Development of FPGA based Error Resilient Self-Triggered Readout Chain for Muon Chamber (MUCH) Detector of CBM Experiment

By
Swagata Mandal
ENGG04201304002

Variable Energy Cyclotron Center

A thesis submitted to the
Board of Studies in Engineering Sciences
In partial fulfillment of requirements
for the Degree of
DOCTOR OF PHILOSOPHY
of
HOMI BHABHA NATIONAL INSTITUTE

October, 2017

Version approved during the meeting of Standing Committee of Deans held during 29-30 Nov 2013

STATEMENT BY AUTHOR

This dissertation has been submitted in partial fulfillment of requirements for an advanced degree at Homi Bhabha National Institute (HBNI) and is deposited in the Library to be made available to borrowers under rules of the HBNI.

Brief quotations from this dissertation are allowable without special permission, provided that accurate acknowledgement of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the Competent Authority of HBNI when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

Swagata Mandal

DECLARATION

I, hereby declare that the investigation presented in the thesis has been carried out by me. The work is original and has not been submitted earlier as a whole or in part for a degree / diploma at this or any other Institution / University.

Swagata Mandal

List of Publications arising from the thesis

Journal

1. **Swagata Mandal**, Rourab Paul, Suman Sau, Amlan Chakrabarti, Subhasis Chattopadhyay "A novel method for soft error mitigation in FPGA using Modified Matrix code" IEEE Embedded system letter, vol:8, pp: 65-68, issue: 8, August 2016 (DOI:10.1109/LES.2016.2603918)

2. **Swagata Mandal**, RourabPaul, SumanSau, AmlanChakrabarti, SubhasisChattopadhyay "Efficient dynamic priority based soft error mitigation techniques for configuration memory of FPGA hardware" Microprocessor and Microsystem, vol: 51, 2016. (https://doi.org/10.1016/j.micpro.2016.12.003)

3. **Swagata Mandal**, Jogender Saini, Wojciech M. Zabołotny, Suman Sau, Amlan Chakrabarti, Subhasis Chattopadhyay" An FPGA-Based High-Speed Error Resilient Data Aggregation and Control for High Energy Physics Experiment", IEEE Transaction on Nuclear Science, Vol: 64, Issue:3, 2017, DOI: 10.1109/TNS.2017.2656464.

4. Jogender saini, **Swagata Mandal**, Amlan Chakrabarti and Subhasis Chattopadhyay, "A real time sorting algorithm to time sort any deterministic time disordered data streams", Journal of Instrumentation (JINST), IOP Science, Vol:12, 2017

Conferences

1. **Swagata Mandal**, Suman Sau, Amlan Chakrabarti, Jogendra Saini, Sushanta Kumar Pal, Subhasish Chattopadhyay "FPGA based Novel High Speed DAQ System Design with Error Correction", In Proceedings of IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2015, Montpellier, France.

2. **Swagata Mandal**, Suman Sau, Amlan Chakrabarti, Susanta Kumar Pal, Subhasish Chattopadhyay," FPGA Implementation of High Speed Latency Optimized Optilal Communication System Based on Orthogonal Concatenated Code", In Proceedings of 24th Asian Test Symposium (ATS), 2015, IIT Bombay, India.

3. **Swagata Mandal**, Jogender Saini, Suman Sau, Amlan Chakrabarti, Wojciech Zabolotny, Subhasis Chattopadhyay, W. F. J. Muller," Integration of GBTx Emulator with XYTER and Data Procissing

Board (DPB) for CBM Experiment", In proceedings of IEEE Nuclear Science Symposium, 2016, Strasbourgh, France.

4. **Swagata Mandal**, Wojciech Zabolotny, Suman Sau, Amlan Chakrabarti, Jogender Saini, Subhasis Chattopadhyay, Sushanta Kumar Pal, " Internal monitoring of GBTx emulator using IPbus for CBM Experiment", In Proceedings of XXXVI-th IEEE- SPIE Joint Symposium Wilga2015, Poland.

5. Suman Sau, **Swagata Mandal**, Jogender Saini, Amlan Chakrabarti, Subhasis Chattopadhyay ," High Speed Fault Tolerant Secure Communication for MUCH using FPGA based GBTx Emulator", In Proceedings of 21st International Conference on Computing in High Energy and Nuclear Physics (CHEP 2015)' April 13-17 ,2015.

6. Swagata Mandal et.al," FPGA Emulator of GBTx for Muon Chamber (MUCH) in CBM Experiment", In Proceedings ofDAE Symposium on Nuclear Physics (snp2014), December 08-12, 2014. Vanarasi. Benaras Hindu University'

7. Swagata Mandal et.al," Electronic Data Aggregation Architecture for High Energy Physlcs Big data taking Experiments", In Proceedings of DAE- BRNS Symposium on Nuclear Physics (snp2016)

Swagata Mandal

DEDICATIONS

I would like to dedicate this thesis to my loving parents, colleagues and friends ...

ACKNOWLEDGEMENTS

I have been dedicated during the course of my research work and tried to give an honest effort. However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them. I am highly indebted to my supervisors Prof. Subhasis Chattopadhyay, Prof. Amlan Chakrabarti, Prof. Tapan Kumar Nayak and Shri. Singaraju Ramanarayan for their guidance and continuous support as well as for providing necessary information regarding the project \& also for their imperial hand holding in completing the project. I would like to express my gratitude towards my mother Smt. Mita Mandal, my father Shri Swapan Mandal. Their cooperation and encouragement help me to complete this project successfully. I would like to express my special gratitude and thanks to all the faculty members of VECC and my lab colleagues S/Shri Jubin Mitra, Jogender Saini, Partha Bhaskar, Shuaib Ahmad Khan and Vinod Singh Negi for giving me such attention and time.

Abstract

The Compressed Baryonic Matter (CBM) experiment is one of the most important experiments at the future Facility for Antiproton and Ion Research (FAIR) in Darmstadt, Germany which is designed to detect rare particles generated during high energy and high density nucleus-nucleus collisions. Gas electron multiplier (GEM) detectors based Muon Chambers (MUCH) are used to detect muons generated during the collisions in the CBM experiment. This thesis mainly focuses on the development of Field programmable Gate Array (FPGA) based self-triggered error resilient readout chain to capture MUCH detector output that will be used during the data analysis and online computing. FPGA is used as the target device due to some of its advantages like on field programmability, inherent parallelism, runtime partial reconfigurability and easy time to market compared to application specific integrated circuits (ASICs). Front end electronics board (FEBs) in readout chain capture data directly from the detector and send it to the computing nodes using Gigabit Transceiver (GBTx) board, data processing boards (DPB) and First level interface board (FLIB) though different high speed interconnects like optical fibre and peripheral component interconnect express (PCIe). As a part of the readout chain we have developed an FPGA prototype of a high speed error resilient communication proto col termed as GBTx Protocol, which is used to transfer data from harsh radiation zone to moderate radiation zone. Though, the ASIC version of the GBTx hardware has already been developed at *CERN*, our work is a first of its kind FPGA implementation of the same. Proper steps have been taken to make the high speed communication robust in the radiation with stabilized latency using orthogonal concatenated code and delay and phase alignment circuit. As the FEBs work in the self-triggered mode and there i s no trigger information in the data stream we have developed an online memory management algorithm to remove the deterministic time disorder in the online data stream that helps further data analysis in the computing no des. In the first phase of our research we have developed an FPGA prototype of the readout chain comprising of FEBs, GBTx emulator, DPB and the related integration. FPGA devices are vulnerable to charge particles and they may cause soft errors in the configuration memory. In order to mitigate single and multi-bit upsets due to soft errors in the configuration memory of FPGA a novel parity based error correcting code (ECC) named as modified matrix code (MMC) and an erasure code known as EVENODD code have been used. Using frame interleaving and selective bit placement along with Hamming pro duct co de (HPC) a multi-bit adjacent ECC is proposed that enhances error correcting capability of HPC without increasing redundancy. At the same time we have proposed dynamic partial re configuration (DPR) along with a simple hardware scheduling algorithm based download manager that helps to perform the error correction in the configuration memory without suspending the operations of other hardware blocks. In general, flash memory is used to store data bits for future usage and now a day to cope up with the huge data volume designers have started to use multi-level memory cells where one memory cell can store multiple data bits instead of single bit. With the increase of memory density probability of adjacent bits being affected by radiation increases and it leads to the formation of clustered error. As traditional multi-bit ECCs are inefficient against clustered error we have proposed a novel clustered ECC using shortened product codes with simple linear block code as component codes. In a nutshell, this dissertation contributes to the state of the art research in the domain of soft error mitigation of embedded systems under radiation environment through the various new methodologies and their system level implementations that have been achieved in the due course of this research.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

To solve the long standing puzzles of the universe that haunt the mankind for thousand of years, scientists have been trying to widen their knowledge specifically in two extreme directions: macroscopic and microscopic. For the macroscopic physics, scientists have developed huge telescopes like hubble telescope, ARIES Telescope to take the image of the universe, supernova and galaxies. They have also prepared spacecrafts like Juno [1] (developed by NASA) to gather information about jupiter, Mangalyaan [2] (developed by Indian Space Research Organization) to study the atmosphere of the Mars. At the same time, to study the matter at the subatomic level, scientists have developed high energy accelerators like Large Hadron Collider (LHC [3]), Standford Linear Collider (SLC [4]), Universal Linear Accelerator (UNILAC) [5]. One of the major objectives of the ongoing accelerator based experiments is to create initial state of the Bing-Bang or a matter limited to the inner region of the dense neutron star in the laboratory. Apart from the study of matter at the subatomic level, accelerators are now a days being used in other domains like structural biology, radio therapy etc. Though the microscopic and the macroscopic approaches are totally different, they are interrelated and influence each other. Giant machines, robust high speed data transmission networks and complex signal processing algorithms are integral parts in both of the approaches. Different modern electronics devices like FPGA, ASICs, Microcontrollers are used to develop such giant machines that help to implement complex data processing algorithms. This thesis work has been mainly carried out in the context of development of the readout system of one

such experiment, the *Compressed Baryonic Matter* experiment at the FAIR [6] complex in GSI, Germany.

1.1 Compressed Baryonic Matter Experiment

FAIR in GSI provides unique research opportunities in the fields of nuclear, hadron, atomic, plasma physics and computational biology etc. There are mainly four researches are going on in the FAIR complex [6]: Atomic, Plasma Physics and Applications (APPA), Compressed Baryonic Matter (CBM), Nuclear Structure, Astrophysics and Reactions (NUSTAR) and Anti-proton Annihilation at Darmstadt (PANDA). The CBM experiment [7] which is under development at the FAIR complex is one of the major fixed target experiments, and the objective of this experiment is the exploration of quantum chromodynamics (QCD), the theory of strongly interacting matter at moderate temperatures and high baryon (Baryon is a subatomic particle made up of three quarks) density [8]. Phase diagram depicts the existence of different states of the matter (like solid, liquid and gas) at different external thermodynamic conditions like temperature and pressure. From the particle physics we know that quark is the fundamental particle that form neutron and proton within the atom. Gluon helps to bind the quarks within the neutron and proton. At a normal temperature and pressure these quarks and gluons can not be separated. At extreme external conditions (like at high temperature or at high net baryon density) these quarks and gluons can be separated and they exist in a plasma like state which is known as quark gluon plasma (QGP) [9]. This state was said to be present at the time of Bing-Bang [10] or at the initial stage of the universe. In order to create high baryon density and high temperature in the laboratory, heavy ions with high ionizing energy are collided in an accelerator that provide different subatomic particles. Different detector systems are placed around the point of collision to detect the generated subatomic particles that helps in the study of the QCD phase diagram.

Based on the arrangement of the collision within the accelerator there are two types of experiments: fixed target experiment and collider experiment. In the fixed target experiment, heavy ion beam within the accelerator hits a stationary target as shown in Figure 1.1(a). Such stationary target may be a chunk of

metal or liquid hydrogen in a large container. Number of collisions in the fixed target experiment can be increased by increasing the thickness of the target. Placement of detector system is quite easier in such experiment. On the other hand in collider type experiment two beams are directed to each other and made to collide within the accelerator like in LHC [11] at $CERN$. As the beams are very narrow they need very precise control and steering mechanism to guarantee that two beams rotating in opposite directions will collide with each other as shown in Figure 1.1(b). Though placement of detector around the point of collision is complicated in collider type experiment it provides high energy at the point of collision. As CBM studies the QCD phase diagram at high baryon density at low temperature it uses fixed target experiment where a range of target and beam are used in the range of proton to uranium. CBM experiment aims to find the phase

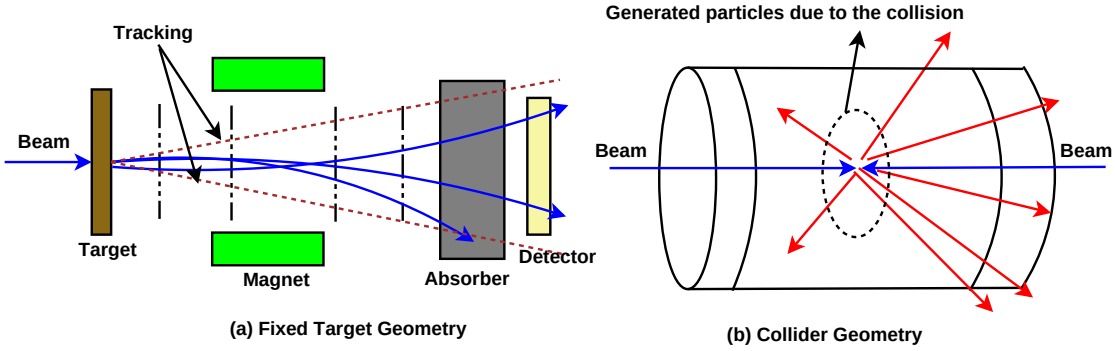

Figure 1.1: Fixed Target Experiment vs Collider Experiment

transition in QCD experimentally where QGP exists. The main challenges of the CBM experiment is to measure these particles with high precision and statistics. Hence, different detector subsystems are required to identify the particles (like muon, mesons, electrons and photon etc.) generated in the CBM experiment. Arrangement of different detector systems that will be used in CBM experiment are shown in Figure 1.2 and functionalities of each detector subsystem is described in Table 1.1. The detectors will send the readout informations to the computer cluster. Hence, a robust readout system is very much necessary for each detector system. In this thesis, we have mainly focused on the development of readout chain of MUCH detector.

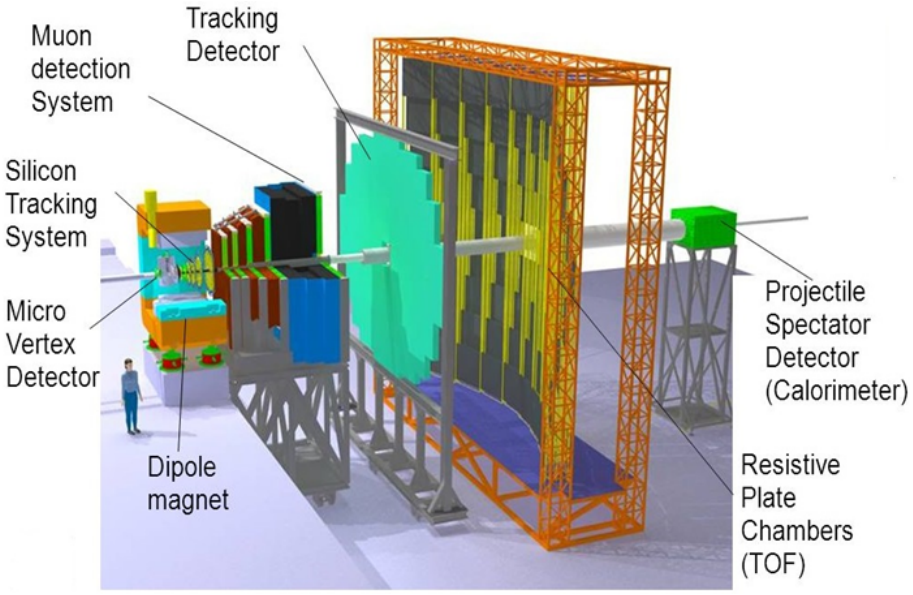

Figure 1.2: Detector system in CBM experiment [12]

Table 1.1: Function of detectors used in CBM Experiment [12]

Name of the Detector	Functionalities
Silicon Tracking System(STS)	Used for track reconstruction and determination of momentum of charged particles
Ring Imaging Cherenkov Detector (RICH)	Identification and suppression of pions with the momentum below 10 GeV/c
Muon Chamber System (MUCH)	Detect muons generated from J/Ψ particle and light vector mesons
Transition Radiation Detector (TRD)	Used to identify electrons and pions
Time of Flight (TOF)	Used to identify hadron through time of flight measurement
Calorimeter	Measure photon and neutral mesons

1.1.1 Muon Chamber (MUCH) Detector

MUCH in CBM experiment is mainly used to detect dimuons generated from charmonia (bound state of charms and anti-charms quarks) and light vector mesons [13] which are very rare particles generated during the collision. MUCH consists of segmented absorber layer (indicated by green color in Figure 1.3) made of iron or carbon and gas electron multiplier (GEM [14]) based detectors are placed in between two segments as shown in Figure 1.3. Each GEM detector consists of 50 μm thin copper claded kapton foil having 50-70 μm hole diameters

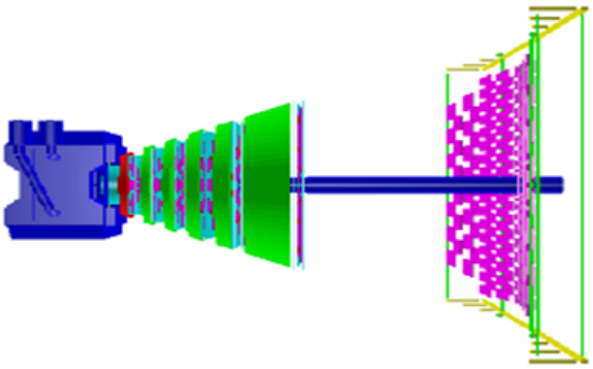

Figure 1.3: Schematic view of MUCH detector with segmented absorber [13]

at a pitch of 140 μm and two copper plates that act like cathode and anode. During the operation, 300-400 volt is applied across the surface of the foils that creates a high electric field inside the holes. The detector volume is filled with mixture of Argon and Carbon-di-oxide and when particles pass through the high field region inside the holes it ionizes the gas molecules and generates primary electrons. These primary electrons are accelerated through drift field and create more electrons through avalanche multiplications. In this way generated electron cloud is collected on the anode plane and produce the output signal. The gain of the GEM detector can be increased by adding multiple foils in between cathode and anode which is known as multi-layer GEM. Figure 1.4 shows architecture of single layer GEM detector. MUCH detectors will be placed in the downstream

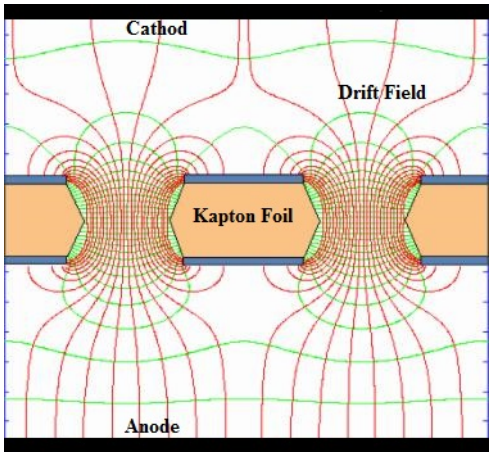

Figure 1.4: Basic structure of single layer GEM detector

of silicon tracking system (STS) [15] detector that measure momentum of the particles.

1.2 Architecture of Data Acquisition System

In order to obtain reasonable statistics for rare particles like J/ψ over a reasonable running period the interaction rate of colliding ions should be very high and for CBM experiment it will reach up to 10^7 events/second [16]. For handling such high reaction rate, CBM in general and MUCH detector in particulars uses a novel approach of data acquisition. MUCH detector in CBM experiment capture data using free running and self-triggered [17] (discussed later) FEBs. On the other hand in traditional accelerator based high energy physics (HEP) experiment, DAQ organizes the data read-out based on hierarchical trigger decisions. Different features [18] of the DAQ system used for MUCH detector are:

- Precise time synchronization.
- Fault resiliency.
- Ability to support high data rate and efficient data aggregation schemes.
- Self-triggered high-speed front-end electronics (FEEs) and compact hardware due to limited space.
- Ability to implement complex data processing algorithm.

In general, DAQ system (shown in Figure 1.5) consists of FEEs, data aggregation and control unit (DACU), a hierarchical DAQ networks and back-end computing cluster nodes. FEEs are responsible for acquisition of analog signal from detector

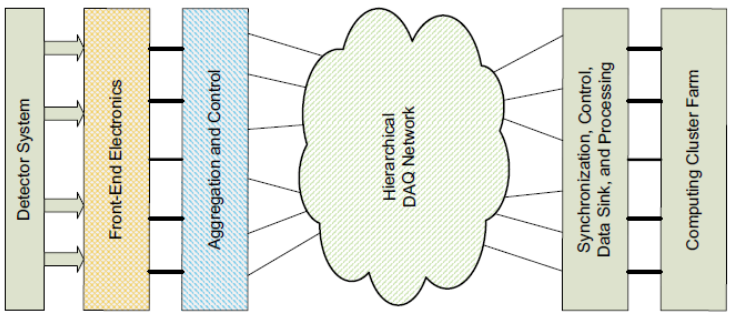

Figure 1.5: Traditional DAQ network architecture [18]

system and conversion of received analog signal to digital signal. DACU helps

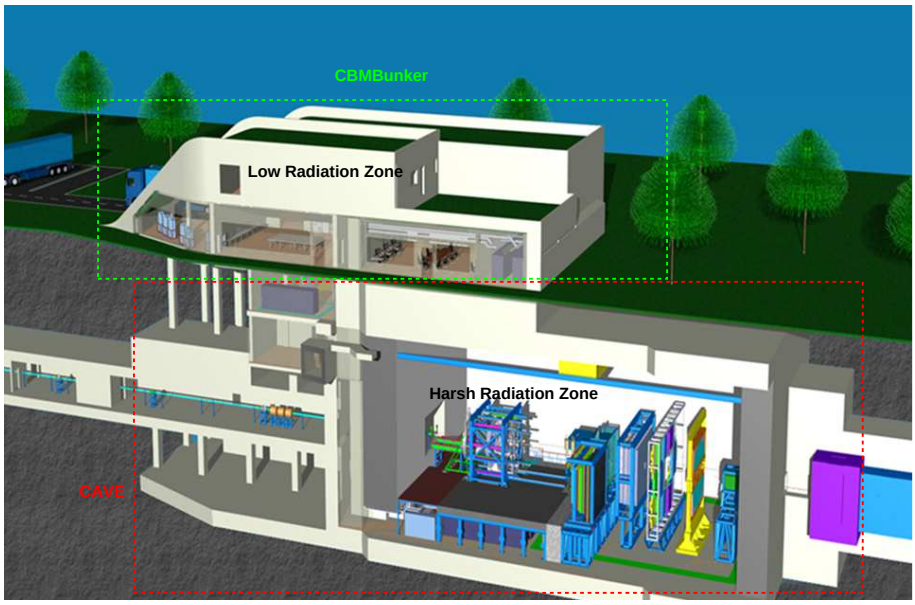

Figure 1.6: Planned Building and Cave Infrastructure for CBM Experiment [12]

in control and data synchronization. Hierarchical DAQ network connects the FEEs to the backend computing nodes where different reconstruction and decision making algorithms are used to analyze the received data by the detector system. In case of DAQ system for MUCH detector FEE, DACU and hierarchical DAQ network are under harsh radiation (*i.e* within the detector cave), and remaining portion of the DAQ chain is in the moderate radiation zone as shown in Figure 1.6.

1.2.1 Errors in DAQ system and its mitigation

Electronic devices used in the design of hardware systems for the data transmission network and data processing algorithm for such HEP experiment are often affected by radiation and channel noises, that may lead to errors in the system's outcome. Errors may occur in (i) communication link (ii) internal processing blocks and (iii) memory unit. Here, we have considered the errors occur in all three places. Most of the modern embedded electronic devices are developed using silicon wafers which are vulnerable to radiation. Faults occurred due to radiation in such solid state devices can be categorized into two broad domains: Temporary fault and permanent fault. Temporary faults lead to temporary malfunctions that occur in solid state devices and their effect is termed as soft errors.

Soft errors occur due to ionization of charged particles and may create latch-up and transient fault. X-ray radiation, photocurrent caused by ultraviolet and other low energy gamma ray are responsible for soft errors. Soft errors are not reproducible and sometimes lead to single bit upsets (SBU) and multiple bit upsets (MBU) in different embedded devices. Effect of soft errors into any logic will be transferred to the output of flip-flop or memory if its period of occurrence is higher than the period of the clock which drives the circuit. Permanent faults in solid state devices occur due to lattice dislocation by protons, α particles, heavy ions and high energy gamma rays. This permanently change the arrangement of atoms within the crystal lattice and leads to single event induced burnout and rupturing of gate of the MOSFET. Sometimes higher ionization dose for short time may partially anneal the damage due to lattice dislocation that may reduce the degree of damage. The permanent faults are of two types: one is permanent recoverable fault and other one is permanent nonrecoverable fault [19]. When incident charge particles permanently damage the logic blocks within the embedded devices, nonrecoverable faults occur and it can only be sorted out by replacing the defective logic blocks physically. Permanent recoverable can be mitigated either using built in error detection and correction (EDAC) code or by rewriting the memory of embedded devices and power on reset.

Methods to protect solid state devices from the effect of radiation can be broadly classified into categories: Physical and logical. Physical methods include:

- Use of different insulating substrate instead of traditional silicon wafer. Generally, silicon on insulator (SOI) or sapphire are used in these cases.
- Use Bipolar junction transistor or emitter couple logic based transistor instead of CMOS based transistor.
- Proper radio active shielding can be used to protect solid state devices from the effect of radiation.
- Magnetoresistive RAM (MRAM) and capacitor based DRAM are more robust compared to static RAM in the radiation zone.
- Use of wide band gap material like silicon carbide or gallium nitride as substrate make the device more robust in radiation environment.

On the other hand logical method includes majority voting, parity based error correcting methods, redundant logic, use of watchdog timer etc. Here we have mainly focused on logical methods using multi-bit ECCs to mitigate the error in the solid state devices occurred due to radiation because other logical methods either consume large memory or have higher latency. This issues justify the need of different low complexity EDAC algorithms to be incorporated within the electronics devices to minimize the errors causing due to SBUs and MBUs in the system. Designing of efficient error correction schemes in the hardware without hampering normal system operation should also consider the trade off between correction coverage and resource utilization. Attaining higher code coverage with low resource requirement is one of the key research issues in error correction mechanism.

As a choice of target platform, the embedded designers have three choices namely, (i) standard micro-controller or microprocessor platform (ii) custom hardware platform based ASICs and (iii) custom as well as re-configurable hardware platform based on FPGAs.

1.2.2 Microprocessor and Micro-controller

Micro-controller is mostly used for embedded system designer due to its flexibility, cost effectiveness and availability. Micro-controllers are single chip microcomputers, that includes processing unit, memory and I/O elements in the same chip. Though micro-controller based system provides easy solution they are normally used for specific application as they have limited resources and are driven by software instructions. Their performance is limited by the fact that they only provide generic processing core, which cannot satisfy the need of higher performance as compared to custom application specific cores. Some of the well known micro-controller platforms for embedded applications are 8051, Atmega etc. In CBM experiment, micro-controllers are used for detector control system, low voltage power distribution etc.

1.2.3 Application Specific Integrated Circuits

ASICs are mainly designed for particular application instead of general purpose usage. Designs implemented on ASICs cannot be changed once the chip is fabricated or manufactured. In the modern embedded application continuous upgradation is needed to cope up with the advancement of technology. In such case ASIC is not good solution. Moreover, with the inclusion of partial programming capabilities in modern programmable logic devices, the downtime of the system is further reduced as the hardware upgradation in the partially reconfigured region can work concurrently while, the other regions are on execution. Though the circuit implementation can be highly optimized in terms of speed, area and power consumption, inability to adapt design changes after manufacturing is a serious disadvantage for ASICs. Modern ASIC fabrication processes are also costly, time consuming and hence, they are used only for some critical applications like high speed signal processing in satellite, mobile devices, televisions etc.

1.2.4 Field Programmable Gate Array (FPGAs)

Hardware design is flexible using general purpose processors that means user can modify the design as per need by using high level programming languages without modifying the underlying processor architecture. General purpose processor does not bother about structure of the algorithm instead their focus is to support the processing need of a large variety of algorithms. Hence, in general, execution time and power consumption for a algorithm is high on general purpose processor. On the other hand ASICs provide less processing time and power consumption for an algorithm but flexibility is less. Reconfigurable devices like programmable logic array (PLA), programmable array logic (PAL), complex programmable logic devices (CPLD) and FPGA fills this gap and in an ideal case, combines the best of both; namely, the speed of ASICs to the flexibility of general purpose processors. Figure 1.7 provides trade-off between performance and flexibility for general purpose processor, ASICs and FPGA. Though there are different reconfigurable devices available in the market we are mainly focused on FPGA in this thesis. Xilinx, Altera, Actel are the leading vendors that manufacture FPGA. Main units inside the FPGA are reconfigurable functional units, programmable

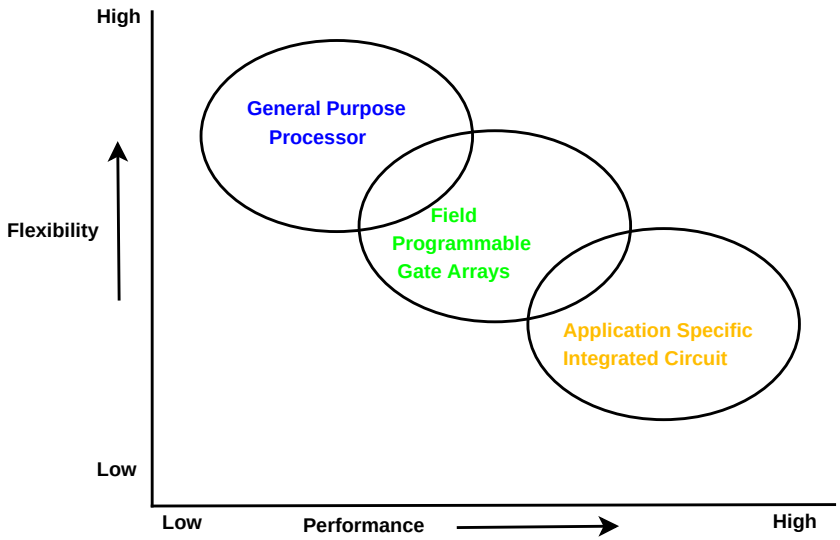

Figure 1.7: Performance-flexibility graph among ASIC, FPGA and general purpose processor

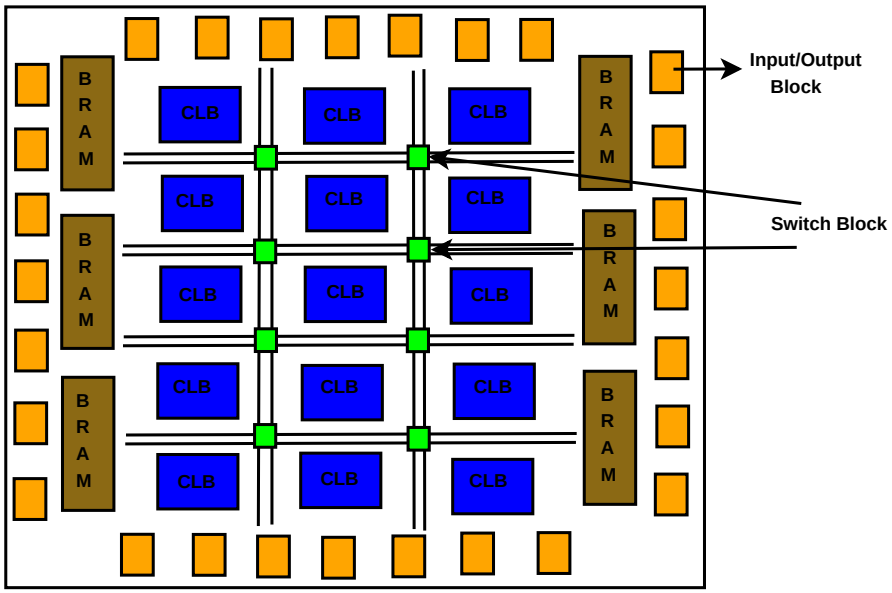

Figure 1.8: A Generic FPGA internal Architecture

switch matrix and I/O interfaces as shown in Figure 1.8.

During implementation of any logic in the FPGA, reconfigurable functional units or fabrics are used and hardware blocks are interconnected by the programmable switch matrices. Performance of the implemented logic on FPGA depends on the flexibility and efficiency of the reconfigurable fabrics. On the

other hand reconfigurability of FPGA devices are fully depends on size of the reconfigurable functional units or granularity. Based on the size of granularity FPGA devices can be classified into two broad categories [20]: fine-grained and coarse-grained. Look up tables (LUTs) are basic functional unit of fine-grained architecture and it is typically used to implement single function on small number of bits. On the other hand Arithmetic logic unit (ALU) is the basic unit of coarse grained architecture and hence, complex and large function can be implemented using this architecture. In general, modern FPGA like seven series FPGA of Xilinx, Arria and Stratix series FPGA of Altera follows fine grained architecture where reconfigurable functional units consist of 6-to-1-bit LUTs which are arranged in clusters. A collection of LUTs, storage element and multiplexers form configuration logic block (CLB). CLB of Xilinx Kintex-7 FPGA consists of two slices [21]: SLICEL and SLICEM. SLICEL is only used for logic operation. Along with logic operation SLICEM is also responsible for storing data using distributed RAM and shifting data with 32-bit registers. Each slice consists of four six input LUTs, eight flip-flops, multiplexers and arithmetic carry logic. The number of LUTs and flip-flops within the CLB varies depending on the different FPGA vendors. Modern FPGAs are integrated with hard processor core like PowerPC in Virtex, ARM in Zynq etc and soft core like Microblaze, NIOS which can be used for any processor based complex embedded design using FPGAs. Configuration data also known as bit file programmed CLB and programmable switching blocks in the configuration memory during reconfiguration of the FPGA devices. Based on the nature of configuration memory FPGAs can be classified into three categories. If configuration data is stored in SRAM within FPGA then it is known as SRAM-based FPGA and if configuration data is stored in Flash memory then FPGA devices are known as Flash-based FPGA. Antifuse-based FPGAs are slightly different from the other two in the sense that they can be programmed only once. Antifuse element does not conduct initially but during programming the antifuse switches are burned out that cannot be returned into the initial state. Maximum modern FPGAs are either flash based or SRAM based.

Now a days FPGAs are widely used in different embedded applications due to their multiple advantages briefed as follows:

- FPGA can be used as co-processor of the CPU in the modern computer or as an accelerator using high speed bus, like PCIe.
- FPGAs are interconnected with high speed peripherals like USB, optical transceiver, Ethernet, PCIe for data transfer with external devices.
- FPGA supports partial reconfiguration that helps to replace a module of the design without hampering the functionalities of other modules.
- Memory access time inside FPGA devices are very less.
- FPGA also supports remote reconfiguration.

FPGAs are commonly used to achieve high performance computation as they offer spatial parallelism involving the configurable blocks.

1.2.5 FPGA Design Flow

The design flow starts with the design specifications given by the user and finishes into a working design in an FPGA. The design flow for a FPGA based system is shown in Figure 1.9. Initial stage involves problem decomposition, project requirement and functional simulation. In the next stage for design entry there are different techniques like hardware description language (HDL), schematic and combination of both. During the logic synthesis HDL codes are mapped into gate level net-list and the net-list is stored as Native Generic Circuit (NGC) file. The implementation process involves three steps: Translation, mapping, placement and routing (PAR). In the translation, all the input net-lists combine with the constraints to form a logic design file and the static data into Native Generic Database (NGD) file. Mapping process divides the whole design into multiple sub-blocks and try to fit into the target FPGA devices to generate the Native Circuit Description (NCD) file. PAR maps the sub-blocks into logical blocks according to the constraints provided by the user and creates the interconnection of the logical blocks utilizing the FPGA fabric. Static timing analysis after MAP or PAR helps to find the path delay and timing violation for the design derived from the design logic. Automatic mapping can also be done by some tools like automatic HDL generators [22]. Lastly, using the programming file (bit file or

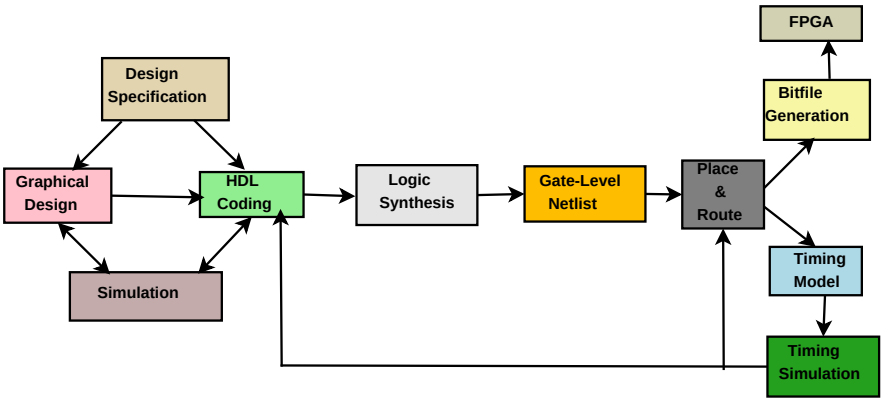

Figure 1.9: HDL based design flow for FPGA based system design

configuration data) the FPGA is configured or programmed. Design softwares offered by the FPGA vendors support the complete design flow.

Embedded systems based on FPGA devices are vulnerable to external noise (specifically created by radiation) since physical access to these devices can be done at an ease. In many of the cases the faults are caused unintentionally due to the natural effects e.g radiation, cosmic ray. Detecting the errors occurring due to these faults and also correcting them in the hardware will make the system more trustworthy. In CBM experiment, radiation level is so high within the cave that the commercial-of-the-self (COTS) FPGAs cannot be used. One of the common solution to prevent FPGA devices from the effect of radiation is to use radiation hardened (Radhard) FPGAs like space-grade FPGAs provided by Xilinx [23], Altera or Microsemi [24] but they are more costly compared to the COTS FPGAs [25] and are also few generation behind than COTS FPGAs. Hence, usage of SRAM-based COTS FPGA is only possible in such radiation environment with appropriate error mitigation techniques. Though ASICs are not very flexible and require more developmental effort, it is much less susceptible to ionizing radiation than COTS FPGA [26]. In some areas within the CBM cave (like near the detector system) use of ASICs are better option compared to FPGAs as ASICs are more radiation tolerant compared to FPGAs. There are some other areas with (comparably) modest radiation levels like in CBMBunker [7] as shown in Figure 1.6 where FPGAs might be the better option to use. Radiation-induced charge particles like alpha, beta can either corrupt user data in the

communication link which connects different electronics devices in the readout chain or can directly damage the FPGA devices itself. Radiation corrupts either single bit or multiple bit in user data stored in the RAM or configuration data in configuration memory of FPGA devices. In this thesis error mitigation techniques have been developed for safe guarding the configuration data of FPGA devices in configuration memory as well as user data in communication links.

1.3 Research Motivation

- Design and development of high speed DAQ system is very much needed that can work without human interference in the accelerator under harsh radiation environment. Hence, there should some methods for monitoring the status of the DAQ system.

- Radiated charge particles may corrupt the data packets during the transmission through high speed interconnect within the DAQ system. Hence, multi-bit error correcting codes are needed to correct the corrupted data in high speed interconnection like optical fiber. Apart from the error in the data stream indeterministic latency added by different memory elements is a critical issue for any high speed data communication. It is very much needed to fix the latency in the high speed communication by bypassing internal memory elements for clock domain crossing and phase alignment.

- Solid state devices like FPGA used for implementation of data processing algorithm may also be affected by radiated charge particles generated during different experiment or cosmic ray during space exploration. In order to mitigate the soft errors arise due to SBUs and MBUs from radiated charge particles efficient error correcting codes with less overhead and simple decoding circuits are needed.

- With the increase of density of CMOS transistors in memory element like flash memory probability of corruption in stored data also increases. When the MBUs are adjacent to each other they form an error cluster and sometimes traditional multi-bit error correcting codes are not efficient against

clustered error. Hence, simple clustered error correction methods with small error correction time is very much needed for high density memory elements.

- To maintain the uninterrupted operation of a real time system there should some provisions for repairing a functional module without hampering functionalities of other modules.

1.4 Research Objective

The main research objectives of this thesis are:

- Testing and characterization of front-end ASICs that will be used to readout the data from MUCH detector and integration of front-end ASICs with the remaining portion of readout chain.

- FPGA implementation of IPBus protocol over one Gigabit Ethernet for remote monitoring of different registers of the DAQ system.

- Development of high speed DAQ systems having error resilient latency optimized communication link and direct memory access controller with PCIe interface. An efficient memory management algorithm is also proposed for data aggregation before data transmission through PCIe.

- Hardware implementation of clustered error or error arises due to adjacent MBUs correction techniques for high density memory element like MLC flash memories are discussed. The proposed methods have less error correction time, low decoding complexity and low overhead compared to the state of the art solutions.

- Development of different SBUs and MBUs mitigation techniques for the configuration memory of FPGAs and their efficient hardware implementation.

- Dynamic partial reconfiguration with proper hardware scheduling algorithm is developed for error correction in a module within a FPGA devices with minimal interface to the normal system operations.

1.5 Organization of the Thesis

In this thesis, our contributions target some specific design and implementation issues to overcome some of the challenges existing in the present state-of-the-art embedded system under radiation environment. Related research work and motivations of the thesis is described in Chapter 2. Chapter 3 deals with testing and characterization of front end ASICs and its integration with FPGA based GBTx emulator and data processing board using differential electrical line and optical fiber. Development of FPGA based data aggregation and control system for radiation environment is proposed in chapter 4. In this chapter we have also proposed an efficient memory management module that helps data aggregation removing path delay of the data packets before processing by back-end computing nodes. Chapter-5 deals with a latency optimized clustered error correction technique in MLC flash memory chips using shortened product code. A novel algorithm using Hamming product code with selective bit placement and frame interleaving for soft error mitigation in the configuration memory of FPGA devices and its hardware implementation is described in chapter-6. Single bit and multi bit upset mitigation techniques using simple parity based coding and erasure code along priority based hardware scheduling algorithm is discussed in chapter-7. Finally chapter-8 gives brief conclusion and future scope of the thesis.

Chapter 2

Related Research Work

In the late sixties and seventies before different high performance electronic devices like FPGA, ASICs came into the market, FEEs in different HEP experiments were read-out by simple minicomputers in the DAQ system that could handle only a few hundred of data channels [27]. Due to lack of parallelism, standard bus architecture and interconnection it could support data rate only upto a few kilobytes per second. In 1964, Atomic Energy Commission of USA defined standards for connector, front end module, power and signal level of Nuclear Instrumentation Module (NIM) used in different HEP experiments but it did not provide any specification for the back-end bus architecture of the DAQ system [28]. In 1969, European studies on Norms for Electronics (ESONE) introduced a computer controlled modular bus architecture known as Computer Automated Measurement and Control (CAMAC) [29] but it was also lacked of parallelism and limited to low data rate applications only. Authors in [29] proposed a PC-CAMAC based DAQ system for the measurement of correlation between the positron annihilation life time and momentum of positron-electron pair using multiple detectors. In 1970, a NIM based DAQ system was developed where FEE modules were readout by a minicomputer using CAMAC as centralized readout bus architecture [30]. With the increase of detector resolution and number of readout channels, event rate of the detector increases which in turn increases data transfer bandwidth. In order to support the huge data transfer rate and data storage requirement, NIM and ESONE jointly developed a new kind of bus architecture known as Fastbus architecture. Fastbus architecture has multi-segment and multi-host architecture

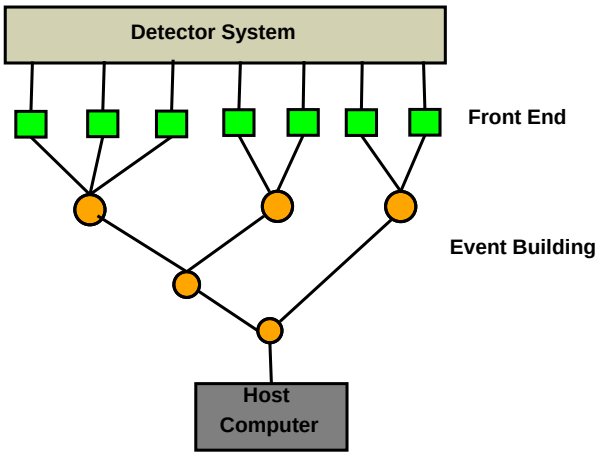

Figure 2.1: Tree like distributed DAQ architecture

and supports wide bandwidth [31]. Use of microcontroller like Motorola 68000 in development of DAQ system had started in late eighties and it led to the development of another new bus architecture known as VERSA Module Europe (VME) bus [32]. Even with the presence of wide bandwidth bus architecture like Fast bus,VME or CAMAC, high speed front end module like NIM, existing DAQ systems were unable to support huge data rate required in different experiments like ALEPH experiment in *CERN* [33]. Typical tree like structure of DAQ system shown in Figure 2.1 consisting of FEEs in the fast level and event builders in higher level help to handle data rate of several megabytes per second. The tree like structure can handle huge data by filtering out the non interesting events using the so called multi-level triggering [34]. These DAQ structures require more complex data format for synchronization and event time stamping. Though the triggering increases complexity of DAQ system it is very useful to remove unnecessary data before storing the data and reduces overload in the high speed interconnection.

Triggering in Experimental Physics Triggering is a mechanism for deciding the events in the particle detector that need to be recorded for future data analysis in the back-end computing nodes. Triggering is very important in the experimental physics due to the limitation of computing power, data handling capability and storage capacity of the memory devices. Selectivity of a trigger is defined as the ratio of the trigger rate to the event rate [35]. In general, two types

of triggers are available in experimental physics: Trivial triggers and non-trivial triggers [36]. In trivial triggers, data is recorded in a periodic interval or in a way that is independent of the properties of events being recorded. On the other hand, in non-trivial trigger events are recorded based on the properties of the events. Two parameters namely the frequency at which events occur and the complexity of each event decide the type of trigger that is needed in the experiment. In both of the cases there is one central trigger unit in the back-end that decides which events are to be recorded and based on that, it sends signal to the FEEs. This makes the FEEs more complex and there is a delay or latency for transmission of trigger from the central trigger unit to the FEEs. Trigger generally uses parallelized design that exploit the symmetry of the detector systems *i.e* the same kind of operations are performed on different parts of the detector systems at the same time. In global sense, the triggers are serial in nature and divided into different levels. In level triggered systems, each level decides the input of the following level that has more time and information to take a better decision. As for example, ATLAS experiment located in *CERN* uses three different levels of trigger [37] as shown in Figure 2.2. The first level is based on the electronics placed near the detector system and reduces event rate from 40 MHz to 75 kHz. Level-2 trigger is a part of the high level trigger (HLT) and is implemented using optimized software algorithm. Level-3 uses event filter (EF) that is also part of HLT and implemented using software algorithm [37]. With the increase in the level of trigger, data rate and global trigger rate reduces so that the first level trigger is always the fastest one compared to higher level triggers.

Though trigger based system reduces the back-end data processing complexity it may create a problem when the objective of the experiment is to record rare events. In order to record the rare events, event rate should be very high. In these cases data will be recorded initially using FEEs and then there will be an online data event selection mechanism that removes unnecessary background events before storing the data for future analysis. This process is known as self-triggered mechanism. Apart from the reduction of chance of misdetection of rare events, advantages of such self-triggered mechanism are:

- Complex event selection algorithm can be easily implemented in software compared to hardware trigger.

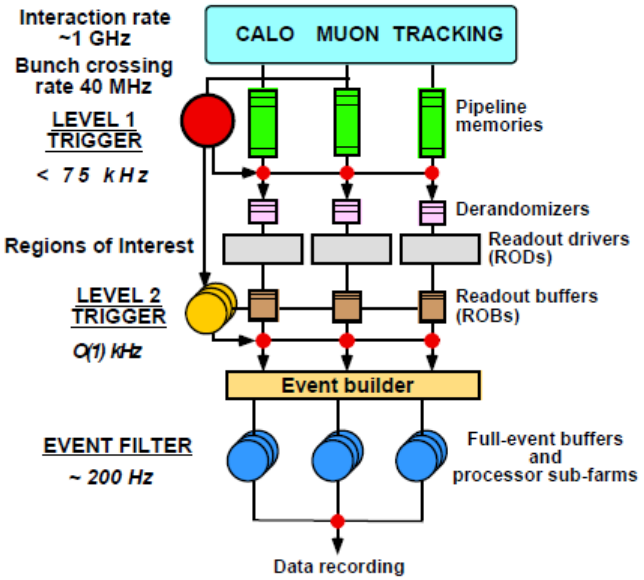

Figure 2.2: Schematic overview of Atlas experiment in *CERN* [37]

- Software based event selection algorithms can easily adapt new criteria compared to hardware based trigger algorithm.
- In self-triggered system, detector dead time due to buffer read out can be reduced.

Disadvantages of self-triggered systems are handling of huge data, global time distribution and time synchronization of the full read out chain.

Use of high speed interconnection like Ethernet, Myrinet [38], optical fiber, switch based event builders, PCIe based back plane interface helped the modern DAQ system to handle data rate and trigger signal in the order of Gigabit per second (Gbps). Apart from high speed data transfer, modern DAQ systems use complex data processing algorithm for different purposes like feature extraction, clustering and data aggregation among others. In order to implement such complex algorithms sometimes arithmetic logic units (ALU) present in general purpose processor may not be sufficient. With the increase of the channel number, more data samples enter into the DAQ in parallel and general purpose processor serialize them before processing. The efficiency of the DAQ system increases many folds if the DAQ can process the data samples in parallel. In the present era

of embedded system only FPGAs and ASICs are suitable to sort out the above mentioned problem due to presence of huge number of digital signal processing (DSP) slices (equivalent to ALU for general purpose processors) [39]. As for example, the number of DSP slices in virtex-6 series FPGA of xilinx varies from 288 to 2016 [40]. These huge number of DSP slices help to implement different complex algorithms and process data in parallel. The on field programmability, modularity makes the FPGA more suitable for modern DAQ system. High gate count in FPGA devices help modern DAQ system in different ways:

- Acceleration of complex data processing algorithm.
- Integrated system on chip based approach helps in implementing various complex functions on FPGA fabrics.
- Reconfigurability for design upgradation.

Use of FPGA like devices and PCIe like back-end interface give freedom to the designer for either on-board data processing or PCIe based backplane data processing. On-board data processing scheme processes data in different level of hierarchical DAQ network and only useful data will be sent to the computing nodes. This reduces the data processing load in the computing nodes and helps to distribute the data processing loads throughout the DAQ network. At the same time it increases the complexity of the DAQ network. In the data processing across the PCIe backplane data will be processed only in back-end computing nodes and devices in different levels of hierarchical DAQ network will act as data aggregator. Here computer in the backend should have multi-core CPU, and provide power and mechanical support to multiple PCIe cards.

In the modern digital world, single core or even dual core 32-bit or 64-bit processors are not sufficient for implementation of complex data processing algorithm or analyze huge data. In order to support big data analysis, different companies have developed multi-core processor like Xeon by intel and Ryzen by AMD [41]. Multi-core technologies are also taking lead roles in embedded computing domain to handle huge data like Xilinx Zynq UltraScale MPSoC [42] has Quad-core ARM® Cortex™-A53 and Dual-core ARM Cortex-R5. Along with

the huge data processing within one device, high speed data transmission between multiple devices are also very important in modern day applications like data communication between different smart phones, biomedical devices, data acquisition system used in satellite and some real time experiments. The need of FPGA based real-time data communication is of special interest where FPGA based DAQ systems are built to capture, store, process and transmit the data generated from real time experiments [43], [44].

In traditional DAQ systems, FEBs capture data from sensors through high speed LVDS link, process it and send it to the back-end storage devices through Ethernet or RS232 for further analysis. Authors used FPGA-based DAQ for positron emission tomography in [45] and for optical tomography in [46]. In [45], each DAQ module comprises of a position sensitive photomultiplier tube, LSO scintillator crystal block, analog signal conditioning circuits, a digitizer, an FPGA module for digital processing and transmission of collected data to the computer cluster. A general purpose FPGA-based DAQ controller is developed by the authors in [47] that can easily be used in different applications with simple modifications in software or firmware part. The generic controller in the proposed framework is developed with both hard processor (PowerPC) and soft processors (Microblaze) and can be used with or without real time operating system like Operating System Embedded (OSE) as per the user requirement. Authors in [48] developed a Gigabit Optical Serial Interface Protocol (GOSIP) for communication over optical fiber and implemented PCIe to optical link interface in FPGA for the DAQ system with a stable data rate of 1.6 Gbps. It is basically a master slave protocol where the front end card works as slave and PCIe card works as master. Here each master is capable of handling multiple slave devices.

When the DAQ system will be used for different critical applications robustness of the system against external disturbance is very important. Main problems of these conventional DAQ systems are low data rate and resiliency against the SBU and MBU in the radiation environment. An optical link between two computing nodes having PCIe interface with a data rate of 8.5 Gbps has been proposed by the authors in [49] where they have used PCIe hard IP available in ALTERA Stratix IV FPGA board. A high-speed custom data transmission protocol over optical fiber is proposed by Hao Xu et.al in [50] for real-time data

acquisition in Beijing Spectrometer III (BESIII) trigger system. Here they have used Virtex-II pro FPGAs of Xilinx that can handle data rate upto 6.25 Gbps. Liansheng Liu et.al presented the development of a fiber channel node with PCIe interface for avionics environments in [51] where each node consists of two modules: FPGA module and PowerPC module. Two nodes are connected by optical fiber through Small Form Factor Pluggable (SFP) interface and maximum data transfer rate achieved in this case was 2.125 Gbps. Serial Front Panel Data Port (SFPDP), another high-speed data transfer protocol implemented on FPGA by authors in [52] used to capture data from DSP unit through Extended Attachment Unit Interface (XAUI) with different speeds: 1.0625 Gbaud, 2.125 Gbaud, and 2.5 Gbaud. In [53], authors have used a bus master DMA along with a 4-lane second generation PCIe link to transfer the stream data from FPGA to PC with the data rate of 784 Mbps.

In new generation high intensity HEP experiments, millions of free streaming high rate data sources are to be readout [54]. Free streaming data can only be controlled by thresholds as there is no trigger information available for the readout. Therefore, these readouts are prone to collect large noise and unwanted data. For this reason, these experiments can have output data rate of several orders of magnitude higher than the useful signal data rate. It is therefore necessary to perform the online processes on the data to extract useful information from the full data set. Without trigger information, pre-processing on the free streaming data can only be done with time based correlation among the data sets. Multiple data sources have different path delays and bandwidth utilizations and therefore the unsorted merged data requires significant computational efforts for real time manifestation of sorting before data analysis. Commonly used sorting algorithms like bubble sort [55], merge sort [56], heap sort [57] either consumes huge memory space or have high latency. In order to increase the speed of the sorting process, various hardware sorting algorithms have been proposed by the authors in [54, 58], but none of them are suitable for sorting online data stream.

In general, the DAQ system in a HEP experiment handles large data in a harsh radiation environment [59]. Error control strategies in the communication channel under radiation can be classified into two broad categories: Automatic Repeat Request (ARQ) and Forward error correction (FEC). In ARQ based sys-

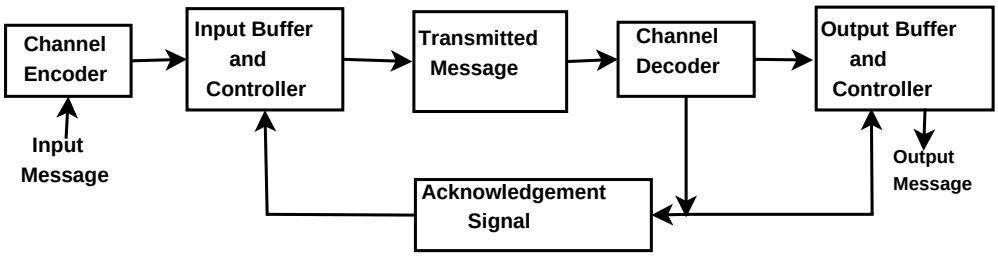

Figure 2.3: Basic architecture of ARQ based communication system

tem communication is in two ways *i.e* from transmitter to receiver and from receiver to transmitter. Error control in ARQ system [60] is achieved by error detection and retransmission but error correction is not possible. A typical ARQ system is shown in Figure 2.3. When errors are detected in the receiver using any error detection algorithm like CRC, channel decoder issues a negative acknowledgment (NACK), otherwise positive acknowledgment will be sent to the transmitter. There are three variants of ARQ schemes: stop and wait scheme, Go-back-N scheme and selective repeat scheme. In stop and wait ARQ scheme, transmitter has to stop after transmitting every code word and wait for acknowledgment from the receiver. Though the input buffer size in transmitter is small to store single message, the delay time between two consecutive words is very high. In Go-back-N scheme, messages will be sent continuously until transmitter receives a NACK and request for retransmission. When the transmitter receives NACK, it goes back N words in the buffer and retransmits messages starting from that point discarding N-1 intervening words. Though it reduces transmission delay, input buffer size in the transmitter side increases. In the selective repeat scheme, transmitter will retransmit only erroneous message. The ARQ based system is not suitable for high speed real time data transmission as data cannot be stored in the communication path.

On the other hand in FEC based error control technique, errors are corrected at the receiver so retransmission is not needed in this technique which is suitable for high speed communication. Though FEC based error control technique automatically corrects errors, it requires more redundant bits, larger bandwidth and complex decoder compared to ARQ based system. Authors in [61] proposed a high speed serial off and on-chip communication system with single error correcting and double error detecting (SECDED)(63,56) Hamming code and im-

plemented using 0.25μm CMOS technology. Block codes like Bose, Chaudhuri, Hocquenghem (BCH) [62], Reed Solomon (RS) are the most commonly used ECCs in a high speed communication system. In [62], authors implemented a braided block code with BCH as the component code for high speed optical communication. Not only block codes, memory based convolutional code, turbo code, and Low Density Parity Check code (LDPC) [63] are sometimes used to make the communication system robust against different environmental hazards, but their decoding circuit complexities are quite high. In general, performance of long distance high speed communication is limited by the low net coding gain of the FEC used in the system. Authors proposed a layered Quasi-cyclic LDPC (QC-LDPC) architecture in [64] that have good error correcting performance, low complexity, high throughput and high code rate. They have implemented both regular and irregular QC-LDPC on FPGA with different throughputs. The most common method for decoding of convolutional code is viterbi algorithm that consumes large area, huge power for high speed decoder. A novel viterbi decoder architecture is proposed in [65] that reduces the computational complexity and power consumption compared to traditional viterbi decoder with negligible performance reduction. An area efficient interleaved convolutional code along with its viterbi decoder architecture is proposed in [66] for high speed communication where authors have used a parallel processing with register exchange methods during the decoder design.

The probability of occurrence of an error in a communication channel can be reduced exponentially by increasing the data block length which also increases decoding circuit complexity. Dave Forney in [67] showed that concatenated code can be used to reduce the probability of error exponentially without increasing decoding complexity further. In general, a concatenated code is composed of an ECC as the outer code and another ECC as the inner code. In high speed communication, block codes are used as a component code instead of convolution code due to its high performance and low implementation complexity. Concatenation between RS(255,239) and BCH(2184,2040) code and two BCH codes are presented in [68], [69] respectively for high speed architecture. In [70], authors, concatenated RS code with turbo code for Advanced Orbiting System (AOS) and showed that the performance of concatenated code is better compared to other

convolutional or block code in terms of coding gain and decoding delay. They achieved the data rate of 3 Mbps. A novel high speed communication system using concatenation of RS and QC–LDPC Code is proposed in [71] where authors proved that concatenation improved BER by 3 dB. Here RS code is used as outer code due to its good burst error correcting capability and QC-LDPC is used as inner code. Random interleaver is placed in between RS and QC-LDPC to enhance error correction capability without increasing redundancy. Two turbo codes are concatenated in [72] to reduce implementation cost compared to standard 3GPP turbo code which achieved the bit rate of 4.11 Mbps compared to 75.264 kbps in standard 3GPP turbo code. In [73], a concatenated code by combining Recursive Systematic Convolutional (RSC) code and LDPC code is developed which shows better Block Error Rate (BLER) compared to conventional LDPC codes and turbo codes.

Minimization of latency in the transceiver of a high speed communication system is a key factor for time critical applications. Latency is the time a packet of data takes to get transferred from one point to another point within the system. Using parallel decoding strategy, authors reduce the latency in the receiver from 1928 clock cycles to 578 clock cycles in [69]. In some critical applications, link latency must remain fixed after each power up or reset. By using changeable delay tuning technology and dynamic clock phase shifting authors proposed fixed latency transceiver in [74]. Different buffers within the transceiver are used for clock domain crossing and phase alignment which increases the latency of the whole system as described in [75].

Apart from the user data in the communication channel or block RAM on FPGA, configuration data in the configuration memory of FPGA devices may also be affected by radiation. In Figure 2.4, configuration memory of FPGA with the series of 1's and 0's represent the function of the routing and logic elements interconnection switch, LUT, and Flip-Flop to implement a logical NAND and NOT gate into FPGA. Figure 2.5 shows a SEU (sometimes known as bit flips) changes the data in configuration memory which leads to an unintended signal routing and change in the logical function. Effect of SBUs and MBUs in FPGA can be broadly classified into two broad categories: temporary or transient and permanent faults [19]. Effect of transient faults into any logic will be transferred

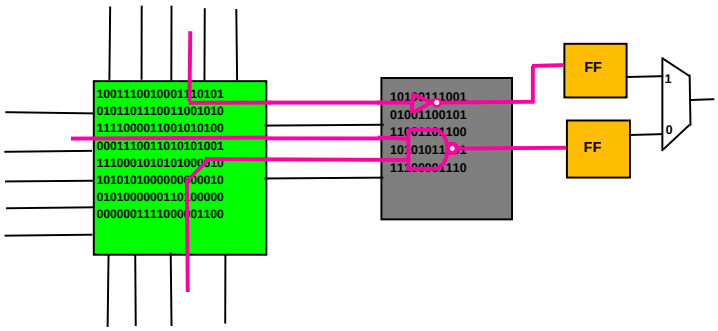

Figure 2.4: Implementation of NAND and NOT logic in an FPGA

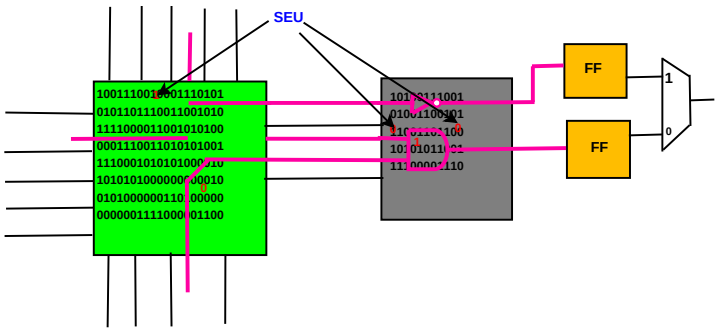

Figure 2.5: SEU causes error in routing logic and NAND gate in an FPGA

to the output of flip-flop or memory if its period of occurrence is higher than the period of the clock which drives the circuit. On the other hand permanent faults are of two types: one is permanent recoverable fault and other one is permanent nonrecoverable fault [19]. When charge particles permanently damage the logic or switching blocks within the FPGA, permanent nonrecoverable faults occur and it can only be sorted out by replacing the defective logic blocks physically (normally it is done by routing [76]). Permanent recoverable faults in FPGA may be corrected either by rewriting the content of the configuration memory or by using built-in EDAC code. In this work, only permanent recoverable faults are considered.

One of the common solution to prevent FPGA devices from the effect of radiation is to use radiation hardened (Radhard) FPGAs like space grade FPGAs, but they are more costly compared to the commercial-of-the-self (COTS) FPGAs [25] and are also few generation behind than COTS FPGAs. Hence, in different commercial applications, COTS FPGAs are used with various error mitigation tech-

niques. Triple modular redundancy (TMR) [77] and concurrent error detection (CED) [78] are most commonly used techniques in FPGAs for error mitigation. In TMR, three identical logic blocks having the same functionalities are connected in tandem and final output from the system will be obtained through majority voting. An additional error detection circuit is used along with the main circuit in CED [78] and when any error is detected, the main circuit recomputes or rolls back the whole operation from the beginning. The above mentioned schemes consume large area, huge power and are not suitable for real time applications. Large overhead of TMR can be reduced by using partial TMR [79] where TMR is used only for critical portions of a circuit instead of the whole design. Another common method of error mitigation in FPGA is scrubbing where one copy of original bit file (also known as the golden copy) is stored separately in a Radhard memory before downloading it into the configuration memory. There are two type of scrubbing: blind and readback scrubbing. Blind scrubbing download the golden copy into the configuration memory with a periodic interval. On the other hand readback scrubbing download the bit file only if error is detected in readback configuration file. It reduces the effect of accumulated error in FPGAs and increase the lifespan of the FPGA devices [80]. Sometimes TMR can also be used intelligently with scrubbing to reduce the effect of SBU as described in [81] for Virtex FPGAs. Both of the above mentioned schemes continuously access external Radhard memory, which increases the cost and introduces some delay. Use of partial reconfiguration with scrubbing can reduce the effect of delay as described by the authors in [82] where a part of the bit file will be downloaded during scrubbing without downloading the whole one.

Different commonly used ECCs like BCH, RS [83], LDPC [84] can be an alternative solutions to correct errors in the configuration memory of FPGAs against soft error though the hardware complexity and resource utilization of the above mentioned codes are very high. Hence, simple low complexity ECCs are always preferred to correct errors in the configuration memory of FPGA devices. With the increase of the error correcting capability of EDAC code, decoding circuit complexity and latency also increase. To support real time applications, EDAC codes with less complex decoding circuit and good error correcting capability are always preferred. The most common technique to correct single bit error in con-

figuration memory of FPGA is Hamming code [85]. Sometimes bit interleaving with Hamming code increases error correction capability many times as shown in [86]. Authors in [87] showed that selective bit placement within the memory element enhances detection and correction capability of Hamming code for multi-bit adjacent errors. Soft error mitigation controller (IP blocks) provided by Xilinx based on CRC and EDAC can correct at most two adjacent bits [88].

Error correction in memory unit can be improved by using two-dimensional ECCs and mix code (combination of multiple ECCs). In [89], authors used the Hamming product code (HPC) to correct MBUs in each configuration frame. They had also proposed one special kind of memory as the hardware implementation of their proposed code is tough in the conventional configuration memory. A new kind of mix code combining Euclidean Geometry LDPC code and Hamming code with low overhead is proposed in [90] for error correction in storage cell and decoding circuit. Authors claimed that they achieved good error correcting performance against MBUs and overhead due to redundant bits are also less compared to the existing multi-bit ECCs. RS code is concatenated with Hamming code to correct errors in dynamic random access memory controller in [91]. In order to reduce the complexity of error correction technique only error detection capability of BCH code or Hamming code is combined with simple parity based ECC as illustrated in [92]. A low overhead Matrix code (MC) combining Hamming code and parity code is proposed in [93] to correct multi-bit errors which shows better performance compared to Reed-Muller [94] code and Hamming code in terms of area, power consumption, critical path delay and EDAC coverage. Authors used convolutional code with very complex decoding circuit to mitigate the effect of SEU in [95]. Nowadays, different erasure codes [96] are used for multi-bit error correction in the memory of FPGA. Ebrahimi *et.al* used interleaved parity check code along with erasure code to protect configuration memory of SRAM based FPGA from MBUs in [97].

DPR along with error correction increases error correction coverage without increasing error correction time or latency. Use of proper hardware scheduling algorithm during partial reconfiguration helps to enhance system performance. Few literature is available in recent years where fault correction and detection approaches are proposed along with hardware scheduling and partial reconfigu-

ration. Checkpointing is a very common method that can be used along with partial reconfiguration to reduce the reconfiguration time. In [98], authors combine scrubbing with checkpointing and set checkpoint frequency in such away that tasks will be completed within their deadline. Whereas in [99], configuration memory scanning, and fault correction processes are placed according to the task execution time and its deadline. An online algorithm for proper placement of the checkpoint is proposed by the authors in [100] that reduces the fault recovery time in a FPGA by storing intermediate states of a program. A new scheme for fault tolerance and energy saving for fixed priority based real time embedded system is proposed by the authors in [101]. In [102], authors designed a download manager to schedule the configuration process of m number of partially reconfigured regions using n number configuration ports (where $m > n$). The proposal raised in our literature comes from a different aspect where fault correction priorities of different hardware blocks are calculated from the task period and its criticality. Operating system (OS) based applications are involved with many timing parameters like worst case execution time and deadlines etc. To meet all of these timing constraints, OS should support applications acquiring computation resources timely. Hence, in [103] the fine-grain task control and performance estimation over timing parameters is one of the most important features for designing a real-time OS (RTOS). The fine control of the typical system can be achieved by reducing the time scale which unfortunately increases overhead because the time scale on the OS depends on the execution of the timer interrupt service routine (ISR). Large time scale resolution may improve performance but make the ISR more hectic. Here the issues of our approach are: 1. Fault Detection and correction can be processed without suspending the system operations. 2. The proposed configuration scheduler works form hardware level. In bare metal platform the approach can be used efficiently, as well as for embedded processor platform, even with standalone or OS environment, the proposed configuration scheduler may not put any significant overhead to Timer ISR. 3. The proposed configuration scheduler block consumes very less resources and power.

Chapter 3

Integration of MUCH-XYTER with DPB using FPGA based GBTx Emulator

CBM experiment is a part of the FAIR facility where the main challenge is to measure the particles generated in nuclear collisions with unprecedented precision and statistics. In order to capture the data generated in each collision, a highly time synchronized fault tolerant self-triggered electronics is required for DAQ system that can support high data rate (~TB/s). Basic readout chain for CBM experiment consists of a FEB with X-Y Time Energy Read-out (XYTER) ASIC, a radiation hardened high speed optical transceiver board with Gigabit Transceiver (GBTx) ASIC followed by a FPGA based Data Processing Board (DPB) and First Level Event Selector Interface Board (FLIB). XYTER ASICs receive charges from the detector, digitized it and send it to GBTx using Low Voltage Differential Signalling (LVDS) electrical line also known as E-link at 320 Mbps. GBTx after capturing data from multiple XYTER aggregates them and send to DPB through optical fiber at 4.8 Gbps which in turn sends data to FLIB using 10 Gbps optical connection. Data having close range of time stamp values are grouped into several clusters in the FLIB and each clusters are termed as micoslice. Then each microslice is sent to a single computing node using PCIe for further analysis. In this chapter, we have integrated MUCH-XYTER ASICs

(XYTER ASICs for MUCH detector) with DPB using FPGA prototype of GBTx ASICs also known as GBTx Emulator. In order to control the readout chain and monitor the internal registers of different electronic devices remotely, DPB is interfaced with IPBus protocol over one Gigabit Ethernet.

3.1 Introduction

HEP experiment involves the study of different laws that governs the universe at the subatomic level. For studying the microsecond old universe different ions like gold, lead are accelerated and made to collide with a fixed target that will generate a large number of particle in CBM experiment. In order to support the large data volume at the output of detector, high speed fault tolerant DAQ system is needed. DAQ system used in HEP experiment has two parts: One is near the detector (sometimes known as on the detector) and placed in the harsh radiation zone and other is usually placed outside the experimental cave i.e in the control room (green cube for CBM experiment) where radiation level is low. Radiation tolerance and low power dissipation make the ASICs suitable to work with detectors having high channel density in harsh radiation environment. On the other hand, electronics which are placed outside the radiation zone i.e in the control room can use commercial FPGA due to some of its inherent advantages over ASICs like on-field programmability, low cost and large logic resources. There is some design specifications for readout chain of MUCH detector:

- FEE boards should have high channel density, high speed analog to digital converter and waveform shaping circuit.
- There should be proper electrical and mechanical interconnection between detectors and FEBs.
- Supports efficient data aggregation schemes and rate conversion.
- Handle data rate upto 1 Tb/sec.
- Low cost commercial FPGAs having high logic resources with proper error mitigation techniques will be used in the control room.

- FPGA boards should support different high speed interconnections.
- Proper timing synchronization is needed between different boards used in the readout chain.
- Design should be very compact due to limited space.
- Readout chain can be controlled remotely.

In the readout chain of CBM experiment, XYTER that acts as FEE ASIC will be placed on the detector and provides high speed waveform sampling, low noise and high dynamic range amplification, pulse shaping, nanosecond timing, digitization and can sustain high radiation. GBTx ASIC which is also within the high radiation zone provides high speed bidirectional fault tolerant data transmission, precise timing information, control and monitoring features. DPB and FLIB which are outside the radiation zone are developed using commercially available Kintex7 FPGA from Xilinx. In order to support high speed data transmission between the electronic devices, high speed interconnects like optical fiber, Ethernet and PCIe are used. At the same time data volume in the readout chain can be reduced by sending trigger signal from the back-end to the front-end ASICs to capture data for interesting events only. Events that are not coherent with the trigger conditions will be discarded. In CBM experiment, a predefined threshold voltage is set in the front-end analog channel of FEE ASICs and if the signal from the detected particles are above the threshold they will be selected. Hence, in this experiment data rate is higher compared to other HEP experiment with conventional trigger. Though there is no trigger from the back-end to FEE ASIC to reduce data volume online computing systems in the back-end removes the unnecessary data before storing the data for future analysis due to the physical limitation of the storage memory.

Simplified readout chain for MUCH detector in CBM experiment is shown in Figure 3.1. The detector systems spread over several square meters of area all equipped with the MUCH-XYTER ASICs sitting on top of the detector using specially designed kapton cable to achieve better noise performance. There is about 50K FEE ASICs that collect data from detectors and send it to the GBTx using LVDS E-links having length of almost 50 meters on average. There is around

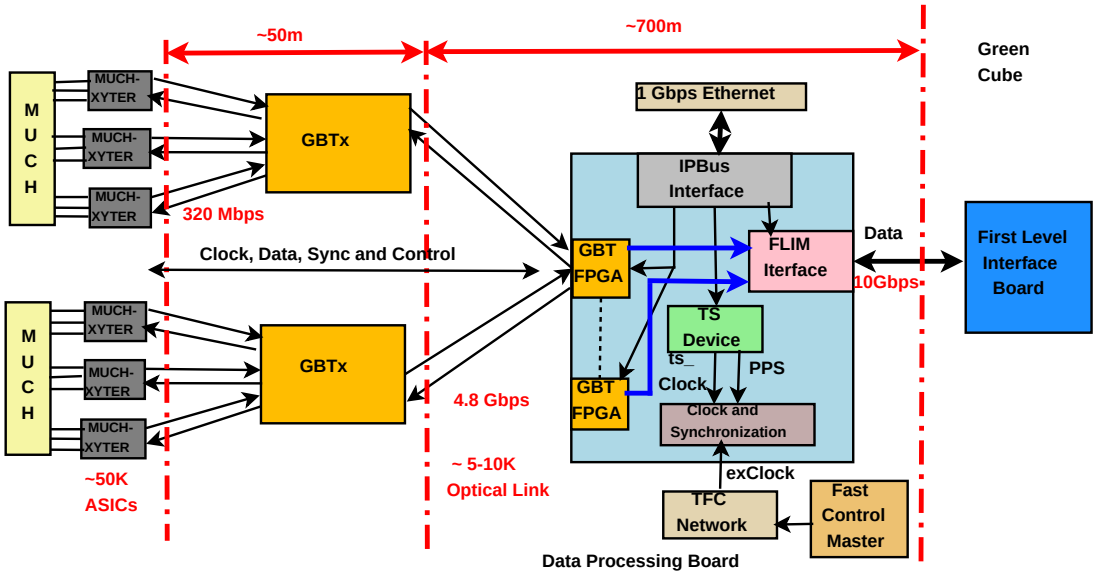

Figure 3.1: Schematic diagram of the read out chain for MUCH detector

5-10K optical link between GBTx and DPB having length of 700 meter that helps to carry the data from the highly radiated zone (within the cave) to moderate radiation zone. Different modules within the DPB like IPBus interface, GBT-FPGA core, TS-Slave, TFC network, FLIM interface, clock and synchronization modules are shown in Figure 3.1, which we are going to discuss in details later in this chapter. The main contributions in this chapter includes:

- FPGA implementation of GBTx ASICs with efficient error handling capability for mitigating soft errors.
- Study and characterization of MUCH-XYTER ASIC.
- Integration of MUCH-XYTER with DPB using GBTx Emulator.
- FPGA implementation of IPbus protocol for controlling and monitoring of readout chain.

3.2 FPGA based GBTx Emulator

GBTx is a Radhard ASIC developed at *CERN* [104] to implement multipurpose high speed optical link of HEP experiment. Optical links are used for data ac-

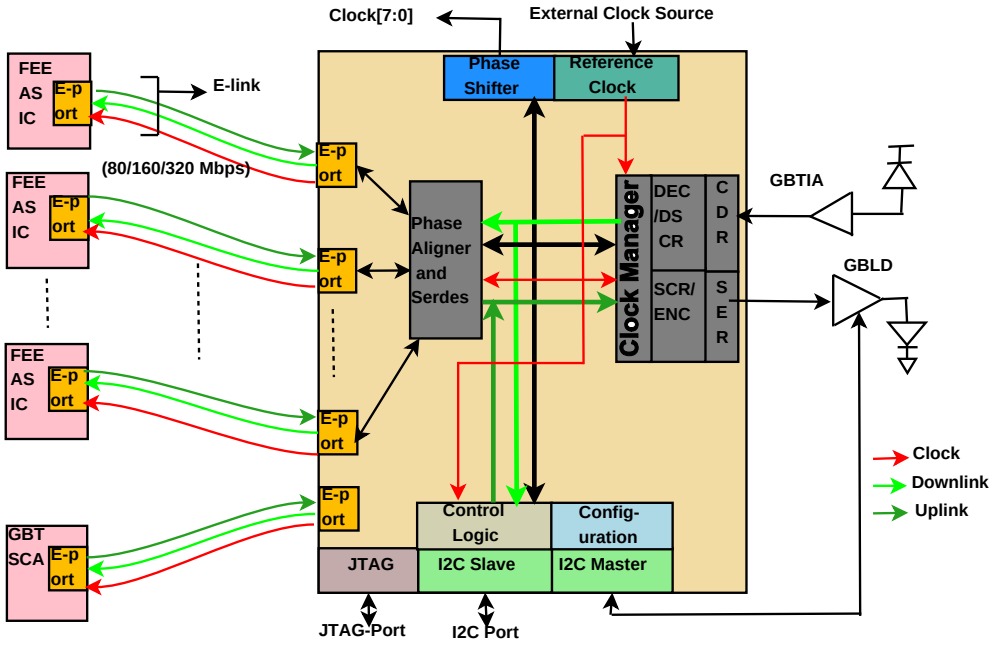

Figure 3.2: GBTx ASIC internal architecture and interfaces [104]

quisition, sending trigger and timing information to the front end detector and distribution of slow control information in the network. Single bi-directional optical link carry these information with high reliability in a very harsh radiation environment. Due to non-ionizing energy loss (NIEL) and total dose, electronics and optoelectronic components placed near the detector face immense radiation. To alleviate the effect of NIEL and total dose the GBTx ASIC is developed using 130 nm CMOS technology with special layout structure. Electronics and optoelectronic components that will work with GBTx are custom built and Radhard. GBTx receives the electrical signals from the FEE ASICs and convert it into optical signal to transmit long distance outside the harsh radiation zone using robust channel coding technique. It helps to develop the DPB that accepts data from GBTx and placed in low radiation zone using commercially available electronics components.

GBTx ASIC is used along with other components like a laser driver (GBLD) for back-end communication over optical fiber, a trans-impedance Amplifier for the optical receiver (GBTIA) and a slow control adapter ASIC (GBT-SCA) for controlling different internal registers within the GBTx. GBTx ASIC along with

GBLD, GBTIA and GBT-SCA forms the GBT chipset. Internal architecture of GBTx ASIC and its different interfaces are shown in Figure 3.2. The communication over optical fiber is asynchronous in nature so a clock and data recovery (CDR) circuit is used to recover clock from incoming data from GBTIA. Clock recovery module in the receiver of GBTx is divided into two parts: CDR circuit and frame aligner (FA) circuit. CDR circuit is again divided into frequency locking and phase locking sub-block. In the receiver side serial data from DPB is parallelized, decoded and finally de-scrambled. During the transmission received data from the FEE ASIC is scrambled, encoded with ECC and finally serialized before giving input to transmitter. A clock manager circuit is used to control different high and low speed clocks within the GBTx ASIC and a programmable phase shifter provides eight different clocks with variable phase and frequency. A crystal oscillator on the ASIC generate reference clock and helps for locking the CDR circuit during the start-up. Internal register within the ASIC can be set or reset using I2C slave interface and JTAG interface is used for boundary scanning. GBLD which is attached with the GBTx ASIC can be configured with I2C interface. Here we have implemented GBTx ASIC prototype on commercially available Kintex FPGA from Xilinx to test its functionality which is known as GBTx Emulator.

Main functional blocks within the GBTx emulator is shown in Figure 3.3. GBTx emulator contains reset modules, clock generator module, multiple GBT banks, pattern generator and pattern checker module. Each GBT bank contains Scrambler/Descrambler, RS Encoder/Decoder, Interleaver/De-Interleaver, Gearbox, Frame aligner and Multigigabit transceiver (MGT). Reset and MGT module work at 156 MHz and 120 MHz respectively which are given directly from external source whereas scrambler, interleaver and RS-Encoder works at 40 MHz which is generated from 120 MHz clock using Phase lock loop (PLL).

Scrambler: Every communication channel shows some filtering effect *i.e* high frequency component gets more distortion compared to low frequency component. Equalizer performs reversal of the distortion in the channel and maintain the constant gain throughout the frequency band in the channel. Equalizer takes the sample from the received data stream and update the tap gain by taking the

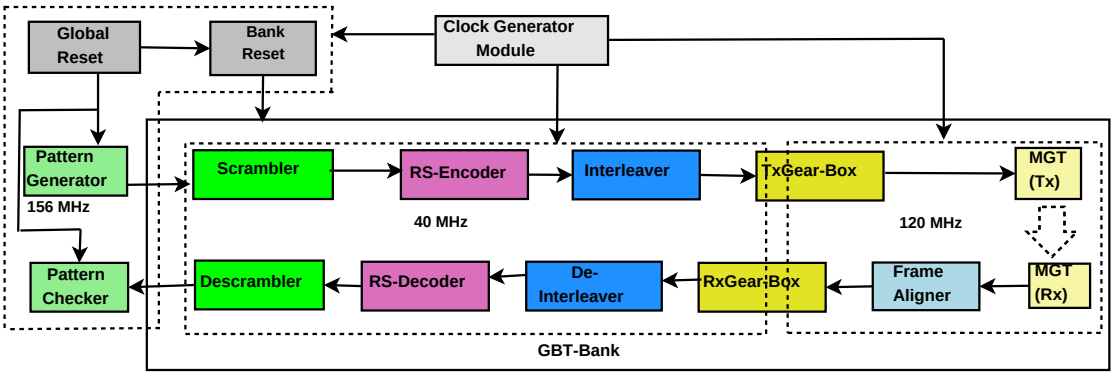

Figure 3.3: Internal modules of a GBT-Bank in GBTx Emulator

weighted sum of sampled input. There should not be any statistical correlation between the aggregated sum at different time instant which is possible only when the input data stream is completely random. Randomness in the data stream also helps to detect the clock edge. Hence, the scrambler is used in the communication system to increase the randomness in the data for avoiding long sequences of '1's or '0's in the data stream in input signal coming from the FEE ASICs. It has a latency of one clock cycle but does not add any overhead in the system like the 8b/10b or 7b/8b line coding. Here 84 bit data (80 bit data with four bit for slow control) is scrambled simultaneously using 21 bit polynomial. These scrambled data from all four blocks are combined together to produce the 84 bit scrambled output data.

Encoder/Decoder block: In the Encoder module RS code is used to correct burst error in data transmission between GBTx emulator and DPB. RS code is an important class of nonbinary BCH code and it is quite different from other code in the sense that here base field and extension fields are same [105]. RS code encode a group of data bits known as symbol instead of individual zeros or ones which is suitable for dealing with burst error in long distance communication. RS (n,k) code with t symbols correcting capability has the following properties:
Block length :$n = 2^m - 1$ symbols; Message size: k symbols;
Parity check symbols: $n - k = 2t$; Minimum distance:$d_{min} = 2t + 1$;
If each symbol length is s and each symbol there is one bit error then RS code can correct t bit errors in $n * s$ bits. Hence RS code is not suitable for random

error correction. RS code is maximum distance separable code [105] that means all the code words in the code space are algebraically far away and uniformly distributed and hence each code word can be properly decoded. RS code is generated using a generator polynomial $G(x)$ where generator polynomial having t symbols correction capability can be written using equation 3.1.

$$G(x) = (x + \alpha)(x + \alpha^2)(x + \alpha^3).....(x + \alpha^{2t}) \tag{3.1}$$

Here α is the element of galois field. Code ward $C(x)$ can be generated using the equation $G(x)I(x)$ where $I(x)$ is the information bits. As for example generator polynomial of double error correcting (15,11) RS code can be calculated by equation (3.2).

$$\begin{gathered} G(x) = (x + \alpha)(x + \alpha^2)(x + \alpha^3)(x + \alpha^4); \\ G(x) = x^4 + (\alpha^3 + \alpha^2 + 1)x^3 + (\alpha^3 + \alpha^2)x^2 + \alpha^3 x + \alpha^2 + \alpha + 1; \\ G(x) = x^4 + g_3 x^3 + g_2 x^2 + g_1 x^1 + g_0; \end{gathered} \tag{3.2}$$

Now the $2t$ parity check symbols can be generated using 3.3

$$P(x) = I(x).x^{n-k} mod G(x) \tag{3.3}$$

Figure 3.4 shows the encoder for systematic RS(15,11) code because here generated redundant symbols will be simply appended at the end of data symbols.The steps during encoding process are as follows:

1. Switch1 will remain close for first k clock cycles so that message symbols can be shifted into $n - k$ stage shift register.

2. Switch2 is at position 2 for first k clock cycle so that message symbol can be transferred directly to the output register.

3. After transfer of k^{th} message to the output Switch1 is opened and Switch2 is moved to position 1.

4. In the remaining n-k clock cycles parity bits are calculated using the data stored in registers with the help of equation 3.3 and will be shifted to the output of encoder block.

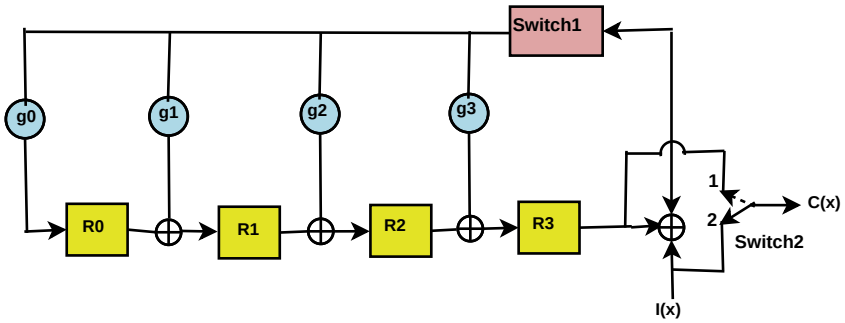

Figure 3.4: Architecture for a systematic RS(15,11) encoder

5. Total n clock cycles are required to complete the encoding process and after n^{th} clock cycle output register will hold original data symbols appended with generated parity bits.

Now suppose $r(x)$ be the received corrupted data in DPB that can be written as $r(x) = c(x) + e(x)$ where $e(x)$ represents the error pattern. First step in the decoding is the syndrome computation. If S_i represents i^{th} syndrome then it can be expressed by equation 3.4.

$$S_i = r(\alpha^i) = r_0 + r_1\alpha^i + r_2\alpha^{2i} + ...r_{n-1}\alpha^{(n-1)i} \quad \forall i \ 1 \ to \ 2t \tag{3.4}$$

As $c(\alpha^i)$ is zero S_i reduces to equation 3.5. It shows that syndromes depend only on the error patterns.

$$S_i = e(\alpha^i) \tag{3.5}$$

Now if $e(x)$ has v error at locations at $X^{j1}, X^{j2}, ...X^{jv}$. Hence $e(x)$ can be written using the equation 3.6.

$$e(X) = X^{j1} + X^{j2} + ... + X^{jv} \tag{3.6}$$

We define error locator number as $\beta_l = \alpha^{jl}$. Now from equation 3.4 and equation 3.6 and putting the error locator number $2t$ syndrome equations can be represented by equation 3.7

$$
\begin{aligned}
S_1 &= \beta_1 + \beta_2 + \ldots\ldots\ldots\ldots + \beta_v \\
S_2 &= (\beta_1)^2 + (\beta_2)^2 + \ldots\ldots\ldots\ldots + (\beta_v)^2 \\
S_3 &= (\beta_1)^3 + (\beta_2)^3 + \ldots\ldots\ldots\ldots + (\beta_v)^3 \\
&\vdots \\
S_{2t} &= (\beta_1)^{2t} + (\beta_2)^{2t} + \ldots\ldots\ldots\ldots + (\beta_v)^{2t}
\end{aligned}
\tag{3.7}
$$

There are $2t$ unknowns in $2t$ equations but they cannot be solved easily as equations are nonlinear. These nonlinear equations in 3.7 can be solved using either Berlekamp-Massey algorithm or Euclid's algorithm. Though Euclid's algorithm is more widely used for its easy implementation but here we have used Berlekamp-Massey algorithm due to its efficient hardware implementation. In the next step one error error locator polynomial (L(x)) is formed from equation 3.7 and roots of this polynomial is calculated using Chien search algorithm. Finally correct value of the erroneous symbols can be calculated using Forney algorithm. Different steps of decoding algorithm is shown in Figure 3.5. Here X_i and Y_i represent error location and magnitude of error respectively. Details of RS decoding algorithm and their hardware implementation can be found in [106].

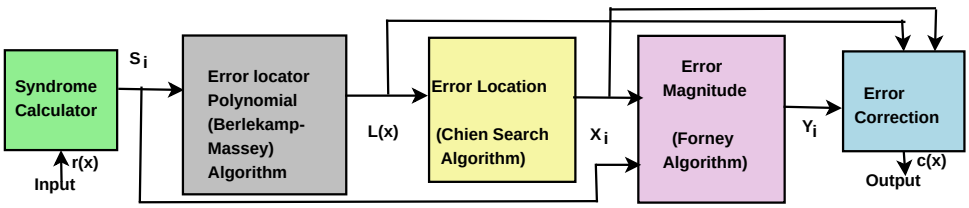

Figure 3.5: Different steps of RS decoding algorithm

Interleaver/De-Interleaver: Interleaving is the reordering of the data that is to be transmitted, so that the consecutive bytes of data are distributed over a larger sequence of data to reduce the effect of burst error. Generally two types of interleaving strategies (block and convolutional interleaver) are used in communication system. Here we have used block interleaver. Initially120 bit data from the encoder is divided into two block of 60 bits of data and then interleaving operation is done on each 60 bit data using block interleaver. The whole process increases the code correction capabilities without any clock latency and overhead. De-interleaver process is used to reorder the data again in the receiver side.

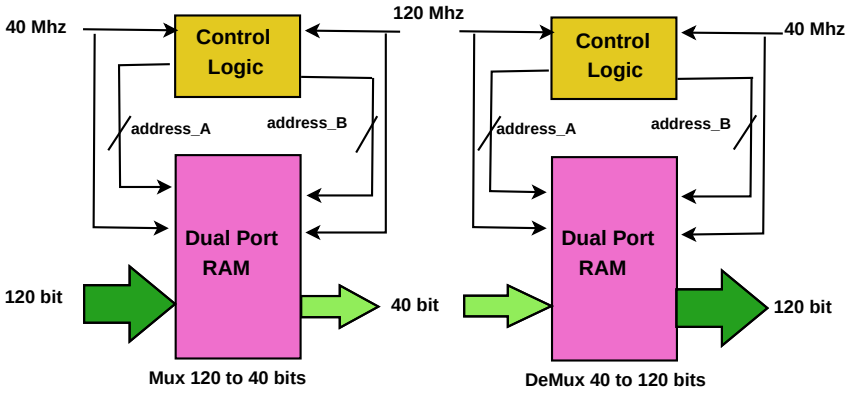

Figure 3.6: Functionalities of Gearbox

The gearbox module shown in Figure 3.6 consists of dual port RAM and read-write controller. In the transmission side TxGear-box breaks down 120 bit frame into three words of 40 bits width. Write frequency of RAM in TxGear-box is 40 MHz and read frequency is 120MHz. In the receiver side RxGear-box merge three words of 40 bit wide into 120 bit frame. Here the write frequency is 120 MHz and read frequency is 40 MHz.

Multi-gigabit transceiver (MGT): MGT is a high speed serializer and De-serializer (SerDes) that operates at the data rate above one gigabit per second. Like other SerDes MGT also helps to transmit parallel data bits as serial stream of data and convert serially received data into parallel data. Apart from serialization and deserialization some of the important functions of MGT in FPGA are elimination of crosstalk and electromagnetic induction, equalization, enhancement of signal to noise ratio by removing inter symbol interference, clock recovery, error detection, frame and phase alignment *etc.* Xilinx Kintex-7 FPGA uses two types of transceiver [107] GTX that supports data rate upto 12.5 Gbps and GTH that supports data rate upto 13.1 Gbps. In the Kintex-7 FPGA there are sixteen [107] GTX transceivers and in this design one of this GTX is used as transceiver. Four GTX (also known as GTXE2_CHANNEL) are grouped together along with one GTXE2_COMMON to form GTX Quad as shown in Figure 3.7. Each GTXE2_CHANNEL represents one transceiver and internal architecture of a transceiver is shown in Figure 3.7. Each GTXE2_CHANNEL consists of one PLL, transmitter and receiver. The PLL in GTXE2_CHANNEL also known as channel PLL (CPLL) uses ring oscillator in the voltage control

oscillator (VCO) and consume low power, small chip area and have wide tunable frequency range. CPLL provides clock used by both physical media attachment (PMA) and physical coding sublayer (PCS) blocks and can also be used by both Tx and Rx data path if both have the same data rate. When the higher clock speed and rigid jitter performance is required clock can be taken from output of QPLL in GTXE2_COMMON. It provides better jitter performance because here VCO is controlled by LC oscillator. Output of QPLL can be shared by multiple GTXE2_CHANNEL within the same quad.

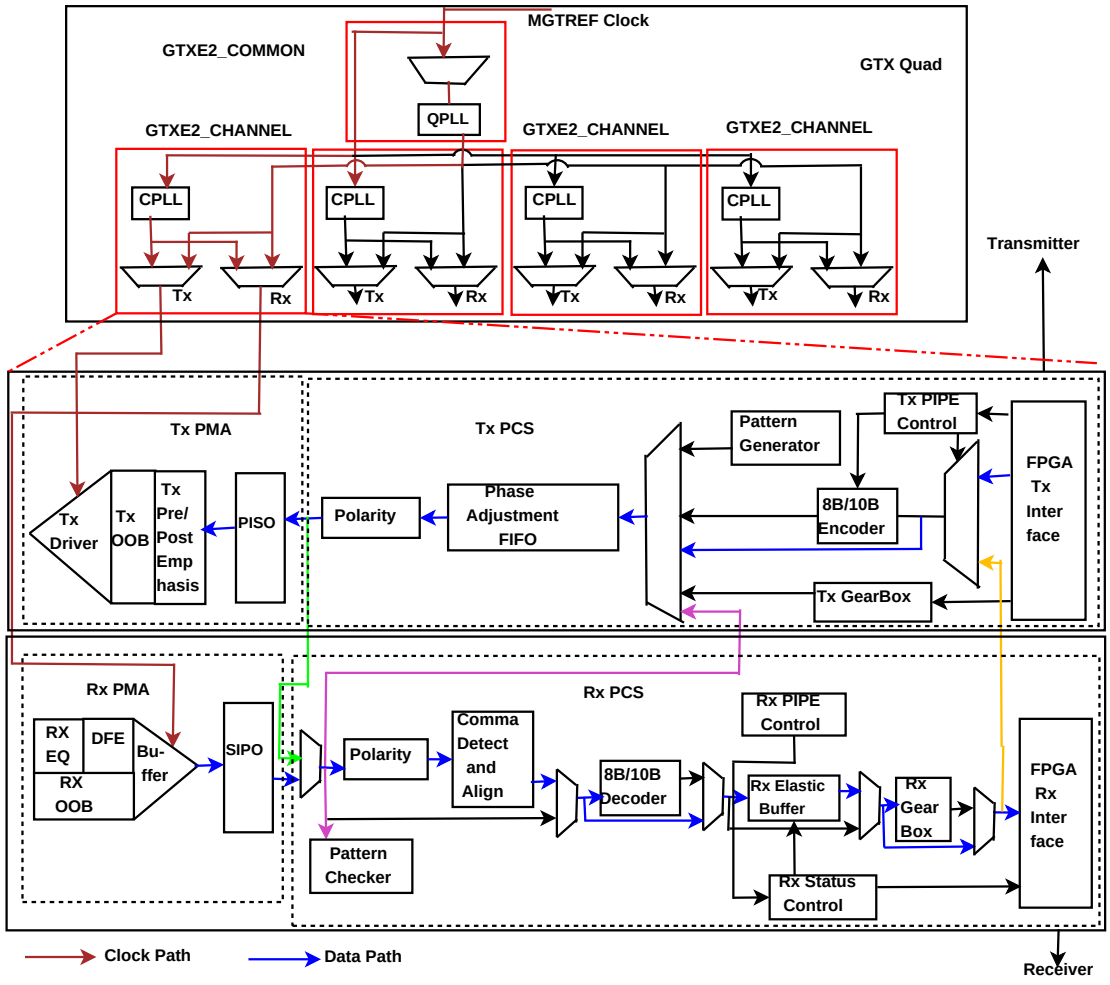

Figure 3.7: Architecture of transceiver of GBTx Emulator

FPGA Tx and Rx interfaces are the gateway for entry and exit of the data stream between logic block and transceiver. The data to be transmitted using

transceiver will be written into the TXDATA port at every rising edge of TXUSRCLK2 by properly selecting the internal bus width of the transmitter. In this design we have used external scrambler for line coding so we have disabled 8B/10B encoder and decoder within the transceiver. In order to drive the Tx PCS another external clock TXUSRCLK must be provided and frequency of which can be calculated by the equation 3.8.

$$F_{TXUSRCLK} = \frac{Line\ Rate}{Internal\ Data\ Path\ Width\ of\ Transmitter}; \tag{3.8}$$

PCS and PMA in the transmitter are running at two different clocks TXUSRCLK and XCLK respectively having different phase and frequency. During the data transmission phase and frequency mismatch between these two clocks must be resolved and for this purpose either Tx buffer or Tx phase alignment circuit must be used. Here we have used internal Tx buffer to eliminate the mismatch between XCLK and TXUSRCLK. In this design we have used external pattern generator having both fixed and dynamic value so internal pattern generator is bypassed. Polarity module helps to prevent accidental swap between data in TXP and TXN that may occur in high speed data transmission. TX preemphasis circuit is used to overcome low pass characteristics of the channel by giving higher boost to the high frequency components compared to lower frequency component.

In the receiver side received data will enter into the equalizer to eliminate the effect of distortion and attenuation introduced in the channel. There are two types of equalizer are available: power efficient adaptive mode equalizer named as low power mode (LPM) equalizer and decision feedback equalizer (DFE). Here we have used DFE because it provides better compensation for the channel loses by using closer adjustment of filter parameter. Filter coefficient of DFE are set by the adaptive algorithm. CDR circuit in each GTXE2_CHANNEL is used to extract clock and data from incoming data stream. As in transmitter side we have already used TXPOLARITY in the receiver side we have to use RXPOLARITY before sending data to comma detect and alignment block. Serial data must be aligned properly at the symbol boundaries before being converted to parallel data. For this purpose transmitter send a special character known as comma character and when the receiver receives the comma character it moves comma

character to data boundary so that adjacent word can be demarcated properly. Comma detect and alignment block performs this function in the receiver. Rx Elastic buffer is used to remove mismatch between frequency and phase of clock in RX PMA (XCLK) and RX PCS (RXUSRCLK) during data reception. In this design we have already used Gear-box externally to the transceiver as shown in Figure 3.6 and hence, Tx Gear-box and Rx Gear-Box are bypassed. In order to test the transceiver at different stages loop back facility are available as shown in Figure 3.7. Green line indicates Near-End PCS loop back, pink line indicates Far-End PMA loop back and yellow line indicate Far-End PCS loop back.

In the receiver side the frame aligner block aligns the 120 bit frames in a proper order by using frame header as an index. This frame header is detected by an efficient pattern search algorithm. Within GBTx Emulator there is options to generate three types of frame format as shown in Figure 3.8. The three type of frames are:

1. GBT frame which has 4 bit header, 4 bit for slow control, 80 bit data and 32 bit for forward error correction.

2. Widebus frame that consists of 4 bit header, 4 bit data for slow control and 112 bit for data. Here no correction is used.

3. 8B/10B frame which contains only 96 bit data and 24 bit redundant data bits are generated during 8b/10b line coding.

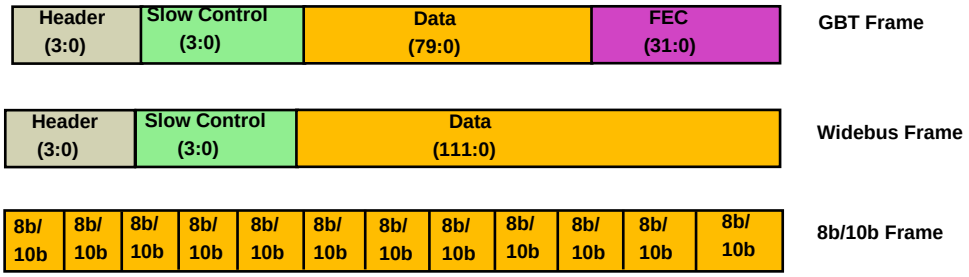

Figure 3.8: Different frame format for data transmission over optical link [104]

Any of the frame format can be used as per the designer requirement. As in the downlink DPB sends different control signal and timing information to MUCH-XYTER error in this data creates problem and hence, data will be sent using

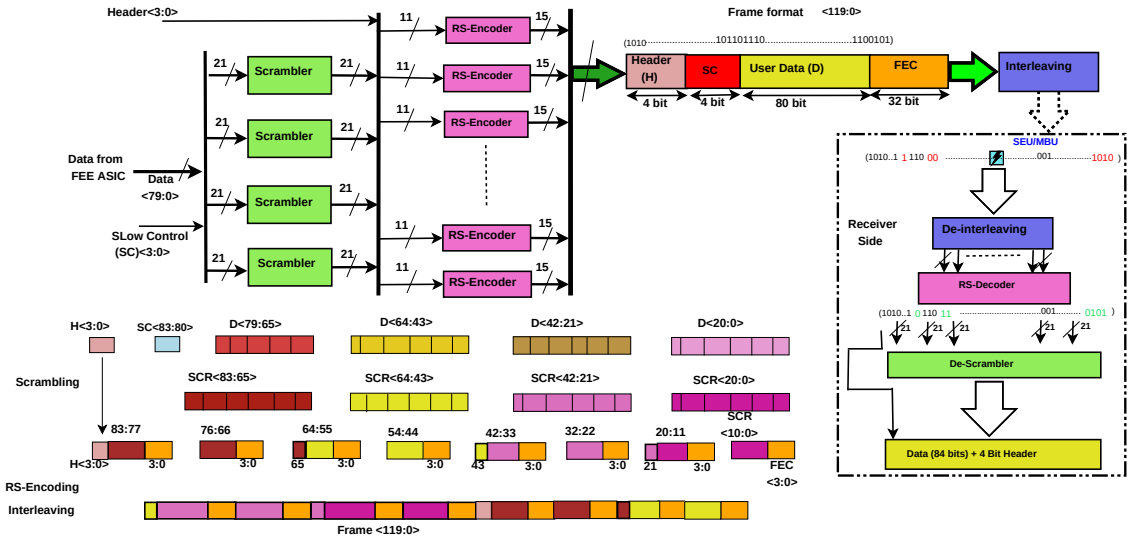

Figure 3.9: Steps of generation of GBT Frame during data transmission

GBT frame which is protected by (15,11) RS code. On the other hand in the uplink MUCH-XYTER sends continuous data to DPB and lose of some data packets may not hamper data analysis in the backend and hence data is sent using widebus frame format. Details of data flow for the generation of GBT frame is shown in Figure 3.9.

Front end ASICs are connected with the GBTx using differential electrical line. Signal amplitude over electrical line is programmable depending on length of the electrical line, BER and power consumption. GBTx chips are connected with the FEE ASICs using different topologies *i.e* one GBTx may be connected with one FEE or multiple FEEs may be connected with one GBTx that can be extended upto forty. Each E-link consists of three signal lines: clock, one uplink and one downlink as shown in Figure 3.2. E-link support variable data rate like 80 Mbps, 160 Mbps and 320 Mbps and the data rate will be chosen based on the number of E-link par group. Each E-link is terminated with built in 100Ω termination and if required can be eliminated. In order to eliminate path delay over E-link phase aligner circuit is attached with each differential line.

3.3 Muon Chamber X-Y Time Energy Readout ASIC

The MUCH-XYTER chip is dedicated for signal detection from the GEM based MUCH detector in the CBM environment. Some of the features provided by MUCH-XYTER chips are [108]:

- It has 128 analog front end channels for processing the charge pulses from the GEM detector. It also provides two test channels for testing the functionalities of the ASIC using test pulse.
- It is based on 180 nm CMOS process and uses radhard layout architecture.
- Power dissipation per channel is less than 10 mW.
- It can accepts both positive and negative pulses.
- Digital back-end with data transmission rate 320 Mbps.

Like most of the other front-end ASICs used in HEP experiment internal architecture of MUCH-XYTER ASIC can be divided into two parts: analog front end and digital back-end as shown in Figure 3.10. Each of the channel in the analog front-end consists of charge sensitive amplifier (CSA), shaping circuit, high speed discriminator, peak detector, flash ADC and latch circuit. In the digital back-end there is a sequencer that will read each of the front end channel sequentially after digitization. Details architecture of the analog front-end channel is shown in Figure 3.11. CSA in front of each channel receives the charge pulses directly from GEM detector for further processing. CSA circuit contains one operational amplifier with one resistance (R_F), three capacitances (C_{F1},C_{F2},C_{F3}) and reset circuit in feedback. CSA is mainly used to match the output impedance of detector and input impedance of FEB and it provides small gain to input pulses. By connecting or disconnecting the switches attached with each feedback capacitor, gain of CSA can be varied. Output charge from the detector is integrated by feedback resistance and capacitance and gives voltage waveform at the output of CSA. Then output of CSA is fed into a polarity selection circuit (PSC) that helps to work MUCH-XYTER with pulses of both polarity. Output signal of PSC is

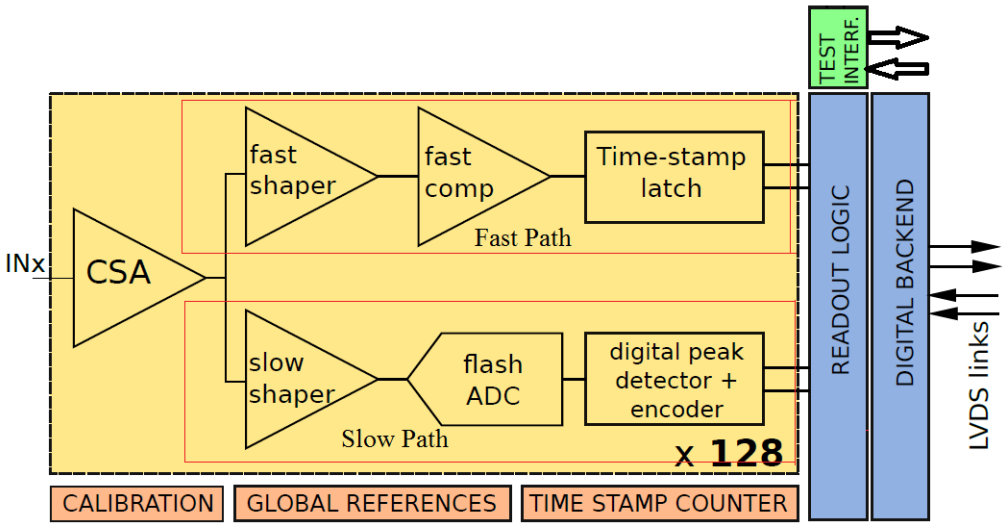

Figure 3.10: Internal architecture of each channel of MUCH-XYTER

divided into fast and slow path as shown in Figure 3.11. Fast path comprising of fast shaper, fast comparator and time stamp latch circuit and responsible to provide timing information of the detected particle. On the other hand slow path comprising of slow shaper, flash ADC and digital peak detector and provides energy of the detected particles. The CR-RC shaper in the fast path has shaping time of around 30 nanosecond. On the other hand slow shaper is based on $CR-(RC)^2$ filter architecture and has shaping time of around 80 nanosecond. As the slow shaper is responsible to provide energy information of the detected particles noise level should be low in this path.

First comparator has three stages of signal processing. In the first stage, it provides further amplification of output signal of the fast shaper and converts it into differential signal. The second stage contains dynamic comparator with regenerative feedback. The third stage converts discriminator output signal into logic signal. Each channel has twelve-bit time-stamp generator that gives the timing information of the detected particles. Flash ADC based peak detector which is connected with slow shaper using AC coupling in each of the 128 channel consists of thirty one comparators and thirty two resistors. It provides energy information of the detected charge particles. Details of the time stamp generation mechanism and function of the peak detector can be seen from [17].

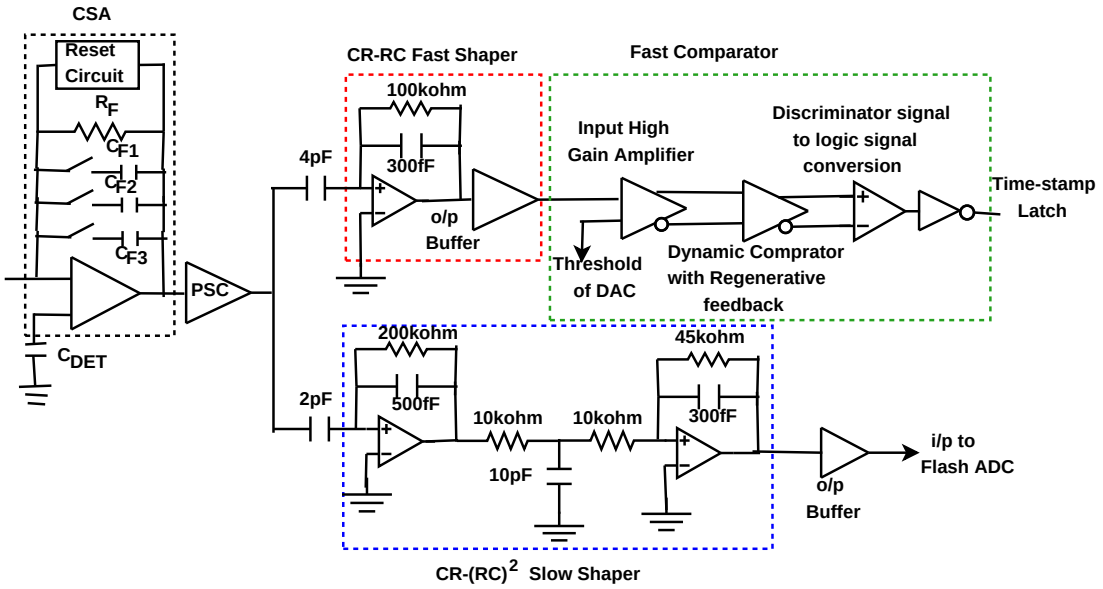

Figure 3.11: Details architecture of analog front end of MUCH-XYTER

MUCH-XYTER used here is a self triggered ASIC *i.e* no trigger information will be transmitted from the back-end computing node to the front-end ASIC and it will accept the charge from the detector when the voltage level corresponding to the detected charge is above a threshold. One of the main objectives of MUCH in CBM experiment is to detect muon particles (μ^+ and μ^-) generated from the J/ψ particles which are very rare in the density of other produced subatomic particles. MUCH in CBM experiment consists of multiple GEM detectors along with absorber in between the detectors [13] as shown in Figure 3.12 and each of the GEM detectors has separate readout chain though they have common back-end computing node. Particles generated from J/ψ particles hit multiple GEM detectors and when the hitted points are interpolated they form a straight line as shown in Figure 3.12. In this way muon generated from J/ψ particles are separated from muon comes from other sources. This acts like trigger in the experiment to remove the unnecessary data which may consume large memory space in storage device.

Digital back-end of MUCH-XYTER ASIC is shown in Figure 3.13. Data from the flash ADC and corresponding time stamp value from each of the 128 channels will be readout sequentially using arbitrator and sequencer. Multiple

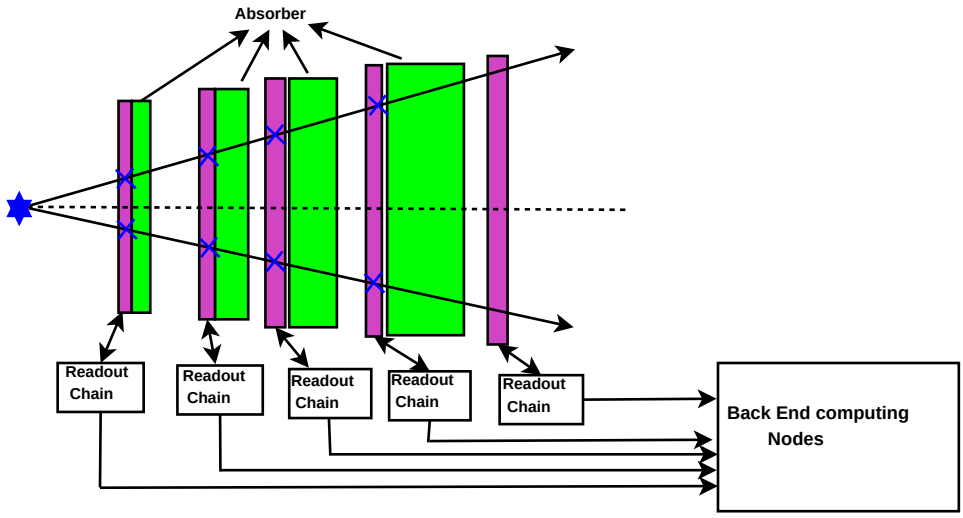

Figure 3.12: MUCH with segregated absorber and multiple GEM detector

serializers are there and will sequentially check the presence of data in each of the analog channels. Whenever they detect the presence of data, they will read the data from fifo and encode it using 8b/10b encoder, serialize it and transmit. Data from ASIC to DPB mainly contain time stamp value, energy information, channel Id, device address. There are different sixteen bit registers in the MUCH-XYTER corresponding to each of the front-end analog channel and by setting the values in the registers, different parameters of CSA, fast and slow shaper and ADC can be changed. Similarly there are some dedicated registers for digital back-end also and we can mask any E-link or change different properties of E-link by setting proper values in the register. Present MUCH-XYTER ASIC prototype has five uplink , one downlink and one clock line as shown in Figure 3.13. Downlink frame mainly contains different control information, acknowledgment frame, read and write address and write value. Details of uplink and downlink frame format have been discussed later in this chapter. Figure 3.14 shows the top view of the FEB that contains MUCH-XYTER ASIC and its 128 analog front channels, five back-end uplinks, one downlink and one clock line. Registers corresponding to analog front-end and digital back-end are arranged in row and column format and have been set for testing the ASICs as shown in Table 3.1 and 3.2. MUCH-XYTER ASICs [108] user's manual can be consulted for details description of the registers. Proper command are sent from DPB to set the register values.

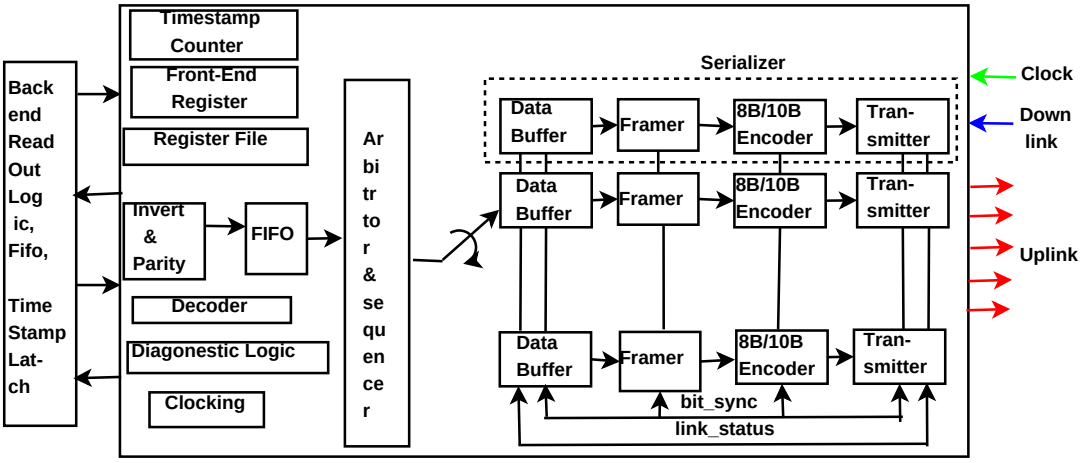

Figure 3.13: Digital back-end of MUCH-XYTER ASIC [108]

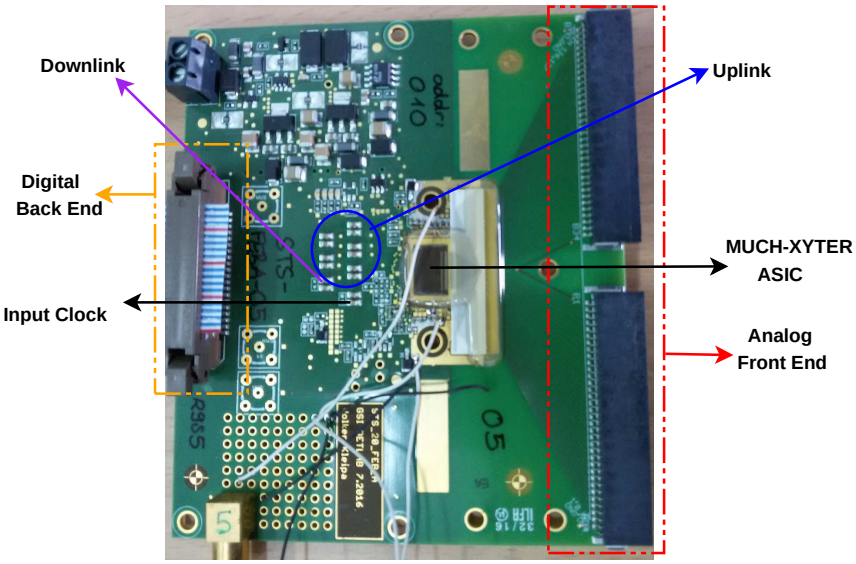

Figure 3.14: Top view of FEB containing MUCH-XYTER ASIC

Table 3.1: MUCH-XYTER Analog Front-end Register Description

Row	Column	Read/Write	Value used during testing
0-129	0,2,4,6,...,60	Read	return **8'b00001111**
0-129	1,3,5,7,...,61	Read	**8'b10000000**
0-129	62	Read	Discriminator counter
0-129	63	Read & Write	**8'b10011000,8'd152**
0-129	64	Forbidden	Not applicable
0-129	65	Read & Write	**8'b11100100, 8'd228**
0-129	66	Forbidden	Not applicable
0-129	67	Read & Write	**8'b00100100, 8'd36**
130	0	Read & Write	**8'b00011111, 8'd31**
130	1	Read& write	**8'b00000111, 8'd7**
130	2	Read& write	**8'b10010011, 8'd147** for -ve polarity **8'b10110011, 8'd179** for+ve polarity
130	3	Read& write	**8'b00011111, 8'd31**
130	4	Read& write	**'0'** as external pulse is used
130	5	Read& Write	**8'b00000000 when** 129^{th} **test channel is selected** **8'b00000001 when** 128^{th} **test channel is selected**
130	6	Read& Write	**8'b00100000, 8'd31**
130	7	Read& write	**8'b11011100, 8'110**
130	8	Read& Write	**Typical value:-8'b00011111, 8'd31**
130	9	Read& Write	**Typical value:-8'b00110000, 8'd48**
130	10	Read& Write	**Typical value:8'b10111100, 8'd188**
130	11	Read& Write	**8'b01100000, 8'd96**
130	12	Read & write	**8'b00011110, 8'd30**.
130	13	Read& Write	**8'b00011111, 8'd31**
130	14	Read& Write	**8'b00011011, 8'd27**
130	15	Read& Write	**8'b00011011, 8'd27**
130	16	Read& Write	**8'b01011000, 8'd78**

3.4 Data Processing Board

Data processing board (DPB) is an intermediate layer between GBTx and FLIB and are placed in the moderate radiation zone. Hence, it uses COTS FPGA and perform the following activities:

1. It receives data from GBTx using long distance optical link having the data rate of 4.8 Gbps. The received data is aggregated, preprocessed and sent to the FLIB using high speed (10 Gbps) optical link.
2. Timing and fast control (TFC) module in DPB provides reference clock for the whole read out chain and helps in the transmission of synchronous commands to the FEE. These are very helpful for synchronization between FEE and GBTx and implementation of flow control in the readout chain.
3. Control system attached with the DPB helps to configure different registers that helps in controlling the analog front-end and digital back-end of FEBs. The control system is attached with the DPB using some standard solutions like Ethernet with Transmission Control Protocol (TCP) or User Datagram Protocol (UDP).

Figure 3.15 gives the detailed firmware architecture of DPB that can be classified into four broad categories: IPBus module which acts like control system interface, GBT-FPGA interface for optical connection with GBTx emulator, TS slave interface for timing and fast control and First level interface module (FLIM) interface for ten gigabit connection between DPB and FLIB. As electrical connection in the back-end of MUCH-XYTER is unable to carry long distance data in the radiation zone, GBTx is used as simple interconnecting block between MUCH-XYTER and DPB. Hence, GBTx will not modify any frame sent by the MUCH-XYTER and it wraps the data packet within its own frame, convert it into optical signal and send it to DPB. GBT-FPGA core in DPB opens the wrapper and send it to MUCH-XYTER Data Readout and Processing block as shown in Figure 3.15. DPB is implemented as an advance mezzanine card (AMC) and placed into commercially available Micro Telecommunications Computing Architecture (MicroTCA.4) for Physics [109]. MicroTCA crate performs the following functions:

Table 3.2: MUCH-XYTER Digital Back-end Register Description for 192^{th} row

Column	Read/Write	Value used during testing
1	Read&Write	Store the 14 bit time stamp counter value. It is gray encoded
2	Write&Read	**14'b00000001111111** for reset **14'b00000000000000** for release of reset
3	Read&Write	**14'b00000000000010**
4-13	Read&Write	Value in each of the 14 bit register from 4 to 12 is**14'b00000000000010** and value in register 13 is **4'b1100**
14	Read&Write	store the time stamp value during the sync command
15	Write	Store value of time-stamp counter
16-17	Spare	Spare
18	Read & Write	**8'b00000010, 8'd2**
19	Read& Write	**11'b00000000000** Not used as only synchronization is done
20	Read& Write	**0** for all 128 channels as only test channels are used.
21	Write&Read	Monitor input reference(Not used)
22	Read	**"010"**
23	Write& Read	**9'b000000000**
24	Read& Write	**12'b000000000000**
25	Write&Read	**10'b00000000000**
26	Read&Write	store the address of previous register that was accessed
27	Read&Write	**11'b000000001010**
28	Read&Write	Not masked any status register
29	Write&Read	Threshold for event miss flag is set to zero
30	Read&Write	**12'b000000000000**
31-32	Read&Write	Not used here
33	Read	Not used here

1. MTCA.4 crate distributes high speed low jitter clock signals between multiple AMC cards placed into it which may be used as system reference clock.

2. MTCA.4 provides one Gbps Ethernet connection to each AMC board in the crate that helps to implement slow control using TCP/IP or UDP protocol.

3. MTCA.4 crate helps to connect multiple serial high speed links with the AMC board placed within it using AMC backplane connector or Rear Transition Module (RTM). AMC boards can carry multiple number of FPGA Mezzanine Cards (FMC) that help to implement various optical links.

4. It helps in broadcasting signal between multiple AMC boards via eight Multipoint LVDS (M-LVDS) lines at maximum clock frequency 100 MHz. These signals are used for exchange of status and control signals between multiple AMC boards and helps to implement inter AMC board communication.

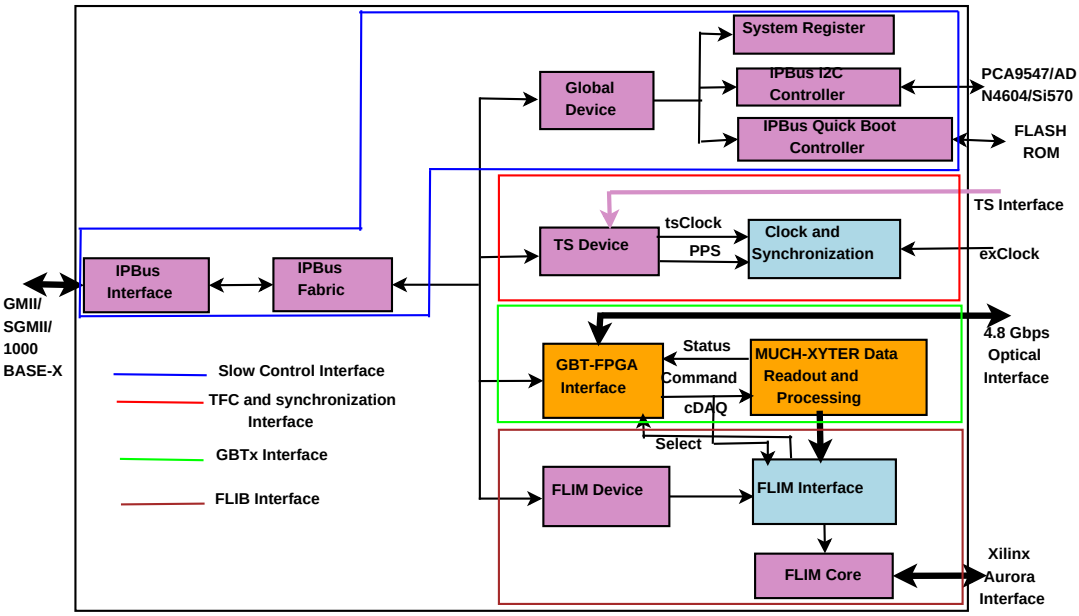

Figure 3.15: DPB Firmware structure

AMC FMC carrier Kintex (AFCK) board shown in Figure 3.16 is used as a prototype development of DPB. AFCK board contains [110] Xilinx Kintex-7 325T FFG900 FPGA, Module Management Controller (LPC1764FBD100), 2 GB DDR3 SDRAM with 32-bit interface, 16020 kb BRAMs for high speed parallel access, SPI Flash, electrically erasable programmable read only memory (EEPROM) with unique MAC ID, mini USB UART and two high pin count (HPC) slots where each slots can handle maximum four GTx. Clock distribution circuit within the AFCK is compatible with the White Rabbit protocol (WR) [111] using CDCM61004RHBT and Si57x chip and jitter cleaner available in it helps to reuse the recovered clock from receiver as the reference clock of transmitter. Different clock domains are available within the AFCK board like120 MHz or 40 MHz for GBT FPGA interface, 125 MHz for 1 Gbps Ethernet used in WR core and 156.25 MHz for 10 Gbps Ethernet.

Though AFCK board is developed to work with MTCA it can work in stand alone mode also using 12V power supply, JTAG programmer, Ethernet and optical interface. The clock generated from Si570 crystal oscillator can be tuned precisely either using SPI controlled SN74AVC8T245 [112] or I2C interface. Apart from the clock generated in the board itself recovered clock from the receiving

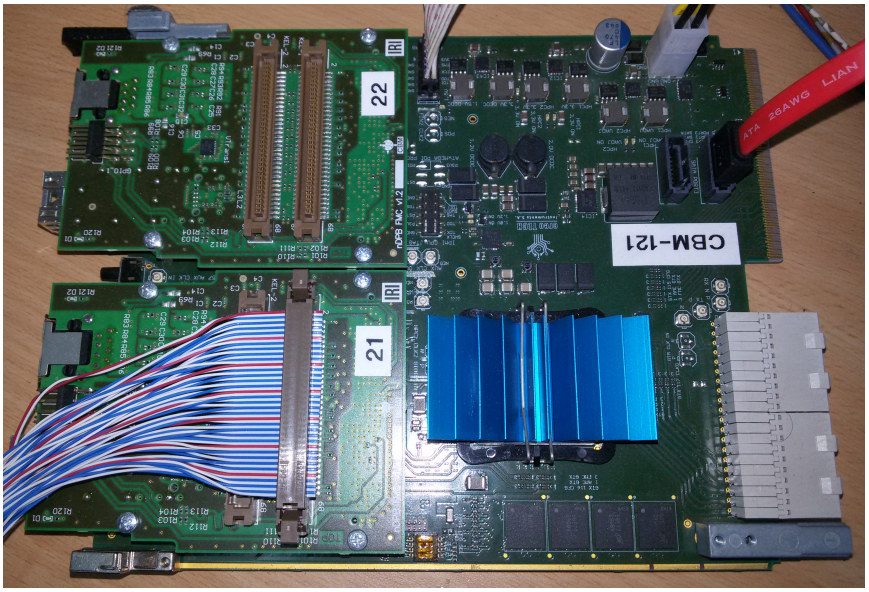

Figure 3.16: Top view of AFCK board

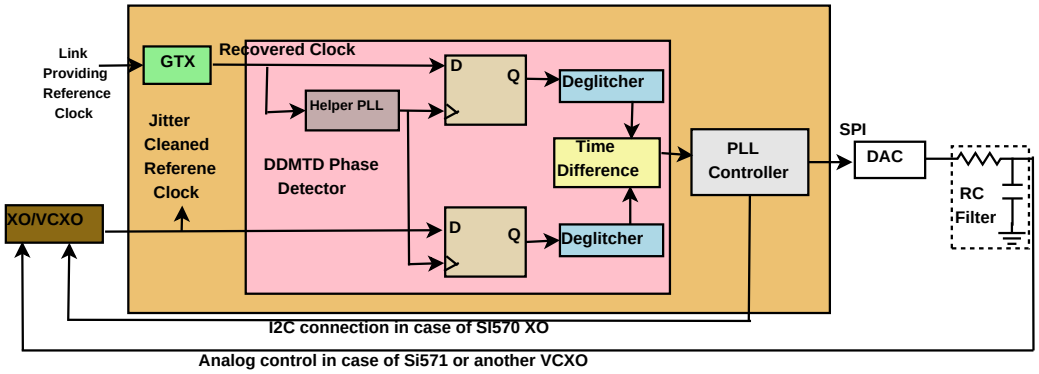

Figure 3.17: Clock Recovery and jitter cleaning circuit of AFCK

link can also be used to drive GTx as shown in Figure 3.17. Recovered clock is given input to the Digital Dual Mixer Time differences (DDMTD) block that allows recovery of high quality, jitter-cleaned clock. DDMTD block comprises of two D Flip-Flop, two deglitching circuit that removes the unwanted glitch, a time difference frequency counter circuit as shown in Figure 3.17. The glitches may occur due to the presence of phase noise in the timing signals or metastability arises due to setup or hold time violation in the Flip-Flop [113]. Output of the time difference circuit is given input to the PLL as an error signal that helps to adjust frequency and phase of recovered clock. The DDMTD module

also helps for precise phase alignment necessary for TFC functionalities. I2C controlled 16×16 digital cross point switch (ADN4604 [114]) is used for routing of the clock.

3.4.1 Communication with time and fast control

In the CBM experiment it is very important to provide high quality ultra jitter cleaned frequency locked clock signal to DPB, GBTx and MUCH-XYTER for timing synchronization. As MUCH-XYTER is a self triggered ASIC, there may be a chance of buffer overflow in the FEE ASIC due to high particle density in some part of the detector. In order to avoid buffer overflow, data must be transferred quickly to DPB with low and constant latency. TFC system responsible for synchronization of full readout chain using externally generated clock pulse is integrated with DPB which are placed within the MTCA.4 crate and equipped with AMC. TFC system uses master slave architecture where the FPGA based TFC master distributes high quality clock (pps pulse) to TFC controller in each MTCA.4 crate as illustrated in Figure 3.18. DPB within the crate acts as a TFC slave and TFC controller in each crate distribute the clock among the TFC slaves. WR core [115] with some modifications is implemented on AFCK for precise timing and synchronization. Like normal WR protocol the steps involved to achieve sub nanosecond accuracy are precise timing protocol, layer-1 synchronization and precise phase measurement. TFC master is connected to TFC slave using twisted pair cable with MLVDS termination in slave side. MLVDS termination provide 45-50% higher differential output voltage compared to LVDS termination that helps in long distance signal transmission. TFC master uses CDCE62005 IC as precise and low jitter clock source.

For establishing reliable timing synchronization first step is to calculate the link latency along both the directions :master to slave and from slave to master. In order to calculate latency TFC master sends a message to slave at t_1 and suppose it reaches to TFC slave at t_2. Then TFC slave generates a message at t_3 and master receives it at t_4. Assuming latency along upstream and downstream is same the one way latency can be calculated using Equation 3.9

$$\delta = \frac{(t_4 - t_1) - (t_3 - t_2)}{2} \tag{3.9}$$

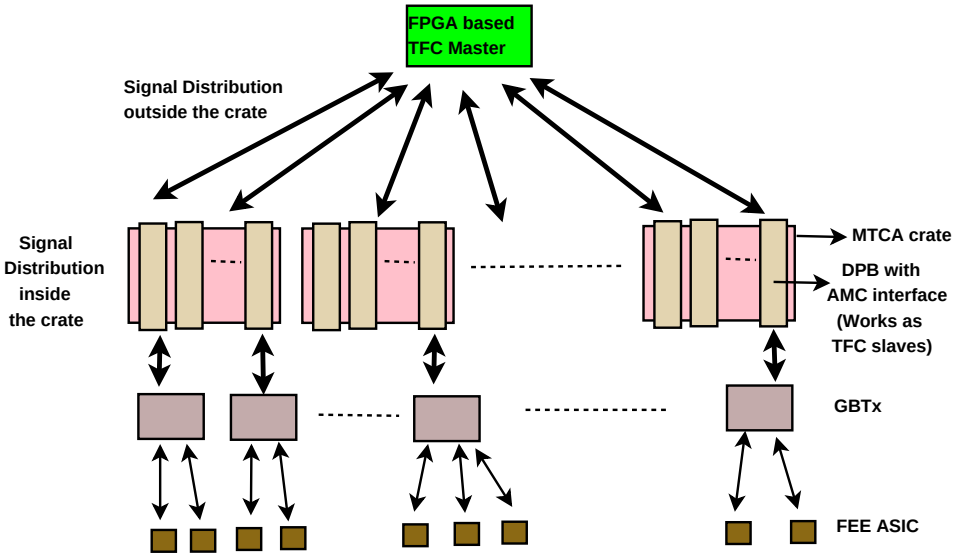

Figure 3.18: TFC system topology

Differential pairs in twisted cable have different ratio to reduce crosstalk between the pairs but it adds unknown skew to the propagation time of signals. Hence, the assumption that latency in upstream and downstream are same become invalid and special data center is chosen for TFC link to calculate latency in upstream and downstream. There is also a small delay of about 4 to 5 nanosecond among four differential twisted cables. Internal architecture of TFC master is shown in Figure 3.19. Out of the four differential cables one pair is used for latency calculation as mentioned above, one pair is used for transmitting a high-quality clock and two pairs are used for transmitting fast control message. Unlike to the WR protocol, here clock is not recovered from the serial data stream so the clock and data do not have known phase relationship. Here a customized WR soft PLL approach [116] is implemented that uses clock link for frequency synchronization and timing link (link for latency calculation) is used for phase locking between TFC slave and master.

3.4.2 Communication through slow control interface

Communication link between AFCK board with control system is implemented using IPbus protocol over one Gbps Ethernet link. Using the IPbus protocol, GTX transceiver in AFCK board can be operated in different modes and val-

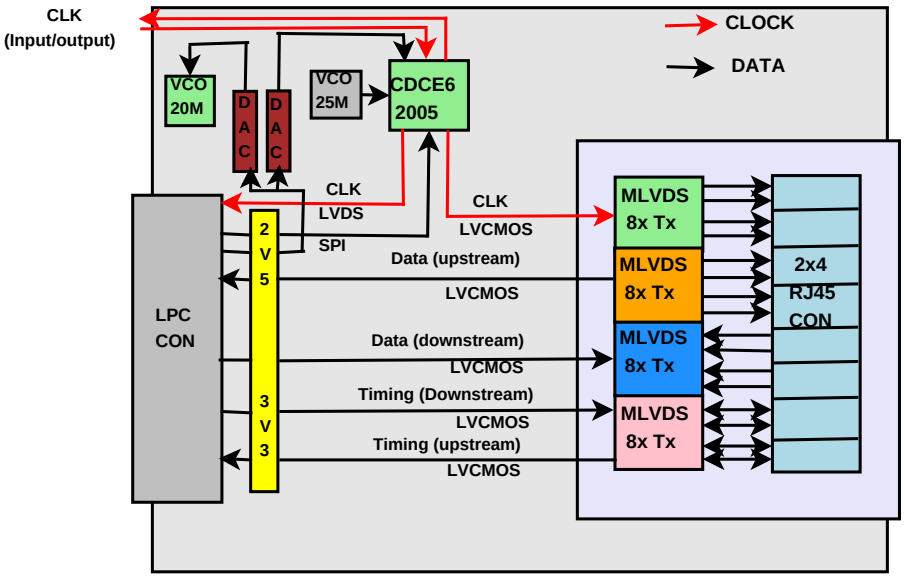

Figure 3.19: Internal architecture of TFC master prototype [116]

ues and different internal registers can be read out that help in monitoring and controlling the readout chain.

IPbus is a simple, Internet Protocol (IP) based control protocol [117]. It describes basic process to control the hardware. Figure 3.20 shows different protocols of standard OSI model [118] used during the implementation of IPBus protocol. Out of the seven layers in the OSI model only Transport, Network, Data Link and Physical Layer are used during the implementation of IPBus protocol and registers to be read or write will be placed in the application layer.

Physical layer provides electrical and physical properties of the data in the channel and comprises of PCS, PMA and Physical Medium Dependent (PMD) sublayer as shown in Figure 3.20. Here, physical layer is connected with the copper medium using BASE-T PHY [119]. Data Link Layer (DLL) provides reliable transmission of data packets between two nodes connected by a physical medium. Media access controller (MAC) within the DLL is responsible to provide identity of the device in the network and earn the permission to transmit data in the network. Logical link control (LLC) in DLL helps to detect network protocol and accordingly prepare the data packets, control frame for synchronization and error checking. In our design we have used IEEE 802.3 Ethernet protocol in the DLL. Ethernet has two frame format [120]: Standard and Virtual Local Area

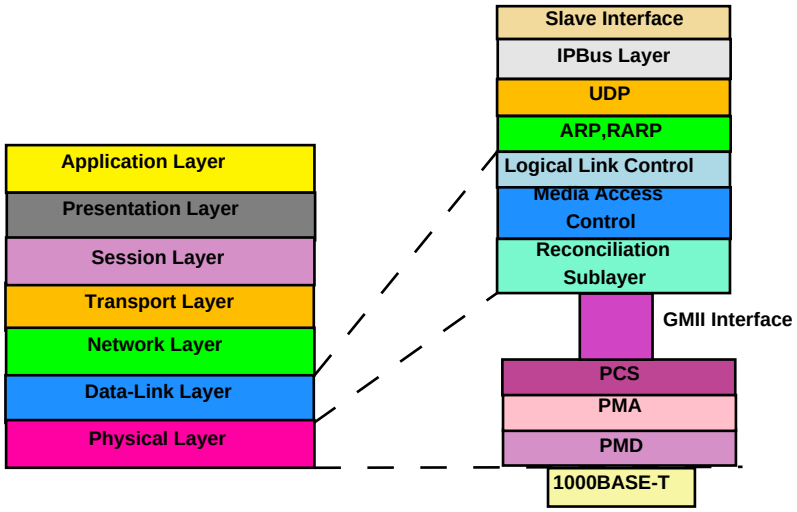

Figure 3.20: Implementation of IPBus protocol using standard OSI model

Preamble (7)	Start of Frame Delimiter (SFD) (1)	Destination Address (6)	Source Address (6)	length/ Type (2)	Data (0-1500)	Pad (0-46)	FCS (4)

Figure 3.21: Standard Ethernet Frame Format

Network (VLAN) frame format. In this design we have used standard Ethernet frame format as shown in Figure 3.21.

Network layer is responsible for forwarding data packets through proper routing and it manages the traffic in the network. Every devices in the network has two addresses: Physical or MAC address and logical address or Internet Protocol (IP) address. MAC address is assigned by DLL and fixed for a device. IP address is assigned to a device from a hierarchical system in the network and change with time and position of the device in the network. Network layer maps from physical address to logical address and vice versa. It also prepares a routing table that helps data packet transmission in the network. It breaks a large packet into smaller packets and helps in flow control, network layer error control and packet sequencing. Message delivery in the network layer is not reliable and depends on the protocol used in this layer. Here we have used IPv4 in this design. It is a connectionless protocol and specially designed for packet switching network. It takes data from trasport layer and fragmented into packet and encapsulate the data packets with its own header format. In order to properly deliver a data packet

to destination there should be a mechanism for mapping between MAC address and IP address. The Address Resolution Protocol (ARP) maps IP address to MAC address and reverse address resolution protocol maps MAC address to IP address.

Transport layer protocols provide point to point communication between processes running on two different systems. Transport layer ensure that the data should reach the destination in proper sequence and remain intact. At the same time it ensure the error and flow control in both source and destination. Commonly used transport layer protocols are TCP, UDP, Datagram Congestion Control Protocol (DCCP), Stream Control Transmission Protocol (SCTP), Resource Reservation Protocol (RSVP) etc. Here we have used UDP protocol over IPv4 and is referred as UDP/IP. Though TCP is most popular protocol in transport layer UDP is designed for low latency and loss tolerating communication system. UDP provides two services to the user: port number that distinguishes different user requests and checksum to detect the data integrity. Different features of the UDP protocol are:

- UDP does not used any acknowledgment frame so data packet may be lost. UDP does not have the facility for checking and resending the lost packets. But it reduces overhead, bandwidth and latency.
- UDP is simple and suitable for query based communication.
- UDP is connectionless protocol and does not have congestion control mechanism.

UDP can also be used for loss less application where other application can manages retransmission of lost packet and rearrange the correctly receive packet. In our application Ethernet based communication is used only for controlling purpose and state machine written in application layer control the frame rearrangement. At the same time there is other mechanism for data retransmission and CRC is used to detect erroneous frames. For this reason we have used UDP protocol instead of TCP protocol in transport layer.

FPGA implementation of IPBus over One Gigabit Ethernet: During the FPGA implementation of IPBus protocol we have used different algorithms

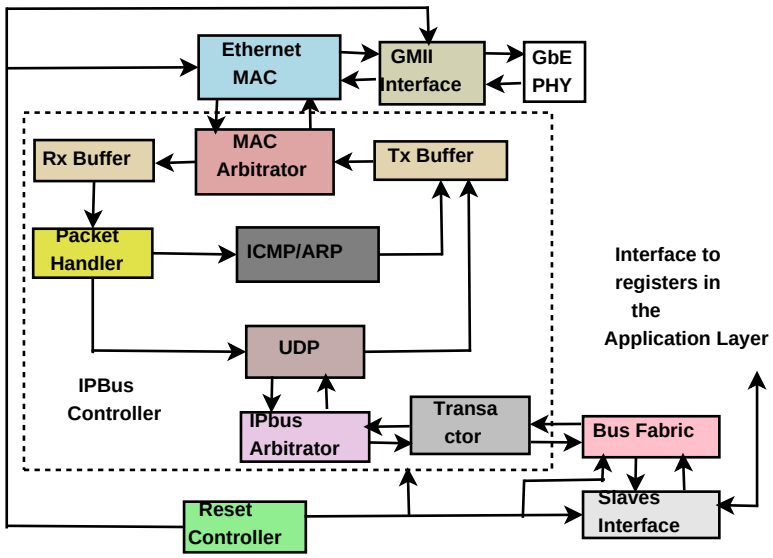

Figure 3.22: Architecture of IPBus controller and its interfacing with registers

of standard OSI layer. IPbus core has different features like reliable data transfer with UDP, simple external interfaces and remote bus access. Gigabit media independent interface (GMII) is used as interface between physical layer (PHY) and MAC in data link layer. Application layer contains the registers whose value will be set in the DPB. IPbus controller is implemented under the assumption that other communicating partner must be an intelligent device like computer. As all control and management program will run in the computer it will set the values of the registers via Ethernet. The internal architecture of IPBus controller and interfacing with registers and external port is shown in Figure 3.22. IPBus controller consists of MAC arbitrator block, Rx and Tx buffer, ICMP/ARP, UDP and arbitrator block. In the data link layer and physical layer Ethernet protocol is used. But instead of using trimode ethernet MAC available from xilinx we have used one simple custom MAC that supports data rate of one gigabit per second. FPGA based UDP/IP stack implementation is shown in Figure 3.23. There are mainly three clock domains within the UDP/IP stack : IPB_ CLK used as common clock within the IPbus controller, Tx_CLK used in the transmitter side and Rx_CLK used in the receiver side.

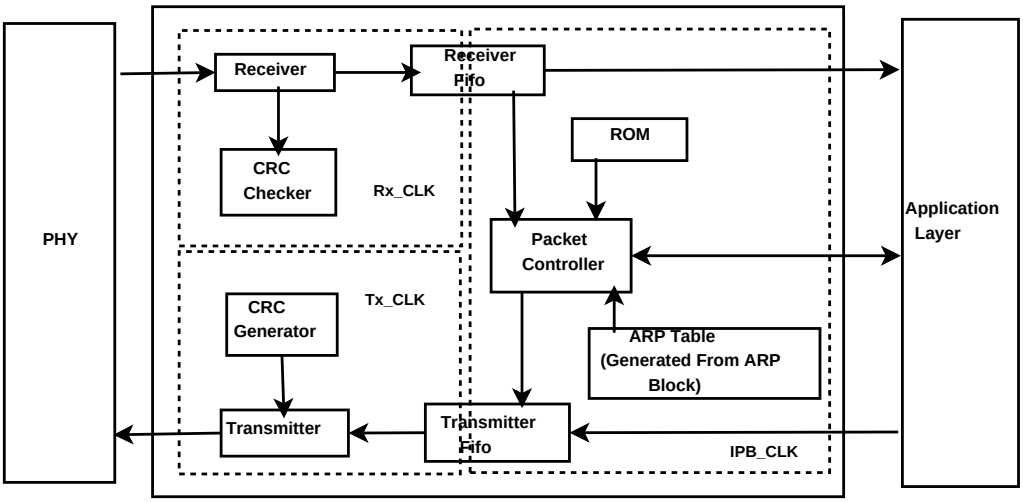

Figure 3.23: Architecture of FPGA based UDP/IP stack

Receiver Block: There are two finite state machines (FSM) in the receiver block. One FSM writes the receive packet in the dual port fifo and second one reads it and send to the Arbitrator block. Data from Arbitrator block will go to the slave (FPGA registers) connected through the bus fabric as shown in Figure 3.22. Received packet will be saved in the receiver fifo in byte wise fashion and at the same time received packets will be sent to the CRC checker to calculate the checksum. If the calculated checksum is matched with the data in the CRC field of the received packet then the packet will be retain in the fifo otherwise it will be deleted from the fifo. After receiving the last byte from the Ethernet PHY, MAC address verification process starts and if the received MAC address match with the machine MAC address then received frame will be processed further otherwise it will be discarded.

Transmitter Block: Similar to the receiver, transmitter also contains two FSM. One is responsible for reception of packets from Arbitrator block and write to the dual port RAM and other one receives packets from the dual port RAM and transmits them via Ethernet PHY. Transmitter will first send the preamble with start of frame (SOF) delimiter as the last nibble. At the same time each byte will be sent to the CRC generator. Generated checksum will be appended with the packet. At the end of the packet End of Frame (EOF) will be sent. Here

32 bit CRC polynomial is used as CRC generator as given by equation 3.10

$$G(x) = x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x+1; \quad (3.10)$$

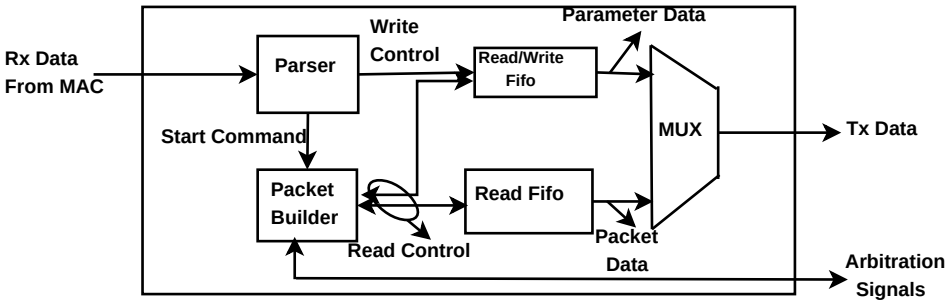

Figure 3.24: Internal Architecture of ARP block

ICMP/ARP Block: This block processes packets related to ARP and ICMP request. ARP request is essential because it resolves IP address related issue and ICMP packets are only needed during Ping process. Figure 3.24 shows internal architecture of ARP/ICMP block. Here the parser block works like a filter because it processes the packets that have ARP request tag. Then the packet is written into the read/write fifo and packet builder sends a request for generation of packet to the arbiter block. After receiving the acknowledgment from arbiter block packet builder starts to generate packet taking data from read/write fifo and some fixed header value from read fifo. Data packet generated using IPBus protocol is shown in Figure 3.25.

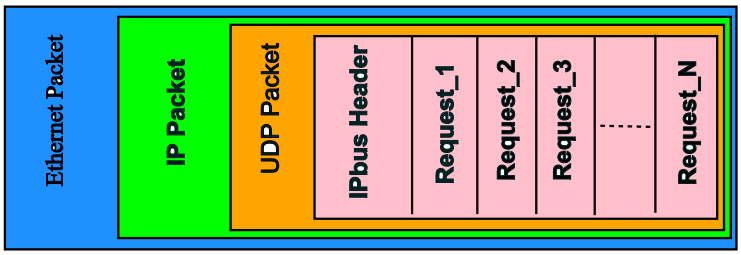

Figure 3.25: Packet format generated using IPbus protocol

3.5 Integration of DPB with MUCH-XYTER using GBTx Emulator

In the DPB, the GBT-FPGA interface module as shown in Figure 3.15 helps to connect AFCK board with GBTx Emulator which in turn is connected with MUCH-XYTER using differential E-link. Communication over the E-link is synchronous because there is separate clock line in the digital back-end of MUCH-XYTER and clock is not recovered from the data stream as shown in Figure 3.13. On the other hand communication between GBTx and DPB is asynchronous and CDR circuit in the transceiver recovered the clock from the data stream itself. In a GBTx emulator there is multiple GBT banks as discussed in section 3.2. Figure 3.26 gives simplified architecture for interfacing of MUCH-XYTER with DPB using single GBT bank. Tx encoder of the MUCH-XYTER sends data at

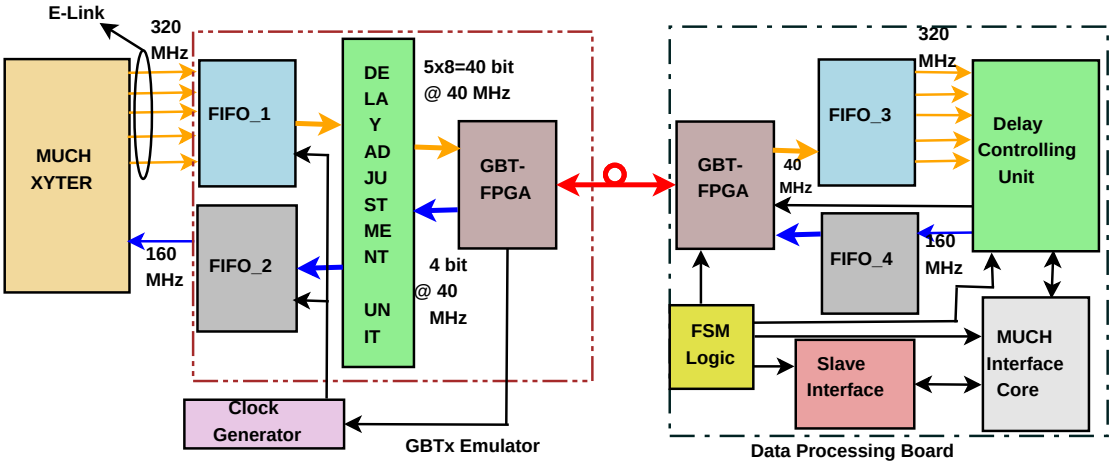

Figure 3.26: Interfacing of MUCH-XYTER with DPB using GBTx Emulator

320 Mbps and Rx decoder receives data at 160 Mbps. Received data within the MUCH-XYTER is decoded by 8B/10B decoder. Though 8B/10B coding add 20% extra overhead, due its simple decoding structure, less resource consumption it is used for line coding instead of scrambling. After decoding data bits are put into a shift register where decoded data are analyzed in parallel by special sequence detectors (EOS, SOS, K28.1, K28.5). GBT core is running at 40 MHz so in the receiver of GBTx Emulator there will be a FIFO (FIFO_1) whose write frequency is 320 MHz and read frequency is 40 MHz. Similarly, in the transmis-

sion side there is also a FIFO (FIFO_2) whose read frequency is 160 MHz and write frequency 40 MHz. GBT core receives data in 8-bit chunk at 40 MHz clock and a single MUCH-XYTER can handle five uplink. Hence, width of the data read from FIFO in receive direction is 40 bit. On the other hand GBT core writes 4 bit chunk at 40 MHz clock in FIFO_2. In DPB side write frequency of FIFO_3 is 40 MHz and read frequency is 320 MHz. Similarly, write and read frequency of FIFO_4 is 160 and 40 MHz respectively as illustrated in Figure 3.26. Clock for the write operation of FIFO_1 and read operation of FIFO_3 must be in same phase. Similarly, clock for the read operation of FIFO_2 and write operation of FIFO_4 must be in the same phase.

In order to maintain the phase of 160 MHz and 320 MHz clock a phase adjustment unit is placed within the MUCH interface core. As one MUCH-XYTER is connected with five uplink there may be path delay between the data transmitted over the E-link. Automatic delay controller attached with each E-link in GBTx Emulator is used to eliminate path delay in each E-link and they are controlled by delay controlling unit in DPB. MUCH-XYTER interface core mainly helps in de-

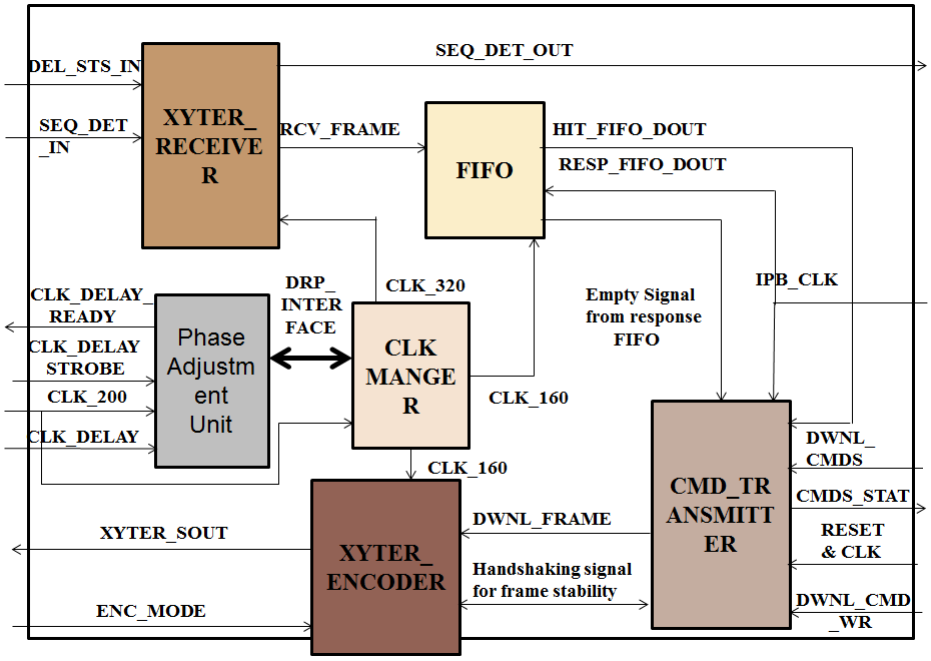

Figure 3.27: Internal Architecture of MUCH Interface core

coding/encoding of data for communication with MUCH-XYTER, clock manage-

ment, transmission of command, phase adjustment *etc.* as shown in Figure 3.27. Data and special characters from delay controlling unit enters into the MUCH interface core through DEL_STS_IN signal and SEQ_DET_IN respectively and decoded by XYTER_RECEIVER. Decoded data will be written into the FIFO and special characters will be sent to slave interface through SEQ_DET_OUT. Slave interface module as shown in Figure 3.26 contains IPBus interface module that is used to connect with computer through one Gbps Ethernet. Python script calculates the path delay after getting the decoded sequence and send the proper delay parameter and command signals to the command transmitter module. Command transmitter send the command and delay parameter to XYTER encoder which transmit it after encoding the data using 8B/10B encoder. Phase adjustment unit adjust the phase of the clock generated from clock manager using the signal CLK_DELAY_READY, CLK_DELAY_STROBE, CLK_DELAY signals. These signals get their values from the python script using IPBus interface in the same way as delay controlling unit get delay parameter.

GBTx encodes the data from the uplink using its own frame format, GBT-FPGA core in DPB decodes it and send it to delay controlling unit. Hence GBTx is simply acts like a black box that helps to carry data from FEE to DPB and DPB to FEE. Frame width from FEE to DPB is 30-bit and there are mainly six types frames are available for uplink. Uplink frame contains hit data, control response (register value, acknowledgments etc) and status data (Timestamp MSBs (TS_MSB), status bits etc) as shown in Figure 3.28. Hit data frame contains eight LSBs of time stamp (plus two additional bits overlapping with remaining time stamp part), seven bit channel address, five bit ADC value and one bit for event missed flag (EMF). TS_MSB data frame contains six most significant time stamp bits triplicated and four bit for CRC. Two types of acknowledgment frame are used here: ACK frame and RDdata_ack. Acknowledge (ACK) frame contains two bit for acknowledgment status, four bit for sequence number for identification of acknowledged command, four status bit for diagnostic purpose, one configuration parity (CP) bit for detecting SEU in configuration registers, six bit for LSB of time stamp and four bit for CRC. Acknowledgment for read data (RDdata_ack) frame has fifteen bit payload which is required to read back from a particular register, three bit sequence number and four redundant bit for

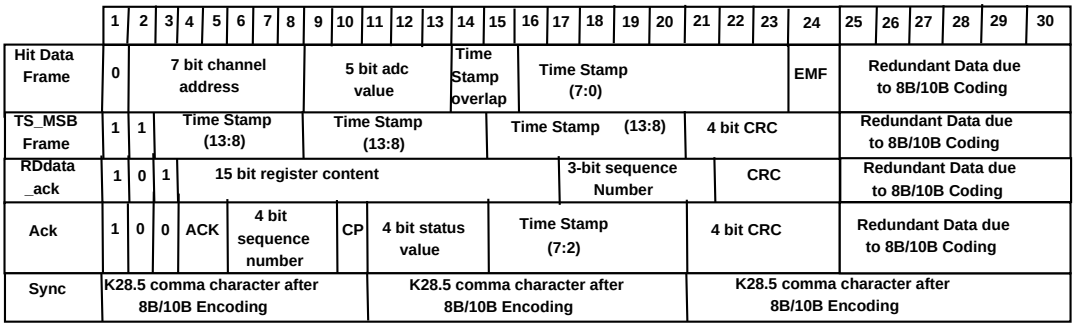

Figure 3.28: Uplink frame format after 8B/10B encoding

CRC. Sync frame contain three consecutive K28.5 comma characters at the rate of 166 MHz to maintain the link in synchronized mode. If DPB does not get these characters in periodic interval E-Link may resynchronized. All the above mentioned frames are 8B/10B encoded and contain extra six redundant bits due to this line coding as illustrated in Figure 3.28. Apart from the above mentioned frame there is another frame called Dummy Hit frame. It is used to keep link synchronized when nothing is to be transmitted and does not contain any ADC and time stamp value. Frame width from DPB to FEE is 60-bit and there are

Frame_bits (59:50)	Frame_bits (49:40)	Frame_bits (39:30)	Frame_bits (29:20)	Frame_bits (19:10)	Frame_bits (9: 0)
Comma Character K28.5	Chip Address(7:4) Sequence Number(3:0)	Request Type (7:6) Payload (14:9)	Payload (8:1)	Payload(0) CRC (6:0)	CRC(14:7)

Figure 3.29: Downlink frame format after 8B/10B encoding [108]

mainly four types of frames are available in downlink: No_op, WRaddr contains address of the register to be accessed, RDdata for reading data from previous address and Wrdata contains data to be written. Down-link frame structure is shown in Figure 3.29 and it mainly contains acknowledgment signal, some special signals like DAQ_start, DAQ_stop, Sync, read and write address and data to be written. This frame is protected by fifteen bit CRC. The data to be sent using down-link frame is placed in payload of the frame.

Before data transmission some special characters like K28.1, start of synchronization (SOS) and end of synchronization (EOS) are required to be exchanged between DPB and FEE for synchronization. FEE after power on reset continu-

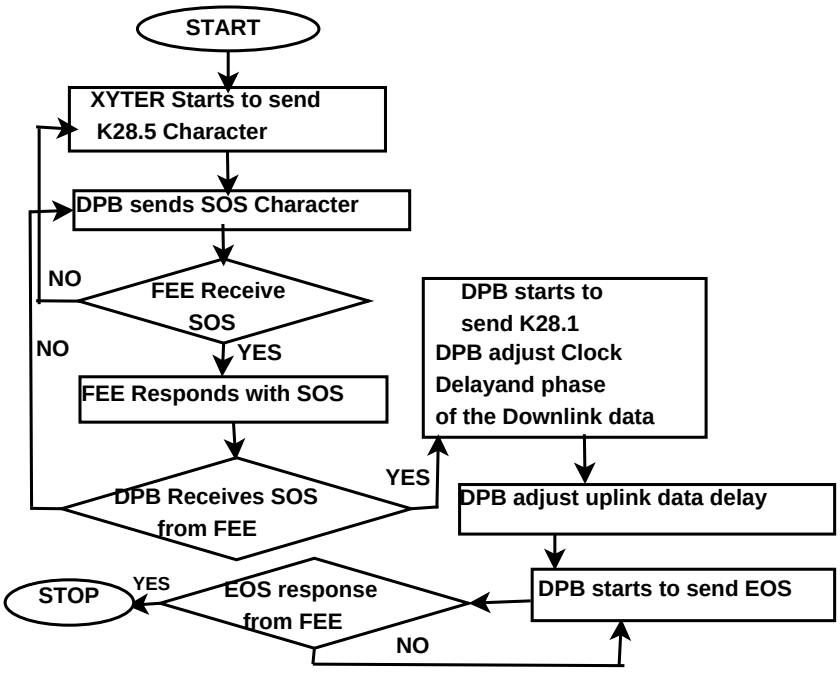

Figure 3.30: Flowchart for synchronous communication over E-Link

ously sends K28.5 characters in the uplink direction and DPB responds with SOS characters. MUCH-XYTER after receiving SOS characters responds with SOS character and DPB then starts to send K28.1 characters. After that FEE will respond by sending K28.1 characters continuously. Based on the response from the FEEs phase of the data and clocks will be adjusted. Finally, DPB sends EOS to FEEs and synchronization process will complete. Details of the synchronization is shown using the flowchart 3.30.

3.6 Results and Performance Analysis

As mentioned there are some specifications for designing the readout chain and we have used certain methods to check whether the design has meet that specifications. The methods are as follows:

- Provide voltage pulse at the input of each channel of FEE ASICs and then convert the voltage pulse to charge using input capacitor of preamplifier to check the functionalities of the ASICs.

- Characteristics of different modules in FEE ASIC like shaper, preamplifier, ADC are tested by varying different register values for analog front end and digital back end of FEE ASIC.
- Functionalities of GBTx and DPB are tested using low cost FPGA with high logic resources.
- Different error detection and correction codes are used for error mitigation in the user data through communication channel.
- UDP based IPbus protocols are used for monitoring of internal register values of ASICs and FPGA boards.

We have the provisions to choose FPGA from Artix-7, Kintex-7 or Virtex-7 for implementation of the prototype of DPB and GBTx Emulator. Virtex-7 is quite costly compared to Kintex-7 but it has much higher logic resources than Kintex-7. On the other hand Airtex-7 is cheaper than Kintex-7 but has less logic resources than Kintex-7. Optimizing logic resources and cost Kintex-7 FPGA is chosen to implement DPB firmware and GBTx Emulator. DPB firmware and GBTx Emulator are implemented on AFCK board and Xilinx Kintex-7 evaluation boards (KC705 from Avnet) respectively. In order to test different characteristics of MUCH-XYTER ASIC we have interfaced it with DPB using E-link as shown in Figure 3.31 without GBTx emulator. As mentioned in Table 3.1, we have set dif-

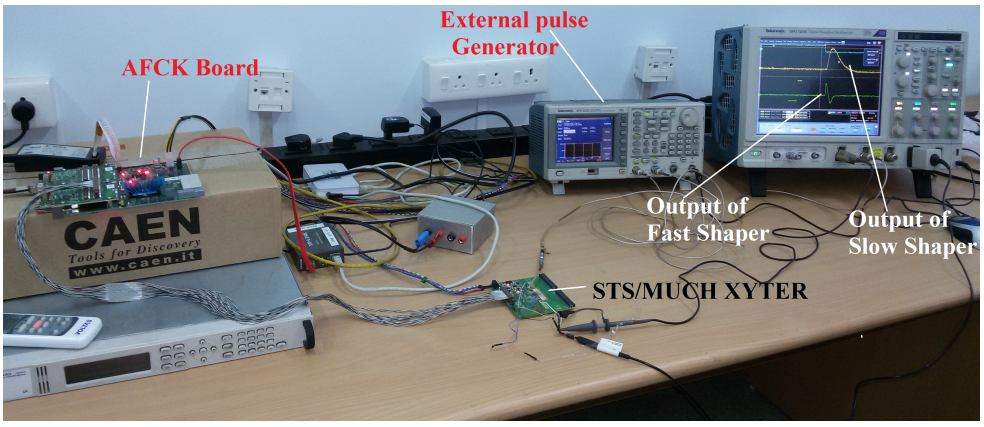

Figure 3.31: Setup for testing MUCH-XYTER using DPB

ferent register values to change different parameters of analog front end. We have

given test pulse of frequency 100 Hz from function generator into the test channel and vary the amplitude from 50 mV to 400 mV. Figure 3.32 shows input/output characteristics of fast shaper for various feedback capacitances of CSA. There are three feedback capacitances having values of 20fF, 20fF and 500fF with 80 fF as offset capacitances. As we have used 129^{th} test channel, 5^{th},4^{th} and 3^{rd} bit of register of MUCH-XYTER at the 129^{th} column and the 65^{th} row are used to select specific feedback capacitance. We have tested for both positive and negative pulses by selecting polarity in register at the 130^{th} column and the 2^{nd} row. '0' in this register indicates negative polarity and '1' in this register indicates positive polarity. Similarly, we have tested the slow shaper also and input/output charac-

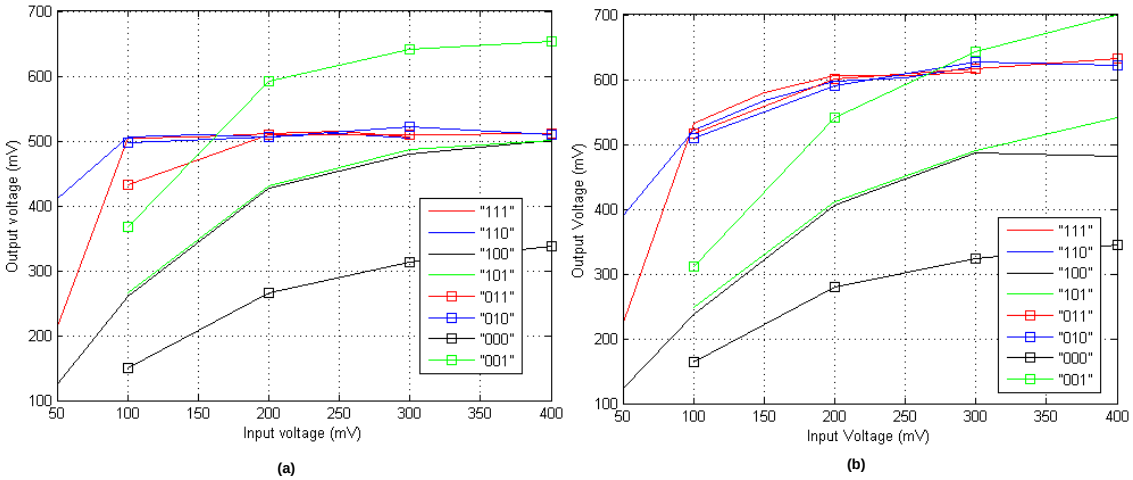

Figure 3.32: (a) Fast shaper output with positive pulse (b) negative pulse

teristics are plotted in Figure 3.33. It can be observed from Figure 3.32 and 3.33 that for same feedback capacitance, fast shaper saturates quickly compared to slow shaper. As for example, when the CSA feedback switches are at "001" with positive test pulse, fast shaper starts to saturate at 200 mV whereas slow shaper even does not start saturation at 400 mV. Figure 3.34 shows the input/output characteristics of slow shaper for pulses of both positive and negative polarity for 600 fF (equivalent to switch position "000") and 80fF (equivalent to switch position "111"). It is clearly seen from the result that when the value of the feedback capacitance is low, it saturates quickly (at around 250 mV) compared to the situation when the feedback capacitance is at highest value (saturation does not come at 400 mV). We have also measured gain of slow and fast shaper

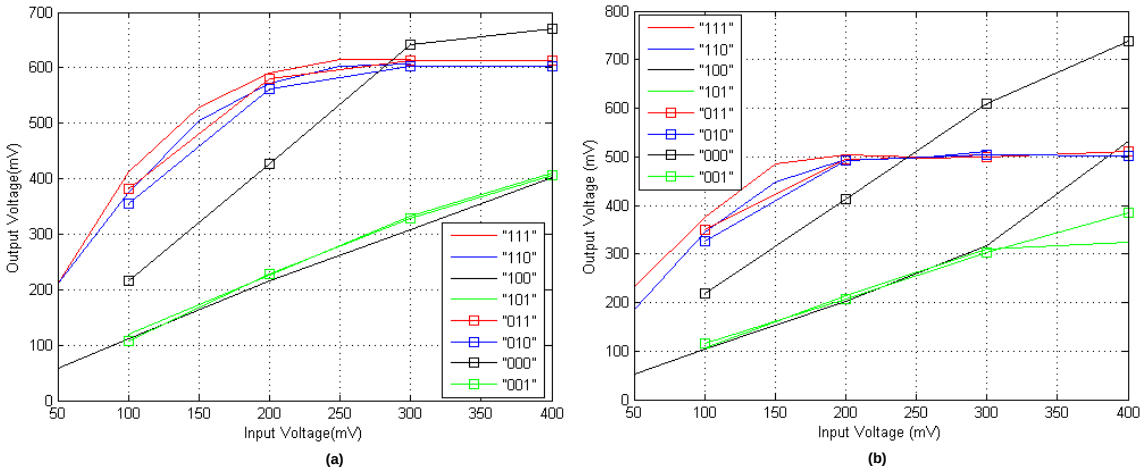

Figure 3.33: (a) Slow shaper output with positive pulse (b) negative pulse

along with CSA as shown in Table 3.3 for different values of feedback capacitance in CSA keeping input capacitance 100 fF. As we have mentioned previously slow shaper provides energy information and fast shaper provides timing information of the incident particles. As rise time of fast shaper circuit is small output voltage is quickly and it helps to properly calculate time of incidence of the charged particle. On the other hand rise time of slow shaper circuit is less compared to fast shaper that helps to calculate energy contents of incident particles. Hence output voltage of slow shaper saturate at high input voltage compared to fast shaper. Table 3.3 and Figure 3.32, 3.33, 3.34 proves that design criteria for CSA and shaper circuits are fully satisfied.

Table 3.3: Gain of slow and fast shaper for different feedback capacitance of CSA

Register Value	Slow shaper (positive pulse) (mV/fC)	Slow shaper (negative pulse) (mV/fC)	Fast shaper (positive pulse) (mV/fC)	Fast shaper (negative pulse) (mV/fC)
"000"	21.25	20.1	13.47	15
"001"	11.25	10	32	27.5
"010"	30	27	50	36.67
"011"	31.67	28	31.67	37.33
"100"	10.5	10	23.33	21.33
"101"	11.25	10.2	23.33	22
"110"	37.5	35	50	38
"111"	41	38	51	39

The main objective of implementation of IPbus in this experiment is to remotely control and access the FPGA board. To remotely access the FPGA using

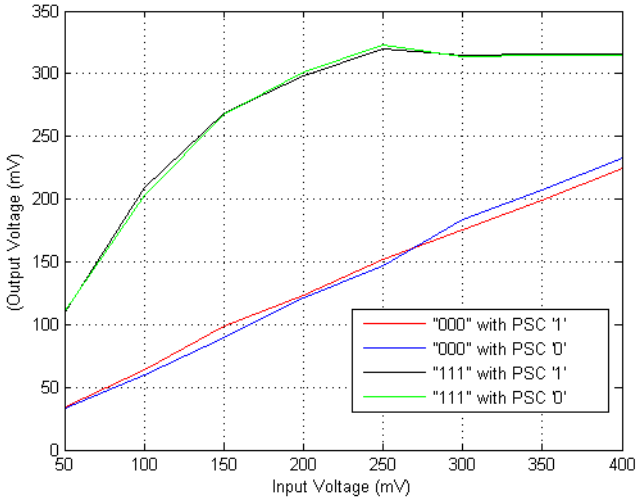

Figure 3.34: Variation of output voltage of slow shaper with polarity switch and feedback capacitance of CSA

Table 3.4: Resource Utilization for different module of IPBus on FPGA

Module Name	**Flip Flop**	**LUT**	**Memory LUT**	**BRAM**	**BUFG**
External PHY	37	33	1	Not Used	3
Ethernet_gmii	331	492	Not Used	2	2
Slaves	131	38	Not Used	1	1
IPADDR Block	141	36	Not Used	Not Used	1
ARP Block	134	120	Not Used	Not Used	1
RARP Block	336	247	25	Not Used	1
UDP Block	2586	1802	66	5	2
IPbus_ctrl	3241	2588	68	17	2
TOP Module	3823	2654	103	20	4

IPbus in the computer side, a socket program is written in python script. In this program *uhal* library from CERN for Ipbus v2.0 is used to connect the software with the FPGA board. At the same time we have written one address table which will be helpful to access individual register on FPGA board. Resource utilization for each functional block of the proposed system is given in Table 3.4. Implemented design consumed total 336 mW power. Resource efficient IPbus implementation meets the criteria for remote access of the DAQ system.

In order to test the GBTx emulator along with MUCH-XYTER and DPB data readout from the E-link FIFO has wrapped within the data generated from the pattern generator of GBTx transmitter and taken out from the GBT frame

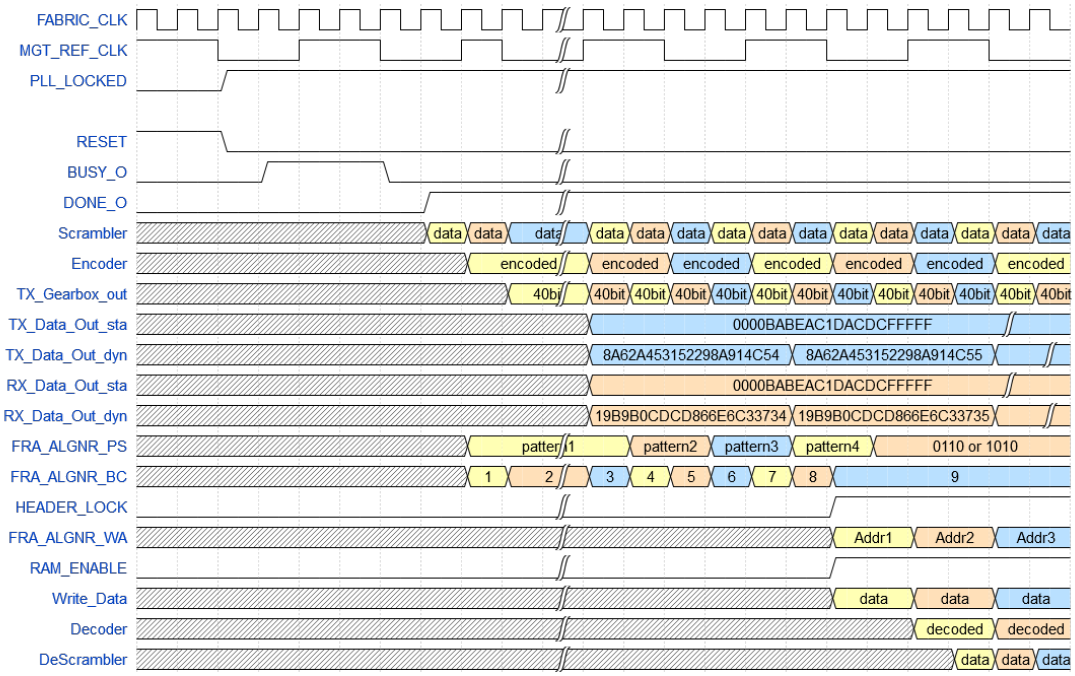

Figure 3.35: Timing diagram of the transmitter and receiver signals

in the pattern checker in the receiver of GBTx. Two types of pattern generator and checkers are used: static that gives fixed data pattern and dynamic that generates data continuously using a counter. Here, GBTx emulator is tested using GBT frame format only but it can be tested with widebus or 8B/10B frame also. Waveform of different signals used in the GBTx Emulator design is shown in Figure 3.35. Tx_Data_out_sta ,Tx_Data_out_dyn, Rx_Data_out_sta , Rx_Data_out_dyn give static and dynamic data pattern at the output of transmitter and receiver respectively. Though we have shown both static and and dynamic data pattern in the same diagram only one type of pattern generator and checker will be used during data transmission. Meaning of each signal used in the waveform shown in Figure 3.35 and their width are illustrated properly in Table 3.5. We have injected the SBU and MBU error by generating random error in the input data stream of GBTx emulator using random error generator [121]. The simulation results of BER is shown in the Figure 3.36 with respect to the noise (Eb/N) which varies from 0 dB to 10 dB. Here we used poison distributed noise in the channel. Figure 3.36 shows the efficiency of GBT frame that comprises of RS code with interleaver and scrambler and gives the best performance

Table 3.5: Description of the signals used in timing diagram

Signal	Width	Function	Use
Fabric_Clk	1	Use to drive different logical blocks	Tx and Rx
MGTREF_Clk	1	Use to drive MGT	Tx and Rx
PLL_Locked	1	Output of PLL and indicates PLL generate stable clock	Tx and Rx
RESET	1	Use to reset the whole system	Tx and Rx
BUSY_O	1	High when System enters a process before ready	Tx and Rx
DONE_O	1	High to indicate that Tx and Rx are ready	Tx and Rx
Scrambler	84	Contains the data of output of scrambler block	Tx only
Encoder	120	Contains the data after RS encoding	Tx only
Tx_Gearbox_out	40	Contains output of of DPRAM that acts like gearbox	Tx only
Tx_Data_Out_sta	1	Contains serialize output of transmitter when static pattern generator is used	Tx only
Tx_Data_Out_dyn	1	Contains serialize output of transmitter when dynamic pattern generator is used	Tx only
Rx_Data_Out_sta	1	Contains serialize output of receiver when static pattern generator is used	Rx only
Rx_Data_Out_dyn	1	Contains serialize output of receiver when dynamic pattern generator is used	Rx only
FRA_ALIGNR_PS	4	Check whether header is matched or not	Rx only
FRA_ALIGNR_BC	5	Store the output of counter until header is not matched	Rx only
Header_LOCK	1	High when header will be detected	Rx only
FRA_ALIGNR_WA	5	store address of RAM where receive data will be written	Rx only
RAM_ENABLE	1	High when RAM is Ready to perform	Rx only
Write_Data	40	Store 40 bit data which is to be written in RAM	Rx only
DECODER	84	Contains the decoded data	Rx only
DESCRAMBLER	84	Contains output data of descrambler block	Rx only

in presence the noise compared to other schemes (only RS coding or without RS coding, interleaver and scrambler). This error correction model meets the the specification regarding the error mitigation in the user data in the communication channel.

Resource utilization and power consumption for each functional block of GBTX Emulator, slave interface and MUCH interface core are given in Table 3.6. Power consumption is estimated using Xilinx Xpower tool and we show the estimated average logic and signal power for the various models of the proposed design.

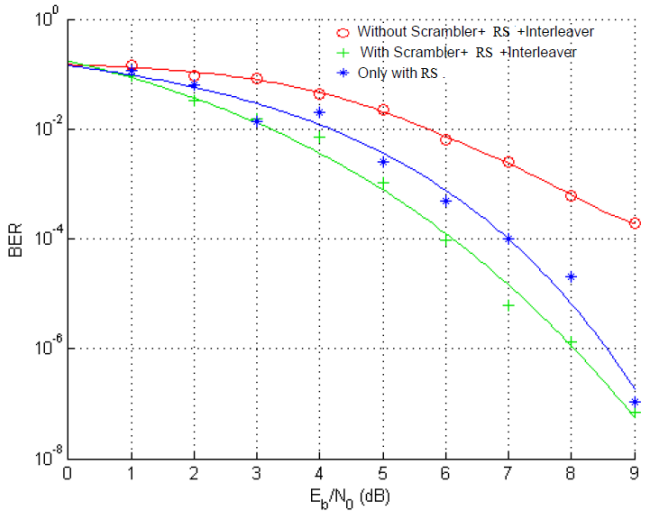

Figure 3.36: Study of of BER of GBT link using MATLAB simulation

Table 3.6: Resource Utilization and Power consumption by integrated design

Module Name	Slice Register	Slice LUTs	LUT FF	BRAM	Logic Power(mW)	Signal Power(mW)
RS Encoder	7	951	0	7	0.02	0.01
RS Decoder	135	446	0	119	0.05	0.07
Scrambler	52	53	5	0	0.04	0.00
Descrambler	104	56	5	0	0.01	0.00
Interleaver	44	40	40	0	0.01	0.01
DeInterleaver	201	82	80	0	0.01	0.02
Frame Aligner	115	308	72	0	1.34	1.07
MUCH Interface	620	804	108	4	4.18	1.91
Slave Interface	725	702	110	2	2.29	0.98

3.7 Conclusion

In this chapter we have tested different parameters of MUCH-XYTER ASIC integrated with data processing board using differential electrical line. We have also tested different functionalities of FPGA prototype of GBTx ASIC that is placed in between FEE ASIC and DPB and helps to carry the detector output from harsh radiation zone to comparatively less radiation zone. In order to monitor internal registers of different Electronics devices in the radiation zone IPBus protocol over custom build one gigabit Ethernet is also implemented on FPGA. In the next chapter we are going to propose a FPGA based DAQ prototype having more robust channel coding and efficient memory management algorithm so that it can work in different critical application.

Chapter 4

An FPGA based High Speed Error resilient Data Aggregation and Control System for Radiation Environment

Due to the dramatic increase of data volume in different critical applications a robust high speed DAQ system is very much needed that can collect data from sensors or detectors under harsh radiation environment. To handle such huge data we have proposed a DAQ prototype using FPGA due to some of its inherent advantages over ASICs. On the other hand FPGA devices are more vulnerable to the radiation compared to the ASICs. Hence, a major challenge in the development of FPGA based DAQ for the radiation environment is to mitigate the error occurred in the high speed data stream as well as configuration data of FPGA devices. CRC as well as orthogonal concatenated code have been used to mitigate the effects of data corruption in the communication channel. At the same time CRC technique is also used along with scrubbing for error mitigation in the configuration memory of FPGA devices. Data from front-end sensors will reach to the back-end processing nodes through multiple stages that may add an uncertain amount of delay to the data packets. We have proposed a novel memory management algorithm that helps to process the data at the back-end

computing nodes removing the added uncertain path delays. To the best of our knowledge, the proposed FPGA-based DAQ prototype utilizing optical link with channel coding and efficient memory management modules can be considered as a first of its kind. Performance estimation of the implemented DAQ system is done based on resource utilization, BER, EDAC efficiency and robustness against radiation.

4.1 Introduction

The main objective of a DAQ system is reading of information from one or multiple sensors or detectors for their use in real-time or to be stored for further off-line analysis. Function of data processing unit of a sensor system can be divided into four categories: acquisition, processing, analysis and integration. In most of the cases development of DAQ covers all of the four functionalities depending on their application and complexity. As for example single sensor based system dose not require much more complex integration schemes or processing algorithms but systems with multiple sensors may use simple processing algorithm along with complex integration schemes. In a traditional DAQ system FEE ASICs capture data from the sensors through differential copper link, processes it and sends it to back-end storage device using Ethernet. for further analysis. Traditional DAQ system face different problems like low data rate [122] and prone to be affected by SBU and MBU in highly radiated area. These traditional DAQ systems are not suitable to support the high data rate application. Hence, new type of DAQ architecture is required that can process and transfer large volume of data in very short time under radiation. Modern DAQ system contains hundreds of powerful processing unit that are interconnected through high speed buses like optical fiber, Ethernet and local area network. At the same time powerful software in the back end computing nodes remove the unnecessary data and stored the receive data for further analysis. Hence, proper protocol must be defined for data transmission throughout the readout chain to assemble data generated at different events. Figure 4.1 illustrates how a typical DAQ system interacts with an experiment. In this chapter, our proposed DAQ systems have been implemented on Xilinx Kintex-7 FPGA boards that involved board to board high speed communication

and PCIe interfacing with a host computer.

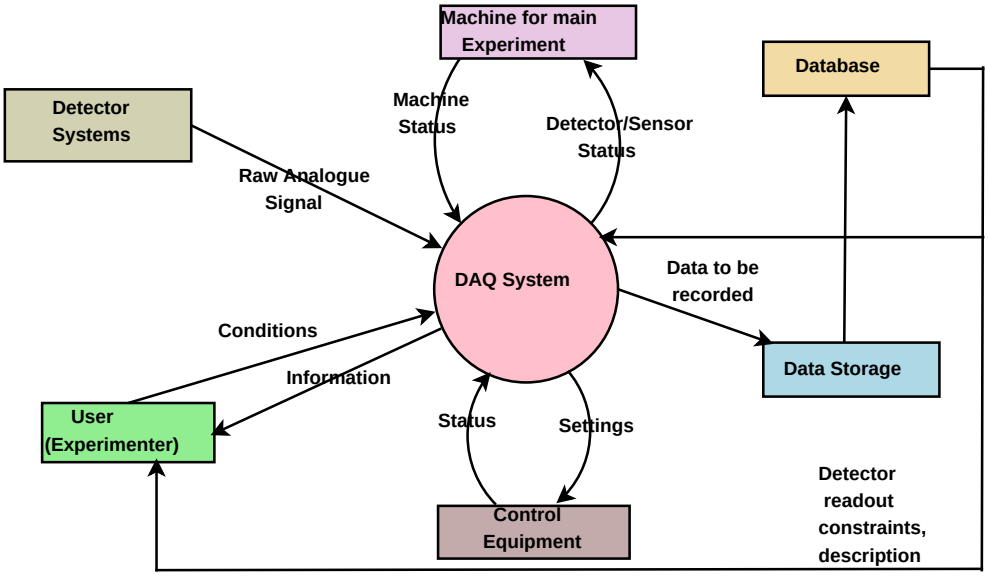

Figure 4.1: General DAQ system and its surroundings

The proposed DAQ system involves three stages of communication. In the first stage data is transferred from FEB to an intermediate board through electrical line and error in the data stream is detected by CRC. Second stage is responsible to transmit data from the intermediate board to control room using optical fiber where error correction is done with the help of orthogonal concatenated code. Finally the received data enters into the computing nodes through PCIe. Our proposed orthogonal concatenated code comprising of BCH and Hamming code reduces BER of data in the communication channel without increasing the complexity of decoding circuit compared to other multi-bit ECCs like RS, BCH and LDPC *etc.* In the proposed DAQ system, data transmission latency is also a critical issue in the high speed data transmission. Buffer within different logical blocks and high speed transceiver add uncertain amount of latency or delay in the transmission of data packets. Buffers are mainly used for clock domain crossing (CDC) and phase alignment. Here we have proposed a method to bypass some of the internal buffers in the transceiver with proper external circuit that performs the functionalities of CDC and phase alignment. The proposed method not only stabilize the data transmission latency but also reduces it compared to the situation when internal buffers are used.

As we are using FPGA to develop the full DAQ system there is a high probability that the design implemented on FPGA devices itself may be affected by the radiation. Hence proper steps should be taken to mitigate the errors in the configuration memory of FPGA devices due to radiation. We have used data scrubbing with CRC for error mitigation in the configuration memory. In this technique, the configuration file of the FPGA is stored in a separate Radhard flash memory that is also placed on the same electronic device that contains FPGA. During runtime configuration memory of FPGA is read back in a periodic interval and if error is detected in the configuration file using CRC, stored configuration file will be downloaded into the configuration memory.

In large DAQ systems sensors are spreaded over large area along with their FEBs. Hence, data from different sensors will reach to the back-end computing node with different path delay. These difference in path delay arises due to different cable delay as well as uncertain delays added by different FPGA boards. As readout chain handles real-time data, there is no scope to store the data on FPGA boards for long time before sending it to the computer. Hence, proper memory management is required to aggregate the data coming from different FEBs at a particular time instant before being forwarded to the host computer. In this chapter our contributions are:

- Efficient implementation of a novel error correction code using orthogonal concatenated code for error resilient high speed communication.
- Special design measures have been adopted to optimize the transmission latency of the hardware.
- FPGA implementation of efficient memory management algorithm to aggregate the data before processing through the back-end computing nodes.

4.2 System Design for High Speed DAQ

The high speed DAQ system works in the radiation environment should have different features like fault resiliency to enhance system lifetime, efficient data aggregation capability, precise time synchronization and contain high speed reusable

data processing modules.The radiation level that creates error in the configuration memory FPGA is only available naturally in the upper layer of atmosphere and to test DAQ using that radiation level designer has to take the help of space craft or space shuttle. The luminosity [123] of cosmic ray on the earth crust is unable to damage a FPGA devices in short run. Another option to get such high level of radiation is within the accelerator where different HEP experiment is carried out like LHC at *CERN* [11]. Hence, here we have proposed a simplified hierarchical readout chain for data aggregation and control system of HEP experiment as shown in Figure 4.2. At the first stage of the proposed DAQ,

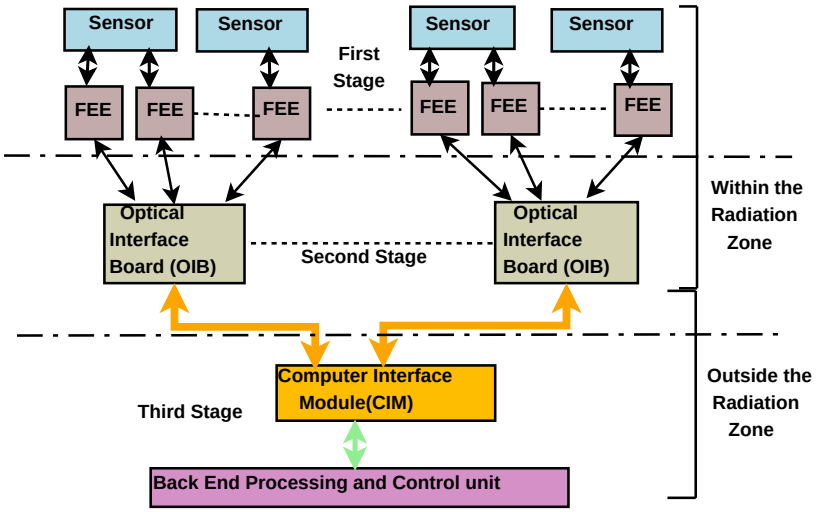

Figure 4.2: Simplified read out chain for multistage data acquisition system

FEE will receive data from the sensors and send it to a data aggregator board named as Optical Interface Board (OIB) through the E-link. Multiple OIBs, placed in the second stage of the hierarchical DAQ send data to Computer Interface Modules (CIM) through the optical fiber which in turn send data to the back-end computing nodes through PCIe. Here computing nodes work as server and CIMs are PCIe plugin cards. First and the second stage of hierarchical DAQ network *i.e* FEE with detector system and OIB are within the radiation zone, and CIM with computing nodes are outside the radiation zone and placed within the control room. Instead of Radhard FPGA from Xilinx or Altera we are using COTS FPGA to develop the DAQ system that can work in the radiation zone. A simplified readout chain using single OIB, CIM and multiple FEBs is shown

in Figure 4.3. Functionalities of each module of the readout chain are described below.

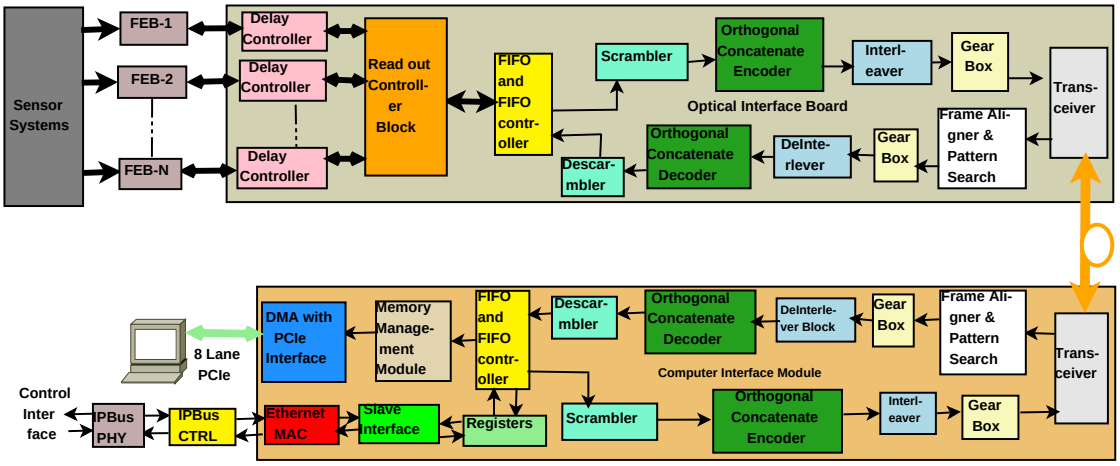

Figure 4.3: FPGA based readout chain prototype having single OIB and CIM

4.2.1 Optical Interface Board (OIB)

Frontend Electronics Board: Main functions of the FEBs are the digitization of the received analog signal from the detectors, packaging and sending it to OIB. Like to the front end ASICs describe in the chapter 3, here also each FEB sends three informations to the backend: voltage of the detected particles, time at which the particles are detected by sensors and position of the sensors. Front end ASICs have mainly two parts one is analog front end and other is digital backend. Interfacing of analog front end and digital backend is analog to digital converter (ADC) and digital backend contains different interface protocol to communicate with other devices in the system. Here, we have implemented only FPGA prototype of digital backend of the XYTER ASICs described in the chapter 3 and connected it with a hit generator that generates the data packets randomly containing energy and timing information as shown in Figure 4.4. Figure 4.4 describes the internal architecture of FPGA-based FEB emulator. Hit generator in the FEB emulator generates the data in the same way as the output of the ADC generates data in the XYTER ASIC described in chapter 3 for particle hits in the detector and appends the channel number randomly with the generated hit data. FEB emulator sends data at 320 Mbps and receives data at

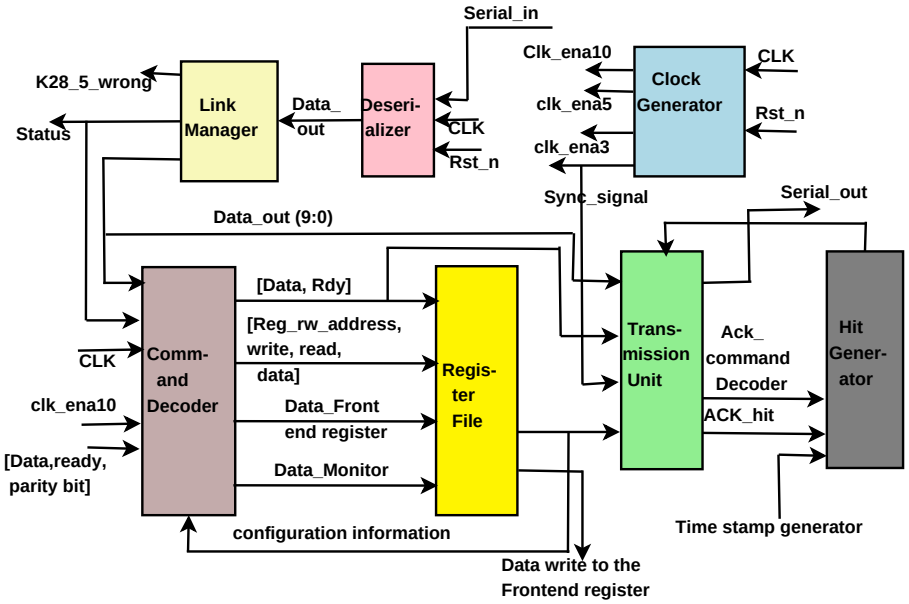

Figure 4.4: Internal architecture of FPGA based FEB emulator

160 Mbps similar to the backend of XYTER ASIC. Digital backend of XYTER emulator and OIB are placed in one FPGA board and CIM with PCIe interface are placed in another board. For the synchronization between FEB emulator and OIB same E-link protocol is used as explained in chapter 3.

Clock generator in FEB emulator regenerates the clock signal after receiving clock from OIB. The command and control signals received from optical module enter directly into deserializer block as shown in Figure 4.4. After deserialization data is sent to the link manager block in FEB emulator where a state machine checks whether the characters K28.5, K28.1, SOS, EOS (used for synchronization in E-link) are received in proper sequence or not and then, received characters are sent to command decoder block. Command decoder block decodes the data using 8b/10b decoder and compares them with the values stored in the register file and sends an acknowledgment signal to the transmission unit in the FEB emulator. After receiving the positive acknowledgment signal, transmission unit reads the value of special characters from the register file, serializes them and sends them to OIB. After completion of synchronization between FEB and OIB, command decoder and transmission unit send an acknowledgment signal to the hit generator which generates the data along with the time-stamp value. After

synchronization, a counter generates the time stamp information and append it with the data generated from the hit generator module.

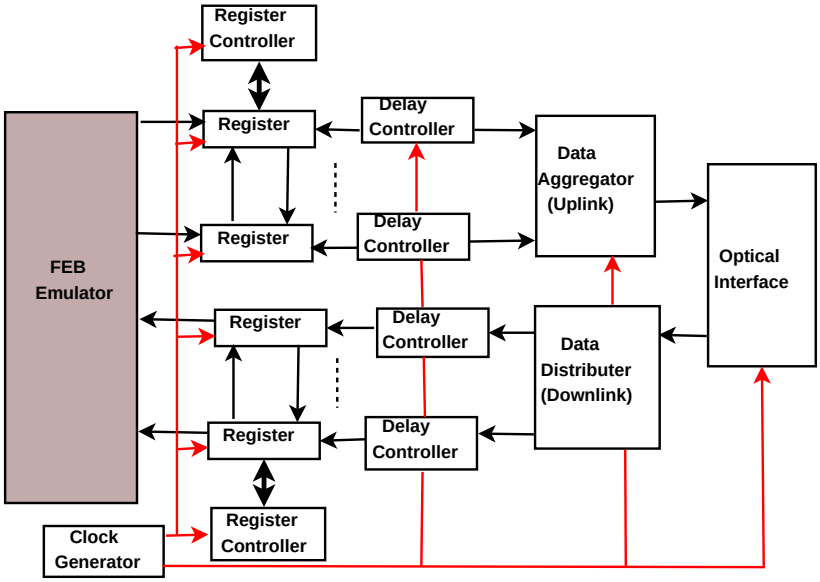

Figure 4.5: Interfacing of FEB emulator with optical module in OIB

Optical module receives data from FEB emulator at 320 Mbps through the E-Links and transmits them to CIM at 5 Gbps. As optical module is running at 52 MHz, there is a register in each E-Link between FEB emulator and optical module that temporarily stores the received data from FEB. Write frequency of register is 320 MHz and read frequency is 52 MHz hence width of each register is six bit. Similarly in the downlink FEB receives data at 160 Mbps and hence width of the register in the downlink is three bits as shown in Figure 4.5. As data width for optical communication is 40 bit (explained later) one FEB can supports maximum six uplinks and thirteen downlinks. Here we have used six uplinks and one downlink. Single optical module receives data from multiple E-Links and the data aggregator block in OIB reads data from multiple registers at the same time and send the aggregated data to optical module in the uplink direction. In the downlink, data from optical module will be written into data distributer block which writes data into multiple registers in the downlink as illustrated in Figure 4.5. Data transmission between FEBs and optical module takes place through multiple E-links with different path delays that can create the problem during data aggregation process. Here FEB emulators and optical

module are within the same board so E-link means data channel through FPGA fabric. The path delay between different channels will be more prominent when FEB emulator will be placed in different FPGA boards outside the OIB and will be connected with OIB through electrical lines. In order to adjust such path delays, an automatic delay controller is attached with each E-Link. During the synchronization process (as described in chapter 3) the delay controller calculates the delay of each link and automatically updates its offset value through downlink frame from CIM. Optical fiber transmits data serially and hence fixed delays are added to the data from each E-link after data aggregator block. Automatic delay controller does not deal with the fixed delay added after the data aggregator block. Data from aggregator block enters into the scrambler in OIB as shown in Figure 4.3.

Similar to GBT-FPGA core described in chapter 3 here also scrambler is used for clock and data recovery and equalization. Unlike to 21 bit scrambler in GBTx here we have used 10-bit scrambler after dividing input 40-bit data into four bit data chunk. Here $X^9 + X^4 + 1$ is used as the scrambler polynomial. The scrambled data from four blocks are concatenated together to produce the 40-bit scrambled data at the output. Descrambler in the receiver side performs just the complimentary function.

Concatenated Code: The error correcting capability and decoding complexity of a concatenated code depends on the choice of the component code. As we are dealing with high speed communication, minimization of latency during encoding and decoding should also be considered for the selection of component code. We have used single bit error correcting BCH code (15,11) and (7,4) Hamming code as the component codes. RS coding which is used in GBTx in chapter 3 is suitable for burst error correction but for random error correction binary version of RS coding *i.e* BCH coding is more efficient. Figure 4.6 shows that BER performance of BCH coding is far better compared to RS coding against random error using binary phase shift keying (BPSK) as illustrated in [124]. In communication channel the probability of occurrence of random error is more compared to burst error. Even if burst error occurs the interleaver after encoder will distribute it as random error. For this reason BCH code is chosen as one of the component code of concatenated

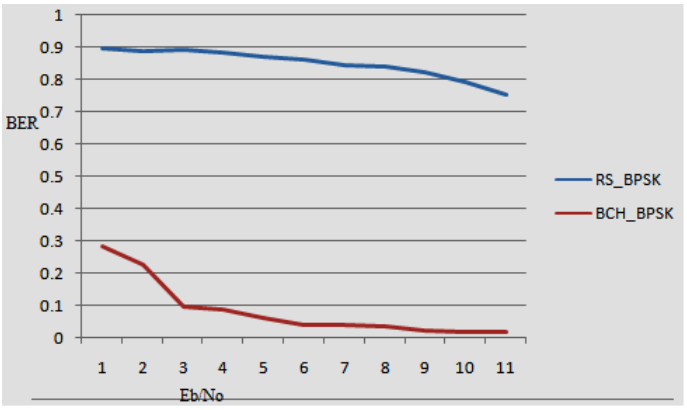

Figure 4.6: BER performance of BCH and RS code against random error using BPSK modulation [124]

code instead of RS code. 40-bit input data along with four-bit header enters into

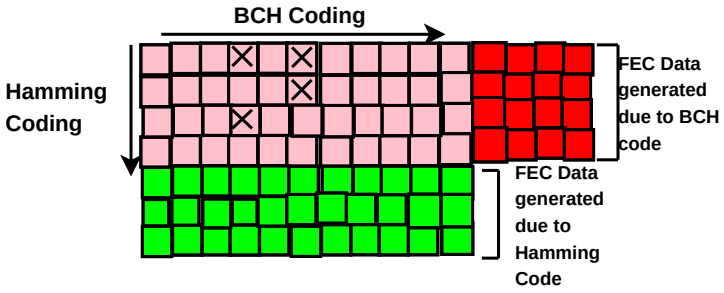

Figure 4.7: Concatenate code using Hamming and BCH code

the encoder and arrange in a 4×11 matrix as shown in Figure 4.7 by the pink colored square box. BCH is a binary ECC, that encodes the data along the row using the polynomial $g(X) = X^4 + X + 1$ and generates 16 redundant bits (4×4) as indicated by red colored square blocks in Figure 4.7. Simultaneously (7,4) Hamming code runs along each column and generate 33 redundant bits (3×11) as shown by the green colored blocks. Hence, the input of the encoder block is 44-bit data, and output is 93-bit (44+33+16) data. If more than one bit error occurs along any row or column of the matrix (as shown by 'X' along the first row and fourth and sixth column of the matrix in Figure 4.7) single bit ECC can not rectify them. These erroneous bits can be corrected using row decoding (BCH coding) and column decoding (Hamming coding) in parallel. It increases error correction capability and reduces BER of the data stream. Errors in the third row and second row will be corrected using row decoding and errors in the first

row will be fixed using column decoding as shown in Figure 4.7. Steps involve in decoding of orthogonal concatenated code are as follows:

- Compute the syndrome along the row and column using BCH and Hamming code respectively.
- Determine the error locater polynomial from the syndrome calculated using BCH decoder.
- Find the error location by solving error locater polynomial and syndrome of hamming decoder.
- Invert the bit where error is detected.

Interleaver: Interleaver is used to eliminate the effect of burst errors in the communication channel as we have discussed for GBTx in chapter 3. Here, we have used matrix helical scan interleaver [125] in our proposed design in stead of block interleaver where the frame header is always within a single block. Hence if header is affected by burst error, there will be a problem in frame detection and synchronization. In helical interleaver, header bits are uniformly scattered throughout the frame. During interleaving data is filled in the matrix along

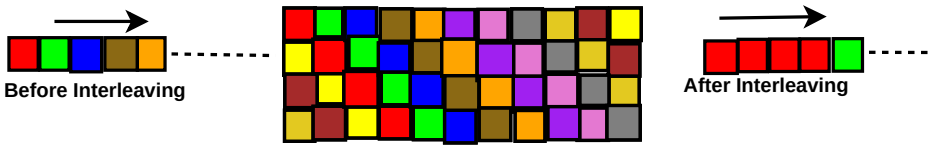

Figure 4.8: Helical Interleaving Process

the row and then read the matrix content in the helical fashion as shown in Figure 4.8. Helical fashion means data is selected along the diagonal of the matrix in such a way that row and column index both increases during the data readout. Array step size is the parameter that calculates the slope of the diagonal i.e the amount by which row index increases as the column index increases by one. Figure 4.8 shows functionalities of a helical interleaver where small square blocks with the same color come out sequentially from the interleaver block. Deinterleaver performs just inverse function at the receiver.

Gearbox and MGT: Gearbox breaks down 96-bit frame into six words of 16-bits width. Data writes in the gearbox at a frequency of 52 MHz and read at 312 MHz, which is also used to drive the MGT. This block is used to synchronize the data rate between MGT and the other parts of the design. In order to minimize the latency we have not used any dual port RAM to break frame into words. MGT works like a high speed serializer/deserializer block. Details of the transceiver architecture and its functionalities are explained in chapter 3. CDR circuit is used to recover the clock that helps in data transmission. CDR in each channel consists of edge sampler, data sampler, De-MUX, phase interpolator (PI) and PLL. This type of CDR circuit uses phase rotator architecture [126]. Data from decision feedback equalizer(DFE) is captured simultaneously by both edge and data sampler. The output of data sampler and edge sampler is fed to the CDR state machine which calculates phase of the incoming data stream and controls the phase interpolator (PI) of both edge and data sampler. Finally phase of the edge sampler is locked at the transition region while the phase of the data sampler will be locked at the middle of data eye. Details internal architecture of CDR circuit is shown in Figure 4.9. The transmitter within the MGT converts the parallel

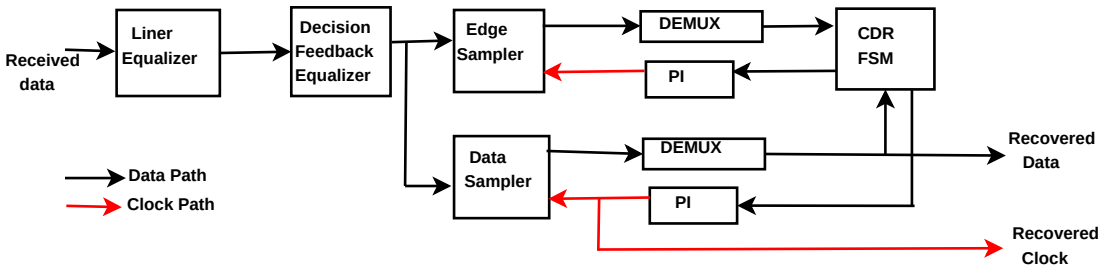

Figure 4.9: Internal architecture of CDR circuit in the Xilinx Transceiver

data to serial data and sends it to SFP module, that converts electrical signal to optical signal, modulates the signal and sends optical signal to the communication channel. The receiver simply converts the serial data to parallel data.

4.2.1.1 Frame Aligner and Pattern Search Block

As communication between OIB and CIM is asynchronous in nature proper header detection using frame aligner and pattern search block plays an important role. Header detection is used for frame synchronization. The frame is synchro-

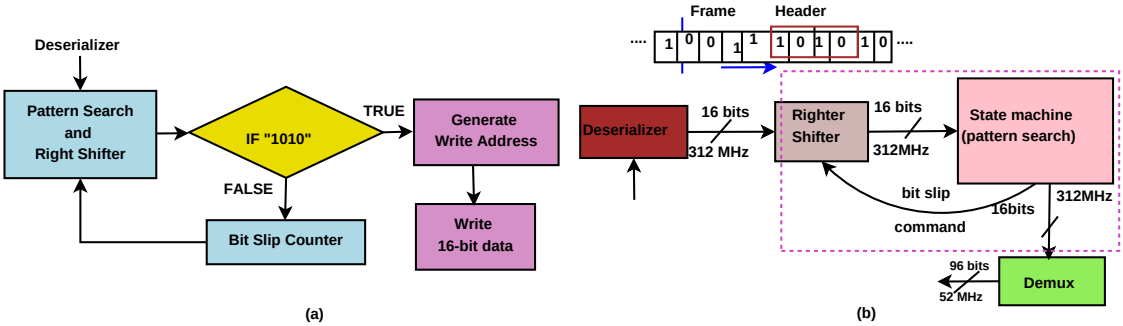

Figure 4.10: (a)Algorithm for Frame Aligner and Pattern Search (b) Data flow diagrams of the Frame Aligner and Pattern Search block

nized using an efficient pattern search algorithm as shown in Figure 4.10(a). Generated frame consists of three fields: Header (4-bit width), Data (40-bit width) and Forward Error Correction(49-bit width). The header field contains "1010". The frame aligner and pattern search block consists of two subblocks: Pattern search and Right shifter block. Right shifter block continuously shifts one bit of data in the received frame on the right side from MSB and send it to pattern search block to check whether the header field is detected or not. After the header is detected properly, one locked status signal is generated to trigger the data capturing process as illustrated in Figure 4.10(b).

4.2.2 Computer Interface Module(CIM)

CIM receives data from multiple OIBs at 5 Gbps data rate and transmits data to the back-end computing node through PCIe. Scrambler/Descrambler, Encoder/Decoder, Gearbox, Frame Aligner and pattern search block perform the same function as in OIB. Functionalities of other blocks like DMA with PCIe interface, Memory management module, and IPbus interface are described in the following subsection:

Data Transfer to Host PC through PCIe: PCIe is a high speed computer expansion bus standard and it is developed to replace older PCI and PCI-X bus standard. In true sense PCIe is more like a network where each card is connected to a switch using a dedicated set wires. It provides dual simplex point to point serial connection, scalable link width (x1, x2, x4, x8, x16) and link speed (2.5,5,8

GT/s). PCIe uses packet based transaction protocol, data integrity using CRC and advanced power management mechanism.

The protocol used for data transmission in PCIe mainly involves generation and processing of basic data packets known as transaction layer packets (TLP), power management, flow control, error detection, status checking, physical link interface initialization, serialization and deserialization etc. Transaction layer, data link layer and physical layers are involve in the generation and processing of TLP. There are mainly three types of data packets are used during the data transmission: Write packet, read request packet and completion packet. Data Payload along with header comes from the software layer. Structure of a memory write request packet and completion packets having width 128 bits are shown in Figure 4.11 (a) and (b) respectively. Here we have not shown the read packet separately because it is same as write packet without data field. Functionalities of each field used in different data packets are illustrated in Table 4.1.

Table 4.1: Function of different fields in data packet used for PCIe communication

Field	Function
R	Reserved
Fmt	**2** for write and completion packet. **2** for read request packet
Type	**0** for write and read request. **0x0A** for completion packet
TD	**0** indicates that transaction layer does not add any CRC i.e ECRC field will be zero
Length	**0x001** indicates that width of each data word (DW) is 32 bit
Requester ID	Used in read request and completion packet. It identifies the devices where to send the response
Tag	Used to match the completion packet with corresponding read request packet
BE	First BE field is used to enable first DW and last BE field indicates whether last and first DW are same or not
Address	Give the address of the device with which host device will communicate
Byte Count	Used only for completion packet and indicate number of packets left for transmission
Lower Address	Used only for completion packet and store seven bit LSB of the address field from which first byte of TLP will be read
Completion ID	Used only in completion packet and give id of the sender
status	Used only in completion packet and indicates whether completion is successful or not

Data link layer (DLL) adds sequence number and link CRC (LCRC) with TLP. Apart from this, DLL also issue a special packets called data link layer packets (DLLP) for maintaining reliable transmission, flow control and power management. Finally physical layers add start and stop bits with the TLP and form the data packet as shown in Figure 4.11(c). In PCIe, two types of interrupt are used: legacy INTx and message signaled interrupt (MSI). Here we have used MSI because this type of interrupt does not require any completion TLP.

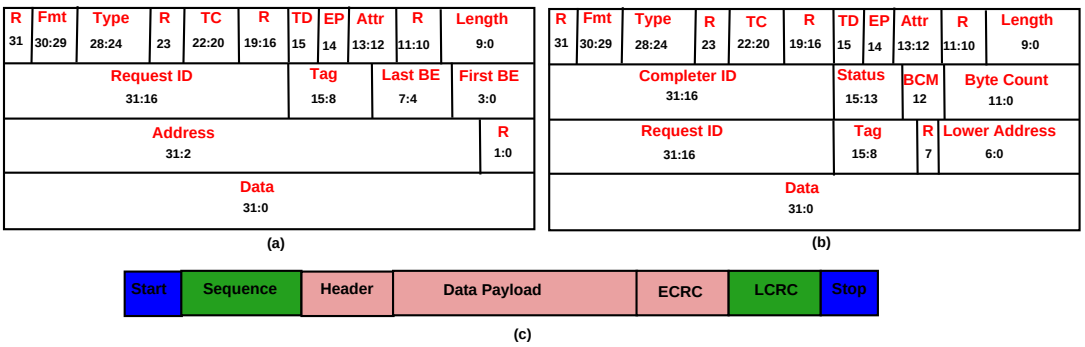

Figure 4.11: (a) Frame format for write request TLP (b) Frame format for completion TLP (c) Structure of TLP packet after passing through physical layer

Details architecture of data transfer between CIM and host computer using PCIe and DMA controller is shown in Figure 4.12. In this communication initially PCIe core transfers the data between memory space in host computer and packet buffer in the PCIe controller and then DMA engine transfer data between packet buffer and registers of the user logic. IRQ and TLP encoding/decoding module shown in Figure 4.12 process interrupt and TLP data packets respectively. DMA Req Buffer holds read request packet during non-posted data transaction until completion packet is received. DMA engine contains DMA registers, scheduler, recorder buffer, packet splitter, descriptor engine and descriptor engine update module as shown in Figure 4.12. Reorder buffer rearranges the receive data packets read from the packet buffer before sending them to user logic and scheduler sets the priority among the channels in a round robin fashion. Descriptor within the DMA specify the source, destination and length of DMA transfer. Each channel has its own DMA descriptor list. Starting address in the descriptor's list is specified by the driver during the link initialization.

Before starting of communication, PCIe will check whether any interrupt is pending or not in IRQ module. If no interrupt is pending PCIe controller checks whether any data packet is present in the Rx and Tx packet buffer and if present they should be processed first to avoid any deadlock. Data packets in Rx packet buffer can be of three types as discussed previously. Completion packets indicate that previously user logic issued any read request and corresponding read data is present in the Rx packet buffer. For a single read request multiple completion packets may be present but all of them have same tag. Now to transfer data

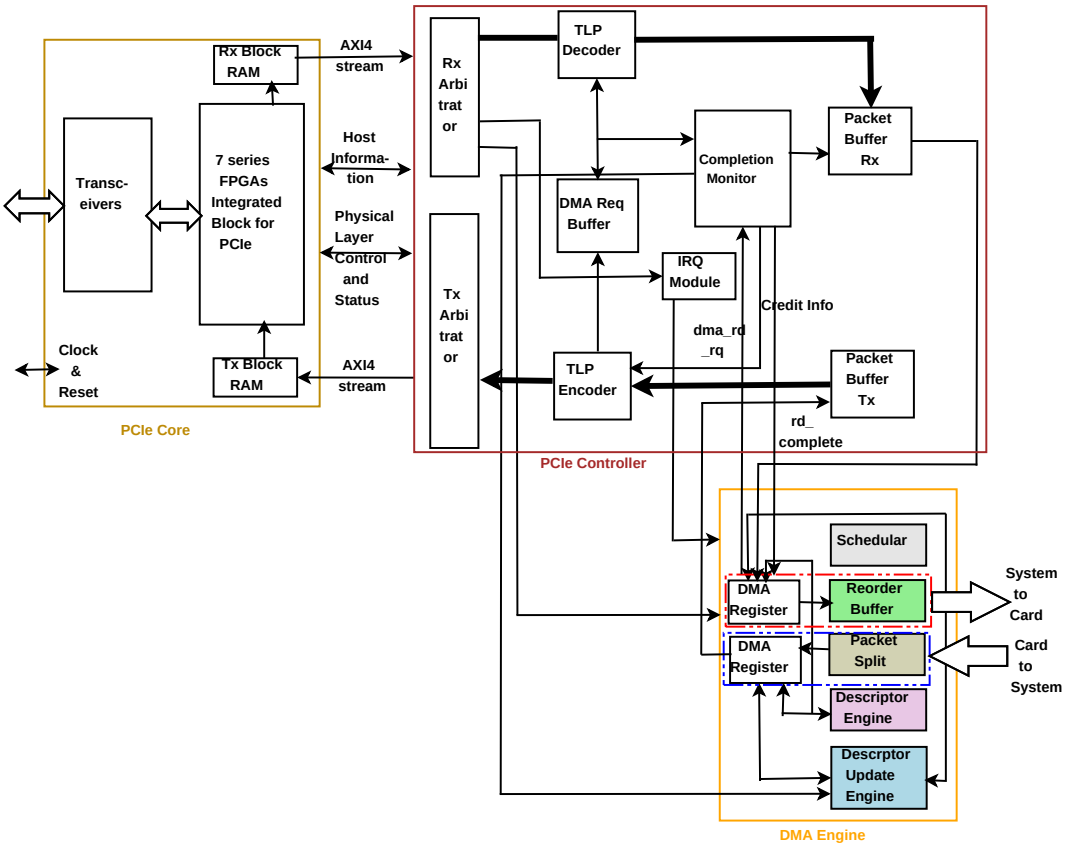

Figure 4.12: Data transfer between PC and CIM through DMA and PCIe

from Rx packet buffer to user logic DMA engine opens the corresponding system to card (S2C) channel and the DMA driver will create descriptor based on the input data size and address. Then DMA engine reads the data from packet buffer based on the descriptor, reorder them if multiple completion packets present for a single read request and write them to destination address. After completion of data transfer DMA issue an interrupt to the host. When the packet in the buffer corresponds to memory read packet, DMA engine opens corresponding card to system (C2S) channel. Based on the descriptor information DMA engine reads the data from the address mentioned in memory read packet and sends it to Tx packet buffer. Then TLP encoder wrap it into a completion packet and sends to host. Finally, if the packet is write request DMA engine reads the data and write it into corresponding user memory space. Similarly, in Tx packet buffer any of the three types of data packets may present. Completion packets in Tx

packet buffer indicates host issued a read request and corresponding completion packet will be wrapped by TLP encoder and sent to host. Write request and read packets in Tx packet buffer will be encoded by TLP encoder and send to host for proper action.

If there is any pending interrupt from any user logic then PCIe controller switches to the corresponding subroutine and data transfer between that logic and host PC through DMA engine starts. When there is no pending interrupt or data packets in the buffer, FPGA sends read request TLP to host computer mentioning start address, number of data bytes to be read and a request identifier tag to initiate the read operation. Sometimes multiple read requests can be issued at a time from the FPGA side and then DMA engine will reorder the received data stored in the packet buffer based on the requester ID in the completion packet and their increasing address. Details of the data transfer between CIM and host computer through DMA and PCIe is shown by a flow chart 4.13.

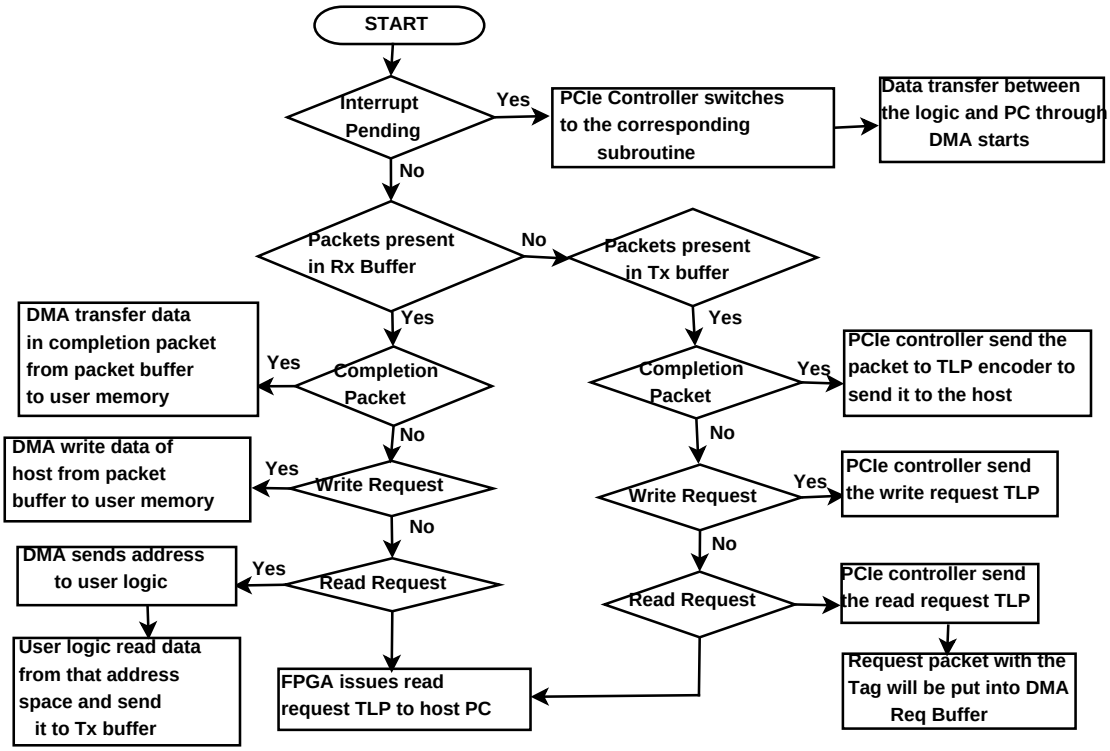

Figure 4.13: Flow chart for data transfer between host PC and CIM through PCIe and DMA

Memory Management Module (MMM): Data generated from different FEBs with similar time-stamp may reach to the CIM at different time instant due to various path delays, and uncertain amount of latency added by the different memory elements. The objective of the MMM in CIM is to aggregate the data generated with similar time-stamp from the hit generators in different FEBs before sending them to computing nodes. Some of the data may reach to the CIM at the expected arrival time, some of them may reach after a certain delay from the expected arrival time, and some of them may reach before expected time. Hence, data need to be held up in the CIM till the maximum expected time of uncertainty. Suppose t be the maximum time of uncertainty in the experiment. Hence, before sending data to the computing node through PCIe, data can be held for maximum t time units in the CIM otherwise, there will be a backlog or queue of the data. In order to handle these kinds of online data streams, a novel algorithm is proposed that mainly comprise of a memory write module (algorithm 1), read module (algorithm 3) and fragment eliminator module (algorithm 2). The prerequisites of the proposed algorithm are:

- Data entering into MMM should be progressive in nature. Progressive means that data average should be either ascending or descending.
- Repetition within the data set is more desired which makes this algorithm more efficient.

For the implementation, we have taken two types of memory space: static memory space (sta_mem) and dynamic memory space (dyn_mem) as shown in Figure 4.14. Each address space in the static memory is fixed for the data generated in a particular time slot (described later) whereas in dynamic memory address spaces are not fixed for a particular time slot.

First data in a particular time slot will be stored in static memory and other data in the same time slot will be stored in dynamic memory as per the vacant space available. The number of memory pair (one static and its corresponding dynamic memory will form a memory pair) required will be determined by the maximum time of uncertainty in the system and frequency of the hit generator in the FEB (resolution of the sensor). Suppose, first a bits (from MSB) of a data

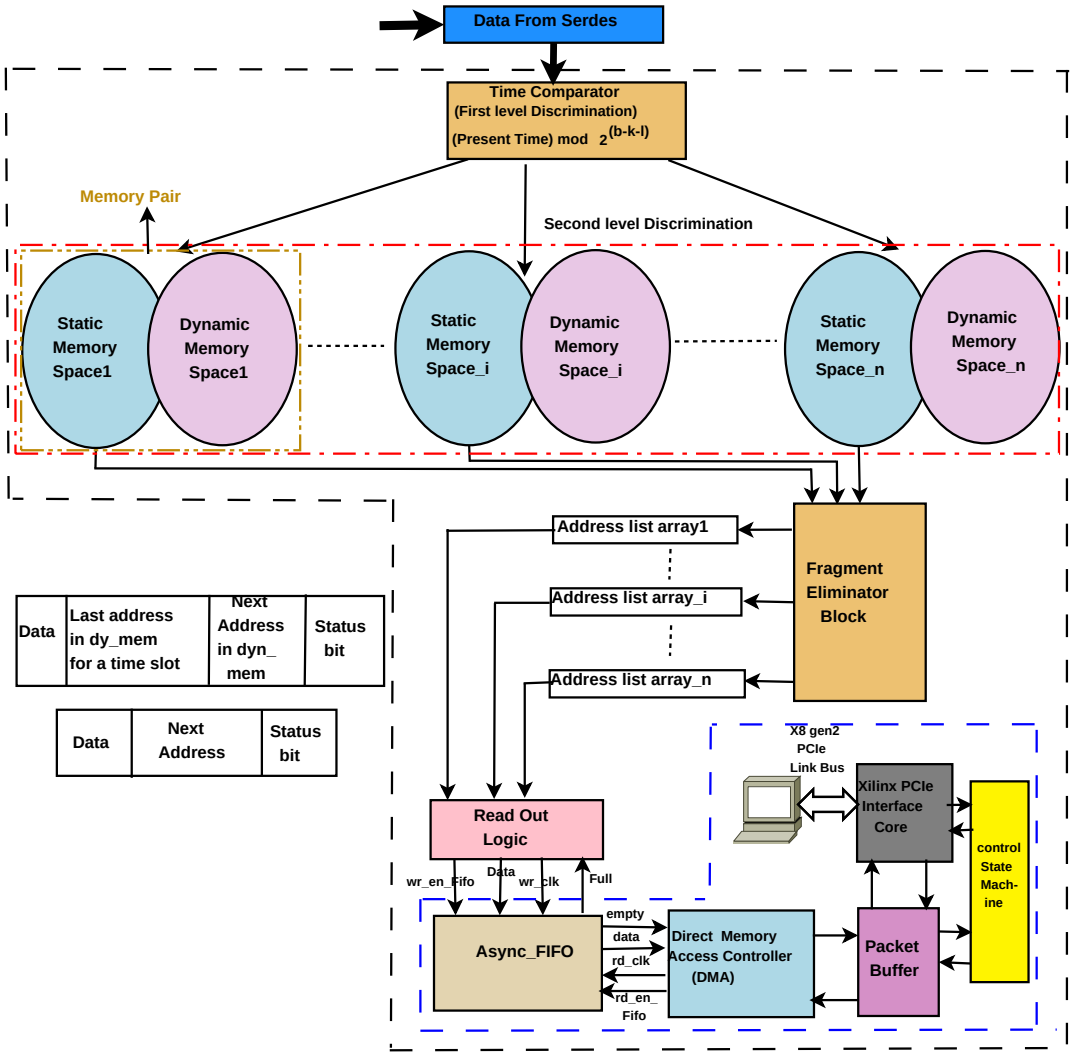

Figure 4.14: Implementation of memory management module with PCIe interface

packet coming from the FEBs represent energy information and last b bits represent time information where data packet width is $a + b$. If f be the global clock frequency which drives the full readout chain, F be the frequency of hit generator then data coming into the CIM within $\frac{1}{F}$ unit time will not be discriminated. This $\frac{1}{F}$ unit time represents one time slot. If k be the number of bits within b from LSB such that it satisfies the condition $\frac{1}{f} \times 2^k = \frac{1}{F}$ then only $(b-k)$ bits are required to determine the number and size of static and dynamic memory. If l be the number of bits that satisfies the condition $\frac{1}{f} \times 2^l = t$ then number of addresses within the static and dynamic memory will be 2^l and 2^{l+1} respectively and the

number of static and dynamic memory will be $(2^{b-l-k} + 1)$. One extra memory pair is required to eliminate the vacant space created during the data writing in static memory using the fragment eliminator module before the reading operation starts. In our case $a + b = 30$ bit and b is 12 bit as mentioned previously, f is 100 MHz, F is 20 MHz, and t is 1 us ($i.e$ maximum time of uncertainty is 1 us). Here, values of k and l are 3 and 7 respectively. Hence, the number of static and dynamic memory i.e memory pair is five. Each static memory has 128 address locations and each dynamic memory has 256 address locations. Maximum time of uncertainty is 1 us means each memory pair holds data of 1 us. Here out of the five memory pair, read operation will be performed on one pair, fragment elimination operation will be performed on another memory pair and the other three will be used for writing present, past, and future data respectively. Hence the number of static and corresponding dynamic memory used for writing will depend on the delay that MMM can afford before transmitting data through the PCIe.

Corresponding to each static memory there is one dynamic memory. Each address location in static memory corresponds to $\frac{1}{F}$ unit time slot and contains one data field, next address field in dynamic memory (nxt_add), last address in dynamic memory corresponding to this time slot (last_add) and status bit to indicate the presence of data in this time slot as shown in Figure 4.14 . Dynamic memory stores the data if multiple data present in the same time slot corresponding to an address space in staatic memory. Each address space in dynamic memory contains one data field, next address of the data presents in the same time slot (next_dyn_add) and the status bit (syn_status) to indicate whether data is present in that address of dynamic memory as shown in Figure 4.14. Each address space in static memory is fixed for a time slot equivalent to $\frac{1}{F}$ unit time but in dynamic memory there is no separate demarcation of the address corresponding to a time slot.

At the very first step after receiving a data packet from serdes block, static memory number is calculated at first level discriminator and second level discriminator decides address space (time slot) within the static memory as shown in Figure 4.14. Suppose for a data packet u be the static memory and v be the address space within the u^{th} static memory. If the status flag of v^{th} time slot is

zero the data will be written into v^{th} time slot and status flag of v^{th} time slot will be raised to one.

Algorithm 1 Algorithm for Memory writing

```
1: Variable:=X,Y,Z,U,P;
2: Read the data from online data stream;
3: Compare time information within the data at the first level discriminator;
4: sta_mem_num=(present time) mod 5; i=sta_mem_num;
5: if (wr_en(sta_mem_num)='1') then
6:     X= Address within the static memory;
7:     if (sta_mem_i(X).status='0') then
8:         sta_mem_i(X).data=data; sta_mem_i(X).status='1';
9:         sta_mem_i(X).next_add=Null; sta_mem_i(X).last_add=X;
10:    end if;
11:    if ((sta_mem_i(X).status='1')&(sta_mem_i(X).next_add)=Null) then
12:        Z=Lowest available address in dyn_mem;
13:        sta_mem_i(X).next_add=Z; sta_mem_i(X).last_add=Z;
14:        dyn_mem_i(Z).data=data; dyn_mem_i(Z).status='1';
15:    end if;
16:    if (sta_mem_i(X).status='1')&(stat_mem_i(X).next_add≠Null) then
17:        P=sta_mem_i(X).last_add;
18:        U=Lowest address available in dyn_mem_i;
19:        dyn_mem_i(U).data=data; dyn_mem_i(U).status='1';
20:        dyn_mem_i(P).next_add=U; dyn_mem_i(U).next_add=Null;
21:    end if;
22: end if;
```

When the status flag of v^{th} time slot is one and next_add field of v^{th} time slot is $NULL$ then data will be written into the vacant space available in dyn_mem and address of this vacant space will be written into the next_add and last_add field of v^{th} static memory. If next_add field of v^{th} time slot is not $NULL$ then data will be written into the vacant space available into the dynamic memory and address of available vacant space in dynamic memory will be written in last_add

Algorithm 2 Algorithm for Fragment Eliminator

```
1: Variable:=j,k;
2: Scan status of all five static memory;
3: stat_mem_num= Static memory for which wr_en and rd_en signals are
   zero; i=stat_mem_num;
4: Frag_elmin_en(stat_mem_num)=1;k=1;
5: for j=1, j<=maximum size of (sta_mem(stat_mem_num)), j++) do
6:     if (sta_mem_i(j).status=1) then
7:         array_list_i[k]=j; k=k+1;
8:     end if
9: end for
10: Frag_elmin_en(stat_mem_num)=0; rd_en(stat_mem_num)=1;
```

field of v^{th} static memory. There is a high probability that a number of time slot in the static memory remain vacant which fragments a static memory into multiple regions. Details of the write logic is explained in the algorithm 1.

During the writing in the static memory some vacant spaces may be created within the static memory that divide total memory space into multiple fragments. In order to make the readout logic more efficient these vacant spaces should be removed which is termed as fragment elimination technique as described by the algorithm 2. After t units of time from starting of writing in a static memory, read and write enable signal of that static memory will be low, then fragment elimination block start scanning for the registers in the static memory whose status bits are high. It prepares a new address array that contains the address of the time slot having the valid data in static memory. Fragment eliminator module may be the bottleneck for the entire operation as at every clock cycle only one address in the static memory can be scanned. In order to avoid this bottleneck multiple fragment elimination operations run in parallel on a static memory to create a array list removing the vacant space from the static memory. The details of the readout logic is explained in algorithm 3. Readout logic module fetches the address from the array generated by fragment eliminator block and reads the data from the main memory at first. If the logic block finds that the next_add field corresponding to a time slot is NULL, then it will read the data from static memory and put it into async_fifo. On the other hand if the next_add

field corresponding to a time slot is not NULL then logic block first read data from static memory and then continue to read data in dynamic memory from the address written in the next_add field. This process will continue until logic reach to the address written in last_add field in static memory. After the read process is over, the control block changes the status of this memory bank to write enable. After writing to n^{th} static memory writing logic will write data in the first static memory again (i.e static memories are connected in circular fashion).

Algorithm 3 Algorithm for Read from memory pair

```
1: Input:=array_list1,array_list2,array_list3,array_list4,array_list5
2: Output:=D_out;
3: Variable:=Y,j,W;
4: Scan status of all five static memory;
5: if ((wr_en_Fifo='1') && (Full='0')&& (rd_en_Fifo='0')) then
6:     if (rd_en(stat_mem_num)='1') then
7:         Y=Maximum size of (array_list(stat_mem_num));
   i=stat_mem_num;
8:         for (j=1, j<=Y, j++) do
9:             if (sta_mem_i(j).next_add=NULL) then
10:                D_out=sta_mem_i(j).data;
11:            end if
12:            if (sta_mem_i(j).next_add≠NULL) then
13:                D_out=sta_mem_i(j).data; W=sta_mem_i(j).next_add;
14:                while (W≠sta_mem_i(j).last_add) do
15:                    D_out=dyn_mem_i(W).data;W=dyn_mem_i(W).next_add;
16:                end while
17:            end if
18:        end for
19:        rd_en(stat_mem_num)='0';wr_en(stat_mem_num)=1;
20:    end if;
21: end if;
```

During the hardware implementation of static and dynamic memory space block RAM is used. Integration of MMM with PCIe module is shown in Figure 4.14 and detailed data transfer mechanism from Async_fifo to PC using PCIe and DMA is already explained previously in this chapter. Data is written into the FIFO at a frequency of 52 MHz at which different logic blocks are running on CIM and data will be read from the FIFO at a frequency of 125 MHz at which PCIe core is running.

4.2.3 Overview of the data flow

Figure 4.15 shows the complete mechanism for the frame generation and the error correction procedure. As we have mentioned previously OIB can read six bits at a clock cycle from a single E-link, maximum six E-links can be attached to one OIB. Thirty-six bits data along with four dummy bits will be scrambled using four 10 bit scramblers. Then 40-bit data along with 4-bit header is mapped into the concatenated coding block. After encoding, 49 bits FEC field is appended to the 44-bit data and the 93-bit data frame is generated from the encoding block. To use the available clock frequency and the data width within the Xilinx Transceiver, three zero bits are appended, and 96-bit data is generated. From the encoding block, the data will enter into the helical interleaver block. Within this block, only 44 data bits will be interleaved. Then 96-bit data will be written into the gearbox at 52 MHz clock frequency, and the same data is read at 312 MHz clock frequency with 16-bit data width. Hence, the writing speed (52×96 = 4.992 Gbps) and reading speed (16×312 = 4.992 Gbps) is same. Now during data transfer from CIM to host PC, Gen2 eight lane PCIe is used. As output of deserializer in CIM is 40 bit payload size for TLP should be two data width (DW) or 64 bit. On a Gen2 link with x8 lanes, this raw link layer runs at 5 Gbps × (8/10) x 8 = 32 Gbps. For 64 bit or 8 byte payload header size is 16 byte and 6 bytes are used as data link layer overhead. Hence to transmit 8 byte data packet size for transmission over PCIe is 30 byte. Then the link rate is limited to 32 Gbps × (8/30) = 8.533 Gbps. As within 64 bit payload valid data is only 40 bit original data transfer rate is 8.533 Gbps × (40/64) = 5.33 Gbps. Hence, our proposed DAQ system can support data rate~ 5Gbps.

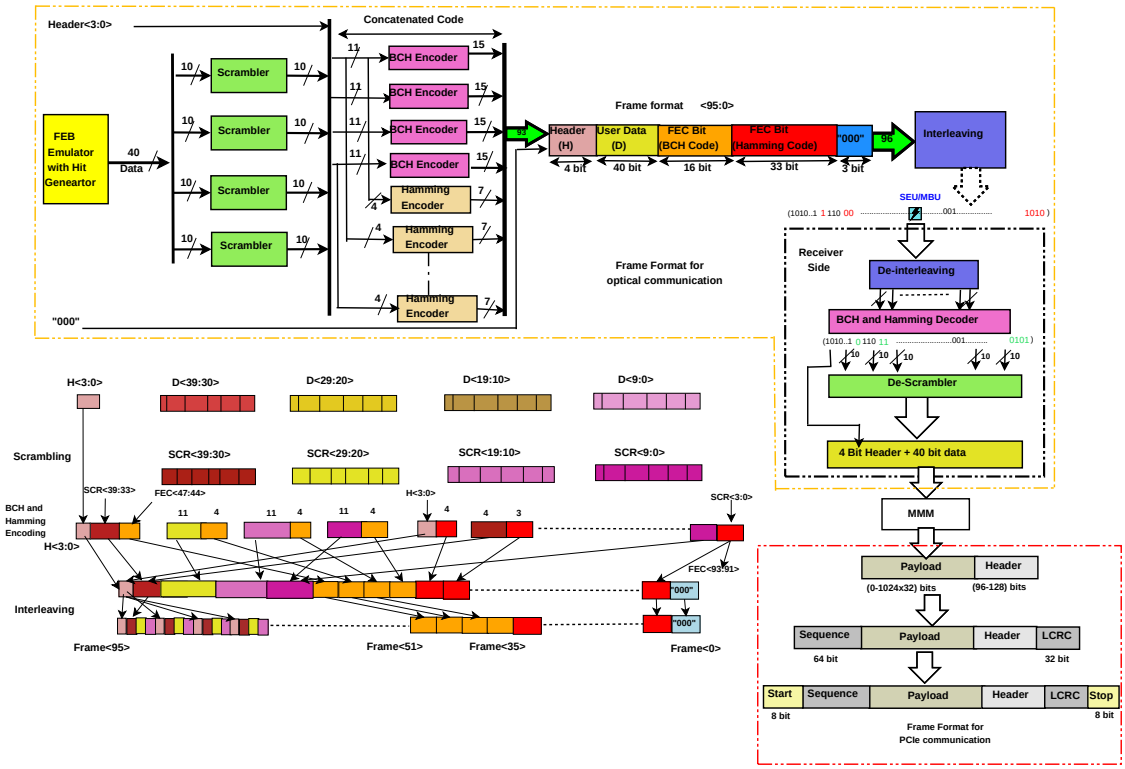

Figure 4.15: Data flow through optical fiber and PCIe

IPBus protocol over one gigabit Ethernet is used for monitoring of different registers in OIB and CIM. At the same time it also helps for sending the command and special characters to the FEB from the host computer that helps in the synchronization between FEB and optical module and reset process of FEB emulator. Details implementation of IPBus protocol over one gigabit ethernet is described in chapter 3. During the synchronization between FEB and optical module, the special characters send by the FEB will reach to the CIM through the optical fiber and from there to the host computer through Ethernet connected with CIM. A Python script written in the host computer checks the status of the received characters and calculate the delays and send the responses to FEB through the downlink. The registers whose values need to be monitored will be connected to the slave interface as illustrated in Figure 4.3. In this design PCIe interface is used for data transfer and Ethernet interface is used for controlling and internal monitoring. As data transfer is high-speed process and control and monitoring is the low-speed process we have separated these two processes.

4.3 Latency Optimization

Latency is the time difference between input of data or signal to the system and output from the system. In general, large electronics system consists of different memory elements for phase and delay alignment that may add uncertain amount of latency during data processing. In order to fix the latency these memory elements need to be bypassed but their functionalities should be implemented by proper external circuit. Fifo within the transceiver are used to align phase and frequency of clock in PCS and PMA domain. We have discussed internal architecture of MGT in details in chapter 3 and here we have shown different clock domains and fifo to align their phase and frequency in the transmitter and receiver in Figure 4.16. Tx data path contains two parallel clock domains: PMA parallel clock domain (XCLK) and PCS parallel clock domain (TXUSRCLK) as shown in Figure 4.16 (a). During the data transmission from PMA to parallel input serial output (PISO) frequency and phase of XCLK and TXUSRCLK must match. Tx Phase adjust fifo perform this function and when it will be bypassed external Tx phase aligner circuit must be used as shown in Figure 4.16 (a). Similarly, in the receiver side also XCLK and RXUSRCLK must be matched in phase and frequency before data transmission from serial in parallel out (SIPO) register to PCS and elastic buffer perform this function. In Figure 4.16(b), external Rx phase and delay alignment circuit is used to reduce the latency bypassing the elastic buffer.

Phase alignment in transceiver must be performed in the following situation:

- After every power on reset in the transceiver.
- Resetting or powering up QPLL and CPLL in the transceiver.
- When line rate in Tx or Rx is changed.
- When phase and frequency of transceiver's reference clock is changed.

Kintex FPGA uses multi-lane transceiver [127] in which multiple circuit lanes in parallel are used to carry data to optical transducer. In such high density transmitter re-timer circuit is used to eliminate the effect of coupling from adjacent lanes. In multi-lane GTx transceiver Tx and Rx buffer bypass and phase alignment must be performed manually and phase aligner circuit is used master

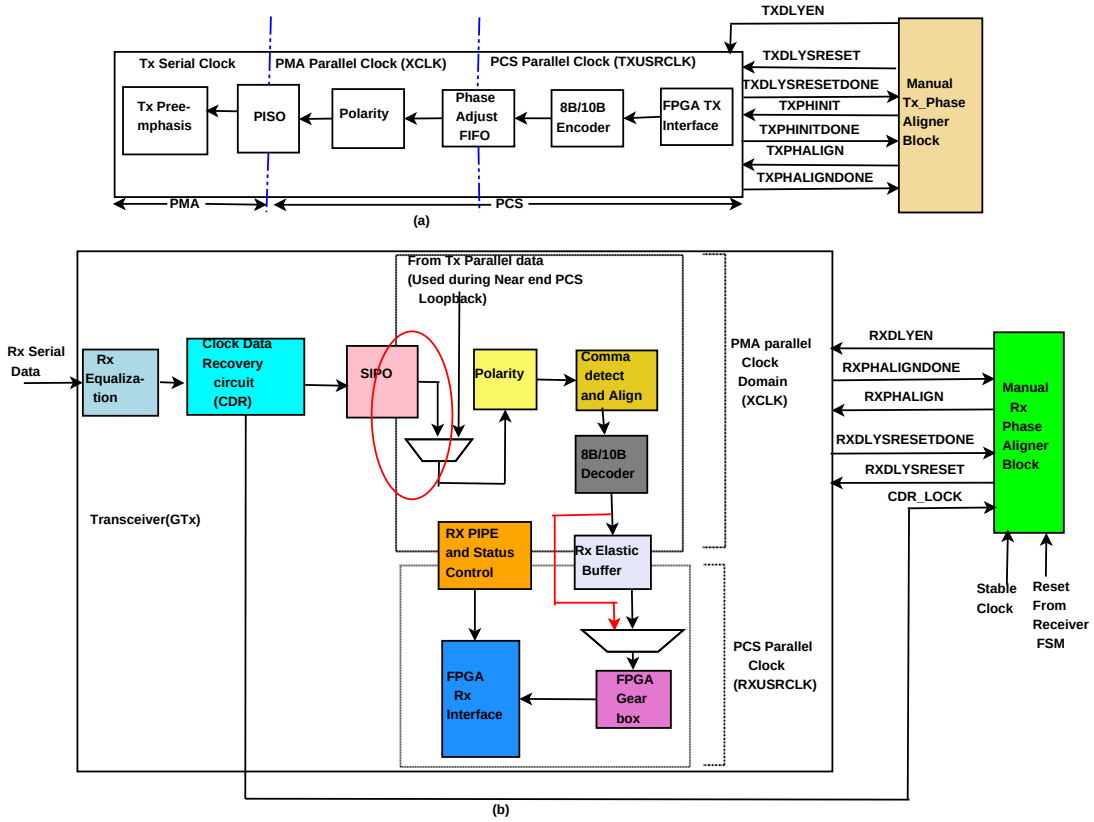

Figure 4.16: (a)Different clock domains in the transmitter (b) Different clock domains and buffer bypass strategy in receiver

slave architecture. Within the multi-lane architecture the lane which uses TX-OUTCLK acts like master and the lanes which use TXUSRCLK generated from TXOUTCLK will act like slaves. The parameters of GTX transceiver [107] to be set during latency optimization is briefed in Table 4.2.

A flow diagram in Figure 4.17 gives details of manual phase alignment in the transmitter after bypassing the Tx phase adjust fifo. In the receiver also same steps are followed for phase and delay alignment bypassing the elastic buffer. For Rx phase alignment name of the signal used in the Figure 4.17 will remain same, only 'Tx' will be replaced by 'Rx' like TXPHDLYSRESET will be replaced by RXDLYSRESET. Another major modification for Rx phase alignment data flow is that there is no such RXPHINIT and RXPHINITDONE signal in receiver corresponding to TXPHINIT and TXPHINITDONE signal in transmitter. Meaning of the signals used in Figure 4.16 and Figure 4.17 are explained in Table 4.3.

Table 4.2: Parameters of the MGT to be set during latency optimization

Transmitter side		Receiver side	
parameter	value	parameter	value
TXBUF_EN	False	RXBUF_EN	False
TX_XCLK_SEL	Selects source of XCLK	RX_XCLK_SEL	Selects source of XCLK
TXOUTCLKSEL	011 or 100 to select reference clock	RXOUTCLKSEL	010 to select recovered clock as source of RXOUTCLK
PCS_RSVD_ATTR[1]	0	PCS_RSVD_ATTR[2]	0
		RXSLIDEMODE	PCS
		RXDDIEN	1

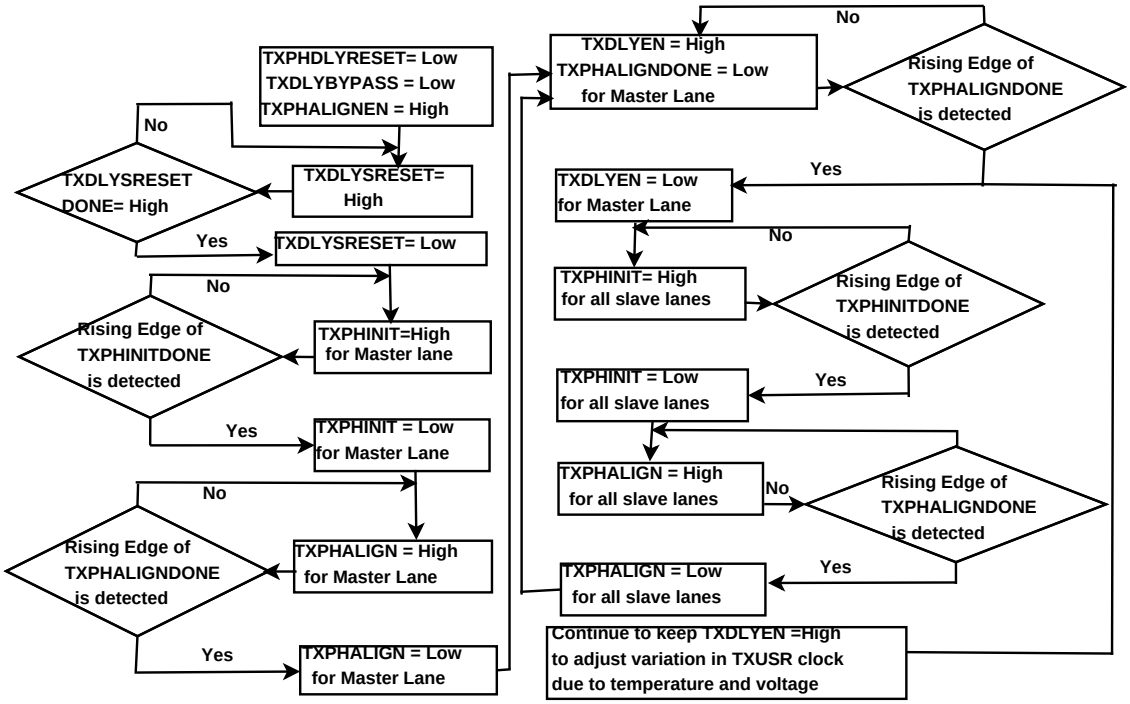

Figure 4.17: Flow diagram for manual phase alignment in the transmitter after bypassing the phase adjust FIFO

Table 4.3: Details of the signals used during latency optimization

TX/ RX PHDLYRESET	Tx or Rx phase alignment hard reset
TX/ RX DLYBYPASS	'0' when Tx or Rx delay alignment circuit will be used '1' when delay alignment circuit is bypassed
TX/RX PHALIGN	Tx or Rx phase alignment circuit
TX/ RX DLYSRESET	Tx or Rx delay alignment soft reset
TX/ RX DLYSRESETDONE	indicates that Tx or Rx delay alignment soft reset complete
TXPHINIT	Tx Phase alignment initialization
TXPHINITDONE	Indicates Tx Phase alignment initialization complete
TX/ RX PHALIGNDONE	Indicates Tx or Rx phase alignment complete
TX/ RX DLYEN	Tx or Rx delay alignment circuit enable in manual mode

4.4 Error Mitigation in FPGA devices

Here we have used scrubbing to mitigate the error in the configuration memory of FPGA. There are two types of scrubbing: blind scrubbing where externally stored golden copy periodically writes into the configuration memory and readback scrubbing in which configuration memory will be refreshed by golden copy only if error is detected in the readback file. Here we have used readback scrubbing where 32 bit CRC is used to detect errors in the readback configuration file. Hardware implementation of the proposed error mitigation scheme for configuration memory of FPGA is shown in Figure 4.18. The design is partitioned into two parts: static area and configuration area. The static area consists of Master Internal Configuration Access Port (ICAP) controller, Slave interface controller, Error detection block, Hardware ICAP (HWICAP) and ICAP interface. Configuration area consists of the application which is to be configured during the runtime. Golden copy or original bit file of the application is stored separately in a Radhard secondary memory on FPGA device. After power is on, bit file will be downloaded from the Radhard secondary memory into the configuration memory through ICAP port. Slave interface block gets the controlling information from the master. Master sends multiple informations namely ICAP_start, bit addresses, and bit length. ICAP acknowledges master using ICAP_done port while dynamic configuration process is done. The HWICAP is an interface to the ICAP. During the bit file downloading TXFIFO inside the HWICAP reads configuration data from the Block RAM and sends it to the configuration memory. Similarly, during read back RXFIFO reads configuration data from configuration memory and sends it to the block RAM inside error detection block.

Slave interface controller is an interface between the proposed ICAP block and master. Custom ICAP controller uses Advanced eXtensible Interface (AXI) to communicate with the proposed ICAP block. Master ICAP Controller is used to move bit files from Radhard secondary memory to the ICAP controller. In this design custom ICAP controller is used as the master. ICAP controller generates the trigger in a certain time interval (user defined) and after getting the trigger ICAP starts to read the data from the configuration memory and sends it to error detection block. If an error is detected error detection block generates a

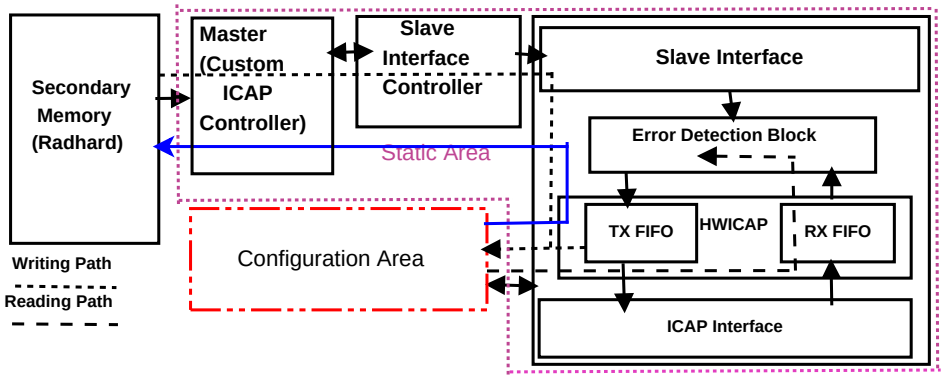

Figure 4.18: Hardware implementation of the proposed EDAC model

signal to the ICAP controller and the ICAP controller downloads the golden copy from the Radhard secondary memory. The flow diagram shown in Figure 4.19 describes the steps involved in the error mitigation in the configuration memory using readback scrubbing.

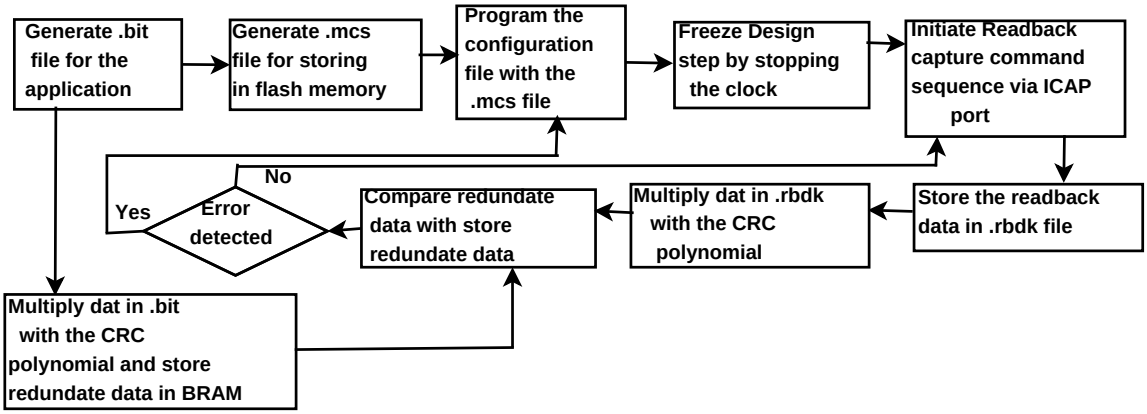

Figure 4.19: Flow diagram for error mitigation using readback scrubbing

4.5 Results and Performance Analysis

The full prototype has been implemented on Xilinx Kintex-7 FPGA board. External clock to drive MGT has been taken from jitter cleaned clock source (CDCE62-005EVM of TI). Here maximum data rate of 5 Gbps has been achieved. Full test setup of our proposed DAQ system is shown in Figure 4.20. Error correction capability of the proposed orthogonal concatenated code is compared with that of other coding schemes like single and double bit error correcting BCH code, concatenated code consisting of BCH and Hamming code. All the above men-

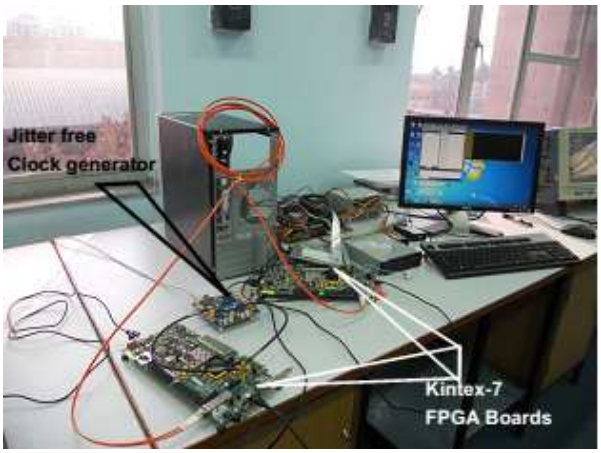

Figure 4.20: System for testing the proposed DAQ system using KC705 and external clock generator

tioned coding schemes are unidirectional in nature whereas our proposed orthogonal concatenated code corrects error along both row and column of a matrix in parallel. At the same time coding efficiency of double bit error correcting BCH code is $38/90 = 42.2\%$ and for proposed orthogonal concatenated code is $44/93 = 43.01\%$. Hence, we are not losing much in terms of coding efficiency. Figure 4.21 shows the BER performance of the proposed error correcting scheme is better compared to the other schemes. Simulation of the BER for the above

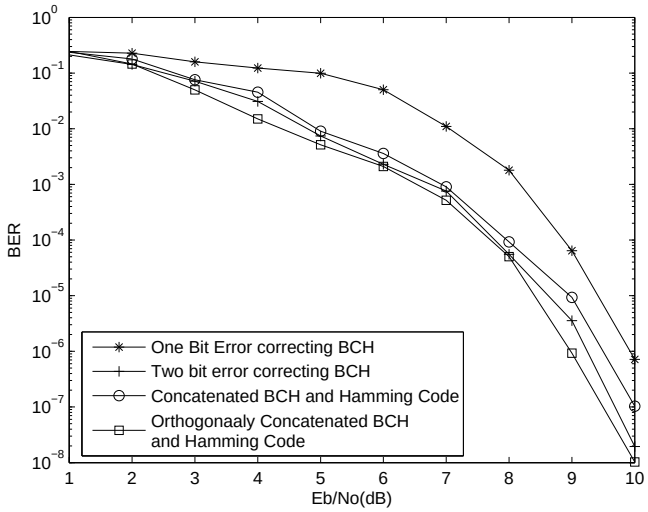

Figure 4.21: Comparison of BER performance of different coding schemes with our proposed coding

mentioned system has been done using Matlab. Noise in optical network obeys Poisson statistics [128] so, the noise (i.e., $\frac{Eb}{No}$ where Eb is the bit energy density, and No is the power spectral density of the noise) is generated using Poisson distribution during simulation. As we have mentioned IPBus protocol over one gigabit Ethernet is used for controlling DAQ chain. Python scripts are used to capture data from Ethernet port of host PC. Before starting of data acquisition E-links are need to be synchronized and each steps of synchronization is shown in Figure 4.22. For synchronization different special characters like SOS, K28.1,

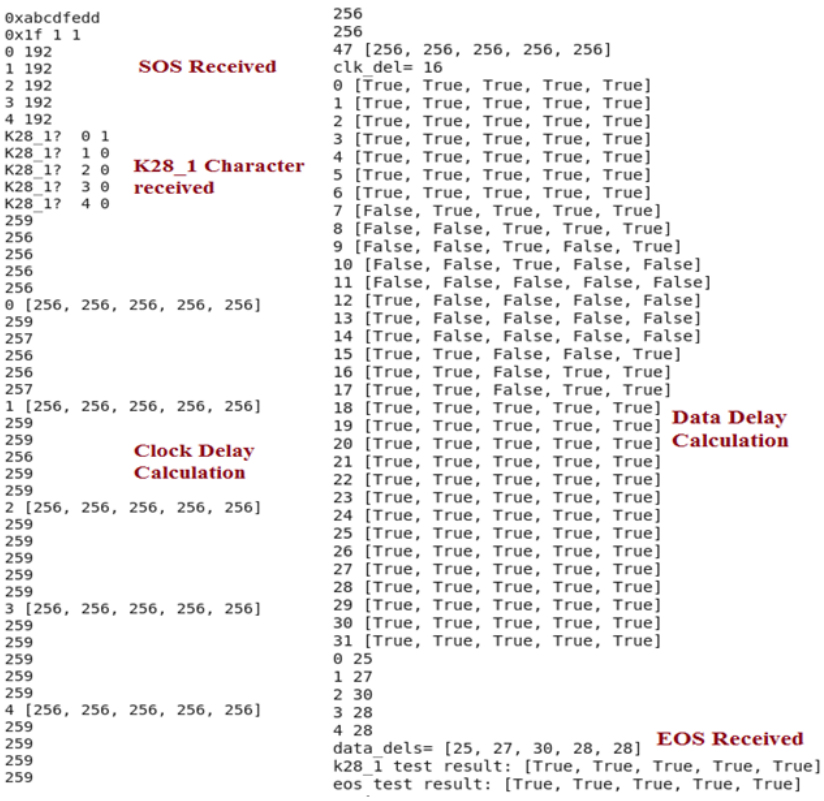

Figure 4.22: Synchronization over Elink between FEBs and OIB

K28.5 and EOS are required as shown in Figure 4.22 and the K28.1 character is mainly used to fix the path delay between different E-links. After the completion of synchronization process hit generator in the FEB emulator generates hit data.

We have tested the efficiency of our proposed error correction model for the configuration memory injecting a different number of faults (starting from 500

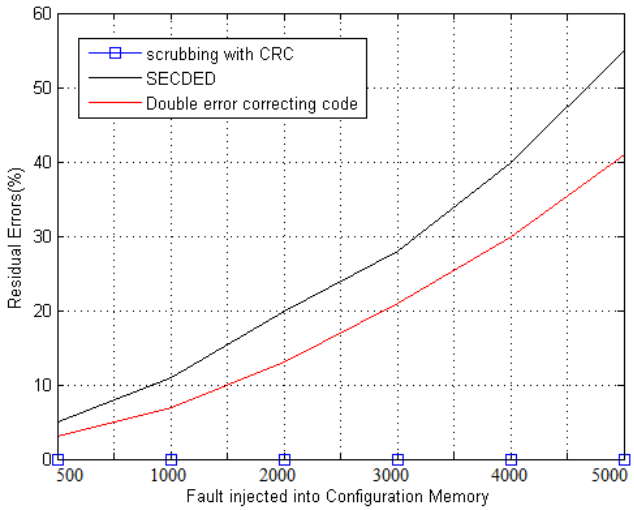

Figure 4.23: Presence of residual error after error correction with different error correcting codes and scrubbing with CRC

to 5000) randomly into the configuration memory through ICAP interface using a fault injector and compared its efficiency with Hamming and BCH code. Figure 4.23 compares different models in terms of residual errors for different number of injected errors. Residual errors indicate the presence of error in the configuration memory after error correction. During scrubbing if CRC detects the error then original bit file will be downloaded without checking the number of erroneous bits, so residual error is always zero in this method. Hamming code can correct single bit errors and only detect double bit errors. On the other hand BCH can correct two-bit errors and hence, BCH code gives good performance compared to Hamming code. There is always a chance for the presence of an undetected error when ECCs are used to detect or correct errors in the configuration memory which increases the chance of residual errors. This is more prominent when the number of injected faults are high. Scrubbing with CRC gives the best performance compared to the Hamming or BCH code as shown in Figure 4.23. To test the efficiency of our proposed error correcting models we have used another parameter called Availability of FPGA devices that measure fault repairing time for the configuration memory as shown in Figure 4.24. It can be defined as in

equation 4.1.

$$Availability = \frac{1}{1 + \frac{MTTD+MTTR}{MTTF}} \tag{4.1}$$

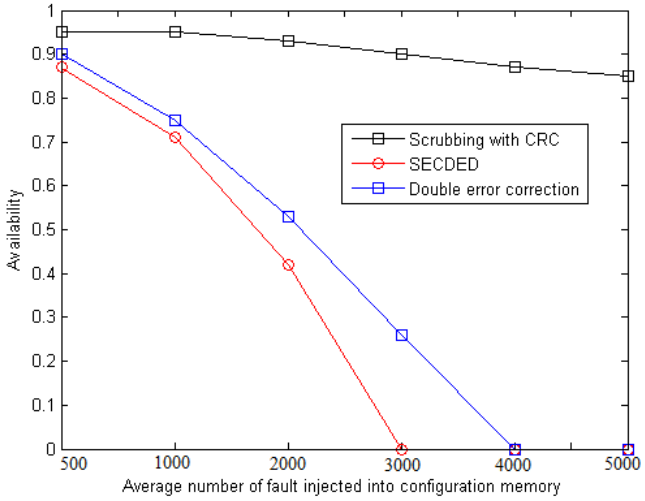

Figure 4.24: Comparison of availability of FPGA devices after error correction using different error correcting schemes and Scrubbing with CRC

Here Mean Time to Failure (MTTF) is defined as the time for continuous error free operation of a memory unit, Mean Time to Detect (MTTD) is the time interval between the error occurred due to the bit flip in the memory unit and the time at which it is detected. Mean time to repair (MTTR) is the average time for error correction. When there is no error MTTD and MTTR will be zero and availability is one that means all the FPGA devices are available. EDAC time in configuration memory of FPGA devices is different for different EDAC models. Hence, MTTD and MTTR will be different for scrubbing with CRC, Hamming and BCH code. When error correcting models unable to detect or correct errors in spite of the presence of errors in the configuration memory MTTD and MTTR becomes infinite, and availability becomes zero. As error correction time is zero in scrubbing with CRC it has less MTTD and MTTR value compared to Hamming or BCH code based model. Hence, the availability of FPGA devices are always high when it uses scrubbing with CRC compared to Hamming or BCH code based models. There is always a chance of the presence of residual errors in Hamming

or BCH code based models so availability may be zero for higher values of errors but in scrubbing based model availability is never be zero as shown in Figure 4.24.

Results of our proposed latency optimization method for high speed communication is shown in Figure 4.25. Figure 4.25 (a) shows that before latency optimization there is a delay between Frame Clock transmitted and recovered clock in the receiver of optical communication module. This delay is resulted as the recovered clock has to go through the elastic buffer for clock domain crossing. When latency optimization method is applied recovered clock need not to

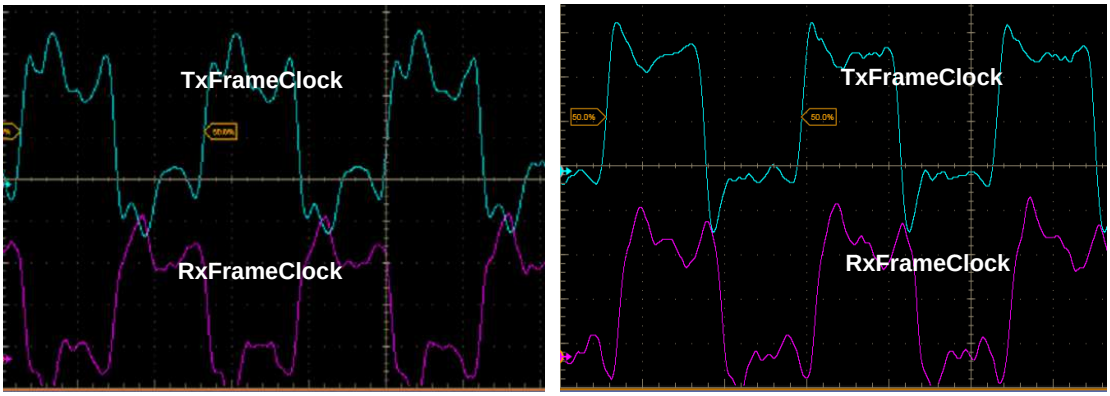

Figure 4.25: (a) TxFRameClock and RxFrameClock before latency optimization (b) TxFRameClock and RxFrameClock after latency optimization

go through any elastic buffer as the same function is done externally. This will reduce latency by 12 ns as shown in Figure 4.25(b).

We have also tested performance of the MMM using simulation and results of simulation are shown in Figure 4.26. To test the algorithm, we have generated one data pattern for every clock pulse. Clock and input data in the simulation result is represented by clk and $dina\_test\_in[29:0]$. Pattern generator generate data with 1 us randomness. Address of the static memory and dynamic memory are represented by sta_addra and dyn_addra respectively and if data is already present in static memory address will be fetched from dynamic memory. Signal mem_fill decides whether data will enter into static memory or dynamic memory and it is mainly derived from status information of static memory. $count\_data[0:4]$ indicates that address in some of the register is incrementing that represents write operation and address in some of the register is decrementing that represents

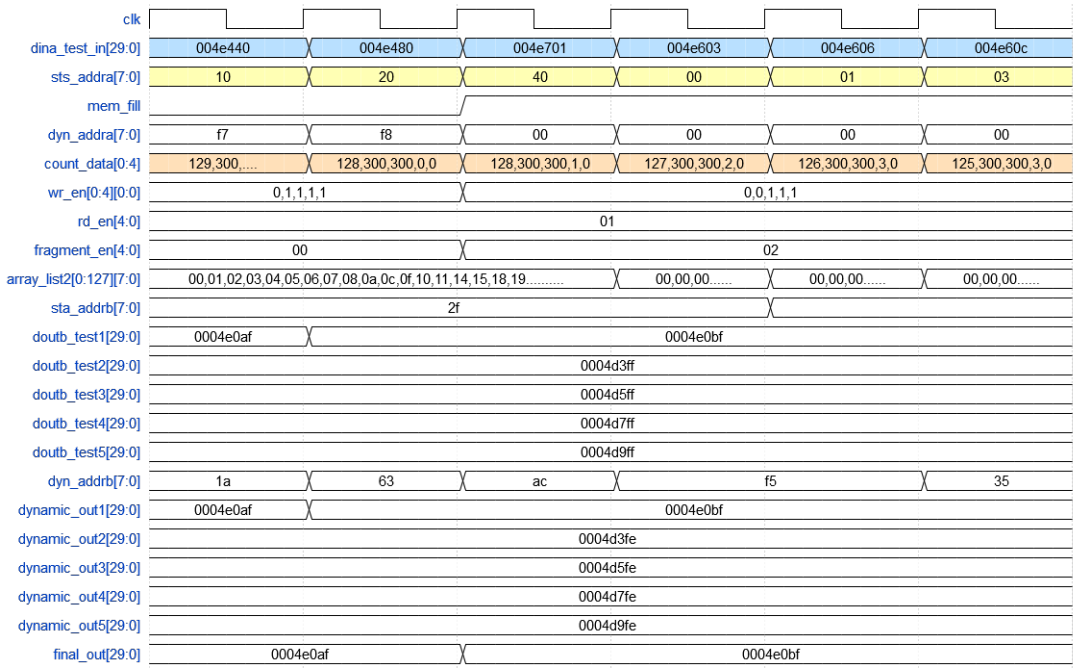

Figure 4.26: Simulation result for memory management module

read operation. It can be seen that with each clock pulse data is writing on one memory bank while data is read out from another memory bank so there is no wastage of clock pulse. $fragment\_en$ represents the number of static memory for which rd_en and wr_en signal are disable and fragment elimination process starts to work. Here fragment elimination process started on 3^{rd} memory bank. In the output section, data is readout serially both from static as well as dynamic memory. Data is always read out first from the static memory and is represented by $doutb\_test[29:0]$ and then data will be read out from the dynamic memory if repetition is there which is represented by $dynamic\_out[29:0]$. In Figure 4.26, read enable signal of 0^{th} memory pair is enabled and hence data is present in $doutb\_test1[29:0]$ for 0^{th} static memory and $dynamic\_out1[29:0]$ for 0^{th} dynamic memory. Finally rearranged data output stream can be seen from the the signal $Final\_out$. Table 4.4 indicates resource utilization by different modules during FPGA implementation of our proposed DAQ system. Table 4.5 summarizes different features of our proposed DAQ system and DAQ systems discussed in the existing literatures and it is very much evident that our scheme is well equipped for high speed error resilient data acquisition.

Table 4.4: Resource Utilization

	LUT	FF	I/O	BUFG
IOB	3567(1.75%)	2482(0.61%)	35(7%)	17(53.12%)
CIM	5672(2.78%)	5791(1.42%)	19(4.72%)	13(40.62%)
IPbus	2530(1.24%)	3276(0.8%)	215(43%)	2(2.65%)
FEB	1709(0.84%)	1290(0.32%)	8(1.6%)	2(6.25%)

Table 4.5: Summery of different features of our proposed DAQ system and different state of the art solutions

Device Used	**Speed (Gbps)**	**Error correcting capability**	**Line coding used**
Lattice SCM40 [48]	1.60	Not mentioned	$8b/10b$
Altera Stratix IV [49]	8.50	Not mentioned	$8b/10b$
Altera EP2SGX90EF1152C5 [51]	2.125	Not mentioned	$8b/10b$
Virtex-II Pro series FPGA [50]	1.75	CRC used for error detection only	$8b/10b$
Kintex-7 [129]	4.80	Reed Solomon code	Scrambler
Kintex-7	**5**	**Orthogonal Concatenated code**	**Scrambler**

4.6 Conclusion

In this chapter we have proposed a novel DAQ design for HEP experiments. The proposed DAQ can work in radiation zone and supports high-speed optical data communication with multi-bit error correction, efficient memory management for easy data processing. The proposed DAQ system is interfaced with a front-end electronics emulator board. The DAQ design has been implemented on Xilinx Kintex-7 board and real test setup has been developed involving board to board communication, board to computer interface over Ethernet for controlling and PCIe interfacing with a host computer for data transfer. A detailed performance analysis of the DAQ implementation is presented in terms of BER performance of the proposed multi-bit error correction code, robustness in radiation zone and controlling using IPbus protocol. In the next chapter we are planning to discuss error mitigation technique in storage memory element.

Chapter 5

Latency optimized clustered error correction for mult-level memory chips using LSBCPC

ECCs are commonly used to mitigate the effect of soft errors arising due to MBUs in physically adjacent cells in flash memory. The necessity of the use of complex ECCs increase many folds with the use of multilevel memory cells (MLC) in highly densed solid state memory devices (SSMD) where each memory cell can store more than one information bit. The probability of formation of adjacent MBUs or clustered error increases with the reduction of noise margin due to partitioning of the threshold voltage of MOS transistor into multiple levels. With the increase of the complexity of ECCs overhead due to redundant bits and latency the error correction time also increases. Again single error correcting Hamming or Hsiao code are also not suited against MBUs. We propose a latency optimized clustered error correction technique using linear shortened block code based product code (LSBCPC) that promises correction of higher number of adjacent erroneous bits compared to other multi-bit ECCs. Here, we have applied a technique called shortening of the code on the component block codes of the product code in our proposed method to enhance EDAC capability compared to the traditional multi-bit ECCs. Though the proposed method has higher overhead compared to multi-bit error correcting RS or BCH code, it offers simple decoding circuit,

higher error correction coverage with enhancement of the lifetime of the flash memory devices. We have measured the performance of the proposed method in terms of redundant bits, error correction coverage, hardware area and decoding latency.

5.1 Introduction

Due to low cost, absence of external refresh circuitry and its nonvolatile nature flash memory chips are now a days used as permanent storage devices in different applications. Flash memories are commonly used as USB flash drives, internal storage devices in different electronics gadget *etc.* There are mainly two types of flash devices: NAND and NOR flash. NOR flash has higher read speed and random access capabilities, which makes it suitable to store data in computer and mobile. Due to the limitation of scaling in the dimension of MOS transistors using NOR technology, memory density can not be increased above a certain threshold. On the other hand fast write and erase capability and higher memory density makes the NAND based flash memory more suitable for storage of large amount of data. In this chapter we will mainly focus on adjacent MBU or clustered error mitigation in the NAND flash memory. MLC techniques in flash device helps a memory cell to store more than one information bits using floating gate technology by partitioning threshold voltage of MOS transistor into multiple levels. MLC NAND flash that can store two bits information is very common while three and four bits per memory cell of a NAND flash memory are reported in recent literatures [130], [131].

Though NAND flash has flourished in modern memory technology it has some inherent problems like limited number of writes [132], [133], error during erase operation and leakage current [134], accumulation of corrupted blocks and disturbance during read and write operation. Reduction in memory cell size and gap between the threshold voltage levels (i.e reduction of noise margin of threshold voltage) degrade system performance. In order to protect the flash memory from different types of error and enhance the memory life time efficient memory management algorithms, workload distribution algorithm and garbage collection methods are now a days using in modern flash memory. Though different flash

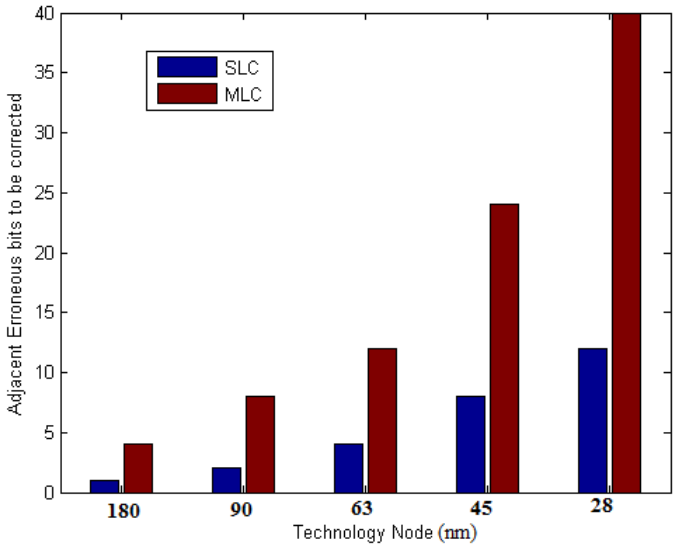

Figure 5.1: Number of adjacent erroneous bits for MLC and SLC with different technology node [138]

management algorithms help to improve system performance, they are not efficient to mitigate the effect of soft errors arising due to MBUs and SBUs and hence, different EDACs become integral part of the flash memory to mitigate the effect of soft errors [135]. Hamming or Hsiao code may be sufficient for single level memory cell (SLC) [136] but they are incapable to protect from the effect of soft errors in MLC flash memory. Figure 5.1 depicts the number of adjacent bits that are affected due to soft error in MLC and SLC flash with the shrinking of technology node from 180 nm to 28 nm and it clearly shows that effect of MBUs are prominent in MLC flash memory compared to SLC flash memory in the same technology node [137]. In general multi-bit error correcting BCH code [139] and its non-binary version, RS codes are used in MLC flash memory to correct MBUs. A concatenated code with BCH as inner code and trellis coded modulation (TCM) as outer code has been developed by authors in [140] to improve storage reliability of flash memory. Product codes comprising of Hamming with BCH code and Hamming with RS code are proposed in [141] that achieve better performance compared to their component codes in terms of area, latency and BER.

Choosing of error correction schemes in flash memory depends on type of errors and distribution of the errors *i.e* whether it is adjacent MBUs, discrete MBUs (when erroneous bits are discrete but not far from each other) or SBU (when the distance between erroneous bits is large). In SLC NAND flash, error distribution may be considered as random but in MLC with increased technology scaling probability of occurrence of MBUs increase. In the previous chapters we have considered errors in the memory mainly due to radiation. Here along with deposition of energy by charge particles error arises during program or erase cycles of flash memory that may create MBUs are also considered. In MLC flash devices, threshold voltage of a MOS transistor is partitioned into multiple levels and hence, charge particles with low energy or slight deviation from narrow read or write cycle can corrupt the stored bit in memory cells. We have studied two models here: fully adjacent error model or clustered error where erroneous bits are adjacent to each other and hybrid model combining adjacent and nonadjacent MBUs (discussed later). In this chapter, our key contributions are:

- An efficient latency optimized product code with component codes as shortened linear block code is proposed to mitigate the effect of MBUs due to both clustered and almost clustered error in MLC NAND flash memory.
- A novel hardware architecture is proposed using pipelining and parallel processing to reduce overall latency during EDAC.
- Error correction efficiency, mean time to failure (MTTF), decoding complexity of our proposed algorithm is calculated using the data taken from a real experiment.

5.2 MLC NAND FLASH Memory Background

In 1989, Toshiba had first introduced NAND flash memory in the semiconductor industry and a now-a-days different vendors like Samsung, SanDisk, Micron, SK Hynix and Intel manufactured flash devices. Instead of slower performance compared to DRAM, endurance and reliability problem with higher density, fast programming and erase speed and lower cost per bit helps to increase the demand

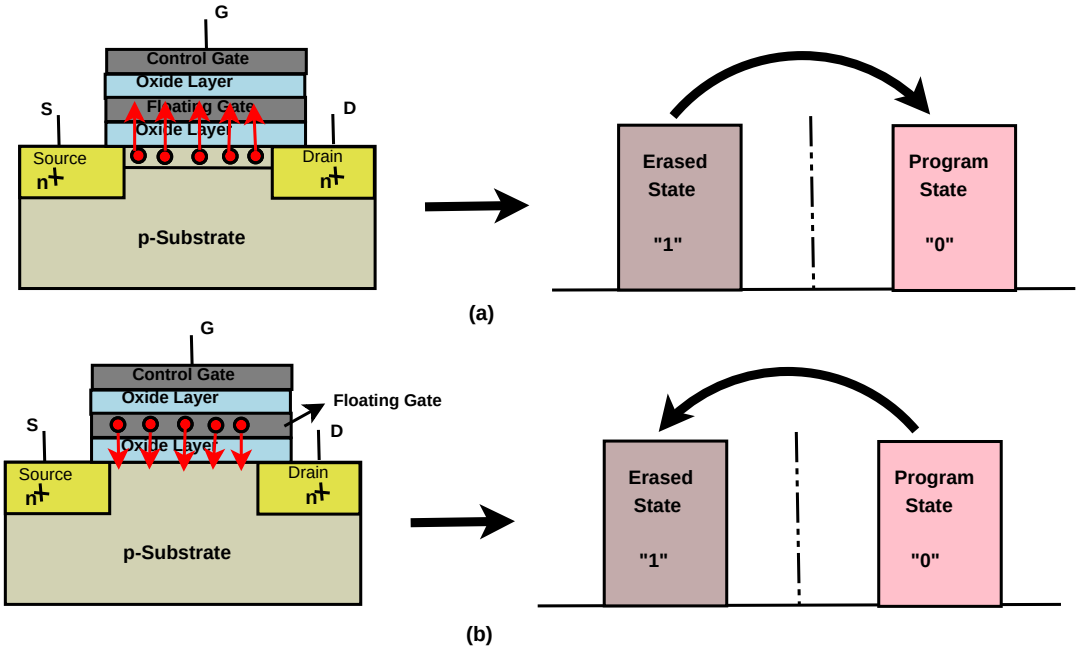

Figure 5.2: Programming and Erasing a Floating Gate Transistor

of NAND flash [142]. NAND flash basically works using floating gate transistor technology. In floating gate MOSFET two gates are used: control and floating gate as shown in Figure 5.2. Floating gate is isolated from any electrical connection by oxide layer in both sides and hence any charge within it remains trapped and trapped charge can be altered by either using hot carrier injection mechanism or Fowler-Nordheim tunneling method [143]. If no voltage is applied to floating gate, the device operates like normal MOSFET where a positive voltage in control gate above threshold creates channel and conduction can be started between source and drain by proper voltage at source and drain. When a high positive voltage is applied to floating gate, electrons from the source, drain and channel tunnel through the oxide and get trapped within the floating gate. This state is called programmed state and designated by "0" as shown in Figure 5.2(a). This in turn increases the threshold voltage to form the channel and on the device. When a negative high voltage is applied to push back the trapped electron into the substrate then this operation is known as erase operation. The state after the erase operation is known as erased state and designated as "1" and shown by Figure 5.2(b). When threshold voltage of a MOSFET in a NAND cell has only

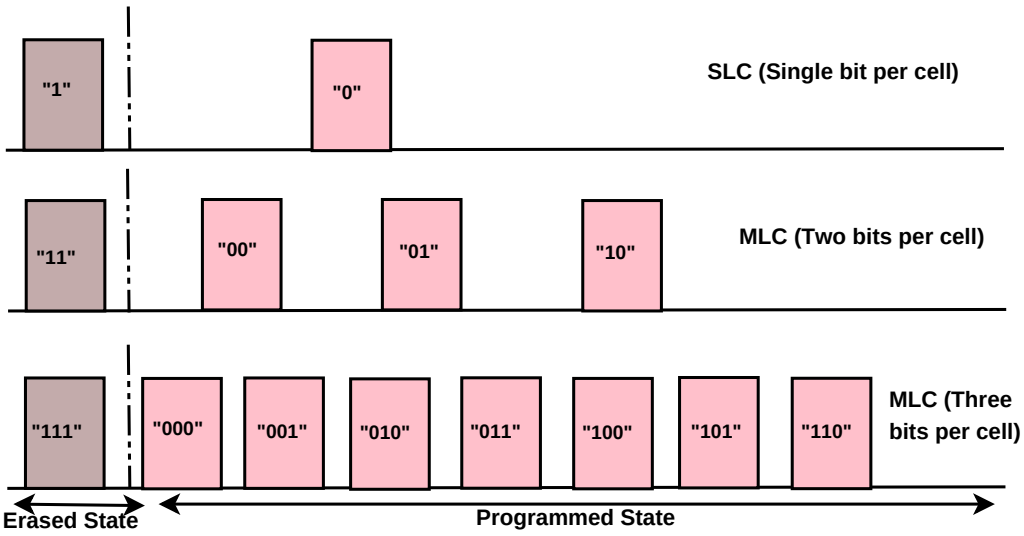

Figure 5.3: Threshold voltage distribution of SLC, MLC with two and three bits per cell

two state it can store only one bit per cell and known as SLC. On the other hand if threshold voltage has more than two states like four states (two bits per cell) or eight states (three bits per cell) then it is known as MLC. In general, when the threshold voltage has 2^k level, MOSFET can stores maximum k bit information. Threshold voltage distribution for SLC and MLC with two and three bits per cell is shown in Figure 5.3.

The smallest unit in the flash memory that can be erased at a time is known as block. Typical block size in flash memory can be 512 Kb, 256 Kb and 128 Kb. Each block comprises of multiple pages and typical page size is 2-16Kb. A number of blocks (typically 1024) form a memory bank as shown in right side of Figure 5.4. Page is the smallest unit on which read and write operations can be done. Different error mitigation techniques are applied on a data set in a page and generated redundant bits are stored on the same page. MOSFET with floating gates are arranged in a matrix as shown in Figure 5.4 and floating gates of all the MOSFETs along the columns are connected through a common line known as "Bit-Line (BL)" whereas gates of all MOSFETs along the rows are connected through a common line known as "Word-Line (WL)". MOSFETs which are connected to a common WL form a physical page and each physical page corresponds to a single logical page in case of SLC and multiple logical pages

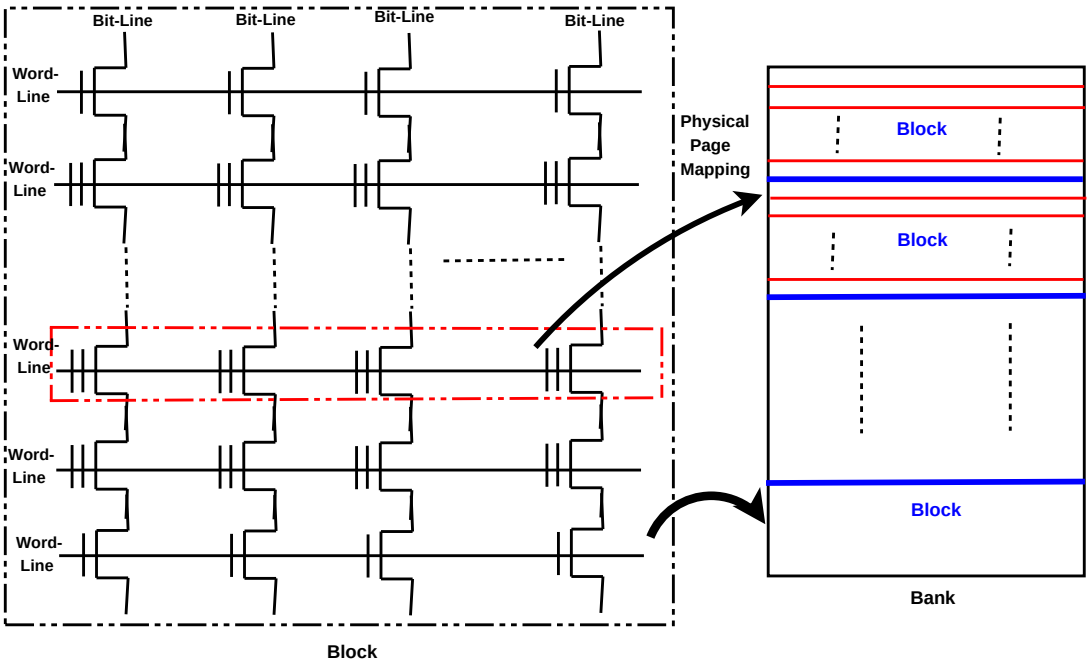

Figure 5.4: Organization of Bank, Block and pages in the MLC flash memory

in case of MLC.

In order to program a page we have to select the "WL" line corresponding to that page (i.e make WL high) and to select a cell on this page, "BL" line corresponding to that cell connects to ground. In this condition the particular cell is programmed to store "0" into it. On the other hand to store "1" into a cell both "WL" and "BL" lines corresponding to the cell should be connected to Vcc. This condition inhibits the cell to be programmed and keeps it into erased state. During the read operation initially "BL" line is to be selected and then apply a voltage V_R whose value is between the erased and program state to the corresponding "WL" line. If the MOSFET is in the erased state, threshold voltage is less than V_R and if it is in the programmed state, threshold voltage will be higher than V_R. During erase operation substrate is connected to high voltage and gate is connected to low voltage which helps to bring back the trapped electrons into the substrate from floating gate.

There are two options of programming in each page of flash memory: single page programming (SPP) and multi-page programming (MPP). In SPP [144], all bits of a cell is in the same page so that they can be programmed at the same time.

On the other hand, in MPP [145], bits of a memory cell are in different pages so that they are programmed at different instant. As for example, when three bits of a Triple-Level-Cell are in three different pages, they will be programmed using MPP and when they are in the same page they will be programmed using SPP. Maximum NAND flashes in general, use MPP due to its higher programming speed but SPP also has some advantages like tighter threshold voltage window that is very helpful in the era of device scaling. As in SPP all the bits of a cell are on the same page so there is more probability of occurrence of adjacent cell upsets compared to MPP. Here we have considered error arising for both SPP and MPP in MLC flash memory with two bits per cell.

5.2.1 Error distribution in MLC Flash

In SLC flash, damage of a memory cell corrupts a single bit and creates SBU but in MLC flash memory damage of a cell corrupts multiple bits. Figure 5.5 (a) shows a single page of two-level MLC which is programmed by SPP and blocks indicated by same color form a memory cell. Figure 5.5 (b) shows a single page of a two-level MLC which is programmed by MPP and each block indicated by different colors represent data bits from different memory cells. In Figure 5.5 (b),

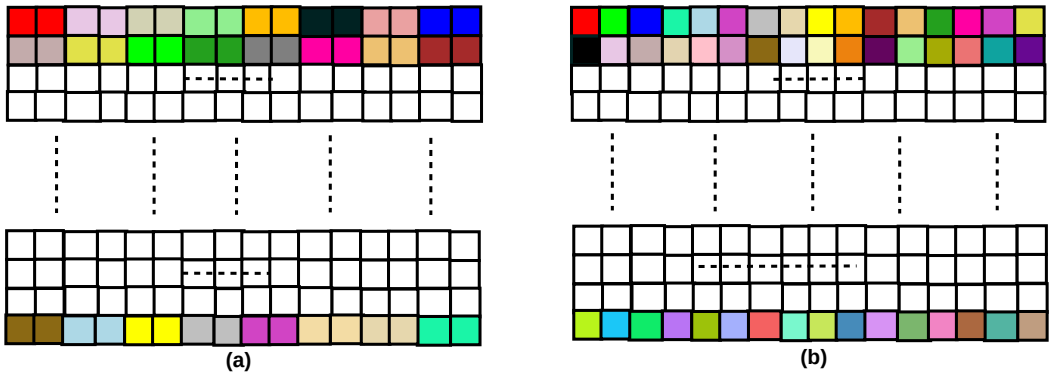

Figure 5.5: (a) Single page programming based MLC (b) Multi-page programming based MLC

adjacent MBUs affect multiple cells that is quite similar to the effects of MBUs in SLC flash memory. On the other hand, in SPP based MLC, adjacent MBUs may affect either data bits of a single cell or data bits from adjacent cells. As for example in two-level MLC flash shown in Figure 5.5(a), errors in the first

and second bits damage both the data bits of a single cell and errors in second and third bits damage two adjacent cells. Hence, MBUs create both adjacent cell and adjacent bit errors in SPP based MLC whereas MBUs in MPP based MLC generate only adjacent bit errors. Now if MBUs are adjacent to each other they form the clustered error as shown in Figure 5.6 (a) and (b) where all bits within the cluster are affected. It may so happen that the MBUs are not adjacent to each other and some bits within the adjacent erroneous bits remain unaffected, then

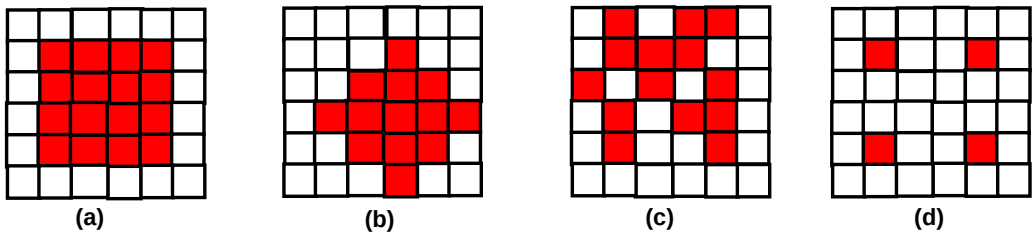

Figure 5.6: Different cluster and almost cluster patterns

the error patterns are termed as almost clustered error as shown in Figure 5.6(c), (d). Cluster size is calculated by the distance between the farthest erroneous bits within the cluster. As for example cluster size in Figure 5.6 (a), (b), (c) and (d) are 4×4, 5×5, 5×5 and 4×4 respectively. Our proposed methods can mitigate the error arises due to both clustered and almost clustered errors both in SPP as well as MPP based MLC flash memory.

5.3 Linear Shortened Block code based Product code

Product codes are a coding technique where small length codes are serially concatenated to form a long length code that has higher error correcting capabilities compared to the constituent codes. As in product codes component codes encode data orthogonally, it provides good error correction capability compared to cross parity check codes with efficient and simple decoding circuit. Consider two block codes A and B having the parameters $[n, k_A, d_A]$, $[m, k_B, d_B]$ respectively where n and m are the length of encoded data, k_A and k_B represent data lengths of A and B before encoding, d_A and d_B are the hamming distance of A and B. The

product code C can be represented by $A \otimes B$ where $\otimes$ is known as *Kronecker product* [146]. If G_A and G_B be the generator matrix of A and B then the generator matrix of the product code C can be written as $G_C = G_A \otimes G_A$. When U represents the matrix of dimension $k_A \times k_B$ and G_A^T be the transpose of G_A then C can be obtained by equation 5.1.

$$C = G_A^T * U * G_B \tag{5.1}$$

The parameters of the resulting product code C are $[mn,k_Ak_B,d_Ad_B]$ with code rate R_AR_B where R_A and R_B are code rates of A and B respectively.

Here we have used multi-bit adjacent error detecting block code as component codes. As the encoding is done using generator matrix, decoding of the product code can be done using another special kind of matrix called parity check matrix. During the decoding, readback data are multiplied with parity check matrix and generate syndrome vectors (S) that helps to detect and correct errors in the corrupted data. If H_A and H_B are parity check matrices of codes A and B, then the syndrome vectors due to row decoding (S_A) and column decoding (S_B) can be calculated using equation 5.2

$$S_A = C * H_A^T \quad \text{and} \quad S_B = C^T * H_B^T \tag{5.2}$$

When all the syndrome vectors are unique they can locate the position of the error and correct them. On the other hand, when a group of syndrome vectors represent a particular error patterns it can not correct errors but can only detect error patterns. In this chapter, error patterns are detected using the syndromes of the component code and then correct the error taking the information from the syndromes of both of the component codes instead of correcting error patterns directly using one component code. It enhances the error correction efficiency of product code against the adjacent MBUs and reduces error correction time because syndrome calculations using the component codes are done in parallel. Parity check matrix can be represented as H=$[I_{n-k},P^T]$ and then generator matrix can be derived from the parity check matrix using equation G=[P, I_k].

Here we have applied a method called shortening of the code which reduces the code length of component code keeping the redundancy fixed. Hence, a

(n,k_A) shortened code can be formed from (p,l) code such that $n - k_A = p - l$ where $k_A = 2^q$ where q is an integer. In order to generate parity check matrix for (n,k_A) block code, first choose lexicographic matrix (L) for (p,l) block code with dimension $(p-l) \times p$ in which i^{th} column contains binary value of i. From L, develop the parity check matrix for both the component codes obeying the following constraints:

- There should not be any column that contains null vector in parity check matrix.
- There may be repetition in column of parity check matrix.
- First $(p-l)$ columns of parity check matrix should contain identity matrix.
- $C_j \cap A_{1j} \cap A_{2j} \ldots\ldots\ldots A_{vj} = \phi$ where $A_{1j} = C_j \oplus C_{j+1} \forall$ j=1 to (n-1), $A_{2j} = C_j \oplus C_{j+1}\ C_{j+2} \forall$ j=1 to (n-2), $A_{vj} = C_j \oplus C_{j+1} \ldots\ldots\ C_{j+v} \forall$ j=1 to (n-v), C indicates columns of lexicographic matrix and ϕ represents null matrix. Here v is the number of the adjacent erroneous bits that needs to be detected.

Here the component codes of the product code will be used only for error detection and syndrome vectors generated due to different number of adjacent erroneous bits should not overlap (i.e different error patterns produce distinct syndrome vectors) which is indicated by 4^{th} constraint. As for example syndromes generated by two adjacent erroneous bits should not overlapped with syndromes generated by three adjacent erroneous bits but syndromes generated by two adjacent erroneous bits at different positions could be same. Pseudocode given by algorithm 4 is used to generate parity check matrix for (21,16) shortened linear block code from lexicographic matrix of (31,26) linear block code. This algorithm is basically a recursive backtracking algorithm where searching starts from a partial parity check matrix that has already satisfied the above mentioned constraints. In every recursion a column from the lexicographic matrix will be added with the partial parity check matrix and it will be checked whether updated partial parity check matrix satisfy the constraints described previously. If the the updated matrix satisfies the constraints, newly added column will be kept otherwise it will be discarded and another new column will be selected. This process will continue until

all the columns of shortened linear block code are selected and full parity check matrix is formed. Here we have given pseudocode for generation of parity check matrix for (21,16) shortened block code and generation of parity check matrix for the code having other dimension obey the same constraints as mentioned earlier. The generated parity check matrix (H_C(21,16)) for (21,16) shortened linear block code is shown in equation 5.3. It can detect upto four adjacent erroneous bits and syndrome for different adjacent erroneous bits are shown in Table 5.1. Now if (21,16) shortened linear block code is used as the component code in the product code for the data set having dimension 16×16 generated product code can correct cluster having maximum size of 4×4.

$$\begin{bmatrix} 1&0&0&0&0&1&1&0&0&0&0&0&1&0&0&0&1&0&0&0&1 \\ 0&1&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0 \\ 0&0&1&0&0&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&1&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0 \\ 0&0&0&0&1&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0 \end{bmatrix} \tag{5.3}$$

In the similar way parity check matrix for (38,32) (H_C(38,32)) and (71, 64) (H_C(71,64)) linear block codes are shown in equation 5.4 and 5.5 respectively which are formed from the lexicographic matrix for (63,57) and (127,120) block code using the same constraints mentioned previously.

$$\begin{bmatrix} 10000001000100000010000100001000010000100001 \\ 01000010100001000010000100001000010000100000 \\ 00100001000000100001000010000100001000010000 \\ 00010000010000010000100001000010000100001000 \\ 00001000001000001000010000100001000010000100 \\ 00000110000010000000000000000000000000000000 \end{bmatrix} \tag{5.4}$$

$$\begin{bmatrix} 10000001010000010000010000010000010000010000010000010000010000010000010 \\ 01000000100000100000100000100000100000100000100000100000100000100000100 \\ 00100000000010000010000010000010000010000010000010000010000010000010000 \\ 00010001000100000100000100000100000100000100000100000100000100000100000 \\ 00001000001000001000001000001000001000001000001000001000001000001000001 \\ 00000100000001000001000001000001000001000001000001000001000001000001000 \\ 00000010000000000000000000000000000000000000000000000000000000000000000 \end{bmatrix} \tag{5.5}$$

Algorithm 4 Generation of parity check matrix from lexicographic matrix

Require: p,l;
Ensure: Parity check matrix H;

```
L(p-l,p)=hammgen(5,[1 0 1 0 0 1]); D= transpose(L);
H= zeros(21,5);H_par(1 : 5, 1 : 5) = D(1:5,1:5);
twocol= zeros(20,5),threecol= zeros(19,5),fourcol=zeros(18,5);
while (m ⩽ 4) do
    twocol(m,1:5)= mod(xor(D(m,1:5),D((m+1),1:5)),2); m= m+1;
end while;
while (n ⩽ 3) do
    threecol(n,1:5)= mod(xor(twocol(n,1:5),D((n+2),1:5)),2); n= n+1;
end while;
while (r ⩽ 2) do
    fourcol(r,1:5)= mod(xor(threecol(r,1:5),D((r+1),1:5)),2); r= r+1;
end while;
u1=6; i= 6;
while (i ⩽ 21) do
    t=0;p=0;
    while (u1 ⩽ 31)&& (p=0) do
        while (z1 ⩽ 20)&& (check1=0) do
            if (twocol(z1,1:5)==D(u1,1:5)) then check1 =1;
            end if;z1=z1 + 1;
        end while;
        while (z2 ⩽ 19)&& (check2=0) do
            if (threecol(z2,1:5)==D(u1,1:5)) then check2=1;
            end if;z2=z2+ 1;
        end while;
        while (z3 ⩽ 18)&& (check3=0) do
            if (fourcol(z3,1:5)==D(u1,1:5)) then check3=1;
            end if;z3=z3+ 1;
        end while;
        if ((check1=1) or (check2=1) or (check3=1)) thencheckfinal=1;
        end if;
        c1(1,1:5)= mod(xor(D(u1,1:5),H((i-1),1:5)),2);
        if (c1(1,1:5)=[00000]) then g1=1;
        end if;
        while ((x21 ⩽(i-1))&&(f21=0)) do
            if (c1(1,1:5)=H(x21,1:5)) then f21=1;
            end if;x21=x21+1;
        end while;
        while ((x23 ⩽(i-3))&&(f23=0)) do
            if (c1(1,1:5)=threecol(x23,1:5)) then f23=1;
            end if;x23=x23+1;
        end while;
```

```
42:         while ((x24 ⩽(i-4))&&(f24=0)) do
43:             if (c1(1,1:5)=fourcol(x24,1:5)) then f24=1;
44:             end if;x24=x24+1;
45:         end while;
46:         c2(1,1:5)= mod(xor(c1(1,1:5),H((i-2),1:5)),2);
47:         if (c2(1,1:5)==[00000]) then g2=1;
48:         end if;
49:         while ((x31 ⩽(i-1))&&(f31=0)) do
50:             if (c2(1,1:5)=H(x31,1:5)) then f31=1;
51:             end if; x31=x31+1;
52:         end while;
53:         while ((x32 ⩽(i-2))&&(f32=0)) do
54:             if (c2(1,1:5)=twocol(x32,1:5)) then f32=1;
55:             end if; x32=x32+1;
56:         end while;
57:         while ((x34 ⩽(i-4))&&(f34=0)) do
58:             if (c2(1,1:5)=fourcol(x34,1:5)) then f34=1;
59:             end if; x34=x34+1;
60:         end while;
61:         c4(1,1:5)= mod(xor(c2(1,1:5),H((i-3),1:5)),2);
62:         if (c4(1,1:5)=[00000]) then g4=1;
63:         end if;
64:         while ((x41 ⩽(i-1))&&(f41=0)) do
65:             if (c4(1,1:5)=H(x41,1:5)) then f41=1;
66:             end if; x41=x41+1;
67:         end while;
68:         while ((x42 ⩽(i-2))&&(f42=0)) do
69:             if (c4(1,1:5)=twocol(x42,1:5)) then f42=1;
70:             end if; x42=x42+1;
71:         end while;
72:         while ((x43 ⩽(i-3))&&(f43=0)) do
73:             if (c4(1,1:5)=threecol(x43,1:5)) then f43=1;
74:             end if; x43=x43+1;
75:         end while;
76:         if ((f21=1)||(f23=1)||(f24=1)||(f31=1)||(f32=1)||(f34=1) ||
77: (f41=1)||(f42=1)||(f43=1)) then final =1;
78:         end if;
79:         if ((g1=0)&&(g2=0)&&(g3=0)&&(final=0)&&(checkfinal=0)) then
80:             H(i,1:5)= D(u1,1:5);twocol((i-1),1:5) = mod(xor(D(u1,1:5),H((i-
    1),1:5)),2);
81:             twocol((i-1),1:5) = mod(xor(D(u1,1:5),H((i-1),1:5)),2);
82:             threecol_mid(1,1:5)= mod(xor(D(u1,1:5),H((i-1),1:5)),2);
83:             threecol((i-2),1:5)= mod(xor(threecol_mid(1,1:5),H((i-2),1:5)),2);
84:             fourcol_mid1(1,1:5)= mod(xor(D(u1,1:5),H((i-1),1:5)),2);
```

```
fourcol_mid2(1,1:5)= mod(xor(fourcol_mid1(1,1:5),H((i-2),1:5)),2);
fourcol((i-3),1:5)= mod(xor(fourcol_mid2(1,1:5),H((i-3),1:5)),2);p=1;t=1;
    end if;u1=u1+1;
  end while;
  if ((u1≥31) or (p=1)) then u1=1;
  end if
  if (t=1) then i= i + 1;
  end if
end while;
```

The syndrome generated for different adjacent erroneous bits during decoding using $H_C(38,32)$ and $H_C(71,64)$ are shown in Table 5.1. When (38, 32) and (71,64) shortened block codes are used as component code in the product code for decoding on data sets having the dimension 32×32 and 64×64 they can correct clustered error of size 5×5 and 6×6 respectively. From Table 5.1 it clearly shows that syndrome vectors generated due to different error patterns are not overlapped which helps during decoding of different error patterns. Now we are going to

Table 5.1: Generated syndromes for different adjacent erroneous bits

Memory size	size of component codes	No. of Adjacent erroneous bits					
		1	2	3	4	5	6
16 × 16	(21,16)	16,8,4 2,1,13	3,6,12,9 24,29,18	7,14,28,21, 26,19,11,25	5,10,17,20, 30,15,27	N/A	N/A
32 × 32	(38,32)	1,2,4,8 16,32,35	3,6,12,24, 48,39,10,9 17,34	7,14,28,56 19,37,11,25 49,50,38	15,30,60,27 23,5,45,57, 51,54,46,29	31,62,21,13, 44,59,55	N/A
64 × 64	(71,64)	1,2,16,32 4,8,9,64	3,6,12,24, 48,96,73,11, 17,36,34	7,14,28,56, 112,105,75,10, 19,25,44,38, 35	15,30,60,120, 121,107,74,26, 27,29,46,39, 51	31,62,124,113, 123,106,90,18, 61,47,55,59	63,126,117,115, 122,82,22

calculate parity check matrix of the shortened linear block codes that can be used as component code of product code to correct almost clustered error. The parity check matrix $H_C(21,16)$ in equation 5.3 can also be used to detect adjacent errors and almost adjacent errors in 16 bit data and hence, it is not mentioned separately. Parity check matrix for detecting adjacent errors and almost adjacent errors for 32 bit and 64 bit data set are represented by $H_R(38,32)$ and $H_R(71,64)$ as shown in equation 5.6and 5.7 respectively.

$$\begin{bmatrix}
1&0&0&0&0&0&1&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0\\
0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0\\
0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1\\
0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0\\
0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0\\
0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{bmatrix} \tag{5.6}$$

Syndromes generated for different error patterns in 16 bit, 32 bit and 64 bit data is shown in Table 5.2. To represent different error patterns in Table 5.2 **'X'** represent erroneous bit and **'_'** represent the bits not affected by error. Different error patterns combining corrupted bits and non-corrupted bits form almost adjacent error patterns as shown in Table 5.2.

$$\begin{bmatrix}
1000000100000010000001000000100000010000001000000100000010000001000000100000001\\
0100000010000001000000100000010000001000000100000010000001000000100000001000000\\
0010000000010000001000000100000010000001000000100000010000001000000100000001000\\
0001000100100010010001001000100100010010001001000100100010010001001000100100001\\
0000100001000000100000010000001000000100000010000001000000100000010000001000000\\
0000010000001000000100000010000001000000100000010000001000000100000010000000100\\
0000001000000100000010000001000000100000010000001000000100000010000001000000010
\end{bmatrix} \tag{5.7}$$

Table 5.2: Generated syndromes for different error patterns

Memory size	size of component codes	No. of Adjacent erroneous bits								
		X	XX	XXX	X_X	XXXX	X_XX/ XX_X	X__X	XXXXX	X_X_X
16 × 16	(21,16)	16,8, 4,2, 1,13	3,6,9, 12,24, 29,18	7,14,28,21, 26,19,11,25	5,10,20, 15,17	N/A	N/A	N/A	N/A	N/A
32 × 32	(38,32)	1,2,4,8 16,32,35 37,41,49	3,6,12, 24,48	7,14,28,56 19,38,47,61 25,50	5,10,20 40,51	15,30,60 27,54	11,22, 44,59, 21,42, 55,13, 26,52	9,18,36 43,53	N/A	N/A
64 × 64	(71,64)	1,2,16,32 4,8,9,64	3,6,12, 24,48,96, 73,11,18, 36	7,14,28,56, 105,75,27, 112,26,28, 44,100,105	5,10,20, 40,80, 41,66, 25,68	15,30,60, 120,121, 107,91, 19,108, 109	N/A	N/A	31,62, 124,113, 123,83, 23,101, 111	21,42 84,33, 82,57, 74,29, 70

When (21,16), (38,32) and (71,64) shortened block codes whose syndromes are shown in Table 5.2 due to different error patterns are used as component codes for the product code they can correct cluster of erroneous bits having dimension of 3×3, 4×4 and 5×5. Still now we have discussed generation of syndrome vectors and parity check matrix for MPP based MLC flash devices. Now the generation

of parity check matrix of the shortened block codes for error detection in adjacent cells in SPP based MLC flash devices are discussed.

In SPP based MLC adjacent MBUs either may corrupt two adjacent cells or a single cell as we have discussed previously. Here adjacent data bits that are arranged along the row in a page may come from single cell or two adjacent cells but data bits along the column always come from different cells as discussed in Figure 5.5(a). Hence, along the row in a data matrix there is a concept of cell error correction and for this reason we have derived new parity check matrix represented by equation 5.8- 5.9. Parity check matrix represented by equation 5.3- 5.7 can be used to detect error along the column. In SPP based MLC we have considered cell corruption due to only adjacent MBUs. Parity check matrix represented by $H_{CC}(21,16)$ in equation 5.8 can detect two adjacent cells with 100% efficiency and three adjacent cells with~50% efficiency.

$$\begin{bmatrix} 1&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&1&0&0&1&0&0&0&1&0&0&0&1&0&0&0 \\ 0&0&1&0&0&1&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0 \\ 0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0 \\ 0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1 \end{bmatrix} \tag{5.8}$$

$$\begin{bmatrix} 100000110000100100001001000010010000010 \\ 010000100000010000000100000001000000001 \\ 001000001001000010010000100100001001000 \\ 000100000100000001000000010000000100000 \\ 000010000010000000100000001000000010000 \\ 000001000000001000000010000000100000000 \end{bmatrix} \tag{5.9}$$

Similarly parity check matrix represented by $H_{CC}(38,32)$ in equation 5.9 can detect three adjacent cells with 100% efficiency.

Table 5.3: Generated syndromes for different error patterns

Memory size	size of component codes	No. of Adjacent erroneous bits		
		1	2	3
16×16	(21,16)	1,2,3,4,5,8, 9,12,16,18,21	6,24,7,10,17, 14,28,29,23,19, 26,11,25,15, 22,27	30,31
32×32	(38,32)	1,2,3,4,8,12, 16,20,32,33,48	5,6,7,24,35, 14,28,56,51,34, 13,21,35,37, 15,50,60,23,45	30,59,38,29,39 31,62,63,58,54, 46,25,55,47,61, 26,27,57

5.4 Encoding/Decoding using LSBCPC and its hardware implementation

The proposed product code is systematic *i.e* during encoding process generated redundant bits are simply appended with the original data bits. The encoding process involves multiplication of the data with the generator matrix to generate redundant bits that will be stored into the same memory unit in different physical locations. Steps involved in the encoding process are:

- **Step1:** Read data from proper memory location.
- **Step2:** Choose proper parity check matrix based on the three parameters: dimension of data set, whether used for clustered or almost clustered error correction, data set belongs to SPP or MPP based MLC.
- **Step3:** Parity check matrix can be represented as $[I_{n-k}, P^T]$ and from this representation calculate generator matrix corresponding to the selected component codes that can be represented as $[P, I_k]$.
- **Step4:** Multiply the data set with the generator matrix.
- **Step5:** Transpose the data set and multiply the transposed data set with the generator matrix.
- **Step6:** Write back the data into the memory location from where it was read out.
- **Step7:** Store the generated redundant data into the same page of MLC flash from where the data set had been read out.
- **Step8:** Go to step 1 and continue the process until the encoding of the whole page is completed.

Decoding process involves generation of syndrome vectors, locating of the error in the data set and then correcting the error. Detailed hardware implementation of decoding circuit of the proposed algorithm is illustrated in Figure 7.6. Steps involved in the decoding process are:

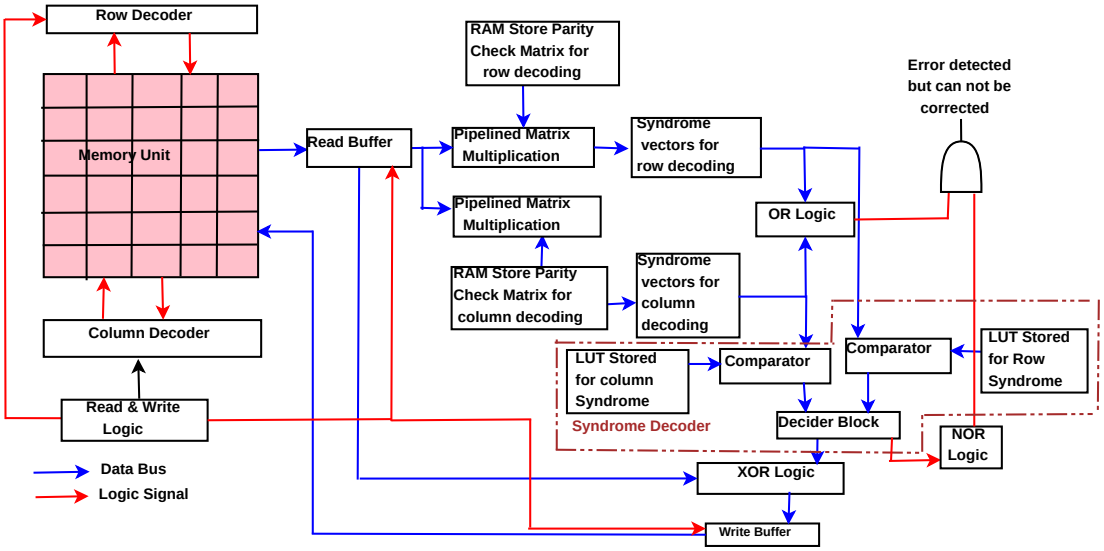

Figure 5.7: Hardware Implementation of proposed LSBCPC

- Read/Write Logic block generates the address of the data to be read and send to Row and column decoder;
- Data and corresponding redundant bits are read from Memory unit and are stored in the Read Buffer;
- Generate the syndromes for row and column decoding using the parity check matrix stored in the RAM with the help of XOR network.
- Zero syndrome vectors indicate no error in the data set and nonzero syndrome vectors indicate presence of either single or multi-bit errors.
- Syndromes due to different error patterns are stored into LUT and are compared with the generated syndrome vectors as shown in Figure 7.6.
- Based on the result of comparison error patterns along row and columns are decided and from that decider block detects exact location of erroneous bit and correct it using the XOR logic.
- Corrected data will be sent to the write buffer and from there data along with redundant information will write back to the memory unit.

The generated syndrome vectors for both row and column decoding are fed to a OR logic that gives high output when non-zero syndromes are generated. If the syndrome decoder block can successfully detect and correct the erroneous bits it gives high signal corresponding to different syndromes to NOR logic block otherwise it gives '0' output. Output of NOR logic block is AND-ed with output of OR logic block and its output becomes zero when errors are successfully corrected otherwise output of AND gate will be high to inform the user that error is detected but can not be corrected.

Encoding and decoding process involves matrix multiplication i.e multiplication of data set with the generator matrix during encoding process and multiplication with parity check matrix during decoding process. In order to speed up the encoding and decoding process row and column operations are done in parallel. At the same time we have introduced pipeline architecture in matrix multiplication as described in algorithm 5. Data with redundant bits are multiplied with parity check matrix and stored in P_1 and P_2. Here we have applied three stage pipelined architecture to speed up multiplication process. The timing diagram of the pipelined architecture of the proposed matrix multiplication is shown in Figure 5.8. Here states S1, S2 and S3 will fill up the pipeline while states S5, S6, S7 will flush the pipeline. In stage4 all the four states will remain active. X0,X1,X2 and X3 registers in algorithm 5 are used to keep track the indices of the data sets. Hence, initially there was no data for three clock cycle and then for every clock cycle data will come out serially. For the efficient hardware implementation of the encoder and decoder some optimization criteria can be applied during the derivation of parity check matrix. Here we have applied two optimization criteria as described in [147]:

- Choose parity check matrix with the smallest number of ones. This criteria helps to implement encoder and decoder circuit with minimum number of logic gates that consume the smaller area.
- Number of ones in different rows of parity check matrix decide the speed of decoding process. Hence, the row having maximum number of one is the deciding factor for latency of encoding and decoding process. With the decrease of number of ones in the heaviest row in a parity check matrix latency of encoding and decoding process reduces.

Algorithm 5 Pipelined Architecture for Row and Column Syndrome vector Generation

Require: A[R,R],H[h,R];
Ensure: RSmatrix[R,h], CSmatrix[R,h];

1: **Variable**:$i_0, j_0, k_0, k_2, i_3, j_3, k_3, u, v, w$,P1,P2,sum1,sum2, Done, Start;
2: Assign X0,X1,X2,X3="000000000000000";
3: i_0:= CONVINTEGER(X0(14 downto 10)), j_0:= CONVINTEGER(X0(9 downto 5));
4: k_0:= CONVINTEGER(X0(4 downto 0)), k_2:= CONVINTEGER(X2(4 downto 0));
5: i_3:= CONVINTEGER(X3(14 downto 10)), j_3:= CONVINTEGER(X3(9 downto 5));
6: k_3:= CONVINTEGER(X3(4 downto 0));
7: Case State is:
8: when S_0=>
9: Done <= '0'; X0 <= "000000000000000";
10: **if** (Start = '1') **then**
11: State := S1;
12: **else**
13: State := S0;
14: **end if**;
15: when S_1=>
16: u=A(i_0, k_0), v =B(k_0, j_0), w=A(k_0, j_0), X1= X0, X0 = X0 + 1, State := S2;
17: when S_2=>
18: u=A(i_0, k_0), v =B(k_0, j_0), w=A(k_0, j_0);
19: P1= u × v, P2= w × v, X2 = X1, X1= X0, X0= X0 + 1, State := S3;
20: when S_3=>
21: u=A(i_0, k_0), v =B(k_0, j_0), w=A(k_0, j_0);
22: P1= u × v, P2= w × v;
23: **if** ($k_2 = 0$) **then**
24: sum1 = P1, sum2 = P2;
25: **else**
26: sum1 = sum1 + P1, sum2 = sum2 + P2;
27: **end if**;
28: X3= X2, X2 = X1, X1= X0, X0= X0 + 1, State:= S4;
29: when S_4=>
30: u=A(i_0, k_0), v =B(k_0, j_0), w=A(k_0, j_0);
31: P1= u × v, P2= w × v;
32: **if** ($k_2 = 0$) **then**
33: sum1 = P1, sum2 = P2;
34: **else**
35: sum1 = sum1 + P1, sum2 = sum2 + P2;
36: **end if**;

```
37: if (k3 = 20) then
38:     RSmatrix(i3, j3)=sum1,CSmatrix(i3, j3)=sum2;
39: end if;
40: X3=X2,X2=X1,X1=X0;
41: if (X0=9260) then
42:     State:=S5;
43: else
44:     State:=S4, X0=X0+1;
45: end if
46: when S5=>
47: P1= u × v, P2= w × v;
48: if (k2 = 0) then
49:     sum1 = P1, sum2 = P2;
50: else
51:     sum1 = sum1 + P1, sum2 = sum2 + P2;
52: end if;
53: if (k3 = 20) then
54:     RSmatrix(i3, j3)=sum1,CSmatrix(i3, j3)=sum2;
55: end if;
56: X3=X2,X2=X1,State:=S6;
57: when S6=>
58: if (k2 = 0) then
59:     sum1 = P1, sum2 = P2;
60: else
61:     sum1 = sum1 + P1, sum2 = sum2 + P2;
62: end if;
63: if (k3 = 20) then
64:     RSmatrix(i3, j3)=sum1,CSmatrix(i3, j3)=sum2;
65: end if;
66: X3=X2, State:=S7;
67: when S7=>
68: if (k3 = 20) then
69:     RSmatrix(i3, j3)=sum1,CSmatrix(i3, j3)=sum2;
70: end if;
71: State:=S8,X0="000000000000000",Done=1;
72: when S8=>
73: if X0=48 then
74:     X0="000000000000000",State:=S0;
75: else
76:     X0= X0 + 1, State:= S8;
77: end if;
```

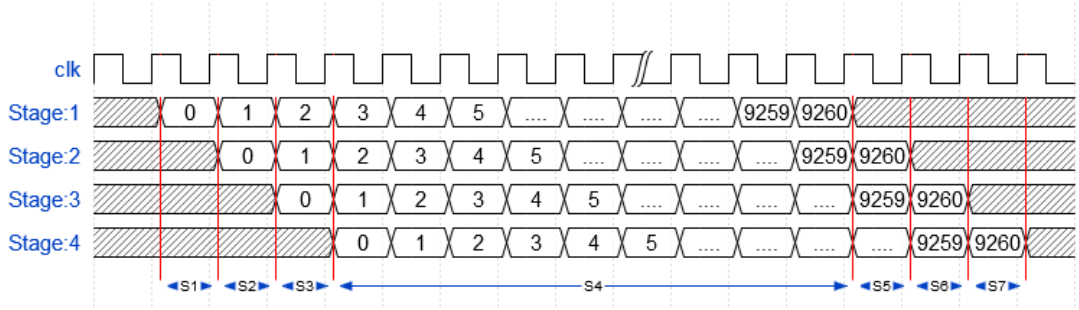

Figure 5.8: Timing Diagram for pipelined architecture of matrix multiplication during syndrome generation

The example shown below is very helpful to demonstrate how the optimization criteria helps to choose proper parity check matrix. We have already shown in equation 5.6 that $H_R(38,32)$ is the parity check matrix used for decoding of (38,32) shortened block code. There is another parity check matrix for(38,32) shortened block code that can be derived from the lexicographic matrix of (63,57) linear block code. It is represented by $H_{RL}(38,32)$ in equation 5.10.

$$\begin{bmatrix} 1&0&0&0&0&0&1&1&1&1&1&0&0&0&0&0&1&1&1&1&1&0&0&0&0&0&1&1&1&1&1&0&0&0&0&0&1&1 \\ 0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0 \\ 0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1 \\ 0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&1&1&1&0&0&0&0&0&1&1&1&1&0&0&0&0&0&1&1&1&1&0&0&0&0&0&1&1&1&1&0&1 \end{bmatrix} \tag{5.10}$$

The total number of ones in $H_{RL}(38,32)$ and $H_R(38,32)$ are 66 and 40 respectively. At the same time number of ones in heaviest row of $H_{RL}(38,32)$ and $H_R(38,32)$ are 18 and 8 respectively. Hence we have selected $H_R(38,32)$ as parity check matrix for decoding of 38 bit encoded data instead of $H_{RL}(38,32)$.

5.5 Results and Performance Analysis

Proposed fault correcting models are implemented on hardware using Xilinx Vivado 2015.4 platform and tested using MATLAB simulations. During the simulation faults are injected in a clustered format (i.e MBUs are created in adjacent memory locations) to test different fault correcting models. We have considered both clustered and hybrid model for MPP based flash memory and only clus-

tered model for SPP based flash memory. Figure 5.9 (a) shows maximum cluster size that can be corrected in MPP based flash memory. As hybrid models can

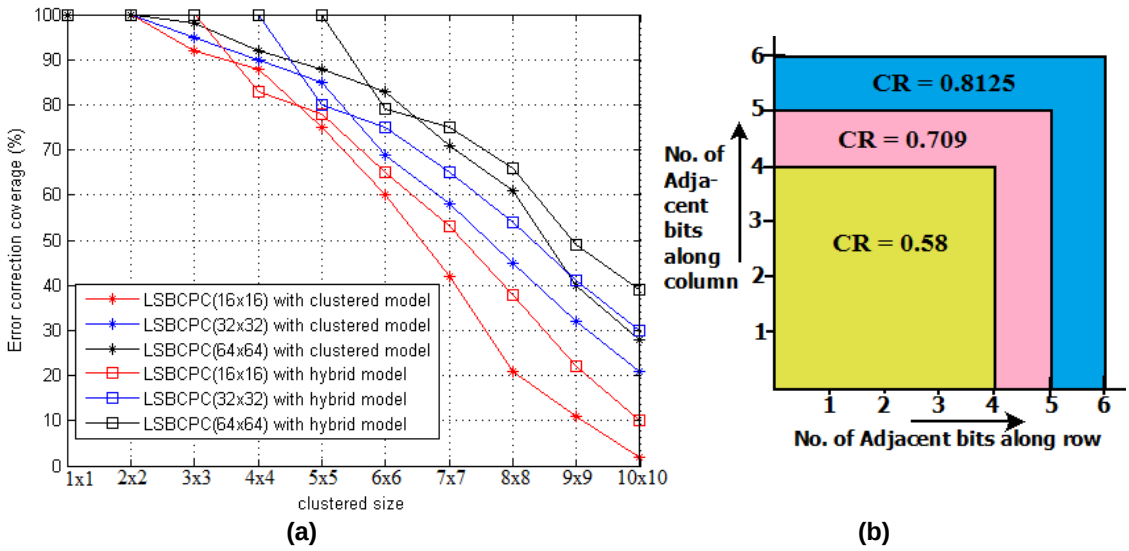

Figure 5.9: Error correction coverage of LSBCPC of different sizes (a) for both adjacent as well as nonadjacent MBUs (b) only for adjacent MBU

correct both adjacent as well as nonadjacent MBUs it shows better performance compared to clustered models with the increase of cluster size for different sizes of LSBCPC. When the cluster size is small clustered model sometimes shows better performance compared to hybrid model. As for example, when the cluster size is 4×4 clustered model of LSBCPC having dimension of 16×16 shows better performance compared to hybrid model having the same dimension. This is due to the fact that component code in the clustered model generate distinct syndromes for upto four adjacent erroneous bits whereas component code for hybrid model generate distinct syndromes only for cluster size 3×3. (21,16) linear block code can generate distinct syndrome vectors only for four adjacent erroneous bits but it can still detect any number of adjacent number of erroneous bits without giving distinct syndrome vectors. Hence, with the increase of cluster size error correction coverage for 16×16 LSBCPC deteriorates. Hybrid model of LSBCPC can correct a cluster of size 3×3 with 100% efficiency and after that efficiency of LSBCPC gradually decreases as depicted in Figure 5.9(a). The same argument also holds for LSBCPC having sizes 32×32 and 64×64. When only clustered or adjacent MBUs are present in the memory element as shown in Figure 5.6(a)

and (b) maximum number of adjacent erroneous bits that can be corrected using LSBCPC along the row and column of the cluster is given in Figure 5.9(b). With the increase of the size of product code, code rate (CR) increases but at the same time multiple iterations may be required to correct the error which in turns increase latency of error correction. CR is the ratio of data bits to the data bits plus redundant bits. For comparison of error correction coverage of different models as shown in Figure 5.9 maximum number of iteration is fixed to two.

Figure 5.10 gives error correction coverage of LSBCPC with different sizes for SPP based two level flash memory. Two adjacent bits along the row in a page matrix may be from a single cell or from two adjacent cells but two adjacent bits along the column are always from two different cells. Hence, along the column we can use the component code in the same way as in clustered model as discussed previously but along the row parity check matrix of component codes are modified as illustrated in equation 5.8 and 5.9 to correct multiple cells. Figure 5.10 shows the number of adjacent cells that can be corrected by LSBCPC having different sizes in a SPP based two level flash memory. Table 5.4 illustrates comparison

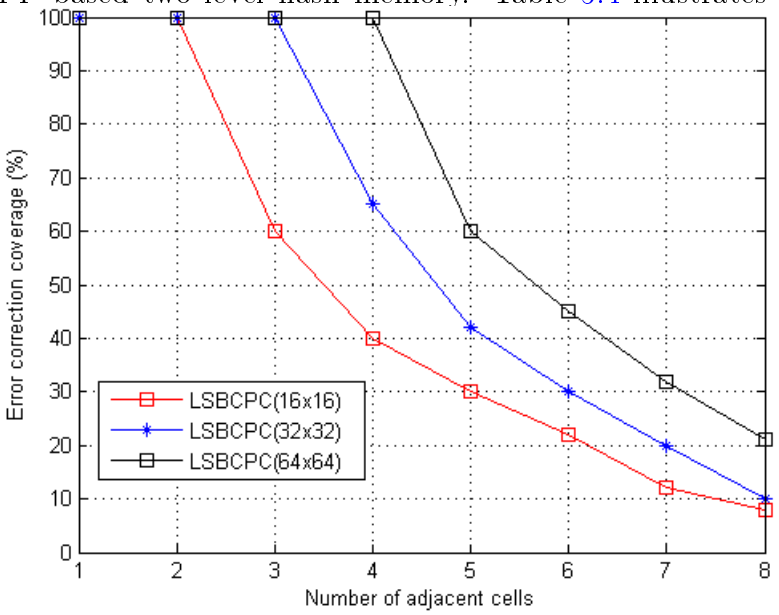

Figure 5.10: Adjacent cell correction coverage for single page programming based two level flash memory using LSBCPC having different sizes.

between the proposed LSBCPC having different sizes and product code developed using component codes proposed by authors in [148] in terms of error correction

coverage, number of redundant bits, power, delay and area of decoder circuit. In cluster error correction methodology error detection capability of component codes are more important compared to error correction. Though authors in [148], did not propose any specific product code that we can use directly, the proposed linear codes can be used to develop a product code. Hence, out of the different linear codes described in [148] we have chosen (23,16), (39,32) and (72,64) to form product codes whose performances are compared with the LSBCPC having same dimension. (23,16) linear block code described in [148] can correct single

Table 5.4: Comparison of the proposed codes with other codes

Window size	component codes	Error detection capability	Error correction capability	Redundant data	Total no. of ones in Parity Check Matrix	Maximum no. of ones in a row	Area (μm^2)	Decoder Delay (ps)	Power (μW)
16x16	(23,16) proposed in [148]	Five adjacent error and any two bit error with sharing syndromes	Single and double adjacent error	7	50	9	783	337	264
	(21,16) proposed in equation 5.3	Four adjacent error without sharing syndromes	No	5	23	6	693	255	210
	(21,16) proposed in equation 5.3	Three adjacent error and almost two errors with distinct syndromes	No	5	23	6	693	260	210
32x32	(39,32) proposed in [148]	Three adjacent error and any two bit error with sharing syndromes	Single and double adjacent error	7	105	18	926	460	403
	(38,32) proposed in equation 5.4	Five adjacent error without sharing syndromes	No	6	40	8	830	340	356
	(38,32) proposed in equation 5.6	Four adjacent error and almost three and two bit error without sharing syndromes	No	6	40	8	850	390	367
64x64	(72,64) proposed in [148]	Three adjacent error and any two bit error with sharing syndromes	Single and double adjacent error	8	189	30	1623	530	722
	(71,64) proposed in equation 5.5	Six adjacent error without sharing syndromes	No	7	72	13	1190	410	585
	(71,64) proposed in equation 5.7	Five adjacent error and almost two, three and four bit error without sharing syndromes	No	7	81	20	1298	450	634

and double adjacent bit errors and at the same time it can detect upto five adjacent errors. During error detection (23,16) linear block code produces non distinguishable syndrome vectors (*i.e* syndrome vectors generated by three, four and five adjacent erroneous bits may overlap) for adjacent erroneous bits. Hence, product code developed using this linear block code can correct cluster of size 5×5 but it may increase error correction time. Though the proposed LSBCPC of dimension 16×16 can correct cluster error having maximum size of 4×4 using clustered model and 3×3 using hybrid model it requires less redundant bits, error correction time, total number of ones in parity check matrix and number of ones in the heaviest row of the parity check matrix. This helps to implement decoding circuit of LSBCPC with less area, delay and power compared to product code developed using (23,16) linear block code as illustrated in Table 5.4. When the size of the product code becomes 32×32 and 64×64 LSBCPC outperforms the product code formed using the (39,32) and (72,64) linear block code not only in terms of decoder delay, power consumption, area and overhead due to redundant bits but also maximum error cluster that can be corrected.

Mean error to failure (METF) [93] is the number of errors that a memory element can withstand in the presence of different error correcting codes before it starts to generate wrong data during the read operation. METF of LSBCPC of different sizes for both clustered and hybrid model and product code formed using the linear codes described in [148] are shown in Figure 5.11. Hybrid model outperforms clustered model of LSBCPC in terms of METF for all the memory sizes. When the memory sizes are small error clusters are nearer to each other and multiple clusters may be found within the same data set. Hence, for small memory size LSBCPC with size 16×16 gives better result compared to LSBCPC with size 32×32 which in turn gives better performance compared to LSBCPC with size 64×64. With the increase of memory size distance between clusters increase and situation is totally reversed and LSBCPC with size 64×64 gives best result as illustrated in 5.11. Product code formed using (23,16) linear block code gives better performance compared to LSBCPC having same size but hybrid model of LSBCPC with sizes 32×32 and 64×64 outperform product codes of same dimension formed using (39,32) and (72,64) linear block codes respectively in terms of METF. In order to test the reliability of memory elements we have

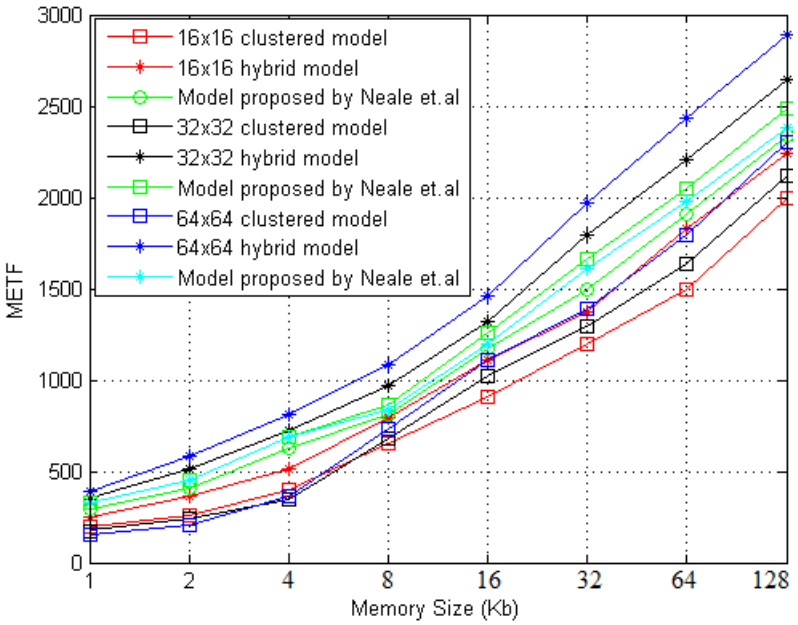

Figure 5.11: Variation of METF with different memory sizes

used another parameter known as mean time to failure (MTTF) which is defined as the time for continuous error free operation of a memory unit and can be defined by equation 5.11.

$$MTTF := \frac{METF}{MemorySize \times MBURate \times Iteration} \tag{5.11}$$

$$MBURate := ParticleFlux \times DeviceCrossSection \tag{5.12}$$

$$DeviceCrossSection := Numberofbitsinthememory \times bitscrosssection \tag{5.13}$$

Here we have modified the MTTF parameter used by the authors in [93] by adding iteration parameter in the denominator. MTTF has been calculated for various memory sizes and is shown in Table 5.5. During error correction using multiple iterations memory will not be available for data storage which reduces the MTTF value. We have taken the value of bit cross section from [149] and used the value of particle flux as illustrated in [150].The particle flux of intensity $10^5 s^{-1} cm^{-2}$ is considered, iteration is fixed to three and value of bit cross section is taken as $2.74 \times 10^{-14} cm^2$. Values of METF are taken from Figure 5.11.

Table 5.5: MTTF in Days for different memory sizes

MBU Rate	Protection Techniques	Memory size			
		2 Kb	8Kb	32 Kb	128Kb
2.74×10^{-9}	16×16 clustered model	178	111	51	21
	16×16 hybrid model	268	136	58	24
	Model proposed by Neale et.al for 16x16 data	278	139	64	25
	32×32 clustered model	165	116	55	23
	32×32 hybrid model	350	166	77	28
	Model proposed by Neale et.al for 32x32 data	309	148	71	27
	64×64 clustered model	144	126	59	25
	64×64 hybrid model	398	186	84	31
	Model proposed by Neale et.al for 64x64 data	309	144	69	25

5.5.1 Cost Analysis

Here, we have defined a new metric called error correction coverage with enhanced performance (ECCEP) to compare the overall performance of the LSBCPC of different sizes with the product code formed using the component code described in [148] in terms of code rate, implementation overhead, error correction capability and error correction time or latency. The new metric ECCEP can be defined as:

$$ECCEP := \frac{Correction\ Coverage \times speedup \times coderate}{Cost} \tag{5.14}$$

Where the cost can be calculated as:

$$\begin{gathered} Cost = Power \times Area \times Decoding\ latency \\ Cost = Power \times Total\ number\ of\ one\ in\ parity\ check\ matrix \times \\ Number\ of\ one\ in\ heaviest\ row\ of\ parity\ check\ matrix \end{gathered} \tag{5.15}$$

Correction coverage is the maximum cluster size that can be corrected. Speedup parameter indicates stages of the pipeline that is used in the hardware implementation of matrix multiplication for syndrome calculation during decoding of the product code and here it is three. Decoding cost of an error correcting model is calculated using area, delay and power consumption of the decoding circuit. Power has been measured using the Xilinx Xpower tool. With the increase of the number of one in parity check matrix area of decoding circuit increases as

mentioned by authors in [147]. Hence, in the case cost estimation area of the decoding circuit is replaced by the total number of ones in the parity check matrix. Decoding latency is the time required by the decoding circuit to detect and correct the errors that can be defined by equation 5.16. Here we have fixed the number of iterations to two and time consumed by one iteration can be replaced by the number of ones present in the heaviest row of the parity check matrix as mentioned by authors in [147].

$$Decoding\ latency = Number\ of\ iteration \times Time\ consumed\ by\ one\ iteration \tag{5.16}$$

ECCEP for LSBCPC and product code developed using component code described by authors in [148] are illustrated in Figure 5.12. It is clearly seen that LSBCPC gives better result compared to the product code formed using the block code described in [148] for all the sizes. As clustered model can correct larger cluster size compared to hybrid model LSBCPC with clustered model performs better in terms of ECCEP compared to LSBCPC with hybrid model.

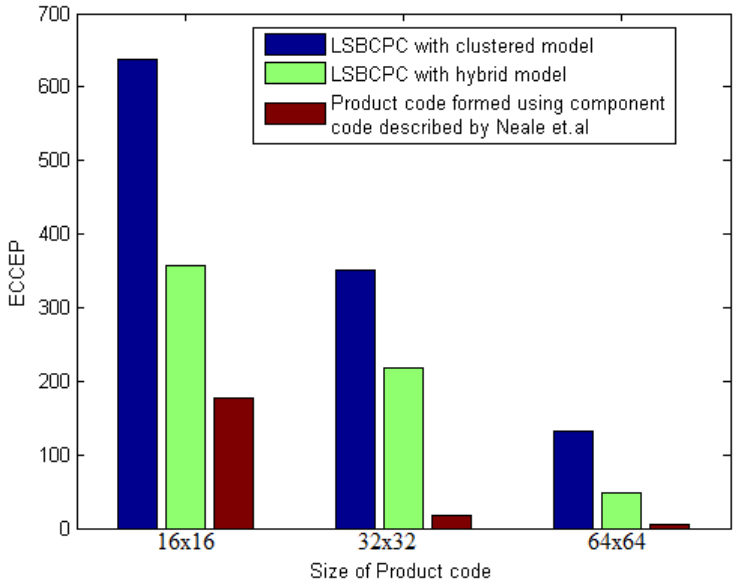

Figure 5.12: Variation of ECCEP for different size of LSBCPC and product code formed using component code described in [148]

Authors in [93] defined another new metric known as cost per chip, which is defined as the ratio of error free memory chip before fault injection to the number of working memory chips after error correction. We have used the same metric for

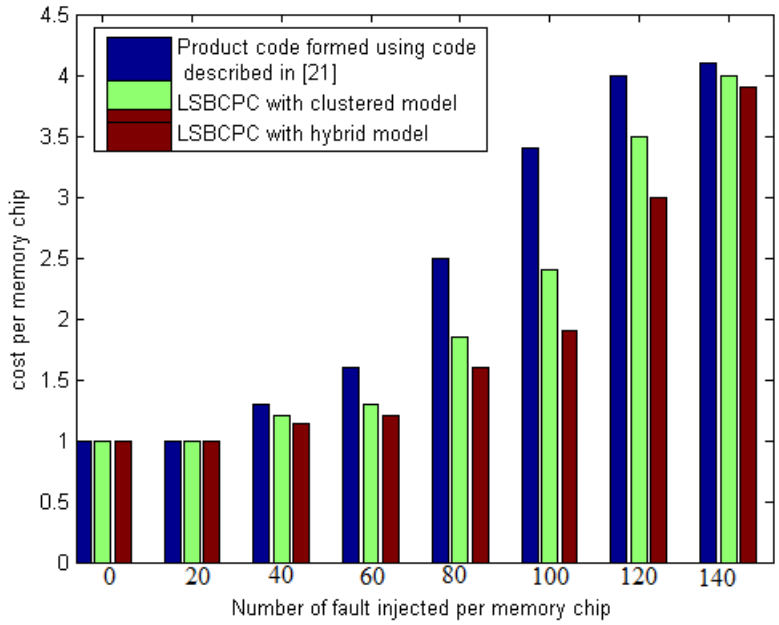

Figure 5.13: Variation of Cost per chip with different number of errors injected per memory chip for LSBCPC and product code developed using block code described in [148] having size 32×32

comparing the performances of different error correcting models but not including any redundant rows and columns, unlike to [93]. We have kept the number of memory chips fixed to 1000 and size of each memory unit is taken as 1024×32 bits. Variation of cost per chip is illustrated in Figure 5.13 for different number of errors injected into the memory chip for different error correcting models keeping the size of the product code fixed to 32×32. When there is no fault or only twenty faults per chip on average, then all chips can be corrected by LSBCPC and product code formed by the component code described in [148] and hence, the cost per chip is one. With the increase in the number of faults per chip cost per chip gradually increases as some memory chips remains faulty after error correction. The largest cluster size that can be corrected using LSBCPC with hybrid model is less than the clustered model but the number of error patterns that can be corrected by hybrid model are higher than clustered model. Hence, error correction coverage of hybrid model is always higher than clustered model against nonadjacent MBUs whose occurrence probability is more than adjacent MBUs. As (39,32) linear

block code proposed by authors in [148] produces indistinguishable syndromes for more than two erroneous bits, error correction coverage of product code developed using this block codes are always less than LSBCPC. With the increase of faults per memory chip LSBCPC with hybrid model shows best performance as shown in Figure 5.13. When the number of injected errors are high i.e., 140 per chip then the value of cost per chip is almost same for all the error correcting models.

5.6 Conclusion

This chapter illustrates development of a low complexity latency optimized product code using simple linear block code as component codes and its performance against clustered error mitigation in the MLC flash memory. During the development of product code we have applied a method on the component codes called shortening of block code to detect adjacent MBUs generated by radiation. The proposed approach has been tested for different memory sizes, different particle flux density and thus proving its effectiveness to detect and correct clustered MBUs. The proposed product code consumes less area, power and has fast decoding circuit compared to the product code developed by the component block codes described by the authors in [148]. At the same time parallel processing and pipelining during syndrome computation reduces error correction time. In the next chapter we are going to describe error mitigation techniques for configuration memory of FPGA devices.

Chapter 6

Soft error mitigation in Configuration memory of FPGA using HPC with selective bit placement and Frame Interleaving

Radiation induced adjacent MBUs are very prominent in the configuration memory of static random access memory (SRAM) based FPGA devices in the age of ultra-large scale VLSI (ULSI) technology. When a charge particle having high energy and low momentum hits the FPGA devices it damages a group of adjacent logic cells and switches and forms the clustered error. In some critical applications like space exploration, satellite, radiotherapy where intensity of radiation is high, there is a high probability that the configuration memory of FPGA is being affected by such type of clustered errors. Commonly used error mitigation techniques in FPGA like TMR, CED and different EDACs either have large overhead, complex decoding circuit or are not very efficient to correct a large number of adjacent erroneous bits. Configuration data of FPGA devices are composed of a number of configuration frames (CF) and there is a high probability that multiple physically adjacent frames may be affected by clustered error. Hence, interleaving between CFs are quite advantageous for mitigation of clustered error in the configuration memory of FPGA. In this chapter, a simple and efficient error cor-

recting model combining Hamming product code (HPC) with frame interleaving and selective bit placement (HPCFISBP) is proposed to correct adjacent MBUs in the configuration memory of FPGA without any modification in traditional FPGA architecture. The efficiency of the proposed method has been compared with HPC in terms of error correction coverage, error correction time, overhead due to redundant bits and residual error.

6.1 Introduction

Reconfigurability, fast time to market, high logic density makes the FPGA devices very lucrative to the embedded system designer though FPGAs are not very robust against radiation. Soft errors, also known as transient errors are the temporary malfunction that occur in solid state devices due to radiation. They are not reproducible and sometimes lead to SEUs or MBUs. These SEUs and MBUs prevent normal functionalities of FPGA devices. Different FPGA vendors like Xilinx, Altera, Microsemi developed radiation tolerant FPGA devices known as space graded FPGA but they have less logic cells and are expensive. Hence, instead of using space graded FPGAs we can consider commercial FPGAs, which can be used with proper error mitigation techniques.

Configuration memory constitutes the major memory share in an FPGA. Hence, the configuration memory is most likely to be affected by radiation more as compared to data memory, which necessitates the role of configuration memory protection. It is always expected that data in configuration memory should remain unchanged after a device is configured for a particular design while the input data in data memory can change any time with the clock. In order to meet the demands of modern high performance computing for computation of different complex algorithms, now a days FPGA devices are fabricated using denser integration schemes like 14 nm technology [151]. With the increase of memory density chance of adjacent memory cells are being affected by alpha, beta or neutrons and formation of clustered error increase many times [97]. Commonly used error correcting techniques in FPGA like TMR,CED [152] are based on majority voting and consume huge hardware resources, power and increase the chance of silent data corruption (SDC) (bit flipped but effect is not reflected at the output of the

circuit [153]). Scrubbing with cyclic redundancy checking [82] and configuration readback [81] eliminate the probability of occurrence of SDC but these methods require continuous access of external radhard memory that increases the system cost and introduces delay.

Now a days, ECCs are used to mitigate the effect of MBUs arising from soft errors in the configuration memory of FPGA. Multi-bit error correcting codes like RS, BCH, Euclidean Geometry Low Density Parity Check (EG-LDPC) code [154] are required to correct large size clustered error. With the increase of error correction capability number of redundant bits will also increase. Product codes like HPC are quite efficient against MBUs as discussed by authors in [89] as it has simple decoding circuit and low overhead. The efficiency of HPC starts to deteriorate as the number of the adjacent erroneous bits increases. On the other hand we can use methods like selective bit placement, interleaving along with the ECC to enhance error correction capability without increasing number of redundant bits. Hamming code along with selective bit placement and shortening enhances the probability of detection and correction of adjacent erroneous bits as shown by the authors in [155] but the method is most suitable for small number of adjacent erroneous bits. In this chapter, we have proposed a new method which combines HPC along with frame interleaving and selective bit placement against adjacent MBUs keeping overhead due to redundant bits and decoding circuit complexity same as in HPC used in [89].

6.2 Proposed Hamming Product code with frame interleaving and selective bit placement

HPC is an efficient multi-bit ECC in terms of BER performance and hardware complexity. In HPC, data bits are arranged in a matrix and Hamming code is applied along both row and column of the matrix to generate the redundant bits. Hence, multiple erroneous bits in a single row or column in the matrix can easily be corrected. If multiple erroneous bits are present along both row and column in some cases they may be corrected using multiple iterations but it will increase error correction time or latency as illustrated in [89]. In some other situation

multiple erroneous bits can not be corrected at all using HPC. In [89] authors apply HPC on a set of data taking from a single CF so there is a fair chance that a clustered error is fully in a single data matrix. Then HPC can not correct this clustered errors. If the data matrix is formed taking data from multiple CFs, erroneous bits of the cluster disperse into multiple data matrix and HPC can easily correct the erroneous bits of that cluster. As the first step of the proposed method interleaving among CFs is applied as shown in Figure 6.1. Available CFs

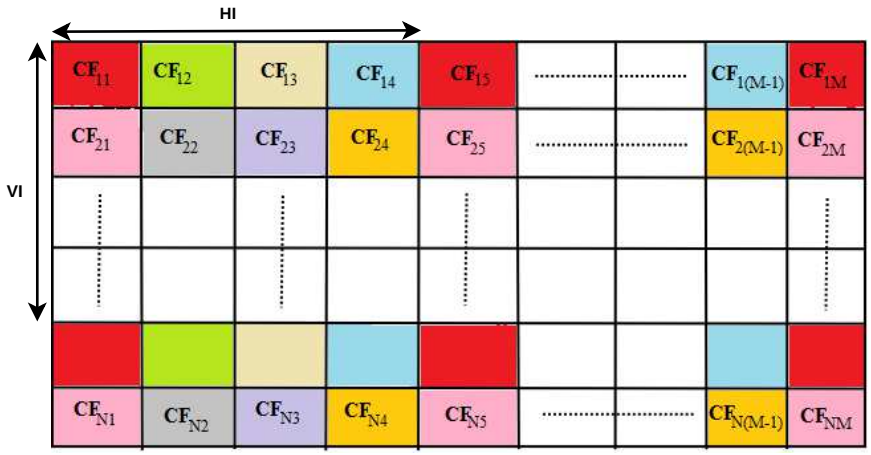

Figure 6.1: Frame Interleaving in the configuration memory of FPGA

are arranged as a matrix shown in Figure 6.1 and frames having the same color make one interleaving group. In Figure 6.1 horizontal interleaving depth (HI) and vertical interleaving depth (VI) are chosen as four but it can be changed to any other value. Next, a window is formed taking a number of CFs from a interleaved group as per the size of the HPC. In Figure 6.2 CFs in a interleaved group are shown where group is formed taking interleaving depth as two along both horizontal and vertical direction. CFs in the interleaved group is arranged in K number of horizontal groups and L number of vertical groups. In Figure 6.2 (7,4) Hamming code is used as component codes for HPC to illustrate window formation so a window is formed taking sixteen CFs indicated by blue line. In the next step, data bits having the same position in the CFs within a window are read out and a data matrix is formed on which HPC is applied. Number of data matrix that will be formed from a window depends on the number of data bits in a CF. As for example in Figure 6.2 one data matrix is formed taking bits marked

with "*" and similarly another data matrix is formed with bits marked with "$" from the CFs in the window. The bits marked with red color in Figure 6.2 are corrupted by charge particles. The bit selection from the configuration frames

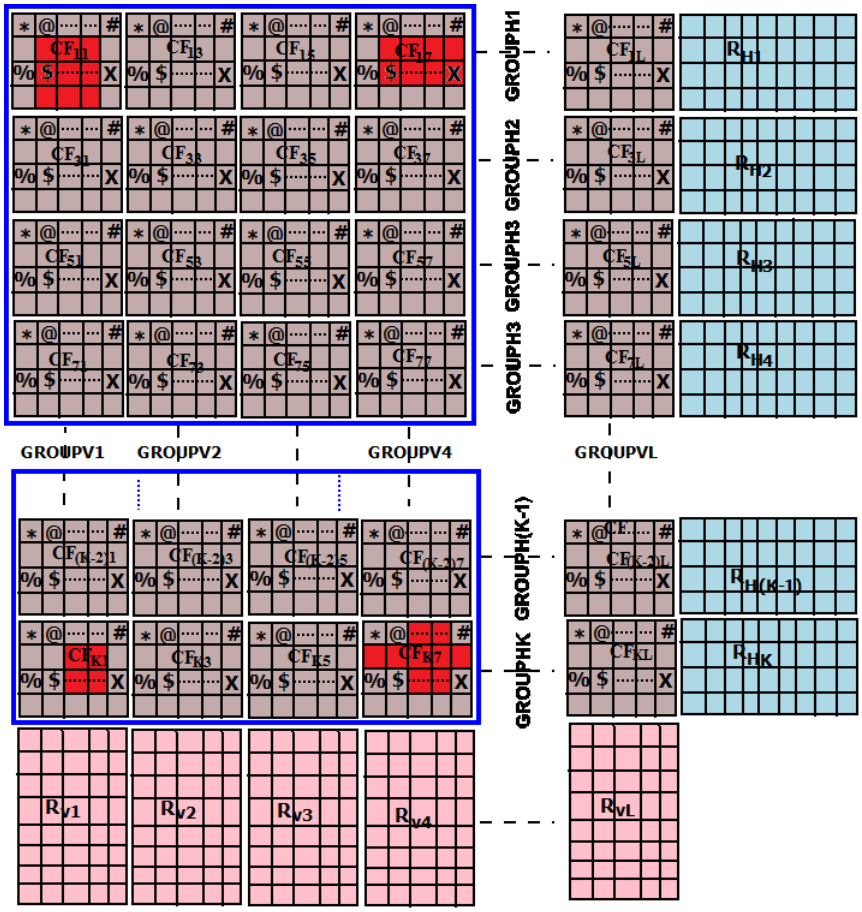

Figure 6.2: Configuration Frames of an interleaving group arranged into multiple horizontal and vertical groups

within a interleaved group is done using a strategy known selective bit placement. Next, we describe the selective bit placement strategy briefly.

6.2.1 Selective bit placement strategy

Encoding and decoding of the data matrix using HPC can be done by multiplying the data matrix with generator and parity check matrix of Hamming code respectively. During decoding using Hamming code syndrome vectors are generated and

if the generated syndrome vector is null or contains all zero elements, then it can be concluded that readback data is not erroneous. On the other hand nonzero values of the syndrome give the positions of single bit error in the received word. Here we have used Lexicographic matrix (similar to the parity check matrix used by the authors in [87]), in which i^{th} column contains binary value of i as shown by the matrix in equation 6.1.

$$H = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \tag{6.1}$$

Here we have described the selective bit placement using (12,8) Hamming code so size of the parity check matrix is 4×12. A (12,8) Hamming coded data having single bit error, when multiplied by the lexicographic matrix generate a four bit syndrome vector, that gives the position of the error in the received data. If more than one bit are erroneous, generated nonzero syndrome vectors may mistakenly flip a bit where the error does not occur at all. Maximum fifteen different syndrome vectors (without considering null vector) can be generated using four bits. Here, data width is twelve bits after encoding by Hamming code, so out of fifteen syndrome vectors only twelve syndromes are sufficient to detect and correct single bit error. Authors in [87] showed that there are some two bits errors which generate syndrome vectors whose values are greater than twelve but less than fifteen. Hence, arranging the bits according to these special combinations in a Hamming coded data, errors in two adjacent bits can be detected successfully. If a simple parity bit is added to the (12,8) Hamming code then the generated syndrome vector having value greater than twelve but less than fifteen can detect maximum three adjacent errors. Table 6.1 shows how the bits will be rearranged to detect two and three adjacent bits in (12,8) and (13,8) hamming coded data respectively. Authors in [87] calculated that out of eleven two adjacent bit errors in (12,8) Hamming coded data and three adjacent bit errors in (13,8) Hamming coded data, nine can be detected using selective bit placement strategy. In [87], authors listed such special combinations not only for (12,8) and (13,8) Hamming code but also for Hamming code having different

Table 6.1: Bit placement strategy in an one dimensional memory array

Bit placement to detect double adjacent error using (12,8) hamming code		Bit placement to detect triple adjacent error using (13,8) hamming code	
bit position before placement	bit position after placement	bit position before placement	bit position after placement
1	1	1	6
2	12	2	8
3	2	3	1
4	3	4	7
5	6	5	11
6	8	6	3
7	7	7	5
8	9	8	9
9	4	9	2
10	10	10	4
11	5	11	p
12	11	12	10
-	-	p	12

values of n and m like (21,16), (22,16),(38,32) and (39,32) where m is length of data and n is the length of encoded data.

Here we have used the same concept during the bit selection from the configuration frames in a interleaved group to form the data matrix. Bit selection from the configuration frame can easily be explained with the help of an example. After hamming encoding data and parity bits are arranged as shown in Figure 6.3. In the same way data and parity bits will be arranged along row and column of

1	2	3	4	5	6	7	8	9	10	11	12
P_1	P_2	D_1	P_3	D_2	D_3	D_4	P_4	D_5	D_6	D_7	D_8

Figure 6.3: Arrangement of data and parity bit for (12,8) Hamming coded data

a 12×12 matrix after encoding by HPC (component codes are (12,8) Hamming code) on a data matrix of size 8×8 formed by taking bits from the configuration frames in an interleaved group as shown in Figure 6.4. As here we are describing the example taking (12,8) Hamming code as component code of HPC window size in the interleaved group will be 8×8. In Figure 6.4 D_{ij} indicates bit of configuration frame in the i^{th} horizontal group and j^{th} vertical group within a window, P indicates parity bit generated using Hamming code on data bits and CP indicate

parity bits generated using Hamming code on parity bits (P). Now the selective bit placement strategy described in Table 6.1 for data array can be applied on matrix shown in Figure 6.4 along both row and column. The matrix shown in Figure 6.4 is designated as A. Next, selective bit placement algorithm described in algorithm 6 is applied to matrix A along both row and column in parallel and arranged data and parity bits accordingly in a 12×12 matrix designated as A_{re}. In algorithm 6, R indicates the number of rows and columns in A and A_{re}, h is the number of rows in the lexicographic matrix. After applying selective bit

Algorithm 6 Selective bit placement strategy in 2D-HPC

Require: A[R,R],H[h,R];
Ensure: A_{re}[R,R];
1: **Variable**: E[R,R]=0,S1[h,R],S2[h,R],U1[1,R],U2[1,R];
2: Choose a memory element of size R×R where data is to be protected by HPC;
3: Randomly invert any two or three bits of E[R,R](Note: Generate different error patterns);
4: A[R,R]=A[R,R]⊕ E[R,R];
5: Perform the operation S1[h,R]=H[h,R]×A[R,R];
6: H_T= transpose(H);
7: Perform the operation S2[R,h]=A[R,R]×H_T[R,h];
8: Convert each column of S1 into a decimal number and store into U1(1,R) and each row of S2 into a decimal number and store them into U2(1,R)
9: i=1;j=1;code length=12;
10: **while** (i ⩽R && j⩽R) **do**
11: **if** U1(1,i)⩾ code length **then**
12: C1= Store the the combination of bits;
13: **end if**
14: **if** U2(1,j)⩾ code length **then**
15: C2= Store the the combination of bits;
16: **end if**
17: i=i+1,j=j+1;
18: **end while**
19: E[R,R]=0; Go to step:2 until all two and three bit error patterns are generated;
20: Rearrange the column and row elements manually based on the information in C1 and C2 and store the rearrange data into A_{re};

placement strategy described by algorithm 6 data bits are rearranged in matrix

A_{re} as shown in Figure 6.5.

CP_{11}	CP_{21}	CP_{31}	CP_{41}	CP_{51}	CP_{61}	CP_{71}	CP_{81}	CP_{91}	CP_{101}	CP_{111}	CP_{121}
CP_{12}	CP_{22}	CP_{32}	CP_{42}	CP_{52}	CP_{62}	CP_{72}	CP_{82}	CP_{92}	CP_{102}	CP_{112}	CP_{122}
P_{11}	P_{12}	D_{11}	P_{13}	D_{12}	D_{13}	D_{14}	P_{14}	D_{15}	D_{16}	D_{17}	D_{18}
CP_{13}	CP_{23}	CP_{33}	CP_{43}	CP_{53}	CP_{63}	CP_{73}	CP_{83}	CP_{93}	CP_{103}	CP_{113}	CP_{123}
P_{21}	P_{22}	D_{21}	P_{23}	D_{22}	D_{23}	D_{24}	P_{24}	D_{25}	D_{26}	D_{27}	D_{28}
P_{31}	P_{32}	D_{31}	P_{33}	D_{32}	D_{33}	D_{34}	P_{34}	D_{35}	D_{36}	D_{37}	D_{38}
P_{41}	P_{42}	D_{41}	P_{43}	D_{42}	D_{43}	D_{44}	P_{44}	D_{45}	D_{46}	D_{47}	D_{48}
CP_{14}	CP_{24}	CP_{34}	CP_{44}	CP_{54}	CP_{64}	CP_{74}	CP_{84}	CP_{94}	CP_{104}	CP_{114}	CP_{124}
P_{51}	P_{52}	D_{51}	P_{53}	D_{52}	D_{53}	D_{54}	P_{54}	D_{55}	D_{56}	D_{57}	D_{58}
P_{61}	P_{62}	D_{61}	P_{63}	D_{62}	D_{63}	D_{64}	P_{64}	D_{65}	D_{66}	D_{67}	D_{68}
P_{71}	P_{72}	D_{71}	P_{73}	D_{72}	D_{73}	D_{74}	P_{74}	D_{75}	D_{76}	D_{77}	D_{78}
P_{81}	P_{82}	D_{81}	P_{83}	D_{82}	D_{83}	D_{84}	P_{84}	D_{85}	D_{86}	D_{87}	D_{88}

Figure 6.4: Arrangement of data and parity bits in a data matrix of size 12×12 before selective bit placement

Now we are going to describe how this rearrangement helps to detect and correct different error patterns in data matrix. Algorithm 7 is used to detect and correct errors in corrupted data stored in A'_{re} and give corrected output A_c. Here, R_r is the matrix of size 12×4 and contains syndrome vectors generated due to the row elements in A'_{re}. Similarly, R_c is the matrix of size 12×4 and contain syndrome vectors generated due to the column elements in A'_{re}. C_1 and C_2 are number of nonzero syndrome vectors in R_r and R_c respectively. Y_k and X_k represent the nonzero syndrome vectors in R_r and R_c respectively where k can vary from 1 to 12. As the redundant bits in $1^{st}, 3^{rd}, 6^{th}$ and 9^{th} row in A'_{re} are protected by only Hamming code along the column not by Hamming code using row operation, syndrome vectors for these rows in R_r matrix can not be used for decoding purpose and we forcefully make them zero. Simple parity bit is added with HPC to enhance error correction capability in data matrix. Rearrangement of bits in the data matrix using selective bit placement strategy for HPC with parity bit is shown in Figure 6.6. Algorithm 7 can also be used for error detection and correction for data matrix shown in Figure 6.6. The only difference is that value of R becomes 13 and instead of two adjacent erroneous bits three adjacent erroneous bits will generate syndrome vectors having the value

Algorithm 7 Error Detection and correction in data matrix

Require: A'$_{re}$[R,R],H[h,R];
Ensure: A_c[R,R],Z_{out};
Variable:$R_r(12,4)$,$R_c(12,4)$,$C_1, C_2, Y_l, X_k, Im_{th}, N$;
i=1;Z_{out}=0;N=0;
R_r =A'$_{re}$[R,R]× transpose(H[h,R]);
R_c = transpose(A'$_{re}$[R,R])× transpose(H[h,R]);
C_1= Number of nonzero syndrome vectors in R_r;
C_2= Number of nonzero syndrome vectors in R_c;
while $(((C_1 \neq 0) or (C_2 \neq 0)) \&\& (i \leqslant Im_{th}))$ **do**
    if $((C_1 = 0) \&\& (C_2 \geq 1))$ **then**
        if (N=0) **then**
            Errors are at $1^{st}, 3^{rd}, 6^{th}, 9^{th}$ row only which can be detected only by nonzero syndrome vectors in R_c but can not be corrected;Z_{out}=1;
        end if
        if (N=1) **then**
            Error in any one row out of $1^{st}, 3^{rd}, 6^{th}, 9^{th}$ row which can be corrected by hamming code along the column direction;N=0;
            Make the corresponding row in R_c zero;
        end if
    end if
    if $((C_1 = 1) \&\& (C_2 = 1))$ **then**
        Single bit error occur in A'$_{re}$ and it will be corrected;
    end if;
    if $((C_1 = 1) \&\& (C_2 \geq 2)$ **then**
        if $((Y_l \geq 13) \&\& (X_k \geq 13) \&\& (X_m \geq 13))$ **then**
            There is a CMBU of size 2×2 is detected. Then bits at the intersection of l^{th} row and and k^{th} and m^{th} columns in A'$_{re}$ will be corrected;
            N=1;Make the corresponding row in R_r and R_c zero.
        end if
        if $((Y_l \geq 13) \&\& (X_k \leqslant 12) \&\& (X_m \leqslant 12))$ **then**
            Bits at the intersection of l^{th} row and and k^{th} and m^{th} columns in A'$_{re}$ will be corrected;
            N=0;Make the corresponding row in R_r and R_c zero.

28: **end if**
29: **end if**
30: **if** (($Y_l \leqslant 12$)&&(More than Two rows of R_c having nonzero syndrome values)) **then**
31: Bits overlapped by l^{th} row, and columns for which R_c gives nonzero syndrome values will be corrected in A'$_{re}$;Make the corresponding row in R_r and R_c zero.
32: **end if**
33: **if** (($C_1 \geq 2$)&&($C_2 = 1$)) **then**
34: **if** ($X_k \geq 13$)&&($Y_l \geq 13$)&&($Y_m \geq 13$) **then**
35: There is a CMBU of size 2×2 is detected. Then bits overlapped by k^{th} column and l^{th} and m^{th} row in A'$_{re}$ will be corrected;
36: N=1;Make the corresponding row in R_r and R_c zero;
37: **end if**
38: **if** ($X_k \geq 13$)&&($Y_l \leqslant 12$)&&($Y_m \leqslant 12$) **then**
39: Bits overlapped by k^{th} column and l^{th} and m^{th} row in A'$_{re}$ will be corrected;
40: N=0;Make the corresponding row in R_r and R_c zero;
41: **end if**
42: **if** (($X_k \leqslant 13$)&&(More than Two rows of R_r having nonzero syndrome values)) **then**
43: Bits overlapped by k^{th} column, and rows for which R_r gives nonzero syndrome values will be corrected in A'$_{re}$;Make the corresponding row in R_r and R_c zero;
44: **end if**
45: **end if**
46: **if** (($C_1 \geq 2$)&&($C_2 \geq 2$)) **then**
47: Bits overlapped by rows and columns for which nonzero syndrome vectors are generated will be corrected in A'$_{re}$;
48: **end if**
49: C_1= C_1-1,C_2= C_2-1;
50: A$_c$= A'$_{re}$;
51: R_r = A'[R,R]× transpose(H[h,R]);
52: R_c = transpose(A'[R,R])× transpose(H[h,R]);
53: i <= i + 1;
54: **end while**

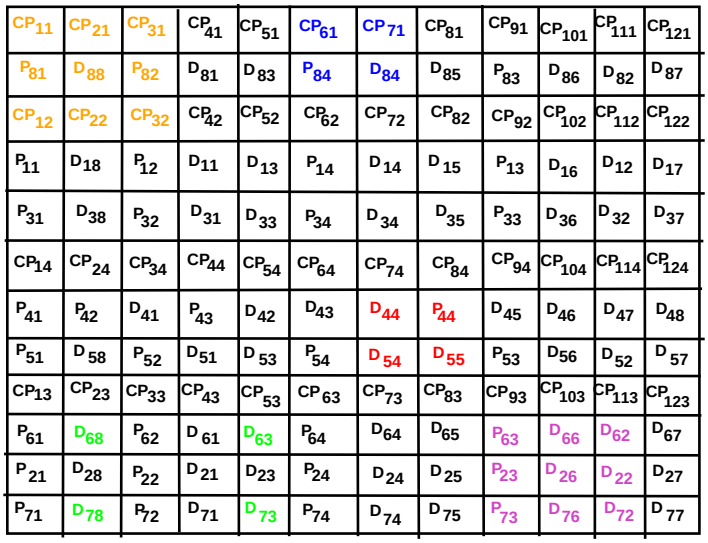

Figure 6.5: Arrangement of data and parity bits in a data matrix of size 12×12 after selective bit placement

greater than twelve. In Figure 6.6, p_r and p_c are parity bits added with HPC along the row and column respectively.

Error detection and correction in data matrix during decoding is described using some examples. Different kinds of error pattern are shown in the Figure 6.5 and Figure 6.6, but it is assumed that they have occurred at different instance. In Figure 6.5, clustered error of size 2×2 at bit positions D_{44}, P_{44}, D_{54} and D_{55} (marked by red color) generate syndrome vectors having the value greater than twelve at 7^{th} and 8^{th} row in both R_r and R_c. From these informations errors at these four bits can be easily corrected. Along with the clustered errors our proposed model is also able to correct multi-bit errors at discrete locations as bits at positions D_{68}, D_{63}, D_{78} and D_{73} marked by green color in Figure 6.5. Here 10^{th} and 12^{th} row in R_r and 2^{nd} and 5^{th} row in R_c generate nonzero syndrome vectors having the value less than twelve and the erroneous bits will be corrected by the decoding algorithm 7.

Sometimes multiple iterations may be required to correct all of the erroneous bits. As for example four erroneous bits at CP_{61}, CP_{71}, P_{84} and D_{84} in Figure 6.5 generate syndrome vectors having the value greater than twelve in second row of R_r and 6^{th} and 7^{th} row in R_c. In this case, first iteration corrects P_{84} and D_{84} and in the second iteration CP_{61} and CP_{71} will be corrected using the Ham-

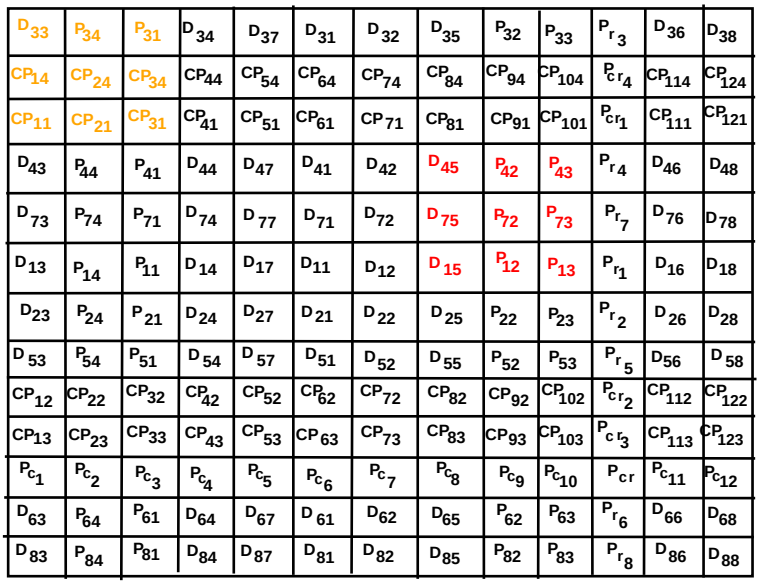

Figure 6.6: Arrangement of data and parity bits in a data matrix of size 13×13 after selective bit placement

ming code along the column direction. Larger cluster size (as for CMBU of size 3×3) can also be corrected by the proposed algorithm. Erroneous bits at $P_{63}, D_{66}, D_{62}, P_{23}, D_{26}, D_{22}, P_{73}, D_{76}$ and D_{72} generates three nonzero syndrome vectors in R_r and three nonzero syndrome vectors in R_c. Bits overlapped by row and column having nonzero syndrome vectors can be easily corrected but it may also increase the chance of mis-correction. Using the selective placement strategy described above, some group of three erroneous bits generate zero syndrome vectors so they can not be detected. Clustered error at bit positions $CP_{11}, CP_{21}, CP_{31}, P_{81}, D_{88}, P_{82}, CP_{12}, CP_{22}$ and CP_{32} will generate nonzero syndrome vectors at 2^{nd} row in R_r and $1^{st}, 2^{nd}$ and 3_{rd} row in R_c. Using these information errors at P_{81}, D_{88}, P_{82} can be corrected only. But the other erroneous bits remain uncorrected.

HPC with parity can detect and correct this type 3×3 adjacent multi-bit errors as illustrated in Figure 6.6. Errors at bit positions $D_{33}, P_{34}, P_{31}, CP_{14}, CP_{24}, CP_{34}, CP_{11}, CP_{21}, CP_{31}$ generate syndrome vectors with values greater than twelve in $1^{st}, 2^{nd}$ and 3_{rd} row in R_r, and R_c which can correct the clustered errors of size 3×3 using HPC with selective bit placement.Similarly, 3×3 CMBU at bit positions $D_{45}, P_{42}, P_{43}, D_{75}, P_{72}, P_{73}, D_{15}, P_{12}, P_{13}$ can easily be corrected using HPC with parity. In algorithm 7, Z_{out} output will be high when error is detected but can

not be corrected. To avoid error accumulation due to residual errors, golden copy of the configuration data stored in a secondary or flash memory that can be downloaded when Z_{out} is high.

In this way selective bit placement helps to detect and correct multiple adjacent erroneous bits in a data matrix. If in a single iteration errors in the interleaved group can not be corrected multiple iterations can be used. Optimizing correction coverage and correction time number of maximum iteration or iteration threshold (I_{th}) would be set by the user. R_{Hi} and R_{Vj} in Figure 6.2 contains redundant bits generated due to Hamming code along horizontal and vertical direction in HPC on a data set. Overhead due to the redundant bits of our proposed method remain same with that of the method proposed by authors in [89] and only decoding circuit complexity may increase marginally due to interleaving logic. Physically the redundant bits will be stored in a RAM on the FPGA board. Hence, frame interleaving (helps to place physically adjacent frames into multiple groups) and selective bit placement (disperse multiple adjacent erroneous bits in a configuration frame into multiple data matrices) along with HPC helps to correct different clustered errors shown in Figure 6.2 that are not possible with traditional HPC used by authors in [89]. Each step of our proposed algorithm named as HPCFISBP is described in algorithm 8.

Algorithm 8 HPC with Frame interleaving and selective bit placement Algorithm

Require: Erroneous configuration frames, I_{th}
Ensure: Corrected or partially corrected configuration frames

1: After read back the configuration data from configuration memory arrange the available configuration frames along columns (1 to M) and rows (1 to N) of a matrix as shown in Figure 6.1;
2: According to value of HI and VI, configuration frames are chosen from the rows and columns of the matrix and forms a interleaving group as shown in Figure 6.2;
3: Arrange the configuration frames of an interleaving groups into multiple horizontal (GROUPH) and vertical groups (GROUPV);
4: Based on the size of the data field of HPC fixed the window size as indicated by blue square in Figure 6.2. For example if (n,k) Hamming code is used as component of HPC then window size is k×k;
5: Prepare a data matrix taking data from all configuration frames within a window according to the selective bit placement strategy.
6: **while** ($I < I_{th}$) **do**
7: Apply HPC on the data set prepare at step-5 and correct the erroneous bits;
8: Generate Syndrome vectors;
9: **if** (syndrome vectors are non zero) **then**
10: I= I + 1;
11: **end if**
12: **if** (syndrome vectors are zero) **then**
13: I=I_{th};
14: **end if**
15: **end while**
16: Go to step-5 and form another data matrix;
17: Go to step-4 and performs the same operation until all the configuration frames within an interleaving groups are covered.
18: Go to step-2 and perform same operations on another group.

6.3 Hardware implementation of HPCFISBP

Application hardware, placed into the configuration area of the FPGA consists of few sub-components, which may be stated as standard or custom IPs. Bit file of Kintex-7 FPGA contains 91,548,896 bit which are arranged in 28326 configuration frames. Each frame consists of 3232 bits that are arranged in a matrix having 101

rows and 32 columns. Here we have used Slave SelectMAP interface [156] with 32 bit bus width to readback data from configuration memory. Here custom build slave interface controller is written that control interface signal of selectMAP port as shown in Figure 6.7. After power on reset RDWR and CSI_B signals go

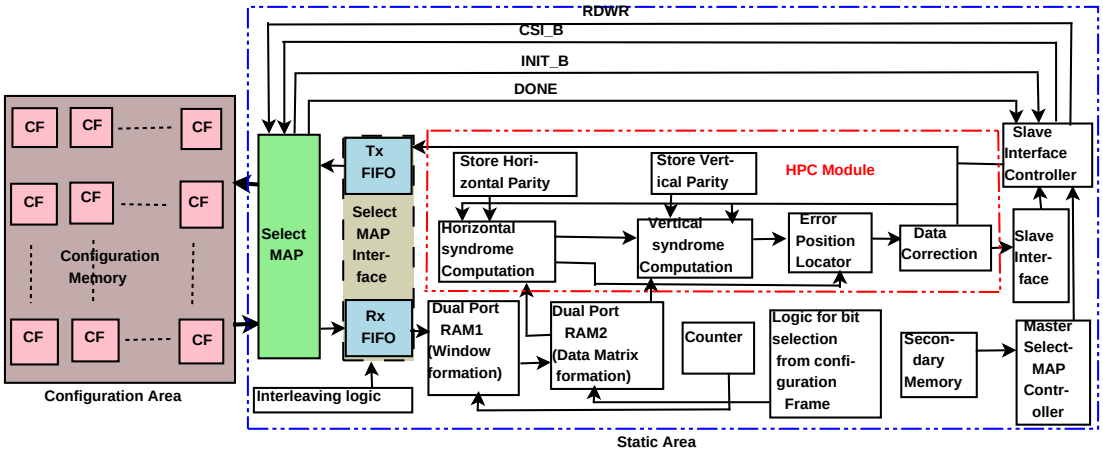

Figure 6.7: Hardware implementation of HPCFISBP

low and bus width detection procedure will be started. After the bus width is confirmed loading of bit stream will be started. The first step of bit stream loading is synchronization and in this step a special synchronization word (0xAA995566) will be sent to the configuration logic that prepare the device for upcoming data and aligns the configuration data with the internal configuration logic. Any data before synchronization in data pin of selctMAP is ignored except bus width auto detection sequence. In the next step, device ID will be checked that prevents configuration bit stream generated for a device to be downloaded into another device. In the third step, configuration data will be loaded into configuration memory and device starts to calculate CRC value from the configuration data. After loading of data is completed bit stream generate *checkCRC* command along with expected value of CRC. If the expected CRC value dose not match with calculated CRC value INIT_B signal goes high to abort the process and bit stream loading process will fail. On the other hand if the expected CRC value matches with the calculated CRC value DONE signal goes high and some special startup sequence will be sent to the device. Steps for bit file loading is shown using a flow diagram in Figure 6.8 and mentioned as configuration phase. for details meaning of each register and signal used in flow diagram user can consult [156].

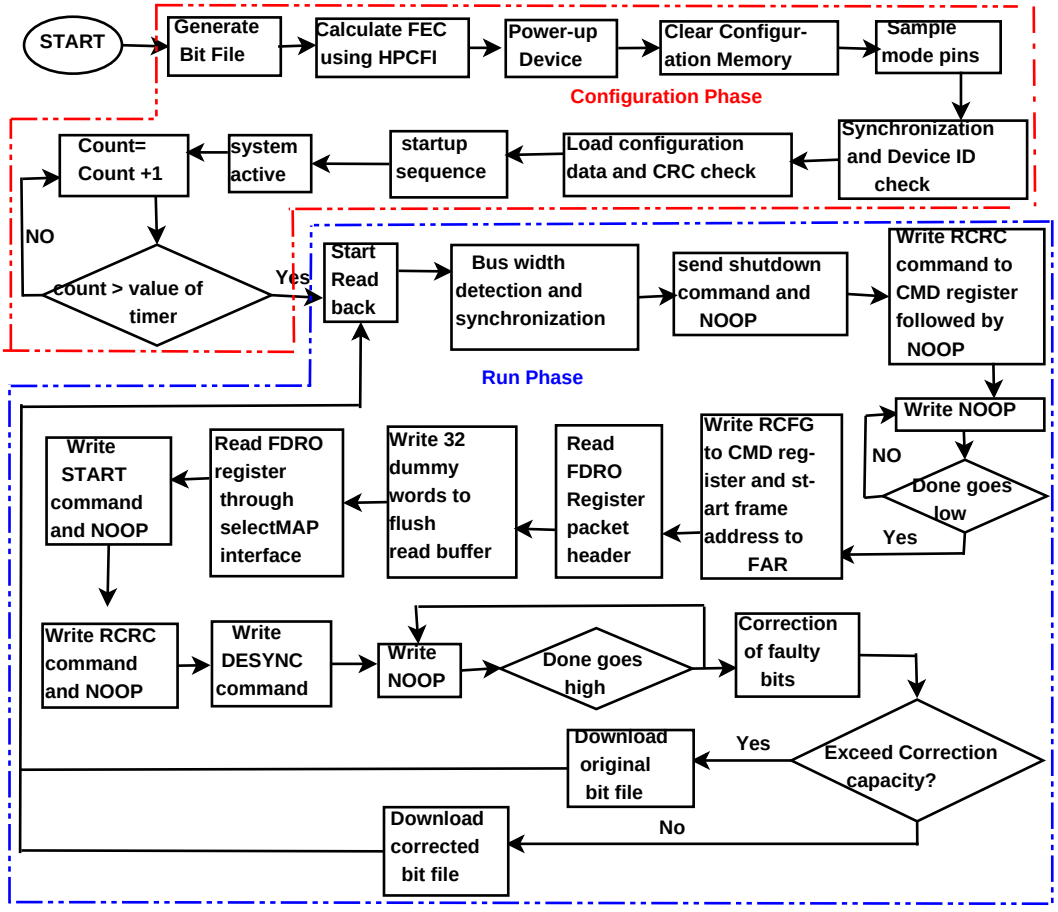

Figure 6.8: Hardware implementation work-flow for HPCFI

After the device is activated a counter starts and when the counter value reaches a certain threshold read back process will start. There are different registers in the configuration memory having specific address. Some of them are read only, some of them are used only for writing purpose and some of them are used both for reading and writing purpose. Some of the important registers used in the design are Frame data register input (FDRI) used for writing the configuration data, Frame data register output (FDRO) used for reading the configuration data and Frame address register (FAR) used to give the address of the register to be read or write. Steps for read back of the data is shown in Figure 6.8 and designated as run phase.

Here configuration data will be readback continuously with some time interval and written into the Rx FIFO. Interleaving logic selects configuration frames from Rx FIFO and write into dual port RAM1 dedicated for window selection. Then

logic for bit selection selects data bit from RAM1 and write them into dual port RAM2. In the next step HPC module read data from RAM2 and send them into the horizontal syndrome computation and vertical syndrome computation block simultaneously as shown in Figure 6.7. Based on the calculated syndromes error positions in a data matrix can be detected and corrected. If there is no others erroneous bits remain in the data matrix data will be written back into the TxFIFO from where it will be sent to configuration memory. Otherwise data will be written into RAM2 again and will be used for syndrome computation for next iteration. Slave interface controller is an interface between the selectMAP and master. This interface can be Advanced eXtensible Interface (AXI), Fast Simplex Link (FSL) and Processor Local Bus (PLB) but here the link is custom as custom selectMAP controller is used as master. When the proposed HPCFISBP is unable to correct all erroneous bits in the configuration data golden copy stored in the secondary memory will be downloaded into the configuration memory through selectMAP master controller.

6.4 Result and Performance Analysis

Proposed fault correcting model is implemented for seven series FPGA using Xilinx ISE 14.5 platform and tested using behavioral simulations. An application design has been used to generate the bit file. Figure 6.9 compares the performance of the proposed HPCFISBP code with the other existing codes in terms of hardware complexity and BER. During the BER calculation input signal to noise ratio is taken as 4dB [89]. Existing multibit error correcting codes with good error correcting performance like LDPC, Turbo, RS product code consume very high hardware resources whereas Hamming code and HPC [89] consume fewer hardware resources, but their BER performance is not satisfactory. Our proposed HPCFISBP code shows good BER performance compared to Hamming code and HPC with slightly higher hardware complexity comapred to HPC as shown in Figure 6.9. Figure 6.10 (a) indicates that the error correction coverage of our proposed model (HPCFISBP) outperforms model proposed by authors in [89] for the different number of injected errors in the configuration memory. Not only clustered error if discrete MBUs present along same rows and columns

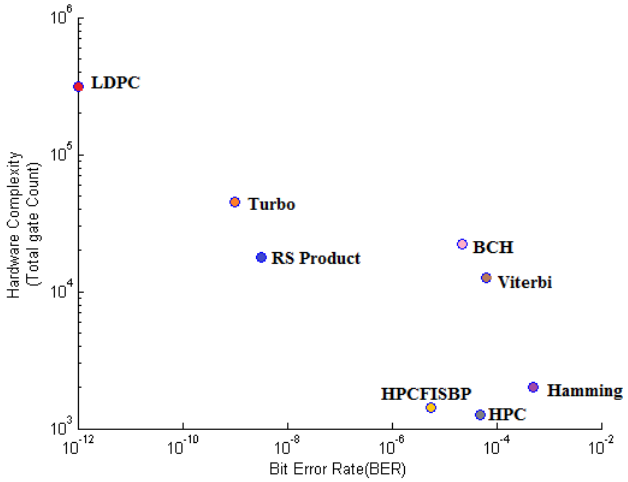

Figure 6.9: Hardware Complexity vs BER for different ECC

HPC model in [89] cannot rectify them. but HPCFISBP can correct these types of MBU. As there is more chance of non-repairable MBUs by HPC at higher values of injected faults, there is a notable improvement in error correction coverage of HPCFISBP over HPC for higher values of injected errors in the configuration memory. For both single and double iterations HPCFISBP gives much better performance compared to HPC as illustrated in Figure 6.10 (a). Residual errors

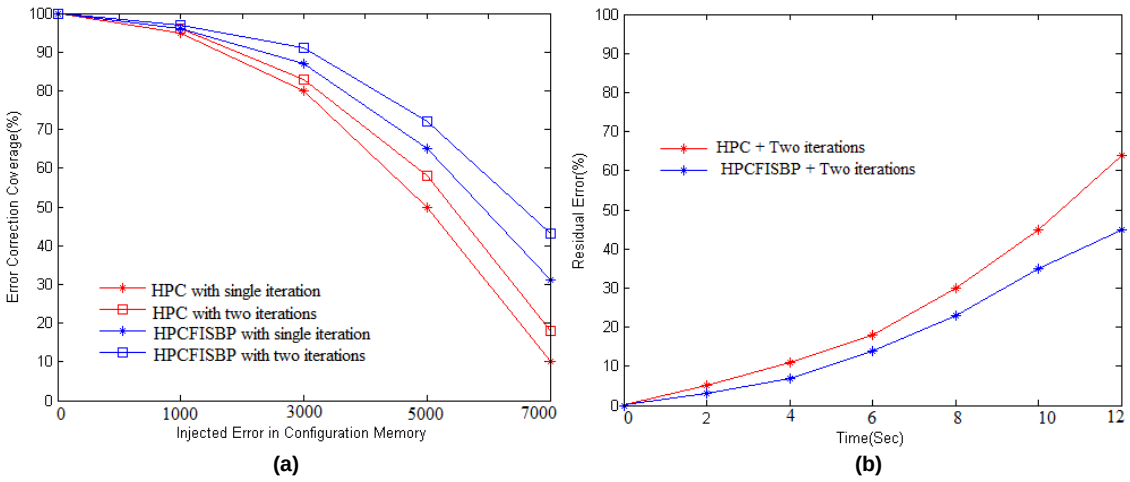

Figure 6.10: (a) Comparison of error correction coverage of HPCFISBP with HPC proposed by authors in [89] (b) Residual errors in configuration memory at different time instance after error correction by HPC and HPCFISBP

are the errors remain in configuration memory after error correction by the error

correcting models. To test the performance of HPCFISBP and HPC for residual errors we have injected 500 errors in every second and then apply HPCFISBP and HPC with I_{th} as two. With the increase of I_{th} residual errors decrease but at the same time error correction time will increase. Here we have set I_{th} as two but it can be tuned to any value for other applications. Figure 6.10 (b) illustrates that residual errors are less in configuration memory when corrected using HPCFISBP compared to the case when corrected by HPC.

Variation of error correction coverage and error correction time with different interleaving depth is shown in Figure 6.11. Here we have kept horizontal and vertical interleaving depth same. With the increase of interleaving depth number of interleaving groups increase that will increase error correction time steadily. This is because error correction operation will be performed on one interleaving group at a time. Sometimes charge particles may damage the adjacent configuration frames so initially with the increase of interleaving depth error correction coverage will increase. In this situation, multiple erroneous configuration frames are in different interleaving groups that enhances error correction coverage. Figure 6.11 shows that error correction coverage of HPCFISBP increases

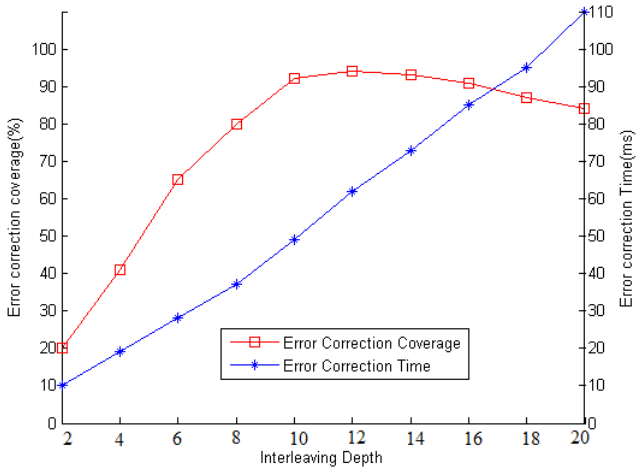

Figure 6.11: Variation of error correction coverage and error correction time with different interleaving depth

steadily up to a certain interleaving depth (Here it is equal to ten). When interleaving depth increases further multiple erroneous configuration frames which are physically far apart come within the same window in a interleaving group. It

degrades error correction coverage of HPCFISBP. In our design, if values of HI and VI increase beyond twelve, error correction coverage of HPCFISBP deteriorates slightly. Changing interleaving depth from ten to twelve will improve error correction coverage marginally. Hence, optimizing both error correction coverage and error correction time, proper interleaving depth is ten for this design as clearly seen from Figure 6.11. Table 6.2 shows the variation of error correction coverage due to the variation of HI and VI.

Table 6.2: Variation of error correction coverage with interleaving depth

	VI=2	VI=4	VI=6	VI=8	VI=10	VI=12	VI=14
HI=2	40%	51%	60%	73%	82%	85%	89%
HI=4	45%	59%	70%	77%	87%	92%	95%
HI=6	51%	65%	78%	87%	90%	92%	96%
HI=8	55%	67%	78%	89%	92%	95%	98%
HI=10	68%	75%	89%	93%	95%	98%	99%
HI=12	70%	79%	87%	93%	100%	97%	99%
HI=14	74%	78%	89%	93%	99%	100%	98%

Table 6.3 compares our proposed code with different existing codes in terms of error correcting performance, latency, decoding circuit complexity, resource utilization, power consumption and redundant data. Table 6.3 is prepared using multiple iterations over 32×32 memory unit keeping the code rate constant. As size of each configuration frame is 101×32 we have chosen memory unit of size 32×32 for preparing the table. For HPCFISBP interleaving depth is taken as four. Due to the random nature of MBU, a range of the error correcting capability for different error correcting codes is given instead of exact value. In Turbo code, the number of redundant data depends on the number of shift registers(v), modulo two adders (K) and input data of length L bit. Here values v,K and L are taken as 2,2 and 1024 bit respectively. It is observed that in terms of error correction LDPC and turbo code give the best performance, but their decoding complexity and overhead are very high. Though overhead of hamming code is small, its error correcting performance is not good. BCH and HPCFISBP require the same amount of redundant data to protect 32×32 memory unit, but HPCFISBP gives better error correcting performance compared to BCH code. It can also be seen that HPCFISBP provides far better error correcting capability compared to HPC

Table 6.3: Comparison between Proposed ECC with the other existing ECC

	BCH Code (127,71) [157]	Hamming Code (Xilinx ECC IP) [158]	LDPC [84]	Turbo [72]	HPC [89]	HPCFISBP
ECC performance	between 30% and 50%	<30%	>95%	>95%	between 60% and 90%	between 80% and 98%
Latency	Low	Very Low	Moderate	Long	Moderate	Moderate
Decoding Complexity	Moderate	Low	High	High	Low	Low
Overhead (Redundant bit)	840	768	1024	$2v(L+K)-L$	882	882
Resource Utilization	2349 (#Slice Reg.)	1282(# Slice Reg.)	1750k (# Gate)	4146 (# Slice Reg.)	-	1152 (#Slice Reg.)
power(mw)	-	-	690	-	-	150.7

keeping same overhead and almost same decoding circuit complexity. Selection of data matrix size or component Hamming code for HPC plays a vital role in error correcting performance and error correction time as shown in Figure 6.12. With the increase in data matrix overall error detection and correction time in a memory element will reduce but at the same time error correcting performance of both HPC and HPCFISBP will degrade and the effect is more predominant in HPC compared to HPCFISBP. The reason behind is that adjacent MBU of different sizes can not be corrected by HPC in a single iteration and some of them can not be corrected when it will be in a single data matrix. To correct these type of error either multiple iterations are required or window size need to be reduced which in turn increases error correction time. On the other hand, these error patterns can be corrected easily by HPCFISBP using single or less umber of iteration compared to HPC and by dispersing the erroneous bits into multiple data matrices. Though, with the increase in the data matrix size, number of data matrix within a interleaved group is reduced but error detection/correction time within a interleaved group will increase due to increase in iteration numbers. Hence, initially error correction time will decrease with the increase of the data matrix size but after a certain data matrix size, error correction time for a interleaved group become almost unchanged as shown in Figure 6.12.

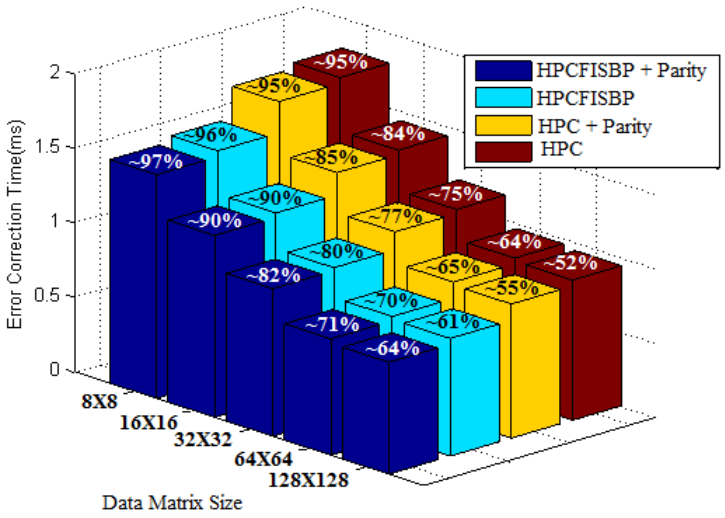

Figure 6.12: Variation of error correction time and error correction capability with size of data matrix

6.5 Conclusion

In this work, we have proposed a novel error correcting model integrating low complexity HPC and frame interleaving with selective bit placement strategy to correct MBUs arising due to soft errors in the configuration memory of FPGA. The proposed error correcting models give good error correcting performance compared to HPC without sacrificing in terms of overhead due to redundant bits and decoding complexity. We have discussed details performance analysis of the proposed method by varying interleaving depth and size of the component Hamming code of HPC. We have also proposed hardware implementation of the proposed method with continuous configuration memory read and write through selectMAP interface. In future, we are planning to use partial reconfiguration along with the proposed model to enhance error correction coverage.

Chapter 7

Efficient Dynamic Priority Based Soft Error Mitigation Techniques For Configuration Memory of FPGA Hardware

Configuration memory of FPGA devices are very susceptible to charge particles present in the cosmic ray or generated during different laboratory experiments. Modern FPGA devices use different multi-bit ECC to mitigate the effect of soft error arising due to radiation. With the increase of the error correcting capability decoding circuit complexity of ECC also increases. In this chapter, we have proposed efficient single bit as well as multi-bit error correcting methods using simple parity equations and Erasure code. At the same time we have separated error detection and correction process that not only enhances error correction coverage, but also reduces error correction time and complexity of decoding circuit. Use of DPR along with a simple hardware scheduling algorithm based download manager helps to perform the error correction in the configuration memory without suspending the operations of the other hardware blocks. We propose a first of its kind methodology for novel transient fault correction using efficient ECCs with hardware scheduling for FPGAs. We have measured different parameters like fault recovery time, power consumption, resource overhead and error correction

efficiency to estimate the performance of our proposed methods.

7.1 Introduction

With the development of fabrication technology, solid state devices are gradually reducing in size, hence node voltages of CMOS transistor also reduces. If the charge injected by incident particles goes above a certain threshold (also known as critical charge [159]) it can create SBUs and MBUs in different embedded devices like FPGAs which tremendously affect the reliability of the devices in the field. Present research outcomes [138] show that with the increase of circuit density, multiple number of adjacent CMOS transistors in FPGA devices are affected by low momentum and highly energized incident charge particles.

An MBU occurs when charged particles hit the memory and affects sensitive zones of multiple cells (mainly channels of MOS transistors). With the shrinking of the transistor size, depletion region of one transistor may span into multiple transistors which increases the probability of charge sharing between the neighboring circuit nodes. The charge sharing may create MBUs in the adjacent memory cells. To study the occurrence probability of SBUs and MBUs, a particle strike simulation is conducted using a well-known simulation tool called Geant-4 [160]. A graph is plotted based on simulation results as shown in Figure 7.1 which illustrates how cells are damaged when radiation from neutron source of different energies (1 GeV, 5 GeV, and 10 GeV) hit a memory element designed for 40 nm technology. It is observed from this graph that above 50% of soft errors in memory are multi-bit in nature. Figure 7.2(b) (taken from [97]) also shows the occurrence probability of MBUs for 45 nm technology. As Kintex FPGA, used in our experiment is 28 nm technology based device it is expected that probability of occurrence of MBUs is more in Kintex FPGA compared to 40 or 45 nm technology based devices. The number of the memory cells in the configuration memory of FPGAs that will be affected by radiation depend on radiative flux density. Authors in [161] showed that soft error rate (SER) increases steadily with the elevation from the ground. Similarly, configuration memory of FPGA devices will be affected tremendously when they are used in different HEP experiments like ALICE [162], CBM [80] Experiment.

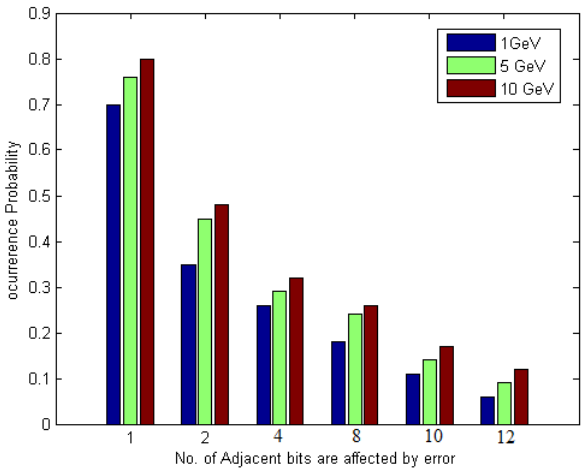

Figure 7.1: Occurrence probability of different MBU and SBU (indicated by '1' along x-axis) patterns for different Neutron energy

The most common methods of error mitigation in configuration memory of FPGA devices are TMR, CED, readback or blind scrubbing and multi-bit ECC. The traditional error correction methods either have large overhead and logic resources or use complex decoding circuit. Authors in some recent literatures tried to use simple block code based ECC like matrix code (MC [93]) against MBUs though they are not sufficient when SER is very high. Optimizing decoding circuit complexity and error correction capability we have proposed simple parity based and erasure coding based error correcting models that provide simple decoding circuit with large error correction coverage. In order to reduce error correction time and decoding circuit complexity further we have separated error detection and correction process as introduced by *Ebrahimi et.al* in [97].

Error detection and correction on full configuration memory at a time using different ECC increase system latency. At the same time downloading of bit file after error correction stops the system operations momentarily that are not acceptable for real time applications. To alleviate these effects, DPR with hardware scheduling technique can be used with EDAC where ECC corrects only a particular portion of the configuration memory at a time and download the corrected portion of the bit file without hampering the normal system operation involving the other blocks in the design. Using partial reconfiguration, a hardware design is partitioned into a number of partially reconfigurable (PR) region as shown in Figure 7.2(a). All PR regions will not be in the active state at the same time

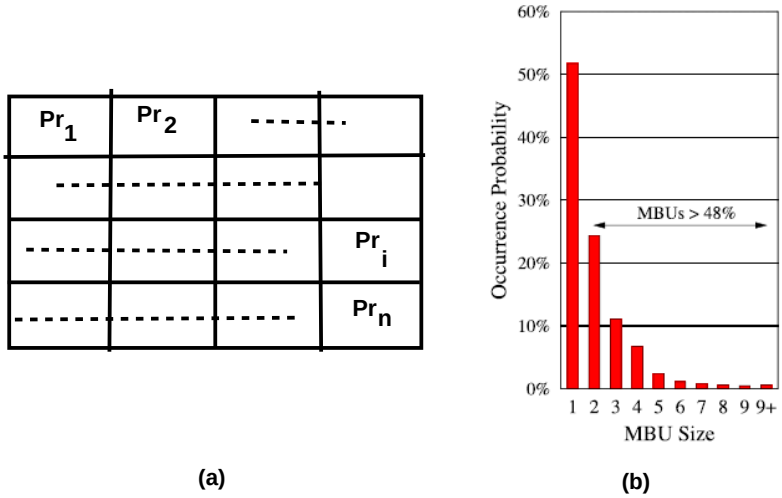

Figure 7.2: (a) Partitioned of configuration memory into n number of regions (b) MBU distribution in 45 nm SRAM based FPGA (Taken from [97]

when a hardware circuit with partial reconfiguration runs on the FPGA. The PR region which is not in the active state can be assumed in the idle state. As for example i^{th} PR region (PR_i) in Figure 7.2(a) is in the idle state at a particular instant. If reading, error correction and downloading of PR_i can be done within its idle state, then it will not hamper the normal function of the system. It will give better performance if the PR region for error correction can be selected according to some priority which is calculated using a predefined scheduling algorithm. This procedure enhances overall system performance and reduces the reconfiguration latency. In some present literatures like [98, 102] fault detection and correction is integrated with hardware scheduling and partial reconfiguration but the proposal raised in this chapter comes from a different aspect where fault correction priorities of different hardware blocks are calculated from the task period and its criticality.

In this paper our key contributions are:

- An efficient erasure code and a parity based low complexity ECCs are used to mitigate the effect of soft errors in the configuration memory of FPGAs. In comparison with the other existing design, the proposed codes have less complex decoding circuits and give better error correcting performances.
- To reduce reconfiguration time and power consumption we have used DPR along with the proposed fault correcting models.

- A novel priority based bit file downloading algorithm is also proposed which prevents suspension of the system operation during downloading of partial bit file.

7.2 Proposed Modified Matrix Code Algorithm

Modified matrix code (MMC) is a simple parity based ECC. MC proposed by the authors in [93] corrects the erroneous data using Hamming code along the row and parity code along the column of a matrix. The proposed method is called modified matrix code since the data to be corrected are arranged in a matrix on which parity equations are used along both the diagonals and vertical directions. MC can correct upto two bit errors in any row and only one bit error in other rows. When number of error is more than two, error detection and correction capability of MC is less than 100% as shown in Figure 7.3 however error detection capability of MMC is always 100%. MMC can correct any three bit errors with 100% efficiency but correction of more than three bit errors depend on the error pattern. It is justified by our simulation of ${}^{n\times n}C_3$ error combinations in a $n\times n$ matrix as shown in Figure 7.3. Code rates of MC and MMC are almost same. As for example, in 64 bit window, code rates of MC and MMC are 0.57 and 0.58 respectively. Hence, modification made in this paper over MC helps to achieve better overall EDAC coverage without sacrificing in terms of code rate. In MMC, each configuration frame is partitioned virtually into multiple R×R matrices (also known as window), and MMC code is used to correct and detect errors in each window as shown in Figure 7.4. R is the number of rows and columns of each window and here the MMC algorithm is described with R=7 in Figure 7.5(a). Circle superimposed on the summation symbol '$\bigoplus$' indicates modulo-2 sum. A simplified example in Figure 7.5(a) illustrates how the proposed MMC can achieve multi-bit EDAC using diagonal and vertical parity bits within a single window. At the beginning of encoding process, data of one window size will be read and stored in a matrix and then parity bits are generated along both diagonal and vertical directions of the matrix using equation 7.1 to equation 7.6. Here X indicates bits within the matrix.

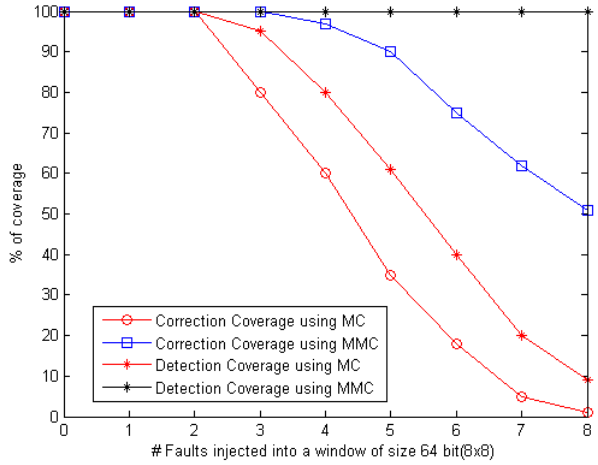

Figure 7.3: Detection and correction coverage of MMC and MC over 64 bit data

$$Cp_i = \bigotimes_{j=0}^{i-1} X_{(i-1)+((R-1)*j)} \forall i = 1 to R \tag{7.1}$$

$$Cp_i = \bigotimes_{j=i-R}^{R-1} X_{(i-1)+((R-1)*j)} \forall i = (R+1) to (2*R-1) \tag{7.2}$$

$$Cn_i = \bigotimes_{j=0}^{i-1} X_{(i+(R*(R-1))-1)-((R+1)*j)} \forall i = 1 to R \tag{7.3}$$

$$Cn_i = \bigotimes_{j=0}^{(2R-i-1)} X_{(i-R)+((R+1)*j)} \forall i = (R+1) to (2*R-1) \tag{7.4}$$

$$Vpo_i = \bigotimes_{j=0}^{((R-1)/2)} X_{(i+(R*2*j)-1)} \forall i 1 \ to \ R \tag{7.5}$$

$$Vpe_i = \bigotimes_{j=0}^{((R-1)/2-1)} X_{((R+i-1)+(R*2*j))} \forall i = 1 \ to \ R \tag{7.6}$$

Cp and Cn are an array of ($2R$-1) bits and store parity bits for the diagonal bits along unit positive and negative slope respectively for each window generated during encoding process (indicated by pink color blocks in Figure 7.5(a)). Similarly,Vpo and Vpe are arrays of R bits and store parity bits generated using the bits of each column in a window for odd and even positions respectively (indicated by yellow and green color respectively in Figure 7.5(a)). These parity bits along with the configuration data will be read back into the MMC block during

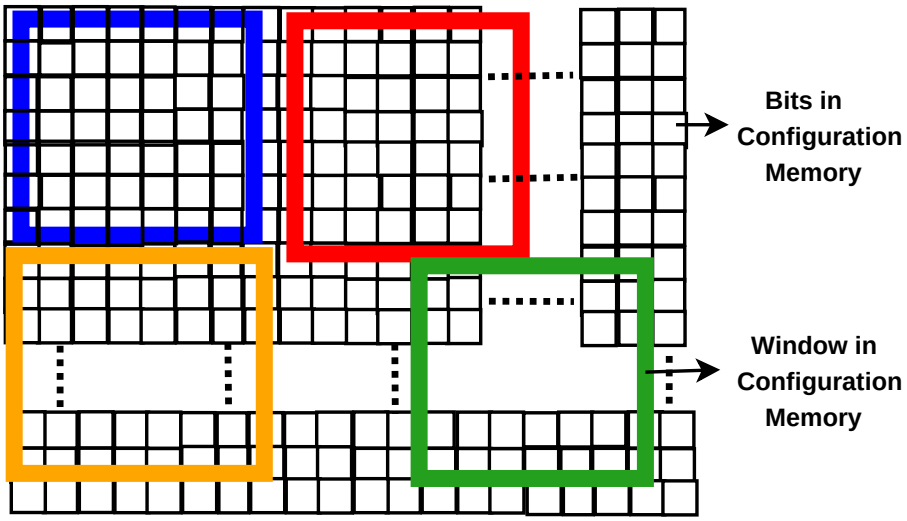

Figure 7.4: Window formation within a configuration frame

the decoding process. Working methodology of proposed MMC is shown in Figure 7.6. At the start of the decoding process erroneous data along with the parity bits (generated during encoding) are read and stored in three dual port RAMs. Diagonal parity bits are computed using data stored in the dual port RAM1 and is sent to the comparator block where it is compared with the stored diagonal parity bits. If a mismatch occurs, then, second parity computation block calculates the vertical parity bits. Results of the first and second parity computation block are sent to a decider block where position of the erroneous bits are generated and are forwarded again to RAM1 to correct the erroneous bits. After multiple iterations, if error is not corrected then error flag becomes high to indicate the presence of uncorrectable errors.

Now the above mentioned EDAC methodology using MMC is explained with the help of the decoding algorithm as described in algorithm 9. The key concept behind decoding process of MMC is parity decoding with majority voting. Erroneous bits in the matrix will be detected if at least two parity equations out of the three parity equations (along positive diagonal (equation 7.1 and 7.2), along negative diagonal(equation 7.3 and 7.4), along even or odd vertical (equation 7.5 or 7.6)) through the bit gives wrong result. On the other hand if any one of the three parity equations give wrong result, error will be detected within the window but can not be corrected. Decoding process can be discussed using the following examples. During the decoding process, the matrix $A^{'}$ stores 49 (7x7) bits of

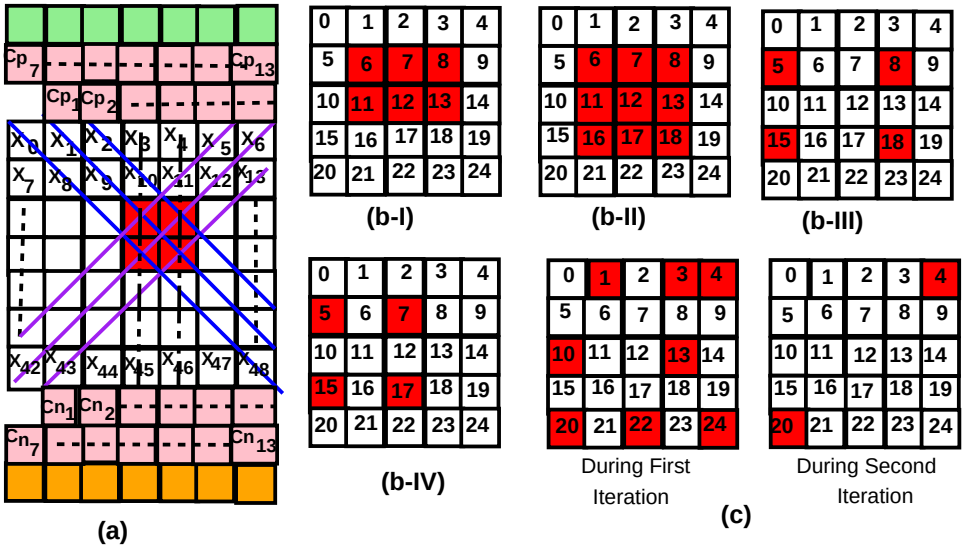

Figure 7.5: (a)Encoding/Decoding using 7×7 window (b)Different error patterns (c) Error Correction using Multiple Iterations

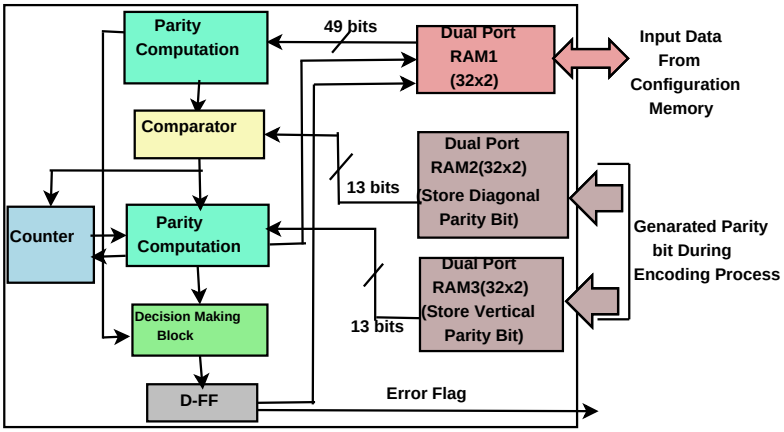

Figure 7.6: Working Methodology of the proposed MMC code

erroneous data obtained from the configuration memory. I_{th} is the number of iterations set by the user. There is a trade-off between latency, Bit Error Rate (BER) and I_{th} as shown in Figure 7.7. Here, the latency is defined as the time required to detect or correct errors in the configuration memory and BER is defined as the ratio of corrected errors to the total number of errors injected. The increment of I_{th} will improve BER performance but at the same time, latency will also increase. As we target real-time applications, optimization of BER and latency are of high priority and hence we choose the value of I_{th} = 3 for this design, whereas the value of I_{th} can be tuned as per the applications need. In Figure 7.5(a), four

adjacent bits X_{17},X_{18},X_{24},X_{25} are affected by errors. Errors at X_{17} and X_{18} are corrected by odd vertical parity equations and parity equations along positive diagonal with unit slope through X_{17} and X_{18} respectively. Similarly, Errors at X_{24} and X_{25} are corrected by the even vertical parity equations and parity equations along negative diagonal with unit slope through X_{24} and X_{25} respectively. Hence, error in these four bits can be corrected using algorithm 9. Some typical error patterns are discussed in Figure 7.5(b). In Figure 7.5(b-I) bits at position $6, 7$ and 8 will be corrected using parity equations along positive diagonal with unit slope and even vertical parity equations through these bits. Similarly, bits at position $11, 12$ and 13 will be evaluated using parity equations along negative diagonal with unit slope and odd vertical parity equations through these bits. Clustered error in Figure 7.5(b-II) can be detected by positive diagonal through bits 6 and 18, negative diagonal through bits 8 and 16 and odd vertical parity equations through bits 11,12 and 13. Apart from the errors in contagious mem-

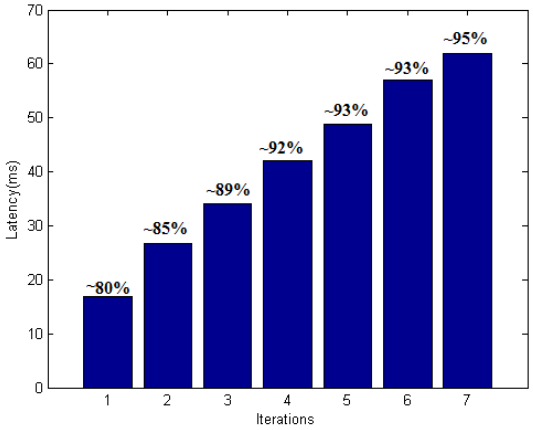

Figure 7.7: Variation of correction coverage of MMC and latency with iteration

ory locations, the proposed MMC code is also able to detect and correct errors in discrete positions. In Figure 7.5(b-III), the error occurred in bit positions 5,8,15 and 18 will be rectified using parity equation along positive diagonals with unit slope and negative diagonals with unit slope through these bits.

Algorithm 9 Decoding Algorithm for proposed MMC

Require: $A^{'}[R,R], Cp, Cn, Vpo, Vpe, I_{th}$

Ensure: Corrected or partially corrected $A[R,R]$;

i = 1;$p1$ = 0;$p2$ = 0;
while ($i \leqslant I_{th}$) **do**
Calculate $Cp^{'}, Cn^{'}, Vpo^{'}, Vpe^{'}$ using equ 7.1 to equ 7.6
    for ($i1$ = 1 →$(2R-1)$) **do**
$M1(i1)$=$Cp(i1) \oplus Cp^{'}(i1)$; $M2(i1)$=$Cn(i1) \oplus Cn^{'}(i1)$;
    end for
    for ($i2$ = 1 → $2R$) **do**
$M3(i2)$=$Vpo(i2) \oplus Vpo^{'}(i2)$; $M4(i2)$=$Vpe(i2) \oplus Vpe^{'}(i2)$;
    end for
$p1$ and $p2$ store the position of first one in $M1$ and $M2$;
    while $((p1 \leqslant (2R-1)) \| (p2 \leqslant (2R-1)))$ **do**
        $c1$ = 0;$x1$ = 0;$store1$ = $p1$;$store2$ = $p2$;
        if $((p1 \leqslant R)) \&\& (M1(p1) = 1))$ **then**
            $t1$ = 1;$k1$ = $p1$;$u1$ =$((R-k1)+1)$;$j1$ = $k1$;
            while $((c1 = 0) \&\& (j1 >= 1))$ **do**
$w1$ = $M2(u1)$;L1=store either $M3(t1)$ or $M4^{'}(t1)$;
                if $(((w1 = 1) \| (L1 = 1)))$ **then**
Update A'(j1,t1),M1(p1),M2(u1),M3(t1) or M4(t1),c1=1;
                end if
                $u1$ = $u1$ + 2;$t1$ = $t1$ + 1;$j1$ = $j1$ − 1;
            end while
        end if
        if $((p1 > R)) \&\& (M1(p1) = 1))$ **then**
            $t1$ =$(p1-R)$+1;$k1$ = R;$u1$ = $t1$;$j1$ = $t1$;
            while $((c1 = 0) \&\& (j1 \leqslant R))$ **do**
$w1$ = $M2(u1)$;L1=store either $M3(t1)$ or $M4(t1)$;
                if $(((w1 = 1) \| (L1 = 1)))$ **then**
Update A'(k1,j1),M1(p1),M2(u1),M3(j1) or M4(j1),c1=1;
                end if
                $u1$ = $u1$ + 2;$k1$ = $k1$ − 1;$j1$ = $j1$ + 1;
            end while

```
        end if
        if ((p2 ⩽ R)&&(M2(p2)= 1)) then
            b1 = 1;d1 =(R − p2 + 1);e1 = d1;j3 = e1;
            while ((x1 = 0)&&(j3 ⩽ R)) do
y1 = M1(d1); L2=store either M3(b1) or M4(b1);
                if (((y1 = 1)||(L2 = 1))) then
Update A'(j3,b1),M1(d1),M2(p2),M3(b1),M4(b1),x1=1;
                end if
                d1 = d1 + 2;b1 = b1 + 1;j3 = j3 + 1;
            end while
        end if
        if ((p2 > R)&&(M2(p2)= 1)) then
            b1 =(p2 − R)+1;d1 = b1;t2 = 1;j3 = b1;
            while ((x1 = 0)&&(j3 ⩽ R)) do
y1 = M1(d1);L2=store either M3(b1) or M4(b1);
                if (((y1 = 1)||(L2 = 1))) then
Update A'(t2,b1),M1(d1),M2(p2),M3(b1),M4(b1),x1=1;
                end if
d1 = d1 + 2;b1 = b1 + 1;j3 = j3 + 1;t2 = t2 + 1;
            end while
        end if
        if (((c1 = 1)||(x1 = 1))) then p1=d1-2;p2=u1-2;
        end if
        if ((c1 = 0)&&(c2 = 0)) then p1=p1+1;p2=p2+1;
        end if
    end whilei = i + 1;
end while
```

Error pattern in Figure 7.5(b-IV) will be detected by parity equations along positive diagonal through bits 5 and 17 and parity equations along negative diagonal through bits 7 and 15.

HPC used in [89] use (10,7) Hamming code as component code. As minimum distance (d_{min}) of Hamming code is three it can detect single bit error but cannot differentiate between single and double bit errors. Hence, HPC cannot detect or correct when multi-bit errors are present along both row and column (as in Figure 7.5(b-I) to (b-IV)) in a matrix. HPC can correct two bit errors with 100% efficiency but if more than two bit errors come in a window error correction coverage will depend on position of erroneous bits. Similarly matrix code [93] also

cannot detect or correct errors shown in Figure 7.5(b-I) and (b-II) and only detect errors shown in Figure 7.5(b-III) and (b-IV). On the other hand MMC can detect and correct the error patterns shown in Figure 7.5(b-I) and (b-III) and detect the error patterns in Figure 7.5(b-II) and (b-IV) as illustrated before. In some cases error may not be corrected in a single iteration, it will take multiple iterations (*i.e.*value of I_{th} is greater than one). In Figure 7.5(c), errors occurring at bit positions 1,3,10,13, 22, 24 will be corrected in the first iteration whereas, errors at bit positions 4 and 20 will be corrected in the second iteration. Proposed MMC is most efficient to correct either discrete multi-bit errors (as shown in 7.5(c)) or clustered errors when cluster size is small as shown in Figure 7.5(a). With the increase of cluster size as shown in Figure 7.5(b-II) efficiency of MMC starts to fall. For multi-bit errors with large cluster size, it is better to perform error correction on multiple configuration frames in parallel instead of single configuration frame at a time. In the next section we are going to discuss such multi-bit error correction schemes.

7.3 Error Detection using Interleaved MMC

In MMC, each configuration frame is scanned using $R \times R$ window and for each window separate diagonal, and vertical parity bits are calculated. In this process EDAC can be done on multiple windows simultaneously. Though error detection is 100%, error correction is less than 100% in MMC. With the decrease of window size, error correction capability of MMC in configuration frame will increase but overhead (redundant bits) for each configuration frame will also increase. Decreasing window size increases memory access time and error correction time for each configuration frame. Hence, overall latency will increase with decrease of the window size as shown in Figure 7.8, but for real time system, high latency can not be tolerated. Sometimes clustered error may not be possible to correct using MMC if multiple error clusters are within a single window. These clusters can be easily corrected if erroneous bits in the cluster can be spread virtually into multiple windows. It can be concluded that error correction capability of MMC depends on the position of the cluster within the configuration frame. MMC can correct one configuration frame at a time. Though it gives good error correcting

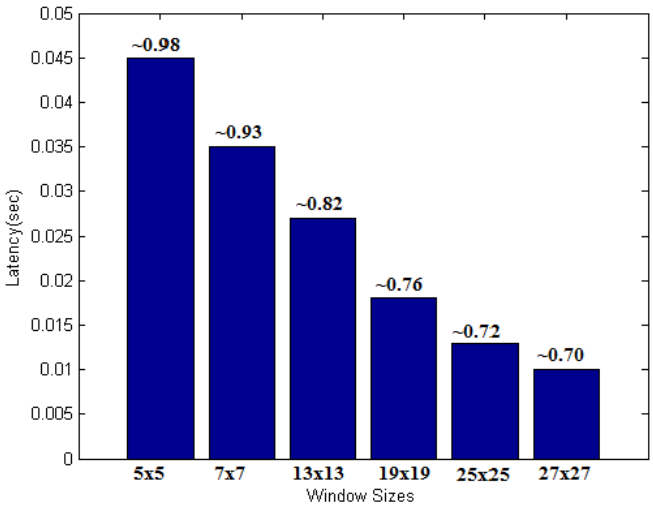

Figure 7.8: Variation of error correcting capability of MMC and latency with different window sizes

performance, it also takes a long time to check each configuration frame separately. Usually, in a configuration memory, some of the configuration frames are affected by errors. Hence, error correction latency can be reduced by performing the error correction operation on multiple configuration frames in parallel. To correct multiple configuration frames at the same time detection process needs to be separated from the correction process. Initially detection process detects erroneous frames and then correction technique corrects the detected erroneous frames. The separation of detection and correction process increases the error correction coverage.

It is already mentioned that error detection capability of MMC is 100% which makes MMC more suitable for error detection purpose. The number of redundant bits generated by MMC can be reduced by using the simple interleaving technique [163] when it will be used only for detection purpose. MMC with the interleaving along both diagonal and vertical directions can be termed as interleaved MMC (IMMC). Error detection performance of IMMC is given in section 7.7 with different interleaving depth. After detection of an erroneous frame, its content can be recovered using special kind of erasure code known as EVENODD coding (described in section 7.4). Hence, the objective of IMMC is to detect erroneous frame and not to locate the exact position of erroneous bits

within the configuration frame.

$$CpI(i1) = CpI(i1) \oplus \bigotimes_{k=0}^{\lfloor \frac{R}{dp} \rfloor} Cp(i1 + (k * dp)) \forall i1 = 1 \; to \; dp \tag{7.7}$$

$$CnI(i2) = CnI(i2) \oplus \bigotimes_{l=0}^{\lfloor \frac{R}{dn} \rfloor} Cn(i2 + (l * dn)) \forall i2 = 1 \; to \; dn \tag{7.8}$$

$$VpoI(i3) = VpoI(i3) \oplus \bigotimes_{q=0}^{\lfloor \frac{R}{ho} \rfloor} Vpo(i3 + (q * ho)) \forall i3 = 1 \; to \; ho \tag{7.9}$$

$$VpeI(i4) = VpeI(i4) \oplus \bigotimes_{r=0}^{\lfloor \frac{R}{he} \rfloor} Vpo(i4 + (r * he)) \forall i4 = 1 \; to \; he \tag{7.10}$$

Parity bits of IMMC technique is generated using the equation 7.7 to equation 7.10. The size of the configuration frame is 3232 bit (101×32) so a matrix of size 57×57 can be chosen to accommodate data of a configuration frame. In this chapter, it is assumed that maximum ten to twelve adjacent bits in a configuration frame are being affected by radiation so that size of the clustered error is quite small compared to a configuration frame. This assumption is validated from the simulation result plotted in Figure 7.1. At the first step of the detection process parity bits are calculated from both original configuration data (Cp, Cn, Vpe, Vpo) and erroneous configuration data $(Cp^{'}, Cn^{'}, Vpo^{'}, Vpe^{'})$ using equation 7.1 to equation 7.6 taking the value of R as 57. Then according to the value of interleaving depth along both diagonals (dp is the interleaving depth along positive diagonal and dn is the interleaving depth along negative diagonal) and vertical direction (he is the interleaving depth for bit in even position in a column, and ho is the interleaving depth for bit in odd position in a column) interleaved parity bits are calculated from the redundant bits of both original configuration data $(CpI, CnI, VpeI, VpoI)$ and erroneous configuration data $(CpI^{'}, CnI^{'}, VpoI^{'}, VpeI^{'})$ using the equation 7.7 to equation 7.10. Hence in IMMC, only $(dp + dn + he + ho)$ bits are required to detect error in a configuration frame that is quite small compared to the bits required to correct an error in a configuration frame using MMC. In Figure 7.9, line with the same color

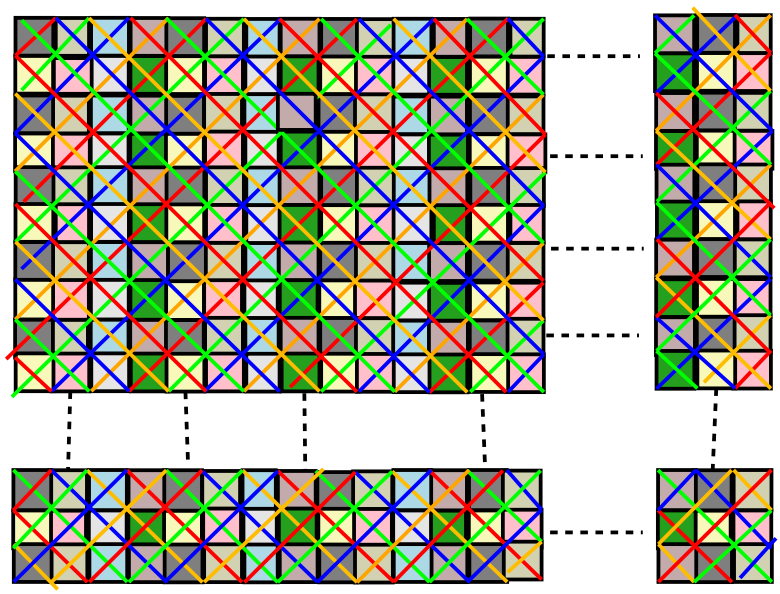

Figure 7.9: Detection using Interleaved MMC

along both positive and negative diagonal with unit slope generate a single parity bit. Similarly, bits in the column with the same color generate single parity bit. Here, interleaving distance along both diagonal and vertical direction is taken as four (*i.e* values of dp, dn, he, ho are equal to four) but they can be changed to any value. Mismatch in the value of interleaved parity bits generated from original configuration data and erroneous configuration data indicate the presence of errors in the configuration frame.

7.4 Error Detection and correction using EVENODD coding

7.4.1 Overview of EVENODD coding

EVENODD is a simple parity based erasure code originally developed to protect data in double disk failure in a Redundant Array of Independent Disks (RAID) architecture as described in [164]. In RAID architecture, data are distributed across the different disks in such a way that provides improved fault tolerance, increase storage capacity and improved overall system performance. A similar concept is used here to protect data in the configuration memory of SRAM-based

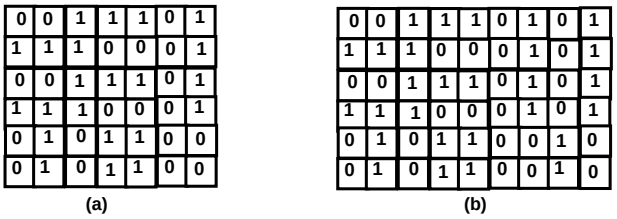

Figure 7.10: Example of EVENODD encoding taking $R = 7$

FPGA. EVENODD is a cyclic code and parity check matrix (H) of EVENODD code is described using equation 7.11. Here, R is the number of columns and (R-1) is the number of rows of matrix that will be used for encoding and β is an element of Galois field [164]. Using H we can derive that the d_{min} of the EVENODD code is 3 [164].

$$H = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 & 1 & 0 \\ 1 & \beta & \beta^2 & \dots & \beta^{R-1} & 0 & 1 \end{bmatrix} \tag{7.11}$$

Decoding of EVENODD code in configuration memory of FPGA can be done either in two steps (where detection and correction are separate process) or in a single step (where detection and correction will be done simultaneously). Here, we will discuss both of these two cases but in both cases encoding process is same which is described using a simple example as shown in Figure 7.10. The input of the encoding process is a matrix A of size $(R-1)\times R$ where R should be a prime number, but it will not be a constraint. If R is not a prime number, we will take next prime number following this arbitrary number assume that all information bits in the added row or columns are 0. In Figure 7.10 (a), a matrix of size 6×7 (taking the value of R as 7) is given as input to the encoding process and Figure 7.10 (b) shows the matrix of size 6×9 after encoding with last two columns as redundant data, generated during encoding process. At the first step of the encoding process syndromes are calculated using equation 7.12

$$S = A(5,1) \oplus A(4,2) \oplus A(3,3) \oplus A(2,4) \oplus A(1,5) \oplus A(0,6); \tag{7.12}$$

Then redundant data in 7^{th} and 8^{th} column can be calculated using equation 7.13

$$\begin{aligned} A(v,7) &= \bigoplus_{u=0}^{6} A(v,u) \\ A(v,8) &= S \oplus \bigoplus_{u=0}^{6} A(((u-v) mod 7), u) \forall u = 0 \; to \; 6, v = 0 \; to \; 5 \end{aligned} \tag{7.13}$$

As d_{min} of EVENODD code is three, external detection technique can detect maximum two erroneous columns of matrix A[(R-1),(R+2)] using algorithm 10 otherwise it can correct single erroneous column of matrix A[(R-1),(R+2)]. Details of encoding and decoding of EVENODD code can be seen from [164].

7.4.2 Recovery based on EVENODD code

Here we will discuss how previously described EVENODD code is applied to correct clustered error or multiple adjacent erroneous bits in the configuration memory of FPGA. Clustered error may affect multiple configuration frames or a large portion of a single configuration frame. As the first step of encoding process, configuration memory is partitioned into multiple regions (described in section 7.5), and it is assumed that partitioned regions will be activated in a periodic interval. Hence, at any instant, only configuration frames of a particular region will be available in the error detection and correction block (EDACB) unlike the other error correction schemes [97] where data of the full configuration memory is available to EDACB at a particular instance. After reading the idle partial region into EDACB, the available configuration frames are divided into several groups as shown in Figure 7.11. Here the group formation is quite different from that of the group formation technique described in [97]. In [97], authors formed the group only in horizontal direction, but here, the groups are formed along both vertical and horizontal directions.

Figure 7.11 describes the group formation along both horizontal and vertical directions and EVENODD coding is used to generate the redundant data to protect the configuration frames using equation 7.14 to 7.19. In Figure 7.11, we have assumed each horizontal group contains M configuration frames and each vertical group contains N configuration frames. NC indicates number of column in each configuration frame and NR indicates number of row in each configuration frame. R_{h1} and R_{h2} indicate redundant frames for each horizontal group and R_{v1} and R_{v2} indicate redundant frames for each vertical group. $A_{i,j}$ indicates configuration frame in i^{th} horizontal and j^{th} vertical group.

$$S_{hk}(i) = \bigoplus_{t_1=1}^{M-1} A_{k,t_1}(M-1-t_1, i) \forall i = 0 \; to \; (NC-1), k = 0 \; to \; (N-1) \quad (7.14)$$

Algorithm 10 Decoding Algorithm for EVENODD

Require: Detected column(i, j),Erroneous A[(R-1),(R+2)]
Ensure: Corrected A[(R-1),(R+2)];
1: A[$(R-1)$,z]=0 $\forall z = 0 \rightarrow (R-1)$
2: **if** $((i = R)\&\&(j = R+1))$ **then**
3: Decoding process will be similar to Encoding Process;
4: **end if**
5: **if** $(i < R)\&\&(j = R+1)$ **then**
6: First i^{th} column will be corrected using equations in line no. 8 in Encoding Algorithm and then j^{th} column can be recovered using similar equations in encoding process.
7: **end if**
8: **if** $(i < R)\&\&(j = R)$ **then**
9: i^{th} column will be corrected as follows and then j^{th} column will be corrected equations in line no. 8 in Encoding Algorithm
10: $t1 = (i-1) \mod R$;
11: **for** $l = 0 \rightarrow (R-1)$ **do**
12: $t2 = (i-l-1) \mod R$;$S = S \oplus A(t2, l)$;
13: **end for**
14: $S = S \oplus A(t1,(R+1))$;
15: **for** $k = 0 \rightarrow (R-2)$ **do**
16: **for** $l = 0 \rightarrow (R-1)$ **do**
17: **if** $l \neq i$ **then**
18: $t3 = (k+i-l) \mod R$;$A(k,i) = A(k,i) \oplus A(t3,l)$;
19: **end if**
20: **end for**
21: $A(k,i) = S \oplus A(k,i) \oplus A(t1,(R+1))$;
22: **end for**
23: **if** $(i < R)\&\&(j < R)$ **then**
24: **for** $l = 0 \rightarrow$(m-2) **do**
25: $S = A(l,m) \oplus A(l,m+1)$;
26: **end for**
27: **for** $u = 0 \rightarrow (R-1)$ **do**
28: **for** $w = 0 \rightarrow R$ **do**
29: $SL(u) = SL(u) \oplus A(u,w)$;
30: **end for**
31: $SU(u) = S \oplus A(u,R+1)$;
32: **for** $t = 0 \rightarrow R$ **do**
33: **if** $(t \neq i)\&\&(t \neq j)$ **then**
34: $v = (u-t) \mod R$;$SU(u) = SU(u) \oplus A(v,t)$;
35: **end if**
36: **end for**
37: **end for**
38: **end if**

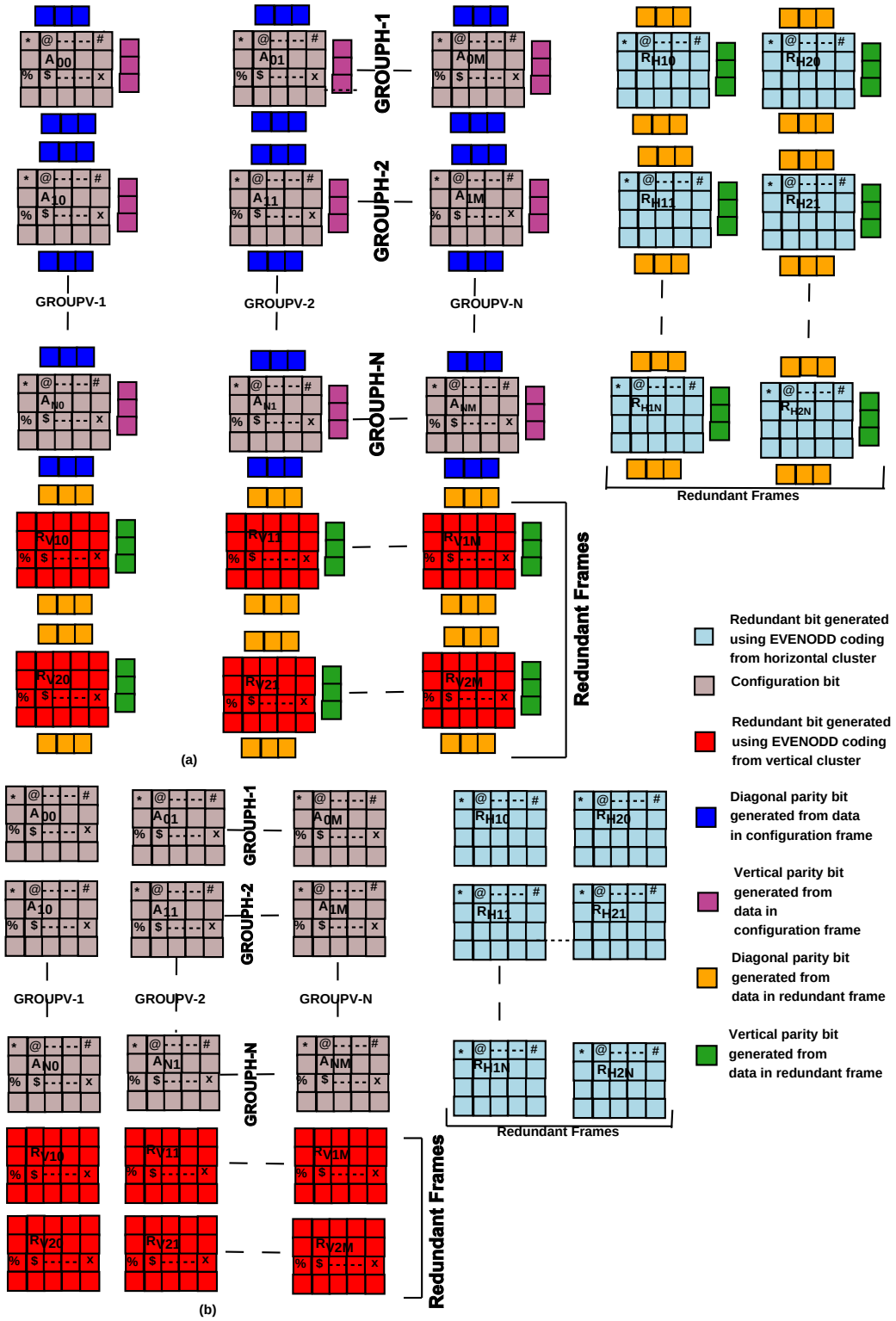

Figure 7.11: Grouping of configuration frames for decoding using EVENODD code: (a) when EDAC is done separately(b) when EDAC is done simultaneously

```
39:     if k ≠(R − 1) then
40:         k =(i − j − 1)  mod R
41:         while k ≠(R − 1) do
42:             v1 =(j + k)  mod R;v2 =(k + j − i)  mod R;
43:             A(k, j)=SL(v1)⊕A(v2, i);A(k, i)=SU(k)⊕A(k, j);
44:             k =(k + i − j)  mod R;
45:         end while
46:     end if
47: end if
```

$$S_{vl}(j) = \bigotimes_{t_2=1}^{N-1} A_{t_2,l}(N-1-t_2, j) \forall j = 0\ to\ (NC-1), l = 0\ to\ (M-1) \quad (7.15)$$

$$R_{h1k}(u,v) = \bigotimes_{t_1=0}^{M-1} A_{k,t_1}(u,v) \forall k = 0\,to\,(N-1), u = 0\,to\,(NR-1), v = 0\,to\,(NC-1) \quad (7.16)$$

$$R_{v1k}(u,v) = \bigotimes_{t_2=0}^{N-1} A_{t_2,k}(u,v) \forall k = 0\,to\,(N-1), u = 0\,to\,(NR-1), v = 0\,to\,(NC-1) \quad (7.17)$$

$$R_{h2k}(u,i) = S_{hk}(i) \oplus \bigotimes_{t_1=0}^{M-1} A_{k,t_1}(((u-t_1) mod M), i) \forall k = 0\ to\ (N-1), \quad u = 0\ to\ (NR-1), i = 0\ to\ (NC-1) \quad (7.18)$$

$$R_{v2l}(u,i) = S_{vl}(i) \oplus \bigotimes_{t_2=0}^{N-1} A_{t_2,l}(((u-t_2) mod N), i) \forall l = 0\ to\ (N-1), \quad u = 0\ to\ (NR-1), i = 0\ to\ (NC-1) \quad (7.19)$$

Columns marked with the same symbols of different frames within a group are encoded by EVENODD coding and generate the two redundant columns marked with the same symbol for two redundant frames. As for example in Figure 7.11, column marked with '#' in all configuration frames are encoded by EVENODD coding and generate column marked with '#' in two redundant frames for N horizontal groups. Similarly, in the vertical direction, columns marked with '$' in all configuration frames are encoded using EVENODD code and then generate columns marked with '$' in two redundant frames for M vertical groups. So for

each group in horizontal and vertical directions we get two redundant frames. Correction algorithm mitigate errors not only in configuration frames but also in redundant frames.

Decoding can be done either in two steps where EDAC are separate process as shown in Figure 7.11(a) or in single step where EDAC is done simultaneously as shown in Figure 7.11(b). Here we are going to describe eror correction using EVENODD coding when IMMC is used to detect erroneous frames. At the beginning of decoding process configuration frames along with redundant frames from a idle partitioned region will read back and the error correction will be done according to the following cases:

1. If errors are detected in either of the redundant frames or in both of the redundant frames in each group they can be corrected using the encoding process itself (using equations 7.14 to 7.19)

2. If any configuration frame either in horizontal group or in vertical group is erroneous but no redundant frames are erroneous they can be corrected using equations 7.14 to 7.19.

3. If error is detected in A_{ij}^{th} configuration frame, R_{h1i}^{th} redundant frame of i^{th} horizontal group and R_{v1j}^{th} redundant frame of j^{th} vertical group then errors in A_{ij} can corrected using the equations 7.20 and 7.21

$$S_h(k) = R_{h2i}(((j-1) mod M), k) \oplus \bigoplus_{l=0}^{M-1} A_{i,l}(((j-l-1) mod M), k)$$
$$\forall k = 0 \; to \; NC-1 \tag{7.20}$$

$$A_{i,j}(t,k) = S_h(k) \oplus R_{h2i}(((j-1) mod M), k) \oplus \bigoplus_{\substack{l=0 \\ l \neq j}}^{(M-1)} A_{i,l}(((t+j-l) mod M), k)$$
$$\forall t = 0 \; to \; (NR-2), k = 0 \; to \; (M-1) \tag{7.21}$$

   Then error in redundant frames R_{h1i} and R_{v1j} can be corrected using the equation 7.16 and equation 7.17 respectively.

4. If error is detected in A_{ij}^{th} configuration frame, R_{h2i}^{th} redundant frame of

i^{th} horizontal group and R^{th}_{v2j} redundant frame of j^{th} vertical group then errors in configuration frame can be corrected using the equation 7.16 and equation 7.17 and errors in redundant frames can be corrected using the equation 7.18 and equation 7.19.

5. Error in configuration frames A_{pi}, A_{pj}, A_{qi} and A_{qj} can be corrected using the equations 7.22 to 7.27

$$S_p(k) = \left(\bigotimes_{l=0}^{(M-2)} R_{h1p}(l,k)\right) \oplus \left(\bigotimes_{l=0}^{(M-2)} R_{h2p}(l,k)\right) \quad \forall k = 0 \; to \; (M-1) \tag{7.22}$$

$$S_p^{upper}(b,k) = \bigotimes_{\substack{l=0 \\ l \neq (i,j)}}^{M-1} A_{p,l}(b,k) \forall \;\; k = 0 \; to \; (NC-1), b = 0 \; to \; (NR-1) \tag{7.23}$$

$$S_p^{lower}(b,k) = S_p(k) \oplus R_{h2p}(b,k) \oplus \bigotimes_{\substack{l=0 \\ l \neq (i,j)}}^{(M-1)} A_{p,l}(((b-l) mod M), k) \quad \forall k = 0 \; to \; (NC-1), b = 0 \; to \; (NR-1) \tag{7.24}$$

Now the errors in configuration frames A_{pi}, A_{pj} can be corrected using the equations 7.25 to 7.27.

$$Y(k) = ((i-j-1) mod M) \tag{7.25}$$

$$A_{p,j}(Y(k),k) = S_p^{lower}(((j+Y(k)) mod M), k) \oplus A_{p,i}(((Y(p)+j-i) mod M), k) \tag{7.26}$$

$$A_{p,i}(Y(k),k) = S_p^{upper}(Y(k),k) \oplus A_{p,j}(Y(k),k) \tag{7.27}$$

Update $Y(k)$= $((Y(k)-j+i)$ mod M). If $Y(k)$= M-1 then stops otherwise go to equation 7.26.

Increase $k = k + 1$ and go to equation 7.25 until k = (NC-1).

Then error in configuration frames A_{qi} and A_{qj} can be corrected using equation 7.17 and equation 7.19.

During the decoding process, the erroneous frames will be detected by IMMC in the first step and then erroneous configuration frames and redundant frames within a group are corrected according to different cases described above. Using EVENODD decoding, maximum two frames (including both configuration and

redundant frame) can be corrected in each vertical and horizontal group whereas the technique proposed in [97] can correct single frame in a group. Though in proposed EVENODD coding overhead is higher compared to the technique proposed in [97]. Authors in [97] suggested that configuration frames in the same group should be physically far apart so that multiple erroneous frames are not present in the same group. This is possible when data from all configuration frames are available to EDACB but in our case, within EDACB, only configuration frames from a particular region of configuration memory are available. Hence, the probability of occurrence of multiple erroneous frames in a group is high in our proposed models.

When IMMC will not be used for detection, one full configuration frame or multiple configuration frames having errors in different portions within a group can be corrected using EVENODD decoding algorithm as described in section 4.2 of [164]. Obvious advantage of this method is that no prior information is required about the erroneous frames in a group before error correction. Sometimes, different portions of multiple configuration frames (more than two) become erroneous within the same group. Then error correction scheme in Figure 7.11(a) and in [97] fails to correct these frames but error correction technique described in Figure 7.11(b) can correct these type of errors. As the SBUs and MBUs are random in nature, there is very low probability that same region in the multiple configuration frames within the same group will be affected simultaneously. As for example, it can be assumed that there is very low probability that columns marked with '#' in the multiple configuration frames in the same horizontal group will be affected by MBUs at a time. Similarly, there is also very low probability that columns marked with '$' will be affected by MBUs in the multiple configuration frames in a vertical group.

MBUs in configuration memory sometimes may affect redundant bits used to detect erroneous frames in IMMC. This may affect error correction functionalities described in Figure 7.11(a) because erroneous redundant bits can not detect erroneous configuration frames properly. Hence, to alleviate this effect, this module is protected by TMR as by the authors in [97]. Since the size of the redundant bits used for error detection is small, area overhead imposed by TMR implementation is not significant. As there is no separate detection technique is used, no TMR

Table 7.1: Summary of error detecting and correcting codes used in this paper

Name of the code	Component code	Error Detection and Correction Capability
MMC	Parity code with majority voting	Error Detection is 100% and error correction capability is 100% for errors up to 3 bit. For more than three bit errors, error correction depends on the position of the erroneous bits
MC [93]	Hamming code and parity code	d_{min} of HC is 3. Hamming code and parity code help to detect and correct two bit errors with 100% efficiency. For more than two bit errors, error detection and correction depends on the position of erroneous bits.
IMMC	MMC with interleaving	Used only for error detection. Error detection varies with interleaving depth as shown in Table 7.2
HPC [89]	Hamming code	d_{min} of hamming code is 3. Error detection and correction depends on the position of erroneous bits
EVENODD [164]	N/A	d_{min} of EVENODD code is 3. Without external detection EVENODD can correct all the errors in the single column of a matrix. With external detection it can correct any two columns of a matrix.

implementation is required to protect redundant bits for error detection in the procedure described in Figure 7.11(b). Table 7.1 illustrates summery of EDAC capability of different error correcting codes used in this chapter

7.5 Dynamic Priority Based Algorithm for download manager

Application hardware, placed into the configuration area of an FPGA chip consists of few sub-components, which may be stated as standard IPs or custom IPs. Here each component will be placed into a partitioned area of the FPGA. Proposed algorithm 11 calculates a dynamic priority for each IP which decides the sequence of fault scanning process of all IPs. The IP with highest priority would be scanned first and the IP with least priority would be scanned last.

Lets us assume in our application available L number of IPs execute periodic process. The read back time, scanning time, partial bit file downloading time, IP active time and IP idle time of i^{th} IP (IP_i) are R_i, S_i, D_i, E_i and I_i respectively. The period of the i^{th} IP is T_i=E_i+I_i where i=1 to L. A_i is the algorithm execution time and t is the period of the input clock which drives the application. W_u and W_f are user defined parameter. Steps of the proposed algorithm as follows:

1. IP of a partitioned region must remain idle during the downloading of bit file for that PR region. The system will be uninterrupted if bit file downloading

Algorithm 11 Algorithm of download manager

Require: clk, w_u, w_f $busy_i$, T_i where $i = 1 \rightarrow L$;
Ensure: FP_{max};

```
for i = 1 → L do
    Triggered the for loop at rising edge of busy_i
    St_i = (E_i+I_i)/t
    if (rising_edge(clk)) then
        if St_i = 0 then
            St_i = (E_i+I_i)/t
        else
            St_i = St_i − 1;
        end if
        if ( then(R_i + S_i + A_i + D_i)<= St_i * t)
            P_i = St_i−((A_i+S_i+D_i)/t)
        end if
    end if
end for
FP_i =[w_u×(1/P_i)+w_f×(E_i/t)];
Find maximum among allFP_i;
```

time D_i can be fitted into IDLE time I_i for i^{th} IP. Hence, the primary assumption taken for the proposed approach is $I_i \geq D_i$.

2. A status register St_i allocated for each IP measures the period from current time until its next transition from IDLE state to EXECUTION state occurs. At the beginning, St_i is initiated by $\frac{E_i+I_i}{t}$ and it is decremented by one in every rising edge of clock. Whenever St_i becomes 0, it is initiated by $\frac{E_i+I_i}{t}$ again. The timing diagram of St_i is shown in Figure 7.12.

3. In the next step priority P_i will be calculated by subtracting $\frac{A_i+S_i+D_i}{t}$ from St_i. Here $(S_i + D_i)$ is fault scanning time which is denoted by FS_i.

4. The IP with large execution time has more share in the overall hardware process. If it is affected by clustered error, then there is a high probability that system will give erroneous result for large time compared to the situation where IP with small execution time is affected by clustered error. So designer will always try to rectify the error with high priority for IPs with large execution time compared to IPs with small execution time. To

satisfy this constraint, $I_i \geq D_i$ may fail which suspends normal system operation. Hence, final priority can be defined as $FP_i = [w_u \times (\frac{1}{P_i}) + w_f \times (\frac{E_i}{t})]$ where $w_u + w_f = 1$. Here P_i and FP_i will be updated in parallel. If $w_u = 1$ and $w_f = 0$ then user set the priority fully based on the logic resources of IP. In this case, the system will never stop, but the probability that system will give the wrong result is high. On the other hand, if $w_u = 0$ and $w_f = 1$ then user set the priority fully based on the execution time of IP or criticality of the IP. In this case the probability, that system will give the wrong result is less but the system may halt. Users can choose the value of w_u and w_f in between 0 and 1 as per their requirement.

5. The value of FP_i decides which PR region will be scanned first. In a time instant, the IP having higher FP_i is the most suitable one to be scanned by Internal Configuration Access Port (ICAP). The IPs with a lower value of P_i or higher FS_i, have large hardware resources compared to IPs with a higher value of P_i. In future, the probability to find large time slot in St_i for accommodating such bulky FS_i is less where as lighter IPs with lesser FS_i can be fitted more frequently in St_i. On the other hand higher value of $\frac{E_i}{t}$ means IP has higher execution time.

6. During the scanning and error correction (if the fault occurs) on a chosen IP_i, FP_i will be monitored by the remaining $(L-1)$ IPs and again the IP with highest FP_i will be chosen for error detection and correction.

7. In a trivial case where no IPs are found which has less FS_i than its St_i, then St_i will be compared with $2 \times (\frac{E_i + I_i}{t})$.

It is to be noted that algorithm 11 is designed in hardware in such a way that line 5 to 9, line 11 and line 15 will be executed in parallel. In Figure 7.12, IP_1 has execution time E_1, Idle time I_1 and status register St_1, IP_2 has execution time E_2, Idle time I_2 and status register St_2, similarly n^{th} IP has execution time E_n, Idle time I_n and status register St_n. In blue vertical line at the third rising edge of clock the value of St_1, St_2,St_n are 3, 3 and 5 respectively. If t is the period of the clock, for IP_1, IP_2 and IP_n the download manager will check the relation between $R_1 + S_1 + A_1 + D_1$ with $3 \times t$, $R_2 + S_2 + A_2 + D_2$ with $3 \times t$. for IP_2 and

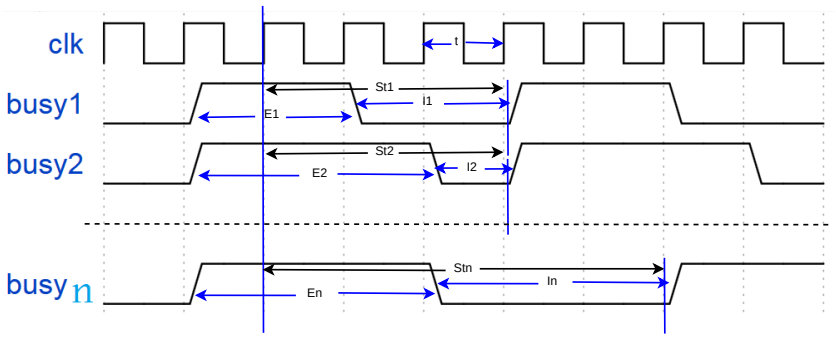

Figure 7.12: Timing diagram of St_i

$R_n + S_n + A_n + D_n$ with $5 \times t$ respectively. According to the line number 11 and 15 of algorithm 11, P_i and FP_i will be calculated. The IP with maximum FP_i will be scanned first.

7.6 Hardware Implementation and its workflow

Hardware implementation of the proposed EDAC schemes for the configuration memory of FPGA is shown in Figure 7.13. The design can be partitioned into two parts: static area and configuration area. The static area consists of Hardware scheduler block, Master ICAP controller, Slave interface controller, EDACB, Hardware ICAP (HWICAP) and ICAP interface. Error in the static area will be detected by CRC. Configuration area consists of the application, built up by multiple PR regions which is to be configured during the runtime as shown in Figure 7.13. After power is on, bit file will be downloaded from a secondary memory into the configuration memory through ICAP port, and then there will be no communication between secondary memory and configuration memory. Slave interface block gets the controlling information from the master. Master sends multiple informations namely ICAP_start, bit addresses, and bit length. ICAP acknowledges master using ICAP_done port while dynamic configuration process is done. The HWICAP is an interface to the ICAP. Redundant data for EDAC is stored locally into block RAM of FPGA devices. During the bit file downloading TXFIFO inside the HWICAP reads configuration data from Block RAM and send it to the configuration memory. Similarly, during read back RXFIFO will receive configuration data from configuration memory and send it to block RAM inside EDACB. Total time spent by the IPs into the application can be divided

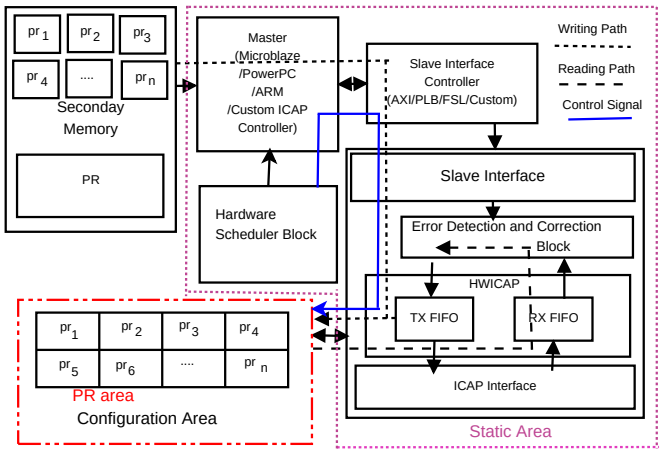

Figure 7.13: Hardware implementation of the proposed Models

into two parts: execution time and dead time. During the dead time of the IPs, configuration frames of IPs will be read back into the EDACB through ICAP port and EDACB fixes the errors. After finishing the error correction process EDACB will write back the configuration frame of the IP to the configuration memory through ICAP if and only if error is detected in any configuration frame of the IP. This partitioning and selective writing process reduces overall reconfiguration time by 2.2 ms on average (discuss further in section 7.7). Otherwise, the complete bit file downloading process may take more reconfiguration time. Improvement of reconfiguration time (δRT) can be calculated using the mathematical equation 7.28.

$$\delta RT = \sum R + S + D + A - \sum_{i=1}^{L} R_i + S_i + D_i + A_i \tag{7.28}$$

Meaning of each symbols in equation 7.28 are already explained in section 7.5. Here scanning time is equivalent to error detection time and algorithm execution time is equivalent to error correction time. If PR is not used total configuration time will be read back and error correction will be executed on full configuration time. On the other hand after using the PR if error is not detected in a region algorithm execution time will be zero for that region and configuration frame of this region need not to be written back into the configuration memory. This will reduce overall reconfiguration time after using partial reconfiguration.

The workflow of the proposed design is described by the following steps and shown in a flowchart in Figure 7.14

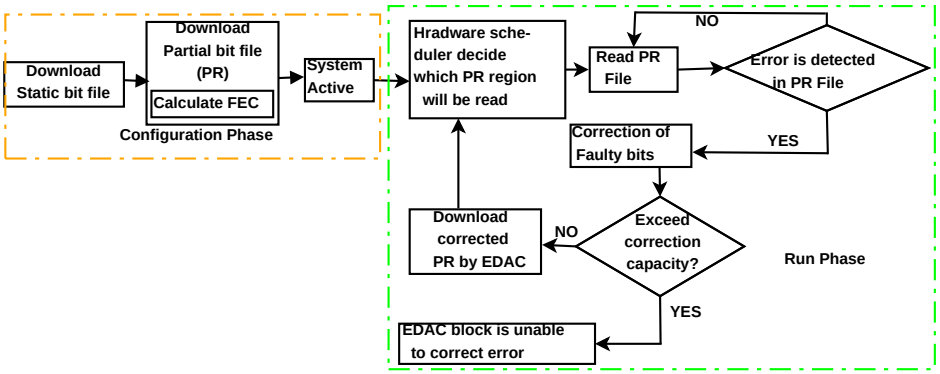

Figure 7.14: Workflow of the proposed error correcting models

Step1: Only the bit file for the static part will be downloaded in the configuration area of the FPGA from the secondary memory. The partial bit file of the whole application (PR) is stored in the secondary memory along with sub-component bit files. Here $PR = pr_1 + pr_2 + ...pr_i + ... + pr_L$.

Step2: The partial bit file is now downloaded into the partitioned region through the proposed ICAP block. During this pass, EDAC code inside the ICAP block calculates the forward error correction (FEC) field. Once downloading of static and partial bits are completed the whole system becomes functional.

Step3: At the onset of the error correction process, based on the decision of hardware scheduler, ICAP starts to read configuration frames from a particular PR region (pr_i) in the configuration memory.

Step4: When error is detected in a configuration frame of pr_i then error correction will be performed only on the erroneous configuration frame. After completion of error correction all configuration frames pr_i will be downloaded. If error is not detected in any configuration frame in the pr_i then partial bit file of that pr_i will not be downloaded.

7.7 Result and Performance Analysis

Proposed error correcting models are implemented on the Xilinx Kintex7 board using Vivado platform and VHDL for design entry. Different application designs are used to generate the bit file. We have tested our design using behavioral simulations. To validate error detection and correction capability of the different

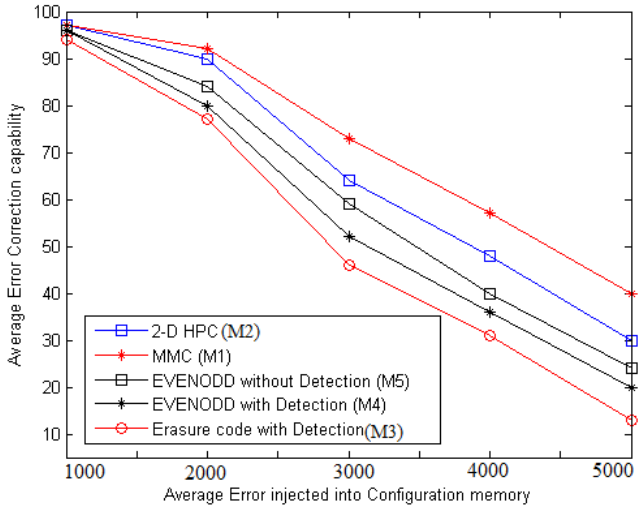

Figure 7.15: Average error correction capability of different error correcting models when single or small number of adjacent bits are affected by random error

error correcting models we have performed fault injection experiment in the configuration memory of Xilinx Kintex-7 KC325T device. Different number of faults starting from 1000 to 5000 are injected randomly into the configuration memory through ICAP interface. Here we have proposed models for correcting both single bit error and multi-bit errors (clustered error), occurred due to the injected fault.

7.7.1 Comparison With existing Error correcting models

Here we have considered two kinds of error: single bit errors and multi-bit clustered errors. To correct errors in the configuration memory due to SBUs and MBUs we have proposed error correcting models based on simple parity equation (MMC in section 7.2) and erasure code (EVENODD with error detection technique, EVENODD without any error detection in section 7.4) respectively. We have compared their performances with the two existing models in [97]. In order to illustrate the performances of different models properly, each of them are designated with different symbols as follows: MMC with M1, model proposed in [89] with M2, model proposed in [97] with M3, model based on EVENODD code with separate detection technique as M4 and model based on EVENODD code without separate detection technique by M5.

HPC in model M2 can be used to correct data affected by multi-bit errors but

when multi-bit errors occur along both row and column in a matrix, HPC may fail to recover the data. In [89], authors give some examples of non-repairable errors. All of these non-repairable errors can be detected, and some of them can also corrected by proposed MMC as described in Figure 7.5(b) of section 7.2. MMC in M1 as well as HPC in M2 show good performance against single bit and small size clustered error compared to model M3, M4 and M5 as shown in Figure 7.15. Here a different number of errors (varying from 1000 to 5000) are injected randomly into the configuration memory such that they will affect either single bit or a small number of adjacent bits.

In model M3, all configuration frames are divided into multiple groups and for each group maximum one configuration frame can be corrected. When errors affect randomly, they may be spread into multiple configuration frames which are not physically adjacent. The main criterion in group formation in model M3 is that configuration frames within the same group must be physically far apart. Model M3 can not correct multiple erroneous configuration frames within the same group. In model M4, maximum two configuration frames can be corrected per group as described in Figure 7.11(a). Model M5 can correct errors in either one full configuration frame or multiple configuration frames having errors at different regions in multiple configuration frames along both horizontal and vertical groups as described in Figure 7.11(b). As for example, if there is an error in the first column in a configuration frame in a group then model M5 can correct it if there is no other configuration frame within the same horizontal and vertical group that has an error in the first column. This stringent rule in model M3, M4 and M5 is not good for single bit random error correction. On the other hand model M1 and M2 correct the error using the windowing technique in each configuration frame individually. Random single bit error or small size clustered error within a configuration frame can be easily corrected with the different window size. From Figure 7.15 it can be observed that with the increase of the number of injected random errors M3, M4 and M5 show more poor performance compared to M1 and M2 as it increases occurrence probability of errors at same regions in multiple configuration frames within the group. In M1 and M2, average error correction capability can be improved by increasing number of iterations as described in section 7.2 and for each iteration M1 gives better result compared to model

M2 as shown in Figure 7.16. From Figure 7.16 it can be observed that when the

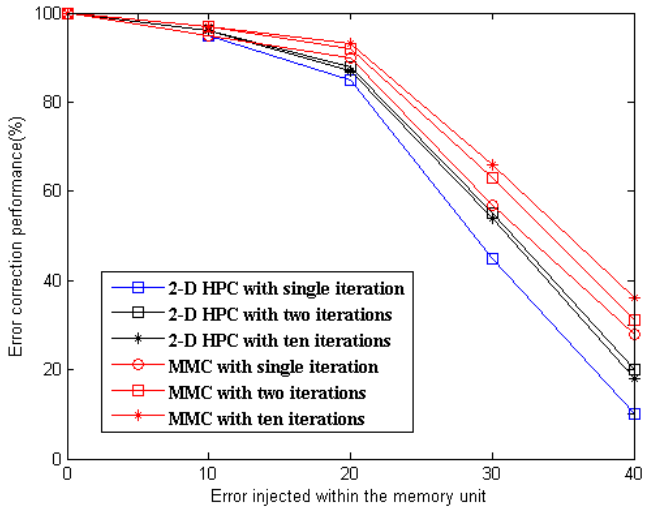

Figure 7.16: Comparison of error correction capability between MMC and HPC

number of injected error is high, HPC gives poor result with the increase in the number of iterations (HPC with ten iterations give poor performance compare to HPC with two iterations). This is due to the fact that SECDED Hamming code gives the wrong result when more than two bits are erroneous in a single Hamming coded data frame (*i.e* Hamming code may flip a bit without any error). Figure 7.17 compares memory overhead due to the parity bits, between the proposed models M1 and M2 for different memory sizes. It clearly shows that the overhead due to the parity bits is less in MMC code compared to HPC. Hence, both in terms of overhead and error correction coverage model M1 outperforms M2. It can be concluded that out of the five models, MMC gives the best result against single bit error and small size clustered random errors.

Model M3, M4 and M5 show good performance compared to M1 and M2 for clustered error correction (*i.e* when a large number of adjacent bits are affected by errors as described in Figure 7.18). In model M3, groups are formed taking all configuration frames at a time so it is possible to form a group taking configuration frames which are physically far apart. In M4 and M5 groups are formed taking configuration frames from a particular PR so there is a high probability of occurrence of multiple erroneous frames in the same group that gives poor performance in model M3. To increase the error correction coverage in model M4 and

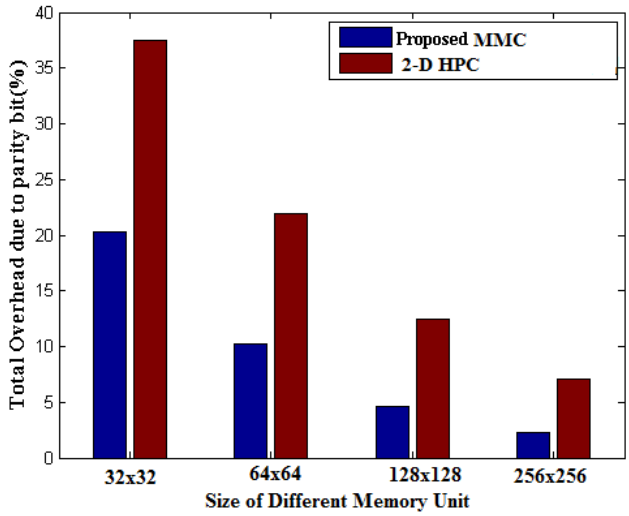

Figure 7.17: Comparison between HPC and MMC due to redundant bits

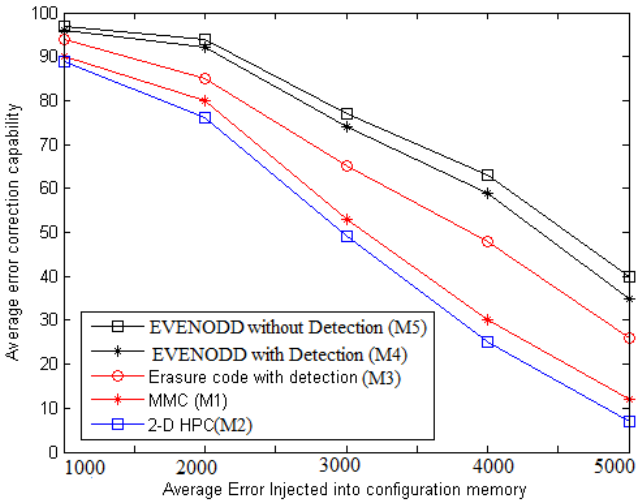

Figure 7.18: Average error correction capability of different error correcting models for clustered error

M5, groups are formed along both horizontal and vertical directions. In model M4, separate detection technique is used to detect erroneous frames. Detection capability of IMMC can be varied by changing depth of interleaving as shown in Table 7.2. Meaning of *he*,*ho*,*dn* and *dp* are mentioned in section 7.3. It can be seen from Table 7.2 minimum 14 bits are required for 100% detection coverage. In M4 maximum, two erroneous frames along both horizontal and the vertical group can be corrected at a time using EVENODD coding so M4 gives better performance compared to M3 as shown in Figure 7.18. In M5, EVENODD code

Table 7.2: Detection capability of IMMC with different interleaving depth

		he=1			*he*=3			*he*=5		
		ho=1	*ho*=3	*ho*=5	*ho*=1	*ho*=3	*ho*=5	*ho*=1	*ho*=3	*ho*=5
dp=1	*dn*=1	57.12%	62.65%	88.87%	59.90%	62.94%	88.71%	88.99%	89.65%	97.95%
	dn=3	61.79%	62.26%	88.82%	62.46%	63.19%	88.9%	89.57%	89.66%	98.07%
	dn=5	95.7%	96.15%	98.93%	96.38%	96.18%	99.01%	98.83%	99.04%	99.77%
dp=3	*dn*=1	61.29%	62.96%	89.42%	63.38%	63.48%	88.99%	89.15%	89.71%	97.83%
	dn=3	64.01%	63.63%	89.55%	62.62%	63.24%	89.45%	89.65%	89.55%	98.18%
	dn=5	96.20%	96.34%	99.03%	96.20%	95.98%	98.70%	99.05%	99.06%	**100%**
dp=5	*dn*=1	96.25%	96%	98.82%	96.04%	96.38%	98.82%	98.98%	98.97%	99.84%
	dn=3	96.02%	96.16%	98.78%	96.21%	96.09%	98.86%	98.92%	99.24%	99.73%
	dn=5	99.51%	99.62%	99.84%	**100%**	99.61%	**100%**	99.84%	**100%**	**100%**

can correct error in a single configuration frame or multiple configuration frames having errors in different regions, and no separate detection technique is used to detect erroneous frames. Practically, there is a high probability that more than two configuration frames in both horizontal and vertical groups may be affected partially by clustered error instead of a whole configuration frame. As error is random is nature it can be assumed that there is high probability of clustered error in different location on multiple configuration frames. Hence, Model M5 gives better result compared to M3 and M4 in presence of large size clustered error on multiple configuration frames in same group as shown in Figure 7.18. Figure 7.19 illustrates the variation of error correction coverage and ratio of redundant bits to the configuration bits with different number of horizontal and vertical groups for model M3, M4 and M5. Here, number on each bar indicates the ratio of redundant bits to the configuration bits. As the configuration bits for a particular FPGA device is fixed, variation of the ratio of redundant bits to configuration bit is due to the variation of redundant bits only. For Figure 7.19 the first term along x-axis indicates number of horizontal group and second term indicates number of vertical group. It is seen from the Figure 7.19 with the increase in the number of horizontal group error correction coverage and redundant bits gradually reduces up to certain value of horizontal group and then again increases with horizontal group for model M4 and M5. Whereas redundant bits and error correction coverage steadily increase with number of horizontal group for model M3. M4 and M5 always outperform M3 in terms of error correction coverage but for small number of horizontal group overhead is very high for M4 and M5 compared to M3. If value of horizontal groups and vertical groups are close to each other

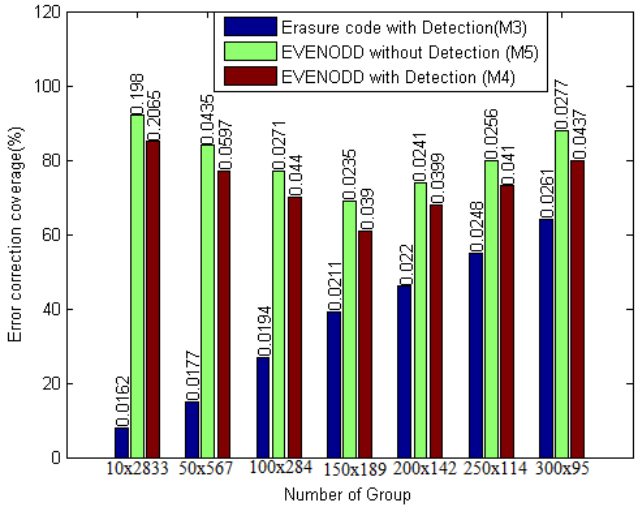

Figure 7.19: Comparison between our proposed EVENODD model and model proposed in [97]for different size of group

Table 7.3: Power Consumption by different models

Model	Power Consumption Without PR(mw)	Power Consumption With PR(mw)
M1	110	110
M3 [97]	150	N/A
M4	145	118
M5	130	112

redundant bits of M5 and M3 becomes almost equal. It is indicated by results of 150 horizontal groups and 189 vertical groups in Figure 7.19. It is clearly seen from Figure 7.19, M5 gives best performance in terms of redundant bits and error correction capability compare to M3 and M4 if number of horizontal and vertical groups are almost same. Users can select any number of horizontal and vertical group as per the requirement sacrificing in terms of redundant bits. It can be concluded from the above discussions that models M3, M4, M5 show good performance compared to M1 and M2 when configuration memory is affected by large size clustered error and M5 is the best candidate among M3, M4 and M5 in terms of both redundant bits and error correction coverage.

We have also measured the power consumption of our models using Xilinx Xpower tool as described in Table 7.3. As for model M2 [89] authors did not give power information, so we have not included M2 in Table 7.3. From the Table 7.3, it can be observed that M1 consumes the lowest power both in PR mode and

Table 7.4: Fault recovery time for different models

Models	Average MTTR without PR(ms)			Average MTTR with PR(ms)		
	Download and readback time	Error Detection Time	Error Correction Time	Download and readback time	Error Detection Time	Error Correction Time
M1	28.8	0	9.341	26.9	0	9.341
M2	28.8	0	9.341	N/A	N/A	N/A
M3	28.8	6.227	0.298	N/A	N/A	N/A
M4	28.8	6.227	0.298	26.9	6.227	0.395
M5	28.8	0	0.298	26.9	0	0.395

without PR mode among all the models discussed in this paper. This is due to the fact that in M1, EDAC is done simultaneously within each configuration frame. As EDAC is separated in M3 and M4, both of them consume almost same power without PR mode. M5 consumes less power compared to M3 and M4 as in M5 there is no separate detection process. With PR models, M4 and M5 consume less power compared to the situation where PR is not used. Overall it can be concluded that use of partial reconfiguration reduces power consumption which justify its usage.

7.7.2 System Recovery Time

Based on the decision of download manager, configuration frames from different PR regions will be read back into EDACB. Kintex FPGA used in this paper has 28326 configuration frames or 91548896 bits [165] and total configuration memory is partitioned into ten PR regions. The Speed of configuration in FPGA is directly proportional to the size of the bit file and bandwidth of ICAP port. In Kintex-7 FPGA maximum clock frequency used to drive ICAP controller is 100 MHz, available data width is 32 bit so maximum bandwidth is 3.2 Gbps [156].

To study fault recovery time of different models we have used a parameter coined as mean time to recover (MTTR) [89] which is the sum of bit file read back time, bit file download time and time required for EDAC on configuration data. Table 7.4 compares the fault recovery time of our proposed models with that of the existing models. It shows that download time and read back time is same for all models because in all models size of the bit files are same. To make the number of configuration frames in horizontal and vertical groups almost same the number of configuration frames in horizontal and vertical groups are taken as 168 and 169 respectively but they can be changed to any other values. In M1 and M2 no separate detection techniques are used, and error corrections are

done on each frame separately so error correction time is quite high in these two models compare to other models. In model M3, error correction on each group is performed in parallel. Similarly, M4 processes horizontal and vertical groups in parallel so both of them have same EDAC time. In M5, no separate error detection technique is required as in M1 and M2, but here error correction is done on each group in parallel unlike to M1 and M2. Hence, M5 has less error correction time compare to M1 and M2 but M5 has the same error correction time as M3 and M4. As authors in [97] did not include partial reconfiguration in their model, MTTR values are not available for this model with partial reconfiguration. Download and read back time of configuration frames are slightly less with PR compared to without PR. Simplified error detection technique compared to standard CRC and selective downloading of partial bit file after error correction helps to reduce download time. As error correction is done on each frame individually in M1, error correction time remains fixed in both with PR and without PR for M1. Error detection time in M4 also remains same in both the case, but error correction time is slightly increased with PR for M4 and M5. This is due to the fact that without PR all configuration frames are corrected in parallel but when PR is used error correction is performed on the configuration frames of a PR region only. Though error correction time increases reduction of download and readback time due to partial reconfiguration helps to reduce MTTR.

Figure 7.20 shows error correction time for different groups. Here only models M4 and M5 are plotted because error corrections in M1 and M2 are window based so the variation of cluster size will not affect their error correction time. Error correction time of M3 and M4 are quite same, so they are not plotted separately. It is clearly observed from Figure 7.20 that error detection and correction time reduces as cluster size along vertical and horizontal directions become close to each other. As there is no error detection in M5 error correction time is less in M5 than M4. From the above discussion, it can be concluded that introduction of PR may increase error detection and correction time for all models but overall MTTR or fault recovery time will be reduced due to the reduction of download time. At the same time, PR reduces the probability of suspending the normal system operation during error correction which is the main criteria for real time system. Hence out of the five models discussed M1 is best suited to mitigate the

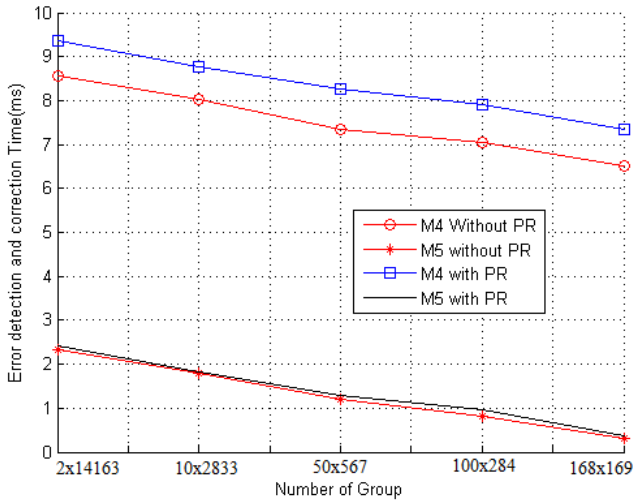

Figure 7.20: Comparison of fault recovery time with different group sizes for error detection with EVENODD and single error correcting EVENODD

effects of random discrete error and small size clustered error and M5 shows best performance in presence of large size clustered error in the configuration memory.

7.8 Conclusion

Protection of configuration memory of FPGA against soft error is a great challenge in the age of ultra-scale VLSI technology especially in different critical applications like avionics, biomedical electronics etc. In this paper, we have proposed error correcting models based on simple parity equation and erasure code to protect the configuration memory from both single bit and multi-bit clustered errors. The proposed error correcting models show good error correcting performance compared to the different existing models. At the same time, decoding complexity of the codes are very less compared to that of the other ECCs. The partial reconfiguration makes the proposed system more compatible for real time applications and also reduces fault recovery time and power consumption. We have also proposed an ICAP based hardware scheduler which can assign priority to the IPs for fault correction process. Experimental results prove that our proposed models deliver better performance in terms of error resiliency for real time applications.

Chapter 8

Conclusion and Future Scope

In this dissertation, our research work emphasizes on the design of high speed data acquisition methodologies in the domain of embedded system applications. We have chosen the DAQ system for HEP experiment as our example application for developing the hardware prototype, as this application gives us an opportunity to benchmark our design in terms of high throughput, error resiliency both in communication channel as well as in the storage devices. We have used FPGA as the target architecture for prototyping and estimating the various performance metrics of the subsystems designed and implemented in the course of this research work. In order to achieve complete DAQ setup we have proposed a high speed error resilient data communication infrastructure over optical fiber that will carry the data from the front end electronics board to the back-end computing nodes and MBUs in communication channel are mitigated using multi-bit error correcting codes like RS and orthogonal concatenated code. Our proposed DAQ system outperforms the state of the art solutions in terms of data rate, BER, error mitigation, area and power consumption. Proper steps have been taken to achieve a stabilized data transfer latency for high speed communication. We have also proposed a method that can monitor status of different modules of the DAQ system continuously and also user can send different commands to the DAQ system remotely as per the requirement.

Electronics devices developed using FPGA as the part of the DAQ system may be affected by radiation and hence proper precautions have been taken to mitigate the effect of radiation in the FPGA. Different error correction methods have

been proposed using parity based error correcting code, erasure code, interleaving and selective bit placement, which not only enhances error correction coverage but also have simple decoding circuit, consume less hardware resources compared to existing error correction techniques. Though use of hardware scheduling algorithm based partial reconfiguration along with error correction slightly increases error correction time, it will not interrupt the normal system operation during error correction. Not only configuration memory of FPGA devices but also storage elements like flash memory may be affected by soft errors due to radiation in the age of ultra large scale vlsi (ULSI) technology. With the increase of data volume and advancement of fabrication technology density of CMOS transistor in the memory element also increases and now a days single memory cell stores multiple data bits like in MLC flash memory. We have proposed a novel cluster error correction methods for MLC flash memory using simple linear block codes that shows better performance compared to the state of the art solutions in terms error correction coverage, cost and latency. An efficient hardware implementation of the proposed method has also been proposed using parallel processing and pipelining.

We have developed the DAQ system for HEP application and are planning to use the DAQ system in other domains like deep space exploration, biomedical applications where remote operation, accuracy and resiliency against fault due to radiation are the key issues. Hence, we are trying to modify the DAQ system so that it can be reconfigured remotely and status of the different modules can be checked using wireless communication without user intervention. In this research we have designed and implemented FPGA based GBT transmitter and receiver sub-systems which communicates through an optical link as a part of the DAQ system. We were successful in testing point to point GBT communication in ~Gb data rate. In future, we will try to develop an improved setup that can perform multiple GBT based communication using Master Salve GBT architecture to optimize the hardware requirement as well as to synchronize all the GBT blocks in a better way and thus reducing synchronization error.

In error correction and detection activity, we have mainly focused on logical errors assuming that they are created by the incident radiation. Our assumption is realistic as our error injector creates random errors in the hardware, which is

the most common approach in addressing hardware errors. But, truly speaking radiation can occur due to various particles (α, β, neutrino etc.) and of various doses and their effect on the semiconductor transport phenomenon must be studied thoroughly to understand the faults that will be created due to radiation effects. These fault models can guide us in understanding the error patterns that will be created in the hardware. In future, we are planning to develop a fault injector emulator that will realize real faults occurring in the semiconductor in respect to various radiation environments, which will test the error correcting capability of our proposed code in presence of various radiation sources. This will make our hardware system more reliable and robust.

References

[1] F. Bagenal. Juno: Mission to jupiter's interior - and poles. In *2012 Conference on Intelligent Data Understanding*, pages 6–6, Oct 2012. 1

[2] The graphic [mangalyaan mars orbiter]. *Engineering Technology*, 8(10):14–14, November 2013. 1

[3] O. R. Jones. Beam instrumentation systems of the large hadron collider: tutorial 50. *IEEE Instrumentation Measurement Magazine*, 17(1):42–48, February 2014. 1

[4] A. R. Donaldson and J. R. Ashton. Slac modulator availability and impact on slc operation. In *Proceedings Particle Accelerator Conference*, volume 1, pages 668–670 vol.1, May 1995. 1

[5] L. Groening*, P. Gerhard*, M. Maier*, S. Mickat*, A. Orzhekhovskaya*, H. Vormann*, and S. Yaramyshev*. *UNILAC Status Report*, volume 2014-1 of *GSI Report*, page 297 p. GSI Helmholtzzentrum für Schwerionenforschung, Darmstadt, 2014. 1

[6] B. F. Bohlender, J. Wiechula, M. Iberler, O. Kester, and J. Jacoby. Construction, characterization and optimization of a plasma window based on a cascade arc design for fair at the gsi hemholtz center. In *2016 IEEE International Conference on Plasma Science (ICOPS)*, pages 1–1, June 2016. 2

[7] P Senger and the Cbm Collaboration. The cbm experiment at fair. *Journal of Physics: Conference Series*, 50(1):357, 2006. 2, 14

[8] B. Friman. *The CBM Physics Book*. Springer, 2011. 2

[9] T. Matsui and H. Satz. j/ψ suppression by quark-gluon plasma formation. *Physics Letters B*, 178(4):416 – 422, 1986. 2

[10] Andrei Linde, Dmitri Linde, and Arthur Mezhlumian. From the big bang theory to the theory of a stationary universe. *Phys. Rev. D*, 49:1783–1826, Feb 1994. 2

[11] Lyndon Evans and Philip Bryant. Lhc machine. *Journal of Instrumentation*, 3(08):S08001, 2008. 3, 81

[12] The compressed baryonic matter experiment at fair. *DPG Frühjahrstagung "Hadronen und Kerne" Münster*. iv, x, 4, 7

[13] S. Chattopadhyay. *Technical Design Report for the CBM*. CBM Collaboration, 2014. iv, 4, 5, 49

[14] Fabio Sauli. The gas electron multiplier (gem): Operating principles and applications. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 805:2 – 24, 2016. Special Issue in memory of Glenn F. Knoll. 4

[15] Pavel Larionov and CBM Collaboration. Overview of the silicon tracking system for the cbm experiment. *Journal of Physics: Conference Series*, 599(1):012025, 2015. 5

[16] Rama Prasad Adak, Ajit Kumar, Anand Kumar Dubey, Subhasis Chattopadhyay, Supriya Das, Sibaji Raha, Subhasis Samanta, and Jogender Saini. Performance of a large size triple gem detector at high particle rate for the cbm experiment at fair. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 846:29 – 35, 2017. 6

[17] K. Kasinski, R. Kleczek, P. Otfinowski, R. Szczygiel, and P. Grybos. Stsxyter, a high count-rate self-triggering silicon strip detector readout ic for high resolution time and energy measurements. In *2014 IEEE Nuclear*

Science Symposium and Medical Imaging Conference (NSS/MIC), pages 1–6, Nov 2014. 6, 48

[18] F. Lemke and U. Bruening. A hierarchical synchronized data acquisition network for cbm. *IEEE Transactions on Nuclear Science*, 60(5):3654–3660, Oct 2013. iv, 6

[19] G.-H. Asadi and M.B. Tahoori. Soft error mitigation for sram-based fpgas. In *VLSI Test Symposium, 2005. Proceedings. 23rd IEEE*, pages 207–212, May 2005. 8, 27, 28

[20] T. J. Todman, G. A. Constantinides, S. J. E. Wilton, O. Mencer, W. Luk, and P. Y. K. Cheung. Reconfigurable computing: architectures and design methods. *IEE Proceedings - Computers and Digital Techniques*, 152(2):193–207, Mar 2005. 12

[21] Xilinx. 7 series fpgas configurable logic block. 2016. 12

[22] W. Zhou, P. Karlstrom, and D. Liu. Automatic synthesizable hdl generator for nogap. In *2012 IEEE/ACIS 11th International Conference on Computer and Information Science*, pages 119–123, May 2012. 13

[23] Xilinx. Highest performance, highest-capacity radhard fpga. 2015. 14

[24] Microsemi. Design techniques for radiation-hardened fpgas. 1997. 14

[25] M. Violante, L. Sterpone, M. Ceschia, D. Bortolato, P. Bernardi, M. S. Reorda, and A. Paccagnella. Simulation-based analysis of seu effects in sram-based fpgas. *IEEE Transactions on Nuclear Science*, 51(6):3354–3359, Dec 2004. 14, 28

[26] Felix Siegle, Tanya Vladimirova, Jørgen Ilstad, and Omar Emam. Mitigation of radiation effects in sram-based fpgas for space applications. *ACM Comput. Surv.* 14

[27] K. Omata, Y. Fujita, N. Yoshikawa, M. Sekiguchi, and Y. Shida. A data acquisition system based on a personal computer. *IEEE Transactions on Nuclear Science*, 39(2):143–147, Apr 1992. 18

[28] IAEA'. Data acquisition and analysis systems for nuclear research and applications: Current status and trends. 1982. 18

[29] M. Morhac, I. Turzo, and J. Kristiak. Pc-camac based data acquisition system for multiparameter measurements. *IEEE Transactions on Nuclear Science*, 42(1):1–6, Feb 1995. 18

[30] J. Toledo, F.J. Mora, and H. Müller. Past, present and future of data acquisition systems in high energy physics experiments. *Microprocessors and Microsystems*, 27(8):353 – 358, 2003. 18

[31] A. J. Lankford and T. Glanzman. Data acquisition and fastbus for the mark ii detector. *IEEE Transactions on Nuclear Science*, 31(1):225–229, Feb 1984. 19

[32] Abhinav Kumar. Vme data acquisition system: Fundamentals and beyond. *Bhabha Atomic Research Centre, Mumbai*, 2011. 19

[33] W. von Ruden. The aleph data acquisition system. *IEEE Transactions on Nuclear Science*, 36(5):1444–1448, Oct 1989. 19

[34] E.D. Platner, A. Etkin, K.J. Foley, J.H. Goldman, W.A. Love, T.W. Morris, S. Ozaki, A.C. Saulys, C.D. Wheeler, E.H. Willen, S.J. Lindenbaum, J.R. Bensinger, and M.A. Kramer. Programmable combinational logic trigger system for high energy particle physics experiments. *Nuclear Instruments and Methods*, 140(3):549 – 552, 1977. 19

[35] X Vidal and R Manzano. Lhc trigger, taking a closer look at lhc. *CERN*. 19

[36] V. Gligorov. Triggering in high energy physics experiments. *TESHEP Summer School, CERN*, 2012. 20

[37] Christian Alexander Steinle. *A First Level Trigger Approach for the CBM Experiment*. disserta,Verlag, 2012. iv, 20, 21

[38] Supratik Majumder and Scott Rixner. Comparing ethernet and myrinet for mpi communication. In *Proceedings of the 7th Workshop on Workshop on*

Languages, Compilers, and Run-time Support for Scalable Systems, LCR '04, pages 1–7, New York, NY, USA, 2004. ACM. 21

[39] W. Zheng, R. Liu, M. Zhang, G. Zhuang, and T. Yuan. Design of fpga based high-speed data acquisition and real-time data processing system on j-text tokamak. *Fusion Engineering and Design*, 89(5):698 – 701, 2014. Proceedings of the 9th IAEA Technical Meeting on Control, Data Acquisition, and Remote Participation for Fusion Research. 22

[40] Xilinx. Virtex-6 fpga configurable logic block, ug364 (v1.2), 2012. 22

[41] G. Blake, R. G. Dreslinski, and T. Mudge. A survey of multicore processors. *IEEE Signal Processing Magazine*, 26(6):26–37, November 2009. 22

[42] Xilinx. Zynq ultrascale+ mpsoc,technical reference manual. Mar 2017. 22

[43] C. Leong, P. Bento, P. Rodrigues, J. C. Silva, A. Trindade, P. Lousa, J. Rego, J. Nobre, J. Varela, J. P. Teixeira, and C. Teixeira. Design and test issues of a fpga based data acquisition system for medical imaging using pem. In *14th IEEE-NPSS Real Time Conference, 2005.*, pages 5 pp.–, June 2005. 23

[44] J Adamczewski-Musch, H G Essel, N Kurz, and S Linev. Data acquisition backbone core dabc release v1.0. *Journal of Physics: Conference Series*, 219(2):022007, 2010. 23

[45] J. Imrek, D. Novak, G. Hegyesi, G. Kalinka, J. Molnar, J. Vegh, L. Balkay, M. Emri, G. Molnar, L. Tron, I. Bagamery, T. Bukki, S. Rozsa, Z. Szabo, and A. Kerek. Development of an fpga-based data acquisition module for small animal pet. *IEEE Transactions on Nuclear Science*, 53(5):2698–2703, Oct 2006. 23

[46] Sergio Garcia Castillo and Krikor B. Ozanyan. Field-programmable data acquisition and processing channel for optical tomography systems. *Review of Scientific Instruments*, 76(9):095109, 2005. 23

[47] C. C. W. Robson, A. Bousselham, and Bohm. An fpga- based general-purpose data acquisition controller. *IEEE Transactions on Nuclear Science*, 53(4):2092–2096, Aug 2006. 23

[48] S. Minami, J. Hoffmann, N. Kurz, and W. Ott. Design and implementation of a data transfer protocol via optical fiber. *IEEE Transactions on Nuclear Science*, 58(4):1816–1819, Aug 2011. 23, 113

[49] E. Kadric, N. Manjikian, and Z. Zilic. An fpga implementation for a high-speed optical link with a pcie interface. In *SOC Conference (SOCC), 2012 IEEE International*, pages 83–87, Sept 2012. 23, 113

[50] Hao Xu, Zhan'an Liu, Yunpeng Lu, Lu Li, Dixin Zhao, and Ya'nan Guo. Fpga based high speed data transmission with optical fiber in trigger system of besiii. In *Nuclear Science Symposium Conference Record, 2007. NSS '07. IEEE*, volume 1, pages 818–821, Oct 2007. 23, 113

[51] L. Liu, C. Liu, Y. Peng, and D. Liu. A design of fibre channel node with pci interface. In *2013 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, pages 1817–1822, May 2013. 24, 113

[52] C. Mattihalli. Design and realization of serial front panel data port (sfpdp) protocol. In *2012 2nd International Conference on Consumer Electronics, Communications and Networks (CECNet)*, pages 2505–2509, April 2012. 24

[53] H. Kavianipour and C. Bohm. High performance fpga-based scatter/gather dma interface for pcie. In *2012 IEEE Nuclear Science Symposium and Medical Imaging Conference Record (NSS/MIC)*, pages 1517–1520, Oct 2012. 24

[54] A. A. Colavita, A. Cicuttin, F. Fratnik, and G. Capello. Sortchip: a vlsi implementation of a hardware algorithm for continuous data sorting. *IEEE Journal of Solid-State Circuits*, 38(6):1076–1079, June 2003. 24

[55] W. Min. Analysis on bubble sort algorithm optimization. In *Information Technology and Applications (IFITA), 2010 International Forum on*, volume 1, pages 208–211, July 2010. 24

[56] A. Davidson, D. Tarjan, M. Garland, and J. D. Owens. Efficient parallel merge sort for fixed and variable length keys. In *Innovative Parallel Computing (InPar), 2012*, pages 1–9, May 2012. 24

[57] W. Zhenhua, L. Zhifeng, and L. Guoliang. Parallel optimization strategy of heap sort algorithm under multi-core environment. In *2015 Seventh International Conference on Measuring Technology and Mechatronics Automation*, pages 768–771, June 2015. 24

[58] D. Mihhailov, V. Sklyarov, I. Skliarova, and A. Sudnitson. Hardware implementation of recursive sorting algorithms. In *Electronic Devices, Systems and Applications (ICEDSA), 2011 International Conference on*, pages 33–38, April 2011. 24

[59] Richard H. Maurer, Martin E. Fraeman, Mark N. Martin, and David R. Roth. Harsh environments: Space radiation environment, effects, and mitigation. *Johns Hopkins APL Technical Digest*, 28(1):17–29, 2008. 24

[60] J.W. Nieto, W.N. Furman, and M.A. Wadsworth. Automatic repeat request (arq) communication system using physical layer monitoring, july 2012. US Patent 8,213,402. 25

[61] T. Suutari, J. Isoaho, and H. Tenhumen. High-speed serial communication with error correction using 0.25 mu;m cmos technology. In *ISCAS 2001. The 2001 IEEE International Symposium on Circuits and Systems (Cat. No.01CH37196)*, volume 4, pages 618–621 vol. 4, May 2001. 25

[62] Y. Y. Jian, H. D. Pfister, K. R. Narayanan, Raghu Rao, and R. Mazahreh. Iterative hard-decision decoding of braided bch codes for high-speed optical communication. In *2013 IEEE Global Communications Conference (GLOBECOM)*, pages 2376–2381, Dec 2013. 26

[63] K. Deergha Rao. *Channel coding techniques for wireless communications*. Springer Publishing Company, 2015. 26

[64] C. W. Sham, X. Chen, W. M. Tam, Y. Zhao, and F. C. M. Lau. A layered qc-ldpc decoder architecture for high speed communication system. In *2012*

IEEE Asia Pacific Conference on Circuits and Systems, pages 475–478, Dec 2012. 26

[65] J. Nargis, D. Vaithiyanathan, and R. Seshasayanan. Design of high speed low power viterbi decoder for tcm system. In *2013 International Conference on Information Communication and Embedded Systems (ICICES)*, pages 185–190, Feb 2013. 26

[66] Jun Jin Kong and K. K. Parhi. Viterbi decoder architecture for interleaved convolutional code. In *Conference Record of the Thirty-Sixth Asilomar Conference on Signals, Systems and Computers, 2002.*, volume 2, pages 1934–1937 vol.2, Nov 2002. 26

[67] Dave Forney. Concatenated codes. *Scholarpedia*, 4(2):8374, 2009. 26

[68] L. Zhang, Z. Wang, Q. Hu, and J. Zhang. High speed concatenated code codec for optical communication systems. In *2009 Symposium on Photonics and Optoelectronics*, pages 1–4, Aug 2009. 26

[69] F. Yang and Q. Hu. Iterative decoding of orthogonally concatenated code for fiber communications. In *2013 2nd International Symposium on Instrumentation and Measurement, Sensor Network and Automation (IMSNA)*, pages 1097–1100, Dec 2013. 26, 27

[70] Z. Borui, H. Zhuli, and X. Kefei. Performance of rs-turbo concatenated code in aos. In *Electronic Measurement Instruments (ICEMI), 2013 IEEE 11th International Conference on*, volume 2, pages 983–987, Aug 2013. 26

[71] K. T. Sarika and P. P. Deepthi. A novel high speed communication system based on the concatenation of rs and qc-ldpc codes. In *2013 Annual International Conference on Emerging Research Areas and 2013 International Conference on Microelectronics, Communications and Renewable Energy*, pages 1–5, June 2013. 27

[72] V. Bhatia and A. Banerjee. Vhdl implementation of two-state multiple turbo codes. In *Communications (NCC), 2010 National Conference on*, pages 1–5, Jan 2010. 27, 167

[73] S. Gounai, T. Ohtsuki, and T. Kaneko. Performance of concatenated code with ldpc code and rsc code. In *2006 IEEE International Conference on Communications*, volume 3, pages 1195–1199, June 2006. 27

[74] X. Liu, Q. X. Deng, and Z. K. Wang. Design and fpga implementation of high-speed, fixed-latency serial transceivers. *IEEE Transactions on Nuclear Science*, 61(1):561–567, Feb 2014. 27

[75] Xilinx. 7 series fpgas clocking resources and gtx/gth transceivers user guide. 2015. 27

[76] S. Golshan and E. Bozorgzadeh. Single-event-upset (seu) awareness in fpga routing. In *2007 44th ACM/IEEE Design Automation Conference*, pages 330–333, June 2007. 28

[77] F.L. Kastensmidt, L. Sterpone, L. Carro, and M.S. Reorda. On the optimal design of triple modular redundancy logic for sram-based fpgas. In *Design, Automation and Test in Europe, 2005. Proceedings*, pages 1290–1295 Vol. 2, March 2005. 29

[78] Daniel P. Siewiorek and Robert S. Swarz. *Reliable Computer Systems (3rd Ed.): Design and Evaluation*. A. K. Peters, Ltd., Natick, MA, USA, 1998. 29

[79] B. Pratt, M. Caffrey, J.F. Carroll, P. Graham, K. Morgan, and M. Wirthlin. Fine-grain seu mitigation for fpgas using partial tmr. *Nuclear Science, IEEE Transactions on*, 55(4):2274–2280, Aug 2008. 29

[80] S. Manz, J. Gebelein, A. Oancea, H. Engel, and U. Kebschull. Radiation mitigation efficiency of scrubbing on the fpga based cbm-tof read-out controller. In *Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on*, pages 1–6, Sept 2013. 29, 170

[81] Ignacio Herrera-Alzu and Marisa López-Vallejo. Self-reference scrubber for tmr systems based on xilinx virtex fpgas. In *Proceedings of the 21st International Conference on Integrated Circuit and System Design: Power*

and Timing Modeling, Optimization, and Simulation, PATMOS'11, pages 133–142, Berlin, Heidelberg, 2011. Springer-Verlag. 29, 148

[82] J. Heiner, B. Sellers, M. Wirthlin, and J. Kalb. Fpga partial reconfiguration via configuration scrubbing. In *2009 International Conference on Field Programmable Logic and Applications*, pages 99–104, Aug 2009. 29, 148

[83] Leilei Song, Meng-Lin Yu, and M. S. Shaffer. 10- and 40-gb/s forward error correction devices for optical communications. *IEEE Journal of Solid-State Circuits*, 37(11):1565–1573, Nov 2002. 29

[84] C. H. Liu, S. W. Yen, C. L. Chen, H. C. Chang, C. Y. Lee, Y. S. Hsu, and S. J. Jou. An ldpc decoder chip based on self-routing network for ieee 802.16e applications. *IEEE Journal of Solid-State Circuits*, 43(3):684–694, March 2008. 29, 167

[85] R. Hentschke, F. Marques, F. Lima, L. Carro, A. Susin, and R. Reis. Analyzing area and performance penalty of protecting different digital modules with hamming code and triple modular redundancy. In *Integrated Circuits and Systems Design, 2002. Proceedings. 15th Symposium on*, pages 95–100, 2002. 30

[86] Ik Joon Chang, Jae-Joon Kim, Sang Phill Park, and K. Roy. A 32 kb 10t sub-threshold sram array with bit-interleaving and differential read scheme in 90 nm cmos. *Solid-State Circuits, IEEE Journal of*, 44(2):650–658, Feb 2009. 30

[87] A. Sanchez-Macian, P. Reviriego, and J.A. Maestro. Enhanced detection of double and triple adjacent errors in hamming codes through selective bit placement. *Device and Materials Reliability, IEEE Transactions on*, 12(2):357–362, June 2012. 30, 151

[88] Xilinx. Logicore ip soft error mitigation controller v3.4,product guide, 2012. 30

[89] Sang Phill Park, Dongsoo Lee, and K. Roy. Soft-error-resilient fpgas using built-in 2-d hamming product code. *Very Large Scale Integration (VLSI)*

Systems, IEEE Transactions on, 20(2):248–256, Feb 2012. viii, 30, 148, 149, 159, 163, 164, 167, 179, 192, 198, 199, 203, 204

[90] Ming Zhu, Li Yi Xiao, Li Li Song, Yan Jing Zhang, and Hong Wei Luo. New mix codes for multiple bit upsets mitigation in fault-secure memories. *Microelectronics Journal*, 42(3):553 – 561, 2011. 30

[91] S. Rhee, C. Kim, J. Kim, and Y. Jee. Concatenated reed-solomon code with hamming code for dram controller. In *2010 Second International Conference on Computer Engineering and Applications*, volume 1, pages 291–295, March 2010. 30

[92] M. Poolakkaparambil, J. Mathew, A. M. Jabir, and S. P. Mohanty. Low complexity cross parity codes for multiple and random bit error correction. In *Thirteenth International Symposium on Quality Electronic Design (ISQED)*, pages 57–62, March 2012. 30

[93] C. Argyrides, D. K. Pradhan, and T. Kocak. Matrix codes for reliable and cost efficient memory chips. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 19(3):420–428, March 2011. 30, 140, 141, 143, 144, 171, 173, 179, 192

[94] Lambert M. Surhone, Mariam T. Tennoe, and Susan F. Henssonow. *Reed-Muller Code*. Betascript Publishing, Mauritius, 2010. 30

[95] L. Frigerio, M. A. Radaelli, and F. Salice. Convolutional coding for seu mitigation. In *2008 13th European Test Symposium*, pages 191–196, May 2008. 30

[96] J. S. Plank. Erasure codes for storage applications. Tutorial Slides, presented at FAST-2005: 4th Usenix Conference on File and Storage Technologies, `http://web.eecs.utk.edu/$~$plank/plank/papers/FAST-2005.html`, 2005. 30

[97] M. Ebrahimi, P.M.B. Rao, R. Seyyedi, and M.B. Tahoori. Low-cost multiple bit upset correction in sram-based fpga configuration frames. *Very*

Large Scale Integration (VLSI) Systems, IEEE Transactions on, PP(99):1–1, 2015. viii, ix, 30, 147, 170, 171, 172, 185, 191, 198, 203, 205

[98] S. Punnekkat and A. Burns. Analysis of checkpointing for schedulability of real-time systems. In *Real-Time Computing Systems and Applications, 1997. Proceedings., Fourth International Workshop on*, pages 198–205, Oct 1997. 31, 172

[99] A. Sari, M. Psarakis, and D. Gizopoulos. Combining checkpointing and scrubbing in fpga-based real-time systems. In *VLSI Test Symposium (VTS), 2013 IEEE 31st*, pages 1–6, April 2013. 31

[100] A. Ziv and J. Bruck. An on-line algorithm for checkpoint placement. *Computers, IEEE Transactions on*, 46(9):976–985, Sep 1997. 31

[101] Ying Zhang and K. Chakrabarty. A unified approach for fault tolerance and dynamic power management in fixed-priority real-time embedded systems. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 25(1):111–125, Jan 2006. 31

[102] F. Dittmann and S. Frank. Hard real-time reconfiguration port scheduling. In *Design, Automation Test in Europe Conference Exhibition, 2007. DATE '07*, pages 1–6, April 2007. 31, 172

[103] Tae Rim Park, Jae Hyun Park, and Wook Hyun Kwon. Reducing os overhead for real-time industrial controllers with adjustable timer resolution. In *Industrial Electronics, 2001. Proceedings. ISIE 2001. IEEE International Symposium on*, volume 1, pages 369–374 vol.1, 2001. 31

[104] K. Wyllie P. Moreira, J. Christiansen. Gbtx manual, v0.15, 2016. iv, 35, 36, 45

[105] National Aeronautics and Texas Space Administration. Tutorial on reed-solomon error correction coding. 38, 39

[106] B. Tiwari and R. Mehra. Design and implementation of reed solomon decoder for 802.16 network using fpga. In *2012 IEEE International Conference on Signal Processing, Computing and Control*, pages 1–5, March 2012. 41

[107] Xilinx. 7 series fpgas gtx/gth transceivers (ug476). 2016. 42, 103

[108] R. Szczygiel W. Zubrzycka K. Kasinski, W. Zabolotny. Sts/much-xyter2 manual v1.30, 2017. v, 47, 50, 51, 68

[109] Thomas Walter, Frank Ludwig, Kay Rehlich, and Holger Schlarb. Novel Crate Standard MTCA.4 for Industry and Research. In *Proceedings, 4th International Particle Accelerator Conference (IPAC 2013): Shanghai, China, May 12-17, 2013*, page THPWA003, 2013. 53

[110] W.M. Zabolotny, G. Kasprowicz, A.P. Byszuk, D. Emschermann, M. Gumiński, B. Juszczyk, J. Lehnert, W.F.J.Müller, K. Poźniak, and R. Romaniuk. Versatile prototyping platform for data processing boards for cbm experiment. *Journal of Instrumentation*, 11(02):C02031. 55

[111] E. F. Dierikx, A. E. Wallin, T. Fordell, J. Myyry, P. Koponen, M. Merimaa, T. J. Pinkert, J. C. J. Koelemeij, H. Z. Peek, and R. Smets. White rabbit precision time protocol on long-distance fiber links. *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, 63(7):945–952, July 2016. 55

[112] Texas Instruments. Sn74avc8t245 8-bit dual-supply bus transceiver with configurable voltage translation and 3-state output. 2017. 55

[113] A. Cantoni, J. Walker, and T. D. Tomlin. Characterization of a flip-flop metastability measurement method. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 54(5):1032–1040, May 2007. 56

[114] Analog Devices. 16x16 digital crosspoint switch. 2017. 57

[115] M. Jiménez-López, J. L. Gutiérrez-Rivas, J. Díaz, E. López-Marín, and R. Rodríguez. Wr-zen: Ultra-accurate synchronization soc based on zynq technology. In *2016 European Frequency and Time Forum (EFTF)*, pages 1–4, April 2016. 57

[116] L. Meder, M. Dreschmann, O. Sander, and J. Becker. A signal distribution board for the timing and fast control master of the cbm experiment. *Journal of Instrumentation*, 11(02):C02001, 2016. v, 58, 59

[117] Robert Frazier and et.al. *The IPbus Protocol,An IP based control protocol for ATCA*. CERN, Switzerland, 2013. http://ohm.bu.edu/ chill90/ipbus/ipbus_protocol_v2_0.pdf. 59

[118] Behrouz A. Forouzan. *Data Communications and Networking*. The McGraw-Hill Companies., New Yorkr, 4th edition. 59

[119] Xilinx. Ethernet 1000 base-x pcs/pma or sgmii v14.3 (pg047). 2014. 59

[120] Xilinx. Tri-mode ethernet mac v9.0. 2016. 59

[121] L. Antoni, R. Leveugle, and B. Feher. Using run-time reconfiguration for fault injection applications. In *IMTC 2001. Proceedings of the 18th IEEE Instrumentation and Measurement Technology Conference. Rediscovering Measurement in the Age of Informatics (Cat. No.01CH 37188)*, volume 3, pages 1773–1777 vol.3, 2001. 74

[122] Wang Lixin, Song Wei, and Lv Chao. Implementation of high speed real time data acquisition and transfer system. In *2009 4th IEEE Conference on Industrial Electronics and Applications*, pages 382–386, May 2009. 78

[123] KM Potter. Luminosity measurements and calculations. 1994. 81

[124] Faisal Rasheed Lone, Arjun Puri, and Sudesh Kumar. Article: Performance comparison of reed solomon code and bch code over rayleigh fading channel. *International Journal of Computer Applications*, 71(20):23–26, June 2013. Full text available. vi, 85, 86

[125] K.M.Cheung and L.Swanson. A performance comparison between block interleaved and helically interleaved concatenated coding system, 1989. 87

[126] Xilinx. 7 series fpgas clocking resources and gtx/gth transceivers user guide. 2011. 88

[127] T. Takemoto, F. Yuki, H. Yamashita, S. Tsuji, Y. Lee, K. Adachi, K. Shinoda, Y. Matsuoka, K. Kogo, S. Nishimura, M. Nido, M. Namiwaka, T. Kaneko, T. Sugimoto, and K. Kurata. 100-gbps cmos transceiver for multilane optical backplane system with 1.3-cm2 footprint. In *2011 37th*

European Conference and Exhibition on Optical Communication, pages 1–3, Sept 2011. 102

[128] Alper Demir. Noise analysis for optical fiber communication systems. In *Proceedings of the 2003 IEEE/ACM International Conference on Computer-aided Design*, ICCAD '03, pages 441–, Washington, DC, USA, 2003. IEEE Computer Society. 108

[129] F. Costa and et.al. The alice c-rorc gbt card, a prototype readout solution for the alice upgrade. In *2016 IEEE-NPSS Real Time Conference (RT)*, pages 1–5, June 2016. 113

[130] Takuya Futatsuyama and et.al. A 113mm2 32gb 3b/cell nand flash memory. In *2009 IEEE International Solid-State Circuits Conference - Digest of Technical Papers*, pages 242–243, Feb 2009. 115

[131] N. Shibata and et.al. A 70 nm 16 gb 16-level-cell nand flash memory. *IEEE Journal of Solid-State Circuits*, 43(4):929–937, April 2008. 115

[132] L. M. Grupp, A. M. Caulfield, J. Coburn, S. Swanson, E. Yaakobi, P. H. Siegel, and J. K. Wolf. Characterizing flash memory: Anomalies, observations, and applications. In *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 24–33, Dec 2009. 115

[133] N. Mielke, T. Marquart, Ning Wu, J. Kessenich, H. Belgal, E. Schares, F. Trivedi, E. Goodness, and L. R. Nevill. Bit error rate in nand flash memories. In *2008 IEEE International Reliability Physics Symposium*, pages 9–19, April 2008. 115

[134] L. Pantisano and K. P. Cheung. Stress-induced leakage current (silc) and oxide breakdown: are they from the same oxide traps? *IEEE Transactions on Device and Materials Reliability*, 1(2):109–112, Jun 2001. 115

[135] Z. Cui, Z. Wang, and X. Huang. Multilevel error correction scheme for mlc flash memory. In *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 201–204, June 2014. 116

[136] D. Rossi and C. Metra. Error correcting strategy for high speed and high density reliable flash memories. *Journal of Electronic Testing*, 19(5):511–521, Oct 2003. 116

[137] Varsha Regulapati. Error correction codes in nand flash memory. pages 1–66, December 2015. 116

[138] E. Ibe, H. Taniguchi, Y. Yahagi, K. i. Shimbo, and T. Toba. Impact of scaling on neutron-induced soft error in srams from a 250 nm to a 22 nm design rule. *IEEE Transactions on Electron Devices*, 57(7):1527–1538, July 2010. vii, 116, 170

[139] H. Choi, W. Liu, and W. Sung. Vlsi implementation of bch error correction for multilevel cell nand flash memory. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 18(5):843–847, May 2010. 116

[140] S. Li and T. Zhang. Improving multi-level nand flash memory storage reliability using concatenated bch-tcm coding. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 18(10):1412–1420, Oct 2010. 116

[141] C. Yang, Y. Emre, and C. Chakrabarti. Product code schemes for error correction in mlc nand flash memories. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 20(12):2302–2314, Dec 2012. 116

[142] Jim Handy. Flash technology:annual update(objective analysis). pages 1–35, 2015. 118

[143] R.H.Fowler and L.Nordheim. Electron emission in intense electric fields. *Proceedings of the Royal Society, Part A*, 119:173–181, Mar 1928. 118

[144] M. Ohkawa and et.al. A 98 mm2 die size 3.3-v 64-mb flash memory with fn-nor type four-level cell. *IEEE Journal of Solid-State Circuits*, 31(11):1584–1589, Nov 1996. 120

[145] K. Takeuchi, T. Tanaka, and T. Tanzawa. A multipage cell architecture for high-speed programming multilevel nand flash memories. *IEEE Journal of Solid-State Circuits*, 33(8):1228–1238, Aug 1998. 121

[146] Kathrin Schacke. On the kronecker product. *Master's thesis, University of Waterloo*, 2004. 123

[147] L. J. Saiz-Adalid, P. Reviriego, P. Gil, S. Pontarelli, and J. A. Maestro. Mcu tolerance in srams through low-redundancy triple adjacent error correction. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(10):2332–2336, Oct 2015. 133, 143

[148] A. Neale and M. Sachdev. A new sec-ded error correction code subclass for adjacent mbu tolerance in embedded memory. *IEEE Transactions on Device and Materials Reliability*, 13(1):223–230, March 2013. vii, 138, 139, 140, 142, 143, 144, 145

[149] Xilinx. Device reliability report,second half 2015. *UG116 (v10.4)*, pages 1–110, April 2016. 141

[150] Sebastian Andreas Manz. *Radiation mitigation for SRAM-Based FPGAs in the CBM experiment*. PhD thesis, 2015. 141

[151] Martin S. Won. Meeting the performance and power imperative of the zettabyte era with generation 10, 2016. 147

[152] Daniel P. Siewiorek and Robert S. Swarz. *Reliable Computer Systems (3rd Ed.): Design and Evaluation*. A. K. Peters, Ltd., Natick, MA, USA, 1998. 147

[153] M. Fayyaz and T. Vladimirova. Detection of silent data corruption in fault-tolerant distributed systems on board spacecraft. In *2014 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 202–209, July 2014. 148

[154] P. Reviriego, M. F. Flanagan, S. F. Liu, and J. A. Maestro. On the use of euclidean geometry codes for efficient multibit error correction on memory systems. *IEEE Transactions on Nuclear Science*, 59(4):824–828, Aug 2012. 148

[155] A. Sánchez-Macián, P. Reviriego, and J. A. Maestro. Hamming sec-daed and extended hamming sec-ded-taed codes through selective shortening and bit placement. *IEEE Transactions on Device and Materials Reliability*, 14(1):574–576, March 2014. 148

[156] Xilinx. 7 series fpgas configuration (user guide). 161, 204

[157] Priya Mathew, Lismi Augustine, Sabarinath G., and Tomson Devis. Hardware implementation of (63, 51) BCH encoder and decoder for WBAN using LFSR and BMA. *CoRR*, abs/1408.2908, 2014. 167

[158] Xilinx. Single error correction and double error detection, 2006. 167

[159] R. C. Baumann. Radiation-induced soft errors in advanced semiconductor technologies. *IEEE Transactions on Device and Materials Reliability*, 5(3):305–316, Sept 2005. 170

[160] J. Allison and K. Amako et.al. Geant4 developments and applications. *IEEE Transactions on Nuclear Science*, 53(1):270–278, Feb 2006. 170

[161] Earl Fuller, Michael Caffrey, Anthony Salazar, Carl Carmichael, and Joe Fabula. Radiation testing update, seu mitigation, and availability analysis of the virtex fpga for space re-configurable computing", presented at the ieee nuclear and space radiation effects conference. In *in Proc. International Conference on Military and Aerospace Programmable Logic Devices*, 2000. 170

[162] K Aamodt and the Alice Collaboration. The alice experiment at the cern lhc. *Journal of Instrumentation*, 3(08):S08002, 2008. 170

[163] T. VanCourt and M. C. Herbordt. Application-specific memory interleaving enables high performance in fpga-based grid computations. In *2006 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 305–306, April 2006. 181

[164] M. Blaum, J. Brady, J. Bruck, and Jai Menon. Evenodd: an efficient scheme for tolerating double disk failures in raid architectures. *Computers, IEEE Transactions on*, 44(2):192–202, Feb 1995. 183, 184, 185, 191, 192

[165] V. Gligorov. Triggering in high energy physics experiments. *TESHEP Summer School, CERN*, 2012. 204