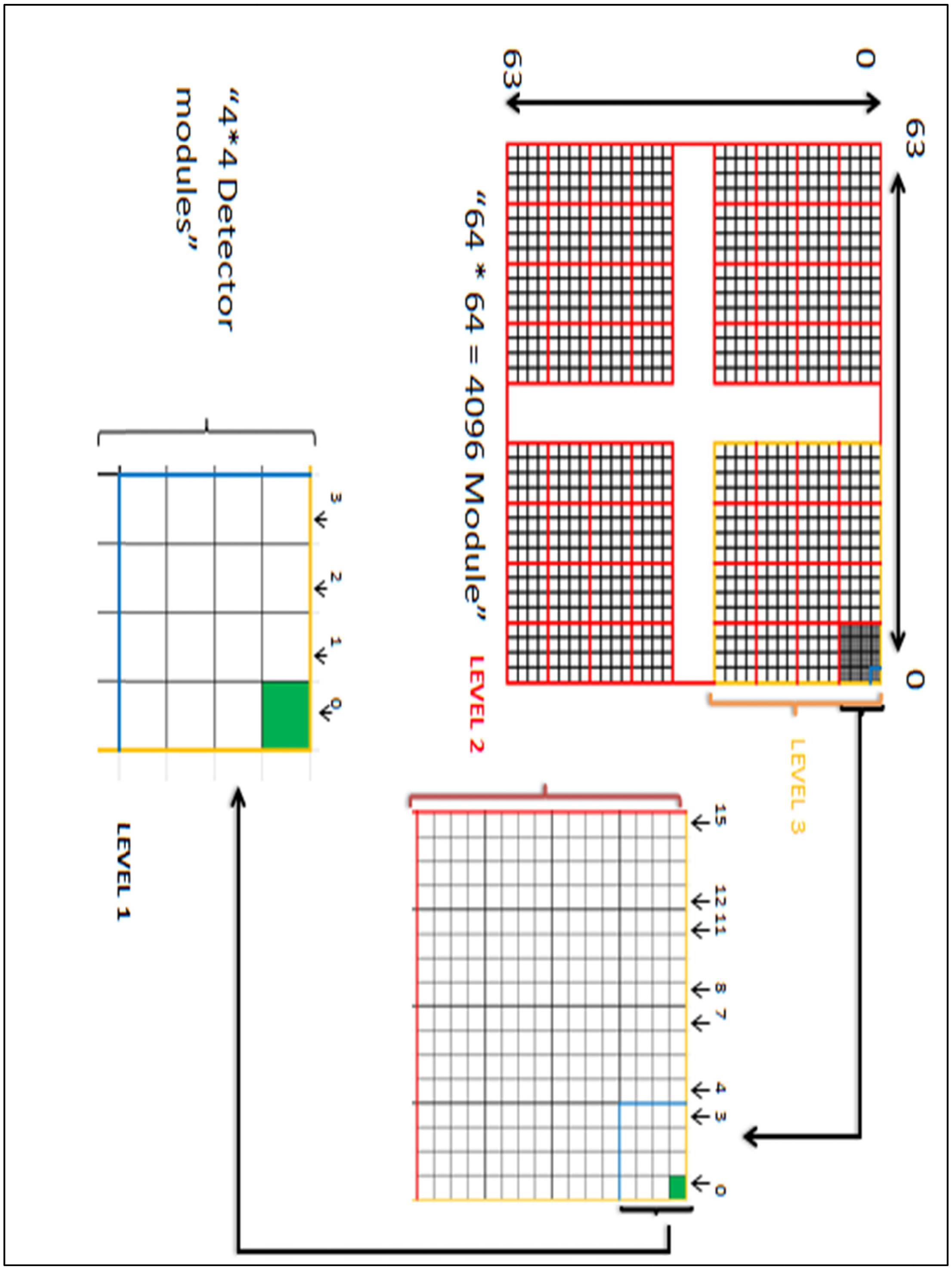


Figure 6.4. The place of each detectors in a 64×64 module.

CHAPTER 7: VHDL DEVELOPMENT

Digital hardware has experienced drastic expansion and improvement in the past 40 years. Since its introduction, the number of transititors in a single chip has grown exponentially, and a sillicon chip now routinely contains hundreds of millions trasisitors.

In the past, the major applications of digital hardware were computational systems. As application become larger and more complex, the task of designing digital cuircuits becomes more difficult. The best way to handle the complexity is to view the cuircuit at a more abstract level and utilize software tools to derive the low-level implementation.[10]

FPGAs have traditionally been configured by hardware engineers using a Hardware Design Language (HDL). The two principal languages used are verilog HDL(Verilog) and Very Hight Speed Integrated Cuircuits(VHSIC) HDL (VHDL) which allows designers to design at various levels of abstarction.[11]

Given the importance of digital image processing and the significant role of hardware implemention in order to achive better performance, this chapter covers starting point for VHDL and guides for future VHDL implemention projects.

The system level is designed by Xilinx components. The ISE software is used to implement VHDL hardware language defining the RTL level of the system. The ISE software controls all aspects of the design flow. Throughout the project navigator interface, the access to all design entries and design implemention tools are acceptable.

VHDL and RTL level for the componets used in the system level are as below:

1. 16-to-1 D-Multiplexer:

Hardware design element select the D-Multiplexer D4_16E (16 × 1) with enable from Xilinx library. When the enable (EN) input of the D4_16E decoder/demultiplexer is high, one of the 16 active-High output (D15-D0) is selected with 4 bit binary address(A3-A0) input. The non selected outputs are Low. Also, when the EN inputs is Low,all outputs are Low. [12]

The part of VHDL code for the above describtion for D-Multiplexer is :

```
Y (0) <= I when S="0000" else '0';
Y (1) <= I when S="0001" else '0';
Y (2) <= I when S="0010" else '0';
Y (3) <= I when S="0011" else '0';
Y (4) <= I when S="0100" else '0';
Y (5) <= I when S="0101" else '0';
Y (6) <= I when S="0110" else '0';
Y (7) <= I when S="0111" else '0';
Y (8) <= I when S="1000" else '0';
Y (9) <= I when S="1001" else '0';
Y (10) <= I when S="1010" else '0';
Y (11) <= I when S="1011" else '0';
Y (12) <= I when S="1100" else '0';
Y (13) <= I when S="1101" else '0';
Y (14) <= I when S="1110" else '0';
Y (15) <= I when S="1111" else '0';
```

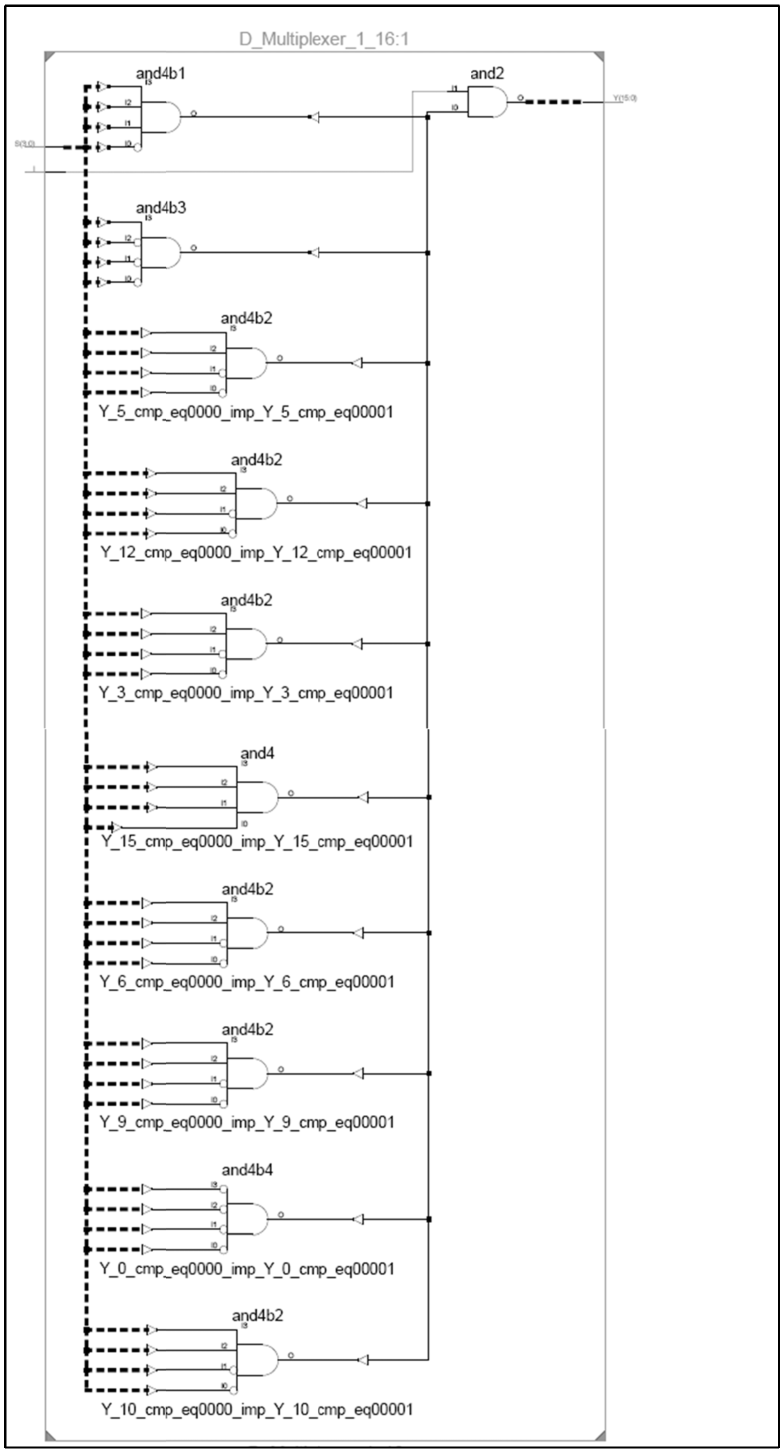

D_Multiplexer_1_16:1
and4b1
and2
Y(15:0)
S(3:0)
and4b3
and4b2
Y_5_cmp_eq0000_imp_Y_5_cmp_eq00001
and4b2
Y_12_cmp_eq0000_imp_Y_12_cmp_eq00001
and4b2
Y_3_cmp_eq0000_imp_Y_3_cmp_eq00001
and4
Y_15_cmp_eq0000_imp_Y_15_cmp_eq00001
and4b2
Y_6_cmp_eq0000_imp_Y_6_cmp_eq00001
and4b2
Y_9_cmp_eq0000_imp_Y_9_cmp_eq00001
and4b4
Y_0_cmp_eq0000_imp_Y_0_cmp_eq00001
and4b2
Y_10_cmp_eq0000_imp_Y_10_cmp_eq00001

Figure 7.1. RTL View of D-Multiplexer, implementation Spartan 3A.

2. Serial – In – Parallel – Out Shift Register:

The SRL16RE is the 16-bit shift register, with shift – left serial input (SLI), parallel outputs(Qn), clock enable (CE), and synchronous rest® inputs is selected from Xilinx library for the system level. The R input, when High, overrides all other inputs during the Low-To High clock© transition and resets the data outputs(Q) low. When CE is High and R is Low, the data on the SLI is loaded into the first bit of the shift register during the Low-To-High clock(C) transition and appears on the Q0 output. During subsequent Low-To-High clock transitions, when CE is High and R is Low, data is shifted to the next highest bit position as new data is loaded into Q0.[12]

The VHDL code that describes above describtion is:

```
signal tmp: std_logic_vector(11 downto 0);
       begin
process (Clk)
begin
  if (Clk ‘event and Clk='1') then
           tmp <= tmp(10 downto 0)& SI;
   end if;
end process;
   PO <= tmp;
```

And the RTL view is shown in Figure 7.2.

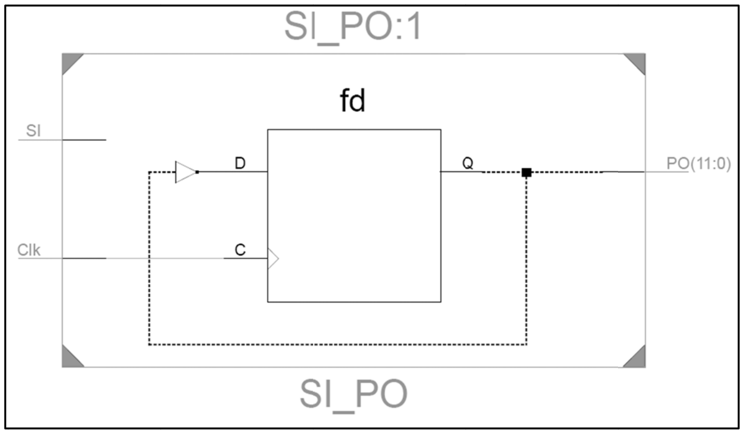


Figure 7.2 . RTL View of SIPO Shift Register, implementation Spartan 3A.

3. 16-to-1 Multiplexer:

M16_1E is a 16-to-1 multiplexer with enable is selected from Xilinx library at the system level. When the enable input(E) is High, the M16_1E multiplexer chooses one data bit from 16 sources (D15-D0) under the control of the select inputs (S3-S0). When E is Low, the output is Low.[12]

The part of VHDL code that shows above describtion is:

```
case S_1 is
     when "0000" => Z <= D1;  --Detector1
     when "0001" => Z <= D2;
    when "0010" => Z <= D3;
    when "0011" => Z <= D4;
    when "0100" => Z <= D5;
    when "0101" => Z <= D6;
     when "0110" => Z <= D7;
     when "0111" => Z <= D8;
     when "1000" => Z <= D9;
     when "1001" => Z <= D10;
     when "1010" => Z <= D11;
     when "1011" => Z <= D12;
     when "1100" => Z <= D13;
```

```
        when "1101" => Z <= D14;
        when "1110" => Z <= D15;
        when "1111" => Z <= D16; --Detector16
        when others => null;
    end case;
```

RTL view is shown in Figure 7.3.

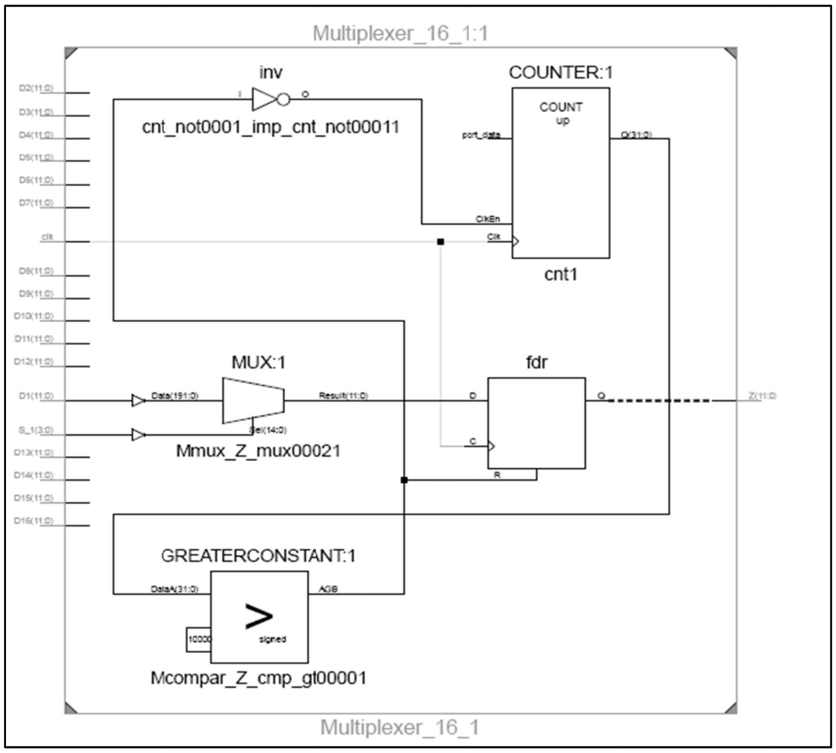


Figure 7.3 . RTL View of SIPO Shift Register.

4. Block Memory RAM:

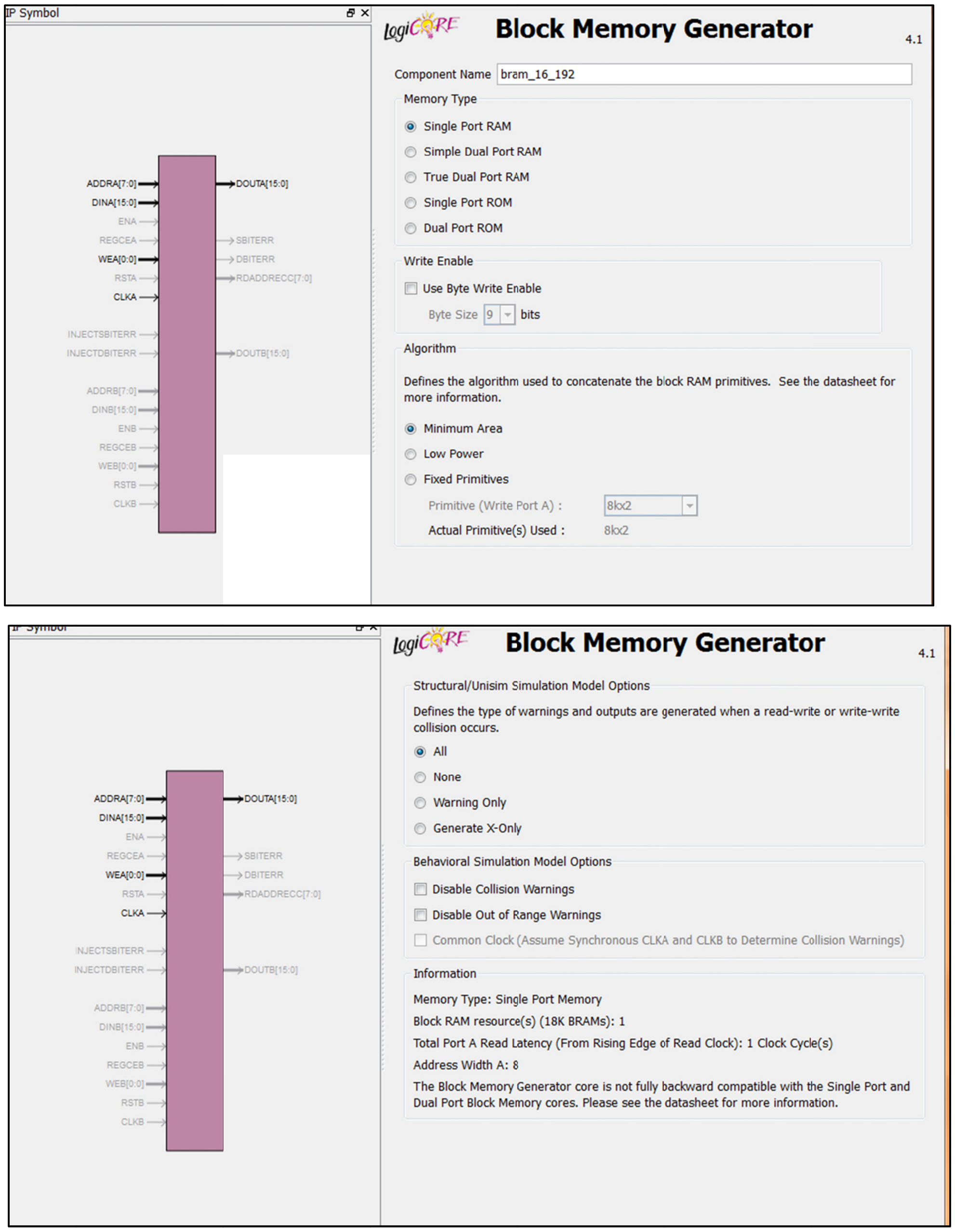
The Block Memory Generator core uses embedded Block Memory primitives in Xilinx FPGAs to extends the functionality and capability of a single primitive to memories of arbitrary widths and depths. Sophisticated algorithms within the Block Memory Generator core produce optimized solutions to provide convenient access to memories for a wide range of configurations. [7]

In level-1, the simple Dual Block Ram (16× 192) is generated to store the image and an 8-bit address by Xilinx Core Generator software. Once the BRAM is generated, this VHDL code must be instantiated to the top level program. Figure 7.4 shows the steps for generating this BRAM.

```
component bram_16_192
        port (
        Clka: IN std_logic;
        wea: IN std_logic_VECTOR(0 downto 0);
        addra: IN std_logic_VECTOR(7 downto 0);
        dina: IN std_logic_VECTOR(15 downto 0);
        douta: OUT std_logic_VECTOR(15 downto 0));
end component;
```

Part of VHDL code for BRAM (16 × 192) is:

```
-- synthesis translate_on
BEGIN
-- synthesis translate_off
U0 : wrapped_bram_16_192
        port map (
        Clka => Clka,
        wea => wea,
        addra => addra,
        dina => dina,
        douta => douta);
-- synthesis translate_on
```



Block Memory Generator 4.1

Component Name: bram_16_192

Memory Type
- [✓ Single Port RAM]
- Simple Dual Port RAM
- True Dual Port RAM
- Single Port ROM
- Dual Port ROM

Write Enable
- Use Byte Write Enable
- Byte Size 9 bits

Algorithm

Defines the algorithm used to concatenate the block RAM primitives. See the datasheet for more information.
- [✓ Minimum Area]
- Low Power
- Fixed Primitives

Primitive (Write Port A) : 8kx2
Actual Primitive(s) Used : 8kx2

Block Memory Generator 4.1

Structural/Unisim Simulation Model Options

Defines the type of warnings and outputs are generated when a read-write or write-write collision occurs.
- [✓ All]
- None
- Warning Only
- Generate X-Only

Behavioral Simulation Model Options
- Disable Collision Warnings
- Disable Out of Range Warnings
- Common Clock (Assume Synchronous CLKA and CLKB to Determine Collision Warnings)

Information

Memory Type: Single Port Memory
Block RAM resource(s) (18K BRAMs): 1
Total Port A Read Latency (From Rising Edge of Read Clock): 1 Clock Cycle(s)
Address Width A: 8
The Block Memory Generator core is not fully backward compatible with the Single Port and Dual Port Block Memory cores. Please see the datasheet for more information.

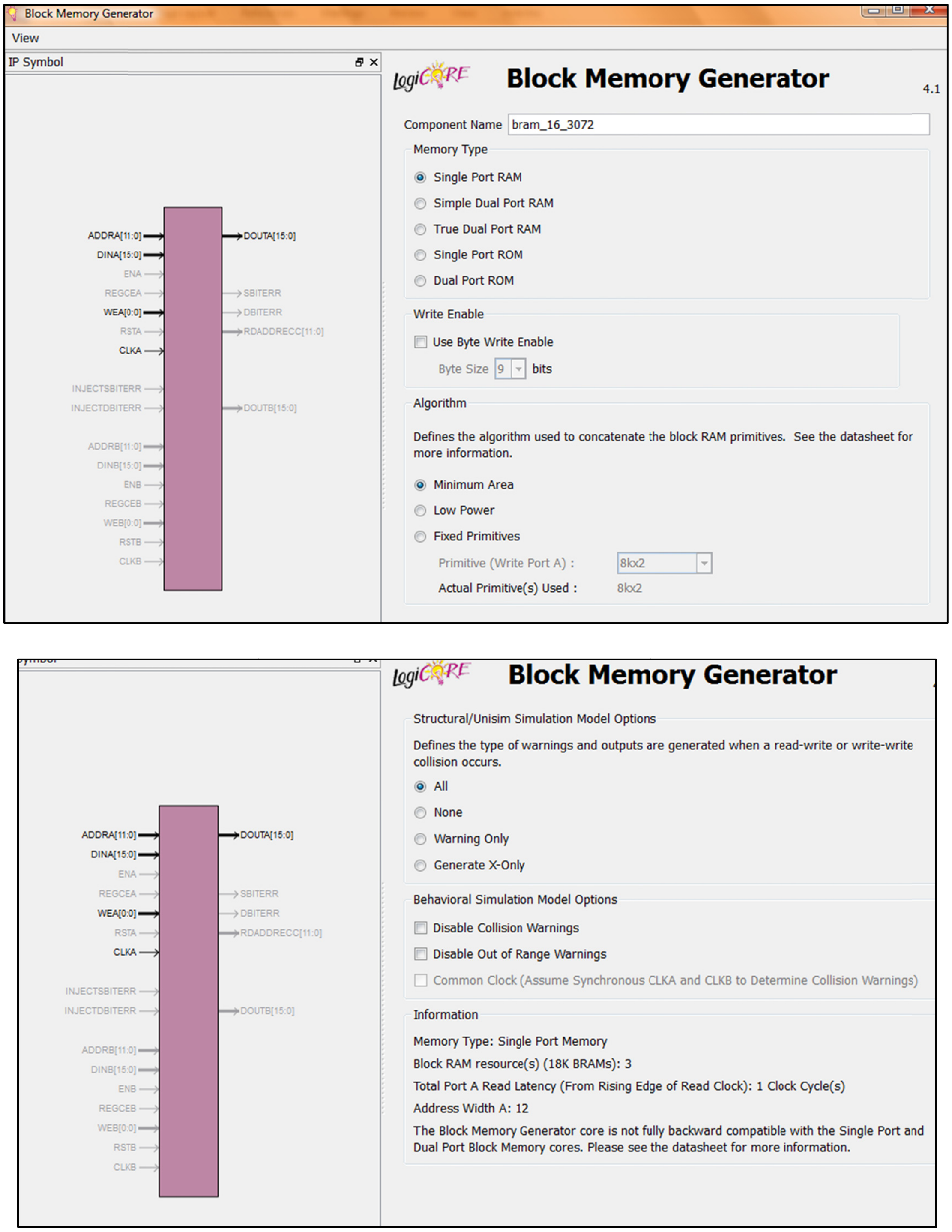
Figure 7.4. Block Memory Generator for BRAM (16 × 192).

In level-2, the simple Dual Block Ram (16× 3072) is generated to store the image by Xilinx Core Generator software. Once the BRAM is generated, this VHDL code must be instantiated to the top level program. Figure 7.5 shows the steps for generating this BRAM.

```
component wrapped_bram_16_3072
          port (
          Clka: IN std_logic;
          wea: IN std_logic_VECTOR(0 downto 0);
          addra: IN std_logic_VECTOR(11 downto 0);
          dina: IN std_logic_VECTOR(15 downto 0);
          douta: OUT std_logic_VECTOR(15 downto 0));
end component;
```

A protion of the VHDL code for BRAM (16 × 3072) is:

```
component bram_16_3072
          port (
          Clka: IN std_logic;
          wea: IN std_logic_VECTOR(0 downto 0);
          addra: IN std_logic_VECTOR(11 downto 0);
          dina: IN std_logic_VECTOR(15 downto 0);
          douta: OUT std_logic_VECTOR(15 downto 0));
end component;
```



Block Memory Generator

Component Name bram_16_3072

Memory Type

[✓ Single Port RAM]
Simple Dual Port RAM
True Dual Port RAM
Single Port ROM
Dual Port ROM

Write Enable

☐ Use Byte Write Enable

Byte Size 9 bits

Algorithm

Defines the algorithm used to concatenate the block RAM primitives. See the datasheet for more information.

[✓ Minimum Area]
Low Power
Fixed Primitives

Primitive (Write Port A) : 8kx2

Actual Primitive(s) Used : 8kx2

Block Memory Generator

Structural/Unisim Simulation Model Options

Defines the type of warnings and outputs are generated when a read-write or write-write collision occurs.

[✓ All]
None
Warning Only
Generate X-Only

Behavioral Simulation Model Options

☐ Disable Collision Warnings
☐ Disable Out of Range Warnings
☐ Common Clock (Assume Synchronous CLKA and CLKB to Determine Collision Warnings)

Information

Memory Type: Single Port Memory
Block RAM resource(s) (18K BRAMs): 3
Total Port A Read Latency (From Rising Edge of Read Clock): 1 Clock Cycle(s)
Address Width A: 12
The Block Memory Generator core is not fully backward compatible with the Single Port and Dual Port Block Memory cores. Please see the datasheet for more information.

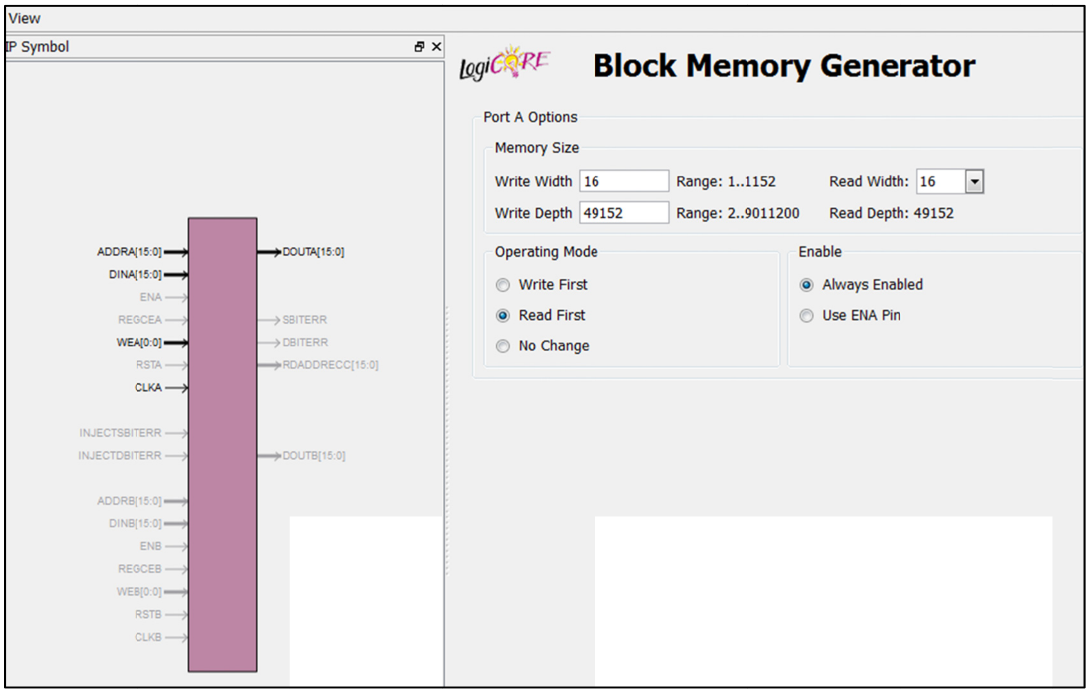
Figure 7.5. Block Memory Generator for BRAM (16 × 3072).

In level-3, the simple Dual Block Ram (16× 49152) is generated to store the image by Xilinx Core Generator software. Once the BRAM is generated, this VHDL code must be instantiated to the top level program. Figure 7.6 shows the steps for generating this BRAM.

```
component wrapped_BRAM_16_49152
        port (
        Clka: IN std_logic;
        wea: IN std_logic_VECTOR(0 downto 0);
        addra: IN std_logic_VECTOR(15 downto 0);
        dina: IN std_logic_VECTOR(15 downto 0);
        douta: OUT std_logic_VECTOR(15 downto 0));
end component;
```

A portion of the VHDL code for BRAM(16 × 49152) is:

```
component BRAM_16_49152
        port (
        Clka: IN std_logic;
        wea: IN std_logic_VECTOR(0 downto 0);
        addra: IN std_logic_VECTOR(15 downto 0);
        dina: IN std_logic_VECTOR(15 downto 0);
        douta: OUT std_logic_VECTOR(15 downto 0));
end component;
```



Block Memory Generator

Port A Options

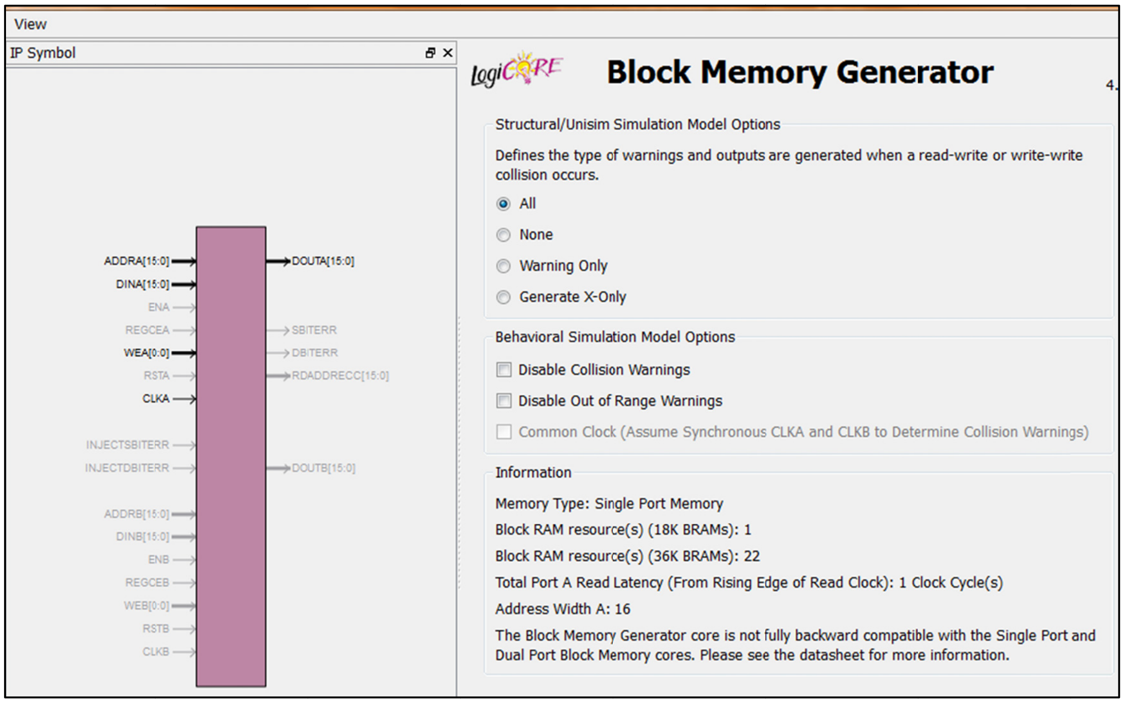
Memory Size

Write Width 16 Range: 1..1152 Read Width: 16

Write Depth 49152 Range: 2..9011200 Read Depth: 49152

Operating Mode: Write First [✓ Read First] No Change

Enable: [✓ Always Enabled] Use ENA Pin



Block Memory Generator

Structural/Unisim Simulation Model Options

Defines the type of warnings and outputs are generated when a read-write or write-write collision occurs.

[✓ All] None Warning Only Generate X-Only

Behavioral Simulation Model Options

- Disable Collision Warnings
- Disable Out of Range Warnings
- Common Clock (Assume Synchronous CLKA and CLKB to Determine Collision Warnings)

Information

Memory Type: Single Port Memory
Block RAM resource(s) (18K BRAMs): 1
Block RAM resource(s) (36K BRAMs): 22
Total Port A Read Latency (From Rising Edge of Read Clock): 1 Clock Cycle(s)
Address Width A: 16
The Block Memory Generator core is not fully backward compatible with the Single Port and Dual Port Block Memory cores. Please see the datasheet for more information.

Figure 7.6. Block Memory Generator for BRAM (16 × 49152).

CHAPTER 8: CONCLUCION

Data acquisition system is the process of sampling analog signals that measure the physical conditions and converting the resulting samples into digital numeric values that can be manipulated by a computer [1]. Data acquisition systems typically convert analog signals into digital values for processing. In this project, a memory mapping process for an image data acquisition system, which is built out of three levels of arrays of FPGA, has been developed. The system has the capability to sample the outputs of large number of photo detectors with high resolution, and at a very high frequency. The received data is sampled, digitalized and passed through a three levels of FPGAs to a fiber channel switch which provides the final high speed serial output. The array has 256 × 256 of photo detectors. Each 16 detectors of the array are grouped to make a 4× 4 photo detector array module. A sample from all detectors will represent one frame. To form one image, 16 frames are required. Therefore, all the detectors of the array are sampled simultaneously sixteen times to form the 16 frames. The analog signal sample out of each detector is converted into 12-bit digital value.

The presented architecture is designed based on three levels FPGAs; in this design the first level has 4096 Spartan3A, the second stage has 256 Spartan3A, and the final level has 16 Virtex6.

The Block Memory Generator core uses embedded Block Memory primitives in Xilinx FPGAs to extend the functionality and capability of a single primitive to memories of arbitrary widths and depths. In level-1, the simple Dual Block Ram is used to store the image, and an 8-bit address is used. The LS four bits of this address defines the place location of each detector in the (4×4) module, and the MS four bits determines the location of module in each frame.

Each 16 level-1 FPGAs transmit 12-bit parallel data to one level-2 Spartan-3A FPGAs. The Block Memory simple Dual ports RAM with the write/read width of 16 and write/read depth of 3072 bits has been generated to store the collected data. The data transition from level-1 to level-2 takes 21.196μs.

In the third level of the architecture, Virtex-6 FPGAs receive data from 16 level-2 FPGAs. The Block Memory simple Dual port RAM (16 ×49,152) has been generated to store the collected data. Data transition from level-2 to level-3 takes 28.44 μs.

The total time required by image acquisition system to collect one frame of data and transfer it to the fiber channel switch is 0.774 μs. This amount of time is less than 1 ms, which meets the requirement. It also gives the system enough time to process the next sample of data.

This work was performed as part of a large project in which several students were involved and worked independently on different part of the project to design and develop the architecture of the system along with memory mapping of the data. My role in this project was the design and development of the memory mapping process which has been done successfully.

REFERENCES

1 Data acquisition system, retrieved January 2012 from URL: http://en.wikipedia.org/wiki/Data_acquisition.

2 S. Thanee S. Somkuarnpanit and K. Saetang, *"FPGA-Based Multi-Protocol Data Acquisition System with High Speed USB Interface"* Proceedings of the international Multi Conference of Engineers and Computer Scientists 2010 Vol II, IMCES 2010, 17, Hong Kong, retrieved January 2012 from URL: www.*pelagiaresearchlibrary*.com

3 Xiao Jun Hu, *"Data acquisition and analysis techniques"; 2010, 6*, retrieved January 2012 from URL: www.sciencedirect.com

4 Yonghai Nig, Zongqiang Guo, Sen Shen, Bo Peng *, "Design of Data Acquisition and storage system Based on the FPGA"*, International workshop on information and electronics engineering (IWIEE), 2012, retrieved January 2012 from URL: www.Sciencedirect.com

5 Texas Instruments "*12- bit octal channel ADC family up to 65 msps data sheet*" 2008, Texas instrument incorporated, retrieved January 2012 from URL: www.ti.com

6 Xilinx "*Spartan3A family overview*", retrieved January 2012 from URL: www.xilinx.com

7 Xilinx "*The Block Memory Generator*", retrieved January 2012 from URL: www.xilinx.com

8 Xilinx "*Virtex family overview*", retrieved January 2012 from URL: www.xilinx.com

9 Xilinx "*XST general guide*", retrieved January 2012 from URL: www.xilinx.com

10 Pong P.CHU, "*RTL Hardware Design Using VHDL coding for Efficiency, Portability, and Scalability*", A JOHN WILEY &SON's, INC., Publication, ISBN-13:978-0-471-72092-8, 2006.

11 Daggu Venkateshwar Rao, Shruti Patil, *"Implementation and Evaluation of Image Processing Algorithms on Reconfigurable Architecture using C-based Hardware Description Languages"* International Journal of Theoretical and Applied Computer Sciences , 2006 Vol I, India, http://www.gbspublisher.com/ijtacs.htm

12 Xilinx "*Library Guide*", retrieved January 2012 from URL: www.xilinx.com

APPENDIX

1. VHDL CODE FOR BRAM_16_192:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
Library XilinxCoreLib;

ENTITY bram_16_192 IS
        port (
        Clka: IN std_logic;
        wea: IN std_logic_VECTOR(0 downto 0);
        addra: IN std_logic_VECTOR(7 downto 0);
        dina: IN std_logic_VECTOR(15 downto 0);
        douta: OUT std_logic_VECTOR(15 downto 0));
END bram_16_192;

ARCHITECTURE bram_16_192_a OF bram_16_192 IS

component wrapped_bram_16_192
        port (
        Clka: IN std_logic;
        wea: IN std_logic_VECTOR(0 downto 0);
        addra: IN std_logic_VECTOR(7 downto 0);
        dina: IN std_logic_VECTOR(15 downto 0);
        douta: OUT std_logic_VECTOR(15 downto 0));
end component;

-- Configuration specification
                        generic map(
                        c_has_regceb => 0,
                        c_has_regcea => 0,
                        c_mem_type => 0,
                        c_rstram_b => 0,
                        c_rstram_a => 0,
                        c_has_injecterr => 0,
                        c_rst_type => "SYNC",
                        c_prim_type => 1,
                        c_read_width_b => 16,
                        c_initb_val => "0",
                        c_family => "spartan3",
                        c_read_width_a => 16,
                        c_disable_warn_bhv_coll => 0,
                        c_use_softecc => 0,
                        c_write_mode_b => "WRITE_FIRST",
                        c_init_file_name => "no_coe_file_loaded",
                        c_write_mode_a => "READ_FIRST",
                        c_mux_pipeline_stages => 0,
                        c_has_softecc_output_regs_b => 0,
                        c_has_softecc_output_regs_a => 0,
                        c_has_mem_output_regs_b => 0,
                        c_has_mem_output_regs_a => 0,
                        c_load_init_file => 0,
                        c_xdevicefamily => "spartan3a",
                        c_write_depth_b => 192,
                        c_write_depth_a => 192,
                        c_has_rstb => 0,
                        c_has_rsta => 0,
                        c_has_mux_output_regs_b => 0,
                        c_inita_val => "0",
                        c_has_mux_output_regs_a => 0,
                        c_addra_width => 8,
                        c_has_softecc_input_regs_b => 0,
                        c_has_softecc_input_regs_a => 0,
                        c_addrb_width => 8,
                        c_default_data => "0",
                        c_use_ecc => 0,
                        c_algorithm => 1,
                        c_disable_warn_bhv_range => 0,
                        c_write_width_b => 16,
                        c_write_width_a => 16,
                        c_read_depth_b => 192,
                        c_read_depth_a => 192,
                        c_byte_size => 9,
                        c_sim_collision_check => "ALL",
                        c_common_Clk => 0,
                        c_wea_width => 1,
                                        c_has_enb => 0,
                                        c_web_width => 1,
                                        c_has_ena => 0,
                                        c_use_byte_web => 0,
                                        c_use_byte_wea => 0,
                                        c_rst_priority_b => "CE",
                                        c_rst_priority_a => "CE",
                                        c_use_default_data => 0);
BEGIN

U0 : wrapped_bram_16_192
                    port map (
                                        Clka => Clka,
                                        wea => wea,
                                        addra => addra,
                                        dina => dina,
                                        douta => douta);

END bram_16_192_a;
component bram_16_192
          port (
          Clka: IN std_logic;
          wea: IN std_logic_VECTOR(0 downto 0);
          addra: IN std_logic_VECTOR(7 downto 0);
          dina: IN std_logic_VECTOR(15 downto 0);
          douta: OUT std_logic_VECTOR(15 downto 0));
end component;

bram_16_192
                    port map (
                                        Clka => Clka,
                                        wea => wea,
                                        addra => addra,
                                        dina => dina,
                                        douta => douta);
```

2. VHDL CODE FOR BRAM_16_3072:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
Library XilinxCoreLib;
ENTITY bram_16_3072 IS
        port (
        Clka: IN std_logic;
        wea: IN std_logic_VECTOR(0 downto 0);
        addra: IN std_logic_VECTOR(11 downto 0);
        dina: IN std_logic_VECTOR(15 downto 0);
        douta: OUT std_logic_VECTOR(15 downto 0));
END bram_16_3072;

ARCHITECTURE bram_16_3072_a OF bram_16_3072 IS

component wrapped_bram_16_3072
        port (
        Clka: IN std_logic;
        wea: IN std_logic_VECTOR(0 downto 0);
        addra: IN std_logic_VECTOR(11 downto 0);
        dina: IN std_logic_VECTOR(15 downto 0);
        douta: OUT std_logic_VECTOR(15 downto 0));
end component;

-- Configuration specification
                generic map(
                        c_has_regceb => 0,
                        c_has_regcea => 0,
                        c_mem_type => 0,
                        c_rstram_b => 0,
                        c_rstram_a => 0,
                        c_has_injecterr => 0,
                        c_rst_type => "SYNC",
                        c_prim_type => 1,
                        c_read_width_b => 16,
                        c_initb_val => "0",
                        c_family => "spartan3",
                        c_read_width_a => 16,
                        c_disable_warn_bhv_coll => 0,
                        c_use_softecc => 0,
                        c_write_mode_b => "WRITE_FIRST",
                        c_init_file_name => "no_coe_file_loaded",
                        c_write_mode_a => "READ_FIRST",
                        c_mux_pipeline_stages => 0,
                        c_has_softecc_output_regs_b => 0,
                        c_has_softecc_output_regs_a => 0,
                        c_has_mem_output_regs_b => 0,
                        c_has_mem_output_regs_a => 0,
                        c_load_init_file => 0,
                        c_xdevicefamily => "spartan3a",
                        c_write_depth_b => 3072,
                        c_write_depth_a => 3072,
                        c_has_rstb => 0,
                        c_has_rsta => 0,
                        c_has_mux_output_regs_b => 0,
                        c_inita_val => "0",
                        c_has_mux_output_regs_a => 0,
                        c_addra_width => 12,
                        c_has_softecc_input_regs_b => 0,
                        c_has_softecc_input_regs_a => 0,
                        c_addrb_width => 12,
                        c_default_data => "0",
                        c_use_ecc => 0,
                        c_algorithm => 1,
                        c_disable_warn_bhv_range => 0,
                        c_write_width_b => 16,
                        c_write_width_a => 16,
                        c_read_depth_b => 3072,
                        c_read_depth_a => 3072,
                        c_byte_size => 9,
                        c_sim_collision_check => "ALL",
                        c_common_Clk => 0,
                        c_wea_width => 1,
                        c_has_enb => 0,
                        c_web_width => 1,
                                        c_has_ena => 0,
                                        c_use_byte_web => 0,
                                        c_use_byte_wea => 0,
                                        c_rst_priority_b => "CE",
                                        c_rst_priority_a => "CE",
                                        c_use_default_data => 0);
BEGIN

U0 : wrapped_bram_16_3072
                    port map (
                                        Clka => Clka,
                                        wea => wea,
                                        addra => addra,
                                        dina => dina,
                                        douta => douta);

END bram_16_3072_a;

component bram_16_3072
          port (
          Clka: IN std_logic;
          wea: IN std_logic_VECTOR(0 downto 0);
          addra: IN std_logic_VECTOR(11 downto 0);
          dina: IN std_logic_VECTOR(15 downto 0);
          douta: OUT std_logic_VECTOR(15 downto 0));
end component;
```

3. VHDL CODE FOR BRAM_16_49152:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

Library XilinxCoreLib;
ENTITY BRAM_3 IS
          port (
          Clka: IN std_logic;
          wea: IN std_logic_VECTOR(0 downto 0);
          addra: IN std_logic_VECTOR(15 downto 0);
          dina: IN std_logic_VECTOR(15 downto 0);
```

```
		douta: OUT std_logic_VECTOR(15 downto 0));
END BRAM_3;

ARCHITECTURE BRAM_3_a OF BRAM_3 IS
component wrapped_BRAM_3
		port (
		Clka: IN std_logic;
		wea: IN std_logic_VECTOR(0 downto 0);
		addra: IN std_logic_VECTOR(15 downto 0);
		dina: IN std_logic_VECTOR(15 downto 0);
		douta: OUT std_logic_VECTOR(15 downto 0));
end component;

-- Configuration specification
						generic map(
						c_has_regceb => 0,
						c_has_regcea => 0,
						c_mem_type => 0,
						c_rstram_b => 0,
						c_rstram_a => 0,
						c_has_injecterr => 0,
						c_rst_type => "SYNC",
						c_prim_type => 1,
						c_read_width_b => 16,
						c_initb_val => "0",
						c_family => "virtex6",
						c_read_width_a => 16,
						c_disable_warn_bhv_coll => 0,
						c_use_softecc => 0,
						c_write_mode_b => "WRITE_FIRST",
						c_init_file_name => "no_coe_file_loaded",
						c_write_mode_a => "READ_FIRST",
						c_mux_pipeline_stages => 0,
						c_has_softecc_output_regs_b => 0,
						c_has_softecc_output_regs_a => 0,
						c_has_mem_output_regs_b => 0,
						c_has_mem_output_regs_a => 0,
						c_load_init_file => 0,
                    c_xdevicefamily => "virtex6",
                    c_write_depth_b => 49152,
                    c_write_depth_a => 49152,
                    c_has_rstb => 0,
                    c_has_rsta => 0,
                    c_has_mux_output_regs_b => 0,
                    c_inita_val => "0",
                    c_has_mux_output_regs_a => 0,
                    c_addra_width => 16,
                    c_has_softecc_input_regs_b => 0,
                    c_has_softecc_input_regs_a => 0,
                    c_addrb_width => 16,
                    c_default_data => "0",
                    c_use_ecc => 0,
                    c_algorithm => 1,
                    c_disable_warn_bhv_range => 0,
                    c_write_width_b => 16,
                    c_write_width_a => 16,
                    c_read_depth_b => 49152,
                    c_read_depth_a => 49152,
                    c_byte_size => 9,
                    c_sim_collision_check => "ALL",
                    c_common_Clk => 0,
                    c_wea_width => 1,
                    c_has_enb => 0,
                    c_web_width => 1,
                    c_has_ena => 0,
                    c_use_byte_web => 0,
                    c_use_byte_wea => 0,
                    c_rst_priority_b => "CE",
                    c_rst_priority_a => "CE",
                    c_use_default_data => 0);
BEGIN
U0 : wrapped_BRAM_3
            port map (
                    Clka => Clka,
                    wea => wea,
                    addra => addra,
                    dina => dina,
                    douta => douta);

END BRAM_3_a;
```

4. D – Multiplexer (1×16):

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity rtl_1 is
          PORT ( I, Clk : in std_logic;   -- input of th edemux
                              S : in std_logic_vector (3 downto 0);   --4 bit address line
                              Y : out std_logic_vector(15 downto 0)); -- 16 out put

end rtl_1;

architecture Behavioral of rtl_1 is

begin
          process (Clk, Y )
          variable cnt : integer range 0 to 16:='0';
          begin
                    if ( Clk'event and Clk='1') then
                         Y(0)  <= I when S="0000" else '0';
           Y(1)  <= I when S="0001" else '0';
           Y(2)  <= I when S="0010" else '0';
           Y(3)  <= I when S="0011" else '0';
           Y(4)  <= I when S="0100" else '0';
           Y(5)  <= I when S="0101" else '0';
           Y(6)  <= I when S="0110" else '0';
           Y(7)  <= I when S="0111" else '0';
           Y(8)  <= I when S="1000" else '0';
           Y(9)  <= I when S="1001" else '0';
           Y(10) <= I when S="1010" else '0';
           Y(11) <= I when S="1011" else '0';
           Y(12) <= I when S="1100" else '0';
           Y(13) <= I when S="1101" else '0';
```

```
        Y(14) <= I when S="1110" else '0';
        Y(15) <= I when S="1111" else '0';

end Behavioral;
```

5. SIPO Shift Register:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity SI_PO is
port(Clk, SI : in std_logic;
   PO     : out std_logic_vector(11 downto 0));
end SI_PO;

architecture beh of SI_PO is
signal tmp: std_logic_vector(11 downto 0);
begin
        process (Clk)
        begin
                if (Clk'event and Clk='1') then
                        tmp <= tmp(10 downto 0)& SI;
                end if;
        end process;
PO <= tmp;
end beh;
```

6. Multiplexer (16 ×1) :

```
library ieee;
use ieee.std_logic_1164.all;
entity Multiplexer_16_1 is
  port (D1,D2,D3,D4,D5,D6,D7,D8,D9,D10,D11,D12,D13,D14,D15,D16 : in std_logic_vector(11 downto 0);
           Clk : in std_logic;
     S_1 : in std_logic_vector (3 downto 0);
     Z : out std_logic_vector(11 downto 0));
end Multiplexer_16_1;
```

```
architecture Behavioral of Multiplexer_16_1 is
signal cnt : integer range 0 to 16  ;

begin
process (Clk,D1,D2,D3,D4,D5,D6,D7,D8,D9,D10,D11,D12,D13,D14,D15,D16)
  begin
             if ( Clk'event and Clk='1' ) then
               if ( cnt > 16 ) then
                         z<=(others=>'0');
                         else
                         cnt<=cnt+1;

    case S_1 is
      when "0000" => Z <= D1;  --Detector1
      when "0001" => Z <= D2;
                                 when "0010" => Z <= D3;
                                 when "0011" => Z <= D4;
                                 when "0100" => Z <= D5;
                                 when "0101" => Z <= D6;
      when "0110" => Z <= D7;
      when "0111" => Z <= D8;
      when "1000" => Z <= D9;
      when "1001" => Z <= D10;
      when "1010" => Z <= D11;
      when "1011" => Z <= D12;
      when "1100" => Z <= D13;
      when "1101" => Z <= D14;
      when "1110" => Z <= D15;
      when "1111" => Z <= D16;
      when others => null;
    end case;
                         end if;
            end if;
  end process;

end Behavioral;
```