CALIFORNIA STATE UNIVERSITY, NORTHRIDGE

32 x 32 ARRAY PHOTO DETECTOR IMAGE DATA ACQUISITION SYSTEM

A graduate project submitted in partial fulfillment of the requirements

For the degree of Masters of Science in
Electrical Engineering

By
Sahitya Venkatayogi
May 2014
The graduate project of Sahitya Venkatayogi is approved:

__________________________  ______________________
Dr. Prof. Ali Amini  Date

__________________________  ______________________
Prof. Vijay Bhatt  Date

__________________________  ______________________
Chair Dr. Prof. Nagi El Naga  Date

California State University, Northridge
ACKNOWLEDGMENTS

It is a feeling of pride to complete my project under the guidance of Dr Nagi El Naga, my project advisor and graduate coordinator. The support and help that was shown by Dr Nagi El Naga throughout the project is greatly appreciated. I thank Dr Nagi El Naga for giving me an opportunity to work in my own way.

I would like to show my gratitude to the rest of the committee professors, Dr prof. Ali Amini and prof. Vijay Bhatt for their support and help in tough times.

I thank my father Raghavendra Swamy Venkatayogi for his incredible support both morally and financially, he stood by me in all situations which helped to complete this project successfully.

Last but not the least I want to thank Bart Kowalski for his help and encouragement in completing this project. I would also like to thank Karthik Malekal for explaining the basic digital concepts in the beginning of stages of my masters with such patience that intrigued me to get specialized in digital system design.
## Table of Contents

SIGNATURE PAGE .................................................................................................................. II

ACKNOWLEDGMENTS ........................................................................................................... III

ABSTRACT ............................................................................................................................... X

CHAPTER 1 : INTRODUCTION ................................................................................................. 1

1.1 INTRODUCTION TO DATA ACQUISITION SYSTEMS ...................................................... 1
1.2 OBJECTIVE ....................................................................................................................... 4
1.3 PROJECT OUTLINE ........................................................................................................... 6

CHAPTER 2 : SYSTEM SPECIFICATIONS AND ANALYSIS .................................................... 7

2.1 SYSTEM SPECIFICATIONS ............................................................................................... 7
2.2 SPECIFICATION ANALYSES ............................................................................................ 8

CHAPTER 3 : DESIGN OF IMAGE DATA ACQUISITION SYSTEM .......................................... 10

3.1 TOP-LEVEL ARCHITECTURE ............................................................................................ 10
3.1.1 First Stage: Conversion of Analog Signals to Digital Signals ........................................ 12
3.1.2 Second Stage: Conversion of LVDS Digital Signals to LVCMOS I/O Single-Ended Signal Standard .................................................................................................................. 15
3.1.3 Third Stage or First Level of FPGA .............................................................................. 18
3.1.4 Fourth Stage of Design or Second Level of FPGA ....................................................... 20

CHAPTER 4 : COMPONENT DESCRIPTION ............................................................................. 21

4.1 PHOTO DIODE: .................................................................................................................. 21
4.2 ANALOG TO DIGITAL CONVERTER: .............................................................................. 23
4.2.1 Register Initialization: .................................................................................................. 24
4.2.2 Power-Up Sequencing and Reset Timing: ................................................................... 25
4.2.3 The functional diagram of ADC: ................................................................................. 25
4.3 HIGH SPEED DIFFERENTIAL LINE RECEIVERS: ......................................................... 27
4.4 20–BIT FET BUS SWITCH: ............................................................................................... 30

CHAPTER 5 : XILINX FPGAS ................................................................................................. 32

5.1 CHOICE OF FIRST LEVEL OF FPGAS: ........................................................................... 32
5.2 CHOICE OF SECOND LEVEL OF FPGAS: ...................................................................... 33
5.3 VIRTEX-6 FPGAS ............................................................................................................ 34
5.4 SERIAL PROTOCOLS: ....................................................................................................... 35
5.4.1 PCIe Interface .............................................................................................................. 35

CHAPTER 6 : SIMULATION AND MODELING OF A DESIGN .................................................. 37

6.1 SIMULATION AND MODELING OF AN A/D CONVERTER ........................................... 37
6.1.1 Power-Up Sequence: .................................................................................................. 38
6.1.2 Serial Interface ............................................................................................................ 39
6.1.3 Analog-to-Digital Conversion ..................................................................................... 42
6.2 MODELING AND SIMULATION OF HIGH-SPEED DIFFERENTIAL LINE RECEIVER ...... 46
6.3 Modeling and Simulation of First Level of FPGAs ........................................ 48
  6.3.1 Serial In Parallel Out ................................................................................. 50
  6.3.2 Multiplexer in Level1 .................................................................................. 52
  6.3.3 Writing Data to Block RAM from Multiplexer .......................................... 53
  6.3.4 Placing a Detector in Level-1 Block RAM .................................................. 54
  6.3.5 Reading the First-Level Data to Transfer into Second Level of FPGAs .... 57

6.4 Modeling and Simulation of the Second Level of FPGAs .......................... 61
  6.4.1 Writing the Data to Level-2 Block RAM from Level-1 FPGAs ................. 62
  6.4.2 Reading the Stored Data and Transferring to Final Block RAM .............. 63
  6.4.3 Writing the Data into Final Block RAM ..................................................... 64
  6.4.4 Resetting the Entire Design ...................................................................... 66

CHAPTER 7 : CONCLUSION .................................................................................. 69
REFERENCES ........................................................................................................ 71
APPENDIX ............................................................................................................. 73
List of Figures

Figure 1-1: Block diagram of hardware design of the high-speed data acquisition system [1]......3
Figure 1-2: Top-level block diagram of architecture.................................................................5
Figure 2-1: Arrangement of array of 32x32 photo detector........................................................8
Figure 2-2: 4x4 photo detector array module (8 x 8 = 64 modules). ............................................9
Figure 2-3: Sampling pulse with 18 samples..............................................................................9
Figure 3-1: Top-level architecture of 32x32 data acquisition system........................................11
Figure 3-2: Stage 1 in data acquisition system for 4x4 module..................................................12
Figure 3-3: LVDS timing diagram [4].......................................................................................14
Figure 3-4: Stage-2 conversion of LVDS signals to LVCMOS I/O standard..............................15
Figure 3-5: Output timing characteristics of high-speed differential receiver [5].......................16
Figure 3-6: Typical DC voltage characteristics..........................................................................17
Figure 3-7: First level of FPGAs or third stage of FPGAs..........................................................19
Figure 3-8: Fourth stage, second level of FPGA.......................................................................20
Figure 4-1: Picture of typical Photo Diode................................................................................22
Figure 4-2: serial interfacing diagram.......................................................................................25
Figure 4-3: Recommended power-up sequencing and reset timing for ADS528X....................25
Figure 4-4: Function diagram of ADS5281...............................................................................26
Figure 4-5: Initialization registers...............................................................................................27
Figure 4-6: Top view of SN65LVDS Family..............................................................................28
Figure 4-7: Output characteristics of SN65LVDS386.................................................................29
Figure 4-8: Pin Configuration of SN74CB3T16210.................................................................31
Figure 4-9: Typical DC voltage characteristics device SN74CB3T16210...............................31
Figure 5-1: Comparison of technology and data transfer rate [6]..............................................36
Figure 6-1: RTL view of an A/D converter...............................................................................37
Figure 6-2: Power sequencing block..........................................................................................38
Figure 6-3: Simulation window showing the assertion of power “seq_done.”

Figure 6-4: (a) Top view of Serial Interface block (b) RTL view of Serial Interface Process.

Figure 6-5: Showing how data bits and address bits have been loaded into register.

Figure 6-6: Figure shows the RTL view of total analog-to-digital conversion process.

Figure 6-7: Window showing the serializer output.

Figure 6-8: Window showing the output digital data and their phase relation with ADclk and Lclk.

Figure 6-9: Phase relation between output data and ADclk and Lclk.

Figure 6-10: SDR interface modes.

Figure 6-11: Differential Input Buffer primitive.

Figure 6-12: RTL view of SN65LVDS386.

Figure 6-13: Single-ended signal output.

Figure 6-14: Block diagram of data samples writing into block RAM in level1.

Figure 6-15: RTL view of the block diagram in Figure 6-13.

Figure 6-16: Window shows how parallel data will read out at ADclk.

Figure 6-17: Multiplexer in level1 selects one of the outputs of serial in and parallel out.

Figure 6-18: Block diagram of writing process to block RAM in Level1.

Figure 6-19: Placing of each detector in its module.

Figure 6-20: Block diagram of Level-1 FPGA.

Figure 6-21: Block Diagram showing the reading of the data from block RAM.

Figure 6-22: Multiplexer to select the one of the block RAM outputs.

Figure 6-23: Complete block diagram of Level-2 FPGA.

Figure 6-24: Block diagram shows the writing process to initial block RAMs.

Figure 6-25: Block diagram showing the reading process.

Figure 6-26: Writing data to final block RAM.

Figure 6-27: Numbering of modules in 32 x 32 array Photo detectors.
Figure 6-28: RTL view of the reset_entire_design. ................................................................. 68

Figure 6-29: Simulation window showing the reset going high at 60,000 clock cycles........... 68
List of Tables

Table 1: Common sensors and their phenomenon............................................................. 1

Table 2: Selection of correct FPGA for the first level of FPGA, XC6VLX240T .......... 33

Table 3: Selection of correct FPGA for the second level of FPGA, XC6VLX240T....... 33

Table 4: Placing of each detector sample from ADC in Level1 FPGA......................... 55
Abstract

32x32 ARRAY PHOTODETECTOR IMAGE DATA ACQUISITION SYSTEM

By

Sahitya Venkatayogi

Master of Science in Electrical Engineering

A data acquisition system collects data from a large number of photo detectors at a very high frequency with high resolution. The received data are sampled, digitalized, and processed at two levels of Field Programming Gate Arrays (FPGAs), and finally the data are sent to a Personal Computer (PC) for further processing through a Peripheral Component Interconnect Express (PCIe) interface.

At California State University, Northridge, I received an opportunity to design and model an image data acquisition for 32x32 array photo detectors. The main focus of this project is to reduce the cost of implementation compared to the cost of previous architectures. In this system, the data flow from photo detectors to the computer is explained in four stages. The design has two levels of FPGAs: two in the first level (also called the “third stage of the design”) and one in the second level (also called the “fourth stage”).

The array has 32x32 photo detectors spaced 5mm apart. Arrays of photo detectors are formed using 4x4 sub arrays. Each 4x4 sub array constitutes a single module. Each data sample makes one frame. Eighteen frames of data are required to form an image. Therefore, all the photo detectors are sampled 18 times to make 18 frames. Each sample is converted to a 12-bit digital value and stored in the block RAM. In this architecture, 64
ADCs have to be interfaced with a single FPGA, which has been a significant challenge in data acquisition systems. The modeling and simulation of the design are done using Xilinx 14.6 ISE design suite.
Chapter 1 : Introduction

1.1 Introduction to Data Acquisition Systems

A data acquisition system is a process of acquiring physical signals and converting these signals into digital signals that can be easily processed by a computer. These physical signals are generally analog in nature such as temperature, wind, speed, humidity, etc. Vast advancements in computer technology provide an advantage to convert these physical signals into digital signals. Also, digital signals have better immunity to noise and thus can be saved and easily reproduced [1].

In general, a data acquisition system consists of the following main parts:

a) Sensors: A sensor is also called a “transducer,” converting the physical data into electrical signals. These electrical signals can be voltage, current etc. Electrical output will be based on the type of sensor.

<table>
<thead>
<tr>
<th>Sensor</th>
<th>Phenomenon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thermocouple, RTD, Thermistor</td>
<td>Temperature</td>
</tr>
<tr>
<td>Photo Sensor</td>
<td>Light</td>
</tr>
<tr>
<td>MicroPhone</td>
<td>Sound</td>
</tr>
<tr>
<td>Strain Gauge, Piezoelectric Transducer</td>
<td>Force and Pressure</td>
</tr>
<tr>
<td>Potentiometer, LVDT, Optical Encoder</td>
<td>Position and Displacement</td>
</tr>
<tr>
<td>pH Electrode</td>
<td>pH</td>
</tr>
</tbody>
</table>

Table 1: Common sensors and their phenomenon.

b) Signal Conditioning: The analog electrical signals from sensors may not be ready for conversion to digital signals. These signals may have high or low amplitude or
be too noisy. Based on the nature of the incoming signal, the conditioning circuit performs the required operation such as amplification, attenuation, noise reduction, multiplexing, etc.

c) Analog-to-Digital Conversion (ADC): This is the most important part of a data acquisition system. The collected analog data samples are converted to digital signals using ADC, where the nature of the output data depends upon the type of ADC selected for conversion. In high-speed data converter applications, digital data output will be in high-speed serial Low-Voltage Differential Signaling (LVDS). Analog-to-digital converters use various methods or architectures such as flash, successive approximation, delta encoded, pipelined, sigma delta, etc.

Examples of some high-speed conversion devices are A/D converters (ADCs) like ADS5281, ADS5240, and ADS5270.

d) Field Programming Gate Array (FPGA): After the conversion of analog signals to digital signals, a device must be present to store the data in its memory before it is given to a computer for further processing. All the samples of data are collected simultaneously and processed in the FPGAs. Choice of a FPGA is a crucial part in the data acquisition system and is generally based on I/O pin count, GTX transceivers count, block RAM memory, and the cost of the device.

e) Peripheral Component Interconnect Express (PCIe) Interface: Data are sent to a computer using various methods like USB, SPI, PCI Express, I2C, Wi-Fi, and Ethernet. Peripheral Component Interface Express, also referred to as “PCIe,” is a general-purpose serial I/O interconnect that can be leveraged for communications, embedded, storage, server, mobile, and desktop applications [6]. In the final stage
of the data acquisition system, the processed data will be sent to a PC by a PCIe interface.

Various algorithms have been proposed to process and present data in a meaningful way. The size, complexity, and cost of a data acquisition system vary based on the application. In applications for medical, military, or space, accuracy of the system is very critical, so these systems have complex signal conditioning and use a wider resolution to represent analog data. In consumer appliances or vehicles applications, cost is more important than accuracy.

Figure 1-1: Block diagram of hardware design of the high-speed data acquisition system [1].

Figure 1-1 depicts the flow of data in a data acquisition system. Sensors convert physical signals to analog electrical signals. These analog electrical signals may not be ready for digital conversion. Analog electrical signals need to be amplified, attenuated, etc. in a signal conditioning circuit before they are given to the analog-to-digital converter (ADC). The ADC converts these analog electrical signals to digital signals. FPGA receives these digital signals for further processing; these digital signals are
captured simultaneously, processed at two levels of FPGA, and finally sent to a computer through PCI Express protocol in a high-speed data stream.

1.2 Objective

The objective of this project is to design and model an image data acquisition system for 32x32 array photo detectors. My first step is to study all information about digital cameras and their working principles. After gaining sufficient knowledge, I will review previous architectures. The main goal is to reduce the cost of implementation compared to the cost of previous architectures.

The design should be capable of reading and collecting data from an array of photo detectors. This design has an ADC that converts samples of analog signals to digital signals. In total, 18 frames must be collected, digitalized, processed, and finally passed through a PCIe interface that provides a high-speed data serial output. These 18 frames are collected from each photo detector. Each analog sample is converted into a twelve-bit digital signal. The array has 32x32 photo detectors, a total of 1,024 photo detectors. Arrays of photo detectors are formed using 4x4 sub arrays. Each 4x4 sub array constitutes a single module.

This design is modeled and simulated in 14.6 ISE design suite Xilinx software. The analog-to-digital converter and high-speed differential receiver are also modeled in Xilinx software. The output waveforms of the above devices should match exactly with the waveforms in the data sheets of the devices.

A pulse comes in, initiating sampling at about 1 KHz. The 18 received samples of each photo detector must be processed in 1 millisecond. After 1 millisecond, the next
data sampling process initiates. There are a total of 18,432 samples that must be
digitalized and processed before the next data sampling process begins.

Figure 1-2: Top-level block diagram of architecture.

Figure 1-2 depicts information about the data flow in the design. In this project,
the design/modeling part of the data acquisition system is emphasized.

The total number of photo detectors (32 x 32) is 1,024. Eighteen samples of each
photo detector are digitalized in the analog-to-digital converter (ADC). The operating
frequency of the ADC in this design is 12MHz. The data output of the ADC is in 12-bit
resolution per sample. The FPGA in level1 receives the digital signals and store all the
data. These data will be sent to the second-level FPGA at a speed of 144mbps. The
second level of the FPGA receives data from all first levels of FPGAs and stores all the
data samples in a single block RAM where address of each detector data samples are
defined. In the next step, stored data in the second level of FPGAs will be read at a 2-
Gb/sec data rate and finally sent to a computer through the PCIe interface for further
processing.
1.3 Project Outline

This project begins with the general description of data acquisition systems and block diagrams. Some important applications of a data acquisition system have been discussed. A block diagram has been shown and explained in a very good manner.

Chapter 1 gives an introduction to data acquisition systems and their importance. The objective of the project is presented in this chapter.

Chapter 2 discusses the specifications of 32x32 array photo detector image data acquisition.

Chapter 3 is the main chapter that discusses the design part of 32x32 array photo detector image data acquisition.

Chapter 4 describes the components used in the design. Each component is briefly explained, and timing constraints are also discussed.

Chapter 5 states the reasons for choosing specific FPGAs for the first and second level of FPGAs and a serial protocol to transfer the data from FPGA to pc for further processing.

Chapter 6 presents modeling and simulation of the design using software Xilinx ISE design suite.

Chapter 7 concludes the project and provides some modifications that can further improve the system.
Chapter 2: System Specifications and Analysis

2.1 System Specifications

The system design and requirements for an image data acquisition system are as follows:

- 32x32 array of photo detectors spaced 5mm apart
- Sixteen photo detectors of the array grouped to make a single 4x4 array module
- Sampling pulse comes into initiate sampling at 1KHz
- 18 samples per detector at sampling frequency greater than 3MHz and 10--14 resolution bits per sample
- Data sampled, digitalized, and read before the next data set begins
- Data sent to a PC through an interface at a 2-Gb/sec data rate
2.2 Specification Analyses

32x32 array photo detectors are spaced 5mm apart, in total a length of 0.16m x 0.16m. The total number of photo detectors is 1,024. Each 4x4 sub array constitutes one module; hence 32x32 arrays are constructed using 8x8 array modules.

Figure 2-1: Arrangement of array of 32x32 photo detector.

The next specification describes a module of 16 photo detectors. Hence, 32x32 photo detectors system is constructed with 8x8 modules.
Figure 2-2: 4x4 photo detector array module (8 x 8 = 64 modules).

The next specification demonstrates a pulse coming into initiate sampling at 1KHz and 18 samples per detector at 12MHz.

Figure 2-3: Sampling pulse with 18 samples.

The amount of data received per sample pulse:

\[ \text{32x32 x 12 bits/frame x 18 frames} = 1,247,232 \text{ bits/sampling pulse} \]

Each photo detector gives 18 samples, and these samples are digitalized in the ADC with 12-bit resolution per sample. So, a total of 1,024 photo detector samples are digitalized, resulting in 1,247,232 bits. These total bits should be read before the next sampling pulse initiates.
Chapter 3: Design of Image Data Acquisition System

3.1 Top-Level Architecture

This architecture is designed in four stages to transmit the data from a photo detector to a computer through a serial protocol PCIe interface. Each stage has its own significance in processing the data.

In the first stage, a total of 128 analog-to-digital converters are connected to 1,024 photo detectors. Each 4x4 module is connected to two ADCs. In the second stage, LVDS output of 128 ADCs are connected to 128 high-speed differential receivers. These high-speed differential receivers are connected to two FPGAs in the third stage, which is also called the “first level of FPGAs.” In this process, 64 high-speed differential receivers are connected to one FPGA and the rest to the second one. In the fourth stage, the second level of FPGAs, two FPGAs are used to receive data from the first stage, and the data are finally sent to a PC through a PCIe interface.
Figure 3-1: Top-level architecture of 32x32 data acquisition system.
3.1.1 First Stage: Conversion of Analog Signals to Digital Signals

Figure 3-2: Stage 1 in data acquisition system for 4x4 module.

Note: “Channel p & n” indicates channel positive and negative terminals, shown on two lines.
As explained earlier, the design of the data acquisition system is divided in four stages for easy understanding. In the first stage, shown in Figure 3-2, 64 ADCs are interfaced to one FPGA; for easy analysis, only two ADCs from the 4x4 module photo detectors are shown in Figure 3-2. In this stage 1, analog data samples from photo detectors are being converted to digital signals. FPGA sends a start signal acquisition, which will activate “Chip Select” (CS) and “Busy” signals. ADS5281 is an analog-to-digital converter used in the design to convert analog signals to digital signals, a serial bit stream with 12-bit ADC resolution provided along with a high-speed bit clock and frame clock. The frequency of the high-speed bit clock depends on the ADC’s resolution and input sampling rate. The frame clock is a digitized version of the analog sample clock, and this clock maintains exact synchronization with data. Twelve bits from each channel are serialized and latched out by the system clock on a pair of pins in LVDS format. A high-speed clock or a bit clock latches the output data, so the phase difference between the clock and data will be determined by the amount of propagation delay specified at that sample rate (12 MSPS for the design). The positive edge of the frame clock represents the start of the frame. The system clock (Lclk) or bit clock is the main clock that we use inside the FPGA to capture the data, and the ADCLK or frame clock is used to identify the start of the frame.

The system clock (Lclk) or bit clock will be six times faster than the frame clock (ADCLK) and input sampling frequency (Clk). Output data will be latched on a positive edge and negative edge of the system clock in Double Data Rate (DDR) format. However, we can program the ADC to output the data in Single-Date Rate (SDR) format by configuring the ADC.
Figure 3-3: LVDS timing diagram [4].

Figure 3-3 shows the LVDS output timing diagram. In the design, the operating frequency of ADS5281 is 12MHz. So, ADC samples the analog signals at a speed of 12 MSPS. The frame clock will be the input frequency, and other control signals like CS and the Busy signal will be controlled by a FPGA. There will be six-clock-cycles latency (six times per system clock period). The frame clock is a 180°-phase shifted version of the clock input.

In Figure 3-2, a 4x4 module photo detector is connected to two ADCs since each ADC has only eight analog input channels. Hence, there would be 128 ADCs required for 32x32 array photo detector data acquisition. The data from the ADC converter are sent to the next stage, where the data are converted to 2.5-V input and output (I/O) standard signal.
3.1.2 Second Stage: Conversion of LVDS Digital Signals to LVCMOS I/O Single-Ended Signal Standard

As shown in Figure 3-4, the conversion of LVDS signals to LVCMOS I/O standard signals is carried out using the SN65LVDS386 high-speed differential receiver. This device meets all design requirements and is suitable for the conversion process.

Figure 3-4: Stage-2 conversion of LVDS signals to LVCMOS I/O standard.

Figure 3-4 depicts the conversion of LVDS signals to 2.5-V I/O standard signals. SN65LVDS386 is a high-speed differential receiver that meets all design requirements.
In this stage, a total of 128 SN65LVDS386s are required to convert the LVDS signals to a 3.3-V single-ended I/O standard signal. For easy understanding, one 4x4 array photo detector is taken into consideration and connected to two differential receivers. In stage two, a high-speed differential receiver receives the LVDS signals from ADC and converts those signals to 3.3 V I/O standard signals.

![Diagram of high-speed differential receiver](image)

**Figure 3-5: Output timing characteristics of high-speed differential receiver [5].**

$V_{IA}$ and $V_{IB}$ are input differential signals. $V_{IA}$ symbolizes high-level input voltage, $V_{IB}$ low-level input voltage, and $V_o$ the output voltage with $V_{OH}$ and $V_{OL}$ as high-level output voltage and low-level output voltage, respectively. In Figure 3-5, $V_{IA} = 1.4$-V and $V_{IB} = 1$-V are the inputs to the device, $V_{OH} = 3$-V and $V_{OL} = 0$-V are the outputs of the device, and 1.5-Vs the output common mode voltage. Output voltages are assumed to be typical values. In normal conditions, $V_{OH}$ will be 2.4-V.
All the data channels, bit clock, and frame clock should be passed through high-differential receivers. The reason for not routing the bit clock and frame clock directly to the FPGA is that there may be a chance for setup and hold time violations in a FPGA. The typical output voltage level values from a differential receiver may not be compatible with I/O standards of a Virtex-6 Xilinx FPGA because a Virtex-6 FPGA accepts only 2.5-V LVCMOS I/O standard single-ended signals. The output from high-speed differential receivers will be 3.3-V I/O standard single-ended signals; hence, these signals are not directly compatible with a Virtex-6 FPGA. There should be an interfacing switch that attenuates the high-level voltages of 3.3-V to 2.5-V and make them compatible with the I/O standards of a Virtex-6 FPGA.

SN74CB3T16210 is a high-speed TTL-compatible FET bus switch with low ON-state resistance, allowing for minimum propagation delay. It is like a voltage-level translator that translates from one voltage level to another. SN74CB3T16210 is organized as two 10-bit bus switches with separate output-enabled inputs. It can be used as two 10-bit bus switches or as one 20-bit switch.

Figure 3-6: Typical DC voltage characteristics.
Whenever the input voltage is beyond the 2.5-V range, it is attenuated to below 2.5-V or near to Vcc levels.

3.1.3 Third Stage or First Level of FPGA

In the third stage, or the first level of FPGAs, only two FPGAs are used. In the previous architectures, the number of FPGAs used was twice the number of FPGAs used in the present architecture, so the cost of implementation of this architecture is greatly reduced.

In this third stage, two Xilinx Virtex-6 FPGAs with part number XC6VLX240T is used, the FPGAs receive data from 1,024 detectors (512 detectors are processed on each FPGA). The first level of FPGAs is shown in Figure 3-7. The first level of FPGAs receives data from a 20-bit FET switch. The I/O standards of the received data meet all the requirements of I/O standards of the Xilinx Virtex-6 FPGA.

The received data will be in a 2.5-V LVCMOS single-ended I/O standard signal. In the first level of FPGAs, data will be sent to serial in and parallel out where data will be parallel latched on a frame clock. From serial in and parallel out, data will be sent to block RAM (12x144) for storage.
Data will be read from the block RAM in level1 FPGAs at a system frequency of 144 MHz and transmitted to the next level of FPGAs; hence, the 12-bit parallel data stream will be at 144mbps. Address of each data sample is defined in the block RAM of level1 and which will be seen in detail in the memory part in Chapter 6.
### 3.1.4 Fourth Stage of Design or Second Level of FPGA

In the second level of FPGA, only one FPGA has been used to receive the data from the first level of FPGAs. Here, Xilinx Virtex-6 with part number XC6VLX240T, package number FF484, is used in the design.

![LEVEL-2 FPGA Diagram](image)

**Figure 3-8: Fourth stage, second level of FPGA.**

This is the most prominent stage in the data acquisition system. In the second level of FPGAs, only one FPGA receives the data simultaneously from the first level of FPGAs and stores those in two block RAMs (12x9216). The data from two block RAMs are read out and finally stored in only one block RAM where the address of each detector sample is defined, and these complete data are read and transmitted to the computer at two Gb/sec through the PCIe interface. Each data sample of 1024 detectors address is defined in block RAMs of level2 FPGAs and which will be seen in detail in the chapter 6 memory mapping part.
Chapter 4 : Component Description

4.1 PHOTO DIODE:

Photodiode is a light sensor which converts light into voltage or current. Photo Diode is a P-N junction diode which is operated in Reverse-Bias. Reverse current flows, when light falls on the junction of the diode.

Photo diode types:

1. PN photodiodes: An SiO₂ is applied to the P-N junction surface, which produces a photodiode with a low level dark current.

2. PIN photodiodes: It is an improved version of the PN type photodiodes. It uses high resistance layer at the P-N junction surface to improve response time.

3. Schottky Photodiode: These kinds of diodes have an advantage of having a thin gold layer between P and N junction.

4. Avalanche Type: A very high reverse bias voltage is applied to accelerate the photo carriers. Carriers collide with atoms to produce Avalanche break down.
Typical photodiodes are packaged in a similar manner as the above image. The ratio of the active area aperture to the total area of the device (including package) is less than 10%, which is usually not favorable. While this type of solution has an advantage in being an off-the-shelf part and is easy to fit with a filter, it is otherwise extremely inefficient in its use of area, and in large quantities, the extra cost of the packaging outweighs the savings of it being an off the shelf part.

In quantities of 100 from OSI (part number: FCI-InGaAs-3000-X) $315 each. In large quantities from OSI (50k to 100K), with custom packaging, the price drops to about $100 each. OSI Optoelectronics is located in Hawthorne, CA.

In large quantities from Pacific Sensors (50k to 100K), with custom packaging, the price also drops significantly. Pacific Sensors is also capable of providing a complete design and fabrication of the entire module, including pre-amp, etc. Pacific Sensors is located in West Lake Village, CA.
These are not Avalanche Photo Diodes (APDs). These are wide active area in gallium arsenide (GaAs) photodiodes with a responsively of about 0.95A/W at 1550nm. APDs typically have a responsively starting at about 10A/W, and up depending on the gain of the design (>3000A/W). However, noise and dark current issues can greatly offset the advantage of the higher sensitivity. Generally, all of these devices have sufficiently fast response times to function well within our 3MHz design. Another issue with APDs that should be noted: they usually need a bias with a much higher voltage, about ten times the voltage than a standard photodiode requires, such as 50V to 75V for an APD, rather than 3V to 7V for the standard photodiodes. This high voltage requirement can increase the costs of the support circuits considerably.

4.2 Analog to Digital Converter:

In this design, we use ADS5281 for analog to digital conversion. ADS5281 is low power, 12 bit resolution, with serialized low voltage differential signaling (Lvds) outputs.

The important Specifications:

- Area: 9mm × 9mm QFN package.
- Sample rate: 10 MSPS to 50 MSPS.
- Channel: 8
- Clock inputs: Range 10 MHz to 50 Mhz.
- Output format: Low voltage differential signaling.
- Temperature range: industrial temperature range -40°C to 50°C.
All ADCs are initialized simultaneously. FPGAs initialize, control the ADC through serial interface. FPGA send start signal acquisition, which will activate CS and Busy signals [1].

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) [3]. 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 [3]. Data can be loaded in multiples of 24-bit words within a single active CS pulse [3]. 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 [3].

4.2.1 Register Initialization:

First step after we power up the ADC, registers must be initialized. Initialization can be done in following ways;

1. Hardware reset: Applying a low going pulse on the \overline{RESET} pin.

2. Software reset: Using the serial interface, set the RST bit high. In this case \overline{RESET} pin stays high (inactive).
4.2.2 Power-Up Sequencing and Reset Timing:

Before the device is ready for serial register write, device has to undergo the reset and power sequencing. After the serial register write, then ADC is ready for data conversion.

4.2.3 The functional diagram of ADC:

The ADC employs a pipelined converter architecture that consists of a combination of multi-bit and single-bit internal stages [3]. Each stage feeds its data into the digital error
correction logic, ensuring excellent differential linearity and no missing codes at the 12-bit level [3]. And filtered output goes to serializer for serializing the parallel digital output. The serialization will be done on clock that generated by the PLL, which is 12 times faster than input clock. Therefore, the data rate is directly proportional to the input clock frequency. 1x and 6x clock also get generated along with the data outputs. This clock maintains perfect synchronization to the data outputs. The data and clock outputs of serializer are buffered externally on LVDS buffers.

![Function diagram of ADS5281](image)

**Figure 4-4: Function diagram of ADS5281**

After device has been powered up, the following registers must be written into through serial interface as part of an initialization sequence.
After the device has initialized with the default values, we can re-configure ADC to best mode of operation.

4.3 High Speed Differential Line Receivers:

The main objective of this project is to reduce the cost of the design compared to previous architectures. In the previous architectures, all the ADC’s are directly interfaced to FPGA which actually increases the number of FPGAs to be used in the design as each Lvds channel needs 2 pins on FPGA. Here, there should be a device which converts Lvds signaling to single-ended I/O standards. SN65LVDS386 converts Lvds signals to 3.3-V I/O single–ended standard signals.

The important features of the device:

- Signaling rate: up to 200mbps.
- Power supply: 3.3-V
- Open-Circuit Fail Safe.
- LVTTL levels 5-V tolerant.

This family of four-, eight-, or sixteen-, differential line receivers (with optional integrated termination) implements the electrical characteristics of low voltage differential signaling (LVDS). This signaling technique lowers the output voltage levels of 5-V differential standard levels (such as EIA/TIA-422B) to reduce the power, increase the switching speeds, and allow operation with a 3-V supply rail. Any of the eight or
sixteen differential receivers provides a valid logical output state with a ±100-mV differential input voltage within the input common-mode voltage range [4].

The input common-mode voltage range allows 1 V of ground potential difference between two LVDS nodes.

In this project, we chose an SN65LVDS386 for our requirement. It is a 16-channel differential line receiver. 8 channel data from ADC and system clock and frame clock are the input channels to the device.
The inputs to the device is a Low Voltage Differential signal (Lvds) which is converted to a 3.3V I/O single ended standard. The output of the device meets the LVTTL I/O standards.

![Figure 4-7: Output characteristics of SN65LVDS386](image)

Since we are connecting only 10 channels to the device, so there are 6 channels that remain unconnected, it should have open fail safe option for the device to work normally when no signal is connected to channels.

**Fail Safe:**

Open-circuit means that there is little or no input current to the receiver from the data line itself. This could be when the driver is in a high-impedance state or the cable is disconnected. When this occurs, the LVDS receiver pulls each line of the signal pair to near VCC through 300-kΩ resistors [4]. The fail-safe feature uses an AND gate with
input voltage thresholds at about 2.3 V to detect this condition and force the output to a high-level, regardless of the differential input voltage [4].

### 4.4 20–BIT FET BUS SWITCH:

The output from SN65LVDS386 may not be compatible to Virtex-6 FPGA, as FPGA has 1.2 to 2.5 IO standards. There must be a device which interfaces 3.3V High speed differential receiver to 2.5 V Xilinx Virtex-6.

The SN74CB3T16210 is a high-speed TTL-compatible FET bus switch with low ON-state resistance (ron), allowing for minimal propagation delay. The device fully supports mixed-mode signal operation on all data I/O ports by providing voltage translation that tracks VCC. The SN74CB3T16210 supports systems using 5-V TTL, 3.3-V LV TTL, and 2.5-V CMOS switching standards, as well as user-defined switching levels.

The important features of Device:

- Supports Mixed-Mode Signal Operation on All Data I/O Ports
  - 5-V Input Down to 3.3-V Output Level Shift with 3.3-V VCC.
  - 5-V/3.3-V Input Down to 2.5-V Output Level Shift with 2.5-V VCC.
- 5-V-Tolerant I/Os with device powered up or powered down.
- Data I/Os Support 0- to 5-V Signaling Levels (0.8 V, 1.2 V, 1.5 V, 1.8 V, 2.5 V, 3.3 V, and 5 V).

The SN74CB3T16210 is organized as two 10-bit bus switches with separate output-enable (1OE, 2OE) inputs. It can be used as two 10-bit bus switches or as one 20-bit bus switch. When OE is low, the associated 10-bit bus switch is ON, and the A port is connected to the B port, allowing bidirectional data flow between ports. When OE is
high, the associated 10-bit bus switch is OFF, and a high-impedance state exists between the A and B ports.

![Figure 4-8: Pin Configuration of SN74CB3T16210](image)

<table>
<thead>
<tr>
<th>INPUT</th>
<th>INPUT/OUTPUT</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>OE</td>
<td>A</td>
<td>A port = B port Disconnect</td>
</tr>
</tbody>
</table>

![Figure 4-9: Typical DC voltage characteristics device SN74CB3T16210](image)

Input voltages are attenuated as per the required I/O standard. Whatever may be the input voltage level the output will be almost equal to Vcc.
Chapter 5: Xilinx FPGAs

5.1 Choice of First Level of FPGAs:

After data has been converted to a compatible 2.5 I/O standard, the right FPGA must be chosen for the design. Based on the I/O requirement and block RAM memory, Xilinx Virtex-6 XC6VLX240T-FF1759 is the right FPGA for the design.

In the design, each FPGA receives data samples of 512 photo detectors, 64 system clocks, and 64 frame clocks.

The estimated I/O requirement analysis is as follows:

- 1024 detectors – 1024 data samples – 1024 pins
- System clock – 128 system clocks – 128 pins
- Frame clock – 128 frame clocks – 128 pins
- Estimated input pins – 1,280 pins
- Estimated output pins – 30 pins
- Estimated total I/O required: around 1,310 pins

Since I decided to use only two FPGAs in the first level, an FPGA with at least a 655 I/O count must be chosen. Hence, Xilinx Virtex-6 XC6VLX240T FF1759 is the perfect choice to meet the above requirements.
5.2 Choice of Second Level of FPGAs:

All the data frames have been captured and stored into block RAM, so the next step is to define the address of each data sample of the corresponding detector in its 4x4 module. Hence, another FPGA is required to receive all the data frames of photo detectors from both the FPGAs in the first level. There is no issue of I/O requirement since there are only a few output pins coming out from the first level of FPGAs. Since we use PCIe interface serial protocol to transfer the data to the PC, selection should be based on the number of GTX transceivers and also block RAM. According to requirements, the following choice is made. Hence, Virtex-6 XC6VLX75T FF484 meets all of the requirements.

Table 3: Selection of correct FPGA for the second level of FPGA, XC6VLX240T.
5.3 Virtex-6 FPGAs:

Virtex-6 FPGAs offer the best solution for addressing the needs of high performance logic designers, high-performances DSP designers, and high-performance embedded systems designers with unprecedented logic, DSP, connectivity, and soft microprocessor capabilities [5].

Three sub families:

- Virtex-6 LXT FPGA’s: High performance Logic with added serial connectivity.
- Virtex-6 SXT FPGA’s: High signal processing capability with advanced serial connectivity.
- Virtex-6 HXT FPGA’s: High Bandwidth with serial connectivity

Important Features:

- GTX transceivers: Transceivers accept the data rate up to 6.6 Gb/s, however data rates below 480 Mb/s are supported by over sampling.[8]
- Powerful mixed-mode clock managers (MMCM): MMCM blocks provide zero buffering, frequency synthesis, clock-phase shifting, input jilter filtering and phase matched clock division.
- High performance parallel SelectIO\textsuperscript{M} technology:
  - 1.2 to 2.5 V I/O operations.
  - Digitally controlled impedance (DCI) active termination.
  - Flexible fine-grained I/O banking.
  - High speed memory interface support with integrated write level capability [8].
• Flexible configurable options:
  • SPI and Parallel Flash interface.
  • Automatic bus width detection.

5.4 Serial Protocols:

5.4.1 PCIe Interface
There are many protocols like SPI, I2C, PCI, PCIE, and USB which are used for higher data transfer rate. The data transfer rate plays a key role in data acquisition system where the processes data has to be sent to a pc at a very higher data rate. This rate is also reliant on the configuration of particular protocol used like PCIe Gen1 can transfer at a rate of 2.5-GTPS and Gen2 can transfer at a rate of 5.0-GTPS, when configured for x1 lanes. Bandwidth depends on the number of lanes, higher the lanes higher the bandwidth. Our project requirement is the serial data transfer at a high speed of approximately 2.5-Giga Bytes per Second (GBPS). So which protocol has to be used can be finalized after having an understanding of different computer bus system and data transfer rates for different protocols [6].
<table>
<thead>
<tr>
<th>Technology</th>
<th>Rate (bit/s)</th>
<th>Rate (byte/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2C</td>
<td>3.4 Mbit/s</td>
<td>425 kB/s</td>
</tr>
<tr>
<td>ISA 8-Bit/4.77 MHz</td>
<td>38.16 Mbit/s</td>
<td>4.77 MB/s</td>
</tr>
<tr>
<td>EISA 8-32 bit/8.33 MHz</td>
<td>266.56 Mbit/s</td>
<td>33.32 MB/s</td>
</tr>
<tr>
<td>STD32-bit/8 MHz</td>
<td>256 Mbit/s</td>
<td>32 MB/s</td>
</tr>
<tr>
<td>PCl32-bit/33 MHz</td>
<td>1,067 Mbit/s</td>
<td>133.33 MB/s</td>
</tr>
<tr>
<td>PCI EXPRESS 1.0 (×1 link)</td>
<td>2 Gbit/s</td>
<td>250 MB/s</td>
</tr>
<tr>
<td>PCI EXPRESS 2.0 (×1 link)</td>
<td>4 Gbit/s</td>
<td>500 MB/s</td>
</tr>
<tr>
<td>PCI EXPRESS 3.0 (×1 link)</td>
<td>7.88 Gbit/s</td>
<td>984.6 MB/s</td>
</tr>
<tr>
<td>AGP8x</td>
<td>17.066 Gbit/s</td>
<td>2.133 GB/s</td>
</tr>
<tr>
<td>PCI-X DDR</td>
<td>17.066 Gbit/s</td>
<td>2.133 GB/s</td>
</tr>
<tr>
<td>Hyper Transport 3.0 (2.6 GHz, 32-pair)</td>
<td>332.8 Gbit/s</td>
<td>41.6 GB/s</td>
</tr>
<tr>
<td>Hyper Transport 3.1 (3.2 GHz, 32-pair)</td>
<td>409.6 Gbit/s</td>
<td>51.2 GB/s</td>
</tr>
</tbody>
</table>

**Figure 5-1: Comparison of technology and data transfer rate [6].**

From above table, it can be seen that PCI Express1.0(x1 link) meets the requirements.
Chapter 6: Simulation and Modeling of a Design

6.1 Simulation and Modeling of an A/D Converter

In this design, we use the A/D converter ADS5281 for analog-to-digital conversion. It is built with an 8-channel in which each channel has 12-bit resolution. The input clock is given to each channel from a clock buffer. The purpose of the simulation and modeling of this device is to produce a functional diagram along with the timing diagram similar to the diagrams in the data sheet; this has been done by using the software Xilinx ISE Design Suite 14.6.

![RTL view of an A/D converter.](image)

Figure 6-1: RTL view of an A/D converter.
The schematic diagram of the ADC shown in Figure 6-1 is exactly similar to the one in the data sheet of ADS5281 (see page 26 for a functional block diagram of ADS5281).

6.1.1 Power-Up Sequence:

The analog-to-digital conversion will not take place right away when the device is turned on. It first must undergo the “power-up sequencing and reset timing” proposed as per the data sheet; hence, this device has been modeled to detect the power-up sequence.

![Figure 6-2: Power sequencing block.](image)

When the device detects the power-up sequence, it asserts the signal “powerseq_done,” which enables the serial interface process.
In the power-up sequence (see fig 4.3 for sequence), Powerseq_done=1 for the given inputs (AVDD, LVDD, RESET, CS, LVDD)

### 6.1.2 Serial Interface

After the power-up sequence, the signal powerseq_done enables the process of initializing the internal registers.

After all registers have been initialized to their default values through a reset operation, the registers detailed in the initialization registers table shown in Figure 4.5
must be written into [3]. As a part of modeling the serial interface process, a memory has been created with a depth of 255 and 16 bits wide.

When CS is low, the following actions occur:

- The serial data (SDATA) have been latched at the serial interface clock (SCLK) to a temporary register.

- Finally, SDATA is loaded into memory at every 24th clocking edge of SCLK.

Data can be loaded to memory in multiples of 24-bit words within a single active pulse. The first eight bits will be taken as the address to the memory and the rest of the bits as data bits.
Figure 6-4: (a) Top view of Serial Interface block (b) RTL view of Serial Interface Process.

Figure 6-5: Showing how data bits and address bits have been loaded into register.
After the serial register writing, the device is now ready for data conversion, but there should be a signal to enable the device to accept the input data samples. This signal is the spi_done signal, which enables the ADC to accept the input samples.

### 6.1.3 Analog-to-Digital Conversion

After the power-up sequence and serial data writing into registers, the device will be ready for data conversion. The input sampling clock (ADclk) is given to a clock buffer tree from which it is then given to all the 8-channels.

![Figure 6-6: Figure shows the RTL view of total analog-to-digital conversion process.](image)

The input samples are sent to the ADC block for 12-bit digital conversion. The 12-bit parallel digital output from the ADC block is being serialized by the serializer block, which operates at a clock 12 times faster than the ADclk. This faster clock is generated in the clocking wizard, which is instantiated from the IP CORE. Thus, the data rate is 144mbps since the operating frequency of the ADC device is 12 MHz.
Figure 6-7: Window showing the serializer output.

Serialized bits are sent out on a single pair of pins in LVDS format. Total 8-channel output, frame clock (1x System clock ADclk), and 12 x ADclk (Lclk) are sent out in LVDS format. Since the 1x clock and 12x clock are generated in the same way as data generated from the serializer, so these clocks maintain perfect synchronization with the data.
output data, which is exactly in phase with frame clock (ADclk).

**Figure 6-8:** Window showing the output digital data and their phase relation with ADclk and Lclk.

The output interface of the ADS528x is normally a DDR interface, which means that the data is being captured on both the edges of the clock. The default phase relation between the system clock and output data is shown in the figure.

**Figure 6-9:** Phase relation between output data and ADclk and Lclk.

Output data maintains an exact synchronization with the frame clock (ADclk). Hence, the posedge clock of the frame clock represents the start bit of channel data. In addition to
programming the phase of LCLK in the double-data-rate (DDR) mode, the device can also be made to operate in the single-data-rate (SDR) mode by setting the EN_SDR bit to “1.” In this mode, the bit clock (LCLK) produces output at 12 times that of the input clock—or twice the rate, as in DDR mode. Depending on the state of FALL_SDR, LCLK may be output in either of the two manners shown in Figure 6-10. Only the LCLK rising (or falling) edge is used to capture the output data in SDR mode.

![Figure 6-10: SDR interface modes.](image)

We can model the A/D converter to work in SDR format by adjusting the phase relation of the data, system clock, and frame clock, as shown in Figure 6-10 in a clocking wizard.
6.2 Modeling and Simulation of High-Speed Differential Line Receiver

As per the design requirements, high-speed differential receivers should be capable of handling high signaling rates up to 144mbps. SN65LVDS386 handles signaling rates up to 200mbps. In previous architectures, all of the ADCs directly interfaced to FPGA, which increased the number of FPGAs used in the design, as each LVDS channel uses two pins on FPGA. This device converts LVDS signals to 3.3V I/O standard single-ended signals, as it needs only one pin per channel to interface with the FPGA instead of the standard two pins.

Since the SN65LVDS386 device is a simple buffer, modeling of this device is a simple task. The Xilinx software library includes an extensive list of primitives to support a variety of I/O standards available in the Virtex-6 FPGA I/O primitives [7].

These seven generic primitive names represent most of the available differential I/O standards:

- IBUFDS (input buffer)
- IBUFGDS (clock input buffer)
- OBUFDS (output buffer)
- OBUFTDS (3-state output buffer)
- IOBUFDS (input/output buffer)
- IBUFDS_DIFF_OUT (input buffer)
- IOBUFDS_DIFF_OUT (input/output buffer)

Out of all the above seven generic primitives, IBUFDS meets the design requirements. It is a differential input buffer with two pins at the input side that act as the P and N channel in the differential pair, and the output obtained is a single-ended signal.
Merely instantiating the template is not enough to synthesize the block. The library ‘UNISIM’ should be included at the top of the code.

library UNISIM;

use UNISIM.VComponents.all;

Dout_p and Dout_n are the differential inputs to the device, and Dout is the single-ended output signal.
Figure 6-13: Single-ended signal output.

6.3 Modeling and Simulation of First Level of FPGAs

The present architecture has two levels of FPGAs, whereas the previous architectures have three levels of FPGAs. In the present architecture, the first level of FPGAs consists of two Xilinx Virtex-6 FPGAs with part number XC6VLX240T. These FPGAs receive samples from 1,024 detectors through the ADC, high-speed differential receiver, and 20-bit FET switch SN74CB3T16210 devices. The output from SN65LVDS386 may not be compatible with Virtex-6 FPGA, as FPGA has 1.2 to 2.5 IO standards. SN74CB3T16210 interfaces a 3.3-V high-speed differential receiver to 2.5-V Xilinx Virtex-6.
Figure 6-14: Block diagram of data samples writing into block RAM in level1.

For simplicity, the figure shown in 6-14 and 6-15 depicts samples from only one ADC, making very clear that we need 64 similar constructions as shown in Figure 6-14 to make level-1 FPGA a complete block diagram. The 8-channel output from ADS5281 with 12-bit resolution is sent to a high-speed differential receiver and the FET switch and then to the first level of FPGA.
6.3.1 Serial In Parallel Out

In Figure 6-14, each channel output from ADC is sent to one serial and parallel out (SIPO), so each ADC is connected to eight SIPOs. The serialized data from ADCs are parallelized in SIPO. Here there is a Lclk (bit clock) and a frame clock corresponding to each ADC. Serialized data are latched into FPGA at a faster clock (Lclk), so data are parallelized in SIPO at a faster clock, and data from SIPO will be read at the frame clock (ADclk) since the posedge of the frame clock represents the start of a sample.
Parallel data latched out at ADclk posedge

**Figure 6-16: Window shows how parallel data will read out at ADclk.**

An enable signal is routed from ADC to the FPGA such that the FPGA can know the exact start of input samples. Here, the enable signal will be in a high state, “1,” as long as there are input samples to the FPGA.

Counters begin to count as soon as they receive the enable signal, but we intend them to count when the data are ready at a particular point. In Figure 6-14, for example, the count of Counter1 is given as input select lines to the multiplexer. Counter1 begins as soon as it receives the clock and enable signals, but data are not ready at the input of the multiplexer since the serial in parallel out takes 12 clock cycles of delay to give the output parallel data. Hence, Counter1 counts unnecessarily for the first 12 clock cycles, and it is obvious that Counter1 cannot count for the last 12 clock cycles, resulting in data loss, since the enable signal comes to the low state “0.” The enable signal stays in a high state (“1”) until there are input samples to the FPGA. The enable has to be delayed till the samples are ready at the multiplexer. Thus, the enable is delayed at the delay block for one cycle of ADclk (ADclk period) and latched out at the posedge of the ADclk. In brief, the enable signal (enaout) for Counter1 will be read from the delay block along with parallel data from SIPO at the same time.
Counter1 begins to count as soon as it receives the enable signal from the delay block, and this count serves as select lines for the multiplexer, which selects one of the outputs of SIPOs.

6.3.2 Multiplexer in Level1

Figure 6-17: Multiplexer in level1 selects one of the outputs of serial in and parallel out.

As shown in the above figure, based on the Counter1 count, one of the outputs of SIPOs is multiplexed and stored in the block RAM (12 x 144), and enaout is also used as the enable signal for block ram. The Counter1 is programmed in such a way that its count (which serves as select lines for the multiplexer) rises to “111” and then comes back to “000” and waits till the next sample is ready at the first input, since the next sample is
ready at inputs for only every one ADclk cycle delay (or 12x Lclk cycles). Thus, the increment from “000” to “001” happens only after four Lclk cycles of delay.

6.3.3 Writing Data to Block RAM from Multiplexer

Figure 6-18: Block diagram of writing process to block RAM in Level1

Each block RAM stores all the data samples of eight detectors, as each ADC can give only samples of eight detectors. Each block RAM consists of data coming from the respective ADC. Counter2 is used for writing an address to the block RAM. It functions in the same manner as counter1; it counts till “1000” and waits for four Lclk cycles, and upon completion of counting, it starts incrementing. This procedure is repeated for every eight clock cycles. It takes 216 Lclk cycles to load the data from each ADC to the
individual block RAM. Here, a total of 144 samples should be loaded in block RAM; therefore, the write depth of block RAM will be 144.

6.3.4 Placing a Detector in Level-1 Block RAM

![Diagram of 4x4 module with detectors and frames]

The above notations refer to the figure in the next page, which depicts the data storage in the block RAM. The address of each detector can easily be defined in the block RAM.

**Figure 6-19: Placing of each detector in its module**

The above notations refer to the figure in the next page, which depicts the data storage in the block RAM. The address of each detector can easily be defined in the block RAM.
Table 4: Placing of each detector sample from ADC in Level1 FPGA.

The place of the detector and corresponding frame can easily be known by the address bits. The three last bits define the detector number, and the other five bits define the frame number.

For example: \(10001\ 110\)

Frame Address (F17)  Detector address (D6)

So, the above address indicates the particular location of detector 6 with frame 17.
Figure 6-20: Block diagram of Level-1 FPGA.

Figure 6-20 gives the top-level architecture in level-1 FPGA, but there are still many components modeled for reading and writing the data that can be seen in further document.

A total of 512 photo detectors samples are stored on a single FPGA. Here, each ADC has a corresponding Lclk and ADclk, so data coming out from each ADC can be
stored on one block ram with a corresponding Lclk as an input clock. Since 64 ADCs are interfaced on a single FPGA, so 64 block RAMs (12x144) would be needed to save all the frames of 512 photo detectors. We have two FPGAs in the first level, but Figure 6-20 is a complete block diagram for one FPGA only.

6.3.5 Reading the First-Level Data to Transfer into Second Level of FPGAs

This is the most important and interesting part of memory mapping. Storing the data in a block ram is an easier task in level1, but reading the stored data from the block RAM is gruesome. The enaout signal, which is the output of the delay block (one ADclk clock cycle), will be sent to another delay block, where the signal is delayed for four Lclk cycles. This delayed enable (enab) is given as the enable to block ram and Counter3. The enable signal is made to stay in a high state, “1,” till it receives the reset signal. Hence, block ram is always enabled. Reading and writing of data will happen simultaneously, but the reading operation takes place only after four Lclk cycles of delay in the writing process.

Each block ram has its dedicated counter to give it a reading address. There are 64 block RAMs in level1; hence, 64 counters are needed for a reading operation in the design. The important part is the way that the stored data are read from each block RAM.

Each detector is sampled 18 times to form an image, so all the data frames from the eight detectors are stored collectively in the block RAM. The data are read from each block ram in a frame-wise manner. First, the frame of all 512 detectors is read out, followed by the second and then the third frame, consecutively.
The delayed enable signal (enab) for four Lclk cycles will enable counter3 and block RAM simultaneously as shown in Figure 6-21. A d-multiplexer is used in the design to enable the reading counters at an exact time and disable the counters when not required. D-mux outputs are connected to all 64 dedicated counters. Based on the counter3 count, one of the outputs of the d-multiplexer comes to a high state of “1” and enables its respective counter to read the data samples of associated block RAM. Meanwhile, the remaining counters will be in a low state, “0.”
Figure 6-22: Multiplexer to select the one of the block RAM outputs.

Counter3 increments for every eight clock cycles; therefore, one of the outputs of the d-multiplexer will be in a high state (“1”) for eight Lclk cycles while all other outputs remain in a low state (“0”). After the first Lclk eight clock cycles of delay, the counter3 count is incremented, and the next output of d-mux will come to a high state (“1”), whereas the previous output comes to a “0” state, and the remaining outputs will maintain their previous states.

This process continues till the counter3 output count for “1000000” (64 in decimal) returns to the initial state “00000001” (1 in decimal). As a result, in all the 512 detectors, the first frame is read followed by the second frame.

The final stage of level1 is transferring the data from each block RAM to the next level of FPGA. A 64 x 1 multiplexer shown in Figure 6-22 selects one of the block RAMs for every eight clock Lclk cycles of delay. The counter for the multiplexer (counter_for_mux) will increment for every eight Lclk clock cycles. In this way, one of
the 64 block RAMs will be selected each time for every eight Lclk clock cycles of delay. The enab signal used previously for the reading process is delayed by one Lclk clock cycle, and this delayed signal is given to counter_for_mux to enable the count. This count serves as select lines of the multiplexer. Based on counter_for_mux count, one of the block RAMs is selected for eight clock cycles of time. This process continues till the counter count reaches “111111” (63 in decimal) and returns to “0000” and selects the block RAM 1.

The component analysis for level-1 FPGA is as follows:

- Total needed SIPOs – 512 SIPOs (in single FPGA)
- Total block RAMs (12x144) needed – 64
- Total Counter 7-bit (for D-multiplexer) – 1
- Total Counter 8-bit (for writing and reading addresses to Block ram) – 128
- Total Counter 3-bit (for multiplexer to select one of the SIPOs) – 64
- Total Counter 6-bit (for multiplexer to select one of the block RAMs) – 1
6.4 Modeling and Simulation of the Second Level of FPGAs

Two FPGAs are interfaced to a single FPGA in the second level. All 18 data frames of photo detectors will be collected and stored in only one block RAM in which placement of each detector sample can be easily known from the address defined in the block RAM. The system clock (Lclk), data frames, and enable flag will come from each FPGA in level1. This enable flag enables writing operation in level1. Since there is no synchronization among data in two FPGAs, two block RAMs are required to store all 1,024 detectors’ data frames, 512 detector frames on each block RAM. As mentioned earlier, writing the data is always an easier task rather than reading the data, which is a tedious process.
6.4.1 Writing the Data to Level-2 Block RAM from Level-1 FPGAs

Figure 6-24: Block diagram shows the writing process to initial block RAMs

Initially, two block RAMs (also called as initial block RAMs) receive data samples from two FPGAs. As shown in the figure, each block RAM receives data samples from a corresponding FPGA. So, 512 detectors frames are stored in one block RAM and 512 in another. The enable flag coming from each FPGA will enable the counters to count, which acts as a writing address for each block RAM.
6.4.2 Reading the Stored Data and Transferring to Final Block RAM

Reading the data from the initial block RAMs in level2 is exactly similar to the process of reading the data shown in the level-1 FPGA (see Figure 6-21). The first frames of the 1,024 detectors are read out sequentially. In this design, the counter_for_dmux count serves as select lines, and it is programmed in such a way that the state of d-mux output changes for every 512 clock cycles. Each block RAM has a dedicated counter to count for each read address, and these counters are enabled for every 512 Lclk cycles. One of the counters will remain enabled for 512 clock cycles while the other counter remains inactive and vice-versa. The first 512 detectors’ first frame is read from one of
the block RAMs followed by the other 512 detectors’ first frame read out from the other block RAM. Next, the second frame of all 1,024 detectors is read out, one after the other, and this process continues.

6.4.3 Writing the Data into Final Block RAM

![Figure 6-26: Writing data to final block RAM.](image)

This is the final step in level2, writing all the data to final block RAM. Here we have 1,024 * 18 =18,432 frames to be stored in a single block RAM, so the depth of the block RAM is 18,432 with width of 12-bit parallel data. Block RAM finds its write address from the counter_for_write_addr counter. A 2:1 multiplexer is used to select one
finally, data from the block RAM are read out at a speed of 2 Gb/sec, and this must be 12-bit output in serial to transfer all serial data to the fiber channel switch. Output of the block RAM goes to the one parallel-in-serial-out shift register to convert 12-bit parallel data to 12-bit serial data.

The 256x256 array photo detectors can be constructed by placing 32x32 array detectors eight times in a row. Hence, a 256 x 256 array photo detector needs eight Virtex-6 FPGAs in level2. Each level-2 FPGA uses the PCIe transfer protocol to produce optical serial output of 5 Gb/sec. After processing the data, the fiber optical switch takes over to generate the 2-Gb/sec data stream.

Next, to place a detector in level 2, at the second level a block RAM (12 x 18,432) is generated to store all data frames. This block RAM is generated with a 15-bit address for writing and reading the data.
Figure 6-27 depicts the arrangement of modules which forms an array of 8 x 8. Each module address is defined in a binary form in the final block RAM as per the number shown in the above figure.

The location of the detector in level2 can be easily defined by the address in the memory.

\[
\begin{array}{ccc}
\begin{array}{c}
XXXXX \\
\end{array} & \begin{array}{c}
XXXXXXXX \\
\end{array} & \begin{array}{c}
XXXX \\
\end{array} \\
\text{Frame number} & \text{Module number} & \text{Detector number in a module}
\end{array}
\]

As we know, a 32x32 array photo detector is constructed using 64 modules, and each module has a 4x4 sub array of photo detectors. The first four LS bits specify detector number in a module; the next seven bits specify module number, and the last five bits specify the frame number.

6.4.4 Resetting the Entire Design

Resetting the entire system is a very important process, as counters in the design must come to the initial state before the next sample process begins. The entire design must be reset before 1ms after the first data sample. The approximate time taken for data frames of one data sample to be processed is 0.381ms; thus, the entire design has to be reset any time after 0.381ms or before 1ms after a first data sample begins.

The VHDL code to reset the entire system is as follows:

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
entity reset_entire_design is
port (Lclk:in std_logic; --system input clock
ena:in std_logic; -- enable signal
```


reset:inout std_logic -- reset system);
end reset_entire_design;

architecture Behavioral of reset_entire_design is

signal enaout: std_logic;

signal count : integer range 0 to 60000 :=0; --setting signal count range to 60000

begin

process(Lclk) --process statement begin

begin

if( Lclk'event and Lclk = '1') then -- trigger at clock edge

if(reset = '1') then     -- reset system

enaout <='0';

count <= 0;

reset <='0';

else

if(ena = '1') then

enaout <= ena;
end if;

if(enaout = '1') then

if(count = 60000) then   -- checking the count range

reset <= '1';

else

count <= count + 1;
end if;

end if;

end if;

end if;

end process;

end Behavioral;
This is a RTL model of the design:

**Figure 6-28: RTL view of the reset_entire_design.**

The system is designed in such a way that reset is in a high state ("1") for only one clock cycle period of time and comes back to the initial state ("0"). When the reset is set to a high state ("1") for one clock cycle period, this period is sufficient to reset the entire design.

**Figure 6-29: Simulation window showing the reset going high at 60,000 clock cycles.**
Chapter 7: Conclusion

In this project, data acquisition system collects the data from a large number of photodetectors at a very high frequency with high resolution. The received data is digitalized and processed at two levels of FPGAs and finally the data is sent to a PC through PCI interface.

The 32 x 32 photo detector array has 1024 detectors. There are total 18 frames that come from each photo detector which forms the image. The analog samples are being converted to digital values in A/D converter.

The better possible architecture for 32 x 32 photo detector array image data acquisition has been given compared to previous architectures. This architecture has two levels of FPGAs where the first level of FPGAs has only two FPGAs and the second level of FPGA has one FPGA. The entire design was simulated and modeled in Xilinx 14.6 ISE design suite.

In order to meet the specifications the system should collect the data samples and transmit the 18 samples of each photo detector in 1ms interval of time. Total 221184 bits have to be processed in less than a one milli second. Below Calculations show that time required to transmit the data samples is very much less than 1ms.

In Level-1:

Level-1 time = 12 Lclk cycles delay due to SIPO + Block ram restoring the data – (restoring the data + the delay the reading the data begin).

Level-1 time = 0.08333 μs + (18 * 12 *6.94 ns) – ((18 * 512 * 6.94) + (4 *6.94)) = 62.48μs
**In Level-2:**

Level-2 time = (Initial block ram restoring the data) – ((data storing the data + the delay the reading the data begin)) + (Block ram to restore the data in final block ram) – (data storing the data +the delay reading has begin) + 12 clock cycle delay in parallel in serial out.

Level-2 time = (127.91µs - (63.95µs + 27.76 ns) + (127.91 µs + (127.91).

Hence total time required= 62.48µs + 64 µs = 0.253 ms.

Total number of bits = 1024 * 18 * 12 = 221184 bits are read out from final Block ram at 2.5 Gb/sec. The time taken to transmit all these bits is 12.875 µs.

The total required for processing the total bits of 32 x 32 array detectors and to send to fiber switch is 0.381 ms which is very less than 1 ms.
REFERENCES

1 Data acquisition system, retrieved January 2012 from URL: 

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.pelagiareresearchlibrary.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 mspis 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:


Appendix

VHDL codes for modeling and simulation of Analog to Digital converter:

VHDL code for ADC Block:

```vhdl
use IEEE.STD_LOGIC_1164.ALL;
library ieee_proposed;  -- new library has been created to include floating values
use IEEE_proposed.fixed_pkg.all; -- new special packages has been included
library ieee; use ieee.numeric_std.all; -- library
entity ADC_8_bit is
port (clk:in std_logic ; -- input clock
reset :in std_logic;-- reset clock
analog_in : in ufixed (7 downto -4);-- analog input (real values in binary form)
data_en:in std_logic;-- data enable
power_seq_done:in std_logic;-- enable signal to initialize the analog to digital conversion process
spi_en:in std_logic;
spi_done:in std_logic; -- if spi_en = '1' then it says the end of serial initializing process.
data_format:in std_logic_vector(15 downto 0);-- output format
digital_out :out std_logic_vector (11 downto 0);-- digital output
data_en_out : out std_logic); -- output along with enable signal
end entity;
architecture original of ADC_8_bit is
constant max_abs_digital_value : integer := 2048; -- maximum digital value
begin
process(clk) -- sensitivity clock
variable digitized_signal: integer  range 0 to 4096 :=0; -- maximum analog value 2^15 = 4096
begin
if(clk'event and clk = '1') then -- for the event at clock edge
if(reset = '1') then
digital_out <= "000000000000";
elsif(data_en = '1') then
if(power_seq_done='1') then
if(spi_en='0') then
if(spi_done = '1') then
if(data_format = "0101010101010101")then
digitized_signal := to_integer(analog_in *11001100.1100);-- anlog to digital conversion
if(digitized_signal > (max_abs_digital_value-1)) then -- if digital output greater than maximum value
```
digitized_signal := max_abs_digital_value-1; -- then output equals to maximum value
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
digital_out <= std_logic_vector(to_unsigned(digitized_signal,digital_out'length));--digital output in un_signed
end if;
da ta_en_out <=data_en;
end if;
end process;
end original;

**VHDL code for serializer Block:**

library UNISIM;
use UNISIM.VComponents.all;

entity serializer is
port ( clk       : in std_logic;
reset:in std_logic;
DataReady    : in std_logic;
Parallel_data: in std_logic_vector(11 downto 0);
Serial_out: out std_logic_vector(0 downto 0)
);
end serializer;

architecture Behavioral of serializer is
Signal Shreg: std_logic_vector( 11 downto 0);
signal sig :std_logic:="0";
begin
process( clk,reset)
variable cnt: integer range 0 to 15:=0;
begin
if(reset = '1') then
    shreg <="000000000000";
    serial_out<="0";
    cnt:=0;
elsif rising_edge(clk) then
    if DataReady='1' then
        if(sig = '0') then
            Shreg <="000000000000";
        elsif sig='1' then
            Shreg <="011111111111";
        end if;
        Serial_out <="011111111111";
        cnt:=15;
    else
        cnt:=cnt+1;
        Shreg<="111111111111";
        Serial_out <="111111111111";
        if cnt=15 then
            cnt:=0;
        end if;
    end if;
end if;
end process;
end original;
Shreg <= Parallel_data;
sig<='1';
else
Shreg <= Shreg(0)& Shreg(11 downto 1);
cnt := cnt +1;
end if;
if(cnt >11) then
sig<='0';
cnt :=0;
end if;
end if;
Serial_out(0) <= Shreg(0);
end if;
end process;
end Behavioral;

**VHDL Code for output buffer in Analog to Digital Converter:**

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library UNISIM;
use UNISIM.VComponents.all;
entity output_buffer is
port ( serial_out:in std_logic_vector(0 downto 0); -- serial in
dout_p:out std_logic; -- differential output
dout_n:out std_logic); -- differential output
end output_buffer;
architecture Behavioral of output_buffer is
signal serial : std_logic_vector(0 downto 0);
begin
serial(0)<= transport serial_out(0) after 7.1 ns; -- propagation delay
ad: OBUFDS -- buffer element
port map (O=>dout_p,
OB=>dout_n,
I=>serial(0)
);
end Behavioral;
**Code for high speed differential receiver:**

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library UNISIM;
use UNISIM.VComponents.all;
entity SN65LVDS386 is
  port ( Doutp:in std_logic; -- differential input
          Doutn:in std_logic; -- differential input
          Dout:out std_logic); -- single ended output
end SN65LVDS386;
architecture Behavioral of SN65LVDS386 is
    ad: IBUFDS -- primitive buffer
    port map (O=>'0', IB=>Doutn, I=>Doutp);
end Behavioral;
```

**Simulation of first Level and second level of FPGAs:**

**Code for serial in parallel out:**

```vhdl
entity ssspo is
  port(Lclk:in std_logic; -- input system clock
       ena: in std_logic; -- input enable
       reset:in std_logic; -- input reset
       SI:in std_logic; --serial input
       ADclk: in std_logic; -- frame clock
       temp: out std_logic_vector(11 downto 0);-- parallel out
       enaout:out std_logic);-- enable out along with data
end ssspo;
architecture Behavioral of ssspo is
    signal temp1: std_logic_vector (11 downto 0):=(others=>'0'); -- temporary signal to store input serial bits
    signal main_count :std_logic:=0';
    signal tempen:std_logic;
begin
```

76
process (Lclk, ADclk) -- sensitivity list

variable i:integer:=0; -- variable integer to count

begin

if(Lclk'event and Lclk = '1') then -- for rising edge clock
    if(reset = '1') then
        temp1<= "000000000000";
        i:=0;
    elsif(ena = '1') then
        if(i<12) then
            temp1(i) <= SI; -- temporary register to store serialized bits
            i:=i+1;
        end if;
        if(i=12) then -- if count = 12 then it should stop incrementating
            i:=0;
        end if;
    end if;
end if;

-- from here starts if i = 12 the ADclk latches the parallel out

if(ADclk'event and ADclk = '1') then -- for rising edge of a clock
    if(reset = '1') then
        temp<= "000000000000";
        main_count <= '0';
    elsif(ena = '1') then
        if(main_count = '1') then
            temp<= temp1;
        else
            main_count <= '1';
        end if;
    end if;
end if;

if(ADclk'event and ADclk = '1') then
    tempen<= ena;
    enaout <= tempen;
end if;

end process;

end behavioral;
Codefor 8x1 Multiplexer:

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mux_sipo is
port ( p01: in std_logic_vector(11 downto 0); -- output from sipo1
       p02: in std_logic_vector(11 downto 0); -- output from sipo2
       p03: in std_logic_vector(11 downto 0); -- output from sipo3
       p04: in std_logic_vector(11 downto 0); -- output from sipo4
       p05: in std_logic_vector(11 downto 0); -- output from sipo5
       p06: in std_logic_vector(11 downto 0); -- output from sipo6
       p07: in std_logic_vector(11 downto 0); -- output from sipo7
       p08: in std_logic_vector(11 downto 0); -- output from sipo8
       sel: in std_logic_vector(2 downto 0); -- select pins
       po_mux : out std_logic_vector(11 downto 0)); -- output
end mux_sipo;

architecture Behavioral of mux_sipo is
begin
process(p01, p02, p03, p04, p05, p06, p07, p08, sel)
variable temp : std_logic_vector(11 downto 0);
begin
  -- multiplexer outputs
  case sel is
    when "000" => temp := p01;
    when "001" => temp := p02;
    when "010" => temp := p03;
    when "011" => temp := p04;
    when "100" => temp := p05;
    when "101" => temp := p06;
    when "110" => temp := p07;
    when others => temp := p08;
  end case;
p0_mux <= temp;
end process;
end Behavioral;
VHDL code for Write Counter of Bram:

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
entity counter is
port(Lclk:in std_logic;-- system clock
     reset:in std_logic; -- input reset system
     enaout: in std_logic; -- input enable
     count1:out std_logic_vector(2 downto 0));-- output count
end counter;
architecture Behavioral of counter is
signal count:std_logic_vector (2 downto 0):="000";
signal temp:std_logic_vector (2 downto 0):="000";
signal temp2: std_logic_vector (2 downto 0):="000";
signal temp3:std_logic_vector (2 downto 0 ):="000";
signal temp4:std_logic_vector ( 2 downto 0 ):="000";
signal tem:std_logic:='1'; --temperoray register
signal tem2:std_logic:='1';
signal tem3:std_logic:='1';
signal tem4:std_logic:='1';
signal maincount :std_logic:='0';
begin
process(Lclk)
begin
if(Lclk'event and Lclk = '1') then
if(reset = '1') then
    count<="000";
    maincount<='0';
elsif(enaout = '1') then
if(maincount = '1') then
    --for every eight clock cycles signal get delayed for four clock cycles
    if(count = "111" or count ="000")then-- it checks the condition and signal get delayed for 4 clock cycles
        temp <= "000";--temporary registers
        tem<='0';
        temp2<= temp;
        tem2<= tem;
```
temp3 <= temp2;
tem3 <= tem2;
temp4 <= temp3;
tem4 <= tem3;
count <= temp4;
maincount <= tem4;
else
count <= count + 1;
temp <= "000";
temp2 <= "000";
temp3 <= "000";
tem <= '1';
tem2 <= '1';
tem3 <= '1';
tem4 <= '1';
temp4 <= "000";
maincount <= '1';
end if;
else
count <= count + 1;
temp <= "000";
temp2 <= "000";
temp3 <= "000";
temp4 <= "000";
maincount <= '1';
end if;
end if;
end if;
end if;
end process;
count1 <= count;
end Behavioral;
VHDL code for write counter of Block Ram:

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
entity counter2 is
port(Lclk:in std_logic;-- system clock
reset:in std_logic; -- input reset
enaout: in std_logic; -- enable
count1:out std_logic_vector(7 downto 0)); -- output clock
end counter2;
architecture Behavioral of counter2 is
signal count:std_logic_vector (7 downto 0):="00000000";
signal temp:std_logic_vector (7 downto 0):="00000000";
signal temp2: std_logic_vector (7 downto 0):="00000000";
signal temp3:std_logic_vector (7 downto 0 ):="00000000";
signal temp4:std_logic_vector ( 7 downto 0 ):="00000000";
signal tem:std_logic:='1';
signal tem2:std_logic:='1';
signal tem3:std_logic:='1';
signal tem4:std_logic:='1';
signal maincount :std_logic:='0';
beginn
process(Lclk)
beginn
if(Lclk'event and Lclk = '1') then -- at positive edge of clock
if(reset = '1') then
    count<="00000000";
    maincount<='0';
elsif(enaout = '1') then
    if(maincount = '1') then
        if(count = "10001111") then --if count is 144 then count will be
            reset
        else
            -- counter gets delayed for 4 lclk cycles for every 8 clock count
            if(count = "00000111" or count = "00001111" or count = "000010111" or count = "00011111"or count ="00100111" or count = "00101111" or count = "00110111" or count = "00111111" or count = "01000111" or count =...}

81
"01001111" or count = "01010111" or count = "01011111" or count = "01100111" or count = "01101111" or count = "01110111" or count = "01111111" or count = "10000111" or count = "00000000") then

```
  temp <= count + 1;
  tem<='0';
  temp2<= temp;
  tem2<='0';
  temp3<='0';
  tem3<='0';
  temp4<='0';
  tem4<='0';
```

else

```
count<= count + 1;
  temp<="00000000";
  temp2<="00000000";
  temp3<="00000000";
  temp4<="00000000";
  tem<='1';
  tem2<='1';
  tem3<='1';
  tem4<='1';
```
end if;
else
```
count<= count + 1;
  temp<="00000000";
  temp2<="00000000";
  temp3<="00000000";
  temp4<="00000000";
  maincount<='1';
```
end if;
end if;
end Behavioral;
VHDL code for D-Multiplexer:

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity demux1_4 is
port (out0 : out std_logic;  --output bit
       out1 : out std_logic;  --output bit
       out2 : out std_logic;  --output bit
       out3 : out std_logic;  --output bit
       out4 : out std_logic;
       out5 : out std_logic;
       out6 : out std_logic;
       out7 : out std_logic;
       sel : in std_logic_vector(3 downto 0)) ;
end demux1_4;
architecture Behavioral of demux1_4 is
begin
process(sel)
begin
case sel is
   when "0001"  => out0 <= '1'; out1 <= '0'; out2 <= '0'; out3 <= '0'; out4 <= '0'; out5 <= '0'; out6 <= '0'; out7 <= '0';
   when "0010"  => out1 <= '1'; out0 <= '0'; out2 <= '0'; out3 <= '0'; out4 <= '0'; out5 <= '0'; out6 <= '0'; out7 <= '0';
   when "0011"  => out2 <= '1'; out1 <= '0'; out0 <= '0'; out3 <= '0'; out4 <= '0'; out5 <= '0'; out6 <= '0'; out7 <= '0';
   when "0100"  => out3 <= '1'; out1 <= '0'; out2 <= '0'; out0 <= '0'; out4 <= '0'; out5 <= '0'; out6 <= '0'; out7 <= '0';
   when "0101"  => out4 <= '1'; out1 <= '0'; out2 <= '0'; out3 <= '0'; out0 <= '0'; out5 <= '0'; out6 <= '0'; out7 <= '0';
   when "0110"  => out5 <= '1'; out1 <= '0'; out2 <= '0'; out3 <= '0'; out4 <= '0'; out0 <= '0'; out6 <= '0'; out7 <= '0';
   when "0111"  => out6 <= '1'; out1 <= '0'; out2 <= '0'; out3 <= '0'; out4 <= '0'; out5 <= '0'; out0 <= '0'; out7 <= '0';
   when "1000"  => out7 <= '1'; out1 <= '0'; out2 <= '0'; out3 <= '0'; out4 <= '0'; out5 <= '0'; out6 <= '0'; out0 <= '0';
   when others => out7 <= '0'; out1 <= '0'; out2 <= '0'; out3 <= '0'; out4 <= '0'; out5 <= '0'; out6 <= '0'; out0 <= '0';
end case;
end process;
end Behavioral;
VHDL code for read counter of Block Ram:

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_unsigned.all;

entity counter is
port(Lclk:in std_logic; -- system clock
 reset:in std_logic; -- input reset system
 enaout: in std_logic; -- input enable
 count1:out std_logic_vector(2 downto 0));-- output count
end counter;

architecture Behavioral of counter is

signal count:std_logic_vector (2 downto 0):="000";
signal temp:std_logic_vector (2 downto 0):="000";
signal temp2: std_logic_vector (2 downto 0):="000";
signal temp3:std_logic_vector (2 downto 0 ):="000";
signal temp4:std_logic_vector ( 2 downto 0 ):="000";
signal tem:std_logic:='1'; --temperoray register
signal tem2:std_logic:='1';
signal tem3:std_logic:='1';
signal tem4:std_logic:='1';
signal maincount :std_logic:='0';
begin
process(Lclk)
begin
if(Lclk'event and Lclk = '1') then
 if(reset = '1') then
 count<="000";
 maincount<='0';
 elsif(enaout = '1') then
 if(maincount = '1') then
 --for every eight clock cycles signal get delayed for four clock cycles
 if(count = "111" or count ="000")then-- it checks the condition and
 signal get delayed for 4 clock cycles
 registers
 temp <= "000";--temporary
 temp2<= temp;
end counter;
tem2 <= tem;
temp3 <= temp2;
tem3 <= tem2;
temp4 <= temp3;
em4 <= tem3;
count <= temp4;
maincount <= tem4;
else
count <= count + 1;
temp <= "000";
temp2 <= "000";
temp3 <= "000";
tem <= '1';
em2 <= '1';
tem3 <= '1';
etm4 <= '1';
temp4 <= "000";
maincount <= '1';
end if;
else count <= count + 1;
temp <= "000";
temp2 <= "000";
temp3 <= "000";
tem <= '1';
em2 <= '1';
tem3 <= '1';
etm4 <= '1';
temp4 <= "000";
maincount <= '1';
end if;
end if;
end if;
end process;
count1 <= count;
end Behavioral;