Hardware Based Implementation of CCSDS Formatted Data Packets

Major Project Report

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology

in

Electronics & Communication Engineering

(Embedded Systems)

By

Princy Teli

(15MECE30)

Electronics & Communication Engineering Department

Institute of Technology

Nirma University

Ahmedabad-382481

May 2017

Hardware Based Implementation of CCSDS Formatted Data Packets

Major Project Report

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology

in

Electronics & Communication Engineering

(Embedded Systems)

By

Princy Teli
(15MECE30)

Under the guidance of

External Project Guide:	**Internal Project Guide:**
Mrs.Bela Vaidya	**Dr. N.P.Gajjar**
Scientist Engineer(SF),	Professor, EC Department,
SAC-ISRO,	Institute of Technology,
Ahmedabad.	Nirma University, Ahmedabad.

Electronics & Communication Engineering Department

Institute of Technology

Nirma University

Ahmedabad-382 481

May 2017

Declaration

This is to certify that

a. The thesis comprises my original work towards the degree of Master of Technology in Embedded Systems at Nirma University and has not been submitted elsewhere for a degree.

b. Due acknowledgment has been made in the text to all other material used.

- Princy Teli
15MECE30

Disclaimer

Certificate

This is to certify that the Major Project entitled **"Hardware based Implementation of CCSDS Formatted Data Packets"** submitted by **Princy Teli (15MECE30)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in Embedded Systems, Nirma University, Ahmedabad is the record of work carried out by her under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination.The results embodied in this major project, to the best of our knowledge,haven't been submitted to any other university or institution for award of any degree or diploma.

Date: Place: Ahmedabad

Dr. N.P.Gajjar

Internal Guide Program Coordinator

Dr. D.K.Kothari **Dr. Alka Mahajan**

Head, EC Director, IT

Certificate

This is to certify that the Major Project entitled **"Hardware Based Implementation of CCSDS Formatted Data Packets"** submitted by **Princy Teli (15MECE30)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in Embedded Systems, Nirma University, Ahmedabad is the record of work carried out by her under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination.

Mrs. Bela Vaidya
Scientist Engineering(SF),
SAC-ISRO,
Ahmedabad

Acknowledgements

I would like to express my gratitude and sincere thanks to **Dr. Alka Mahajan**, Director, Institute of Technology **Dr. Dilip Kothari** Professor and Head of Electronics and Communication Engineering Department, and PG Coordinator of M.Tech Embedded Systems program for allowing me to undertake this thesis work and for his guidelines during the review process.

I take this opportunity to express my profound gratitude and deep regards towards **Dr. N.P.Gajjar**, guide of my major project for his exemplary guidance, monitoring and constant encouragement throughout the course of this thesis. The blessing, help, and guidance given by him time to time shall carry me a long way in the journey of life on which I am about to embark. I heartily express my gratitude to the management and staff of Space Applications Centre, ISRO, Ahmedabad, for providing the most compatible environment for working along with the bonus of information about a very dynamic field of their latest works and providing their valuable resources for the implementation of Project work. I would like to thank, **Mr. D.R. Goswami** (Group Director, PCEG) and **Mr. Ashish Mishra** (Head), for their motivation and support throughout the course of project work at the PCSVD-PCEG-SEDA, SAC. I particularly record my profound gratitude to **Mrs. Bela Vaidya**, my project guide for her proficient and enthusiastic guidance, for always helping, giving me good suggestions, solving my doubts and guide me to complete my project in a better way.

Lastly, I thank almighty, my parents, brother and friends for their constant encouragement without which this assignment would not be possible.

- Princy Teli

15MECE30

Abstract

In the field of satellite-based Data Reception, two major parameters, rate, and quality of data transmission are required to meet standard. CCSDS (Consultative Committee for Space Data System) is a protocol for satellite Data Communication. CCSDS standard allows configuring the rate and quality of data transmission across various platforms of a satellite. It defines the implementation for data exchange, in addition to facilitate interoperability between satellites as well as the satellite to ground station.In order to support CCSDS in ground segment, the overall requirement can be split into two broad categories, namely (1) Real Time Data Acquisition and (2) CCSDS Prescribed Processing.

The project is targeted to implement the CCSDS protocol in the ground segment using the FPGA-based real-time platform at 50 MHz. CCSDS processing typically comprises of Attach Sync Marker, Randomization and Reed-Solomon(RS) Encoding at the transmitter side and the Frame Synchronization, De-randomization and the Reed-Solomon Decoding and the Data acquisition System at the receiver side. The Real Time data Acquisition is continuous DMA(Direct Memory Access) of decoded data of satellite and sustained transfer to the host. Here the blocks of CCSDS at transmitter and receiver side are designed, simulated and implemented using VHDL. The hardware platform used is Xilinx based Zynq Soc (Z7020) and the simulation is carried out using Xilinx VIVADO. The real-time data acquisition system is implemented using the built-in ARM core of Zynq.

The testing of Transmitter side is carried out by implementation of ramp pattern of 50 MHz. It is given as an input of RS encoder and generates the parities for error correction of transmitted signal. At the receiver side encoded data is given as an input of decoder. For the testing of data acquisition system AXI IP of Ramp generator is implemented and that is given as input of AXI DMA.

Contents

List of Figures

List of Tables

Abbreviation Notation and Nomenclature

ARM	Advanced RISC Machines
ASM	Attached Sync Marker
AXI	Amba Extensible Interface
BUFG	Global Buffer
CAN	Controller Area Network
CCSDS	Consultative Committee for space Data System
DMA	Direct Memory Access
ELF	Executable and Linking Format
FF	Flip Flop
FPGA	Field Programmable gate Array
GPIO	General Purpose Input Output
I2C	Inter Integrated Circuit
IDE	Integrated Development Environment
IO	Input Output
JTAG	Joint Test Action Group
LDPC	Low Density Parity Checker
LUT	Look Up Table
LUTRAM	Look Up Table based Random Access Memory
PL	Programmable Logic
PRBS	Pseudo Random Binary Sequence
PS	Processing System
RS	Reed Solomon
SD Card	Secure Digital Secure Digital
SDK	Software Development Kit
SoC	System on Chip
SPI	Serial Peripheral Interface
VHDL	VHSIC Hardware Description Language

VHSIC	Very High Scale Integrated Circuit
UART	Universal Asynchronous receiver Transmitter
USB	Universal Serial Bus
ZedBoard	Zynq Evaluation and Development Board

Chapter 1

Introduction

1.1 Motivation

In recent times the number of Electro-Optical Imaging Sensors being developed by ISRO has gone up substantially. Traditionally, the interfaces and formats related to data were specific to different payloads. However, in order to meet shorter turn-around time ISRO is adopting CCSDS standards. Data Acquisition and Testing are extremely crucial activities in the process of sensor development.
So I have taken up the development of the real-time data acquisition system of CCSDS formatted data streams, and the CCSDS Prescribed Processing that may contribute to the development of future sensors being developed in ISRO.

1.2 Objective

The prime objective is the study of CCSDS standard and its implementation for payload data acquisition and processing in Real time. While there can be different implementations, one of them can be a solution comprising of FPGA and Microcontroller. To develop this type of an application, I need a development board, which has FPGA and processor preferably, on a single platform. Xilinx Zedboard is one such platform, which can fulfill this requirement and to develop an application, it also provides a Xilinx Vivado IDE. So my second objective is to study the Zynq platform and Vivado design suite and finally to develop and deliver a working solution, meeting the above objectives.

1.3 Scope

This project can be used as a real-time high-speed data reception system of the satellite. It can do data decoding and acquisition of the real-time data.

1.4 Requirements

Requirements are

- Knowledge of Windows and Linux Platform
- Xilinx Vivado IDE
- Each Device on Zynq Soc and Device Functionality
- Knowledge of C Language
- VHDL Programming
- Knowledge of FPGA
- Xilinx Software Development Kit
- Visual Studio

1.5 Thesis Organization

The rest of the thesis organized as follows.

Chapter 1 contains Motivation, Objective and the scope of the project also including the requirements for the project.

Chapter 2 describes Literature review part, and the background study of the CCSDS standard, Frame Formation in data transmission and reception process, basic tools, technologies and hardware used in the data acquisition systems.

Chapter 3 describes the Design and Implementation of CCSDS specified baseband data simulator with the brief description of all the processing blocks and algorithm applied on the actual data to transmit also including the simulation and the implementation results.

Chapter 4 describes the Design and Implementation of CCSDS specified baseband data processing with the brief description of all the processing blocks and the algorithm applied to the received data to decode also including the simulation results.

Chapter 5 describes the interface and the hardware used of the data acquisition system. It includes the description of DMA registers for the specific length of data transfer says direct register mode of DMA and also includes the scatter-gather mode of DMA for continuous data transfer.

Chapter 6 describes the hardware implementation for the data transfer, Processing sequence and test results.

Chapter 7 contains conclusion and future scope of the project.

Chapter 2

Literature Review

2.1 CCSDS Standard

The Consultative Committee for Space Data System(CCSDS) has published a set of standards for reliable space communication, which have been accepted worldwide. The purpose of this proposed standard is to specify synchronization and channel coding schemes used with telemetry space data link protocol[1] .The recommended standard proposed a packetized transmission scheme which provides the specification in synchronization and channel coding scheme in terms of the data formats and procedures. Standard provides the functions like frame synchronization, pseudo-randomization, and channel coding scheme to enable a reliable data transfer through space link[1]. It provides additional functionality necessary for transferring data packets over a space link[1].

The CCSDS proposed standard provides frame synchronization using a sync marker. It specifies a pseudo randomization to improve data reception using a random number generator[1]. And give an optional Reed-Solomon error correction codes to provide noise immunity[1].

Figure 2.1 shows the layers of the data transmission and the reception process. At the transmitter side how the data formatted and the at the reception side how it will be decoded. Figure 2.2 shows how the frame generation of this layered architecture implemented practically using a multiplexer. And Figure 2.3 shows the location of frame sync, actual data and the parities for the error correction of transfer frame in every data frame.

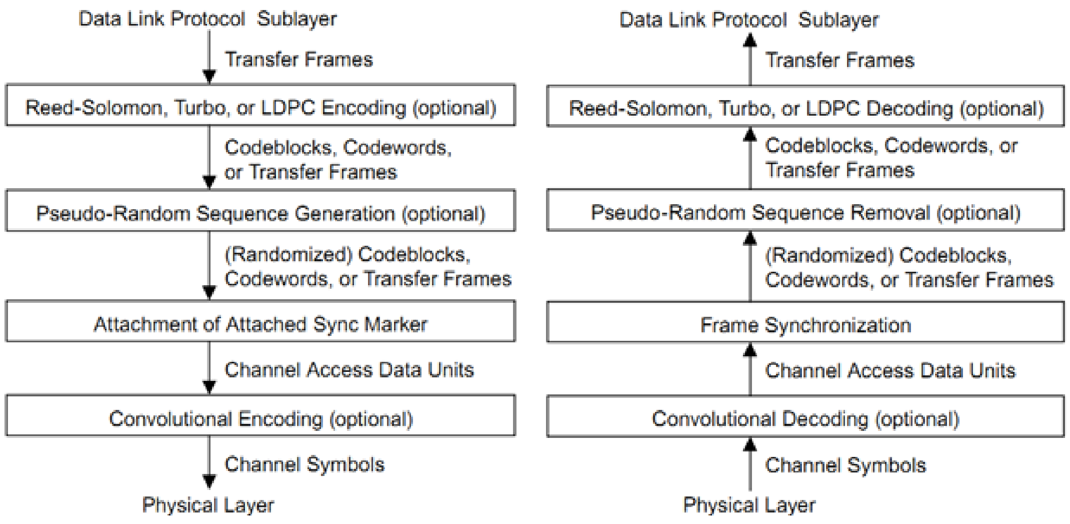

Data Link Protocol Sublayer
Transfer Frames
Reed-Solomon, Turbo, or LDPC Encoding (optional)
Codeblocks, Codewords, or Transfer Frames
Pseudo-Random Sequence Generation (optional)
(Randomized) Codeblocks, Codewords, or Transfer Frames
Attachment of Attached Sync Marker
Channel Access Data Units
Convolutional Encoding (optional)
Channel Symbols
Physical Layer
Data Link Protocol Sublayer
Transfer Frames
Reed-Solomon, Turbo, or LDPC Decoding (optional)
Codeblocks, Codewords, or Transfer Frames
Pseudo-Random Sequence Removal (optional)
(Randomized) Codeblocks, Codewords, or Transfer Frames
Frame Synchronization
Channel Access Data Units
Convolutional Decoding (optional)
Channel Symbols
Physical Layer

Figure 2.1: Data Transmission and Reception Process[1]

2.1.1 CCSDS Specified Data Transmission Process

Attached Sync Marker

Frame synchronization is necessary for proper decoding and subsequent processing of the Transfer Frames[1]. Furthermore, it is necessary for synchronization of the pseudo-random generator. The data unit consists of the ASM and the Transfer Frame[1]. The Transfer Frame, Codeword, may or may not be randomized. The ASM shall immediately follow the end of the preceding codeword[1].

Pseudo-Randomizer

In order for the receiver system to work properly, every data capture system at the receiving end requires that the incoming signal have sufficient bit transition density

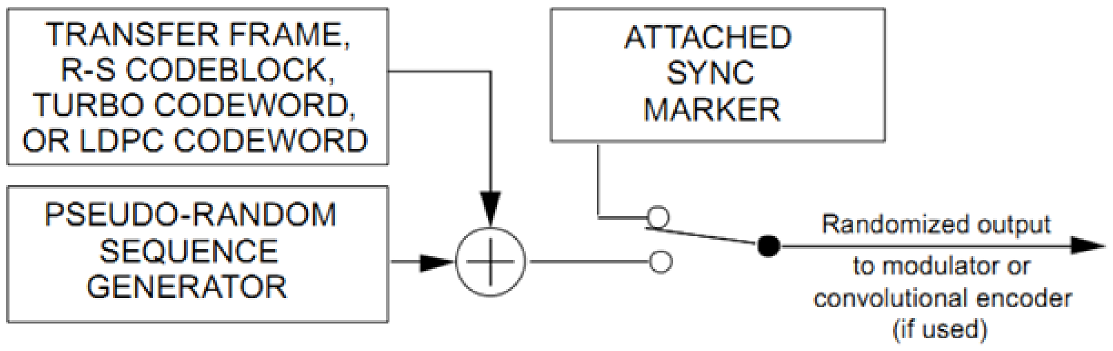

TRANSFER FRAME, R-S CODEBLOCK, TURBO CODEWORD, OR LDPC CODEWORD
PSEUDO-RANDOM SEQUENCE GENERATOR
ATTACHED SYNC MARKER
Randomized output
to modulator or convolutional encoder (if used)

Figure 2.2: Frame Generation

[1]

and allow proper synchronization of the decoder[1]. In order to ensure proper receiver operation, the data stream must be sufficiently random[1].

The Pseudo-Randomizer defines the preferred method to ensure sufficient randomness for all combinations of CCSDS recommended modulation and coding schemes[1].The presence or absence of pseudo-randomization is fixed for a Physical Channel and is managed by the receiver. Transfer Frame shall be randomized by exclusive-Oring the first bit of the Transfer Frame with the first bit of the pseudo-random sequence, followed by the second bit of the Transfer Frame with the second bit of the pseudo-random sequence, and so on[1]. On the receiving end, the original Transfer Frame shall be reconstructed using the same pseudorandom sequence[1].

Reed-Solomon Encoding

With the Reed-Solomon Codes specified only certain specific lengths of Transfer Frames may be contained within the codewords data space[1]. In some cases, these lengths can be shortened at a small sacrifice in coding gain. Since these R-S codes have a symbol length of 8 bits, the length of the codeword is a multiple of the interleaving depth, which provides octet compatibility[1]. The Reed-Solomon encoder reads in k data symbols, computes the (n - k) parity symbols, and appends the parity

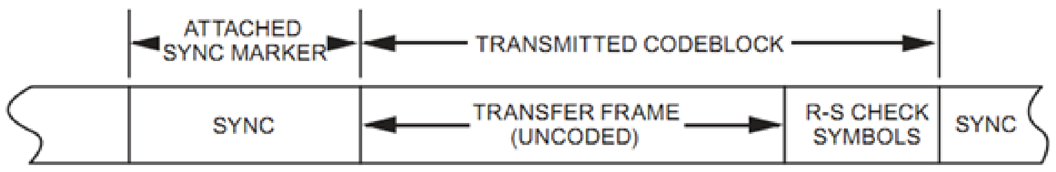


Figure 2.3: Frame Format[1]

symbols to the k data symbols for a total of n symbols. The encoder is essentially a 2t tap shift register where each register is m bits wide.

2.1.2 CCSDS Specified Data Reception Process

Synchronization

Codeblock synchronization of the Reed-Solomon decoder is achieved by Synchronization of the Attached Sync Marker associated with each code block [1].

De-Randomizer

At the receiving end, the original Frame is reconstructed using the same pseudo random sequence. After locating the ASM in the received data stream, the pseudo-random sequence is exclusive-ORed with the data bits immediately following the ASM[1]. The pseudorandom sequence is applied by exclusive-Oring the first bit following the ASM with the first bit of the pseudo-random sequence, followed by the second bit of the data stream with the second bit of the pseudo-random sequence, and so on[1].

RS-Decoder

The Reed-Solomon decoder tries to correct errors and/or erasures by calculating the syndromes for each codeword. Based upon the syndromes, the decoder is able to determine the number of errors in the received block[6, 8]. If there are errors present, the decoder tries to find the locations of the errors using the Berlekamp-Massey algorithm by creating an error locator polynomial. The roots of this polynomial are found using the Chien search algorithm. Using Forney's algorithm, the symbol error values are found and corrected. For an RS (n, k) code where n - k = 2T, the decoder can correct up to T symbol errors in the code word. Given that errors may only be corrected in units of single symbols (typically 8 data bits), Reed-Solomon codes work best for correcting burst errors[9].

2.2 Tools and Technology

Zynq devices are newly proposed to be System-on-Programmable-Chip. The general architecture of the Zynq comprises two sections: the Processing System (PS), and the Programmable Logic (PL) and they can be used separately or together. And each section has the separate power circuitry, which can enable either the PS or PL and be supporting powered down mode if not in use[2]. However, when both parts of the Zynq are used together represents the most compelling use and therefore it is important to understand the reliable structure of both sections, as well as the interfaces between them[2].
For high-speed logic the PL section is ideal for implementation. It has Configurable Logic Blocks (CLBs) for arithmetic and data flow subsystems and Input/ Output Blocks (IOBs) for interfacing, while the software routines and/or operating systems are supported by the PS section[2]. So the overall functionality of any fully functioned and designed system can be efficiently partitioned between hardware and software. Links between the PL and PS are made using industry standard Advanced Extensible Interface (AXI) connections.[2] Figure 2.4 shows the Zynq Architecture which contains Programmable Logic and the Processing System, which are connected via AXI bus. All the hardware peripheral implemented on FPGA and the process will operate it as per the requirement.

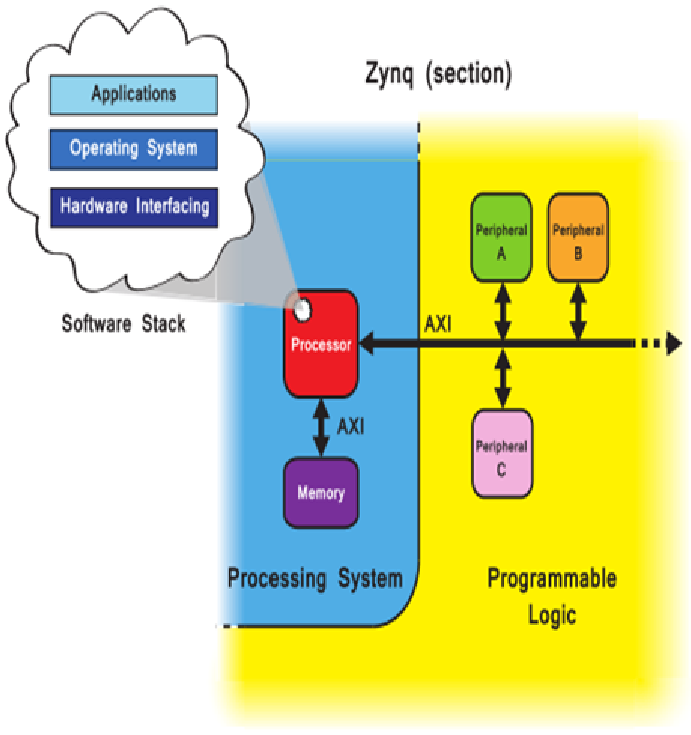


Figure 2.4: Zynq Architecture[2]

2.2.1 Processing System

All Zynq devices have almost same architecture, As an Application Processing unit it contains a dual-core ARM Cortex-A9 processor[2]. It is a hard coded on the device. It also consists soft processor like Xilinx MicroBlaze, which is formed by combining the element of programmable logic elements. Hard coded processor can achieve higher performance as compared to soft processor[2].
ARM processing unit contains processing engine, Floating point unit, Memory Management Unit, and cache memory[2]. It has Snoop Control Unit and some interfaces with PL too. To program a deployed components on the Pl section there is a support of Xilinx Software Development Kit which includes an ARM instruction set[2] .

2.2.2 Processing System External Interface

The Zynq Processing System has many interfaces by which it communicates with external peripherals and also with the PL section. External interfaces is achieved via Multiplexed Input/Output(MIO) and Extended MIO[2].
The Zynq PS features a variety of interfaces, both between the PS and PL, and between the PS and external components[2].
I/O Interface:

- SPI (x2)
- I2C (x2)
- CAN (x2)
- UART (x2)
- GPIO
- SD (x2)
- USB (x2)
- Ethernet

2.2.3 Programmable Logic

The second principal part of the Zynq architecture is the programmable logic. This is based on the Artix-7 and Kintex-7 FPGA fabric[2].
The Logic Fabric: The PL part of the Zynq device is depicted with various features highlighted. The PL is predominantly composed of general purpose FPGA logic fabric, which is composed of slices and Configurable Logic Blocks (CLBs), and there are also Input/ Output Blocks (IOBs) for interfacing[2].
Configurable Logic Block (CLB) CLBs are small, regular groupings of

logic: Elements that are laid out in a two-dimensional array on the PL, and connected to other similar resources via programmable interconnects. Each CLB is positioned next to a switch matrix and contains two logic slices[2].

- Slice A sub-unit within the CLB, which contains resources for implementing combinatorial and sequential logic circuits. Zynq slices are composed of 4 Lookup Tables, 8 Flip-Flops, and other logic[2].
- Lookup Table (LUT) A flexible resource capable of implementing (i) a logic function of up to six inputs; (ii) a small Read Only Memory (ROM); (iii) a small Random Access Memory (RAM); or (iv) a shift register. LUTs can be combined together to form larger logic functions, memories, or shift registers, as required.[2]
- Flip-flop (FF) A sequential circuit element implementing a 1-bit register, with reset functionality. One of the FFs can optionally be used to implement a latch[2].
- Switch Matrix A switch matrix sits next to each CLB, and provides a flexible routing facility for making connections (i) between elements within a CLB; and (ii) from one CLB to other resources on the PL[2].

2.2.4 Special Resources

The Block RAMs in the Zynq-7000 are equivalent to those on Xilinx 7 series FPGAs, and they can implement Random Access Memory (RAM), Read Only Memory (ROM), and First In First Out (FIFO) buffers, while also supporting Error Correction Coding (ECC)[2].

2.2.5 Programmable Logic External Interfaces

The PL includes another hard IP component: the XADC block. This is a dedicated set of Analogue to Digital Converter (ADC) mixed-signal hardware, which features two separate 12-bit ADCs both capable of sampling external analogue input signals at 1Msps[2].
clocks The PL receives four separate clock inputs from the PS, and additionally has the facilities to generate and distribute its own clock signals independently of the PS[2].
Programming and Debug A set of JTAG ports are provided in the PL section to facilitate configuration and debugging of the PL[2].

2.3 Vivado IDE

Vivado IDE is an integrated development environment for creating the hardware system part of the SoC design, i.e. the processor, memories, peripherals, external interfaces and bus connections. Vivado IDE interacts with other tools in the Vivado Design Suite, and also includes facilities for integrating and packaging IP, which enhances possibilities for design reuse[3]. SDK is a software design suite based on the popular Eclipse platform, which includes driver support for all Xilinx IPs, GCC library support for ARM and NEON extensions using the C and C++ languages, and tools for debugging and profiling[3].

2.3.1 Vivado Design Flow for Zynq

- The implementation process begins with launching Vivado, which is from design entry through bitstream generation[3].

- From Vivado, Create Block Design and configure settings to make the appropriate design decisions such as selection/de-selection of dedicated PS I/O peripherals, memory configurations, clock speeds, etc[3].

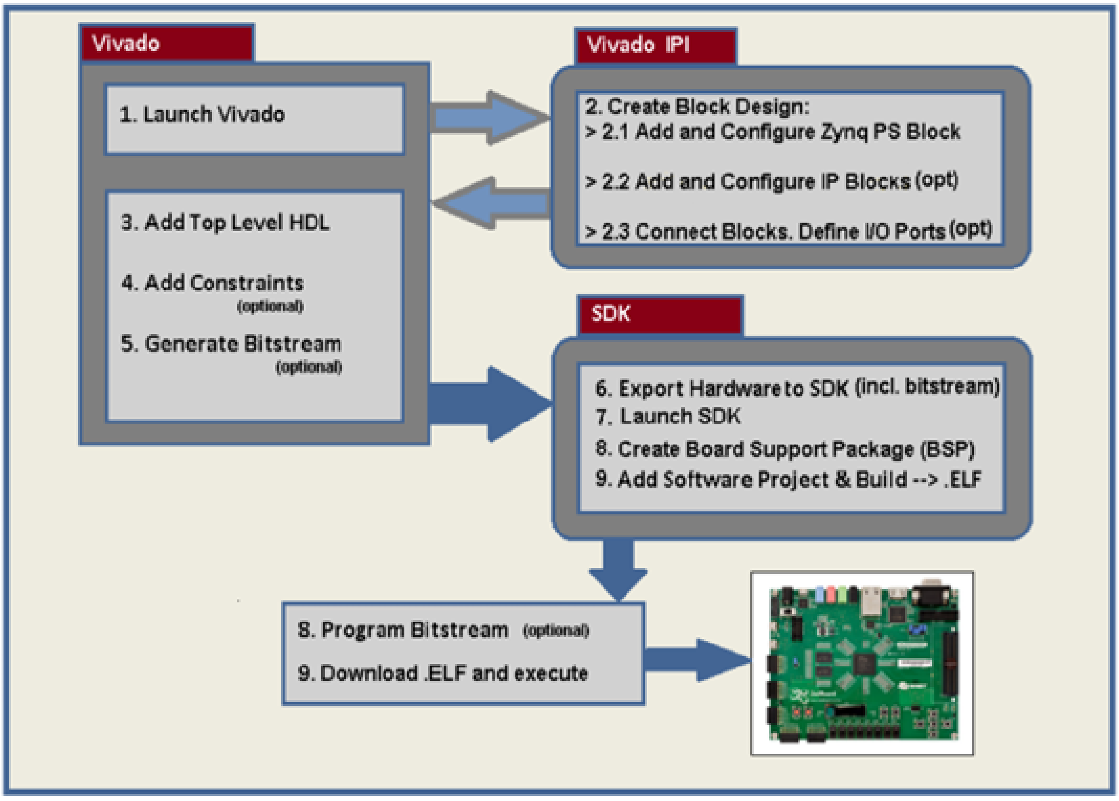

Vivado
1. Launch Vivado
3. Add Top Level HDL
4. Add Constraints (optional)
5. Generate Bitstream (optional)
Vivado IPI
2. Create Block Design:
> 2.1 Add and Configure Zynq PS Block
> 2.2 Add and Configure IP Blocks (opt)
> 2.3 Connect Blocks, Define I/O Ports (opt)
SDK
6. Export Hardware to SDK (incl. bitstream)
7. Launch SDK
8. Create Board Support Package (BSP)
9. Add Software Project & Build --> .ELF
8. Program Bitstream (optional)
9. Download .ELF and execute

Figure 2.5: Vivado Design Flow[3]

- At this point, you may also optionally add IP from the IP catalog or create and add your own customized IP. Connect the different blocks together by dragging signals/nets from one port of an IP to another. Use the design automation capability of the IP Integrator to automatically connect blocks together[3].

- When finished, generate a top-level HDL wrapper for the system[3].

- Ensure that the appropriate PL related design constraints are defined as required by the tools. If a supported evaluation board is specified during project creation (i.e.: the ZedBoard) then constraints may not be needed since the software is board aware[3]. If any signal coming from the PL section to an I/O pin is not defined then the tools will generate an error during the bit-stream generation[3]. Also, do not include pin constraints which are connected to the dedicated pins as the tools will generate error messages. This is done via creation/addition of a Xilinx Design Constraints (XDC) file to the Vivado project[3].

- Generate the bit- stream for configuring the logic in the PL if soft peripherals or other HDL are included in the design, or if any hard peripheral IO were routed through the PL[3]. At this stage, the hardware has been defined in ¡system¿[3]. hdf, and if necessary a bit-stream ¡system¿.bit has been generated. The bit-stream could be programmed into the FPGA from within Vivado, or it could be done from within SDK[3].

- Now that the hardware portion of the embedded system design has been built, export the design to the SDK to create the software design. A convenient method to ensure that the hardware for this design is automatically integrated with the software portion is achieved by Exporting the Hardware[3]. In order to export the design successfully, the Design Block MUST be open and the implemented design, if exists, MUST be open, otherwise the tools will report an error. Once the hardware has been exported, as a separate step, the SDK can then be launched[3].
  Figure 2.5 shows the graphical representation of this Vivado design flow for creating any system on the Zync board.

2.3.2 System Setup and Requirements

Figure 2.6 shows the setup for the development of the any system based on the Zynq Board.

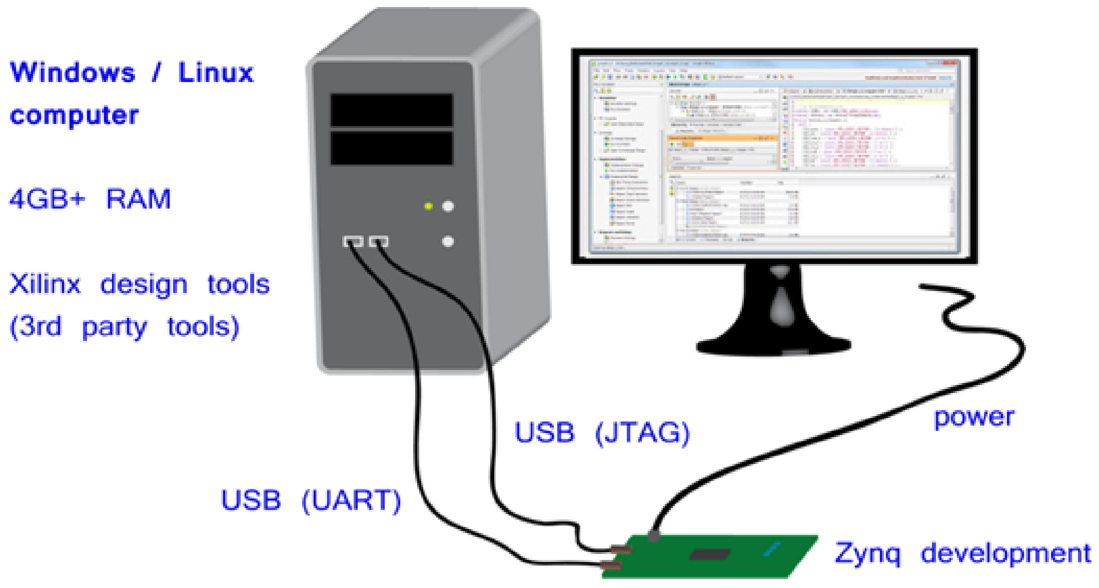


Figure 2.6: System Setup[3]

2.3.3 An Outline of the Design Flow

Figure 2.7 shows the work flow to develop any system. It shows the step to step procedure for implementation of any new system to follow.

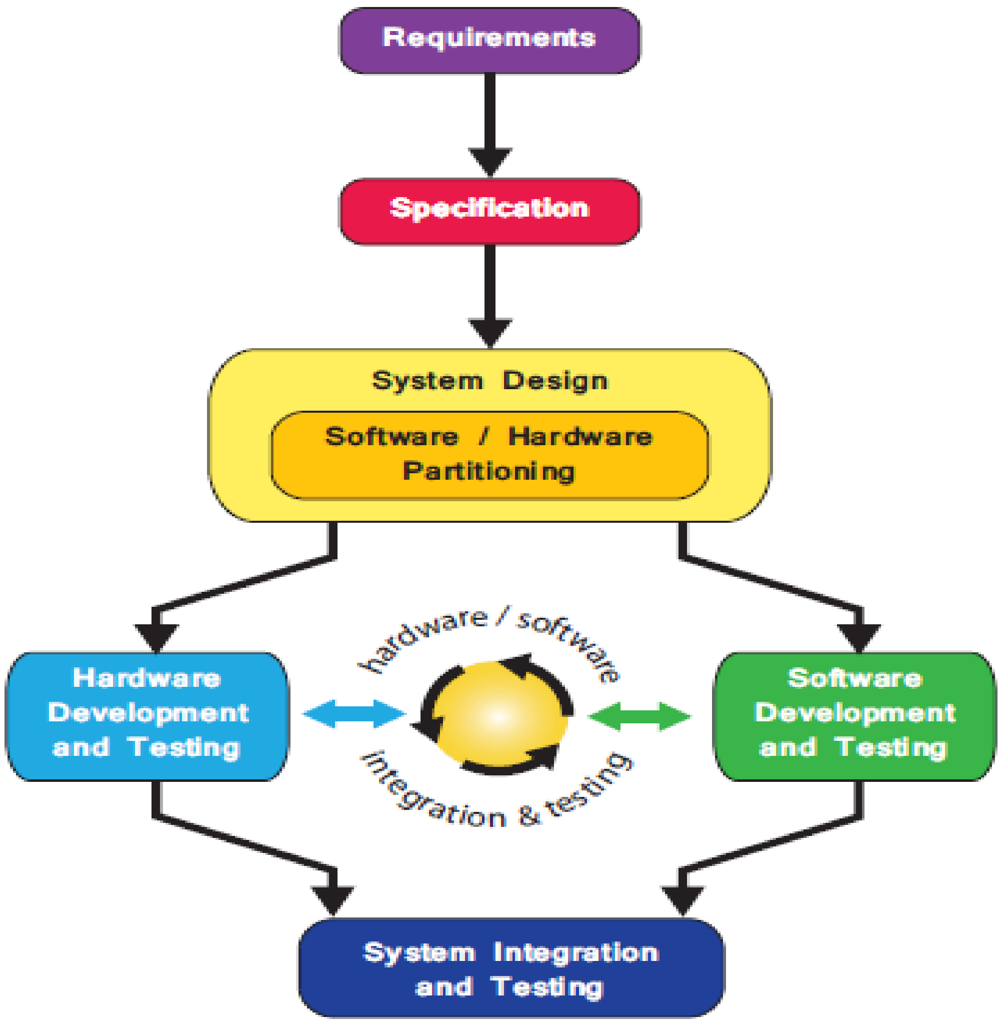

Requirements
Specification
System Design
Software / Hardware Partitioning
Hardware Development and Testing
hardware / software
integration & testing
Software Development and Testing
System Integration and Testing

Figure 2.7: Design Flow [3]

Chapter 3

Design and Implementation of FPGA-Based CCSDS Specified Baseband DATA Simulator

3.1 Introduction

CCSDS based data transmission standard is planned for future programs such as GISAT. The space data link protocols are defined for creating transfer frames from the space sensor data, received from the space packet service defined by the Space Packet Protocol, these transfer frames can be transmitted, they must be encoded using error correction codes, Synchronized using CCSDS specified Sync Marker and Randomize using Pseudo Random Sequence. Sync marker provides the synchronization between two frames in the space data packets. Randomization is used for the proper transition of bits in the communication channel. And the encoder facilitates the reception of the Baseband data stream in the event of any errors which are introduced due to the noisy channel environment of the space link. For this purpose, the transfer frame is encoded using the ReedSolomon(RS) Code as defined for the CCSDS data transmission standard.

The reception and baseband data acquisition of such CCSDS encoded serial streams with RS coding requires the design and development of RS decoder unit. Before that, it is required to develop a Data Simulator with CCSDS RS encoding, equivalent to that proposed for the satellite sensor. The subsequent sections in this technical note provide a brief on RS Coding and explain the design, development and implementation details of FPGA-based of RS Encoder (255,223).The design of the RS encoder is general purpose in nature.

3.2 Synchronization

Synchronization is necessary for proper decoding of any transmitted code-blocks or the transfer frames. Here synchronization is necessary for Reed-Solomon code-blocks and subsequent processing of the Transfer Frames[1]. Furthermore, it is necessary for synchronization of the pseudo-random sequence generator because after the attachment of sync the random number generation and simultaneously the randomization of encoded data will start. If the Physical Channel is not Reed-Solomon, turbo or LDPC coded the synchronization is achieved by using a stream of fixed-length Frames with an Attached Sync Marker (ASM) between them[1]. The data unit that consists of the ASM and the Frame is called the Channel Access Data Unit. The ASM is NOT a part of the encoded data space of the Reed-Solomon Codeblock, and it is not presented to the input of the Reed-Solomon encoder or decoder[1].

3.3 Pseudo Random Sequence Generator

In order for the receiver system to work properly, every data capture system at the receiving end requires that the incoming signal has sufficient bit transition density, and allow proper synchronization of the decoder[1]. In order to ensure proper receiver operation, the data stream must be sufficiently random. The Pseudo-Randomizer defined in this section is the preferred method to ensure sufficient randomness for

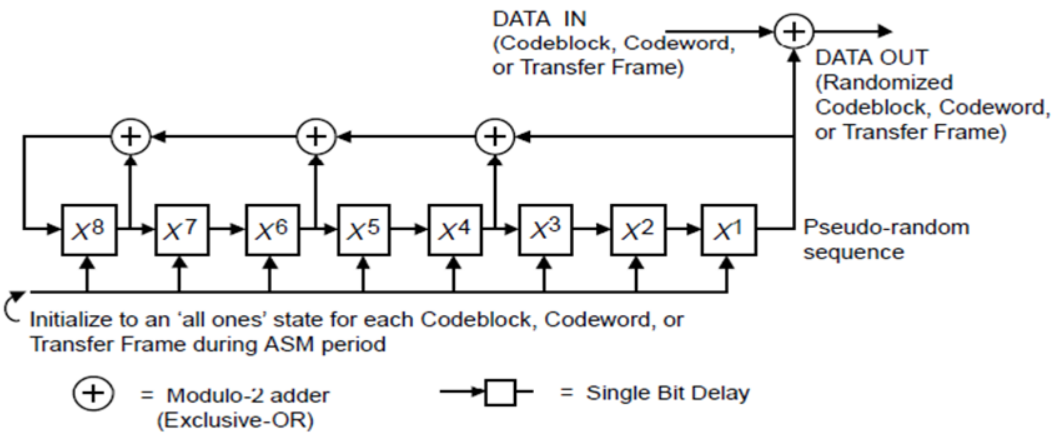

DATA IN
(Codeblock, Codeword, or Transfer Frame)
DATA OUT
(Randomized Codeblock, Codeword, or Transfer Frame)
X8
X7
X6
X5
X4
X3
X2
X1
Pseudo-random sequence
Initialize to an 'all ones' state for each Codeblock, Codeword, or Transfer Frame during ASM period
= Modulo-2 adder (Exclusive-OR)
= Single Bit Delay

Figure 3.1: Block diagram of Pseudo Random Sequence Generator[1]

all combinations of CCSDS-recommended modulation and coding schemes. The method for ensuring sufficient transitions is to exclusive-OR each bit of the Frame with a standard pseudo-random sequence[1]. If the pseudo-randomizer is used, on the sending end it is applied to the Reed-Solomon code block after encoding. On the receiving end, it is applied to derandomize the data after Frame synchronization but before Reed-Solomon decoding.The pseudo-random sequence is applied starting with the first bit of the Frame[1]. On the sending end, the Frame is randomized by exclusive-Oring the first bit of the Frame with the first bit of the pseudo-random sequence, followed by the second bit of the Frame with the second bit of the pseudo-random sequence, and so on. The pseudo-random sequence shall NOT be exclusive-ORed with the ASM[1]. Figure 3.1 shows the block diagram of Pseudo Random Sequence Generator in HDL using the simple D flip-flop. Initially, the value of the register is one and with respect to clock it will generate a random sequence of bits and after implementation, we will be noticed that it will repeat the bits after 255 clock cycles.

3.4 RS-Encoding Algorithm

The ReedSolomon (RS) codes are particularly useful when the communication channel is prone to burst errors, and on channels where the set of input symbols is large. RS codes are nonbinary cyclic BCH codes i.e. the symbols made of mbit binary sequences, where m2. There are 2m1 symbols in an RS codeword[4]. To be able to correct errors in t symbols, there must be r = 2t parity check symbols among the 2m1 symbols. These mbit symbols are considered as elements of a finite field of 2m1 elements, also called a 'Galois Field' which is explained in the next section[4].
An important characteristic of RS codes which makes them desirable to be used is that they have the largest possible code minimum distance among all block codes, in terms of number of symbols in which two codewords differ, for the same input and output block length[4]. For RS Codes, this code minimum distance dmin = n - k + 1. So, this code is capable of correcting t or fewer errors where, t = (dmin - 1)/2 = (n - k)/2. The most attractive RS codes have high code rates (k/n), i.e. low redundancy. The (255, 223) RS code, i.e. with m = 8, t = 16 has become a standard and widely used code. A modified version of this code suited to needs of space data systems is defined by CCSDS[4].

Linear Feedback Shift Register Method In the linear feedback shift register method [10], the polynomial division can be achieved efficiently by means of an n - k stage shift register. This method lends itself to easier hardware implementation as shown in Figure 3.2, as the process is divided into unit processes, which are in the form of either combinational (XOR, AND, OR) or sequential logic elements (latches) and hence, this method is more amenable for implementation in programmable digital logic devices described in VHDL or Verilog[10]. In the figure below, the block diagram for (255, 223) RS Encoder is given which is implemented using n - k = 32 stage shift register. Each stage stores an 8bit symbol (m=8). Two multiplexers control the inputs to the shift register as well as the output that is shifted out[10].

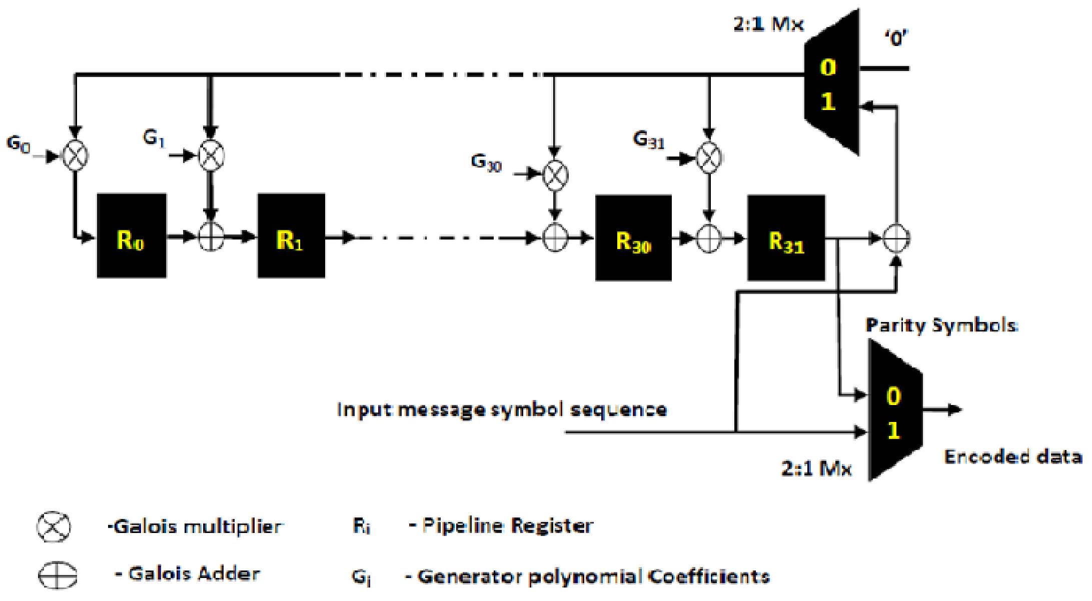


Figure 3.2: Reed-Solomon Encoder Block Diagram

[1]

Introduction to Galois Fields A finite field with p^n elements is denoted by GF (p^n), where p is a prime number and is also called the Galois Field, in honor of the founder of finite field theory, variste Galois[4]. Arithmetic operations (addition, subtraction, multiplication) on integers are done, as usual, followed by reduction modulo p. For instance, in GF (5), 4+3 = (7 modulo 5) = 2. Division is multiplication by the inverse modulo p. Elements of GF(p^n) may be represented as polynomials of degree strictly less than n with coefficients in GF(p). Operations are then performed modulo R where R is an irreducible polynomial of degree n over GF(p)[4]. The addition of two polynomials P and Q are done as usual i.e. W = P + Q. The multiplication may be done by first computing W = $P \cdot Q$ as usual, then computing the remainder modulo R. But there exist better ways to implement this more efficiently[4]. A particular case of interest is GF(2) i.e. when the prime is 2, where addition is exclusive OR (XOR) and multiplication is AND. It is conventional to express elements of GF(2^n) as binary numbers. They can also be expressed as polynomials where the coefficient of each term of the polynomial is same as the corresponding bit in element's binary expression[4].

Obtaining Galois field GF (2^n) The field GF (2^n) is called the extension of the binary field GF(2)[4]. Besides the numbers 0 and 1, there are additional unique elements in the extension field that represented with a new symbol α, called the primitive element of the field. Each nonzero element in GF (2^n) can be represented by a power of α. To obtain the field GF (2^n), an infinite set of elements F is formed by starting with the elements $0, 1, \alpha$, and generating additional elements by progressively multiplying the last entry by α[4], which yields the following:

F= $0, 1, \alpha, \alpha^2, \alpha^3, ..., \alpha^j = 0, \alpha^0, \alpha^1, \alpha^2, \alpha^3, ..., \alpha^j$

To obtain the finite set of elements of GF (2^n) from F, a condition must be imposed on F so that it may contain only 2m elements and is closed under multiplication. The condition that closes the set of field elements under multiplication is characterized by the irreducible polynomial shown below[4]:

$$\alpha^{2m-1} + 1 = 0 or equivalently, \alpha^{2m-1} = 1 = \alpha^0 \tag{3.1}$$

Using this polynomial constraint, any field element that has a power equal to or greater than 2m1 can be reduced to an element with a power less than 2m1, as follows[4]:

$$\alpha^{2m+n} = \alpha^{2m-1}\alpha^{n+1} \tag{3.2}$$

Therefore, the elements of a finite field[4],

$$GF(2^n) = 0, \alpha^0, \alpha^1, \alpha^2, \alpha^3, ..., \alpha^j, ..., \alpha^{2n-1} \tag{3.3}$$

Obtaining the polynomial representation using a Primitive polynomial An irreducible polynomial f(X) of degree m is said to be primitive if the smallest positive integer n for which f(X) divides X^{n+1} is n = $2^m - 1$. A polynomial A divides B if the division yields a nonzero quotient and a zero remainder[4]. Also, the primitive element is a root of the primitive polynomial i.e. $f(\alpha)$.i.e. $f(\alpha) = 0$ This equation can be used to represent each element of the Galois Field as distinct polynomial of degree m1 or less as illustrated by the following example[4]: Let us take the example of GF (2^3) i.e. m=3, for which we have primitive polynomial f(X) = $1 + X + X^3$. In this case, we will have $2^3 = 8$ elements which can be represented as distinct polynomials of degree m - 1 = 2. Using the above relation by putting X = α, polynomial becomes[4], $\alpha^3 + \alpha + 1 = 0$ So,

$$\alpha^3 = \alpha + 1 \tag{3.4}$$

$$\alpha^4 = \alpha^3 \cdot \alpha = (\alpha + 1) \cdot \alpha = \alpha^2 + \alpha \tag{3.5}$$

Similarly,

$$\alpha^5 = \alpha^4 \cdot \alpha = (\alpha^2 + \alpha) \cdot \alpha = \alpha^2 + \alpha + 1 \tag{3.6}$$

$$\alpha^6 = \alpha^5 \cdot \alpha = (\alpha^2 + \alpha + 1) \cdot \alpha = \alpha^2 + 1 \tag{3.7}$$

$$\alpha^7 = \alpha^6 \cdot \alpha = (\alpha^2 + 1) \cdot \alpha = 1 = \alpha^0 \tag{3.8}$$

Since, $\alpha^7 = \alpha^0$, we have only 8 elements in GF (2^3)[4].

ENCODING PROCEDURE

- (n , k) = ($2^{(m-1)}, 2^{(m-1-2t)}$) = (7 , 3) double symbol error correcting RS code Where 2t = n - k is number of parity symbol and t is symbol error here 2t = 4 roots[4]
- Input message symbol sequence : 010 110 111
- Input symbol length : 3
- Generator Polynomial:

$$G(x) = (x - \alpha^1)(x - \alpha^2)(x - \alpha^3)(x - \alpha^4) \tag{3.9}$$

$$= \alpha^3 + \alpha^1 x + \alpha^0 x^2 + \alpha^3 x^3 + x^4 \tag{3.10}$$

$$= g_0 + g_1 x + g_2 x^2 + g_3 x^3 + ..[4] \tag{3.11}$$

Table 3.1: Step to Step Procedure of encoder with respect to clock [4]

Input			**Register Content**				**Feedback**	**O**
α^1	α^3	α^5	0	0	0	0	α^5	α^5
	α^1	α^3	$\alpha^5 \cdot \alpha^3 = \alpha^1$	$\alpha^5 \cdot \alpha^1 = \alpha^6$ $\alpha^6 + \alpha^0 = \alpha^6$	$\alpha^5 \cdot \alpha^0 = \alpha^5$ $\alpha^5 + 0 = \alpha^5$	$\alpha^5 \cdot \alpha^3 = \alpha^1$ $\alpha^1 + 0 = \alpha^1$	$\alpha^1 + \alpha^3 = \alpha^0$	α^5
		α^1	$\alpha^0 \cdot \alpha^3 = \alpha^3$	$\alpha^0 \cdot \alpha^1 = \alpha^1$ $\alpha^1 + \alpha^1 = 0$	$\alpha^0 \cdot \alpha^0 = \alpha^0$ $\alpha^6 + \alpha^0 = \alpha^2$	$\alpha^0 \cdot \alpha^3 = \alpha^3$ $\alpha^5 + \alpha^3 = \alpha^2$	$\alpha^2 + \alpha^1 = \alpha^4$	α^3
			$\alpha^4 \cdot \alpha^3 = \alpha^0$	$\alpha^4 \cdot \alpha^1 = \alpha^5$ $\alpha^5 + \alpha^3 = \alpha^2$	$\alpha^4 \cdot \alpha^0 = \alpha^4$ $\alpha^4 + 0 = \alpha^4$	$\alpha^4 \cdot \alpha^3 = \alpha^0$ $\alpha^0 + \alpha^2 = \alpha^6$		α^1
								α^6
								α^4
								α^2
								α^0

Table 3.1 shows the step to step encoding procedure with respect to the clock. It shows the Input sequence, register contents, Feedback and the Output at every clock cycle.

After the third clock cycle, the register contents are the four parity symbols, Then switch in RS encoder block is change its position and the parities symbols contained in the register are shifted to the output. So the output of the encoder will be[4],

$$U(x) = \alpha^0 + \alpha^2 x + \alpha^4 x^2 + \alpha^6 x^3 + \alpha^1 x^4 + \alpha^3 x^5 + \alpha^5 x^6 \tag{3.12}$$

3.5 Hardware Implementation in Xilinx FPGA using VHDL

The Hardware based CCSDS ReedSolomon Encoder is implemented in FPGA and described in VHDL. It is based on the linear feedback shift register implementation. VHDL Blocks The entire design, development is carried out using VHDL (Xilinx). The task has been divided into the following functional blocks to facilitate the modular implementation, and ease of simulation and final testing.

encoder_top.vhd: This is the top module of RSEncoder which contains many sub-modules:

- Test Stimulus module
- Data Encoder module.
- PRBS Generator
- Mux
- Clock Divider module
- Parallel to Serial Converter and Randomizer module

Test_Stimulus.vhd:

This module implements control logic for RS Encoder it generates controls signals like start, reset data input and data output registers data path as well as it generates input block sequence of rs encoder. For this Design and implementation, the ramp signal of 50 MHz is generated as an input of (255,223) RS encoder block design.

data_encoder.vhd:

Implements data path for RS Encoder in 32 LFSR architecture, which reduces the latency of RS Encoder to 3 clock pulses. Input and output message data are registered through input and output registers which ensure glitch free operation and both the registers are controlled by enable signals which avoid unnecessary switching of the data path and in turn saves power.

mux.vhd:

This module implements the parametric multiplexer that is used to implement switches in the data path. It selects either data or parities goes out and also selects either sync or encoded data goes out.

clk_divider.vhd:

This module divides the onboard clock frequency by eight because RS-Encoder performs byte operations and finally output will be serial.

parl_ser_conv.vhd:

This module converts byte of encoded data into the serial stream of bits.

DATA_shifter:

This module performs shift operation of serial data to manage the sync period.

rs_encode_pack.vhd:

The package that defines data types and functions for projects. This VHDL package has been developed to provide the RS encoder functions. It provides Galois addition and multiplication functions.

3.6 Schematic View

Figure 3.3 shows the schematic view of the data simulator. As shown in the figure, the clock is taken as an input of a data encoder from the onboard clock oscillator. It shows the implemented components for data simulator. and we get an encoded, randomized and frame synchronized serial data as an output.

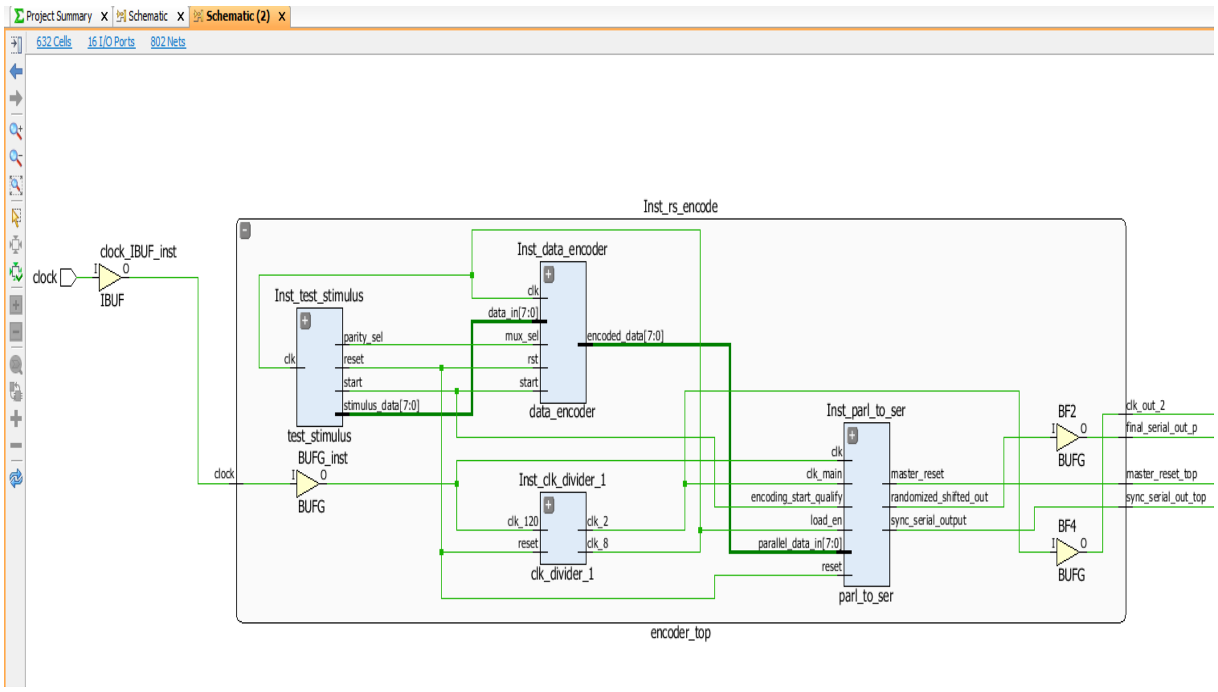


Figure 3.3: RTL View for Data Transmission Process

3.7 Device Utilization

Table 3.2 shows the available Devices on Zed board to implement our logic and how much are used in the implementation of Data Simulator.

Table 3.2: Device Utilization

Resources	**Utilization**	**Available**	**Utilization %**
LUT	334	53200	0.63
LUTRAM	2	17400	0.01
FF	399	106400	0.38
IO	29	200	14.50
BUFG	6	32	18.75

Chapter 4

Implementation of FPGA-Based CCSDS Specified Baseband Data Processing

4.1 Introduction

As a part of the design and development efforts towards the realization of the ground data reception systems for the future Remote Sensing Sensors, which will transmit the data in the CCSDS formats, the design and development were taken up for the individual hardware modules required to be implemented for the total data reception chain. CCSDS standards propose a packetized transmission scheme consisting of independent layers which separately process the data. This packetized data are attached with sync marker, randomized and encoded with CCSDS Prescribed processing. So the satellite data reception scheme consisting of the frame synchronization, de-randomization and the decoding of the transmitted data. For the encoding of the space data Forward error correcting codes have become commonplace in modern digital communications. These codes work by adding extra information (redundancy) to the original data. The encoded data can then be stored or transmitted. When data is recovered it may have errors introduced, because of radio frequency interference noise in the transmission channel. The added redundancy allows a decoder (with certain restrictions) to detect which parts of the received data are corrupted and correct them. The number of errors the code can correct depends on the amount of redundancy added. As a part of the design and development efforts towards the realization of the ground data reception systems for the future Remote Sensing Sensors, which will transmit the data in the CCSDS formats, the design and development were taken up for the individual hardware modules required to be implemented for the total data reception chain. The RS decoder module is one such element which is required to realize the error correction before the formation of the actual information/data packets can be done.

4.2 Frame Synchronization

The Frame Synchronization is one such module which is essential to synchronize the data frame before the actual decoding procedure starts[1]. At the transmitter side, it has attached the some specified known bit pattern to synchronize the data frame so at the ground data reception system it should be identified and removed for the further processing of the actual data. For that digital correlator is designed and implemented using VHDL[1].

4.3 De-Randomization

In order for the receiver system to work properly, every data capture system at the receiving end requires that the incoming signal has sufficient bit transition density, and allow proper synchronization of the decoder[1]. The data stream is sufficiently random. The method for ensuring sufficient transitions is to exclusive-OR each bit of the Received Frame with a standard pseudo-random sequence.The pseudo-randomizer is used, on the sending end so it is required to de-randomize the data after Frame synchronization but before Reed-Solomon decoding.The pseudo-random sequence is applied starting with the first bit of the Frame. Implementation of the standard sequence is same as done for transmitting side[1].

4.4 RS-Decoding Algorithm

The RS decoder module is one such element which is required to realize the error correction before the formation of the actual information/data packets can be done. The design, development, and realization of RS decoder are implemented in FPGA using the hardware description language (VHDL). A top-down approach has been followed while describing the functionality of the module.The desired parameters of the Reed-Solomon decoder are code word length, error correcting capability, the initial root of the code generator polynomial, the size of the Galois field and the field generator polynomial. The decoding procedure consists of the following steps

- **Syndrome Calculation:**

  The first step is the generation of syndromes from the received input symbols. These are obtained by evaluating the polynomial representation of the input received symbols by the roots of the generator polynomial[5]. In the case of no error, the syndromes will be zero while non-zero syndromes indicate the presence of an error in the codeword. The syndromes depend only on the errors, not on the underlying encoded data. Syndrome calculation can be done by an iterative process, such that the answer (2t syndrome symbols) is available as soon as the last parity has been read in. The circuit below will generate the i^{th} syndrome[5]. To correct t errors, t error locations and t error magnitudes are to be found i.e. 2t unknowns. The syndromes actually represent polynomial equations of t error locations and t error values[5]. So to find 2t unknowns, these 2t simultaneous equations are to be solved. Figure 4.1 shows the hardware implementation of the syndrome generation block to know whether received data is satisfying the roots of the input polynomial or not.

Working principle: The syndrome generator module is made of 2t syn-

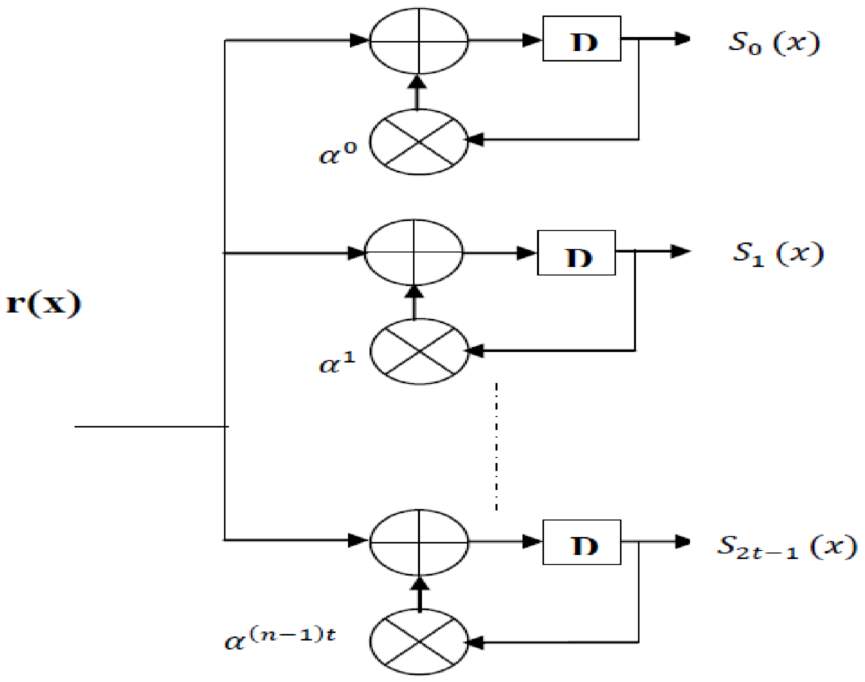


Figure 4.1: Block Diagram of Syndrome Generation[5]

drome cells into all of which the received code symbols are fed as input. After symbols have been input, the final syndromes are available in the 2t registers[5]. The syndrome cells calculate the syndromes in an iterative fashion, which can be described as follows:

for j= 1 to n, $S_j = (S_{j-1}+r_j)*$ 2t such syndrome cells generate 2t syndromes which are used by the Key Equation Solver Module to generate the error locator and evaluator polynomials[5].

- **Error locator Polynomial:**

The next step, after the computing the syndrome polynomial is to calculate the error values and their respective locations. This stage involves the solving of the 2t syndrome polynomials, formed in the previous stage[5]. These polynomials have 'v unknowns, where v is the number of unknown errors prior to decoding. If the unknown locations are ($i_1, i_2, i_v,$) the error polynomial can be expressed as[5],

$$e(x) = Y_1 x^{1i} + Y_2 x^{2i} + Y_3 x^{3i} + ... + Y_v x^{vi} \tag{4.1}$$

where Y_1 is the magnitude of the 1^{th} error location. If X^1 is the field element associated with the error location 1, then the syndrome coefficients are given by[5],

$$S_j = \sum_{t=1}^{v} Y_l X_l^t \tag{4.2}$$

Where, j=1,2,.,2t. And Y_l is the error value and X_l is the error location of the lth error symbol. The expansion of gives the following set of 2t equations in the v unknown error locations $X_1, X_2, ...X_v$ and v unknown error magnitudes $Y_1, Y_2, ...Y_v$[5].

S_1(x) $= Y_1 x_1 + Y_2 x_2 + Y_3 x_3 + ... + Y_v x_v$

S_2(x) $= Y_1 x_1^2 + Y_2 x_2^2 + Y_3 x_3^2 + ... + Y_v x_v^2$

.

.

.

S_{2t}(x) $= Y_1 x_1^{2t} + Y_2 x_2^{2t} + Y_3 x_3^{2t} + ... + Y_v x_v^{2t}$

(4.3)

The above set of equations must have at least one solution because of the way the syndromes are defined. This solution is unique. Thus the *decoders* task is to find the unknowns given the syndromes[5]. This is equivalent to the problem in solving a system of non-linear equations. Clearly, the direct solution of the system of nonlinear equations is too difficult for large values of v. Instead, intermediate variables can be computed using the syndrome coefficients S_j

from which the error locations, $X_1, X_2, ..., X_v$, can be determined. The error-locator polynomial is introduced as,σ(x) = 1 + σ_1x + $\sigma_2 x^2$ +...+$\sigma_v x^v$ [5] The polynomial is defined with roots at the error locations 1 i.e X_l^{-1} for l=1,2,v. The error location numbers l, X indicate errors at locations i_l for l=1, 2, v This can be written as[5],

$$\sigma(x) = (1 - xX_1)(1 - xX_2)....(1 - xX_v) \tag{4.4}$$

The process of solving the simultaneous equations consists of two stages[5]. First, the error locator polynomial σ is calculated, the roots of which are the error locations and then the error evaluator polynomial Ω are calculated, which helps in calculating the error values at each of the error locations, in the error evaluation stage[5]. There are several methods of finding the error locator polynomial σ, the two most popular are Euclids algorithm (easier to implement) and the Berlekamp-Massey algorithm (more efficient use of hardware resources)[5].The Berlekamp-Massey algorithm iteratively solves the error locator polynomial σ by solving one equation after another and updating the error locator polynomial σ. If it turns out that it cannot solve the equation at some step, then it computes a discrepancy to correct the error locator polynomial, thereby increasing the size of the polynomial, and iterates again[5]. A maximum of 2t iterations are required. For n symbol errors, the algorithm gives a polynomial with n coefficients[5]. At this point the decoder fails if there are more than t errors, and no corrections can be made. It will introduce more errors than there were originally[5].

Working Principle

Inversion less Berlekamp-Massey Algorithm

Initial Condition:

Where, σ is the error locator polynomial, $\tau(x)$ is the error locator support polynomial, $\omega(x)$is the error evaluator polynomial, $\Delta^(i)$ is the ith step discrepancy, δ is the previous nonzero discrepancy, D is the degree of the error locator polynomial[5]. $D^{-1} = 0, \delta = 1, \delta^{-1}(x) = \tau^{-1}(x) = 1, \Delta^(0) = S_1$

for (i=0 to 2t-1)

$$\sigma^{(i)}(x) = \delta\sigma^{(i-1)}(x) + \Delta^{(i)} x\tau^{(i-1)}(x), \tag{4.5}$$

$$\Delta(i+1) = S_{i+2}\sigma_0^{(i)} + S_{i+1}\sigma_1^{(i)} + ... + S_{i-t+2}\sigma_t^{(i)} \tag{4.6}$$

if $(\Delta^{(i)} = 0 or 2D^{(i-1)} \geq i+1)$ then

$$D^{(i)} = \Delta^{(i-1)}, \tau^i(x) = x\tau^{(i-1)}(x) \tag{4.7}$$

else

$$\Delta^{(i)} = i+1-\Delta^{(i-1)}, \delta = \Delta^{(i)}, \tau^i(x) = \sigma^{(i-1)}(x) \tag{4.8}$$

Finding the Error locator Polynomial Roots by Chien Search Once the error locator polynomial is known, we can find the location of the errors in the re-

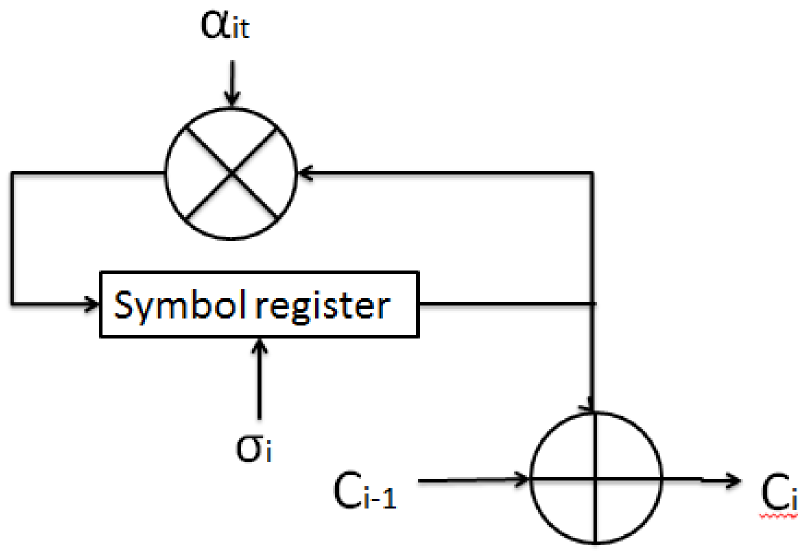


Figure 4.2: Block Diagram of Chain Search[5]

ceived symbol block by evaluating the roots of the polynomial[5]. The most commonly used algorithm for this is the Chien search which employs exhaustive search of the roots among the n possible error locations. Figure 4.2 shows the hardware implementation block for the chien search algorithm[5].

- **Finding the Error Magnitudes:**
  Once, the error locations are known, the next step is to use the syndromes and the error polynomial roots to derive the error values. This is usually done using the Forney algorithm. This algorithm is an efficient way of performing a matrix inversion. The algorithm first calculates an error evaluator polynomial ω by convolving the syndromes with the error locator polynomial σ (from the Berlekamp-Massey results). ω is then evaluated at each error location, and divided by the derivative of σ to give the error symbol at that location[5]. To correct the received symbols, the symbols are read again from an intermediate store, and at each error location, the received symbols are subtracted (by performing XOR operation) with the error symbol. The parity symbols are usually stripped off[5].

Decoding Procedure

- Earlier a test message encoded using (n , k) = $(2^{(m-1)}, 2^{(m-1-2t)})$ = (7 , 3) double symbol error correcting RS code
  Where 2t = n - k is number of parity symbol and t is symbol error here 2t = 4 roots

- Now, assume that during transmission this code word becomes corrupted so that two symbols are received in error[4]. For this example let the error pattern e(x) be such that,

$$e(x) = 0 + 0x + 0x^2 + \alpha^2 x^3 + \alpha^5 x^4 + 0x^5 + 0x^6 \tag{4.9}$$

- Received message symbol sequence

$$r(x) = U(x) + e(x) = \alpha^0 + \alpha^2 x + \alpha^4 x^2 + \alpha^0 x^3 + \alpha^6 x^4 + \alpha^3 x^5 + \alpha^5 x^6 \tag{4.10}$$

- **syndrome calculation:** The syndrome is the results of parity check performed on r to determine whether r is a valid member of the codeword set[4]. If in fact r is a member, the syndrome S has value 0. Any nonzero value of S indicates the presence of errors. S is made up of n-k symbols.

- U(x)=m(x)g(x) from this structure it can be seen that every valid codeword is multiple of generator polynomial so the roots of g(x) must also be the roots of U(x).

- For this example the four syndrome are found as follows[4]:

$$S_1 = e_1\beta_1 + e_2\beta_2 = r(\alpha) = \alpha^3, \tag{4.11}$$

$$S_2 = e_1\beta_1^2 + e_2\beta_2^2 = r(\alpha^2) = \alpha^5, \tag{4.12}$$

$$S_3 = e_1\beta_1^3 + e_2\beta_2^3 = r(\alpha^3) = \alpha^6, \tag{4.13}$$

$$S_4 = e_1\beta_1^4 + e_2\beta_2^4 = r(\alpha^4) = 0 \tag{4.14}$$

  the results says that the received codeword contains an error.

- **Error Location:** Once a nonzero syndrome vector has been computed, that signifies that an error has been received. Next we have to find error location of error[4]. An error-locator polynomial,

$$\sigma(x) = 1 + \sigma_1 x + \sigma_2 x^2 + ... + \sigma_v x^v \tag{4.15}$$

  There are 2t unknowns (t error value and t error locations), and 2t simultaneous equations[4],

- we form the matrix from the syndrome using autoregressive method

$$\begin{pmatrix} S_1 & S_2 \\ S_2 & S_3 \end{pmatrix} \begin{pmatrix} \sigma_2 \\ \sigma_1 \end{pmatrix} = \begin{pmatrix} S_3 \\ S_4 \end{pmatrix} \tag{4.16}$$

$$\begin{pmatrix} \alpha^3 & \alpha^5 \\ \alpha^5 & \alpha^6 \end{pmatrix} \begin{pmatrix} \sigma_2 \\ \sigma_1 \end{pmatrix} = \begin{pmatrix} \alpha^6 \\ 0 \end{pmatrix} \tag{4.17}$$

We begin our search for the error location by solving for the coefficient of the error-locator polynomial,σ(x)[4].

$$\begin{pmatrix} \sigma_2 \\ \sigma_1 \end{pmatrix} = \begin{pmatrix} \alpha^1 & \alpha^0 \\ \alpha^0 & \alpha^5 \end{pmatrix} \begin{pmatrix} \alpha^6 \\ 0 \end{pmatrix} \tag{4.18}$$

From equation 4.17,

$$\sigma(x) = \alpha^0 + \sigma_1 x + \sigma_2 x^2 = \alpha^0 + \alpha^6 x + \alpha^0 x^2 \tag{4.19}$$

- Testing of σ(x) polynomial with each of the field elements, $\sigma(\alpha^0) = \alpha^6 \neq 0$, $\sigma(\alpha^1) = \alpha^2 \neq 0$, $\sigma(\alpha^2) = \alpha^6 \neq 0$, $\sigma(\alpha^3) = 0$, $\sigma(\alpha^4) = 0$, $\sigma(\alpha^5) = \alpha^2 \neq 0$, $\sigma(\alpha^6) = \alpha^0 \neq 0$,[4]

  The error locations are at the inverse of the roots of the polynomial so error locations are 3 and 4.

- **Error Value:** we have to use two equation from any of four equations, let use S_1 and S_2. so[4],

$$\begin{pmatrix} \beta_1 & \beta_2 \\ \beta_1^2 & \beta_2^2 \end{pmatrix} \begin{pmatrix} e_1 \\ e_2 \end{pmatrix} = \begin{pmatrix} S_3 \\ S_4 \end{pmatrix} \tag{4.20}$$

$$\begin{pmatrix} \alpha^3 & \alpha^4 \\ \alpha^6 & \alpha^8 \end{pmatrix} \begin{pmatrix} e_1 \\ e_2 \end{pmatrix} = \begin{pmatrix} \alpha^3 \\ \alpha^5 \end{pmatrix} \tag{4.21}$$

- From the equation the error polynomial is ê(x)=$\alpha^2 x^3$+$\alpha^5 x^4$ by adding error polynomial to the received polynomial it repairs the received polynomial and correct the received data and finally delivers the actual data[4]. That is,

  U(x) = r(x) + ê(x) =α^0 + α^2x + $\alpha^4 x^2$ + $\alpha^0 x^3$ + $\alpha^6 x^4$ + $\alpha^3 x^5$ + α^5 x^6 + $\alpha^2 x^3$ + $\alpha^5 x^4$ = α^0 + α^2x + $\alpha^4 x^2$ + α^6 x^3 + $\alpha^1 x^4$ + $\alpha^3 x^5$ +$\alpha^5 x^6$ [4]

4.5 Hardware Implementation in Xilinx FPGA using VHDL

The Hardware based CCSDS ReedSolomon Decoder is implemented in FPGA and described in VHDL. It is based on the Syndrome Generation, Chain Searcher, RS Decoder, Error Location and Error Evaluator polynomial implementation.
VHDL Blocks The entire design, development is carried out using VHDL (Xilinx). The task has been divided into the following functional blocks to facilitate the modular implementation, and ease of simulation and final testing.

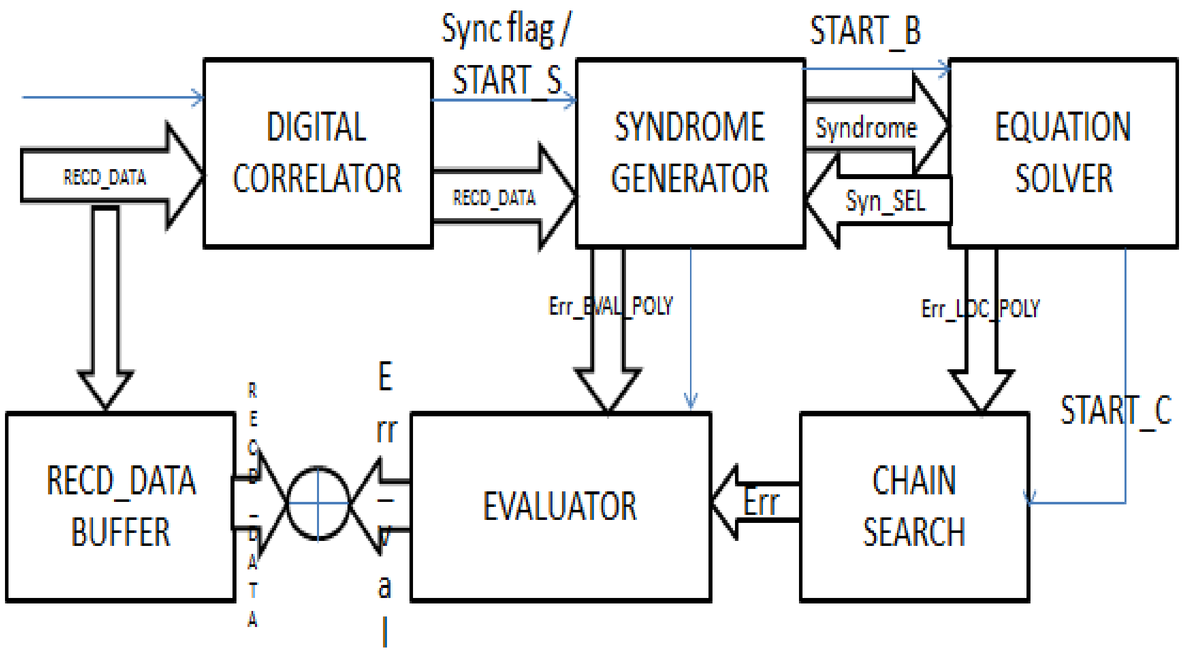


Figure 4.3: Block Diagram of Data Reception Process

Figure 4.3 shows the block diagram for the data reception system. It shows the data flow from one block to other and the generated output signals after completion of the every block. Description of every block are as below,
decoder_top.vhd: This is top module of RS-Decoder which contains many sub modules:

- Correlator Module
- PRBS Generation Module
- Syndrome Generation Module
- Error locator Module
- Chain Searcher Module
- Error Evaluator module
- clock divider module

Correlator.vhd:

Correlator module is implemented to synchronize the data in data reception system. It will find the sync from the received data using the correlation between sync and incoming data and it will count the number of one in the match if there is a match of minimum 29 bit it will generate one sync flag at the end of the sync and it will be treated as a start pulse of a syndrome generation module.

PRBS_Generator.vhd:

After getting the start pulse from the correlator Pseudo Random Sequence will start its generation as per the CCSDS standard specification and the encoded data will be De randomize this Pseudo Random Sequence. Simultaneously De-randomize Serial data will be converted into the byte of eight bit.

Syndrome_Generation.vhd:

The Parallel Byte of De-randomized data is latched and it will go to the Syndrome Generation Block. In this module, the roots of the encoded data are put down in the received data if the roots will satisfy the polynomial that means there is no error and the syndromes are zero otherwise we will get some non-zero value of syndromes. It will take the 255 cycles after that it will generate end flag.

Error_Locator.vhd:

This module will find the error location polynomial using the Berlekamp-Massey algorithm if we get the non-zero value of syndrome and the end flag of syndrome module is treated as a start of the error locator module.

Chain_Searcher.vhd:

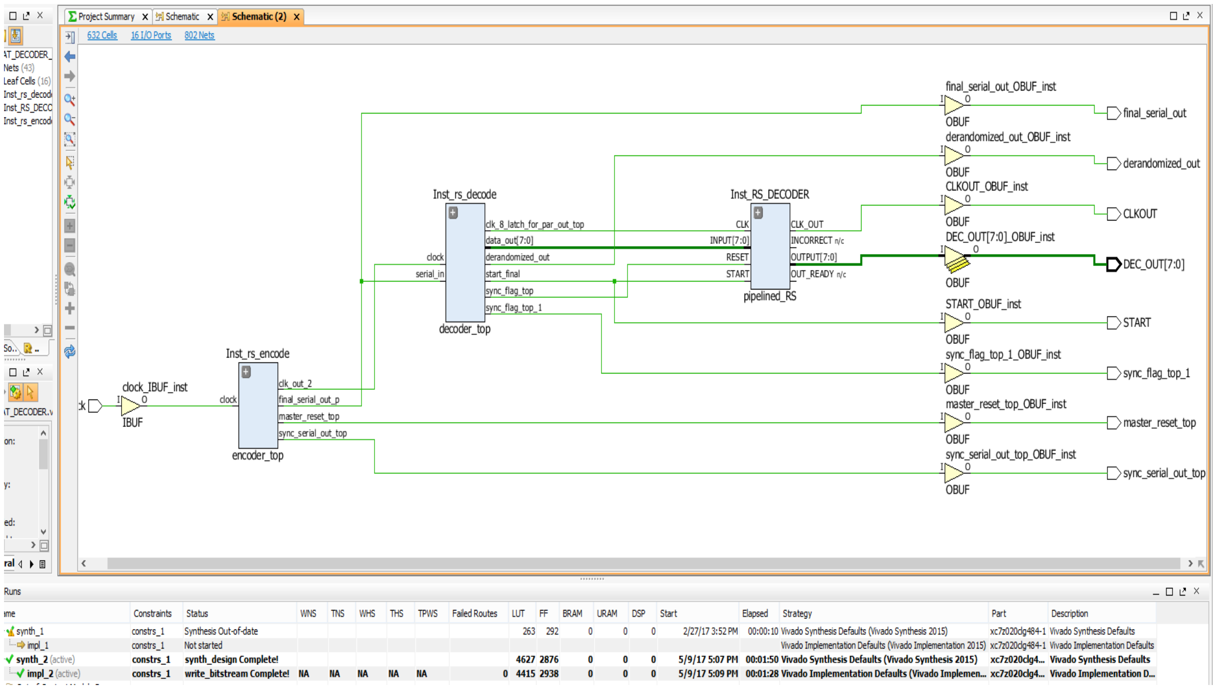
This module is used to find the roots of the error locator polynomial. The roots of the polynomials are the actual location of the error. After the completion of the process of finding the roots of the polynomial, it will give the end of the flag after the 255 cycles.

Error_Evaluator.vhd:

This module will find the error evaluator polynomial using the same hardware which is used for the error locator module. chain search mode will find the roots of the Error evaluator polynomial. It is at the place of error location. After that, if we ex-or with the original message symbol it will give the corrected data.

4.6 Schematic View

Figure 4.4 and Figure 4.5 shows the schematic view of the data reception. As shown in the figure, the encoded data is taken as an input of a data receiver system. It shows the implemented components for synchronization, de-randomization and error correction block for data decoding procedure. we get decoded data as an output.



ame	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Failed Routes	LUT	FF	BRAM	URAM	DSP	Start	Elapsed	Strategy	Part	Description
synth_1	constrs_1	Synthesis Out-of-date							263	292	0	0	0	2/27/17 3:52 PM	00:00:10	Vivado Synthesis Defaults (Vivado Synthesis 2015)	xc7z020clg484-1	Vivado Synthesis Defaults
impl_1	constrs_1	Not started														Vivado Implementation Defaults (Vivado Implementation 2015)	xc7z020clg484-1	Vivado Implementation Defaults
synth_2 (active)	**constrs_1**	**synth_design Complete!**							**4627**	**2876**	**0**	**0**	**0**	**5/9/17 5:07 PM**	**00:01:50**	**Vivado Synthesis Defaults (Vivado Synthesis 2015)**	**xc7z020clg4...**	**Vivado Synthesis Defaults**
impl_2 (active)	**constrs_1**	**write_bitstream Complete!**	**NA**	**NA**	**NA**	**NA**	**NA**	**0**	**4415**	**2938**	**0**	**0**	**0**	**5/9/17 5:09 PM**	**00:01:28**	**Vivado Implementation Defaults (Vivado Implemen...**	**xc7z020clg4...**	**Vivado Implementation D...**

Figure 4.4: Schematic View for Data Reception Process

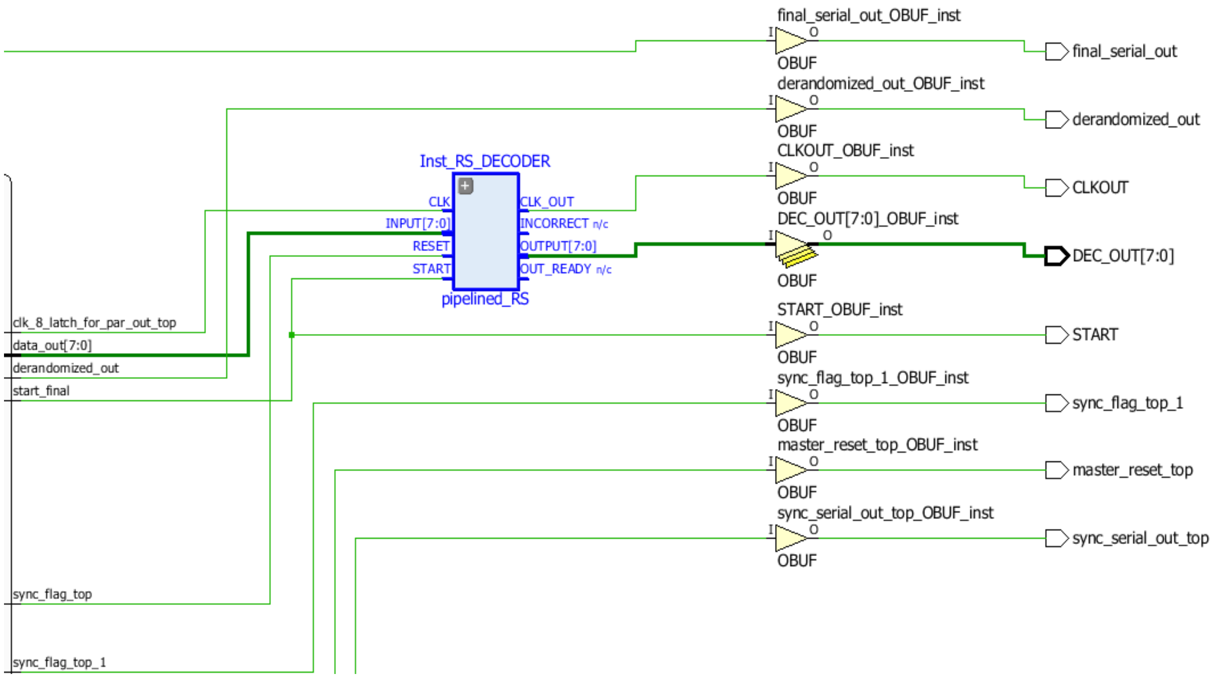


Figure 4.5: Schematic View for Data Reception Process

4.7 Device Utilization

Table 4.1 shows the available Devices on Zed board to implement our logic and how much are used in the implementation of Data Reception System. As we know that data reception system is complicated than data formation so the device utilization is also very high then data formation unit.

Table 4.1: Device Utilization

Resources	**Utilization**	**Available**	**Utilization %**
LUT	4415	53200	8.30
LUTRAM	223	17400	1.28
FF	2938	106400	2.76
BRAM	0.50	140	0.36
IO	16	200	8.00
BUFG	6	32	18.75

Chapter 5

Hardware Based Data Acquisition System

5.1 Introduction

A Communication system is meant to transport the information from source to user destination via preferably a reliable communication channel. But that is not always the case. However, error free communication can still be achieved even through a non-ideal channel, by means of certain techniques. Before data transmission, the encoder attaches parity symbols, does randomization and synchronization using the pre-determined and prescribed algorithm. At the receiving side, the major tasks are to synchronize, de-randomize and decoding of the received data. Subsequent to the above processing, the data is still required to be transferred to host at real time and on sustained basis for implementation of an operational reception system.
The performance of the communication can be increased by performing the compute-intensive tasks on the FPGA, as all the processing load required to decode and transmit the data may not be possible without a very high-end processor system. So, it is preferable to do it on hardware and can be achieved through parallelism with control and monitoring by the processor. This idea will bring us to the requirement

of the combination of programmable logic and the processing system on the same platform.

Such a requirement can be met by Zynq device, which is intended for the wide variety of application as a flexible and is a compelling platform. Just same as metal zinc, that can be mixed with various other metals to form alloys with differing desirable properties, single silicon chip can be used to implement the functionality of an entire system. A specific flavor of SoC is implemented on a programmable, reconfigurable device. The natural solution has long been the FPGA. FPGAs are inherently flexible devices that can be configured to implement any arbitrary system, including embedded processors if needed.

As per my requirement, it performs the CCSDS prescribed processing on the pro-

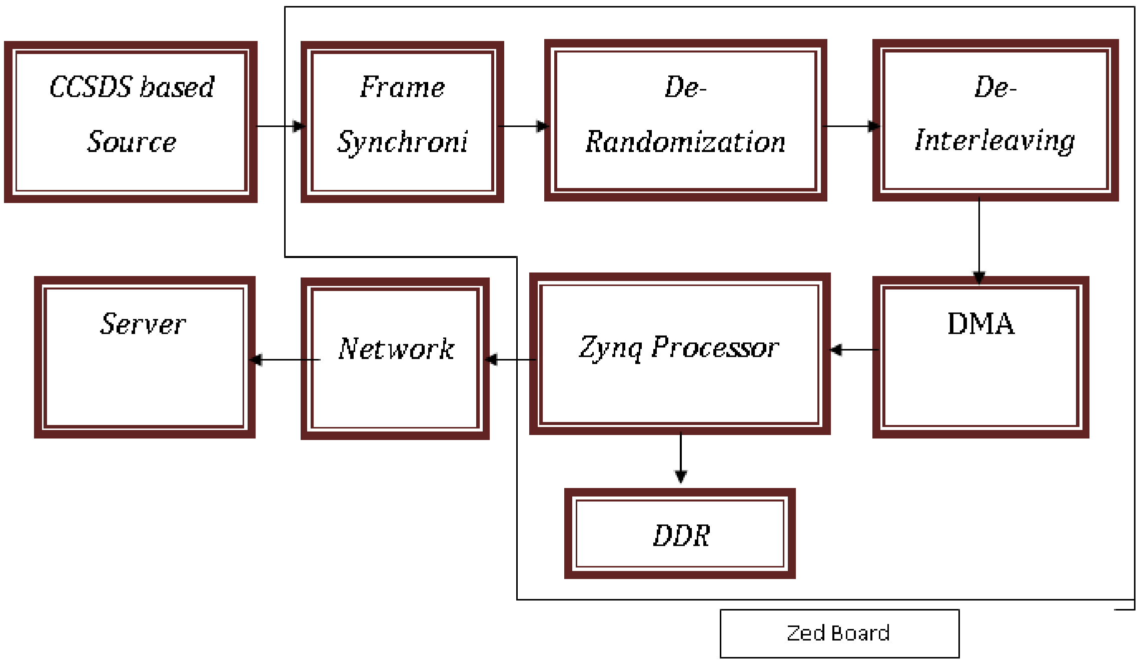


Figure 5.1: Block diagram

grammable logic of the Zed board and transfers the processed data to the memory of the processing system for further use. But in the continuous data transfer hardware will interrupt the processor every time and it should be served by the processor, as it will take multiple cycles of the processor. To overcome the drawback it should be transferred by the DMA (Direct Memory Access) in which processor access it separately via DMA. The data so acquired in the memory can then be streamed out to a host over 1G Ethernet provided on the Zed board.

5.2 Embedded Hardware Design and Interface with Zynq Processor

Embedded Hardware and the software was implemented on the real-time platform like Zedboard which can receive the data from the outside source. The received data can be processed on FPGA and the processor can transmit the data to the host. To meet the requirement whole system is developed on Vivado IDE. Zynq processor is the centralized processor which is hard coded on the board. For the system, the communication between the processor and the peripherals are on AXI(Amba Extensible Interface) bus. So the all the components are AXI. Here AXI DMA is used for the direct data transfer to the memory. The peripherals used in the system are programmed by the zynq processor. Linux is taken as an operating system of the processor.All the blocks which will be used in the system are described below.

5.2.1 AXI Standard

On any SoC, there are many components like UART, Memory, Ethernet, CPU, ADC/DAC etc.So, the major task is how they are connected to each other, how they communicate with each other.There is a kind of interface that all of the units talk to each other it is a bus which obeys a set of rules as per the design.There are many SoC buses like IBM core connect, Wishbone, AXI etc. Here in Zynq SoC it obeys AXI Standard[2].

AXI Interfaces

The transaction of data from one point to another point on the hardware is possible by AXI Interfaces. Transactions are either read or write. The module which initiates the transaction is called AXI master and the module which receives and respond to the transaction is called AXI slave. So it is a Point-to-point connection for passing data, addresses, and hand-shaking signals between master and slave clients within the system[2]. Master should send set of command for example if read transaction it should send read address to the slave and slave provides the read data and read the response to the master. for write transaction master should send write address and write data to the slave and slave provide write the response to the master. There are two type of Interface Stream Interface and Memory mapped Interface[2].
Stream interface: Stream Interface has No addresses just data is flowing one block to other. It has only two signals to initiate the transaction for read and write transaction. READY and VALID signal[2]. Memory mapped interface: In memory mapped interface there is address allocation to every slave. It have many signals for transaction like AWVALID, AWLEN, AWREADY, AWID, AWADDR etc..for write transaction and ARVALID, ARREADY, ARID, ARADDR etc. are for reading transaction[2].

AXI Interconnect

An interconnect is effectively a switch which manages and directs traffic between attached AXI interfaces. The connections between these interconnects are also formed using AXI interfaces.One master and slave talk to each other with AXI Interconnect.It is a unit which connects two master and slave[2]. It is a slave device for AXI master and Master device for AXI slave.The possible connection is up to 16 Master and Slaves through one AXI interconnect.It will be Slave device for AXI Master and Master device for AXI Slave.It should do the transaction according to address the range of AXI Slave.It can also perform Width Conversion, Clock Domain Transfor-

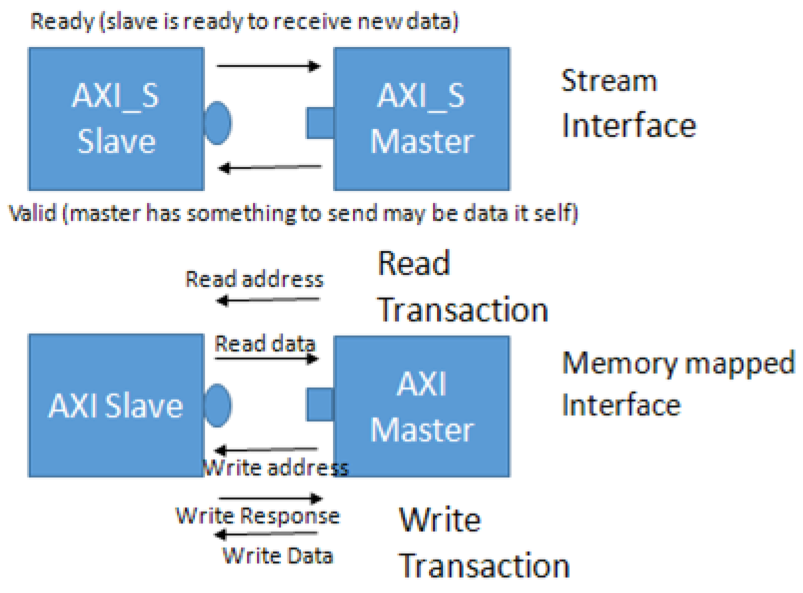


Figure 5.2: AXI Signal for Transaction

mation etc..[2].
AXI interconnect has to address decoding table when any master wants to initiate the transaction the interconnect first look at the address of the transaction and then puts the transaction to the specific slave. AXI interconnect is responsible for connecting multiple masters and slave also useful for width conversations and also responsible for clock domain transaction. Also, Contains Register Slices (To improve performance)[2]. Figure 5.2 shows the read and write transaction signal for the stream and memory map interface.

5.2.2 AXI DMA

Direct Register Mode

Direct Register mode of DMA provides a lower performance but it uses less FPGA resource. It performs data transfer by providing destination address and the length of the transfer to the DMA controlled register[11]. Suppose a data stream is entering to a system or our DMA engine and CPU defines the transfer task for each packet meaning it tells the DMA engine where the packet should go. So after sending every single packet, DMA engine interrupts the CPU for next transfer task. So after every completion of task CPU reprogram the DMA[11]. Figure 5.3 shows the DMA controlled register in the direct register mode which will be programmed for the operation of the DMA.

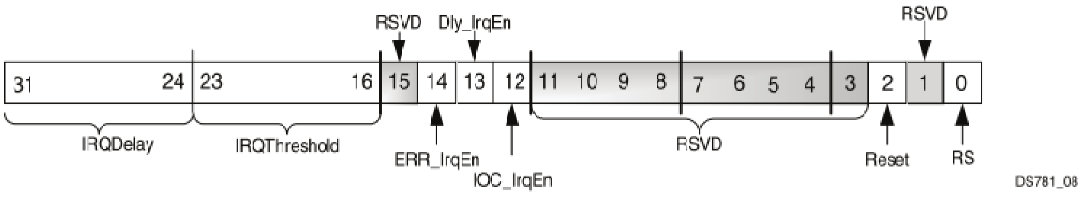


Figure 5.3: DMA Controlled Register in Direct Register Mode[11]

Scatter Gather Mode

The AXI DMA provides the high bandwidth direct memory access between the AXI interfaces and it will support the scatter-gather mode, which offload the CPU from the data movement tasks. Suppose the data stream is entering the system from the source, the source can be any data from USB, PCI slot or any connectors here in my implementation the source is sample generator[11]. As per the application data should be processed and to be stored somewhere, can be dram memory of processor so the data stored in memory but the memory blocks are not necessarily continuous it can be distributed across the entire physical memory[11]. So basically our DMA engine or DMA controller or any subsystem receives the data which are coming and scattering it in different physical locations after entering to a system and processed. But in reverse direction when the data stored in different physical memory and want to gather the data, DMA controller will gather the data from

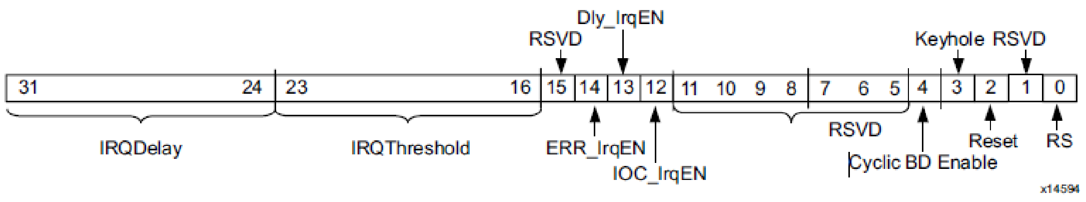


Figure 5.4: DMA Controlled Register in Scatter Gather Mode[11]

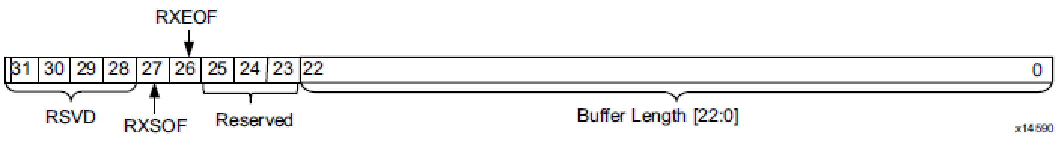


Figure 5.5: Controlled Register for Scatter Gather Discriptor[11]

different location and put it together and make a stream of data. Here the header of data is coming from DRAM memory but body of the packets are coming from actual physical memory[11]. Figure 5.4 shows the controlled register in the scatter gather mode for the DMA to program and work. Figure 5.5 shows the controlled register for the discriptor which provides the information about the data transfer length through each discriptor and it also tells the processor whether the transfered data has start of frame or end of frame or both.

Chapter 6

Design Implementation of Data Acquisition System

The data acquisition system for the satellite data communication system is implemented in the two ways. (1) Using direct register mode of DMA, in this the specific length of the data can be transferred to the memory and after that, we have to reprogram the DMA for the further data transfer. So it is not a continuous data transfer. (2) Using the scatter-gather mode of DMA, in this mode, the continuous data can be transferred to the memory via the each descriptor. descriptor are defined in the block RAM and the processor just tells the DMA to perform the operation as per the descriptor.

6.1 DMA in Direct Register Mode

Figure 6.1 represents the generalized block diagram of the data acquisition system which will be implemented on the hardware for the DMA in direct register mode. AXI IP of Sample Generator is implemented using VHDL which generates the continuous data after we provide the enable signal via the AXI GPIO using the processor.

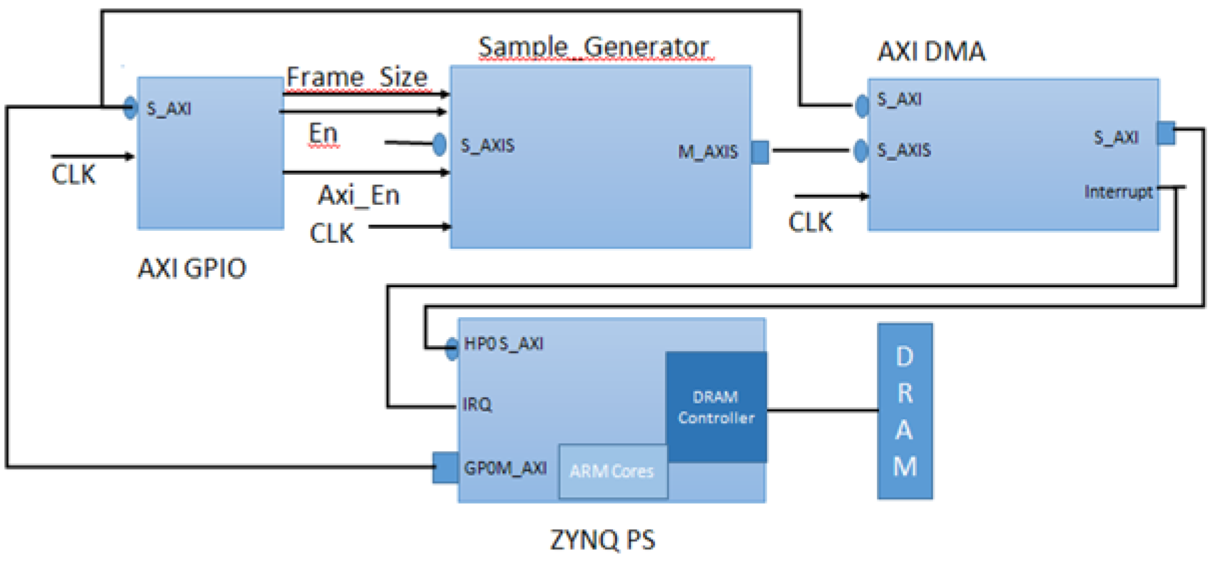


Figure 6.1: Generalized Block diagram for DMA in Direct Register Mode

6.1.1 Programmin Sequence for booting the linux with our hardware on Zynq Processor

Programming Sequence:

- Create a hardware as shown in Figure 6.2 in Vivado. Follows the basic steps for creating it.
- Generate a bitstream for hardware. From the file menu exports the hardware, include bitstream and then launch the SDK[3].
- Set the jumper for the booting Linux from the SD card.
- In SDK first the FSBL(First Stage Boot Loader) will need to be created.
- Now we have to prepare an SD card for booting the Linux from that which contains BOOT.bin, devicetree.dtb, ramdisk.image.gz, zimage and zmage.bin. BOOT.bin : Which is created by zynq boot image utility under a xilinx tools[3].

BOOT.bin includes FSBL, generated in SDK, system_wrapper.bit was exported to SDK and U-boot.elf provided with the software[3].

- Copy all the above files to the SD card and and insert to the SD card slot on the board. Now switch on the board and Linux will be booted on the processor any terminal can show the booting process.

6.1.2 Programming Sequence for DMA in Direct Register Mode

Programming Sequence:

- Define the base address of DMA and GPIO, destination address and the offset of the DMA registers before main function.
- In the main define the pointer which will be used to assign a virtual address to all the components of the hardware.
- Using a mmap function map a physical address of a components to the virtual address.
  follow the below sequence for programming the DMA and program the virtual address for all the register and buffer using pointer.
- Start the S2MM channel running by setting the run/stop bit to 1 (S2MM DMACR.RS = 1)[11].
- Enable interrupts by writing a 1 to S2MM DMACR.IOC IrqEn and S2MM DMACR.Err IrqEn[11]. NOTE: The delay interrupt, delay count, and threshold count are not used when the AXI DMA is configured for Simple DMA mode[11].
- Write a valid destination address to the S2MM DA register[11].
- Write the length in bytes of the receive buffer in the S2MM LENGTH register[11].

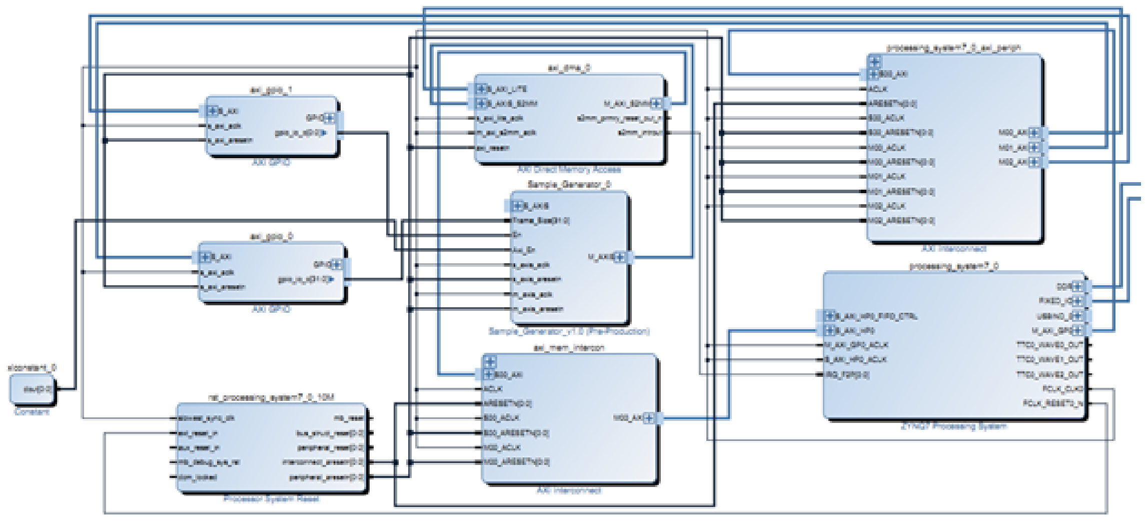


Figure 6.2: Implementation Block Diagram for DMA in Direct Register Mode

- NOTE: A value greater than or equal to the largest received packet must be written to S2MM LENGTH. A receive buffer length value written that is less than the number of bytes received produces undefined results[11]. Figure 6.2 shows the hardware blocks, interface and the interconnection for DMA in direct register mode which was developed in Vivado.

6.2 DMA in Scatter Gather Mode

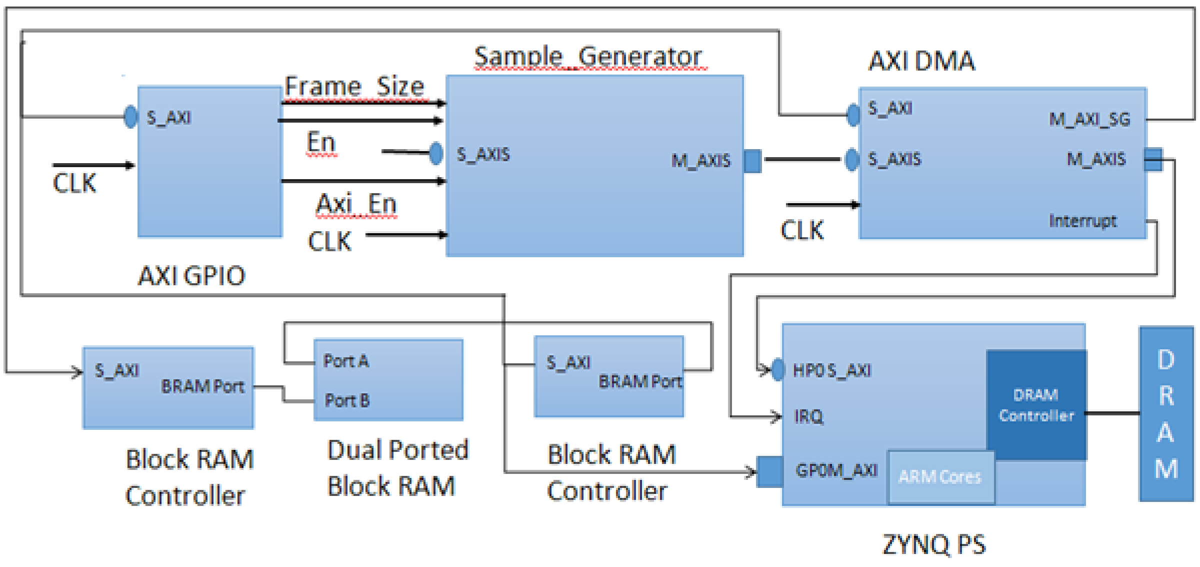


Figure 6.3: Generalized Block Diagram for DMA in Scatter Gather Mode

6.2.1 Programming Sequence for DMA in Scatter Gather Mode

Programming Sequence:
AXI DMA in scatter gather mode requires a memory it consists the list of operations which should be performed. Operation begins with the setting up a control register and descriptor pointer[11].

- Define the base address of DMA,Descriptor,GPIO, destination address. Define the offset of the DMA registers before main function.
- In the main define the pointer which will be used to assign a virtual address to all the components of the hardware.
- Using a mmap function map a physical address of a components to the virtual address.

follow the below sequence for programming the DMA and program the virtual address for all the register and buffer using pointer.

- Write the address of the starting descriptor to the current descriptor register[11].
- Start the S2MM channel running by setting the run/stop bit to 1 (S2MM DMACR.RS = 1)[11].
- Enable interrupts by writing a 1 to S2MM DMACR.IOC IrqEn and S2MM DMACR.Err IrqEn[11].
- Write a valid address to the tail Descriptor register meaning next descriptor pointer or in the case of only one descriptor tail descriptor is same as current descriptor pointer[11].

Programming of descriptor field

- Write the address of the next descriptor to the current descriptor register[11].
- Write the buffer address at the eighth offset of the descriptor pointer[11].
- Write the buffer length and the control field like start of frame and end of frame flag at the eighteenth offset of the descriptor pointer[11].

Figure 6.5 shows the hardware blocks used for the DMA in scatter-gather mode and the Figure 6.6 shows the same blocks implemented on Vivado.

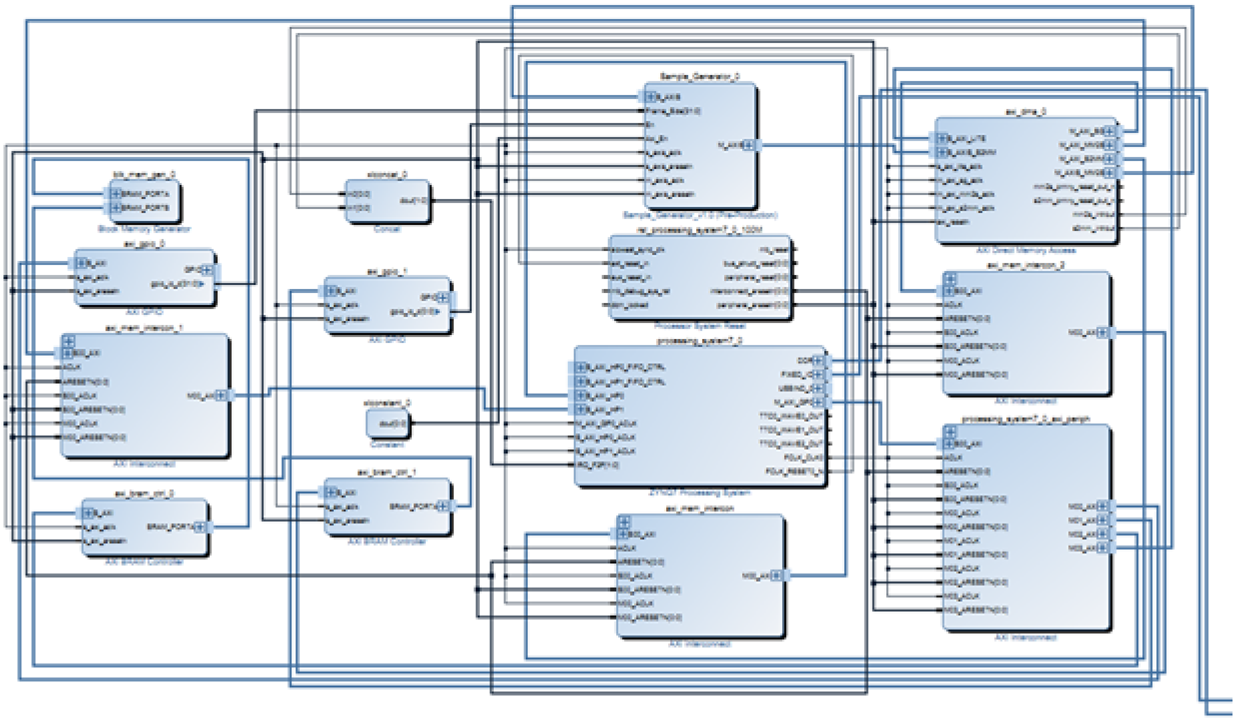


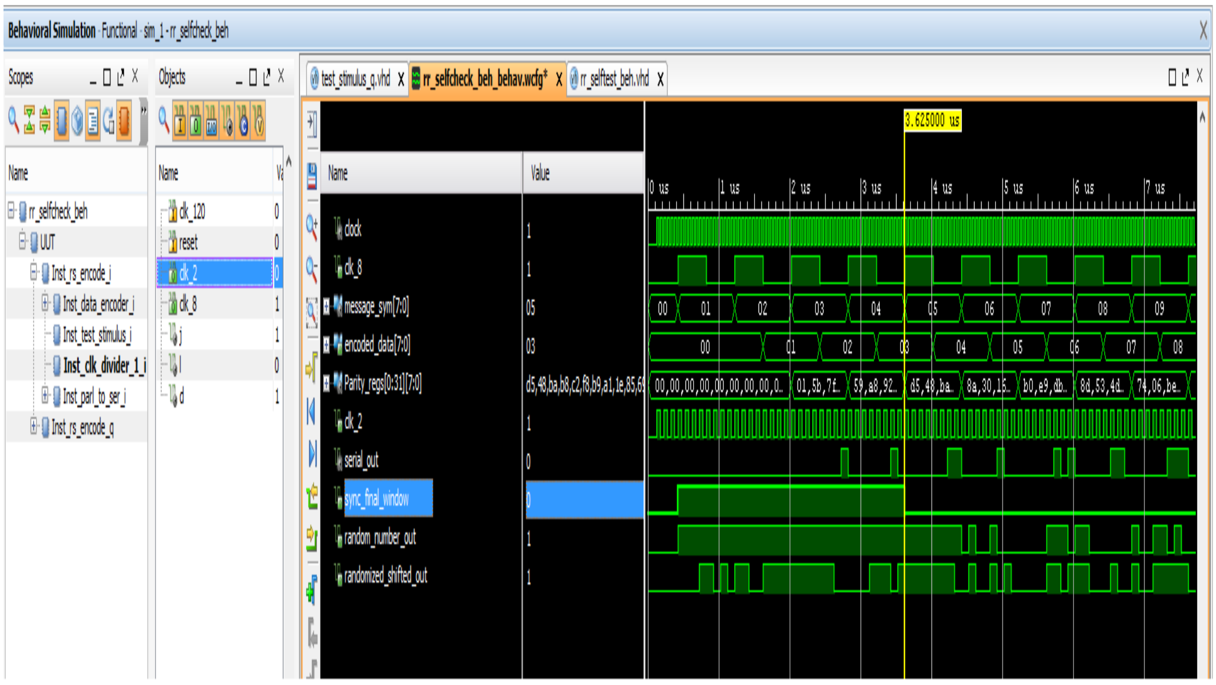
Figure 6.4: Implementation Block Diagram for DMA in Scatter Gather Mode

Chapter 7

Results

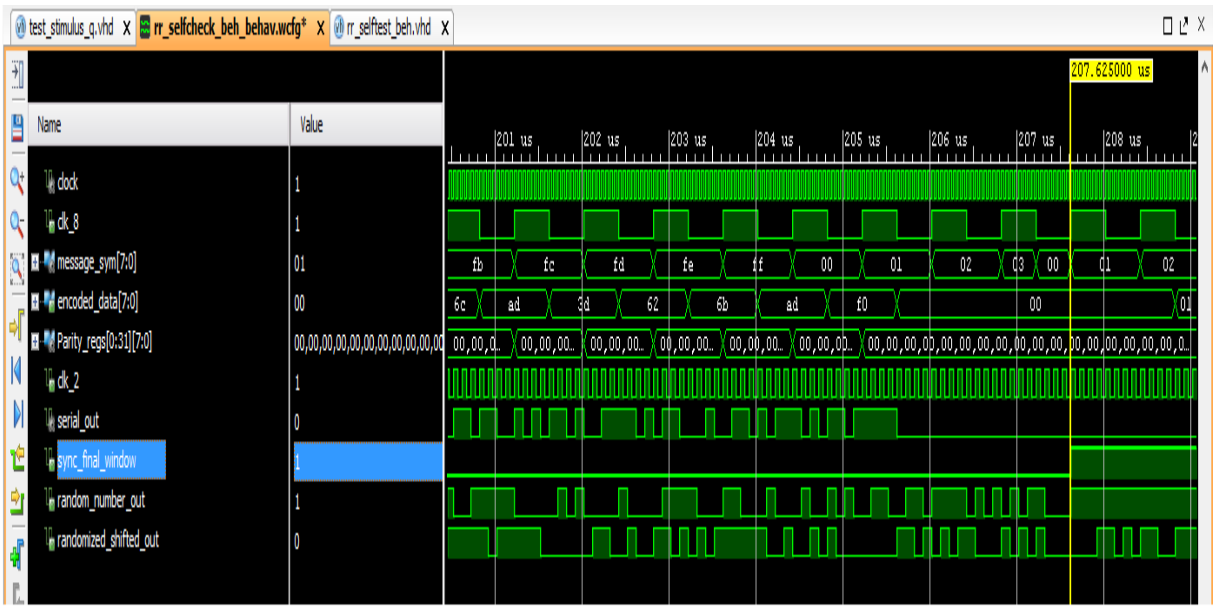
7.1 Results for Data Transmission Process

7.1.1 Simulation Results



Name	Value
clock	1
clk_8	1
message_sym[7:0]	05
encoded_data[7:0]	03
Parity_regs[0:31][7:0]	d5,48,ba,b8,c2,f8,b9,a1,1e,85,69
clk_2	1
serial_out	0
sync_final_window	0
random_number_out	1
randomized_shifted_out	1

Figure 7.1: Simulation Result for Data Transmission Process at the start of packet



Name	Value
clock	1
clk_8	1
message_sym[7:0]	01
encoded_data[7:0]	00
Parity_regs[0:31][7:0]	00,00,00,00,00,00,00,00,00,00,00
clk_2	1
serial_out	0
sync_final_window	1
random_number_out	1
randomized_shifted_out	0

Figure 7.2: Simulation Result for Data Transmission Process at the end of the Packet

Figure 3.4 shows the simulation results for the data transmission process. For the simulation, Testbench was implemented in VHDL. Ramp signal of 100MHz was taken as an input to the encoder. The figure shows the final serial, randomized, encoded data signal with attached sync marker at the start of packet as an output of data encoder.

Figure 3.5 shows the same results at the end of the frame. Here as one packet finished new packet will start so the real-time data transmission system is achieved.

7.1.2 Implementation Results

Figure 3.6 shows the implementation results for the data transmission process. It shows that as one packet finished new packet will start so the real time continuous data is being generated for transmission. Figure 3.7 shows the actual results of the

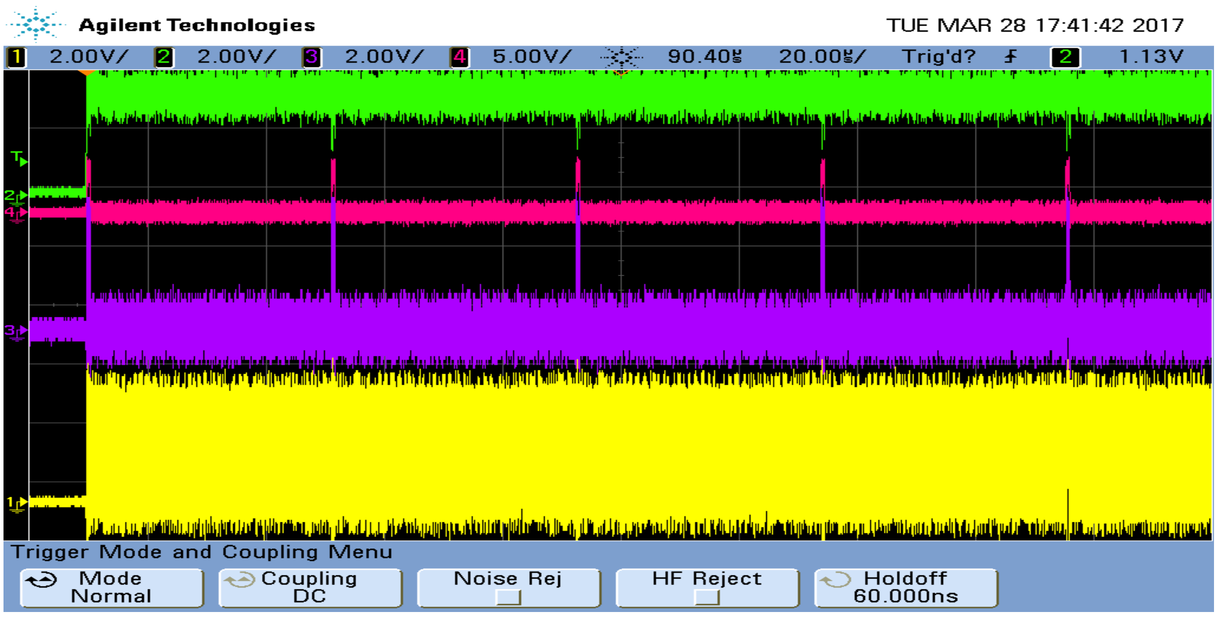


Figure 7.3: Implementation Result of continuous data transmission

implementation on the hardware. The first channel (green signal) shows the master reset for the whole implementation. The second channel (yellow signal) shows the serial, encoded and randomized final output. The third channel (purple signal) shows the serial output of encoded data for reference. The fourth channel (pink signal) shows the random number generation at the start of frame. Figure 3.8 shows the same result as shown in figure 3.7, but it is at the end of the frame.

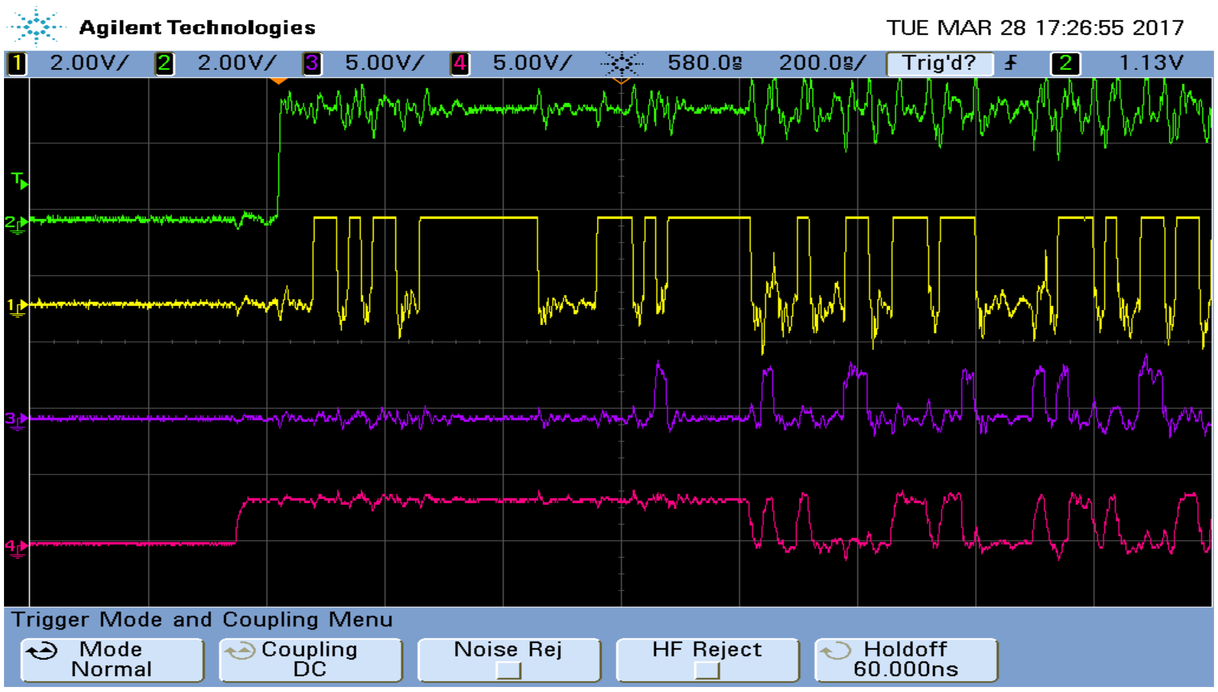


Figure 7.4: Implementation Result at the start of the Packet

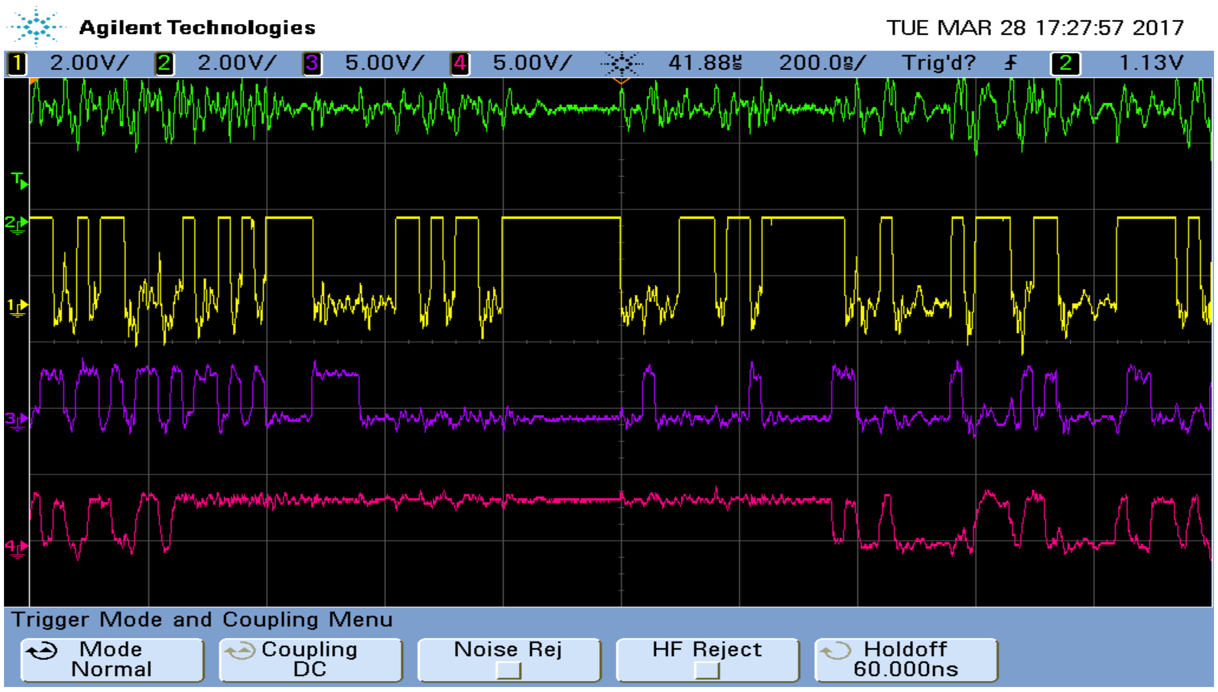
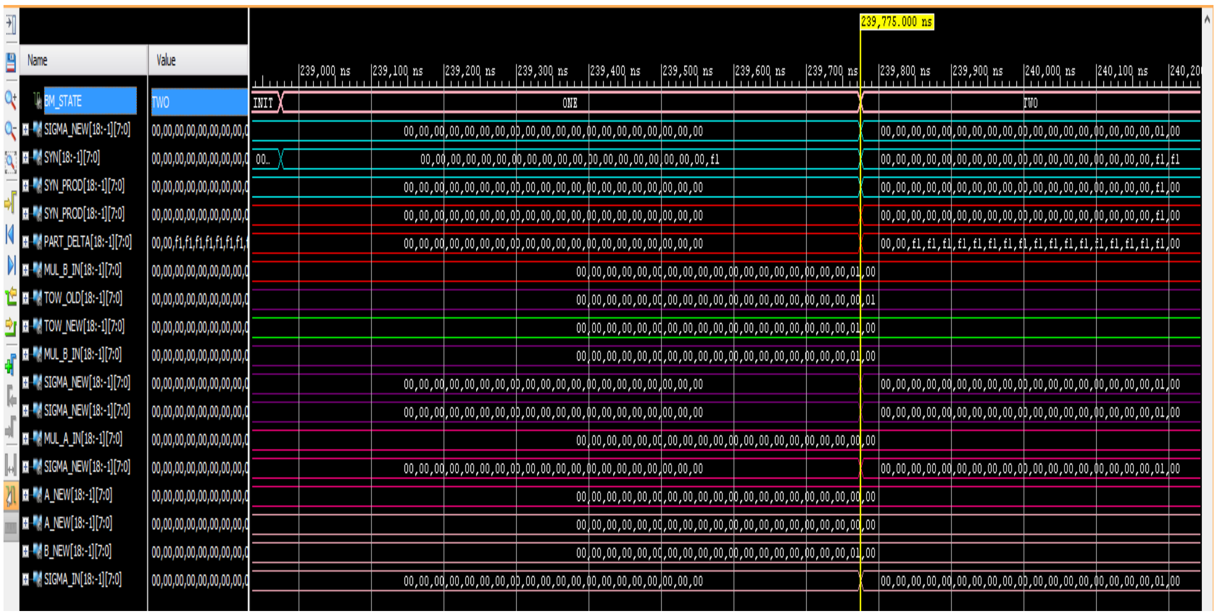


Figure 7.5: Implementation Result at the end of the packet

7.2 Results for Data Reception Process

7.2.1 Simulation Results

Figure 4.5 shows the simulation results for the data reception process. For the simulation, encoded data with some error was given as an input of data decoder. and the figure shows the internal signals generated for the data decoding at the start of the Berlekamp-Massey algorithm. As a final output, we get the new value of error locator polynomial at every clock cycle.



```
Name                      Value
BM_STATE                  TWO
SIGMA_NEW[18:-1][7:0]     00,00,00,00,00,00,00,00,0
SYN[18:-1][7:0]           00,00,00,00,00,00,00,00,0
SYN_PROD[18:-1][7:0]      00,00,00,00,00,00,00,00,0
SYN_PROD[18:-1][7:0]      00,00,00,00,00,00,00,00,0
PART_DELTA[18:-1][7:0]    00,00,f1,f1,f1,f1,f1,f1,f1,
MUL_B_IN[18:-1][7:0]      00,00,00,00,00,00,00,00,0
TOW_OLD[18:-1][7:0]       00,00,00,00,00,00,00,00,0
TOW_NEW[18:-1][7:0]       00,00,00,00,00,00,00,00,0
MUL_B_IN[18:-1][7:0]      00,00,00,00,00,00,00,00,0
SIGMA_NEW[18:-1][7:0]     00,00,00,00,00,00,00,00,0
SIGMA_NEW[18:-1][7:0]     00,00,00,00,00,00,00,00,0
MUL_A_IN[18:-1][7:0]      00,00,00,00,00,00,00,00,0
SIGMA_NEW[18:-1][7:0]     00,00,00,00,00,00,00,00,0
A_NEW[18:-1][7:0]         00,00,00,00,00,00,00,00,0
A_NEW[18:-1][7:0]         00,00,00,00,00,00,00,00,0
B_NEW[18:-1][7:0]         00,00,00,00,00,00,00,00,0
SIGMA_IN[18:-1][7:0]      00,00,00,00,00,00,00,00,0
```

Figure 7.6: Simulation Results of Data Decoding Procedure

Figure 4.6 shows the results at the end of the decoding procedure. It shows the final output with error correction and also showing the error location and error correction value. It shows the initial latency before getting final output but after that we get the continuous output.

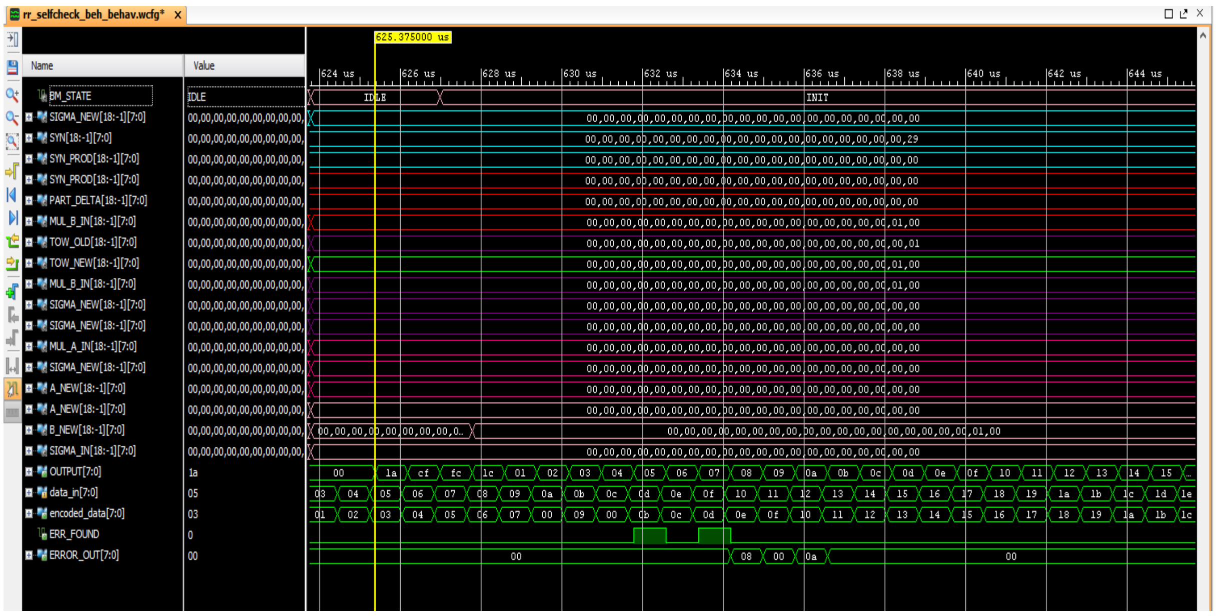
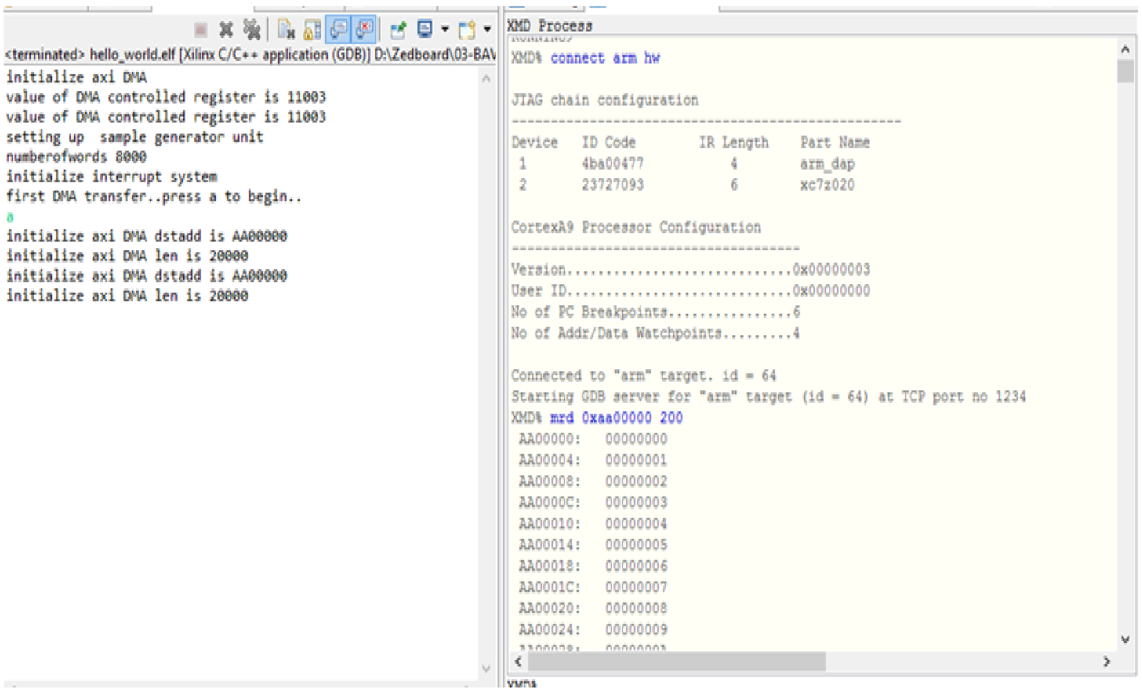


Figure 7.7: Simulation Results of Final Decoded Data

7.3 Results for Data Acquisition System

7.3.1 Results for DMA in Direct Register mode

Figure 6.3 Shows the implementation results after programming of the all blocks which are implemented on the hardware. Here linux is the operating system of the zynq processor. Figure 6.4 shows the same implementation results as figure 6.3 but it is at the end of frame.



```
<terminated> hello_world.elf [Xilinx C/C++ application (GDB)] D:\Zedboard\03-BAV
initialize axi DMA
value of DMA controlled register is 11003
value of DMA controlled register is 11003
setting up  sample generator unit
numberofwords 8000
initialize interrupt system
first DMA transfer..press a to begin..
a
initialize axi DMA dstadd is AA00000
initialize axi DMA len is 20000
initialize axi DMA dstadd is AA00000
initialize axi DMA len is 20000
```

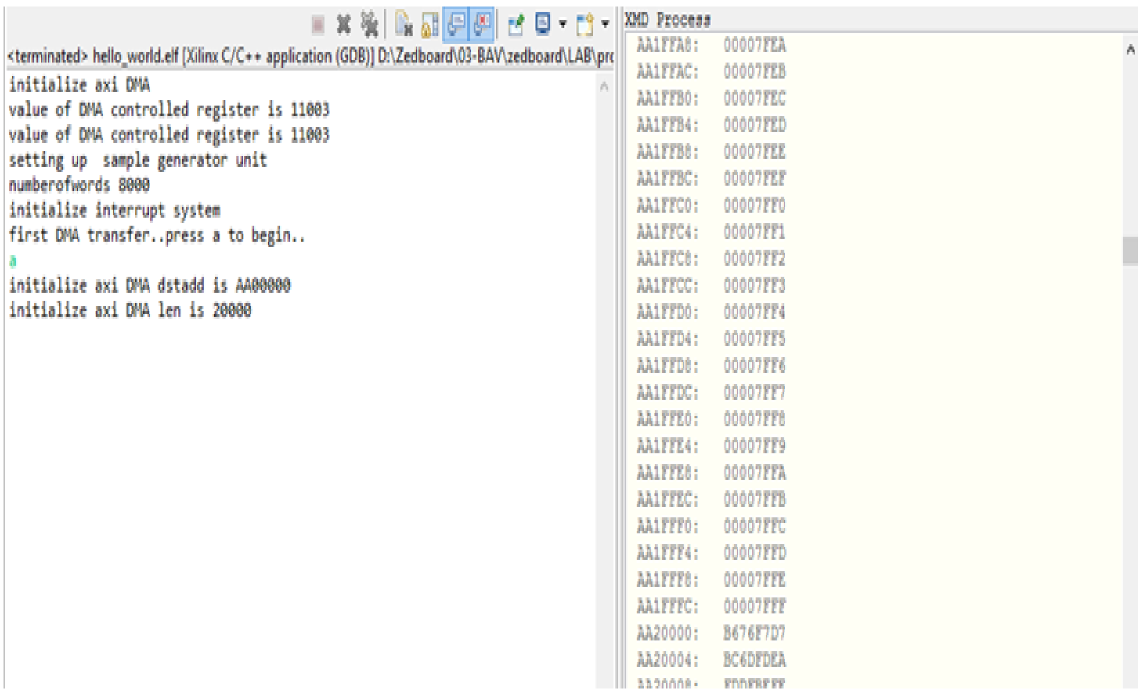
```
XMD Process
XMD% connect arm hw

JTAG chain configuration
--------------------------------------------------
Device   ID Code        IR Length    Part Name
 1       4ba00477          4         arm_dap
 2       23727093          6         xc7z020

CortexA9 Processor Configuration
-------------------------------------
Version.............................0x00000003
User ID.............................0x00000000
No of PC Breakpoints................6
No of Addr/Data Watchpoints.........4

Connected to "arm" target. id = 64
Starting GDB server for "arm" target (id = 64) at TCP port no 1234
XMD% mrd 0xaa00000 200
 AA00000:   00000000
 AA00004:   00000001
 AA00008:   00000002
 AA0000C:   00000003
 AA00010:   00000004
 AA00014:   00000005
 AA00018:   00000006
 AA0001C:   00000007
 AA00020:   00000008
 AA00024:   00000009
```

Figure 7.8: Transferred data with destination address at the start of frame



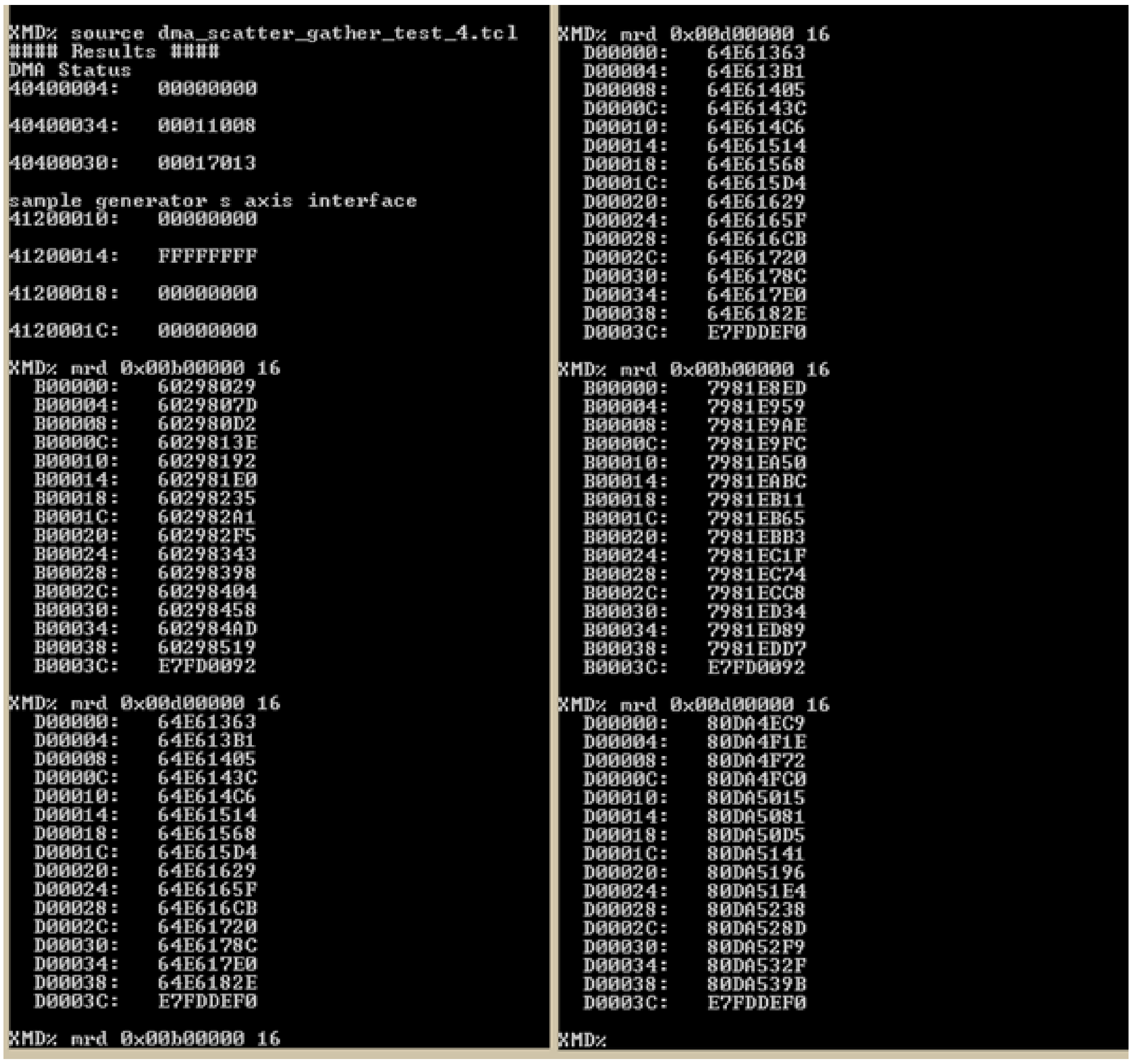
```
<terminated> hello_world.elf [Xilinx C/C++ application (GDB)] D:\Zedboard\03-BAV\zedboard\LAB\pro
initialize axi DMA
value of DMA controlled register is 11003
value of DMA controlled register is 11003
setting up  sample generator unit
numberofwords 8000
initialize interrupt system
first DMA transfer..press a to begin..
a
initialize axi DMA dstadd is AA00000
initialize axi DMA len is 20000
```

```
XMD Process
 AA1FFA8:   00007FEA
 AA1FFAC:   00007FEB
 AA1FFB0:   00007FEC
 AA1FFB4:   00007FED
 AA1FFB8:   00007FEE
 AA1FFBC:   00007FEF
 AA1FFC0:   00007FF0
 AA1FFC4:   00007FF1
 AA1FFC8:   00007FF2
 AA1FFCC:   00007FF3
 AA1FFD0:   00007FF4
 AA1FFD4:   00007FF5
 AA1FFD8:   00007FF6
 AA1FFDC:   00007FF7
 AA1FFE0:   00007FF8
 AA1FFE4:   00007FF9
 AA1FFE8:   00007FFA
 AA1FFEC:   00007FFB
 AA1FFF0:   00007FFC
 AA1FFF4:   00007FFD
 AA1FFF8:   00007FFE
 AA1FFFC:   00007FFF
 AA20000:   B676F7D7
 AA20004:   BC6DFDEA
```

Figure 7.9: Transferred data with destination address at the end of frame

7.3.2 Results for DMA in Scatter-Gather mode

Figure 6.7 shows the final output for the DMA in scatter gather mode of DMA which shows the continuous data transfer from source to the destination memory buffer.



```
XMD% source dma_scatter_gather_test_4.tcl
#### Results ####
DMA Status
40400004:   00000000

40400034:   00011008

40400030:   00017013

sample generator s axis interface
41200010:   00000000

41200014:   FFFFFFFF

41200018:   00000000

4120001C:   00000000

XMD% mrd 0x00b00000 16
  B00000:   60298029
  B00004:   6029807D
  B00008:   602980D2
  B0000C:   6029813E
  B00010:   60298192
  B00014:   602981E0
  B00018:   60298235
  B0001C:   602982A1
  B00020:   602982F5
  B00024:   60298343
  B00028:   60298398
  B0002C:   60298404
  B00030:   60298458
  B00034:   602984AD
  B00038:   60298519
  B0003C:   E7FD0092

XMD% mrd 0x00d00000 16
  D00000:   64E61363
  D00004:   64E613B1
  D00008:   64E61405
  D0000C:   64E6143C
  D00010:   64E614C6
  D00014:   64E61514
  D00018:   64E61568
  D0001C:   64E615D4
  D00020:   64E61629
  D00024:   64E6165F
  D00028:   64E616CB
  D0002C:   64E61720
  D00030:   64E6178C
  D00034:   64E617E0
  D00038:   64E6182E
  D0003C:   E7FDDEF0

XMD% mrd 0x00b00000 16
```

```
XMD% mrd 0x00d00000 16
  D00000:   64E61363
  D00004:   64E613B1
  D00008:   64E61405
  D0000C:   64E6143C
  D00010:   64E614C6
  D00014:   64E61514
  D00018:   64E61568
  D0001C:   64E615D4
  D00020:   64E61629
  D00024:   64E6165F
  D00028:   64E616CB
  D0002C:   64E61720
  D00030:   64E6178C
  D00034:   64E617E0
  D00038:   64E6182E
  D0003C:   E7FDDEF0

XMD% mrd 0x00b00000 16
  B00000:   7981E8ED
  B00004:   7981E959
  B00008:   7981E9AE
  B0000C:   7981E9FC
  B00010:   7981EA50
  B00014:   7981EABC
  B00018:   7981EB11
  B0001C:   7981EB65
  B00020:   7981EBB3
  B00024:   7981EC1F
  B00028:   7981EC74
  B0002C:   7981ECC8
  B00030:   7981ED34
  B00034:   7981ED89
  B00038:   7981EDD7
  B0003C:   E7FD0092

XMD% mrd 0x00d00000 16
  D00000:   80DA4EC9
  D00004:   80DA4F1E
  D00008:   80DA4F72
  D0000C:   80DA4FC0
  D00010:   80DA5015
  D00014:   80DA5081
  D00018:   80DA50D5
  D0001C:   80DA5141
  D00020:   80DA5196
  D00024:   80DA51E4
  D00028:   80DA5238
  D0002C:   80DA528D
  D00030:   80DA52F9
  D00034:   80DA532F
  D00038:   80DA539B
  D0003C:   E7FDDEF0

XMD%
```

Figure 7.10: Transferred Data from the DMA in scatter-gather mode

Chapter 8

Conclusion and Future Scope

8.1 Conclusion

Real-time CCSDS specified baseband data transmission and reception system has been successfully designed developed and implemented at 50 MHz on the reconfigurable platform using VHDL. The real-time data acquisition system based on continuous DMA of data has been experimentally tested to work satisfactorily. Through this there are several learning outcomes such as new tools have been studied, boards have been explored. This would help in the real-time data reception system of the satellite.

8.2 Future Scope

In this project, the main focus is on data transmission and reception system of the future satellite like GISAT and the high-speed data acquisition system now the communication of target system SoC to host PC will be implemented in a manner such can be used in data reception system.

Bibliography

[1] ”TM SYNCHRONIZATION AND CHANNEL CODING”, [Online], Website, May 1997, https://public.ccsds.org

[2] ”The Zynq Book ebook”, *http : //www.zynqbook.com*

[3] zedboard refdoc Vivado 2014-2 [Online],Website, 9th March 2014, https://weble.upc.edu/asig/ESDC/ Tutorials/zedboard refdoc Vivado 2014-2/zedboard refdoc Vivado 2014-2.pdf

[4] Sklar, Bernard. ”Reed-solomon codes.” URL http://www. informit. com/content/images/art. sub.–sklar7. sub.–reed-solomo-n/elementLinks/art. sub.–sklar7. sub.–reed-solomon. pdf (2001): 1-33.

[5] Hsie-chia Chang, CB Shung, Chen yi lee, A reed-solomon product code (RS-PC) decoder chip for DVD application, (IEEE) Journal of Solid State Circuits, (Volume: 36, Issue: 2, Feb 2001).

[6] Reed-Solomon(RS) Coding Overview, VOCAL Technologies, Ltd., Rev. 2.28n, 2010.

[7] J.Y Chang and C. Shung, ”A high speed Reed-Solomon codec chip using look forward architecture ”, IEEE APC CAS94, PP. 212-217, Dec. 1994.

[8] Lee H., ”A high speed, low complexity Reed-Solomon decoder for optical communications”, IEEE Transactions on Circuits and Systems II, PP. 461-465, 2005.

[9] J. I. Hall, ”Notes on Coding Theory”, Dept. of Mathematics, Michigan State University, East Lansing, MI 48824 USA, Jan. 3, 2003–”Chapter 5: Generalized Reed-Solomon Codes” Internet Article, pp. 63-76, Jan. 3, 2003.

[10] J. I. Hall, ”Notes on Coding Theory”, Dept. of Mathematics, Michigan State University, East Lansing, MI 48824 USA, Jan. 3, 2003–”Chapter 5: Generalized Reed-Solomon Codes” Internet Article, pp. 63-76, Jan. 3, 2003. Sklar, B., ”Digital Communications: Fundamentals and Applications”, Second Edition; Prentice Hall, 2001.

[11] pg021 axi dma, [Online], Website, 5th october 2016, https: //www.xilinx.com /support/ documentation /ip_documentation /axi_dma/ v7_1/ pg021 _axi_dma.pdf