Spring 2015

ESD event locator

Syed Abrar Ul Huq

Department:

ESD EVENT LOCATOR

by

SYED ABRAR UL HUQ

A THESIS

Presented to the Faculty of the Graduate School of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE IN COMPUTER ENGINEERING

2015

Approved by

Dr. David J. Pommerenke, Advisor
Dr. Yiyu Shi
Dr. Daryl G. Beetner

ABSTRACT

Electrostatic Discharge (ESD) presents one of the most common threats to electronic systems especially in the cases of high-speed digital systems. ESD event locator systems have been designed to determine the location of the source of the ESD event using different techniques. One such method of determining the origin of the ESD event, is the use of four antennas and a high-speed Analog to Digital Converter (ADC) system to capture the ESD events.

This thesis presents such an implementation which deals with the design of an analog front-end to a fast TI LM97600 ADC and a Virtex-5 Field-Programmable Gated Array (FPGA) based system to control the ADC and to capture and transfer the sampled data between the FPGA and a Personal Computer (PC). The thesis focuses on specific objectives of the ESD event locator system which include; first, to program the FPGA to capture and acquire data on the high-speed output lanes of the ADC, second to develop a trigger logic to store data only on the occurrence of an ESD event, third to develop robust code to transfer the data from the FPGA to the PC using communication protocols such as Universal Serial Bus (USB).

ACKNOWLEDGMENTS

I am extremely thankful to my advisor Dr. David Pommerenke for his constant support and the knowledge imparted by him, which has both helped me in my thesis as well as my own personal development as a student. I have learned innumerous things during my work in the EMC lab as a result of his advice. I would also to like to thank him for supporting my work with graduate research assistantship during my pursuit of my Master's degree.

I would also like to specially thank Dr. Yiyu Shi for recommending me for the research, for his constant guidance and advice on academic matters and also for serving on my research committee. I would also like to thank Dr. Daryl Beetner for his advice and for accepting my request to serve on my research committee.

I would like to thank John Lee from Samsung Electronics Co. Ltd. for his support and guidance in performing the research. I would also like to thank the other faculty members in the EMC lab for their guidance and teaching and for providing an excellent research environment. I would also like to thank my colleagues in the EMC lab for their teamwork and assistance on all aspects of my research work.

Lastly, I'd like to thank my family members for their constant support and encouragement.

TABLE OF CONTENTS

Page

LIST OF ILLUSTRATIONS

Figure Page

LIST OF TABLES

1. INTRODUCTION

Electrostatic discharge (ESD) is one of the most common events which results in damage and malfunctioning of devices such as Integrated Circuits (IC) due to its current and field. Static electricity can be induced by the human body by simply walking across a carpet, the effect also called triboelectric charging, which in turn leads to an ESD event when a metal is touched by the human body. An extreme example of an ESD event would be a lightning bolt. While the discharge of static electricity from the human body may not be a visible or a violent event, they may easily result in damage to ICs. The system level block diagram is shown in Figure 1.1.

In order to reduce the negative impact of ESD events, it is important and necessary to determine the origin and location of the ESD event. This would minimize the risk by allowing us to isolate or remove the device or entity responsible for the ESD event. This is especially important in environments such as electronics manufacturing environments which are highly susceptible to ESD and can cause potential problems.

Initially, identification of ESD events was performed as well as the strength of the ESD events was measured in an actual IC manufacturing environment, but determining the actual location of the ESD event was not performed. An ESD Event Locator System (EELS) was also built which involves the use of four EMI detectors as well as the use of a high-speed real-time digital oscilloscope. This is based on the principle of the Global Positioning System (GPS) on a smaller scale which determines the location of the ESD event to a resolution of 1 cm in each direction. The main issues in the system are the ease of installation and the overall cost of the system as it is based on an expensive real-time oscilloscope. Additional methods involve the use of four EMI detectors involving the principle of trilateration-based estimation as

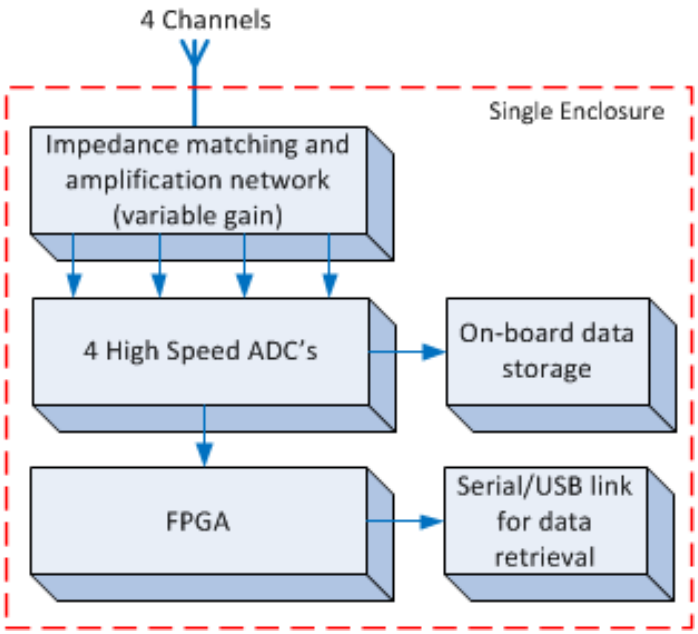

Figure 1.1. System Block Diagram

well as the relation between the EMI strength and distance from ESD sources, in an EMI-based strength system. This method applies to smaller test area. However, the received signal strength indicator (RSSI) system applies to a larger test area and also uses the reverse GPS principle involving the use of mobile and stationary nodes to determine the location of the ESD event.

This paper discusses an alternative to the EELS which uses four EMI detector but rather than using a high-speed real-time digital oscilloscope, we replace it with high-speed sampling ADCs. The ESD locator system is made cost-effective and also provides ease of installation and use. These ADCs along with the front-end analog receivers can be mounted in a single enclosure with a single power-supply to reduce the complexity of the system. Provisions can also be made to directly use the system with a laptop that processes and saves the data in a suitable format. While the paper discusses the principles behind the working of the system, it focuses on the

implementation of the ADCs and using the high-speed ADCs to detect and capture the ESD events.

2. DESIGN CONCEPT

An ESD event locator system uses reverse time of arrival process to localize ESD events in a 3D space. A previously implemented system included 4 antennas to receive and capture radiated emission generated by the ESD event. The system essentially used a high-speed sampling scope to capture and store the data which can be processed later. An inherent difficulty of the system was high cost of the oscilloscope and its under-utilization for this purpose, so the current approach uses high-speed sampling ADC's instead of the scope. The high-level block diagram of the system is shown in Figure 2.1.

The input closed loop buffer is used to match the input impedance with the antenna. A biased tee network is used to power the remote input buffer from the next stage though the coax cable. Next, the high dynamic range of GALI84+ device is used to increase the signal strength to an acceptable level. The device provides a gain of around 25 dB from DC-6GHz with P1dB of 21dBm. A wideband bias choke is used to isolate the RF and DC at the output of the amplifier. A digital switchable attenuator can be controlled in steps of 0.5dB up to 31.5 dB either from FPGA or using manual DIP switches. Optional amplification stage is provided to further improve the signal level.

The amplified signal is provided through a resistive splitter to a Trigger circuit and an ADC input. The Trigger circuit utilizes a window detector to produce logic high when the signal exceeds the threshold. A differential PECL comparator along with a PECL latch provides the window detector with the ability to capture signals with pulse widths as low as 500ps as required for ESD signal detection. The reference levels of the comparator can be controlled through an on-board DAC from the FPGA

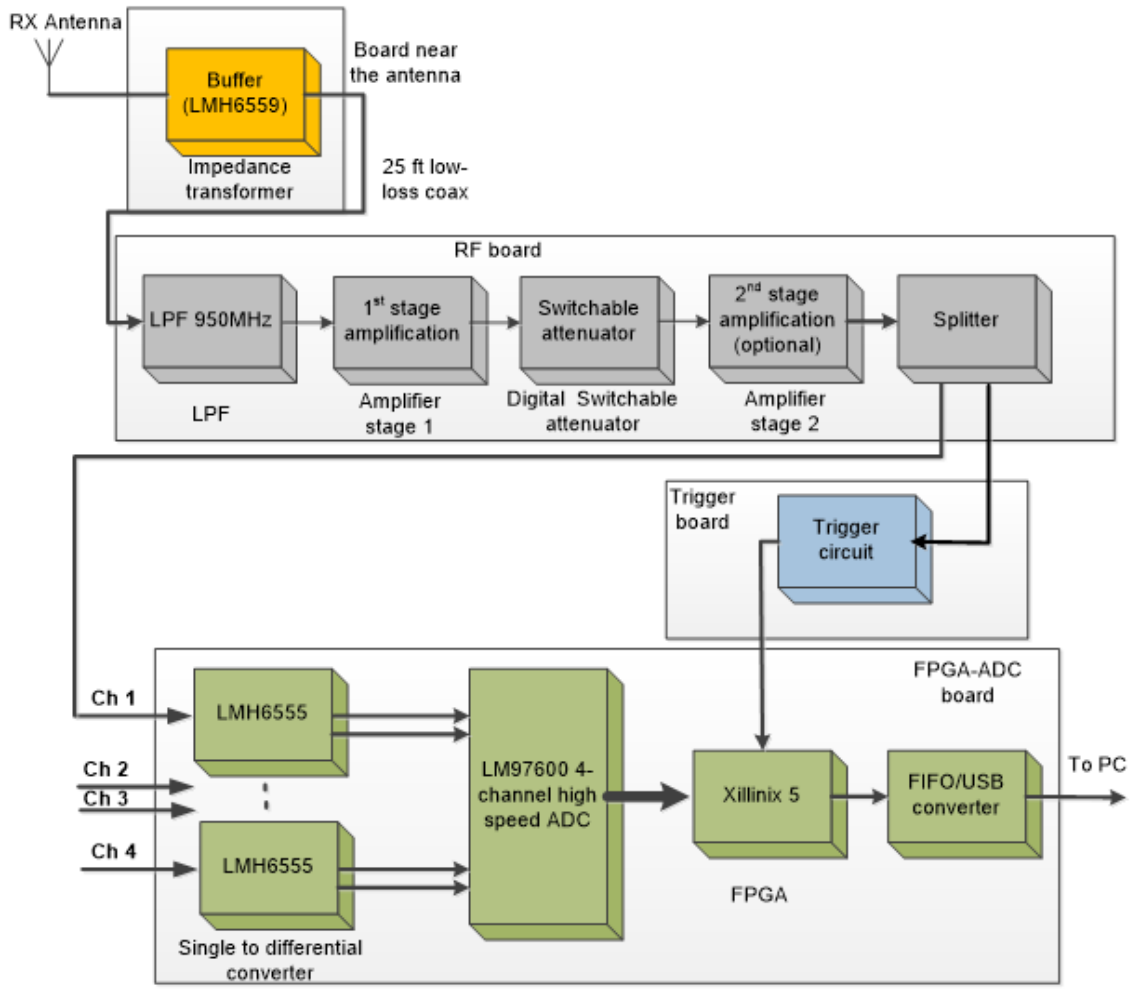

Figure 2.1. Block diagram of the ADC-FPGA based data acquisition system

or manually through potentiometers. The trigger signal forces the FPGA to capture ADC sampled data.

The ADC is a single/dual/quad channel high performance ADC that digitizes signals at sampling rates up to 5/2.5/1.25 GSPS. The digital output samples are transmitted to the FPGA at 50-Gbps using 8B/10B on 10 LVDS pairs. The input data from the RF stage is provided to the ADC through a high-speed single-ended differential driver. A fully integrated delta-sigma PLL and VCO IC is used to provide a 2.5 GHz low-jitter AC coupled differential clock signal to all converters in ADC.

The ADC can be controlled in extended control mode through a SPI interface from FPGA.

The rocket I/O trans-receiver provided in Xilinx Virtex 5 FPGA is utilized to receive digital samples from the ADC. The data from the ADC is temporarily stored in on-chip SRAM before being pushed though the FIFO/USB converter to PC. An on-board PROM facilitates storage of FPGA configuration during power-off. FPGA I/O ports are used to interface the ADC, PLL, trigger, and RF board control signals.

To power the ADC, FPGA, and other peripherals, a lower noise regulated power scheme provides power-up and power-down sequencing of 8 different supplies.

3. HARDWARE

3.1. ANTENNA AND RF BOARD

The RF board will be near the antenna to provide impedance matching with the antenna and buffer the signal to be sent over the cable to the ADC board. The layouts of the board is shown in 3.1 [6].

LMH6559 (U1) is a closed loop buffer used as an impedance transformer to match the antenna source impedance to the cable impedance. For board 1, power supply is provided by coax cable feed. The coax cable carries DC from the board 2 to board 1. Here it is decoupled using the ferrite beads L1 and L2. C2 blocks the DC from entering the buffer. L1 and L2 filters the DC from AC and provides it to the VCC (pin 1). Also it is used to provide a bias to the input using R1 and R2.

Port 1 connects the 25ft coax cable from board 2 to board 1. L4 and L3 ferrite beads form the filter that is used to the DC to the cable feed. A low pass filter U2 (LFCN 850+) is used to filter out higher frequency component greater than 1GHz. U3 is a GALI84+ wideband amplifier with high dynamic range. It provides 25.6dB gain @100MHz with P1db as 21.9dBm in the range of DC-6GHz. The output pin of the U3 is RF out and bias, so DC bias is provided using a L6-L5 ferrite beads and filtering network. The next IC is U4 (Dat-31R5-PP+) which is digitally switchable attenuator. It provides 31.5dB of attenuation in steps of 0.5dB. It can be controlled using 6 bit parallel interface. It can handle up to 24dBm of power. The board provides both the provision for controlling it from FPGA (software) and using manual DIP switches. Another optimal amplifier stage is provided if more gain is required. It is the same amplifier as before with the same biasing configuration.

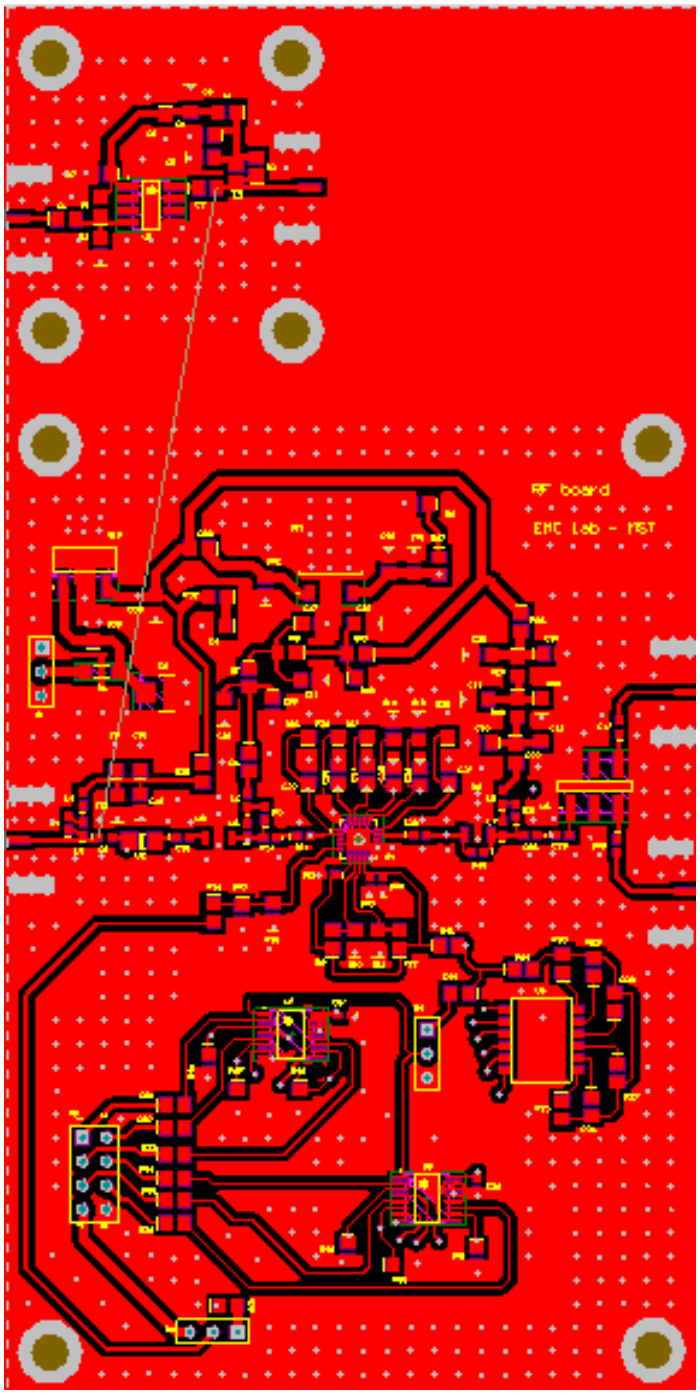

Figure 3.1. Antenna and RF Board

U7 and U8 (SN54LV157A) are data selectors to select digital inputs either from the FPGA (Through P5 header) or from the 6-DIP switch (U9). A 3-pin header P4 is used to select between the FPGA and the DIP switch inputs.

3.2. TRIGGER BOARD

With a constant stream of 50 GSps of data from the ADC, it would be challenging if not altogether impossible to transmit and process all the data inside the FGPA in real time. However, only data around an ESD pulse is meaningful, so the majority of the data can be ignored. To simplify the design, a trigger circuit was needed to monitor the analog input and trigger on desired input signals. This would allow the FPGA to capture only the desired data while removing the need to do any data processing. The objective was to design a circuit that could detect a pulse as small as 1ns while ignoring as much non-ESD phenomena as possible.

The first step was to determine what type of trigger method to implement. One of the most basic solutions was to create a simple comparator. However, an ESD pulse could be positive or negative. The next option was to implement an adjustable window comparator. With the voltages adjustable, the circuit could easily be tune above the noise floor to prevent being triggered on undesired signals. The layout of the trigger board is shown in Figure 3.2 [6].

Since an ESD pulse has a fairly unique profile, other triggering methods were considered that would help differentiate between random noise and an ESD event. One such method was to monitor the rise and fall times using a capacitor as a differentiator. Another option was to create a circuit to measure the width of a pulse and only trigger on those signals that fall within a specific range.

However, the simple window comparator was chosen as the most practical solution to design first. The window detector was implemented using the PECL

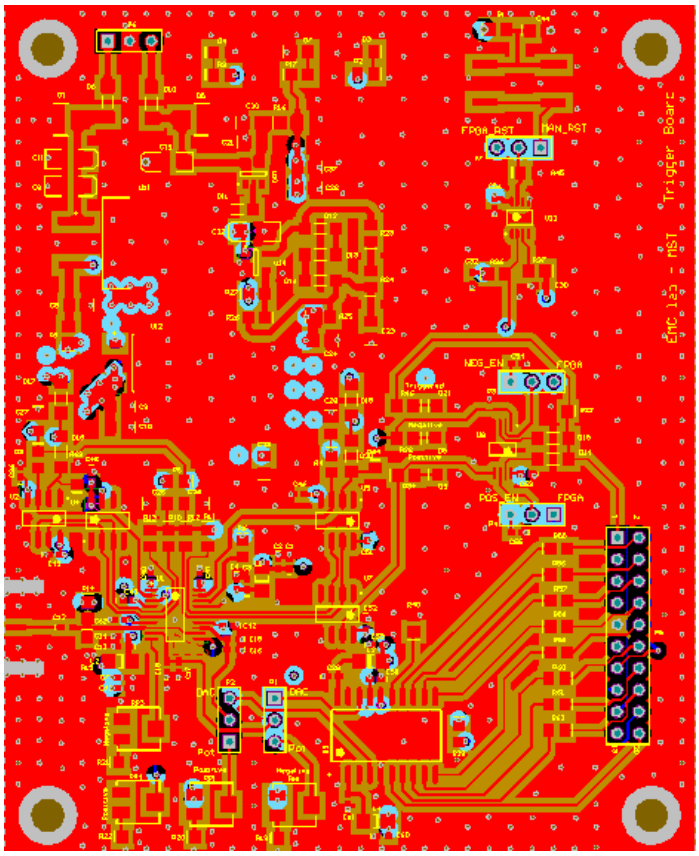

Figure 3.2. Trigger Board

ADCMP562 comparator. TLL, CMOS, and even LVDS were not fast enough to trigger on a 1ns ESD pulse so PECL was the only option. Although faster comparators exist, the ADCMP562 was chosen because its propagation delay of 700ps would reduce the possibility of it ringing and triggering on undesired signals. Other reasons the ADCMP562 was selected are that it has a latch input and internal hysteresis that can be controlled by an external pin which will help the circuit latch on the correct signal level.

To use the hysteresis pins, 50k potentiometers were placed on the pin to GND. Originally a DAC was going to control the hysteresis, but it was decided manual control would suffice since hysteresis would not need to be changed all the time. The input to the comparator was expected to be a low plus and minus voltage signal and

the comparator input voltage range was -2 to 3V. However, to avoid having to supply negative voltage references, the input was biased to 1.5V with a capacitor in series and pull-up and pull-down resistors. The standard input range then became 0-3V.

For the comparator circuit to be useful at all, the positive and negative reference signals had to be adjustable. Potentiometers were placed on the board to allow for manual adjustment. For automatic adjustments from FPGA a DAC was placed on the trigger board. A simple jumper pin selects between manual and automatic (FPGA) mode. On the first design, the DAC was connected incorrectly. By default the DAC is current driven and to use it in voltage mode there was only one small diagram in the datasheet that showed that the output and reference signals must be swapped from the standard configuration (OUT was then connected to the input and REF was the output).

The next step was to design a circuit that guarantees that the trigger signal will be received by the FPGA. The simplest method seemed to be to integrate a low pass filter to lengthen the pulse. However, this would not be the most stable method and it would be difficult to lengthen the pulse long enough for TTL logic to recognize it. Even then it requires the FPGA to be able to capture a very fast pulse. To avoid these challenges, the latch input on the ADCMP562 was utilized. The trigger board uses the MC10EL31 PECL D flip flop with a propagation delay of 475 ps to latch the comparator. The D flip flop also allows the trigger circuit to be reset after a latch.

An interesting caveat with the MC10EL31was how the D input needed to be pulled high. On simple TTL logic, this is no issue. But for this chips PECL levels, a high input at 25 degrees Celsius is 3.870 to 4.190V. Although a resistor divider circuit definitely could have been implemented, 2 diodes with the total voltage drop equal to 0.81 to 1.13V were connected to 5V with the hope of using less current than a resistor divider circuit. To guarantee that the diodes sustained the correct voltage

drop, a resistor was placed from the D input to GND. This resistor could possibly be removed or set to a very high value.

The interfacing buffers to the FPGA are all TTL so any signals connecting to the FPGA must be converted to or from PECL. The SN65ELT21 was used to translate the output trigger signals to 5V. The SY10ELT20VZG was used to convert the 5V reset output from the FPGA to PECL logic levels.

To enable or disable the positive and negative triggers, the dual AND gate 74HC2G08 was implemented. Jumpers were placed on the board to allow the channels to be enabled or disabled manually or to select FPGA control. Since there is no disable on the comparator, the AND gate just passes the trigger signal or blocks it. Finally, two diodes are used to OR the signals together so there is only one trigger signal going to the FPGA.

The final step was the power supply circuit. For 5V the LDO UA7805 was implemented. The 3V was supplied from a simple 3V LDO called the BA30BC0FP-E2. The -5V LDO MIC5270 supplied the -5V. However, the ADCMP562 comparator required -4.96 to -5.45V. Although a -5.2V LDO would have worked better, no simple LDO was discovered with that output voltage. To add to the challenge, no -5V LDO guaranteed high enough accuracy not to go above -4.96. So a diode was placed on the GND pin of the MIC5270 to drop the voltage slightly. Finally, a 3V sinking source was required for the PECL terminations. Since PECL voltage levels are always higher than 3V, the supply will only be sinking current. For this application, a negative output adjustable LDO was implemented. However, the input and GND pins were swapped to make it positive 3V. Since the device can only sink 200mA, diodes were placed on the output signal to add a voltage drop and allow the device to sink more current

The trigger board was implemented on 4 layers. The top and the bottom layers are the signal layers. The Second layer is the GND layer and the forth layer

is the power layer. The main challenge of the layout was the PECL termination and the high speed signals (the comparator input and the latch). All the single trace high speed signals are 17mils wide and the differential lines are 12mils wide with 10mils separation where possible. The differential pairs were routed to be as close as possible in length and as short as possible. The termination resistors were placed on the input side of the differential traces.

The main expectation was to be able to latch on a 1ns signal when the input exceeds the positive or negative references levels. The majority of the testing was done on a function generator that provided a trapezoidal pulse with a rise and fall time of down to 0.1ns and a pulse width of down to 0.1ns. The signal was fed directly into the trigger board and the inverted signal was fed into the scope.

The power supplies all provided the correct voltages, but they got warmer than expected. The trigger board was designed to be powered with +-12V, but with that much voltage drop across the LDOs, the board became almost too hot to touch. So during testing the voltage input was close to +-7.8V.

When the potentiometer on the hysteresis pin was set to zero ohms, the trigger circuit did not work. With zero resistance the hysteresis will be at a maximum and so the input would never exceed the hysteresis value. However, with the hysteresis set too low it could be possible for the comparator to be latched on a low level and therefore it would miss the trigger signal. So for the circuit to work well the hysteresis must be set lower than the desired input and higher than the noise. Testing showed 20mV or 20k ohms to be a good comprise.

As mentioned above, the DAC did not work at all during the first power up. However, only two pins for each channel were swapped. When connected correctly, the following data was collected from the DAC over the range of the output.

The trigger board implemented one of the most basic methods to latch on an ESD pulse. However, it was successful in triggering on even less than a 1ns pulse and

seemed to do quite well with not triggering on random noise. All of the other sections of the design including the power supply, the enable circuit, the reset, and the DAC functioned correctly.

3.3. POWER BOARD

Power for the FPGA and ADC board is separated on a new board for ease of debug and to reduce the complexity of the main board. The board also provides power to the Trigger boards as well. The LM97600 ADC requires power sequencing when it is turned on and the power sequencing is implemented for both the ADC as well as the FPGA. The power sequence of the Power Board is shown in Figure 3.3.

The Power board is responsible for supplying power to the four RF Boards, four Trigger Boards and the Main Board. The RF Boards and the Trigger Boards take power supplies of ± 12V while to provide the different power supplies to the Main Board for the FPGA and the ADC, it takes ± 5V and ± 1.5V .

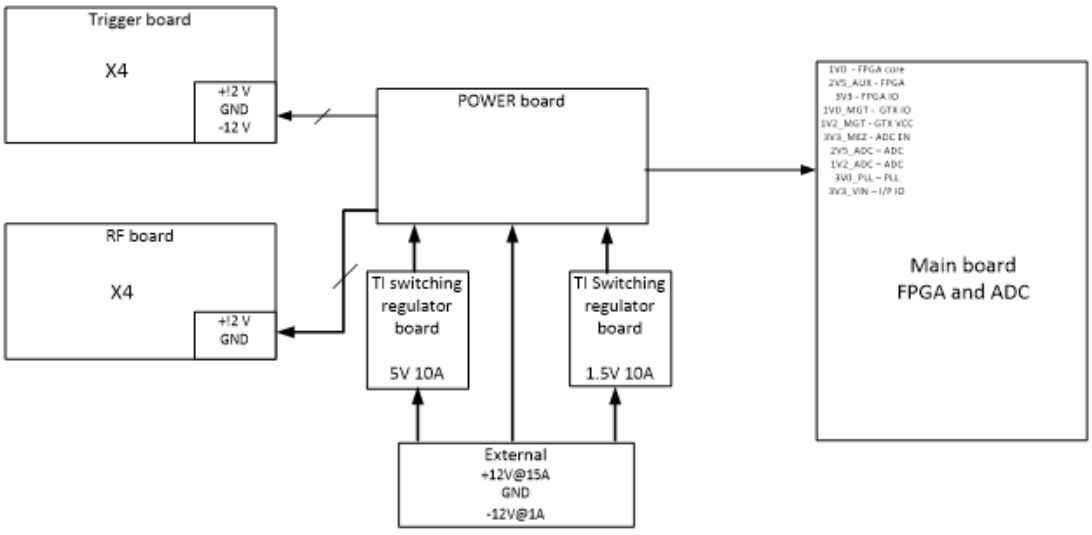

Figure 3.3. Power Scheme

The master input switch and enable for the power board consists of a 5V input to a 3.3V LDO which is the TPS73733 while a switch S1 is used as an enable to generate a 3V3 which acts as bias for the power sequence of the FPGA.

The power sequence of the FPGA consists of using the 3V3 as a bias. The FPGA Core takes a core voltage of 1 V and it uses the U9 TPS74901 which is a linear regulator with a programmable soft-start. It is open for a start time of 0.1 ms for the FPGA core voltage. The FPGA Aux takes a voltage of 2.5 V and uses the same regulator for a delayed start of 5ms.

The voltage to the IO banks of the FPGA is provided by the FPGA IO voltage which uses a 3.3 V linear regulator without any delayed start. The power supply for the GTX_IO and the GTX_VCC for the GTX Transceivers of the FPGA require a delayed start and use the TPS74901 for a start time delay of 6ms. The power sequencing for the FPGA is shown in Figure 3.5.

The power sequencing for the ADC is much more critical than the FPGA for the correct working of the ADC, so power sequencing circuits had to be designed for the ADC as well. The enable for the ADC used the 3V3 TPS73733 LDO which would be powered on after the FPGA was powered on. This would give a 3V3_MEZ to be used for generating the sequence of signals for the ADC.

The LM3880 is a simple sequencer IC which would give a power sequences with a delay of 10ms, 20ms and 30ms. The first sequence would be for the 2V5 for the ADC which used a low noise voltage regulator, the LP3878MR IC. This would take an input voltage of 5V. The second sequence would use the TPS74901 for the 1V2 ADC with a delay of 20ms. The third and final sequence would be for the 3V0 PLL which would involve a delay of 30ms and use the low noise LP3878MR IC as well. The power sequencing of the ADC is shown in Figure 3.4 and the layout of the Power Board is shown in 3.6 [6].

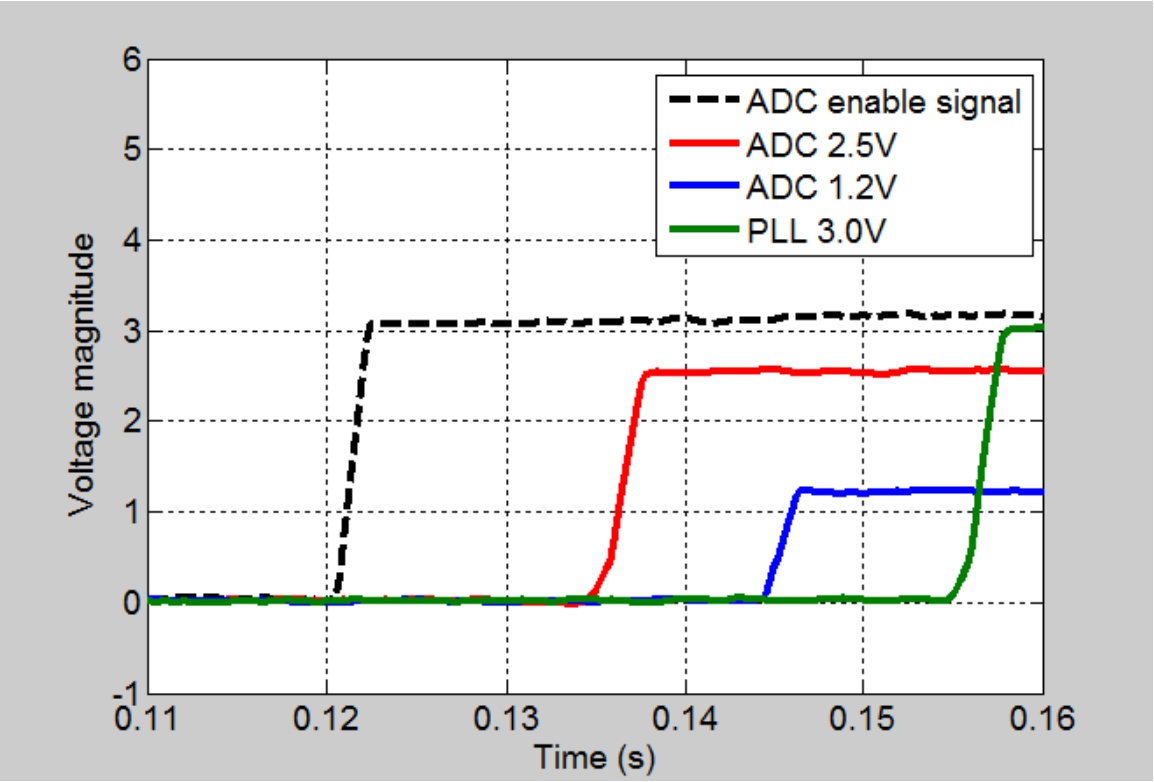

Figure 3.4. Power Sequencing for the ADC

3.4. MAIN BOARD

The Main Board is an 8-layer board which forms the primary data acquisition system and consists of an ADC and an FPGA that is interfaced to the ADC. It interfaces with the all the other boards such as the RF Boards and the Trigger Boards in order to complete the ESD Event Locator system. The layout of the Main Board is shown in Figure 3.7 [6].

The design of the board was referenced from the LM97600RB which uses the LM97600 ADC and the Virtex-5 FPGA for the data sampling and subsequent acquisition. The LM97600 is a high-speed 7.6-bit ADC with a maximum sampling rate of GS/s with 4 input channels to it. It operates in Single, Dual and Quad-mode with speeds of 5 GS/s, 2.5 GS/s and 1.25 GS/s respectively in the various modes. It

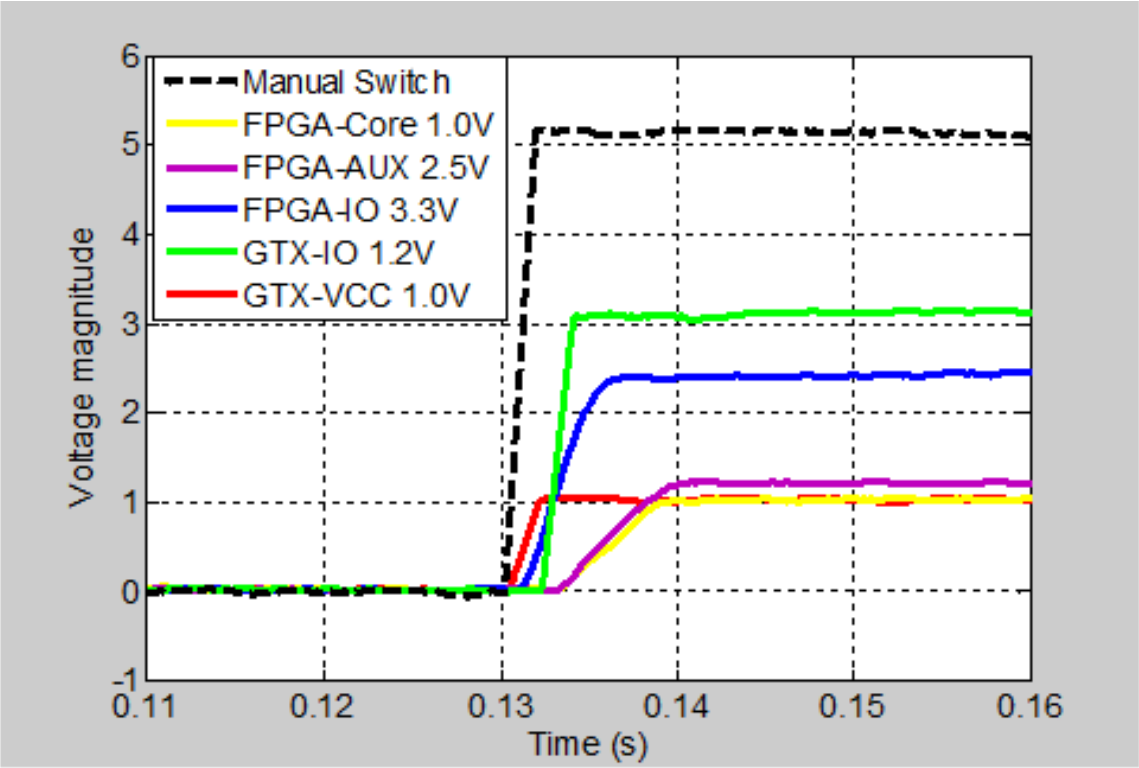

Figure 3.5. Power Sequencing for the FPGA

has a 10-lane high speed serial data output which uses LVDS to transfer out the data from the ADC.

The primary reason for the choice of this ADC was the availability of 4 channels on the ADC for each of the 4 signals from the antennas. The data from the ADC is encoded in 8b10b format. This ADC greatly reduces the synchronization challenges associated with the other ADCs. The system reduces to a single ADC and FPGA for the signal capture on all the four antennas. This would drive down the cost drastically, but the complexity in terms of handling data from ADC and FPGA design increases significantly..

The Virtex-5 FPGA, specifically the XC5VFX70T, was chosen as the FPGA to interface with the LM97600 due to the availability of 16 high-speed RocketIO GTX Transceivers which would be used to capture the data samples on the 10 output lanes

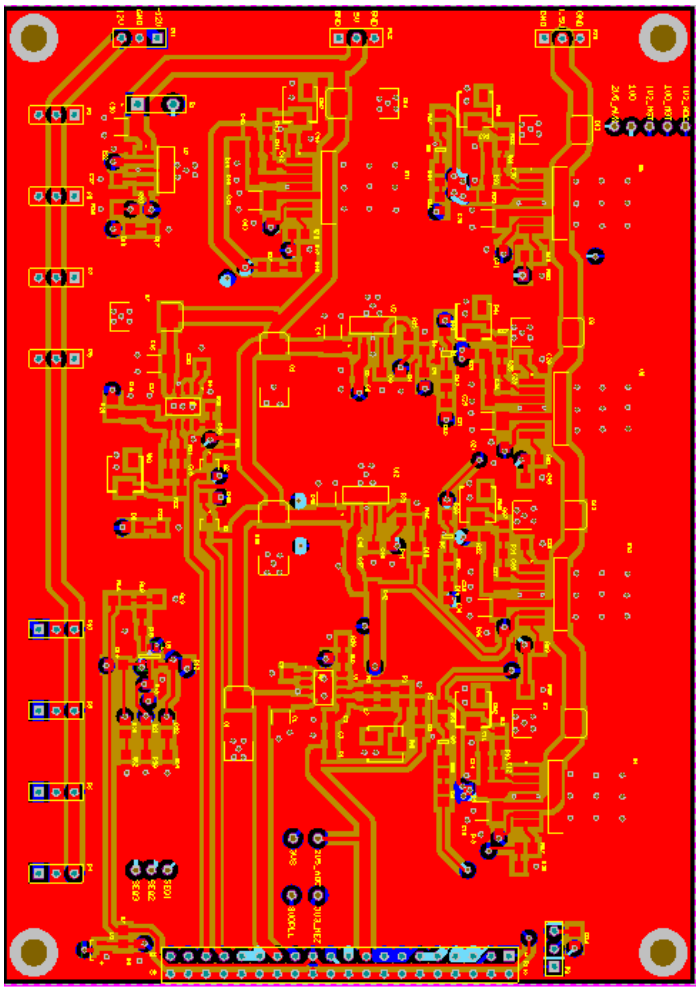

Figure 3.6. Power Board

of the high speed ADC. The GTX Transceivers are able to handle speeds from 150 Mb/s to 6.5 Gb/s which makes it an excellent option to be used with the ADC. The Virtex-5 FPGA would be responsible for interfacing and controlling the LM97600 for data capture and storage of the data samples as well as modifying the registers of the ADC via SPI lines setup between the FPGA and the ADC.

The Virtex-5 FPGA is volatile which means that the FPGA has to be reloaded with the FPGA image each time the Main Board is booted up. A Flash EPROM is

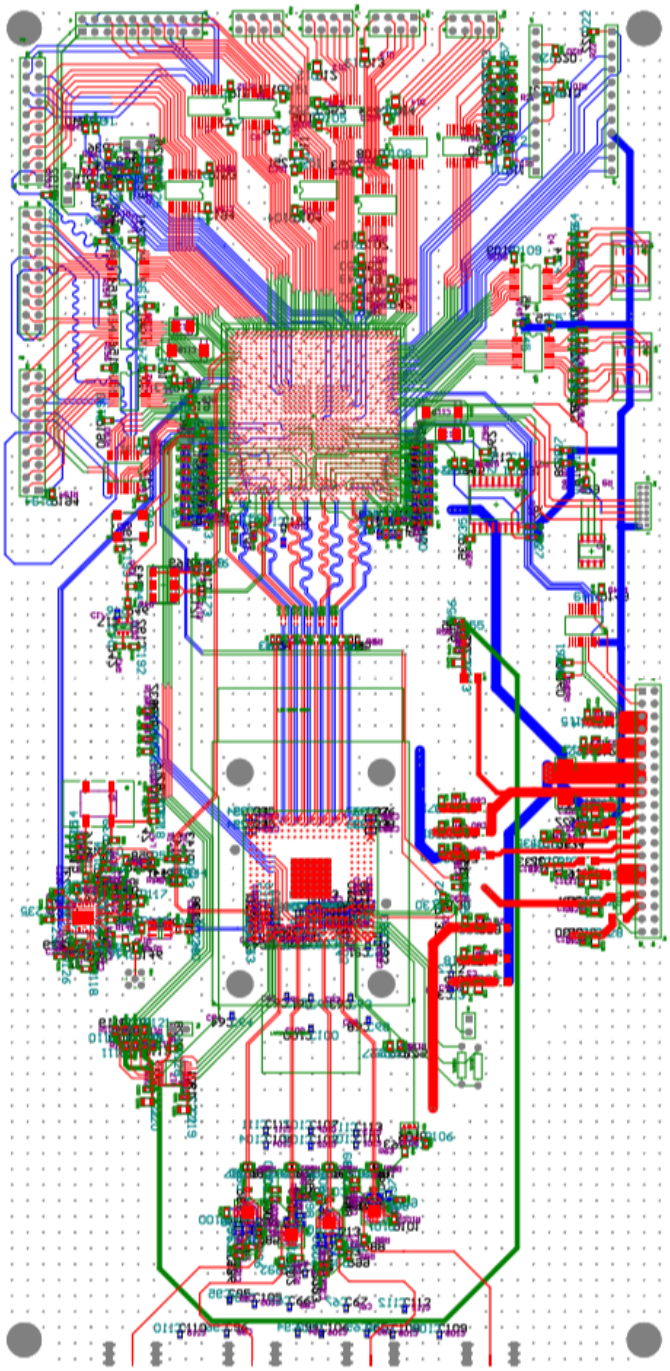

Figure 3.7. Main Board

placed on the Main Board to automatically load the FPGA image onto the FPGA each time the Main Board is re-booted.

The FPGA is provided with a clock oscillator of 100 MHz and the FPGA is also programmed with a PLL to provide clocks for the various modules present in the FPGA code. The GTX Transceivers have their own 200 MHz clock oscillator connected to the banks of the FPGA consisting of the GTX Transceivers.

There is a JTAG interface provided on the Main Board and connected to the FPGA for debugging and programming purposes. This allows the Main Board to be troubleshooted when the FPGA is being programmed to verify the proper working of the ADC and the FPGA. In order to transfer data from the FPGA to the PC, a USB FTDI IC is provided to interface with the FPGA to read/write to the registers of the FPGA and subsequently transfer off the data from the internal memory of the FPGA to the PC for post-processing.

The IO pins of the Virtex-5 FPGA are also responsible for interfacing with the trigger board since the trigger board will send the trigger signal to the FPGA for the FPGA to capture the ESD signals from the ADC on the occurrence of the trigger signal. The FPGA is also responsible for resetting the trigger board once the data samples have been acquired. In the automatic mode, the FPGA is also responsible for setting the reference voltages of the comparator via the DAC provided on the trigger board. The Main Board provides pin connectors for these signals from the four trigger boards.

The Main Board also provides pin connectors for the control of the 6-bit Digital Attenuators on the RF Boards and these are also set by the FPGA if they are in manual mode.

The ADC is provided with a 2.5 GHz PLL and VCO and low jitter for the four converters in the ADC. The PLL is also interfaced through SPI lines with the

Virtex-5 FPGA for extended control. The power supplies for the FPGA and the ADC are from the power board as has been previously discussed.

4. DATA ACQUISITION

4.1. OVERVIEW

The data acquisition portion of the project refers to the acquisition of the data samples from the ESD events which are stored on the Block RAM (BRAM) of the FPGA. Since the FPGA is only responsible for the capture and the storage of the data samples on the occurrence of the trigger signal, we require the transfer of the data samples from the BRAM of the FPGA to the Personal Computer using a communication protocol such as a Universal Serial Bus (USB). The Virtex-5 FPGA on the Main Board is responsible for the storage of the data samples and a USB Microcontroller (MCU) is located on the Main Board which interfaces with the FPGA for communication between the PC and the Main Board.

The objective was to transfer the data samples to the PC using a software such as MATLAB©which would enable the post-processing of the data samples for determining the location of the ESD event. WaveVision 5©, a data analysis and data acquisition software from Texas Instruments®, was used as a reference to manipulate the registers of the FPGA as well as the registers LM97600 Analog-Digital Converter to achieve the final objective of acquiring the data from the BRAM of the FPGA.

4.2. WV5 DLL

The WV5 DLL provides a set of functions that will allow the caller to configure and communicate with the data I/O boards and the devices connected to those I/O boards as illustrated in the high-level architecture of WV5 in the Figure 4.1. The main objective of the DLL is to provide access to the data both generated and consumed

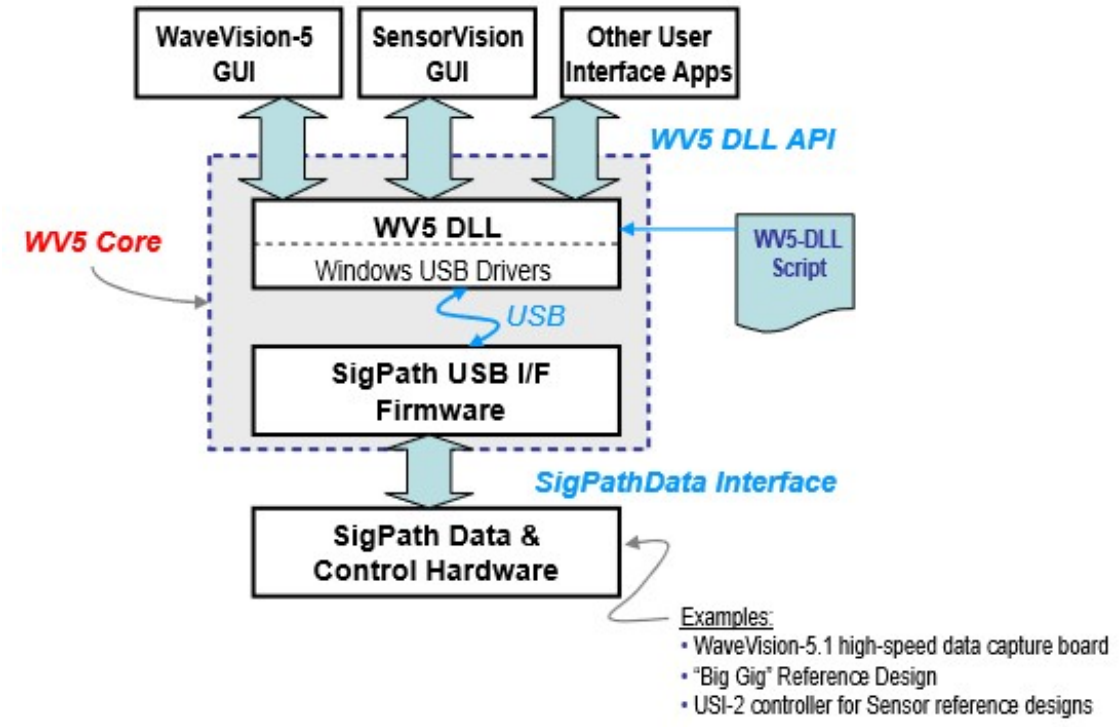

Figure 4.1. High-Level Archtecture of WV5

by the device under test (DUT). The interaction between the application and WV DLL is illustrated in Figure 4.2.

The WV5 DLL gives the option to interface with the board by using alternative applications such as MATLAB©in order to gather the samples from the FPGA. In the figure above, the WaveVision-5 GUI application can be replaced with an alternative application such as MATLAB/textcopyright as long as the WV5 DLL is present and use the APIs provided by the wv5.dll to communicate with the board. The interaction between the application and the WV DLL is also illustrated in Figure 5.2. The WV DLL provides a set of APIs using which the communication between the PC and the Main Board is established, the registers of the FPGA and the ADC are modified and finally the data acquisition takes place.

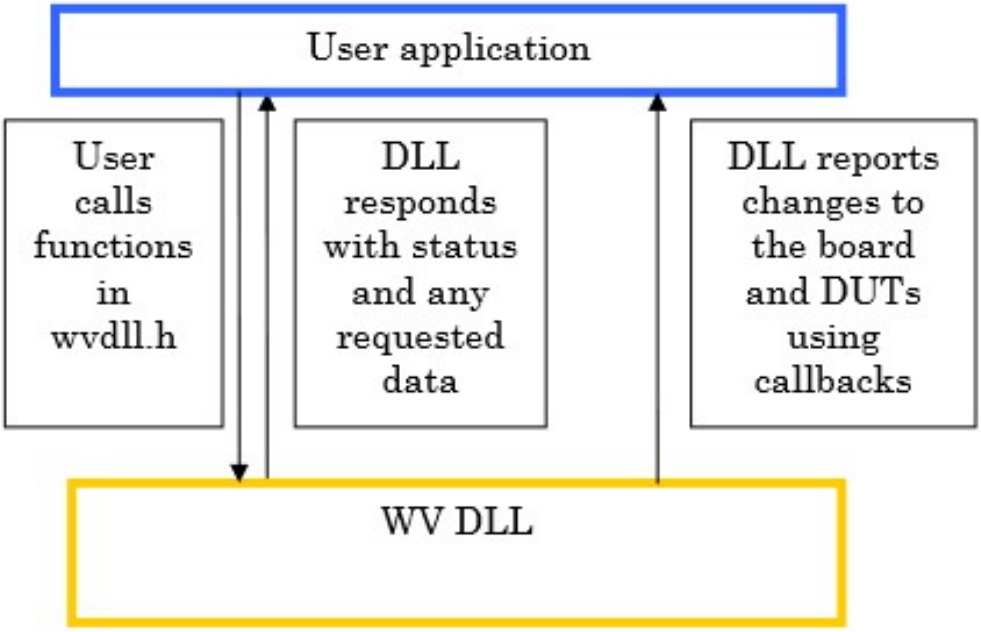

Figure 4.2. Interaction between application and WV5 DLL

4.3. FILES REQUIRED

The application to interface with the Main Board requires the presence of the files in Table 4.1:

4.4. APPLICATION INTERFACING

MATLAB©was used to interface with the hardware and to acquire data from the FPGA to the PC using the API functions provided by the WV DLL with the presence of the files mentioned in Table 5.2. This forms the core of the second part of the thesis which is to acquire the data from the Main Board to the PC in order to perform post-processing on the signals. This section describes the various API functions used to successfully communicate to the Main Board from the PC using the

Table 4.1. Files Required

Name	**Description**
wvdll.dll	DLL
libcint.dll	C Scripting Engine
wv5.sys	Windows Communication Driver
wv5.inf	Information file
wvdll.h	Programmer's interface
hardware\firmware_images\wv4.x_cy7c68013a.bix	Cypress Microncontroller Image
hardware\fpga_images\spio5.5_xc5vsx95t_lm97600.bit	FPGA Image
hardware\scripts\image_map.xml	File to map boards and devices to their support files

USB communication protocol. As previously mentioned, the Main Board is equipped with a USB microcontroller which provides a hardware interface to the FPGA from where the samples are acquired.

4.4.1. Initialization. The initialization section involves loading the API functions of the WV DLL library and then subsequently loading the API functions in a series of steps to initialize and interface with the FPGA board by identifying the board connected, loading the MCU image and then finally loading the FPGA image onto the FPGA. Since the FPGA is volatile, it needs to be loaded with the FPGA image each time the Main Board is booted up. The initialization routine was written in the file labelled Initialization.m. The APIs used in the initialization routine are listed below and their purpose defined. The APIs are listed in the order of their use in the initialization routine.

WvBoardEnum ('ulongPtr', 'cstringPtrPtr', 'WvVersionNumberPtr')

This API enumerates the boards that are connected to the USB and lists the number of boards connected, their names and the board names.

WvBoardOpen ('ulong', 'cstring', 'ulongPtr', 'WvVersionNumberPtr')

This API will open the board for access and determine the Cypress MCU on the board and it's version and will load the required Cypress MCU firmware by selecting it from the firmware images folder.

WvBoardDUTEnum ('ulong', 'ulongPtr', 'cstringPtrPtr')

The API will determine the number of DUTs present on the board and will list the names of the DUTs present on the board.

WvBoardLoadFPGA ('ulong', 'ulong', 'cstring', 'WvVersionNumberPtr')

This API will load the FPGA image from the FPGA images folder based on the DUT name and in our case, it is the LM97600 DUT.

WvBoardReadDUTInfo ('ulong', 'ulong', 'WvDUTInfoPtr', 'WvVersionNumberPtr')

This API will gather information on the DUT such as the version number of the DUT based on the DUT that is selected.

WvGetSamplingFrequency ('ulong', 'ulong', 'ulong', 'doublePtr')

This API will determine the sampling frequency of the DUT which for the LM97600 is 5GS/s. The initialization is completed at this stage and the Main Board and the FPGA is ready for the data to be acquired.

4.4.2. Register Initialization. This section deals with configuring the registers of the LM97600 since the ADC uses all four of its converters and there are 4 channels from which receives an input. The 4 inputs are from the 4 antennas which all capture the ESD event. In order to use all the 4 channels and to configure the inputs to the different channels so that the ADC is operating in the Quad mode, the register has to be set as specified in the datasheet which is shown in Figure 4.3.

The FPGA communicates with the LM97600 ADC through the Serial Peripheral Interface (SPI) and the WV DLL provides APIs for communicating with the ADC registers via the Cypress MCU and the Virtex-5 FPGA. In order modify the configuration register to work in the Quad mode, low-level API functions are used to write to the registers. The configuration register is a 16-bit register and the Quad-mode requires the bits 6:15 modified based on the datasheet specifications. The functions used for this purpose are described below. The library provides two ways to write to the ADC registers. The first method is using the API method and the second method is using the manual method which uses bit-banging to configure the ADC register via SPI. One of the two methods has to be selected and this is done by configuring the SPI Interface Configuration Register of the FPGA shown in Figure 4.4. The FPGA also uses the SPI Manual Control Register shown in Figure 4.5, for bit-banging and the two methods are described below. The API method requires that the all the bits of the SPI Interface Configuration Register are set to the value 16. The SPI Address High and te SPI Address Low registers are written with the addresses and the values that need to be written to the registers. However, issues were experienced using the API method and the ADC registers were unable to be written to, using this method. The API function used for writing to the register of the FPGA is shown below.

Addr: 01h (0 0001b)																POR state: 0000h
Bit	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Name	Mode		Input Select								DC	Res	TPM	Res	BMP	STA
POR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits 15:14	ADC Mode: Selects the operating mode as follows: 00 = Single ADC (Samples at 2xFclk) 01 = Dual ADC (Samples at Fclk) 10 = Quad ADC (Samples at Fclk/2) 11 = Invalid setting. Do not use.
Bits 13:6	Input Select: Depending on the ADC Mode selection, determines which inputs are used as follows: Mode = Single ADC Selected input for all converters set by bits 7:6. Other bits unused. 00 = Input 1 01 = Input 2 10 = Input 3 11 = Input 4 Mode = Dual ADC Selected input for converters A and C set by bits 7:6 Selected input for converters B and D set by bits 9:8 Other bits unused. 00 = Input 1 01 = Input 2 10 = Input 3 11 = Input 4 Mode = Quad ADC Selected input for converter A set by bits 7:6 Selected input for converter B set by bits 9:8 Selected input for converter C set by bits 11:10 Selected input for converter D set by bits 13:12 Other bits unused. 00 = Input 1 01 = Input 2 10 = Input 3 11 = Input 4
Bit 5	DC Coupled Mode Select The default setting of 0b selects AC coupled mode for all inputs. Setting this bit to 1b selects DC coupled mode for all inputs.
Bit 4	Reserved Must be set to 0b.
Bit 3	ADC Test Pattern Mode Select Settings this bit to 1b replaces the normal ADC output with a configurable test pattern output as set by Register Dh.
Bit 2	Reserved Must be set to 0b.
Bit 1	Clock Bump Setting this bit to 1b "bumps" or "swallows" one input clock, to shift the order of the interleaved converters one later. This bit must be cleared before setting it again.
Bit 0	Set t_{AD} Adjust Setting this bit to 1b enables the built in t_{AD} adjustment circuitry. The amount of aperture delay added is selected via Register Ch.

Figure 4.3. LM97600 Configuration Register

Address	Register	Reset	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0xD022	**SPI CFG**	**8'h00**	-	-	SCK_SL		BDL	SDS		

Bit [5:4]	R/W	SCK_SL[1:0]	Clock Frequency Select These bits select the clock frequency for the ADC SPI interface. When these bits are all 0, manual mode is selected and all the accesses to the ADC are performed by bit-banging the appropriate bits in the SPI_MCTL register. SCK_SL encodings are shown in the table below.
Bit [3]	R/W	BDL	Bidirectional Data Line When this bit is set to 1, the SPI interface uses one bidirectional signal rather than two separate signals for input and output. The MOSI signal becomes bidirectional and the MISO signal is unused.
Bit [2:0]	R/W	SDS	SPI Device Select These bits select one of 8 target devices to be accessed by the SPI interface. In the current implementation only one device is present and therefore these bits should be set to 0.

Figure 4.4. SPI Interface Configuration Register

WvNewestHardwareWrite ('ulong', 'ulong', 'WvDebugAccessMethod', 'ulong', 'ulong', 'ulong', 'ulong', 'ulong', 'ulong', 'ulong')

This API allows writing to the registers of the FPGA as well as communicate to the ADC registers throught the SPI lines between the FPGA and the LM97600 ADC. The ADC configuration register is set to Quad-Mode using this method.

This API allows writing to the registers of the FPGA as well as communicate to the ADC registers through the SPI lines between the FPGA and the LM97600 ADC. The ADC configuration register is set to Quad-Mode using this method.

In the bit-banging method, the above API is repeatedly called and the SCLK and the SDO are repeatedly asserted and de-asserted to pass the data to the ADC registers until the bits that need to be asserted are all sent resulting in the ADC being set into Quad-Mode. This process has to be repeated every time the Main Board is powered on and the FPGA is programmed. The first method using the API functions failed to set the registers in Quad-mode but the second method of bit-banging set the

Address	Register	Reset	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0xD021	**SPI _MCTL**	**8'h00**	-	-	EBD	OBEN	SCS	SCLK	MOSI	MISO

Bit [5]	R/W	EBD	External Buffer Direction Control This bit is effective only when the BDL bit in the SPI_CFG register is set to 1 and Manual Mode is selected. It controls the direction of an external buffer that may be placed on the SPI data line. It must be 0 while data is being written to the SPI target and 1 while data is being read.
Bit [4]	R/W	OBEN	Output Buffer Enable When set to 1, the serial data output buffer is enabled. This bit is used in Manual Mode only
Bit [3]	R/W	SCS	SPI Interface Select This bit drives the chip select signal. This bit is used in Manual Mode only
Bit [2]	R/W	SCLK	SPI Interface Clock This bit drives the clock signal. This bit is used in Manual Mode only
Bit [1]	R/W	MOSI	Master Output / Slave Input This bit drives the serial data output signal This bit is used in Manual Mode only
Bit [0]	R/W	MISO	Master Input / Slave Output This bit returns the status of the serial data input signal

Figure 4.5. SPI Interface Manual Register

configuration register of the LM97600 ADC to work in Quad-mode.

4.4.3. Synchronization Request. The DUT which is the LM97600 and the Main Board were ready for data to be captured on all the 4 channels, however the FPGA SERDES Receiver has to be configured before the data being captured is considered valid. The synchronization is required when switching between internal and external clocking or when the frequency is changed. Synchronization will cause the GTX Receivers in the FPGA to be configured for the data rate that is set. This synchronization request must be performed before the start of data capture which would otherwise result in invalid data being captured by the FPGA. The Synchronization Control Register is shown in Figure 4.6

Address	Register	Reset	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0xD0A1	**SYNC_CTL**	**8'h00**	SYNC	EXT	DELAY			PS		
Bit [7]	R/W	SYNC	Sync pulse will be asserted and then this bit will be self cleared.							
Bit [6]	R/W	EXT	Setting EXT will set the pulse to a static high level.							
Bit [5:3]	R/W	DELAY	The Sync pulse has a delay compensation setting to compensate for FPGA routing delays. Delay will influence the phase shift selection by advancing the phase shift. Effective phase shift is determined by PS[2:0] – DELAY[2:0].							
Bit [2:0]	R/W	PS	The Sync pulse phase shift setting is determined by PS[2:0] as shown in the table below							

Figure 4.6. Synchronization Control Register

The synchronization is performed by performing two register modifications in the FPGA those of which have been shown in the figures. The EXT bit of the Synchronization Control Register needs to be asserted for the synchronization request to initiate and the EXT bit is automatically reset after the synchronization.

The second register that is modified is the GTX Reset Register which resets the GTX transceiver logic as well as the buffer of the transceiver by asserting the lowest 2 bits of the register sequentially.

The *WvNewestHardwareWrite* API is used to manipulate the registers of the FPGA by setting the addresses and the values to be written to them. While the synchronization request is completed at this stage, verification needs to be performed to confirm that the synchronization request is completed. This is performed by the ChanByteAlign routine.

4.4.4. Channel and Byte Alignment. The Byte and Channel Alignment routine is performed to verify that the synchronization request is completed successfully. This is done by reading the byte alignment registers and the channel alignment

A1ddress	Register	Reset	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0xD0F7	**GTX_RST**	**8'h00**	-	-	-	-	-	-	RX_RST	BUFF_RST
Bit [1]	R/W	RX_RST	Setting this bit will reset the transceiver Rx logic. This bit will self clear after the reset is asserted to the GTX transceiver modules.							
Bit [0]	R/W	BUFF_RST	Setting this bit will reset the transceiver elastic buffer. This bit will self clear after the reset is asserted to the GTX transceiver modules.							

Figure 4.7. GTX Reset Register

registers as illustrated in the figures. The lower 5 bits of the byte alignment registers and the lower 5 bits of the channel alignment registers are all set if the synchronization request is completed successfully.

If the synchronization request is not completed successfully, then the byte and channel alignment registers are not set completely resulting in faulty capture of data. The 4 registers are read using the *WvNewestHardwareRead* API function and the lower 5 bits of each of the registers are checked to verify this.

If the registers have been set, then the synchronization request is determined to have been run successfully and the data from the board is now considered valid. The Byte Alignment registers shown in Figure 4.8 and the Channel Alignment registers shown in Figure 4.9 need to checked to verify that the synchronization completed successfully.

4.4.5. Data Capture. The Main Board is ready for the data to be captured once the initialization and the synchronization is completed. The WV DLL offers API functions to acquire the data from the FPGA and certain parameters can be specified based on which the data is acquired from the FPGA to the PC.

Address	Register	Reset	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0xD0E7	**BYTE_ALIGNED_0**	**8'h00**	-	-	-	GTX4	GTX3	GTX2	GTX1	GTX0
Bit [4:0]	R/O	GTX#	This signal from the comma detection and realignment circuit is High to indicate that the parallel data stream is properly aligned on byte boundaries according to comma detection. 0: Parallel data stream not aligned to byte boundaries 1: Parallel data stream aligned to byte boundaries There are several cycles after RXBYTEISALIGNED is asserted before aligned data is available at the FPGA RX interface.							

Address	Register	Reset	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0xD0E8	**BYTE_ALIGNED_1**	**8'h00**	-	-	-	GTX9	GTX8	GTX7	GTX6	GTX5
Bit [4:0]	R/O	GTX#	This signal from the comma detection and realignment circuit is High to indicate that the parallel data stream is properly aligned on byte boundaries according to comma detection. 0: Parallel data stream not aligned to byte boundaries 1: Parallel data stream aligned to byte boundaries There are several cycles after RXBYTEISALIGNED is asserted before aligned data is available at the FPGA RX interface.							

Figure 4.8. Byte Alignment Registers

Address	Register	Reset	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0xD0F5	**CHAN_ALIGNED_0**	**8'h00**	-	-	-	GTX4	GTX3	GTX2	GTX1	GTX0
Bit [4:0]	R/O	GTX#	Indicates if the channel is properly aligned with the master transceiver according to observed channel bonding sequences in the data stream..							

Address	Register	Reset	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0xD0F6	**CHAN_ALIGNED_1**	**8'h00**	-	-	-	GTX9	GTX8	GTX7	GTX6	GTX5
Bit [4:0]	R/O	GTX#	Indicates if the channel is properly aligned with the master transceiver according to observed channel bonding sequences in the data stream..							

Figure 4.9. Channel Alignment Registers

WvBoardWriteCaptureStart ('ulong', 'ulong', 'WvCaptureSetupPtr')

This API is used to define the parameters of the capture such as the number of samples to be captured, initialize the buffer to hold the data to be received as define the mode of the capture such as single, dual or quad mode capture. The structure also defines the data format for the data to be captured in as well.

WvBoardReadCaptureData ('ulong', 'ulong', 'ulongPtr', 'ulong', 'ulongPtr')

The API is the one which reads the data from the FPGA and stores the data onto the PC based on which the waveforms will be plotted. This API can only be called once the previous API with the capture parameters has been used.

WvBoardWriteCaptureEnd ('ulong', 'ulong')

This is the final API used in the capture routine to complete the capture routine once the PC has acquired the data requested from the FPGA.

WvBoardClose ('ulong')

This API is required to close the communication between the DUT and the PC after the data has been acquired by the application.

4.5. RESULTS

The LM97600 ADC was given sinusoidal waveforms of different frequencies and initially the signal was connected to different input channels of the ADC to test out that data from each channel of the ADC was being captured successfully and then all the 4 channels were connected to different sources, with each source being sent from a signal generator. The number of samples was first set to 16000 and then set to 1000 which is an approximation of the ESD event.

The waveform in Figure 4.10 shows the result of capturing the data in single mode, while the waveform in Figure 4.11 shows the result from the second channel. The waveforms in Figure 4.12 and Figure 4.13 show the results from the third and fourth channel respectively. The waveforms in Figure 4.14 and Figure 4.15 show the results in the quad-mode for 16K and 1K samples respectively.

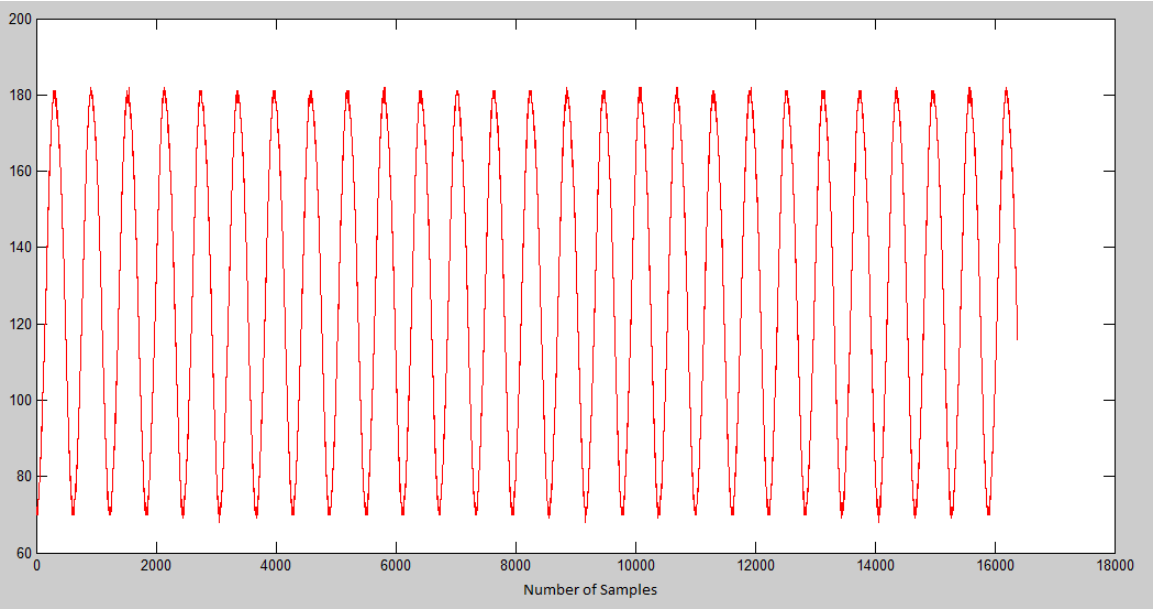

Figure 4.10. Signal from the first channel

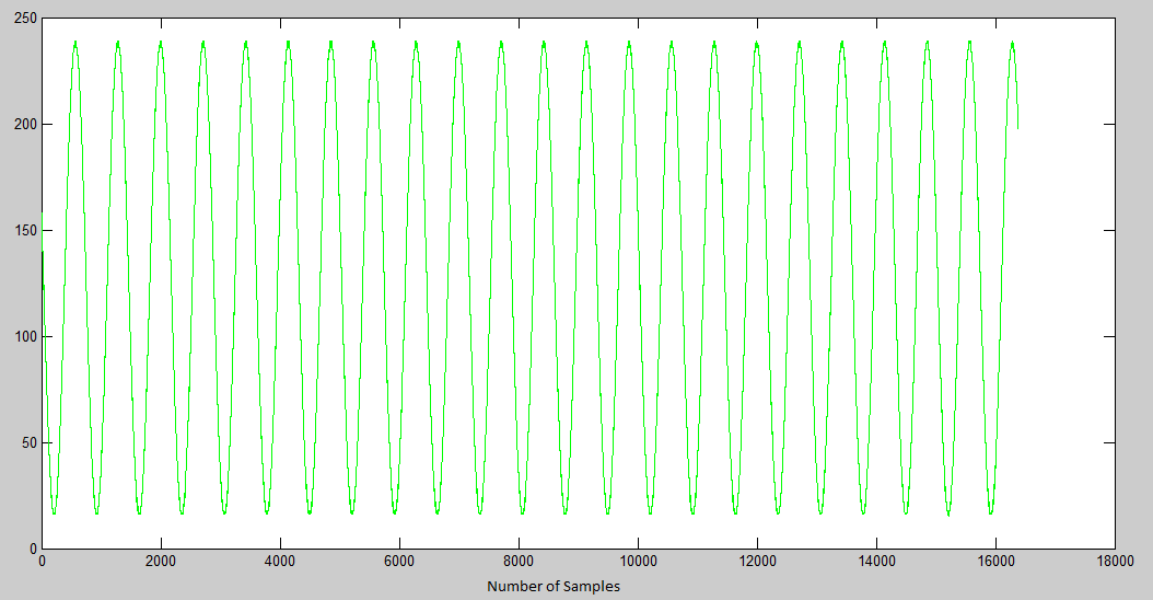

Figure 4.11. Signal from the second channel

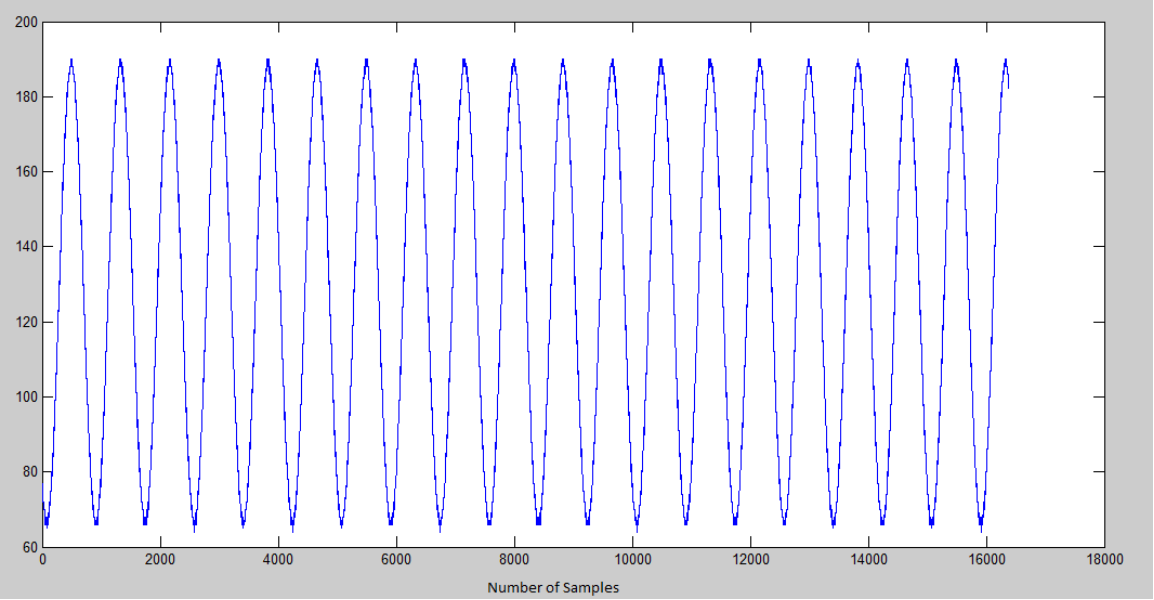

Figure 4.12. Signal from the third channel

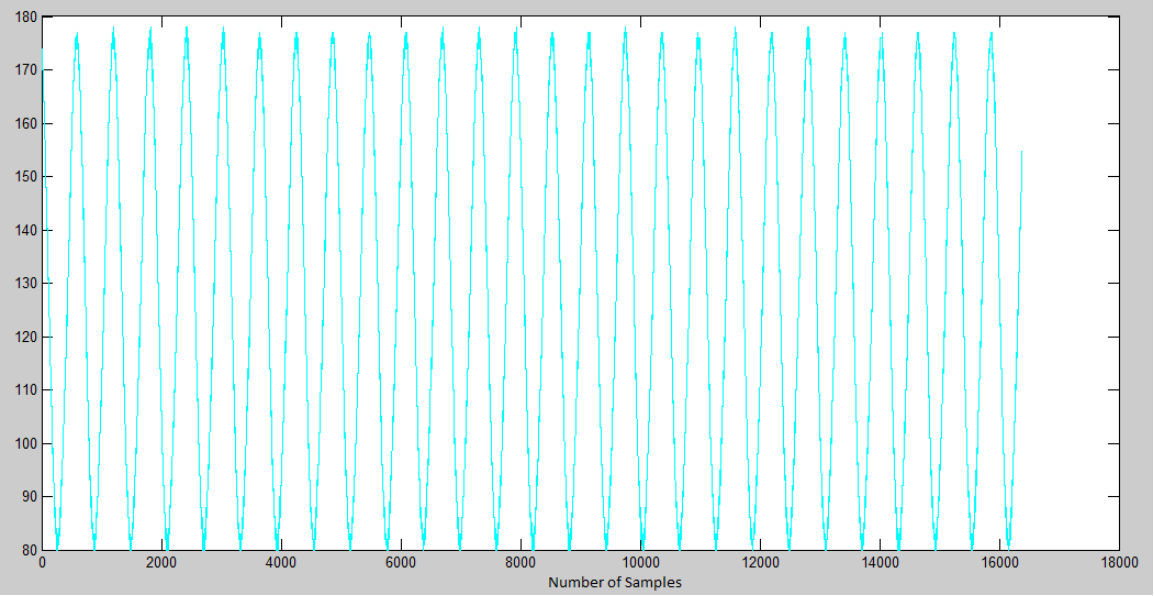

Figure 4.13. Signal from the fourth channel

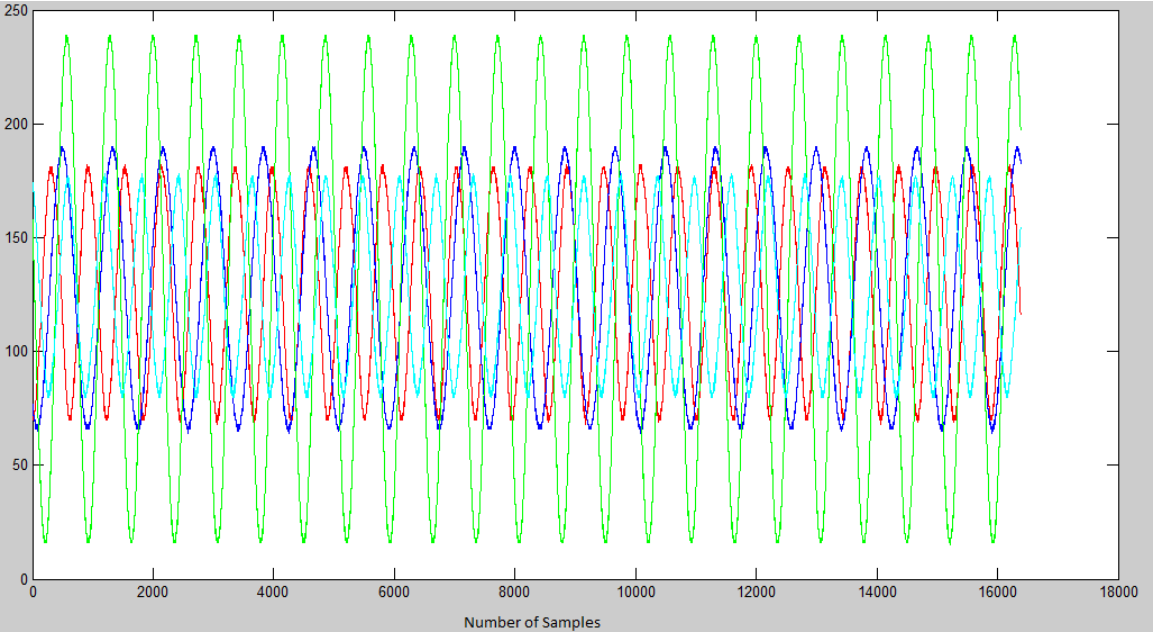

Figure 4.14. Signals from all 4 channels

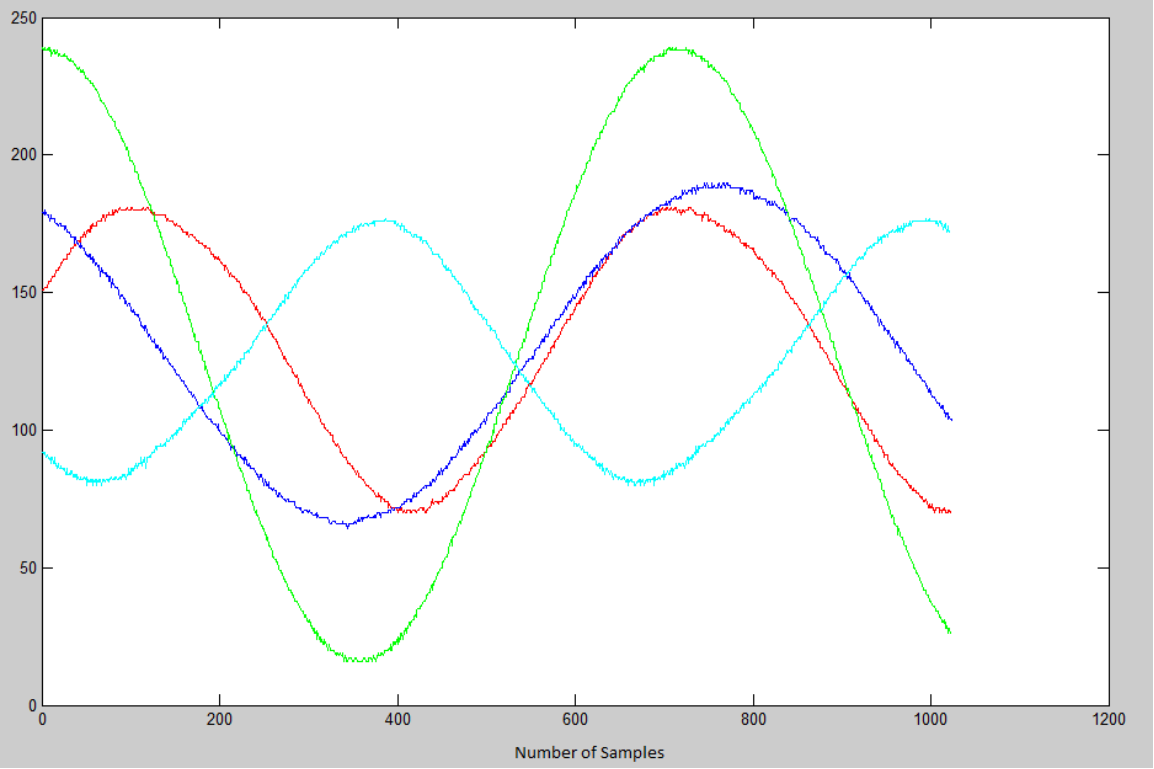

Figure 4.15. Waveforms captured from all 4 channels

5. DATA STORAGE ON FPGA

5.1. OVERVIEW

The LM97600 ADC integrated with the Virtex-5 FPGA required that the data samples of the ADC be stored in a location since there would be real-time acquisition of the data due to the difference of data rates between the ADC and the FPGA as well as the data rates between the FPGA and the PC. The LM97600RB which served as the reference board for the design of the Main Board, used a 32 MB Static random-access memory (SRAM) for the storage of the data samples which would later be retrieved using the slower USB communication protocol.

However, to keep the design of the Main Board simplistic, no SRAM was integrated on the Main Board and alternatives had to be studied in order to store the data samples from the ADC. Since the ESD events have a very short rise time as well as a short pulse width, 32 MB of memory and integration of an SRAM chip on the Main Board was not a necessity. The obvious alternative to storing the data samples was the internal memory of the FPGA.

FPGAs are equipped with memory elements such as flip-flops and more dedicated blocks of memory for storage purposes. In this instance, the Virtex-5 FPGA provides powerful 36-KBit block RAM/FIFOs which could be used for storage purposes. The Virtex-5 XCV5VFX70T provides 5328 KBs of memory for storage purposes. It should be remembered though, that this entire memory would not be available for the storage of the data samples and most of the memory would be implemented in setting up the FPGA registers for the interfacing and control of the LM97600 ADC using the FPGA.

The residual memory after the entire logic is implemented is studied and the memory remaining is considered for the data sample storage. This chapter of the

thesis looks at the implementation of a memory solution for the storage of the ADC data samples.

5.2. ANALYSIS OF AVAILABLE MEMORY

As previously mentioned, the Virtex-5 XCV5VFX70T provides 5328 KBs of memory for storage purposes and the implementation of the storage solution on the LM97600RB was studied and the LM97600RB uses an SRAM of 32 MB for the storage of the data samples. Using that as a reference design, we looked at the memory that was available on the FPGA after the acquisition logic to receive the data samples from the ADC was completed. The Table 5.1 shows the memory resource consumption of the FPGA in the reference design and the remaining memory that is available.

Table 5.1. Memory Consumption of the LM97600RB

Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of BlockRAM/FIFO	84	148	56%
Number using BlockRAM only	68		
Number using FIFO only	16		
Number of 36k BlockRAM used	63		
Number of 18k BlockRAM used	9		
Number of 36k FIFO used	12		
Number of 18k FIFO used	4		
Total Memory Used (KB)	2,934	5,328	55%

The table shows that there is 45% memory available which is 2,394 KB within the FPGA to store the data samples if it was chosen to store the data samples before they were transferred to the PC. It should be remembered that only the samples pertaining to the ESD events would have to be stored and not all data or signals arriving at the input channels of the ADC would need to be stored which would make the requirement easier as compared to storing all data arriving at the input channels of the LM97600 ADC.

The memory required for the storage of the ESD events is not huge since the ESD events are very fast, short events that do not span over a large time period. In order to calculate the memory and the number of samples that can be stored in the internal memory of the FPGA, the available memory is considered, the number of samples that can be stored is calculated and finally an estimate of the number of ESD events that can be stored in the available memory is calculated by basing an approximate time window for each ESD event. The calculation can be considered as follows:

$$2394 * 1000 = 2394000 \; bits$$

$$2394000/8 = 299250 \; samples$$

$$\sim 59.85 \; \mu s$$

If each ESD event is considered as $100 \; \mu s$, then the FPGA memory would be able to hold *600* ESD events which would be considered sufficient.

5.3. MEMORY DESIGN

The reference design for the LM97600RB was studied regarding the transfer of data samples from its acquisition from the ADC to them being stored in the SRAM. The objective is to replace the SRAM with the internal memory of the FPGA without upsetting the rest of the operation of the data sample acquisition from the ADC.

The Figure 5.1 shows the data flow path which consists of the data coming in through the 10 high-speed LVDS lanes between the Virtex-5 FPGA and the LM97600 ADC. The Virtex-5 FPGA was chosen solely based on the presence of the high-speed GTX transceivers since the ADC operates at 5GS/s. The FPGA consists of a SERDES which will process the incoming data and it is sent to the ADC Remapper

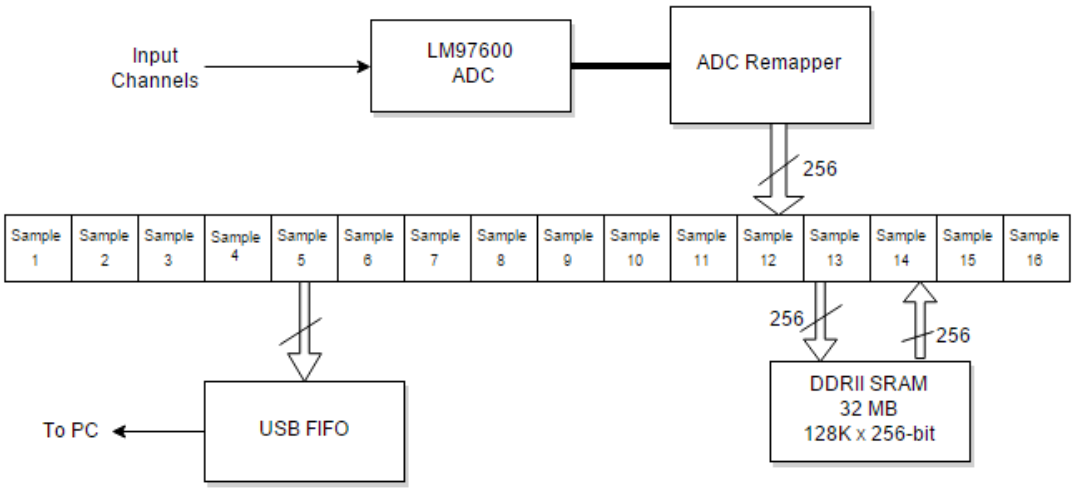

Figure 5.1. Data flow path of the LM97600RB

to rearrange the data samples arriving. The LM97600 encodes the samples in 8b10b format which is then remapped by the ADC Remapper.

The Data Transfer Control (DTC) module within the Virtex-5 FPGA is responsible for routing the data to either the SRAM in the LM97600RB or it sends the data from the SRAM to the PC via the USB MCU once it receives the data acquisition command from the PC. The SRAM is 256-bit wide and 128K deep which is a lot of memory and is not required for the current application.

The conceptual design of the LM97600RB is used and the functionality of the LM97600RB is preserved to achieve our goal quickly. The 256-bit data is made up of 16 samples which are sent to the SRAM for storage. The SRAM data is sent to the PC via USB when it receives the opcode from the PC requesting the data samples to be sent.

5.3.1. Memory Type. With the FPGA memory considerations already performed, the next step was to determine the type of memory to implement in order to

store the data samples in the internal memory of the Virtex-5 FPGA. The Virtex-5 FPGA offers the following memory options:

- FIFOs
- Block Memory
  - Single Port RAM
  - Simple Dual Port RAM
  - True Dual Port RAM
  - Single Port ROM
  - Dual Port ROM
- Distributed Memory
  - ROM
  - Dual Port RAM
  - Simple Dual Port RAM
  - Single Port RAM

The simplest solution to storing the data samples would be to use a FIFO which is 256 bits wide and store the data samples as they come to DTC via the ADC remapper and the data samples would be stored in 256-bit words sequentially. However, while the storing of the data samples is simple, it would be difficult to retrieve the data samples from the FPGA with the current setup as the USB MCU in the reference board is configured to use address locations to retrieve the data from the SRAM. There would need to be an alternate method to be devised in order to read the data from the FIFO.

Using the internal memory as Distributed Memory was discarded due to the lower performance as a result of the routing delays of the address present in it. While the Distributed Memory has a faster speed, the overall performance after the delays involved means that it is slower overall compared to the Block RAM.

The Block Memory especially the Single Port RAM gives us the simplest option of storing the data samples after the FIFO. However, it is easier to implement in this case, since it is quite similar to the SRAM in its implementation and would be easier to interface with the USB MCU when reading out the data samples from the internal memory of the FPGA to the PC. The Dual Port RAM could also be used, but the Single Port RAM serves the purpose of the application, so the BRAM option was chosen to store the data samples. The Figure 5.2 shows the block diagram for the sample storage in the BRAM.

5.3.2. Memory Implementation. The next step after the memory consideration is to implement the design and to generate the memory block within the Virtex-5 FPGA. Xilinx ©provides a CORE generator ®which makes it easy to implement a BRAM based on the requirements. The dimensions of the BRAM have to be defined and since the design is being kept as similar as possible to the SRAM, the width of the BRAM is defined as 256 bits as the data coming in from the ADC via the DTC to the BRAM would be of 256 bit width.

The depth of the memory though had to be calculated based on the available memory requirements. As previously discussed, the total remaining memory present on the Virtex-5 FPGA is 2,395 KB. This would mean that the maximum depth of the BRAM could be 9576. However, this would completely exhaust the internal memory of the FPGA so the depth of the BRAM was set to 8196 keeping in mind the address generation as shown in the Figures 5.3 and5.4.

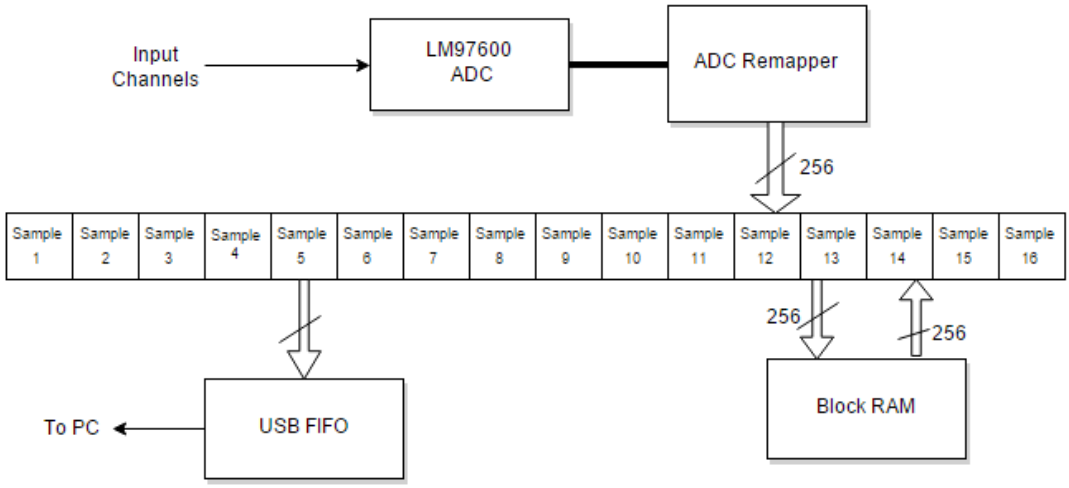

Figure 5.2. Data sample storage in Block RAM

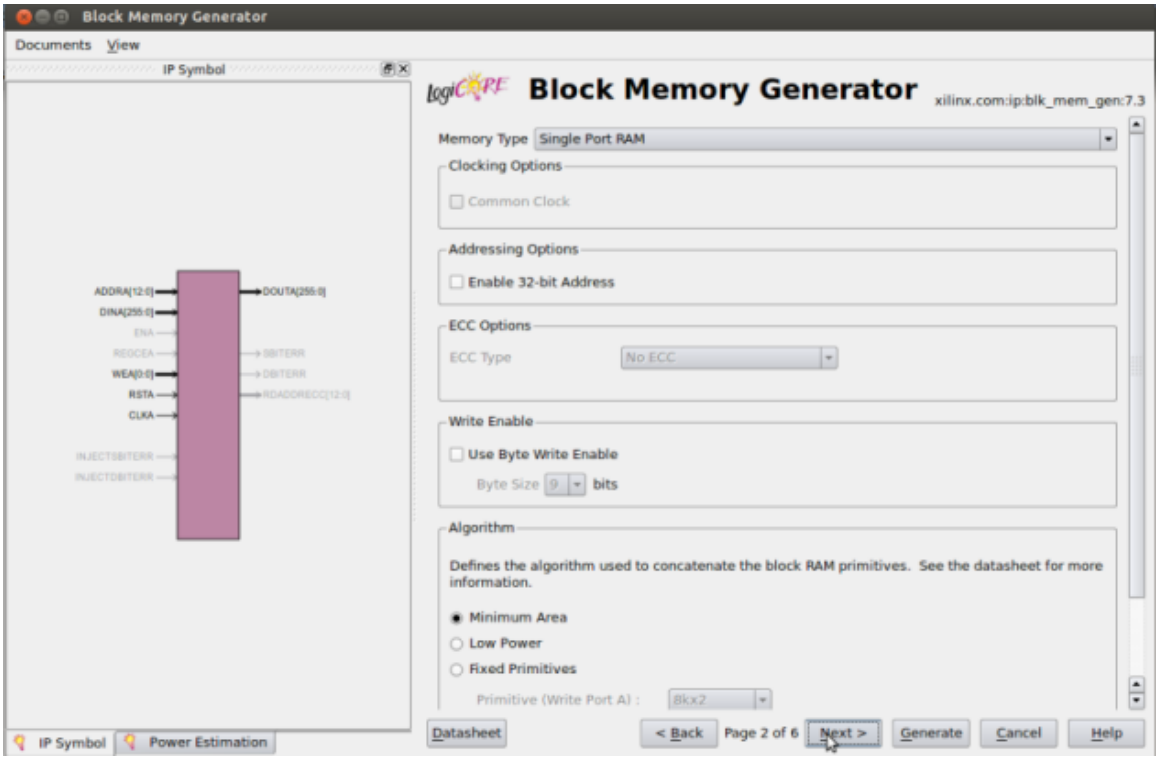

Figure 5.3. Block RAM Generation 1

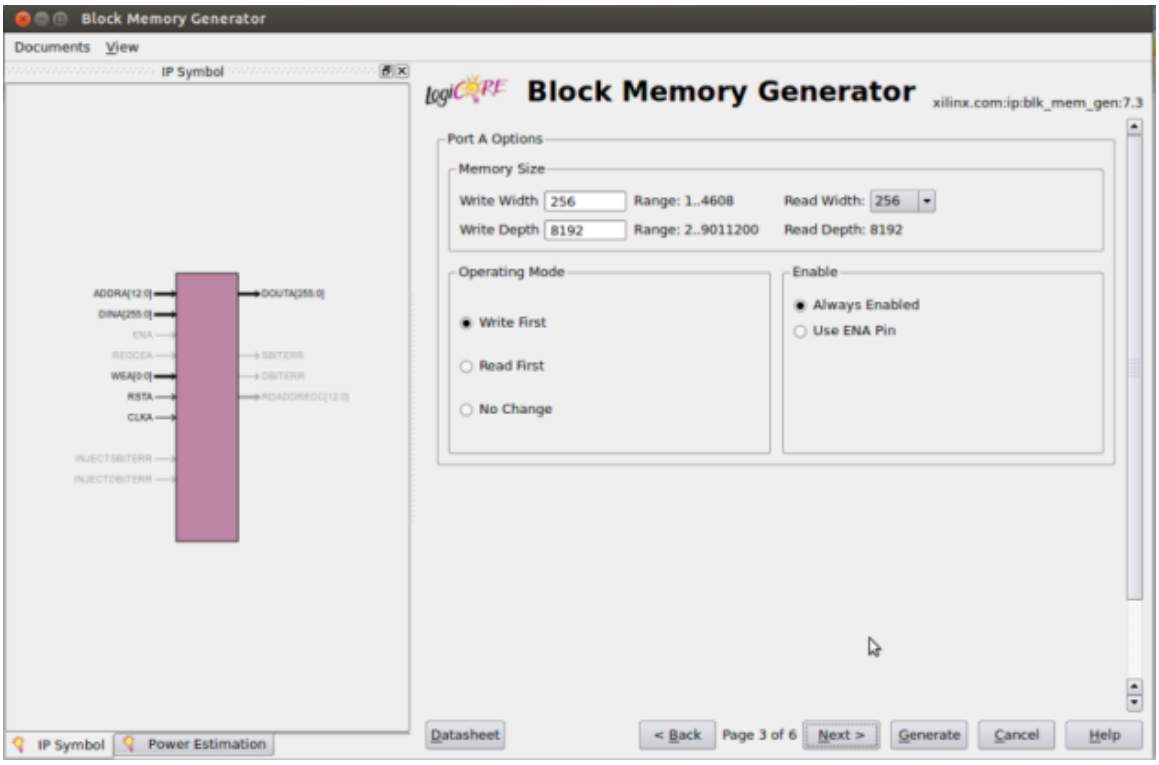

Figure 5.4. Block RAM Generation 2

5.3.3. BRAM Address Generation. The BRAM module was generated based on the specifications mentioned previously. In order to integrate the BRAM module so as to store the data samples on it, a key aspect of the storage was the address locations at which the data samples would be stored. The BRAM required addresses to store each of the 256-bit words in a different location.

The simplest method of address generation was to use a loadable up-counter which would generate a new address location for every clock cycle. Since there were 8192 address locations, the BRAM required a 13-bit address for reach location. Two 8-bit up counters were used to generate the address such that the 8-bits of one counter and the lower 5-bits of the other counter were concatenated to generate a 13-bit address.

This 13-bit address was cyclic counter and data samples are constantly over-written due to the high-speed sampling and constant influx of a large number of data samples.

5.4. RESULTS

The Block RAM for the data storage samples was successfully implemented without any issues and the data samples were acquired on a PC using both WaveVision ®and MATLAB ©. The code was tested out on the LM97600RB by replacing the SRAM component with the BRAM component. All the I/O pins from the Virtex-5 FPGA were disconnected and the BRAM was implemented by testing out by synthesizing the code and running it to capture the data samples from the LM97600 ADC.

The memory consumption of the Virtex-5 FPGA is shown in Table 5.2 after the BRAM has been implemented. The resulting waveforms from the data samples stored on the BRAM are also shown in Figure 5.5.

Table 5.2. Memory Consumption of the Virtex-5 FPGA including BRAM

Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of BlockRAM/FIFO	127	148	85%
Number using BlockRAM only	127		
Number of 36k BlockRAM used	120		
Number of 18k BlockRAM used	9		
Total Memory Used (KB)	4,482	5,328	84%

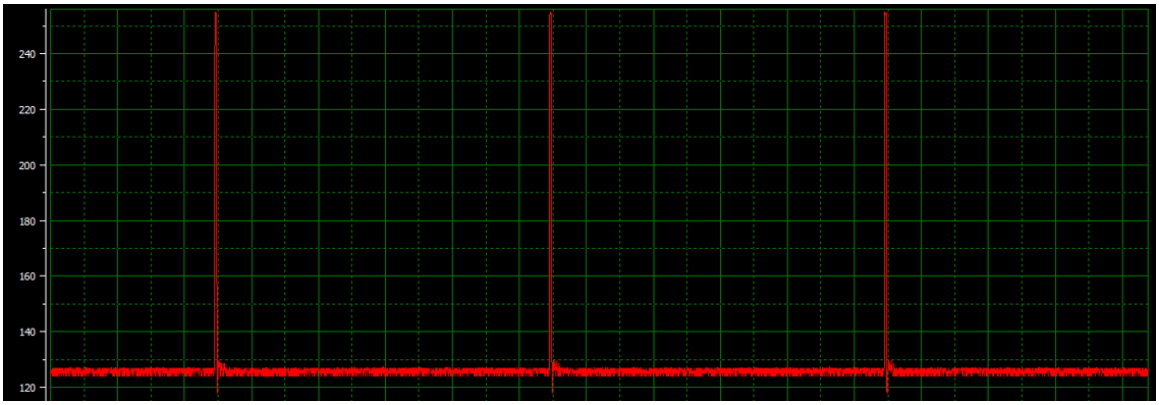

Figure 5.5. Data captured from BRAM

6. TRIGGER IMPLEMENTATION

6.1. OVERVIEW

This section deals with the implementation of a trigger logic on the Virtex-5 FPGA to capture the data samples on the arrival of a trigger signal. As discussed in the previous section, the BRAM of the Virtex-5 FPGA is capable of storing only limited data samples and this has to be used towards storing the ESD samples. The trigger mechanism for the ESD event is handled by the trigger boards which will trigger a TTL voltage on the arrival of an ESD event. The TTL voltage from the trigger board is sent to the Virtex-5 FPGA while the ESD event is sent to the LM97600 ADC which will sample the data and send the data to the Virtex-5 FPGA via the high-speed LVDS lines and received by the GTX Transceivers of the FPGA.

The implementation of the trigger logic involves the storing of the ESD event on the arrival of the trigger signal from the trigger boards. The 4 trigger boards each have a trigger output which is connected to the I/O pins of the Virtex-5 FPGA. The challenge lies in utilizing the limited internal memory of the FPGA to store the ESD events which can be acquired to the PC for additional processing and to determine the location of the ESD event based on the arrival times of the signals from each of the four channels of the ADC.

The FPGA will have to store the data from the ADC into the BRAM from all four channels of the ADC regardless of which trigger signal arrives first. The logic schemes for the implementation of the trigger are discussed in this section as well as their outcomes and potential results are seen.

6.2. TRIGGER SIGNALS

The trigger boards have the following signals interfaced with the Virtex-5 FPGA and their names as well as their descriptions are listed in Table 6.1. The TIG*_OUT and TIG*_RES are the only signals that are currently controlled by the FPGA while the other nets are controlled manually on the trigger boards.

Table 6.1. Trigger Signals to the FPGA

Signal Name	**Input/Output**	**Description**
TIG*_OUT	Input	Trigger Signal to the FPGA
TIG*_RES	Output	Reset Signal for Trigger
TIG*_DB[7-0]	Output	Signals to DAC for Hysterisis
TIG*_A0	Output	DAC Select Signal
TIG*_C1	Output	Signal to AND Gate
TIG*_C0	Output	Signal to AND Gate

6.3. SIMPLE TRIGGER

There are various methods of implementing a trigger scheme and the simplest method of implementing a trigger is to store the data samples from the LM97600 ADC into the BRAM on the arrival of the trigger signal. This will result in the FPGA storing the data samples arriving immediately once the trigger signal arrives at the I/O of the Virtex-5 FPGA.

The trigger signal arriving from the trigger board is logic high, when an ESD event occurs and the trigger board determines that the signal is an ESD signal. The same ESD is signal is sent to both the trigger board as well as the input channels of the LM97600 ADC using a splitter. The trigger signal is latched and remains high as long as it is reset. The trigger reset can either be done manually or through a signal

from the FPGA.

6.3.1. Implementation. The storage of the data samples from the LM97600 into the BRAM of the Virtex-5 FPGA has already been discussed before and the simple trigger works on the mechanism of storing the data samples from the ADC as long as the trigger signal is high and it stops storing the data samples once the trigger signal goes low.

Since it is difficult to manually reset the trigger signal from the trigger board, the trigger would also have to be reset by a signal from the FPGA. ESD events are signals with a very small rise time as well as a very small pulse not spanning more than a few nanoseconds. So it would be seem an unwise method to keep storing the data samples after more than a few nanoseconds. The implementation scheme is shown in Figure 6.1

In order to limit the amount of time the samples are captured after a trigger occurs, an approximate time interval of 200 ns is selected so that the ESD event is not missed and can be completely accommodated within the trigger time. This is implemented using a simple down-counter which counts down for 200ns. The TIG_RES is also asserted high at the end of the down counter, resetting the trigger event and making the TIG_OUT low.

6.3.2. Address Storage. There can be numerous ESD events that occur until the internal memory of the Virtex-5 FPGA is completely filled. It is important to know the location at which the trigger event occurred so that it is easy to determine the starting point on the trigger based on which the location of the event can be estimated. In order to overcome this obstacle, the address of the location at which the trigger event occurred has to be stored.

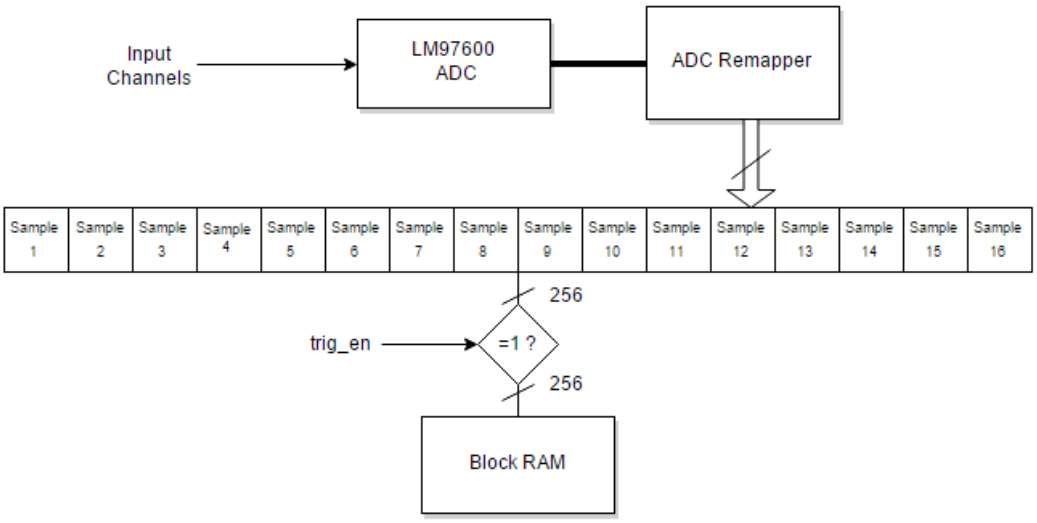

Figure 6.1. Simple Trigger Implementation

A simple solution to storing the address of the beginning of the trigger events, is to use a FIFO which was generated using the Xilinx CORE Generator©as shown in Figure 6.2.

6.3.3. Drawbacks. The simple trigger is the easiest solution to implement since it requires storing the data samples only after the arrival of the trigger signal and the data before the trigger signal is not of vital importance so it is lost. However, as will be discovered later, the obvious drawback of this method of storing the data samples is that the trigger signal and the ESD signal from the ADC have to arrive at the FPGA I/Os at the same time or the ESD Signal to the ADC should follow the trigger signal especially since the signal is only a few nanoseconds.

If the ESD signal arrives before the trigger signal, then the ESD signal is lost and the ESD event is not captured by the internal memory of the FPGA. This is potentially a huge drawback and the method can only be used if either both the signals arrive at the same time or the ESD signal arrives just after the ESD signal.

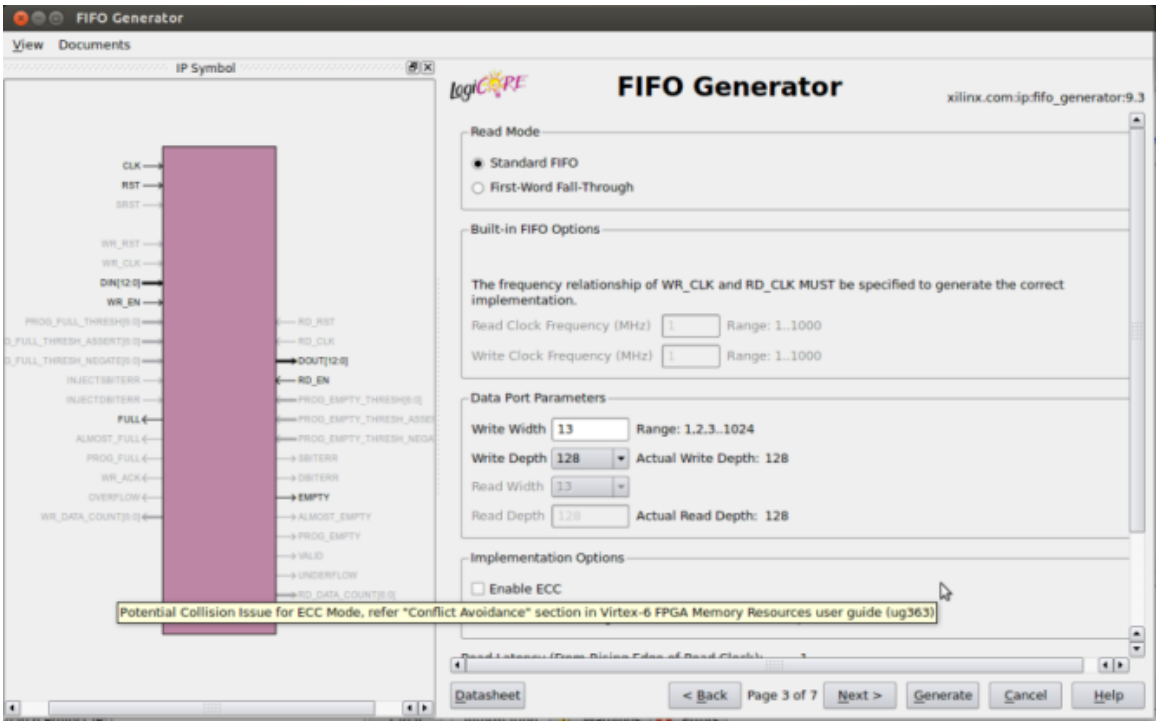

Figure 6.2. FIFO Generator

If there is a delay determined between the ESD signal and the trigger signal, with the samples of the ESD signal arriving first at the FPGA I/O, then it must be rectified by ensuring a delay is inserted manually on the ESD signal to delay it so that it follows the trigger signal.

6.4. PRETRIGGER I

An alternative method to the simple trigger logic had to be studied in order to ensure that the samples of the ESD signal aren't lost if the ESD signal arrives before the trigger signal. This would require that the data samples arriving before the trigger signals are not discarded due to the potential issue of discarding the ESD event itself.

The trigger logic would have to be designed and implemented in a way that ensures that data samples are stored for a certain period of time before they are discarded. There are two ways in which this can be implemented and this section talks about the first method of storing the data samples before the trigger signal, which can also be referred to as the pretrigger data.

6.4.1. Design. The first method of designing the pretrigger logic involves the use of a BRAM block which is split into small cyclical buffers holding no more than 1K samples within them which is sufficient for an ESD signal. The sample data coming in from the ADC keeps getting stored into the cyclical buffer which is 64 locations deep. The data samples keep being recycled until a trigger signal occurs which causes the location at which the current data sample is being stored, to be recorded. This is indicated in the Figure 6.3

The objective is to save 50% of the data before the trigger occurred so we have an equal length of pretrigger data. The data samples arriving after the trigger signal are only stored for 50% longer before the trigger signal is reset by asserting the TIG_RES is asserted causing the TIG_OUT to be deasserted as well.

The first buffer now holds the first ESD event and the second buffer is now used to store the data samples until the next ESD event occurs. Subsequently, when the entire BRAM is filled up, it now consists of samples of 128 ESD events which include the pretrigger data as well as the post-trigger data.

However, it should be noted that the pretrigger time is still not huge and the pretrigger data will store only 100ns prior to the trigger signal arriving. So, if the delay between the trigger signal and the ESD signal is greater than 100 ns, the event will again fail to be captured.

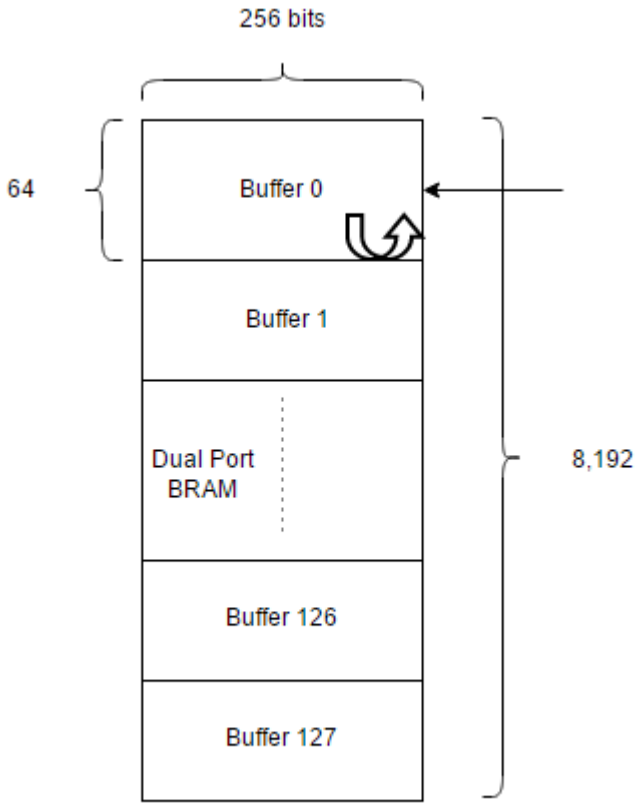

Figure 6.3. Pre Trigger I Design

6.4.2. Implementation. The implementation of this trigger logic required the use of a Simple Dual Port RAM with one port required to read the data samples and the other port required to write the data samples to the BRAM. The design had to be first simulated before it could be implemented into the hardware code.

The dual port ram would still be the same size which is 256 bits wide and 8,192 locations deep. However, since a dual port ram is being considered for the application, there would be two address bus' required two address generators to be implemented in the design as shown in Figure 6.4. The read address generator is a 13-bit address generated using two 8-bit up counters. The second address generator is important since the BRAM is being handled as 128 small buffers. The address

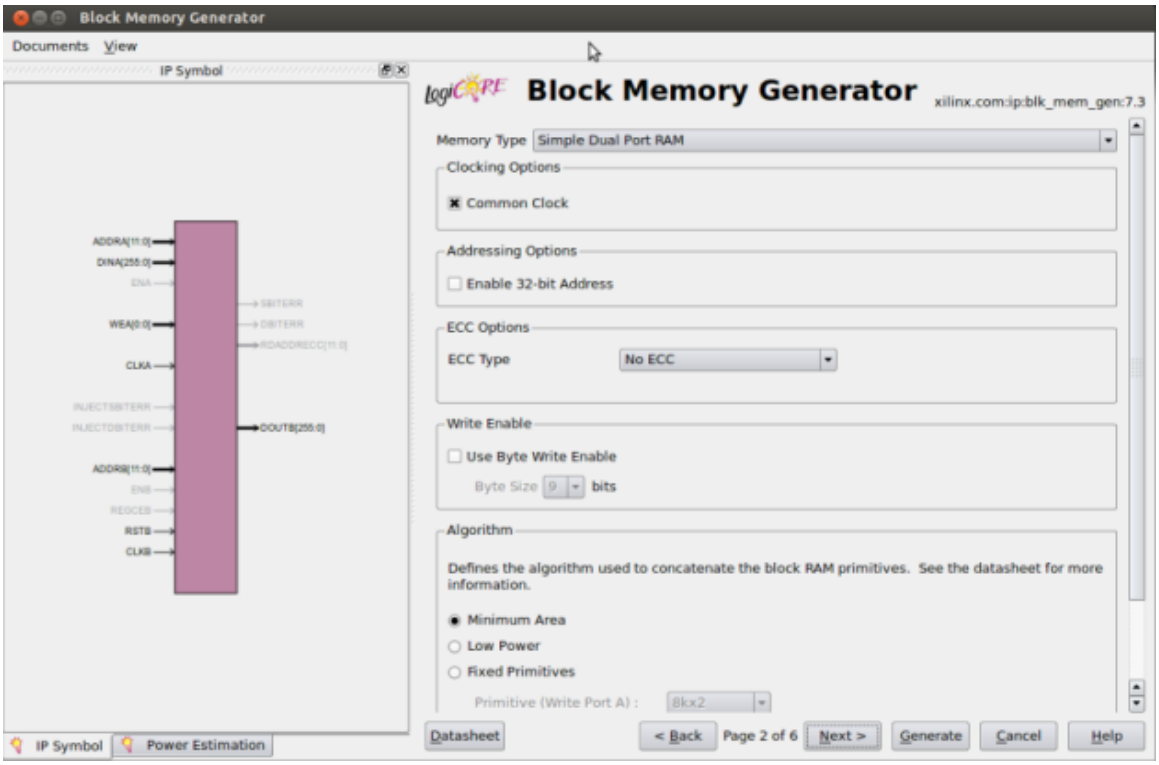

Figure 6.4. Simple Dual Port RAM

although 13-bits as well, the lower 6 bits are used to cycle through the buffer while the upper 7 bits are used to cycle through the buffers themselves.

The lower 6-bits are a normal up counter and count up on the positive edge of the clock. The higher 7-bits which control the buffer is also an up counter, but it counts up on the negative edge of the trigger signal so that the buffer element changes after each trigger signal so that the samples of the previous ESD event are not overwritten.

A 5-bit down counter is started every time a trigger signal occurs so that the samples in the previous 32 data locations of the buffer are preserved and the buffer stores the next 32 data locations with incoming data samples.

6.4.3. Address Storage. Each time the trigger signal occurs, the address of the location at which the trigger occurred is stored in a FIFO so that the starting address of the trigger event can be determined. The location of the starting point of the trigger signal is given by concatenating the lower 6-bits which were used for cycling through the buffer and the upper 7-bits are formed by the counter handling the buffer elements.

It must be ensured that the address of the Trigger location is at the first rising edge of the clock or the location will be distorted and the address stored will be incorrect.

6.4.4. Drawbacks. This solution is a better option than the simple trigger since it accounts for any ESD signal that may have arrived just before the trigger. However, it does have its complexities in the way the data samples are stored and the starting address of the trigger event. There needs to be a way devised to retrieve the addresses of the starting locations of the ESD events from the FIFO and this has not yet been accomplished.

Additional post-processing would be required for the data samples once they have been retrieved from the FPGA since they would need to be re-arranged based on the location the trigger occurred.

6.5. PRETRIGGER II

An additional alternative to the pretrigger was designed, simulated and implemented which is similar to the previously discussed pretrigger logic. However, the implementation differs in how the logic is implemented. The drawbacks of the previous pretrigger logic included the task of rearranging the data samples based on the

address locations once the data was acquired on the PC. This would involve reading both the FIFO as well as using the address locations to determine the starting location and rearranging the data samples based on it.

The pretrigger logic in this method of implementation has also been designed to store the same amount of pretrigger data as the previous design.

6.5.1. Design. This method of designing the pretrigger logic involves the use of two BRAM blocks rather than one as seen in the previous design. Since the memory available within the FPGA is limited, the memory that was allocated to one BRAM block has to be divided into two BRAM blocks.

The two BRAM blocks serve separate purposes where one BRAM block is used to store all incoming data from the ADC samples similar to a large cyclic buffer. The second BRAM will be responsible for holding samples of ESD events based on the number of trigger events that occur. This design allows the storage of larger pretrigger data since the cyclic buffer is larger when compared to the previous design. This is shown in the Figure fig:Design of PreTrigger II.

When a trigger event occurs, the address of the first BRAM block at the instance of the trigger is offset by a certain value based on the delay between the trigger signal and the ESD signal. The block of data from the offset address is transferred to the second BRAM block depending on the number of samples that are to be considered for the ESD event.

6.5.2. Implementation. Two BRAM blocks have to be used in this design so the BRAM blocks are divided into smaller sizes of 256 x 4196 each. In this case, both the BRAM blocks are Dual Port RAMs. The addresses are 12-bit addresses for both the BRAM blocks. Since both the BRAMs are dual port RAMs, there need to be 4 address generators for the 4 address ports.

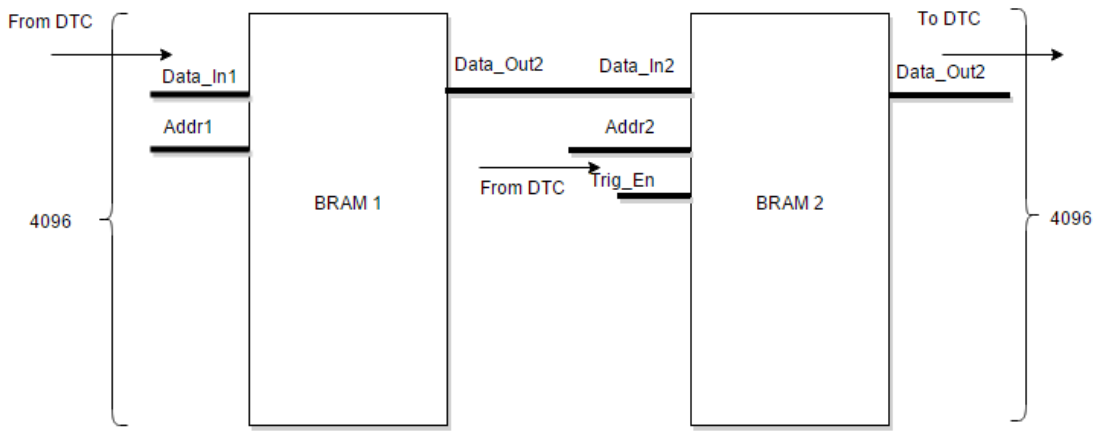

Figure 6.5. Design of PreTrigger II

The first address port of the first BRAM is 12 bits and the address is generated by using two 8-bit up counters which count up on the rising edge of the clock cycle. This is the address for the write port and the data is constantly written to the first BRAM. The second address port of the first BRAM is used to read the data from it. Since the design hinges on the transfer of data between the two BRAMs, the second address port of the first BRAM is used to load the offset address of the location from which the data starts to be transferred to the second BRAM.

The second BRAM also being a dual port RAM also requires two address generators. The first address is a 12-bit address generated by two 8-bit up counters for writing the data samples of the ESD event to the second BRAM.

When a trigger signal occurs, the write address of the first BRAM is offset by a certain value based on the pretrigger data required and then the second address is loaded with this offset value. A down counter is started and the transfer of the data samples from the first BRAM to the second BRAM is initiated until the down counter is completely 0.

In this way, the ESD event samples are transferred from the first BRAM to the second BRAM whenever a trigger signal occurs.

6.5.3. Drawbacks. This solution is a slightly more complex compared to the pretrigger logic discussed before this and also requires the addresses to be offset and data transfer between two BRAMs. In addition, the number of ESD events that can be stored, is cut down by 50% due to the internal memory being split into two BRAMs. However, the main advantage of this method is that, the data samples would not have to be rearranged once they've been transferred to the PC requiring them only to be split up into different ESD events and requiring lesser post-processing.

6.6. RESULTS

6.6.1. Simulations. The simulations were run for the trigger methods and the simulations were successfully executed to give the desired results. The Figure 6.6 shows the results of simulating the simple trigger technique which starts a down counter on the arrival of the trigger and stores the address of the BRAM at that particular location. The Figure 6.7 shows the results of the PreTrigger I scheme which use small cyclical buffers to store the ESD events. As in the previous case, a down counter is started on the arrival of a trigger for 100ns and when the down counter is completed, it moves on to the second buffer.

The Figures 6.8, 6.9 and **??** show the simulation results of the PreTrigger II technique which uses two BRAMs for the acquisition of data samples. The first simulation waveform shows the initial results once the trigger occurs while the second and third figures shows the waveforms when the down counter finishes counting down to zero.

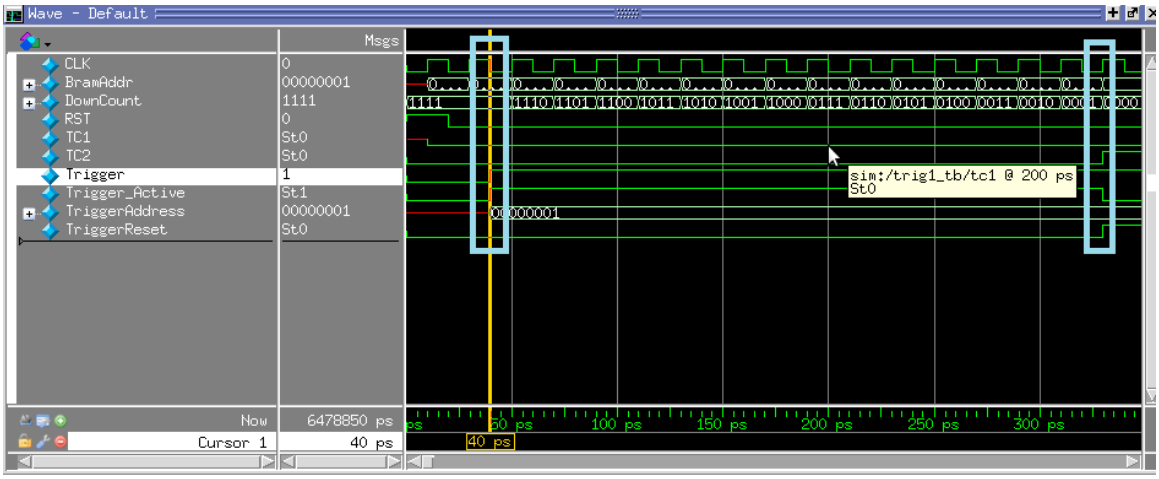

Figure 6.6. Simple Trigger Simulation

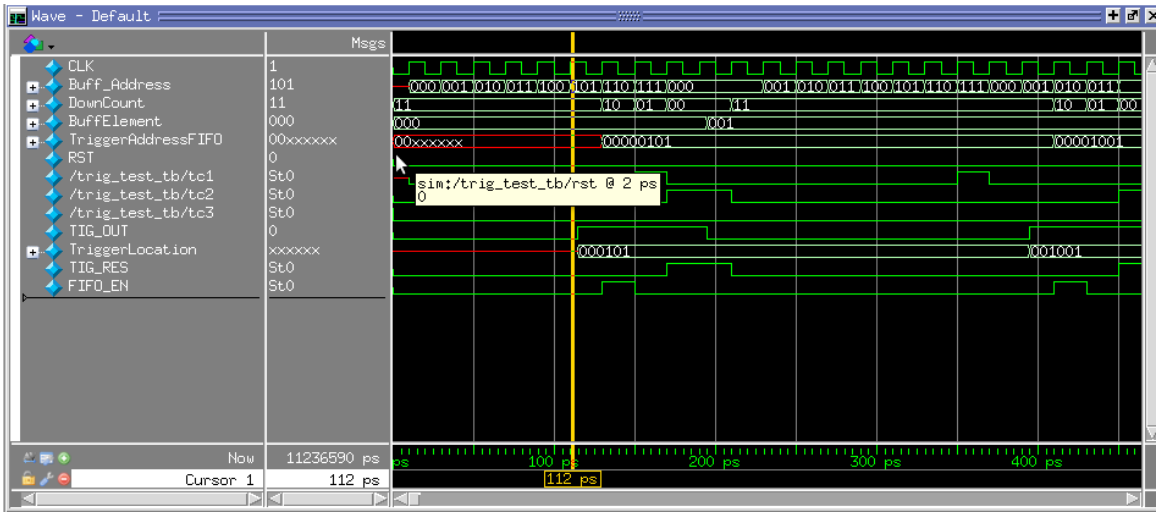

Figure 6.7. PreTrigger I Simulation

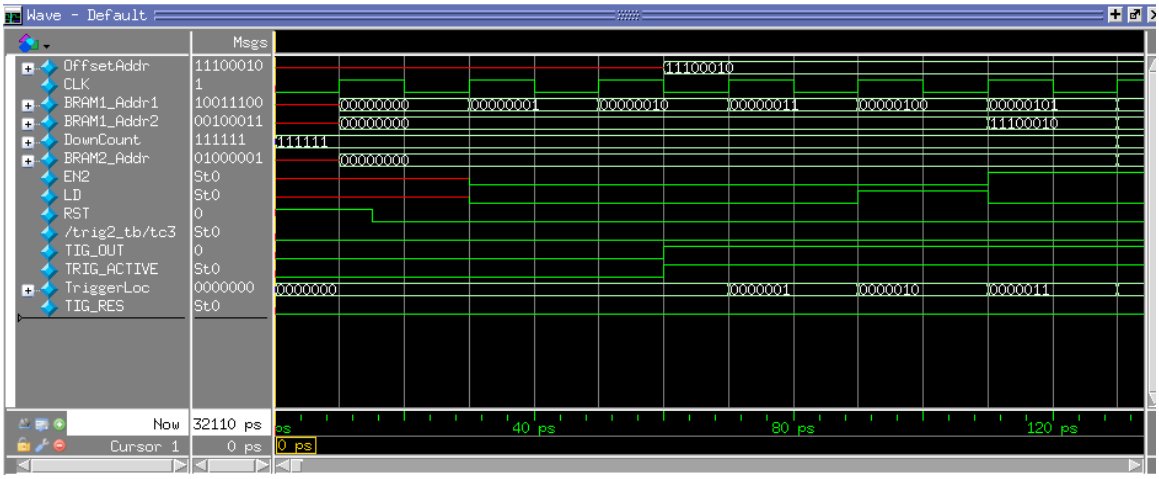

Figure 6.8. PreTrigger II Simulation 1

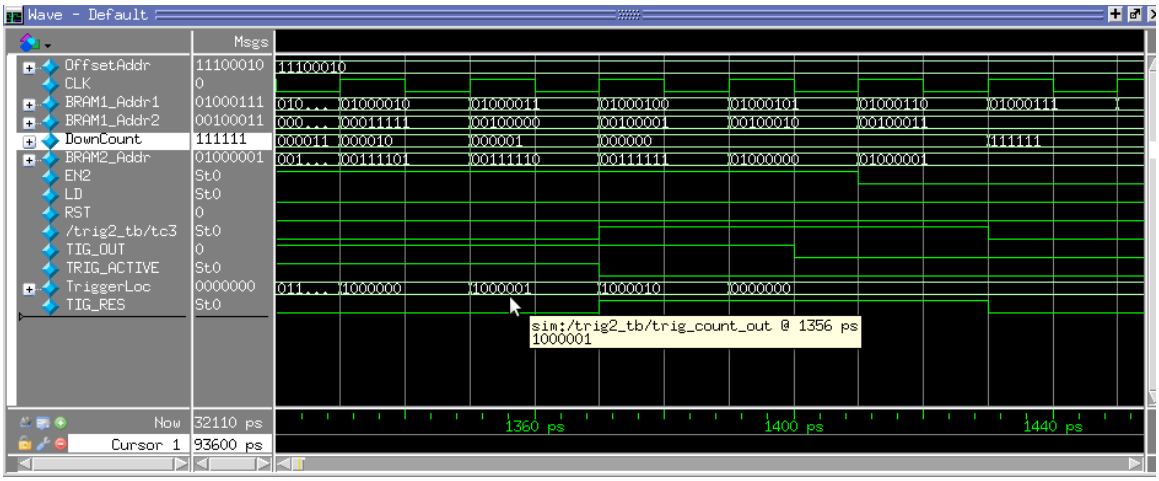

Figure 6.9. PreTrigger II Simulation 2

BIBLIOGRAPHY

[1]Lin, Don L., DeChiaro, Louis F., Jon, Min-Chung., "A Robust ESD Event Locator System With Event Characterization," *33rd Electrical Overstress/Electrostatic Discharge Symposium*, (1997).

[2] Thongpull, K.; Jindapetch, N. ; Teerapabkajorndet, W., "Wireless ESD Event Locator Systems in Hard Disk Drive Manufacturing Environments," *IEEE Transactions on Industrial Electronics*, Vol.60, No.11, pp.5252-5259, (2012).

[3] Xilinx, 2009, "Virtex-5 FPGA RocketIO GTX Transceiver", Xilinx Website, http://www.xilinx.com/ .

[4] Xilinx, 2012, "LogiCORE IP Block Memory Generator v7.3," , Xilinx Website, http://www.xilinx.com/ .

[5] National Semiconductor, 2009. "Programmers' Guide to WV5_DLL API" .

[6] Maheshwari, P.; Pillai, V.; Shah. R.; Eldredge. C.; Akella. R.; "Design of ESD Event Locator Hardware", *EE456, Signal Integrity, High Speed Digital and RF Design Laboratory*, Missouri S&T

VITA

Syed Abrar Ul Huq earned his Bachelors degree in Electrical Engineering from KNS Institute of Technology, India in 2011. He has been a graduate student in the Computer Engineering Department at Missouri University of Science and Technology since August 2012 and worked as a Graduate Research assistant under Dr. David Pommerenke from October 2013 to May 2015. He completed his Masters in Computer Engineering at Missouri University of Science and Technology in May 2015.