# Implementation Of RapidIO Physical Layer : Encoding of 64b/67b

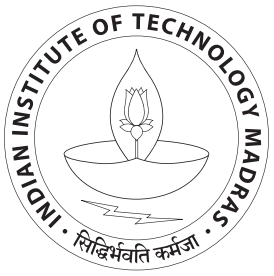
*A Project Report*

*submitted by*

**SHASHI KANT RANJAN**

*in partial fulfilment of the requirements*
*for the award of the degree of*

**MASTER OF TECHNOLOGY**




**DEPARTMENT OF ELECTRICAL ENGINEERING**
**INDIAN INSTITUTE OF TECHNOLOGY MADRAS.**

**May 2018**

# THESIS CERTIFICATE

This is to certify that the thesis entitled **Implementation Of RapidIO Physical Layer : Encoding of 64b/67b** , submitted by **SHASHI KANT RANJAN**, to the Indian Institute of Technology, Madras, for the award of the degree of **Master of Technology**, is a bonafide record of the research work carried out by her under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

**Prof. V . KAMAKOTI**
Research Guide
Professor
Dept. of Computer Science and Engineering
IIT-Madras, 600 036

Place: Chennai

Date: 21 MAY 2018

# ACKNOWLEDGEMENTS

My foremost gratitude and respects to my guide Dr. V. Kamakoti for his guidance, motivation and moral support in my execution of this project work. His passionate talks about the subject and the many hour-long discussions renewed the zeal in me to keep trying despite the unsatisfying results. His patience with students has been incredible.

A special thanks is also due to the faculty advisor, Prof. C. Saurabh Saxena who patiently listened, evaluated, criticized and guided us periodically. The invaluable inputs and suggestions from them were instrumental in steering the project work in the right direction. My special thanks and deepest gratitude to Gopi Nathan M, Neel Gala, Vishwesh, Arjun Menon, Rahul, who has been very supportive. They have enriched the project experience with their active participation, deep understanding and knowledge of the subject matter and invaluable suggestions. I would like to thank my parents for supporting and encouraging me throughout my studies at IIT and for tolerating my prolonged absence from home.

I thank all my classmates and seniors in IITM for their presence that made life happy and incredible here. Their inputs have been significant in every step of these two years.I thank God for making all of this happen.

# ABSTRACT

KEYWORDS: Control Symbol,Control Codeword, Scrambler, Descrambler, Encoder, Decoder,Ordered Sequencing,IDLE,Linear Feedback Shift Registers(LFSR),High Speed Serializers

The project describes the design and implementation of ”Encoding of 64b/67b and control symbol generation in RapidIO Physical Layer ”.In data transmission in RapidIO, 64b/67b is a line code that transforms 64-bit data to 67-bit line code to provide enough state changes to allow reasonable clock recovery and alignment of the data stream at the receiver. It was defined by the IEEE 802.3 working group as part of the IEEE 802.3ae-2002 amendment which introduced in RapidIO.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# ABBREVIATIONS

| | |
|---|---|
| **BSV** | Bluespec System Verilog |
| **ASIC** | Application Specific Integrarted Circuit |
| **BSW** | Bluespec Development Workstation |
| **CPU** | Central Processing Unit |
| **DSP** | Digital Signal Proecssor |
| **DW** | Double Word |
| **FIFO** | First In First Out |
| **HDL** | Hardware Description Language |
| **IREQ** | Initiator Request |
| **IRESP** | Initiator Response |
| **SRIO** | Serial Rapid IO |
| **CW** | Control Codeword |

# CHAPTER 1

# INTRODUCTION

## 1.1 Overview

This chapter covers the background and motivation to design the encoding of 64b/67b encoder and decoder and control symbol for Serial RapidIO (SRIO) physical layer. The encoding of 64b/67b is designed using the Bluespec System Verilog language. The support for Verilog HDL language is also provided along with this design.The Verilog files are generated using the bluesim compiler. The basic testbench module is developed using BSV (also available in Verilog HDL) to verify the functional verification of the design and later, testbench will be enhanced to cover more corner cases.

## 1.2 Motivation

Over the last couple of years, the performance of processors has grown phenomenally. With this rise in the speed of processors, the need for using high speed interconnects has also increased. Typical interconnect speeds today are touching almost 40 Gbps and the speeds are further increasing. Just as we have different types of processors for different types of applications, like general purpose processors, graphics processors, DSPs etc, different types of interconnects have evolved for different applications. SRIO was mainly designed and developed as

an embedded system interconnect for the processor fabrics. It offers high reliability operation with a low latency. The SRIO operations are highly deterministic and support various processor architectures. SRIO supports memory to memory transactions which can be carried out without any software intervention. Peer to peer transactions are supported by SRIO. Multiple processors communicate through shared memory to each other. In SRIO, the receiving end of the packet has a destination ID and the sender is identified with a soured ID. An SRIO switch identifies the port to which the packet needs to be sent with the help of the destination ID. With these basic features of both the standards and recognising the fact that the need for multi core pro-cessing has increased like never before in todays time,there is a necessity to use this types of interconnects together, with necessary bridging between them and that is the motivation behind this project.

## 1.3 Major Contribution

1. Designing control Symbol .
2. Encoding of 64b/67b.
3. Ideal Sequence Generator.
4. Serialising
5. Deserialising
6. Frame Lock Process
7. Scrambling and Descrambling

## 1.4 Organization of the Thesis

Chapter 3 describes the prerequisites of the first peripheral protocol. In this chapter, the necessary features of the SRIO protocol including its packet structure and

various fields, that have been used in the Logical and Transort Layer are described. In Chapter 4, describes the architecture and format for control symbol. In this chapter, the necessary features of the control symbol generation and various fields, that have been used are described. Using the architecture, functions provided by the Physical Coding Sublayer (PCS) and Physical Media Attachment (PMA) sublayer used for 64b/67b encoded links is detailed in Chapter 5. Following this, the experiments were carried forward. Chapter 7 concludes the work with analysis and future direction.

# CHAPTER 2

# Bluespec System Verilog

The hardware decription language used for coding the arithmetic operations on posits is BSV.

## 2.1 Introduction

Bluespec System Verilog is a Hardware Description Language (HDL), which is used for specification, synthesis, modeling and verification of ASIC and FPGA design. With a radically different approach to highlevel synthesis, bluespec offers significantly higher productivity. It allows designers to express intended hardware through high-level constructs, where all behavior is described as a set of guarded atomic actions.

## 2.2 Limitataion of Verilog

Verilog focusses more on simulation than logic synthesis. The source text of verilog often explicitly contains aspects of circuit that could be readily determined by the compiler, such as size of registers, width of busses etc. This makes the design less portable. Handling concurrency in hardware is relatively difficult in verilog as the designer should manage all the aspects of handshaking between combinational circuits. Shared use of register and other memory resources should also be elaborated. The behavioral specification of design in verilog often consumes multiple

clock cycles. Attempts to resolve this problem results in a highly unreadable code with possible bugs. In practice,this problem is solved by separating the combinational and sequential parts of the circuit. Due to these shortcomings, the synthesis and verification of hardware in verilog is slowed down. This is a huge problem during the design of SOC.

## 2.3 Bluespec

Bluespec is based on atomic transactions, which increases the level of concurrency abstraction above SystemC and RTL without compromising the control over hardware design. It enables automatic synthesis of complex control logic, which is the source of many bugs. This results in highly adaptable, reusable and re-configurable designs. Control adaptive parametrization in bluespec provides flexibility, where a significantly different micro-architecture can be generated by changing the parameters in the design with the associated control structures generated automatically. Bluespec allows user defined data types and static type checking. It provides several features of the modern high level languages and all of them can be synthesized.

In recent times, several attempts have been made to move the hardware design language towards a more software like specification of the circuit behaviour. Languages like C, C++ are used to express designs as sequential programs. However, the semantic gap between the software model and the hardware results in sub optimal designs with unpredictable speed and area. Bluespec System Verilog tackles this problem by building upon the traditional hardware semantics. It exploits advanced concepts from software only for static elaboration and static verifica-

tion. It uses the standard hardware structure model of verilog such as modules, module instances, hierarchy etc. For communication between modules it uses the System verilog model of interfaces and interface instances. These added with the advanced features of the high level languages, makes designing and verification in bluespec much faster.

**Design activities by BSV**

- Virtual Platforms - Virtual platforms written in BSV are much faster and accurate than software virtual platform.
- Architectural Modeling - It is very difficult to decide analytically that a system component should be hardware or software. BSV gives architecture model as IP 5 blocks for exploration and validation.
- Design and Implementation - BSV enables designing of IP blocks at a much higher level of abstraction and with better maintainability.
- Verification Environments - BSV is used for writing synthesizable transactors, test benches and both system and reference models.
- Executable Specifications.

## 2.4 Features of BSV

BSV is feature rich Hardware Descriptor Language and at the same time uses important feature of C/C++, and Object Oriented Programming. It includes all major data types present in a High Level Language. Instead of ports as in Verilog, it uses Interfaces. Each functional component is defined as Module. Coordination and Control among different modules are done through interfaces. Methods are responsible for transferring data and control signals across different modules. Atomicity and Concurrency is achieved by Rules. Functions written in C and Verilog can also embedded in BSV.

### 2.4.1 Compilation of BSV code

1. A designer writes a BSV program. It may optionally include Verilog, System Verilog, VHDL and C components.
2. The BSV program is compiled into a Verilog or Bluesim specification. This step has two distinct stages:
   - Pre-elaboration - parsing and type checking.
   - Post-elaboration - code generation.
3. The compilation output is either linked into a simulation environment or processed by a synthesis tool.

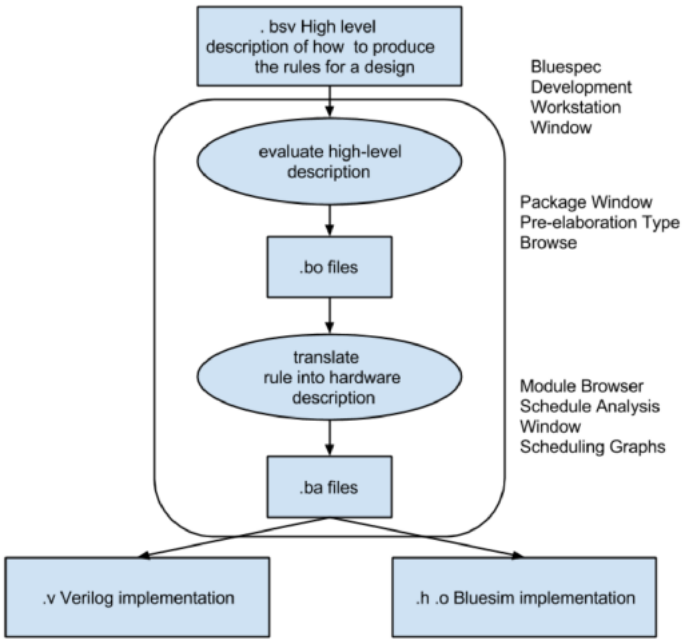



Figure 2.1: Compilation Process of BSV

### 2.4.2 Modules and Interfaces

Module is the basic element of the hardware design hierarchy in bluespec. A module can be instantiated multiple times, and also different parameters can be passed during every instantiation. Unlike verilog, bluespec does not have input, output and in-out pins as interface to modules. Methods are used to drive signals

and busses in and out of modules. These methods are grouped together into interfaces. Modules contain rules, which use methods in other modules.

In BSV, the interface declaration is done separately, outside the module definition. This allows declaration of common interfaces which can be used in multiple modules,without having to declare them repeatedly. All the modules which share the same interfaces also share same methods and therefore share same number and type of inputs and outputs.

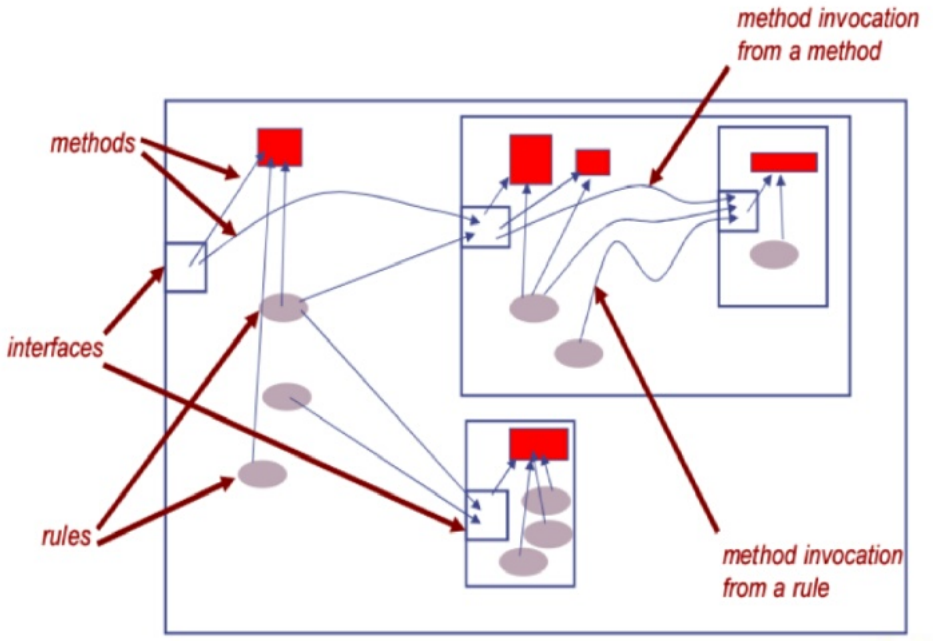



Figure 2.2: Representation of Methods, Interfaces and Rules in a module hierarchy

### 2.4.3 Data Types

In BSV, every variable has a type and only the values of compatible types can be assigned to a variable. The BSV compiler provides a strong, static type-checking environment. Type checking is done before the program elaboration and it ensures that the object types are compatible and the conversion functions are valid for the

context. Bluespec also allows the usage of user-defined types. BSV has a type class which can be considered as a set of types. It implements overloading across related data types. Overloading is the ability to use a common name for a collection of types, with the specific type for the variable being chosen by the compiler based on the types on which it is actually used. Functions and operators are shared by all the data types within a type class.

Some common scalar types used in Bits type class are Bit(n), Bool, UInt(n) and Int(n). The values stored in registers, FIFOs and other memory elements and also the values passed by wires, must be in the Bits type class. Other common data types include Integer, which belongs to the Arith type class and String, which belongs to the Literal type class etc.

Apart from basic data types BSV has user define data types that really helps in modeling wires of circuit.

- **Typedef** - ”typedef” is used to define new and more reliable synonym.
  *typedef existingDataType NewDataType*
  typedef bit[63:0] Data;
  typedef bool flag;
- **Enum** - ”enum” defines new data type to a set of scalar values with symbolic name. Due to strong type-checking enum variables can not be compared with its equivalent numeric value.
  *typedef enumIdentifier1 identifier2 ... NewType*
  typedef enum Reset, Count, Decision State deriving(Bits, Eq);
- **Structs**- A structure or record (struct) is a composite type made up of a collection of members. Each member has a particular type and is identified by a member, or field, name.
  typedef structIdentifier1, Identifier2, ... NewType deriving (Bits,Eq)
  typedef structint x; int y; Coord;
  typedef structAddr pc; RegFile rf; Memory mem; Proc;
- **Tagged Union** - A tagged union is a composite type made up of a collection of members. A union value only contains one member at a time while a structure value contains all of its members.

### 2.4.4 Rules

Rules manage the movement of data from one state to another, within the module.It consists of two parts: rule conditions and rule body. Rule conditions are boolean expressions which decide whether the rule can be fired. Rule body is a set of actions for state transitions. Rules in BSV are atomic. The actions within the rule completely describes the state transition. The process of determining the functional correctness of a design is greatly simplified by one-rule-at-a-time semantics. That is, because of the atomic property of rules, each rule can be looked at in isolation, without considering the actions of the other rules to determine functional correctness. Multiple rules can be executed concurrently in the hardware implementation.

The actions in a rule are executed simultaneously. This can be thought of as similar to the execution of non-blocking statements in always blocks of verilog. Also, as the rule has atomic property, the entire body of rule is executed and there is no partial exe- cution of a rule. When there are several rules within a module, the execution of rules is ordered by the compiler. No two rules can execute simultaneously. The ordering of the rules by the compiler is called scheduling.

BSV does not have always blocks like Verilog. Rules execute as logically ”instantaneous”, ”complete”, and ”ordered” with respect to the execution of all other rules. By ”instantaneous” it means that all the actions in a rule body occur at a single common instant, there is no sequencing of actions within a rule. By ”Complete” it means, when it is fired, the entire rule body executes; there is no concept of ”Partial” execution of a rule body. By ”ordered” it means that each rule execution conceptually occurs either before or after every other rule execution, never simultaneously[4].

Rules are made up of two components.

- **Rule Condition** - A boolean expression which determines, if the rule body is allowed to execute.
- **Rule Body** - A set of actions which describe the state updates that occur when the rule fires.

### 2.4.5 Methods

Method is a procedure which takes arguments and returns a value. It could also return a value without taking any arguments. It becomes a bundle of wires when translated into RTL. The method definition is written within the definition of the interface and it can be different in different modules sharing a common interface. A method also contains implicit conditions which are handshaking signals and logic automatically generated by the compiler. They do not alter any state within the module. Action methods cause actions to occur. They create state changes within the module. Action value methods are a combination of value methods and action methods. They cause state changes and also return values.

Methods are carrier for sending requests and getting responses across Modules. Methods carry data and commands on wires. Methods are different from functions because they are associated with some interface, whereas functions are independent. Methods are of following types:

- Action Method
  method Action MethodName(parameter1, parameter2, ...)
- Action Value Method
  method ActionValue (returnType ) MethodName(parameter1, parameter2, ...)
- Value Method
  method Value(returnType ) MethodName(parameter1, parameter2, ...)

# CHAPTER 3

# Prerequisites of the Peripheral Protocol: Serial RapidIO

## 3.1 Why RapidIO

RapidIO interconnect standard was primarily developed for embedded systems applications. It is being used these days to interconnect processors, general purpose computing platforms, memories, storage devices. The standard promises performance levels upto 60 Gbit/s for board to board and chip to chip communications. There are two variants of the RapidIO architecture: Parallel and Serial. The parallel interface is used for processor and sys- tem connectivity and the serial one is used for serial control, DSP and serial backplane applications. Both variants have the same addressing mechanisms, transactions and programming models. Serial variant has been used for the current project. Layered architecture approach is followed in RapidIO, to keep it backward compatible and to allow scala- bility for future requirements. There are three layers in the SRIO hierarchy viz. the Logical Layer, the Transport Layer and the Physical Layer. The packet formats and the protocols are defined in the Logical Layer. The Transport Layer is the middle layer and provides necessary routing information for the movement of the packet. The bottom most layer i.e. the Physical Layer is responsible for handling the electrical characteristics at the device level and error control etc.

## 3.2 SRIO Transaction Flow

Request and response transactions form the basis of SRIO operation. The communication between endpoints happens through packets. A request transaction is generated by the initiator and is transmitted towards the target. A response packet is generated by the target and is sent to the initiator to complete the transaction. Packets contain important fields that are responsible for ensuring the delivery of required information to the target in a reliable manner. Typically, there exists a SRIO fabric that is a group of switches which provides system connectivity to link all the endpoints together. There are a group of control symbols that manage the transaction flow in the interconnect.

The operation begins when a request transaction is generated by the initiator. The fabric receives the packet and acknowledges it with a control symbol

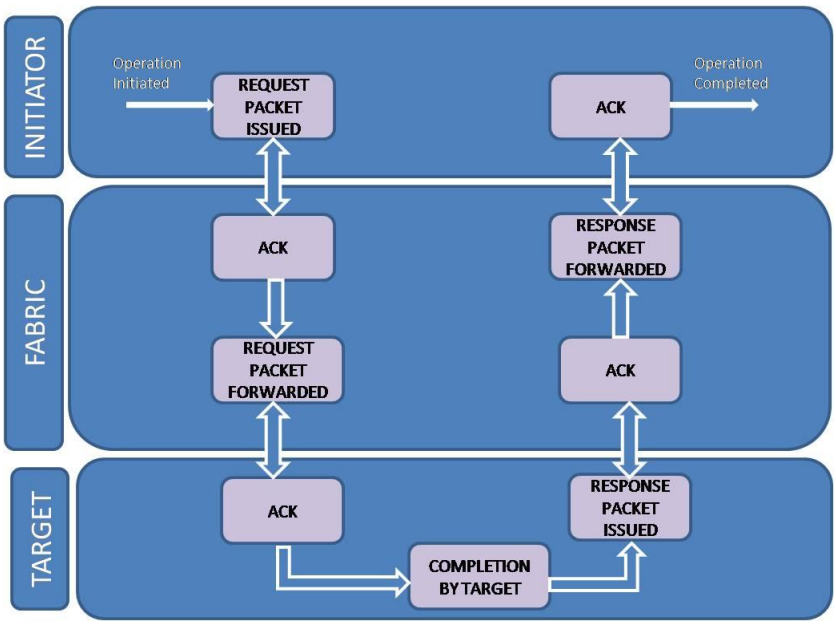



Figure 3.1: SRIO Transaction Flow[†]

and forwards it towards the target device. A response transaction is then generated by the target a complete the request which is passed onto the fabric and this response reaches the initiator with the help of the control symbols. The operation is complete when the response is acknowledged at the initiator.

### 3.2.1 Read Operations

The read operation is used by a processing element to read data from a specified address. It consists of the NREAD and RESPONSE transactions as shown in Figure **??** . Data of requested size is returned in the response packet.

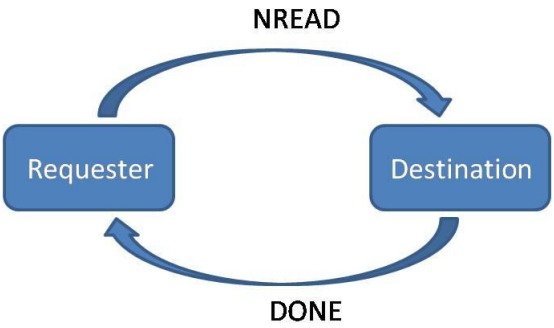



Figure 3.2: NREAD OPERATION

### 3.2.2 Write and Streaming-Write Operations

Whenever data is required to be written to a specified address, write and streaming-write(SWRITE) operations are used. These comprise of the NWRITE and SWRITE transactions. In an NWRITE transaction, multiple double-word, word, half-word and byte writes are allowed with properly padded and aligned data payload. The SWRITE transaction has less header overhead and is a double-word-only version of the NWRITE. There are no responses for NWRITE and SWRITE transactions, and hence sender has no information as to when the transaction has completed at the destination. These transaction can be shown as in Figure 5.4

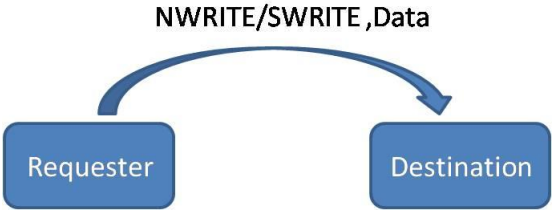



Figure 3.3: NREAD OPERATION

### 3.2.3 Write with Response Operations

The write with response operation is identical to the write operation with only difference that it receives a response to notify the sender that the write has completed at the destination. These transaction comprise of the NWRITE(R) and RESPONSE transactions. This is shown in Figure 5.5.

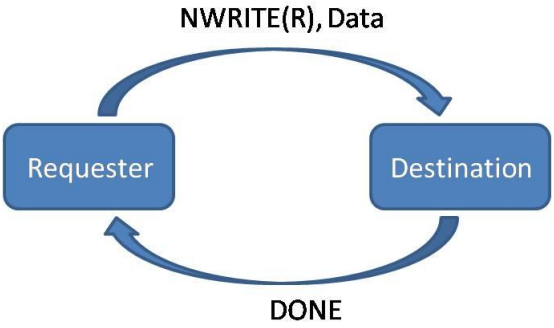



Figure 3.4: NWRITE OPERATION

### 3.2.4 Write with Response Operations

The read-modify-write operation is used by processing elements to carry out synchronization using non- coherent memory. It consists of the ATOMIC and RESPONSE transactions (typically a DONE response). The atomic operation

is a read and write operation combined. Here, the destination reads the data at the specified address, returns the read data to the requester, performs the required operation to the data, and then writes the modified data back to the specified address. No intervening activity is allowed to that address. The operation is as shown in Figure 5.6.

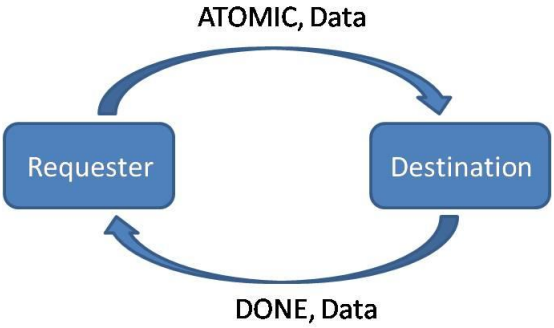



Figure 3.5: ATOMIC(read modify write) OPERATION

### 3.2.5 Packet Formats

A typical SRIO packet can be represented as shown in Figure5.15.


Figure 3.6: ATOMIC(read modify write) OPERATION

The packet consists of the following fields:

- Ftype : It is the Format type, which is as a 4-bit value. The first four bits in the logical packet stream indicate Ftype value .
- Wdptr: Word pointer.It is used in conjunction with the data size (rdsize and wrsize) fields
- Rdsize : It is the Data size for read transactions, used in conjunction with the word pointer (wdptr) bit
- Wrsize: It is the Write data size for sub-double-word transactions, used along with the word pointer (wdptr) bit.
- Rsrv : Reserved
- SrcTID: The packets transaction ID 18
- Transaction : It is also called type or Ttype.It indicates the specific transaction within the format class to be performed by the recipient.
- Extended Address: Optional. Specifies the most significant 16 bits of a 50-bit physical address or 32 bits of a 66-bit physical address.

– Xamsbs: These are the most significant bits of the extended address. The address specified by the address and extended address fields can be extended by 2 bits using this field.
– Address : Indicates the address value

Table 3.1: SRIO Packet Types

| Ftype | Ttype | Packet Type |
|---|---|---|
| 0010 | 0100 | NREAD |
| 0101 | 0100 | NWRITE |
| | 0101 | NWRITE_R |
| 0110 | - | SWRITE |
| 1000 | 0000 | MAINTENANCE(CONFIG READ REQ) |
| | 0001 | MAINTENANCE(CONFIG WRITE REQ) |
| | 0010 | MAINTENANCE(CONFIG READ RESP) |
| | 0010 | MAINTENANCE(CONFIG WRITE RESP) |
| 1010 | - | DOORBELL |
| 1011 | - | MESSAGE |
| 1101 | 0000 | RESPONSE WITHOUT DATA |
| | 1000 | RESPONSE WITH DATA |

### 3.2.6 Packet Details for SRIO

The bit positions for various fields for different packet types as being used for the Bridge are explained as follows: The NREAD packet has the structure shown in Figure 5.16. The packet is identified with the help of the Ftype and Ttype fields. The address field length used for this project is 50 bits of which the first 24 bits are received in the first clock cycle and the remaining 26 bits in the second clock cycle. The other fields are as per the packet format explained in Section 3.6. Priority, Destination Id, Source Id fields of the packet belong to the Transport Layer.

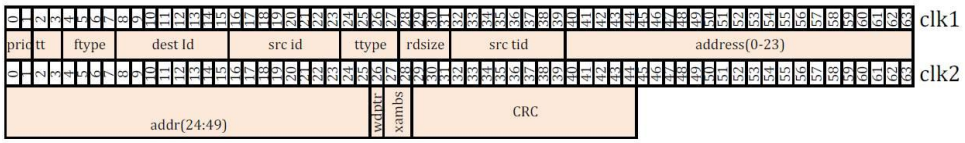



Figure 3.7: NREAD Request Packet (Ftype=2)

The NWRITE packet has the structure shown in Figure 5.17. The packet is identified with the help of the Ftype and Ttype fields. The address field length is 50 bits of which the first 24 bits are received in the first clock cycle and the remaining 26 bits in the second clock cycle. The NWRITE packet has a fixed data length of 64 bits of which the first 35 bits are received in the second clock cycle and the remaining 29 bits are

received in the third clock cycle. The other fields are as per the packet format explained in Section 3.6. Priority, Destination Id, Source Id fields of the packet belong to the Transport Layer.

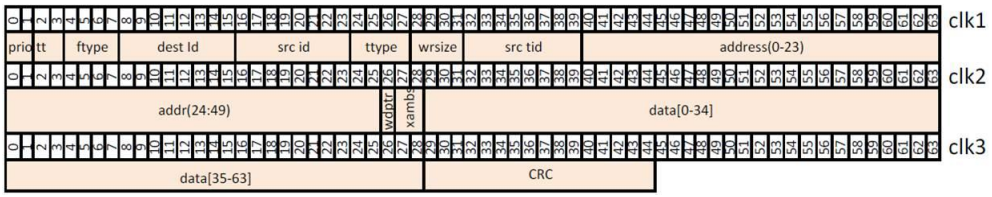



Figure 3.8: NWRITE Request Packet (Ftype=5)

The SWRITE packet has the structure similar to the NWRITE packet and is shown in Figure 5.18. Here a sample packet is shown for 3 data cycles. The difference here is that the data keeps continuously coming till the eof signal is received. Other difference here is that the address starts immediately after Srcid field. The other fields have the same meaning as explained earlier.

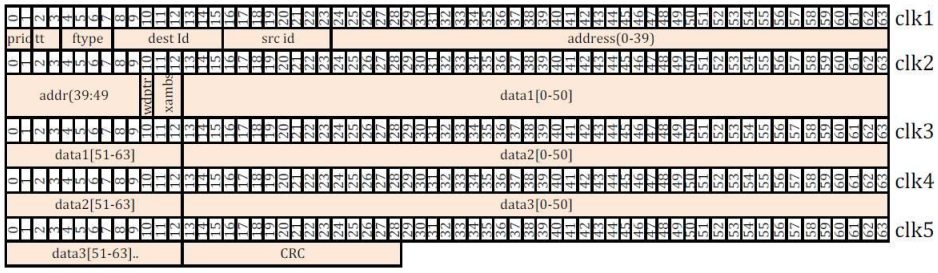



Figure 3.9: NWRITE Request Packet (Ftype=5)

The Response with data packet looks like as shown in Figure 5.19 . Response with data packet has a fixed data length of 64 bits which start after the target id field and continues in the second clock cycle as well.


Figure 3.10: Response with Data packet (Ftype=13,Ttype=8)

Response without data packet is similar to with data case and since there is no data, the packet gets delivered in one clock cycle itself. The packet is shown in Figure 5.20.

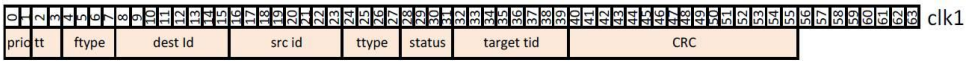



Figure 3.11: Response without Data packet (Ftype=13,Ttype=0)

# CHAPTER 4

# Architecture and Implementation

The user interface of SRIO contains four ports - Initiator Request, Initiator Response, Target Request, and Target Response. A packet is issued or consumed using one of these four ports. The SRIO wrapper feeds these ports for packet generation and consumption.

The user interface of SRIO contains four ports - Initiator Request, Initiator Response, Target Request, and Target Response. A packet is issued or consumed using one of these four ports. The SRIO wrapper feeds these ports for packet generation and consumption.

## 4.1 Control Symbols

### 4.1.1 Introduction

This chapter specifies RapidIO LP-Serial Physical Layer control symbols. Control symbols are the message elements used by ports connected by a LP-Serial link to manage all aspects of LP-Serial link operation. They are used for link maintenance, packet delimiting, packet acknowledgment, error reporting, and error recovery.

### Control Symbol Field Definitions

Table 4.1: Control Symbol Field Definitions

| Field | Definition |
|---|---|
| stype0 | Encoding for control symbols that use parameter0 and parameter1.<br>The encodings are defined for Control Symbol 64 |
| parameter0 | Used in conjunction with stype0 encodings.<br>For the description of parameter0 encodings. |
| parameter1 | Used in conjunction with stype0 encodings.<br>For the description of parameter1 encodings |
| stype1 | Encoding for control symbols |
| cmd | Used for Control Symbol 24 and Control Symbol 48 in conjunction with the stype1 field to define the link maintenance commands. |
| reserved | Set to logic 0s on transmission and ignored on reception |
| alignment | Fixed value of 0b00. These bits are discarded when the control symbol is encoded into codewords and reinserted when it is decoded from codewords. |
| CRC-5 | 5-bit code used to detect transmission errors in Control Symbol 24. |
| CRC-13 | 13-bit code used to detect transmission errors in Control Symbol 48. |
| CRC-24 | 24-bit code used to detect transmission errors in Control Symbol 64. |

## 4.2 Control Symbol Format

There are three kind of control symbol

1. Control Symbol 24,
2. Control Symbol 48 and
3. Control Symbol 64

### 4.2.1 Control Symbol 64

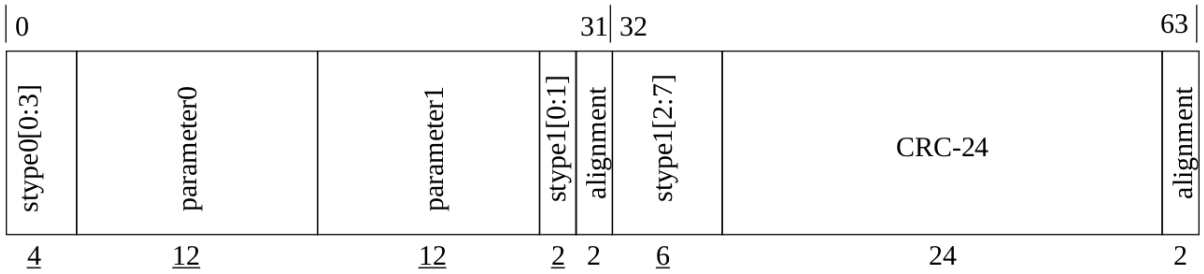



Figure 4.1: Format of Control Symbol 64

The functions encoded in stype0 are status functions that convey some type of status about the port transmitting the control symbol. The functions encoded in stype1 are requests to the receiving port or transmission delimiters. The alignment fields are used to bring the length of the unencoded control symbol up to 64 bits, the number of bits encoded into a 64b/67b codeword. The alignment fields are positioned to allow a Control Symbol 64 to be split across codewords. Control symbols are defined with the ability to carry at least two functions so that a packet acknowledgment and a packet delimiter can be carried in the same control symbol. Packet acknowledgment and packet delimiter control symbols constitute the vast majority of control symbol traffic on a busy link. Carrying an acknowledgment (or status) and a packet delimiter whenever possible in a single control symbol allows a significant reduction in link overhead traffic and an increase in the link bandwidth available for packet transmission. A control symbol carrying one function is referred to using the name of the function it carries. A control symbol carrying more than one function may be referred to using the name of any function that it carries. For example, a control symbol with stype0 set to packet-accepted and stype1 set to NOP is referred to a packet-accepted control symbol. A control symbol with stype0 set to packet-accepted and stype1 set to restart-from-retry is referred to as either a packet-accepted control symbol or a restart-from-retry control symbol depending on which name is appropriate for the context.

### 4.2.2 Stype0 Control Symbols

The encoding and function of stype0, and the information carried in parameter0 and parameter1 for each stype0 encoding,

Table 4.2: Stype0 Control Symbol 64 Encoding

| **stype0** | **Function** |
|---|---|
| 0b0000 | packet-accepted |
| 0b0001 | packet-retry |
| 0b0010 | packet-not-accepted |
| 0b0011 | timestamp |
| 0b0100 | status |

The packet-accepted, packet-retry and packet-not-accepted control symbols are collectively referred to as packet acknowledgment control symbols. Status is the default stype0 encoding, and is used when a control symbol does not convey another stype0 function.

### 4.2.3 Packet-Accepted Control Symbol

The packet-accepted control symbol indicates that the port sending the control symbol has taken responsibility for sending the packet or packets

to its final destination and that resources allocated to the packet or packets by the port receiving the control symbol can be released. This control symbol shall be generated only after the entire packet or packets has been received and found to be free of detectable errors.

Table 4.3: Packet-Accepted Control Symbol field usage and values.

| **Format** | **stype0** | **Parameter0** | **Parameter1** |
|---|---|---|---|
| Control Symbol 64 | 0b0000 | packet_ackID | buf_status |

### 4.2.4 Packet-Retry Control Symbol

A packet-retry control symbol indicates that the port sending the control symbol was not able to accept the packet due to some temporary resource conflict such as insufficient buffering and the packet must be retransmitted.

Table 4.4: Packet-Retry Control Symbol field usage and values.

| **Format** | **stype0** | **Parameter0** | **Parameter1** |
|---|---|---|---|
| Control Symbol 64 | 0b0001 | packet_ackID | buf_status |

### 4.2.5 Packet-Not-Accepted Control Symbol

The packet-not-accepted control symbol indicates that the port sending the control symbol has either detected an error in the received character stream or, when operating in multiple VC mode, has insufficient buffer resources and as a result may have rejected a packet or control symbol. The control symbol contains an arbitrary/ackID_status field and a cause field.

Table 4.5: Packet-Not-Accepted Control Symbol field usage and values.

| **Format** | **stype0** | **Parameter0** | **Parameter1** |
|---|---|---|---|
| Control Symbol 64 | 0b0010 | arbitrary/ackID_status | 0b000_0000,cause |

The cause field is used to provide information about the type of error that was detected for diagnostics and debug use. The content of the cause field is informational only. The contents of both the arbitrary/ackID_status field and the cause field are informational only, unless bit 9 of the Port n Latency Optimization CSR register is set within both the transmitting and receiving port's configuration space. If both ports support Error Recovery with ackID in PNA Enabled, then the contents of the Parameter0 and Parameter1 fields are functional for packet-not-accepted control symbols.

Table 4.6: Cause Field Definition

| Cause | Definition |
|---|---|
| 0b00000 | Reserved |
| 0b00001 | Received a packet with an unexpected ackID |
| 0b00010 | Received a control symbol with bad CRC |
| 0b00011 | Non-maintenance packet reception is stopped |
| 0b00100 | Received a packet with bad CRC |
| 0b00101 | Received an invalid character or codeword, or valid but illegal character |
| 0b00110 | Packet not accepted due to lack of resources |
| 0b00111 | Loss of descrambler sync |
| 0b01000-0b11110 | Reserved |
| 0b01000-0b11110 | General error |

### 4.2.6 Timestamp Control Symbol

Timestamp control symbols are used to set the timestamp generator value of the link partner with a high degree of accuracy, and for links operating at Baud Rate Class 1 or Baud Rate Class 2 as a response to a loop-timing request (loop-response). Timestamp control symbols contain 10 bits of a time value that is spread across the parameter0 and parameter1 fields.

Table 4.7: Timestamp Control Symbol field usage and values

| Format | stype0 | Usage |
|---|---|---|
| Control Symbol 64 | 0b0011 | Timestamp |

### 4.2.7 Status Control Symbol

The status control symbol indicates receive status information about the port sending the control symbol. The control symbol contains the ackID_status and the buf_status fields. The ackID_status field allows the receiving port to determine if it and the sending port are in sync with respect to the next ackID value the sending port expects to receive. The ackID_status field is informational. The buf_status field indicates to the receiving port the number of maximum length packet buffers the sending port has available for reception on VC0.

Table 4.8: Status Control Symbol field usage and values

| Format | stype0 | Parameter0 | Parameter1 |
|---|---|---|---|
| Control Symbol 64 | 0b0100 | ackID_status | buf_status |

## 4.3 Stype1 Control Symbols

Table 4.9: Stype1 Control Symbol 64 Encoding

| stype1(8 bits) | Functions | Packet Delimiter |
|---|---|---|
| 0x00-0x07 | Reserved | |
| 0x00-0x07 | Stomp | yes |
| 0x09-0x0F | Reserved | |
| 0x10 | End-of-packet-unpadded | yes |
| 0x11 | End-of-packet-padded | yes |
| 0x12-0x17 | Reserved | |
| 0x18 | Restart-from-retry | * |
| 0x19-0x21 | Reserved | |
| 0x10,ackID[0:5] | Start-of-packet-unpadded | yes |
| 0x11, ackID[0:5] | Start- of - packet - padded | yes |

### 4.3.1 Start-of-Packet Control Symbol

The start-of-packet control symbol is used to delimit the beginning of a packet. The Control Symbol 64 start-of-packet control symbol has two variants start-of-packet-unpadded and start-of-packet-padded to indicate if a previous packet has padding appended to achieve a total length that is a multiple of 8 bytes. The start-of-packet-unpadded control symbol shall be used for cases where the start-of-packet does not terminate a previous packet or where the start-of-packet terminates a packet that was not padded. The start-of-packet-padded control symbol shall be used when the start-of-packet terminates a packet that was padded to multiple of 8 bytes. It is needed to differentiate between padded and non-padded packets so devices like switches that do not completely decode the packet can separate link overhead from the packet. For Control Symbol 64, the stype1[2:7] bits contain the most significant 6 bits of the packet ackID. The stype1[2:7] bits are reserved when the link is operating in Error Free Mode.

Table 4.10: Start-of-Packet Control Symbol field usage and values.

| Format | stype0 | Parameter0 | Parameter1 |
|---|---|---|---|
| Control Symbol 64 | 0b0100 | ackID_status | buf_status |

### 4.3.2 Stomp Control Symbol

The stomp control symbol is used to cancel a partially transmitted packet. The protocol for packet cancellation

Table 4.11: Stomp Control Symbol field usage and values

| Format | stype1 | cmd |
|---|---|---|
| Control Symbol 64 | 0x08 | N/A |

### 4.3.3 End-of-Packet Control Symbol

The end-of-packet control symbol is used to delimit the end of a packet.

Table 4.12: End-of-Packet Control Symbol field usage and values.

| Format | stype1 | cmd |
|---|---|---|
| Control Symbol 64 | 0x08 | N/A |

### 4.3.4 Restart-From-Retry Control Symbol

This control symbol is used to mark the beginning of packet retransmission, so that the receiver knows when to start accepting packets after the receiver has requested a packet to be retried. The restart-from-retry control symbol cancels a current packet and may also be transmitted on an idle link.

Table 4.13: Restart-From-Retry Control Symbol field usage and values.

| Format | stype1 | cmd |
|---|---|---|
| Control Symbol 64 | 0x18 | N/A |

# CHAPTER 5

# 64b/67b PCS and PMA Layers

## 5.1 Introduction

This chapter specifies the functions provided by the Physical Coding Sublayer (PCS) and Physical Media Attachment (PMA) sublayer used for 64b/67b encoded links. (The PCS and PMA terminology is adopted from IEEE 802.3).

The aim of the experiment is to design and impplement the 64b/67b encoding.

### 5.1.1 PCS Layer Functions

The Physical Coding Sublayer (PCS) function is responsible for idle sequence generation, lane striping, scrambling and encoding for transmission and decoding, lane alignment, descrambling and destriping on reception. The PCS uses a 64b/67b encoding for transmission over the link. The PCS also provides mechanisms for determining the operational mode of the port as Nx or 1x operation, and means to detect link states. It provides for clock difference tolerance between the sender and receiver without requiring flow control.

### 5.1.2 PMA Layer Functions

The Physical Medium Attachment (PMA) Layer is responsible for serializing / de-serializing 67-bit parallel codewords to/from a serial bitstream on a lane-by-lane basis. Upon receiving data, the PMA function provides alignment of the received bitstream to 67-bit codeword boundaries, independently on a lane-by-lane basis. It then provides a continuous stream of 67-bit codewords to the PCS, one stream for each lane. The 67-bit codewords are not observable by layers higher than the PCS. If a LP-Serial port supports either baud rate discovery or adaptive equalization, these functions are also performed in the PMA Layer.

## 5.2 Architecture of 64b/67b Encoding

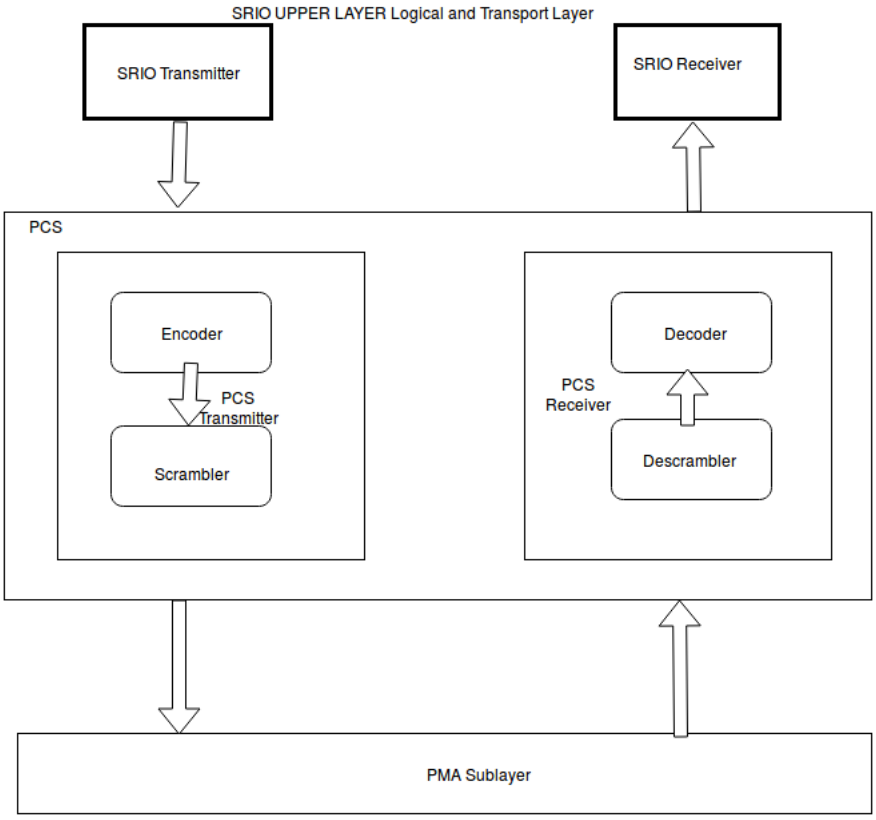



Figure 5.1: Architecture for 64b/66b Encoding

## 5.3 64b/67b Transmission Code

The 64b/67b transmission code used by the PCS encodes 64-bit blocks of data and/or control information into 67-bit codewords for transmission and reverses the process on reception. There are two types of codewords: data codewords and control codewords. Data codewords encode 64 bits of data. Control codewords encode 64 bits of control information or some combination of data and control information. Codewords are scrambled to statistically achieve an acceptable transition density for baud rate recovery in the receiver. Codewords are selectively inverted based on the running disparity and codeword disparity to ensure that the transmitted signal on each lane is DC balanced within +/- 66 1s or 0s at all times.

The inverted bit indicates whether the data_field has been inverted to control the running disparity of the transmitted signal: 0b0 - data_field[0:63] has not been inverted. 0b1 - data_field[0:63] has been inverted. The type bit indicates the type of codeword: 0b0: Control, the codeword encodes a block that contains control information and may contains data information. 0b1: Data, the codeword encodes a block that contains only data information. The !type bit is the complement of the type bit. The transition between the !type and type bits indicates a fixed offset from

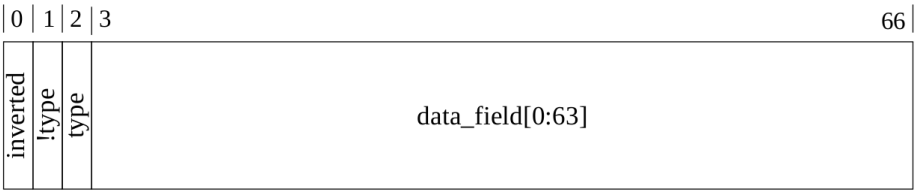



Figure 5.2: 64b/67b codeword format

the beginning of the codeword, for use in codeword lock state machine in Section 5.19.4. The format and content of the data_field depends on the information encoded in the codeword. Codewords shall be transmitted from left to right, bit 0 to bit 66 starting with the inverted bit and progressing to data_field[63].

### 5.3.1 Data Codeword

The format of a data codeword (type bit = 0b1) shall be as shown in Figure5.2.

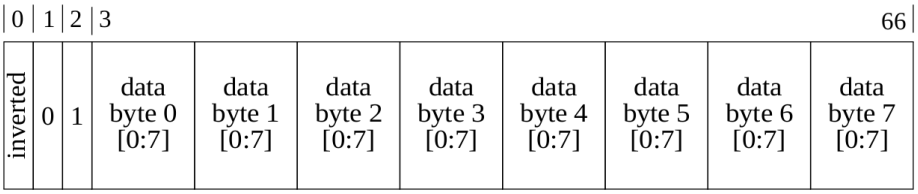



Figure 5.3: 64b/67b Data codeword format

### 5.3.2 Control Codeword

The format of a control codeword (type bit = 0b0) depends on the information the codeword encodes. The 2 bits at location [30:31] of a control codeword data_field are a cc_type field that specifies the contents and format of data_field[0:29,32:63]. The general format of a control codeword shall be as shown in Figure 6.3.

The encoding of the control codeword functions are shown in Table

Control codewords can be further sub-divided into two categories: Symbol Bearing and Non-Symbol Bearing. Symbol Bearing control codewords include: CSB, CSE and CSEB. Non-Symbol Bearing control codewords include: Skip-Marker, Lane Check, Descrambler Seed, Skip and Status/Control.

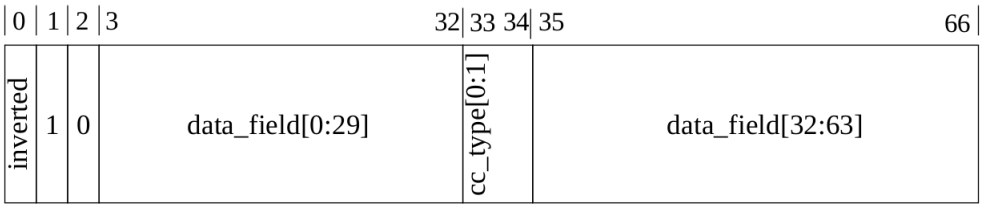



Figure 5.4: General 64b/67b Control Codeword Format

Table 5.1: Control Codeword function encoding

| **cc_type[0:1]** | **data_field[32:35]** | **Name** |
|---|---|---|
| 2'b00 | 4'b1011 | Skip-Marker |
| | 4'b1100 | Lane Check |
| | 4'b1101 | Descrambler |
| | 4'b1110 | Skip |
| | 4'b1111 | Status/Control |
| 2'b01 | – | Control Symbol Begin |
| 2'b10 | – | Control Symbol End |
| 2'b11 | – | Control Symbol End and Begin |

### 5.3.3 Skip Marker Control Codeword

The Skip-Marker control codeword is used together with the Skip control codeword to provide clock compensation.

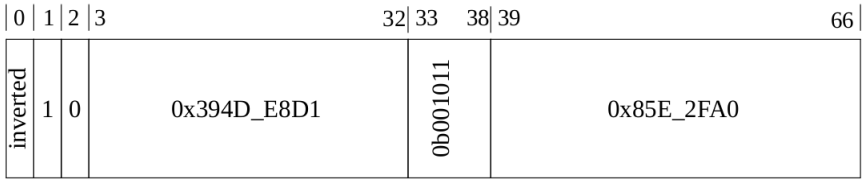



Figure 5.5: Skip-Marker Control Codeword Format

### 5.3.4 Lane Check Control Codeword

The Lane-Check control codeword is used to monitor the BER of the lanes in the link. The Lane-Check control codeword is only intended to be used for bit error rate estimation and shall not influence or trigger error recovery.

### 5.3.5 Descrambler Control Codeword

The format of the Descrambler Seed control codeword shall be as shown in Figure 5-6. Seed codewords transmitted on lane k of a link shall

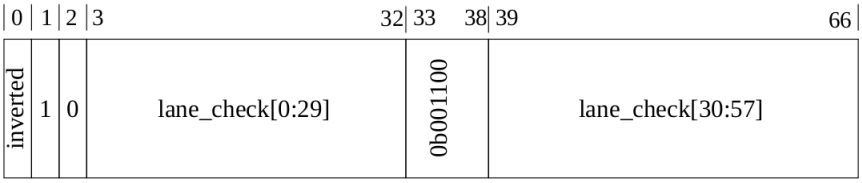



Figure 5.6: Lane-Check Control Codeword

contain the value of the lane k transmit scrambler (seed) which would have been used to scramble the Descrambler Seed Control Codeword if it was a codeword that is scrambled before transmission. The lane k transmit scrambler state (seed) is used to set or check the state of the lane k descrambler in the connected receiver at the time that the Seed codeword would have been descrambled. The Descrambler Seed control codeword is transmitted only as part of a Seed ordered sequence. For more information, refer to Section 5.9.1, ”Seed Ordered Sequence”.

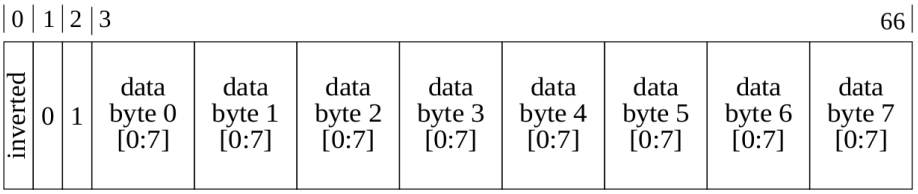



Figure 5.7: Descrambler Seed Control Codeword Format

### 5.3.6 Skip Control Codeword

The Skip control codeword is used together with the Skip-Marker control codeword to provide clock compensation. The format of the Skip control codeword shall be as shown in Figure 5-7. The codeword data field has a disparity of 0 and a Hamming distance of 32 from the Skip-marker control codeword. The codeword shall be transmitted only as part of a Skip ordered sequence. For more information, refer to Section 5.9.3, ”Skip Ordered Sequence”.

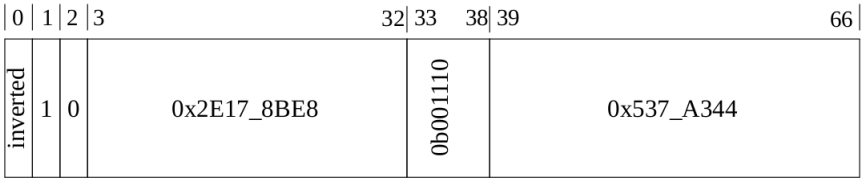



Figure 5.8: Skip Control Codeword Format

### 5.3.7 Status/Control Control Codeword

The Status/Control control codeword is use to communicate various link level information between two link partners, this includes link training control, link initialization and asymmetric link width control. The format of the Status/Control control codeword shall be as shown in Figure 5-8.

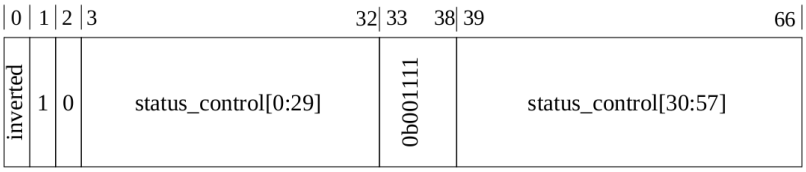



Figure 5.9: Status/Control Control Codeword Format

**The format of the CSB control codeword shall be as shown in Figure .**

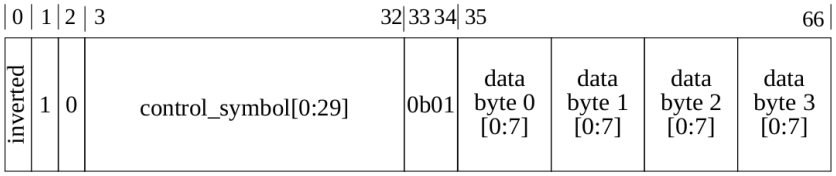



Figure 5.10: CSB Control Codeword Format

### 5.3.8 CSE Control Codeword

The format of the CSE control codeword shall be as shown in Figure .

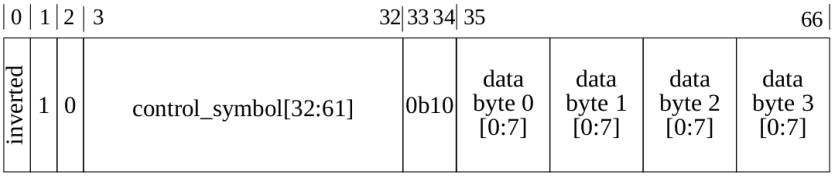



Figure 5.11: CSE Control Codeword Format

### 5.3.9 CSEB Control Codeword

The format of the CSEB control codeword shall be as shown in Figure .

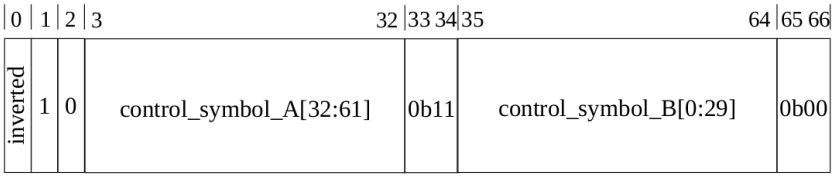



Figure 5.12: CSEB Control Codeword Format

### 5.3.10 Scrambling

A scrambler is a device which maps a digital sequence into a second sequence whose statistical properties are 'more random' in some sense. One simple form of scrambling consists of the addition of a pseudo-random sequence to the input data sequence. At the receiving end, descrambling is performed by subtraction of the same sequence.

The pseudo-random sequence is generated by a linear feedback shift register (see diagram). The serial data enters the LFSR, where each stage is a D-type flip-flop equivalent to one time unit of delay. The delayed signal is tapped, with XOR (modulo-2) feedback with the input signal. For the descrambler, the XOR process is feed-forward. Scrambling smooths the spectrum of the ports's transmit signal and reduces the spectrum's peak values.It reduces the electromagnetic interference and DC component.For idle encoding our encoder should be bit-synchronized. Scrambling is used to get a DC-balanced output before transmission of the data over the optical link. Otherwise long strings of binary ones or zeros cause the average signal at the AC-coupled receiver to drift up and down. If the receiver input drifts to near the threshold value, any noise will cause bit errors.

In addtion, scrambling is quite useful for clock recovery. Long strongs of binary ones or zeros have no state transitions from which timing information can be extracted. By increasing the density of state transistions, it becomes much easier for the PLL at the receiving end to extract the clock signal from the data, lock onto the clock, and minimize jitter (and bit errors).

### 5.3.11 Scrambling Rules

The first three bits:Invert,!type and type should not be scrambled. For Symbols that need to be scrambled, the least significant bit is scrambled first and the most significant bit is scrambled last. The scrambling function can be implemented with one or many Linear Feedback Shift Registers(LFSRs) on a multi-Lane Link.

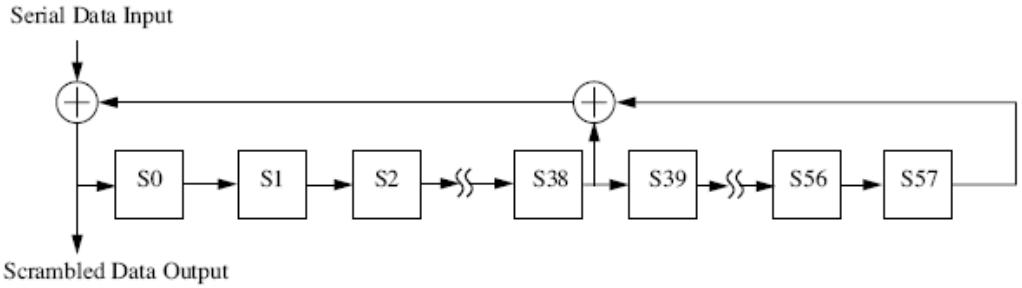



Figure 5.13: Scrambler

Table 5.2: Scrambler Initialization Values

| **Lane** | **Initilization Value** $[x^1 - x^{58}]$ |
|---|---|
| 0 | 0x1ec_f564_79a8_b120 |
| 1 | 0x1a1_af7d_7264_5f9e |
| 2 | 0x2ef_2b62_302b_d094 |
| 3 | 0x14a_da90_3a26_68aa |
| 4 | 0x1fe_d572_55e7_da1d |
| 5 | 0x283_8ff2_c69c_3618 |
| 6 | 0x3bc_111d_3429_3ece |
| 7 | 0x1c0_3994_44ae_4a2b |
| 8 | 0x09f_ebc2_faee_77fb |
| 9 | 0x239_7200_3b8e_9cff |
| 10 | 0x00a_45db_c14e_f218 |
| 11 | 0x36d_3a42_6876_e9c4 |
| 12 | 0x2c1_a537_55d7_8dea |
| 13 | 0x2b9_e833_dd9d_6b34 |
| 14 | 0x1cb_c090_ab7f_79b3 |
| 15 | 0x26f_aa25_7342_3ae5 |

### 5.3.12 Descrambler Synchronization

Each lane descrambler shall synchronize itself to the data stream it is receiving by using Seed ordered sequences.If a descrambler sync test fails while receive_enable is asserted, an initialized port shall immediately enter the Input Error-stopped state if it is not already in that state and resynchronize the descrambler.

### 5.3.13 Selective Codeword Inversion

Selective codeword inversion is used to bound the running disparity of the signal transmitted over each lane of a LP-Serial link that uses 64b/67b encoding.

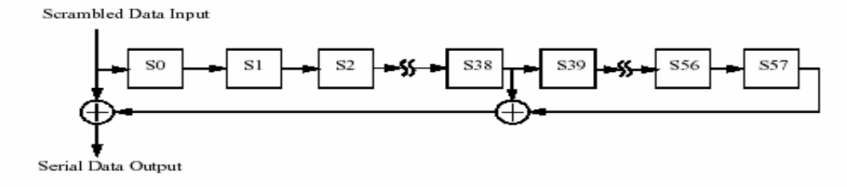



Figure 5.14: Descrambler

**Selective Codeword Inversion Rules**

Selective codeword inversion shall be applied to the signal transmitted over each lane of an LP-Serial link according to the following rules.

1. The transmitter shall start with 0 as the initial value for the running disparity calculation for each lane.
2. After each codeword is formed and if appropriate, scrambled, compute the disparity of the resulting codeword.
3. If the signs of the codeword disparity and the running disparity of the lane over which the codeword will be transmitted are different, the codeword shall be transmitted as is without inversion. The running disparity at the end of the codeword shall be the running disparity at the beginning of the codeword plus the disparity of the codeword.
4. If the signs of the codeword disparity and the running disparity of the lane over which the codeword will be transmitted are the same, invert bits [0,3:66] of the codeword and transmit the resulting codeword. The running disparity at the end of the codeword shall be the running disparity at the beginning of the codeword minus the disparity of the codeword before inversion.
5. At the receiver, invert bits [0,3:66] of each received codeword if codeword bit[0] = 0b1 (the codeword was inverted before transmission).

## 5.4 Transmission Order

The parallel 67-bit codeword output of the encoder shall be serialized and transmitted with bit 0 transmitted first and a sequential bit ordering towards bit 66. This is shown in Figure. Figure gives an overview of a set of characters passing through the encoding, serializing, transmission, deserializing, and decoding processes. The left side of the figure shows the transmit process of encoding a character stream using 64b/67b encoding and the 67-bit serialization. The right side shows the reverse process of the receiver deserializing and using 64b/67b decoding on the received codewords. The dotted line shows the functional separation between the PCS, that provides 67-bit codewords, and the PMA Layer that serializes the codewords. The drawing also shows on the receive

side the bits of the type field containing the pattern that is used by the receiver to establish 67-bit codeword boundary synchronization.

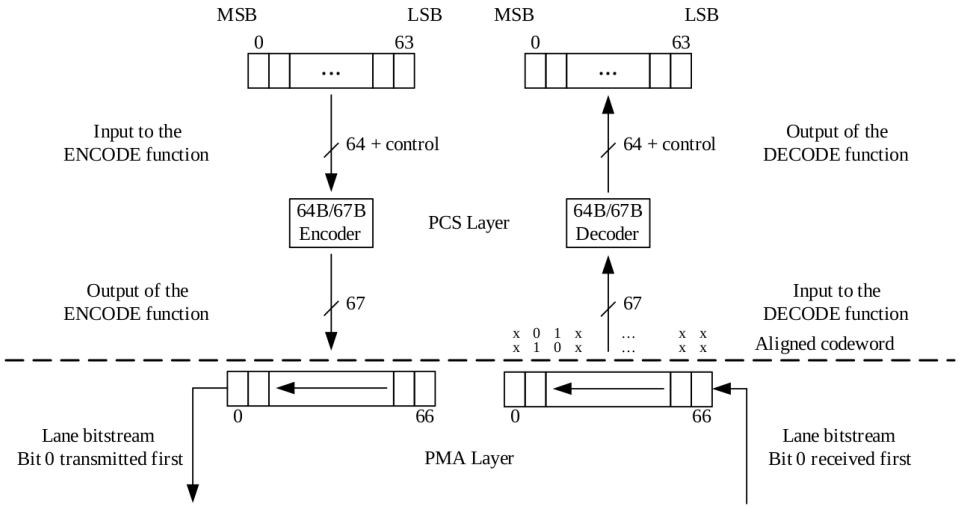



Figure 5.15: Transmission

## 5.5 Ordered Sequencing

To facilitate error detection, the Seed, Status/Control, Lane Check, Skip Marker and Skip control codewords shall be transmitted only in "ordered sequences". Each ordered sequence is comprised of a sequence of two or more control codewords with fixed ordering and with sufficient known content and redundancy to detect corruptions in a received ordered sequence.

| **Seed ordered sequence** |
| --- |
| Seed Control Codeword |
| Seed Control Codeword |

Figure 5.16: Seed Ordered Sequence

| **Status/Control sequence** |
| --- |
| Status/Control control codeword |
| Status/Control control codeword |

Figure 5.17: Status Ordered Sequence

| Skip ordered sequence |
| --- |
| Skip-Marker control codeword |
| Skip control codeword |
| Skip control codeword |
| Skip control codeword |
| Lane Check control codeword |
| Seed control codeword |
| Seed control codeword |

Figure 5.18: Skip Ordered Sequence

### 5.5.1 IDLE Sequencing

The idle sequence defined for 64b/67b encoded links is referred to as IDLE3.
The IDLE3 sequence is a sequence of codewords transmitted by a LP-Serial port on each of its active output lanes when the port is not initialized, and when the port is initialized and there are no packets or control symbols to transmit. The IDLE3 sequence enables a LP-Serial receiver to acquire and retain bit, codeword and lane alignment, as well as supporting clock compensation.

The IDLE3 Sequence shall be a continuous sequence of ordered sequences and data codewords containing pseudo-random data. Data codewords containing pseudo-random data will be referred to as ”pseudo-random data codewords”. The exact sequence of ”ordered sequences” and pseudo-random data codewords comprising a specific IDLE3 sequence is implementation dependent. The IDLE3 sequence shall be generated according to the following rules.

1. Pseudo-random data codewords shall be generated by first forming data codewords filled with bytes of 0x00 and then scrambling those data codewords with the transmitters per lane scrambler(s).
2. An ordered sequence, once begun, shall be transmitted in its entirety.
3. When IDLE3 sequence is being transmitted: A Status/Control ordered sequence shall be transmitted once every 18 to 53 codewords transmitted per lane. A Seed ordered sequence shall be transmitted at least once every 53 codewords transmitted per lane. The Seed ordered sequences transmitted as part of Skip ordered sequences can be counted as part of the Seed ordered sequences that are transmitted to meet the minimum Seed ordered sequence transmission rate.
4. If a port is transmitting in 1x mode: The IDLE3 sequence may begin with a pseudo-random data codeword or any ordered sequence. An ordered sequence may begin at any codeword boundary that is not interior to another ordered sequence. The IDLE3 sequence may be terminated after the last codeword of an ordered sequence or after any data codeword.

5. If a port is transmitting in a multi-lane mode: The IDLE3 sequence begins at a column boundary An IDLE3 sequence shall be transmitted in parallel on all active lanes The sequence of ordered sequences and pseudo-random data codewords shall be exactly the same for all active lanes. The IDLE3 sequence and each ordered sequence in the IDLE3 sequence shall begin in the same column for all active lanes and shall end in the same column for all active lanes, i.e. the IDLE3 sequence and all ordered sequences in the IDLE3 sequence are aligned across the active lanes. The IDLE3 sequence may begin with a pseudo-random data codeword or any ordered sequence, subject to the following restriction on Status/Control ordered sequence spacing. Status/Control ordered sequences shall be separated by at least 16 non-Status/Control codeword columns, regardless of whether the last Status/Control ordered sequence was part of this IDLE3 sequence or a previous IDLE3 sequence. (This requirement is to ensure that Status/Control ordered sequence columns that are used for lane alignment are separated by a minimum of 16 codeword columns.) An ordered sequence may begin at any codeword column boundary that is not interior to another ordered sequence.

**Idle Sequence 3 Generation**

A primitive polynomial of at least 7th degree is recommended as the generating polynomial for the pseudo-random sequence that is used in the generation of the idle sequence. The polynomials $[x^7 + x^6 + 1]$ and $[x^7 + x^3 + 1]$
are examples of primitive 7th degree polynomials which may be used as generator polynomials. The pseudo-random sequence generator is clocked (generates a new pseudo-random sequence value) once per idle sequence codeword (column). Five of the pseudo-random sequence generator state bits may be selected to generate the pseudo-random value for Status/Control ordered sequence spacing. The selection of the state bits and their weighting has an impact on the distribution of values for Status/Control ordered sequence spacing. One way to achieve the random spacing requirements from Section 5.9.1 is to repeatedly send one of the following sequences depending on the need to transmit Skip ordered sequences:

1. Sequence starting with a Seed ordered sequence: One Seed ordered sequence A pseudo-random number between 14 and 45 of Data codewords A Status/Control ordered sequence
2. Sequence starting with a Skip ordered sequence: One Skip ordered sequence A pseudo-random number between 9 and 40 of Data codewords A Status/Control ordered sequence

   The above sequences provide the required spacing of 16 to 47 codewords between Status/Control ordered sequences. Transmission of Skip ordered sequences should be minimized, as completion of the seven codeword Skip sequence delays the start of packet transmission. It should

be kept in mind that transmitting Skip ordered sequences too often can impact link efficiency if packets arrive for transmission when a Skip ordered sequence is being transmitted. This is because the Skip ordered sequence is seven codewords long and it has to be completed before packet transmission can start.

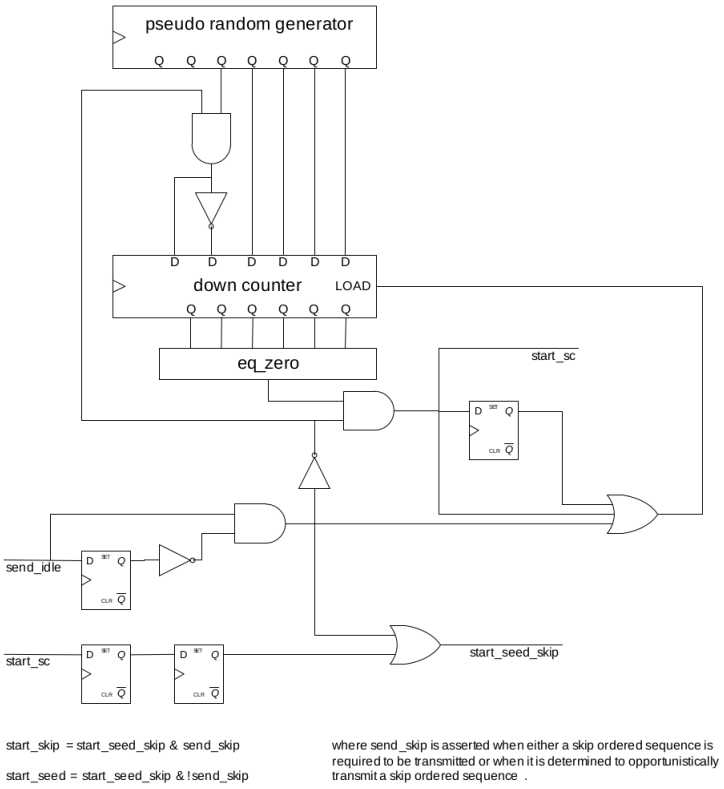



Figure 5.19: IDLE sequence generator

### 5.5.2 Frame Lock Process

The recovery of DME training frame boundaries in a lane receiver shall be controlled and monitored by the Frame_Lock state machine. There shall be one Frame_Lock state machine for each lane receiver. The frame lock state diagram determines when the PCS control function has detected the frame boundaries in the received data stream.

**frame_lock**
Boolean variable that is set to TRUE when the receiver acquires training frame delineation and is set to FALSE otherwise.
**good_markers**
Count of the number of consecutive frame marker matches.
**bad_markers**
Count of the number of consecutive frame marker mis-matches.

**SLIP**
Causes the next candidate frame sync position to be tested. The precise method for determining the next candidate frame sync position is not specified and is implementation dependent. However, an implementation shall ensure that all possible bit positions are evaluated.

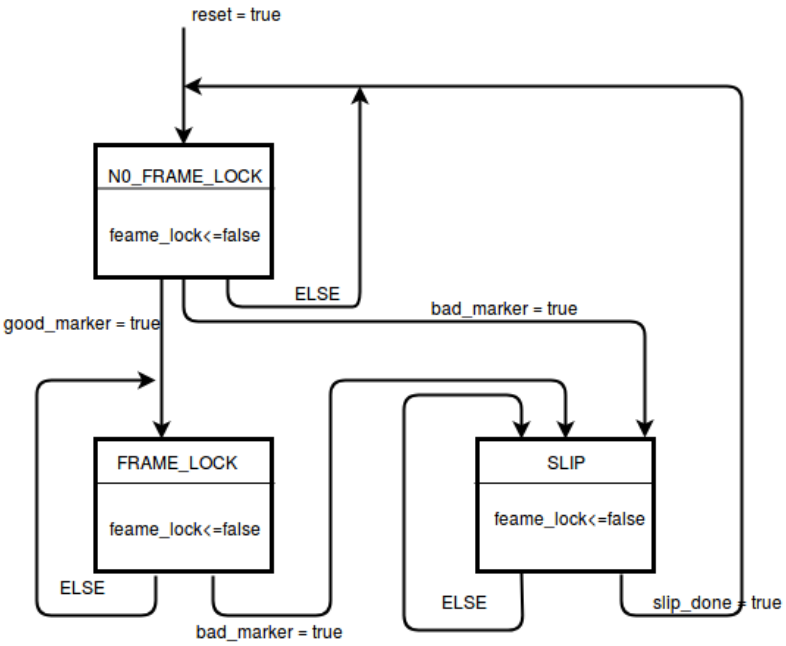



Figure 5.20: Frame Lock State Machine

# CHAPTER 6

# Experiments and Result

This cahpter describes the experimental setup. It includes data sets and configuration of Bluespec Tool. It also deccribes, timing diagram of simulations performed in BlueSim and the complete functionality was verified by writing a number of testcases.

## 6.1 Verification Setup

The Bluespec code that is written in .bsv format is given to the Bluespec compiler in the Bluespec Development Workstation (BSW).The Bluespec Compiler compiles and generates the Verilog code in .v format.

Table 6.1: COMPARATIVE OVERVIEW 8B/10B AND 64B/67B ENCODING

| **Encoding Tech.** | **8B/10B** | **64B/67B** |
|---|---|---|
| Run Lendth | 5 | Relies on Scrambler |
| DC Balance | Running Disparity of 2 | Not Guaranteed |
| Word Synchronisation | ”Comma” K-Characters | Synchronisation Header |
| Implementation Technique | Scrambler and Descrambler | Scrambler and Descrambler |
| Overhead Fraction | 25 percentage | 3.175 percentage |

Input= 11111111111111111111111111111111111111111111111111111111100001111
one count = 60
zero count = 4
one count = 64
Running Disparsity of Data = 56

## 6.2 Simulation

The simulated result of Encoded bit.

The simulated result of Scrambler and Descrambler.
Decoded 64bit

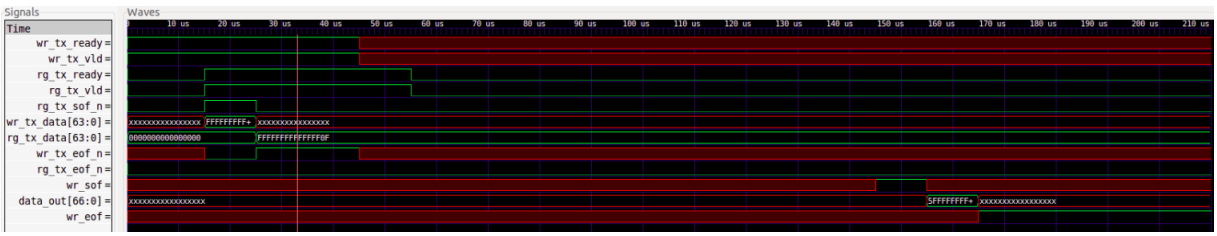



Figure 6.1: Encoding of 64 bit

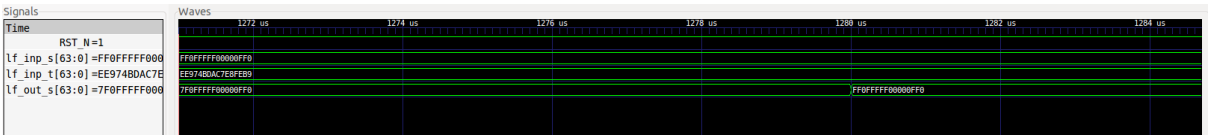



Figure 6.2: Scrambling and Descrambling

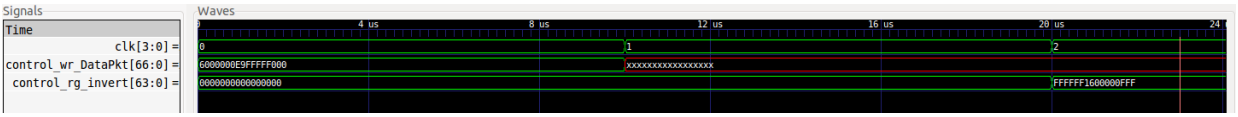



Figure 6.3: Decoding of 64 bit

# CHAPTER 7

# CONCLUSION AND FUTURE WORK

In this project, a encoding of 64 bit codeword was done to 67bit,it was developed at the physical layer level of SRIO, the data packet and control symbol was coming from upper layer . Also I designed the differnt control symbol for my packets.Like when packet is ready to go from transmitter to reciever start of packet control symbol will be generated. When the packet will completly transferred then the end of packet will be generated. These control symbol later used in encoding. For making control symbol control codeword. In future we can implement lane stripping and lane alignment.For that we need different states machine as per our requirement of RapidIO specification.

## 7.1 REFERENCE


1. A. G. Sam Fuller, RapidIO: The Embedded System Interconnect. *Wiley*, (2010).
2. D. Kanter, The common system interface: Intels future interconnect,. *Wiley*, (2007).
3. R. S. Nikhil and K. Czeck, BSV by Example. *Bluespec, Inc*, (2010).
4. B. Inc., Bluespec System Verilog. *Bluespec*, (2010).
5. http://www.rapidio.org/rapidio specifications/, RapidIO Specification 4.0 *RapidIO.org*, (2016).
6. https://www.xilinx.com/support/documentation/ipdocumentation/srioug50/, Xilinx Serial RIO UserGuide. *Xilinx*, (2011).
7. Bijan Raahemi, ERROR CORRECTION ON 64/66 BIT ENCODED LINKS. (2005).
8. 802.3-2015_SECTION5.pdf, (2005).