Verification Component Development for Interlaken Protocol

Major Project Report

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology

in

Electronics & Communication Engineering

(VLSI Design)

By

Rabadiya Vipul Mansukhlal

(13MECV22)

Electronics & Communication Engineering Branch
Electrical Engineering Department
Institute of Technology
Nirma University
Ahmedabad-382 481

May 2015

Verification Component Development for Interlaken Protocol

Major Project Report

Submitted in partial fulfillment of the requirements
for the degree of

Master of Technology
in
Electronics & Communication Engineering
(VLSI Design)

By

Rabadiya Vipul Mansukhlal

(13MECV22)

Under the guidance of

External Project Guide:
Mr. Gaurav Dave
Member, Technical Staff,
eInfochips Pvt. Ltd.,
Ahmedabad.

Internal Project Guide:
Prof. Akash Macwan
Assistant Professor (EC Dept.),
Institute of Technology,
Nirma University, Ahmedabad.

Electronics & Communication Engineering Branch
Electrical Engineering Department
Institute of Technology
Nirma University
Ahmedabad-382 481
May 2015

Declaration

This is to certify that

a. The thesis comprises my original work towards the degree of Master of Technology in VLSI Design 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.

- Rabadiya Vipul M.

Certificate

This is to certify that the Major Project entitled **"Verification Component Development for Interlaken Protocol "** submitted by **Rabadiya Vipul Mansukhbhai .(13MECV22)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in VLSI Design , 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

Prof. Akash Macwan
Internal Guide

Mr. Gaurav Dave
External Guide

Dr. N.M. Devashrayee
PG co-ordinator

Dr. P.N.Tekwani
Head of EE Dept.

Dr. K Kotecha
Director, IT-NU

Acknowledgements

I would have never succeeded in completing my Thesis without the cooperation, Encouragement and help provided to me by various people.

Firstly, my sincere thanks to the **INFIREA team** during this training. Their wisdom, clarity of thought and support motivated me to bring this project to its present state.

I wish to thank **Mr. Gaurav Dave** (Group Manager) and **Mr.Chetan Shah** (Mentor) for giving me an opportunity to work with them. I wish to place on record my gratitude to **EINFOCHIPS PVT. LTD**, Ahmedabad, for providing me an opportunity to work with them. My stay in the organization has been a great learning experience and a curtain raiser to an interesting and rewarding career.

I wish to express my deep gratitude towards my guide **Prof. Aakash Macwan** and **Dr. N. M. Devasharyee**, my faculty coordinator for guiding me and helping me throughout this training period.

Finally, I would like to thank my family for their interest and never-ending support during my studies.

- Vipul M Rabadiya

13MECV22

Abstract

Interlaken protocol is a networking protocol, which transmit data in Gbps speed. Interlaken is a high speed interconnect protocol with high bandwidth and easier packet transfers. Interlaken also uses serial links for a logical connection between components with backpressure capability, logical channels and data-integrity. Advantage of interlaken protocol is high speed, we can use multiple number of logical channel, also can used large number of lanes. Interlaken has also more secure with crc24, crc32 error checking logic.

Interlaken protocol is verify by using UVM with system verilog coding, in verification, verify the DUT of design by designer engineer. Implement same DUT functionality and compare both logic for verification.

eInfochips At A Glance

- eInfochips is a partner of choice for Fortune 500 companies for product innovation and hi-tech engineering consulting. Since 1994, eInfochips has provided solutions to key verticals like Aerospace & Defense, Consumer Electronics, Energy & Utilities, Healthcare, Home, Office, and Industrial Automation, Media & Broadcast, Medical Devices, Retail & e-Commerce, Security & Surveillance, Semiconductor, Software/ISV and Storage & Compute.

- Covering every aspect of the product lifecycle, eInfochips draws from an experience of building 500+ products that have over 10 Million units deployed âĂŞ to provide solutions on Product Design and Development, QA and Certifications, Reengineering, Sustenance and Volume Production. Being an innovation driven company, 5% of our revenues are earmarked for building reusable IPs that will accelerate product design cycles and reduce product risks.

- About 80% of eInfochips business comes from companies with revenues over $1 Billion, and 60% of total business from building life and mission critical products. eInfochips has the experience, expertise and infrastructure to deliver complex, critical and connected products.

- Today, more than 1200 chipmates operate from over 10 Design Centers and dozen Sales Offices spread across Asia, Europe and US.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The high speed chip to chip interface protocols for networking application are **xaui** and **spi4.2**. **spi4.2** is more useful in per-channel backpressure, channelization and programmable burst sizes. The width of the interface limit is scalability and source synchronous nature. When **xaui** is a 4 lane interface, long reach, and variety of implementations like backplanes, FR4 on PCB & cable. Packet based interface has a several applications. Both protocols has a fixed configurations, limiting the ability of the designer interface and limiting the capacity to application. So define a new protocol- **Interlaken**. interlaken enables the design of a high-speed, narrow, channelized packet interface.

Interlaken is high speed, narrow chip to chip interface. There are two fundamental structures in the Interlaken Protocol- 1-Data transmission format and 2-Meta Frame. The data transmission format basically on the concepts of the **spi4.2** protocol. If data sent across the interface then it's divided into number of bursts, which are subsets of original data. Every burst is bounded by two control words, one before data burst and one after data burst. These control words affect the data following for functions like sop(start-of-packet), eop(end-of-packet), error detection, crc24 bits, channel no, and other. Each burst is define with a logical channel. this logical channel can represent a physical networking port in the system. Packet data is transmitted sequentially one by one data bursts and the size of the bursts is a random parameter. Dividing the

data into data bursts, the interface also allows the interleaving of data transmissions with different channels. The MetaFrame is defined the transmission of the data burst over a SerDes lanes. MetaFrame has a set of four unique control words, which are defined as a synchronization word(lane alignment), scrambler word, clock compensation (skip word), and diagnostic word functions. The MetaFrame runs in-band(IB) with the data transmissions, using the perticular formatting of the control words from the data.

This report also presents by using UVM, how to system verilog program will be easier. Work of Interlaken protocol using system verilog and UVM. Interlaken protocol is more useful than other networking protocol, also its high speed communication protocol.

The content is organized as follows.

- Chapter 2 Features of System verilog .
- Chapter 3 UVM Testbench and Testcase.
- Chapter 4 Verification of Interlaken protocol.
- Chapter 5 Conclusion.
- Chapter 6 Referrence. And appendix.

Chapter 2

Basic of System Verilog

This chapter include starting from the basic of system verilog. Discuss about data types, array, task and function. Define a class which is a heart of SYSTEM VERILOG. Also include interface and clocking block.

2.1 Introduction of system verilog

Using SystemVerilog, we can improves the readability, productivity, and reusability of Verilog codes. The SystemVerilog provide more easy hardware descriptions language and providing an easy route. random testbench development, assertion based verification and coverage driven verification(CDV) is very easy by using Systemverilog.

The Accellera provides a higher level of abstraction for verification with Verilog Hardware Description Language (VHDL).

SystemVerilog adds extended and new constructs to Verilog likes data types, queues, casting, Enhanced process control, Enhanced tasks and functions, Classes , random constraints, semaphores, mailboxes, Clocking blocks, program block, interface, Functional coverage, Direct Programming Interface (DPI), system task and system function.

2.2 Data types

SystemVerilog supports the C language, also implementation of C compiler. However, in verilog has a data type like int and long. so avoid these duplication of int and long without more change, in SystemVerilog, int is 32 bits and longint is 64 bits. The float data type is called shortreal in SystemVerilog, so it is not be confused with the Verilog data type.

2.2.1 Integer data types

Integral is represent a single integer data type. this type can have high-impedance (Z) values are called four state types. These are logic, integer, time and reg. all these have 4 values. The other types do not have unknown(high impedance) values and only have a 2 values (0,1). these are called 2 state types, for example int and bit.

Data Types	Specifications
Int	two state SystemVerilog data type, 32 bit signed integer
Shortint	two state SystemVerilog data type, 16 bit signed integer
Longint	two state SystemVerilog data type, 64 bit signed integer
Integer	four state Verilog-2001 data type, 32 bit signed integer
Bit	two state SystemVerilog data type, user defined vector size
Byte	two state SystemVerilog data type, 8 bit signed integer or ASCII character
Time	four state Verilog-2001 data type, 64 bit unsigned integer
Logic	four state SystemVerilog data type, user defined vector size
Reg	four state Verilog-2001 data type, user defined vector size

Table 2.1: Integer data types

The difference between integer and int is that integer is four state logic and int is two state logic. 4-state values have additional bits that are the X(unknown value)

and Z(high impedance value) states. 2-state data types can simulate fasterand also take less memory.
Integer only use integer arithmetic value and can be signed or unsigned. This can affects the certain operators. The data type int, longint, byte, shortint and integer default to sign. The data types reg, logic and bit default to unsigned.

2.2.2 String data type

SystemVerilog has a string data type. string data type has a variable size. Basically string is dynamically allocated array of bytes. SystemVerilog has a lots of special methods for strings. Verilog supports string literals, but at the lexical level.
In SystemVerilog string data type is the same as in Verilog. However, SystemVerilog supports the string data type. When using the string data type instead of integral variable. Literal strings are converted to the string type when assigned to a string type.

2.2.3 User-defined types

Data type User defined type identifiers have the same rules as data identifiers type, except hierarchical reference identifiers shall not be allowed. User defined identifiers defined within an interface through ports. they are redefined before used. We can define data type as a random number of bits.

2.2.4 Class

A class is a set of subroutines and collection of data that operate on data. Class is a one type of data type, it's a heart of data types. The data in a class are referred to class subroutines, and its properties are called methods. Class is a main data type of the SystemVerilog. All functionality are called by using class.

2.3 Array

An array is a set of variables, all are same type. accessed using the same name. In C language, arrays are indexed from zero by integers but in SystemVerilog the array can be initialized, each element must be write or read separately in statements. SystemVerilog has a packed and unpacked arrays with multiple dimensions. SystemVerilog has the ability to change the size of one of the dimensions of unpacked array. Unpacked arrays with fixed size can be multi-dimensional and have fixed storage allocated for all the elements of the array. A dynamic array allocates storage for elements at runtime change the size of its dimensions.

2.3.1 Packed and unpacked arrays

There are two types of arrays : 1. packed array and 2. unpacked array. A packed array is subdividing a vector into subfields which can be array elements. A packed array is to be represented as a set of bits. A packed array different as compare to an unpacked array. A packed array appears as a primary, it is present as a single vector. If a packed array is declared as signed array, then the array viewed as a single vector shall be signed.

Packed arrays allow any of length integer types, so a 64 bit longint can be made up of 64 bits. These 64 bit longint can be used for 64 bit arithmetic. The maximum size of a packed array is limited, it's at least 65536 (216) bits. Packed arrays can only be made of the single bit types like wire, bit, logic, reg and the other net types are: byte, int, shortint, longint, and integer.

2.3.2 Dynamic arrays

A dynamic array is one dimensional unpacked array. Size of dynamic array not be fixed earlier but can be set at runtime value. The space for a dynamic array doesn't exist until the array is created at runtime. Array size is defined at run time.

The syntax to declare a dynamic array is:

data_type dynamic_array_name [];

In the syntax the data_type is the data type of the array elements. Dynamic arrays support the same types as fixed-size arrays.

2.3.3 Associative arrays

Dynamic arrays are useful for runtime changes and dealing with collections of variables with changes size dynamically. When the data space is fixed or the size of the collection is unknown then an associative array is a better option. this array do not have storage allocated until it is not used. An associative array implements of the elements of its declared type. The data type to be used as an index serves as a key and imposes an ordering.

The syntax of an associative array is:

data_type associative_array_id [index_type];

2.4 Tasks & Functions

Verilog has static task and function and automatic task and function. Static task and function has the equal storage space for all call of the task or function. Automatic task and function allocate unique storage for all instance of task and function. SystemVerilog has an ability to declare automatic variables within static task and function, and also static variables within automatic task and function.

SystemVerilog also adds: More capabilities for declaring tasks and functions. Like Function inout and output ports, Void functions.task and function has a ability of Multiple statements without requiring any other block like a fork...join or begin...end block, Returning from a task or function before reaching the end of the task or function.

Tasks	Functions
Tasks is time delay.	Function is not depends on time.
Tasks and functions is called in tasks	Only functions can call in function.
Tasks can't return a single value.	Function can return a single value.
Gives one or more arguments.	Required at least one argument.
Multiple statement without begin...end	Multiple statement without begin...end

Table 2.2: Tasks and Functions

2.5 Classes

SystemVerilog has an object oriented class. Class is a one type of data type. Classes have an objects, that are dynamically created, deleted and assigned via object handle. Classes has features inheritance and abstract type modeling. These features gives the advantages of C function pointers with none of the type-safety problems, bringing true polymorphism.

A class includes data and subroutines means functions and tasks that operate on that data. A class data is follow to as class properties, and its subroutines are called methods, properties and methods are members of the class. The class properties and methods together define the capabilities of some kind of object.

2.5.1 Objects (class instance)

A class is a data type. An object is an instance of class. An object is used by declaring a variable of the class and then creating an object of that class. create object by using the new function and assigning it to the variable.

2.5.2 This

The 'this' keyword is used to class properties or methods of the class instance. The this keyword defines a predefined object handle that refers to the object that was used to the subroutine. by using 'this' keyword, we can use variable of super class

without define in subclass. So 'this' keyword is more useful for large programing. The this keyword will be used in non-static class methods.

2.5.3 Super

The super keyword is used in a derived class for a members of the parent class. It is necessary to use super.variable to access members of a parent class. by using super keyword, call super class without defining in super class.

2.5.4 Polymorphism

Polymorphism allows the use of a variable in the super class for subclass objects, and to referred those subclasses directly from the super class variable. without define variable in subclass, can be used variable of superclass in subclass. Assume the base class for the Packet objects, Base Packet define as virtual functions, all the methods that are to be generally used by its subclasses, methods such as send, receive, print, etc.

2.6 Interface

Interface is very useful for a communication between two components or a component to DUT or DUT to component communications. The communication between components of a digital system is a critical area that can affect everything of RTL coding. Hide the communication between components, the interface construct facilitates design reuse. The inclusion of interface capabilities is one of the major advantages of SystemVerilog.

2.6.1 Virtual interfaces

Virtual interfaces provide a mechanism for separating models and test programs from the actual signals that make up the design. A virtual interface allows the same

subprogram to operate on different portions of a design, and to dynamically control the set of signals associated with the subprogram. Instead of referring to the actual set of signals directly. Changes to the underlying design do not require the code using virtual interfaces to be re-written. By abstracting the connectivity and functionality of a set of blocks, virtual interfaces promote code reuse. A virtual interface is variable that represents an interface instance.

2.7 Summary

In this chapter studied about data types of Systemverilog , more data type compare to verilog and C++. Class is also one type of data type. In Systemverilog , also have user define data types, chandle data type, casting , string data types all are more useful for easier programming. Array is number of bits in single variable, different types of array type define in this chapter.

Class is a data types, class is a heart of systemvarilog. In class, there are different functions , this, super, inheritance, polymorphism, so many. In Task and function, variable and logic define. Task is delay time logic.

Interface is data transfer between two component. Clocking block is use for transmit data per clock.

Chapter 3

UVM Testbench

The Universal Verification Methodology(UVM) is used for developing a testbenches and also create architecture of a verification component. describes the basic concepts of uvm and uvm components like agent, driver, monitor, etc.. that make up a typical verification environment. Combine these components using a hierarchical architecture to create reusable verification components.

3.1 Introduction to UVM

For a coverage, UVM gives the best work for achieving coverage-driven verification (CDV). By using CDV, it gives combine automatic test generation(atg), self-checking, and coverage metrics to reduce the time for verifying a design.

CDV is support constrained and directed random testing method. Best approach of CDV is constrained-random testing to do the most work with less effort to writing time-consuming that are too difficult to reach randomly. CDV environment covered all the corner cases. So CDV coverage is more useful for finding verification coverage. Reusable components are more used in verification. An UVM is reusable verification component. It is called verification components. A verification component is ready for use, encapsulate, easy to use for an interface protocol. The verification compo-

nent is used for the device under test (DUT) to verify the logic of the protocol. Then compare your logic for verification. UVM is a very easy for users because it is not required any converter for design engineer to verification engineer.

Communication between UVM components are by using standard TLM interfaces, which improve secure and reuse. Using a SystemVerilog implementation of TLM, a components may communicate by its interface to any other component that implements interface. Every TLM interface port consists of 1 or more methods used to transfer data. TLM interface provide an implementation. So, one component may be connected at the transaction level to others component.

3.2 Transaction-Level Modeling (TLM)

Universal verification methodology provides a transaction level communication(TLM) interfaces and channels. it's connect components at the transaction level. By using TLM interfaces isolates each component from other components in the environment. For a coupled of phases, has a flexible build infrastructure in UVM, they have the same interfaces. All is required to replace the TLM with a thin layer of compatible components to transfer data between the transaction-level and the pin-level activity at the DUT. TLM also convert transaction level to signal level.

3.2.1 TLM-1, and TLM-2

Transaction-level modeling(TLM), is a modeling for build highly abstract models of components. In general, TLM provides an abstraction levels beginning with cycle-accurate modeling. Common transaction-level abstractions also include: approximately timed, cycle-accurate, token-level and loosely-timed. The TLM code elements used to create transaction-level models. Two TLM modeling systems: TLM-1 and TLM-2 have been developed as industry standards for transaction-level models.

TLM-1 and TLM-2 developed by many people, and also share a common heritage. TLM-1 and TLM-2 has a different function. 1 is a message passing system. TLM-2 enable to transfer a data and synchronization between two processes. bBth these facilities has been implemented in SV and is also available as a small part of UVM.

3.2.2 Transaction-Level Communication

TLM provides a lots of methods for use of transaction objects as arguments. Using TLM will communicate 2 components by using port-export. A TLM port to be used for a particular connection, while a TLM export supplies the implementation of these methods. Connecting a port-export allows the implementation to be executed when the port method is called.

- **Basic TLM Communication**

  The transaction level modeling allows one component to put a transaction to another component. The square box symbol on the producer indicates a port and the circle symbol on the consumer indicates the export. So interface from port to export. Also possible vise versa.

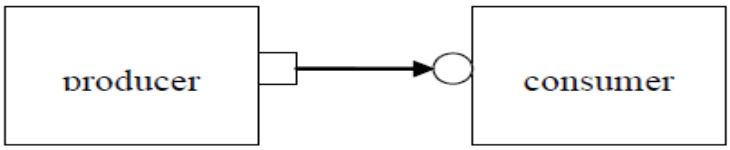

Figure 3.1: Simple Producer/Consumer

- **Communicating between Processes**

  In this processes, use fifo for transaction. The producer puts a transaction on TLM port-export, transaction put the data in fifo, transaction will block if the fifo is full, otherwise it will put the data into the fifo and return immediately. Then the get operation will return transaction data immediately if a transaction is available, otherwise it will block data on fifo until a transaction is not

available.

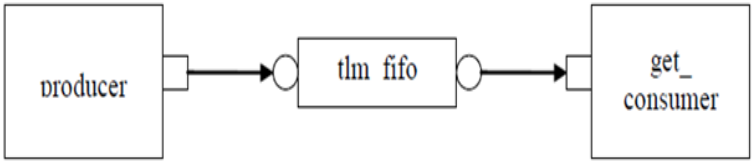

Figure 3.2: Using a uvm-tlm-fifo

3.2.3 Analysis Communication

All components are responsible for communicating using TLM interface with other components in the system. analysis communication is deffer form transaction level communication is that, analysis communication can be connected with more than one components at a same time. So in any complex verification environment, analysis communication is more usefull, particularly where randomization is applied. TLM port is interface with one to one component, where analysis port is connected with one to multiple.

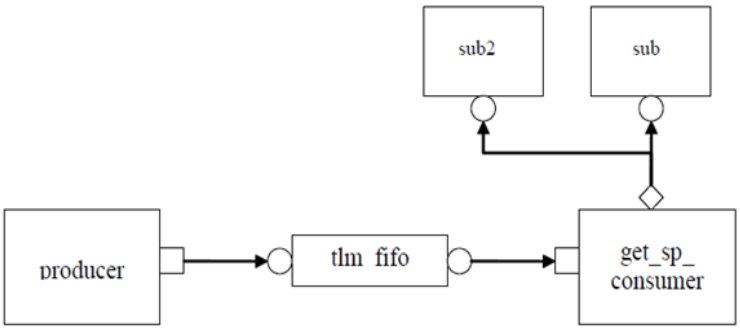

Figure 3.3: Analysis Communication

3.3 Verification Components

The basic concept and component that make a verification environment. Also combine these components using a hierarchical architecture to create verification components.

3.3.1 UVM Testbench

Uvm_env, uvm_components and uvm_test are the 3 main blocks of a testbench for a uvm based verification. In these 3 blocks, uvm_env class is extended from uvm_component class and does not contain any other functionality. Uvm_env is used for create and connect uvm_component like momnitor, driver, sequencer etc. The uvm_env class can also be used as a sub environment for other environments. There is no any difference between uvm_component and uvm_env.

All components are developed under uvm testbench. Main basic 3 components are driver, monitor and sequencer. In these 3 components, driver and sequencer are active mode where monitor is in passive mode, it's useful for find coverage.

3.3.2 Transaction-Level Components

TLM interfaces provide a set of communication methods for sending and receiving transactions between components. transaction level components first generate sequence and then transmit to the driver and convert transaction level to signal level and stimulus at the dut interface. After that, covert signal level to transaction level and transmit to the monitor. monitor transmit this data to the scoreboard by analysis component.

Transaction level verification environment are:

- A stimulus generate sequence to create at transaction-level to the DUT.
- A driver- driver convert these transactions level to signal-level stimulus and transmit to the DUT interface.
- A monitor- monitor received signal-level activity on the DUT interface and convert this signal level to transactions level stimulus.
- A coverage collector or scoreboard by using analysis components, to analyze and check transactions data.

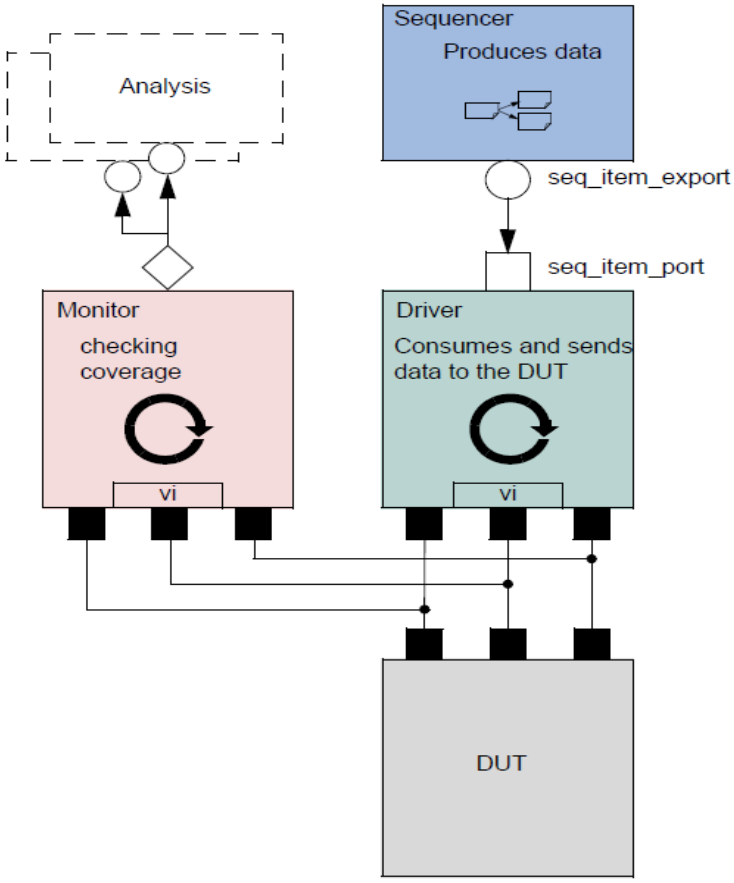

Figure 3.4: Transaction-Level Testbench

3.3.3 Creating the Environment

Environment class is extended from uvm_env class. Environment class is implementing verification environments. It covered the all general operation of transaction-level verification components. Environment class is used for how to arranged these components into a reusable environment. your environment will be correct, as compare with other verification components, and reusable.

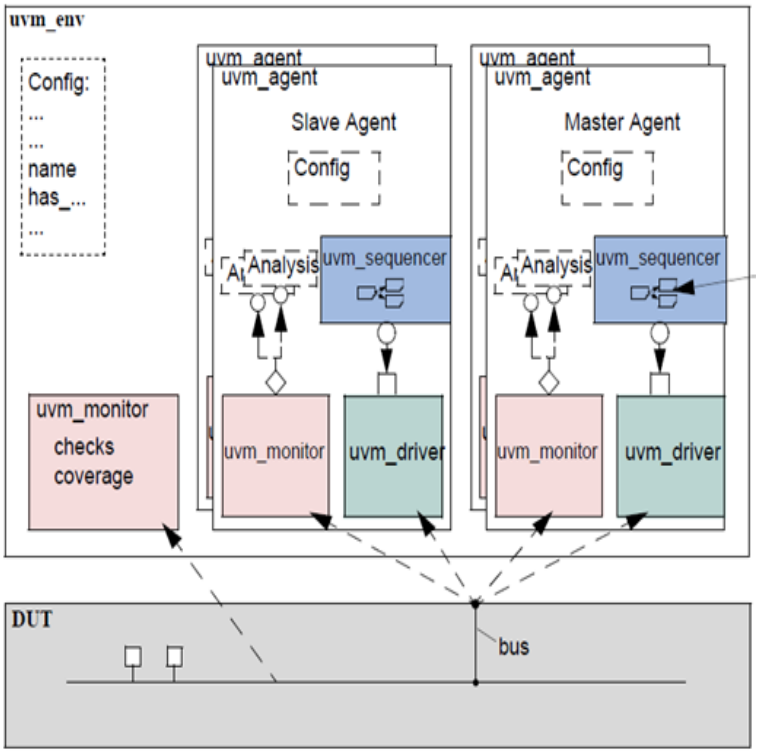

Figure 3.5: Typical UVM Environment Architecture

Below steps for define the environment class.

- Extend uvm_env class for define environment class.
- Declare the utility Marcos. For implement create() methods.
- Define the construct method for new() methods.
- Define build method. Call super.build() method.

- Define connect method, called automatically after build method.

3.3.4 Creating the Agent

3 components are extended from uvm_agent. these are driver, monitor and sequencer. An agent connects these 3 components together by using TLM connections. For better flexibility, the agent also gives configuration information and other parameters. The verification component developer creates an agent. The agent provides protocol-specific stimulus checking and coverage for a device. In an environment, an agent is either a master or a slave component. uvm_agent is extended form uvm_env class.

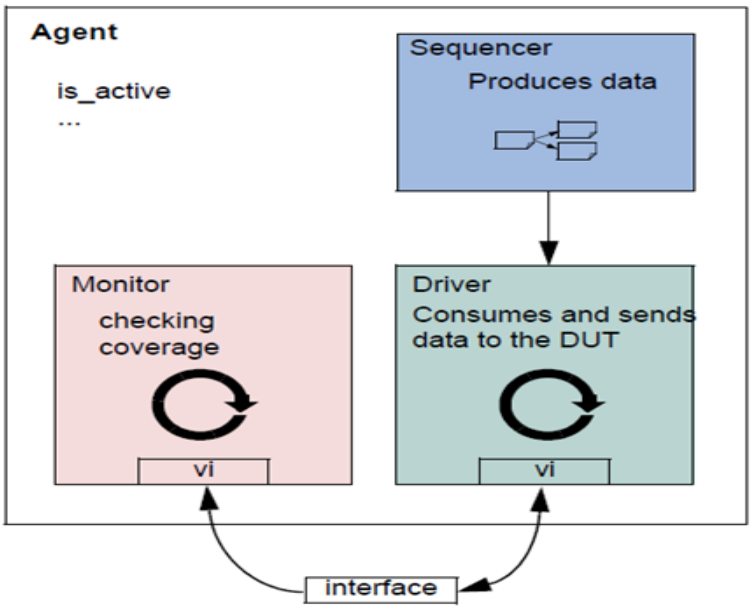

Figure 3.6: Agent- connection between component

An agent class has 2 basic operating modes:

- Active mode - In this mode the agent is in active mode, so all 3 components (driver, monitor and sequence) are active and it's worked. a device in the system and drives DUT signals. this mode reqiured that driver and sequencer also worked. A monitor is also worked for checking and coverage.

- Passive mode - In this mode the agent does not worked a driver or sequencer and operates passively. Driver and sequencer are not worked in passive mode. The monitor is only worked and configured. This mode is used when only checking and coverage collection is desired.

3.3.5 Creating the Driver

The driver is to drive data items to the interface protocol from the transaction level and sequencer. The driver takes data items from the sequencer. The UVM Class has an inbuilt base class the uvm_driver, from uvm_driver class all other driver classes should be extended, either directly or indirectly. The driver class has a run() phase, all operations and all functions execute in this phase, as well as a communicates with the sequencer by TLM ports.

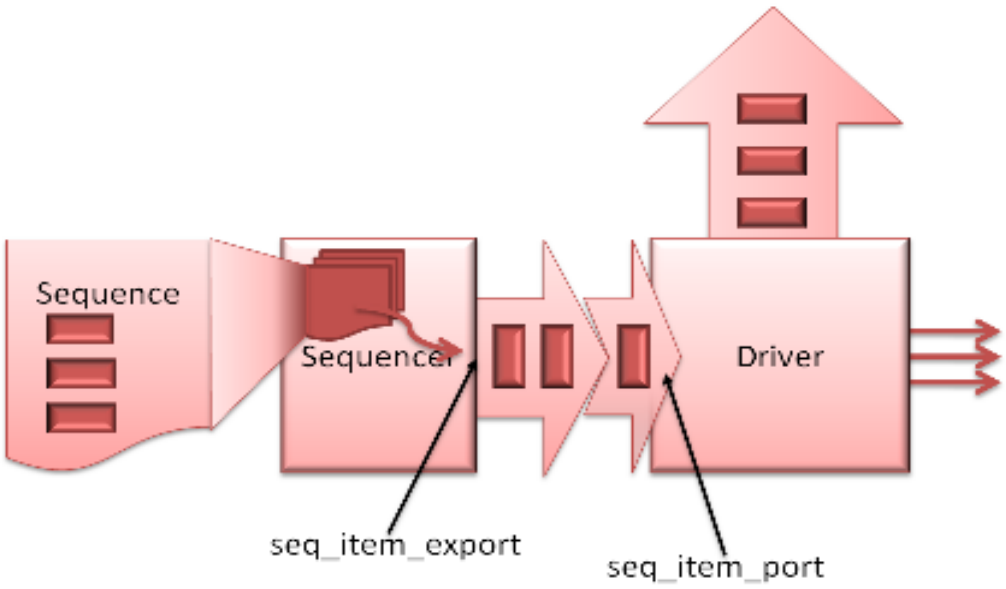

Figure 3.7: Transaction from sequencer to driver

The driver may also implement more than one parallel run-time phases to refine its operation. Data packet should be drive per clock, so it is required clocking blocks. Derive driver class from base class the uvm_driver. Then add UVM macros from

connected with factory class and take The next_data_item from the sequencer. After that, create virtual interface for connect driver to the DUT.
Driver takes transaction fron sequencer by using seq_item_port. This transaction will be driven to DUT though interface. Driver also transmit transaction to scoreboard using uvm_analysis_port.

How to create driver class:-

- **Driver logic:**

```
class sample_driver extends uvm_driver #(sample_item);
sample_item sam_item;
virtual dut_if vif;
‘uvm_component_utils(sample_driver)
// define Constructor

    function new (string name = "sample_driver", uvm_component parent);
super.new(name, parent);
endfunction : new

    function void build_phase(uvm_phase phase);
string ins_name;
super.build_phase(phase);
if(uvm_config_db#(virtual dut_if)::get(this,
" ","vif",vif))
endfunction : build_phase

    task run_phase(uvm_phase phase);
forever begin
// Next data_item from sequencer.
```

```
seq_item_port.get_next_item(sam_item);
drive_item(sam_item);
seq_item_port.item_done();
end
endtask : run


    task drive_item (input sample_item item);
endtask : drive_item
endclass : sample_driver
```

The example derives sample_driver from uvm_driver and uses methods in the seq_item_port to communicate with the sequencer class. Then, also include the ‘uvm_component_utils macro and a constructor to register the driver type with the common factory.

3.3.6 Creating the Sequencer

A Sequencer class is extended by uvm_sequencer class. uvm_sequencer is a in-built class. uvm_sequencer has a one port seq_item_export which is used to connect uvm_sequencer with uvm_driver for transfer transaction. The sequencer create transaction data and passes it to a driver.

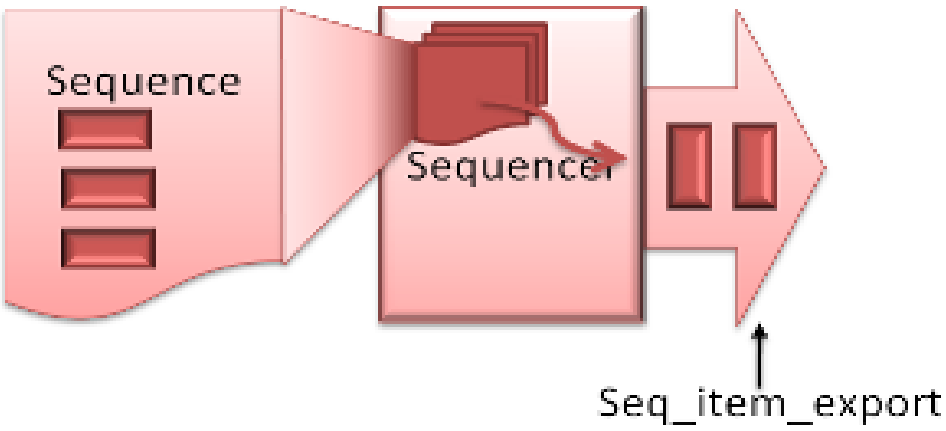

Figure 3.8: Transmit sequence from sequencer

The UVM Library has a uvm_sequencer base class. These class is parameterized by the request and response types. The uvm_sequencer class has all of the functionality that required for a sequence to communicate with a driver. The uvm_sequencer class gets instantiated directly, with appropriate parameterization. In this class the response type is the same as the request type.

3.3.7 Connecting the Driver and Sequencer

The sequencer class and The driver class are connected via TLM port-export, with the port seq_item_port of driver is connected with the export seq_item_export of sequencer. The driver send data items through its seq_item_port and sends responses. The component that instances of the driver and sequencer makes the connection between them. Interaction between the sequencer and the driver is done using the get_next_item() and item_done().

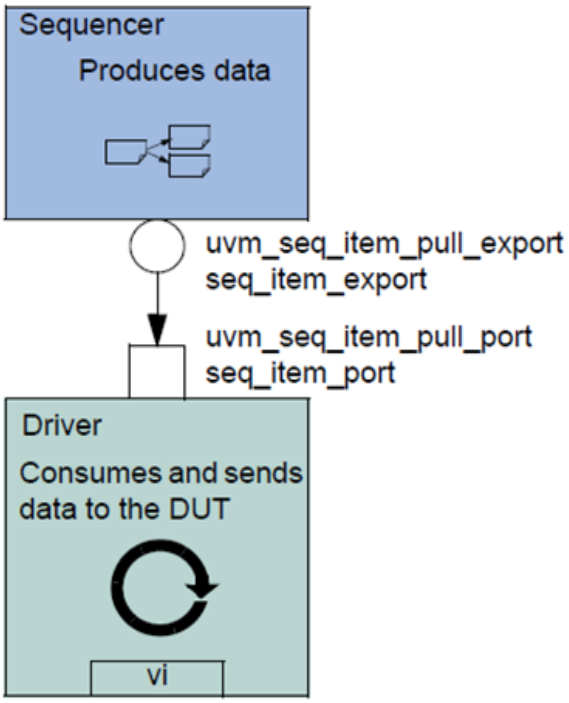

Figure 3.9: Sequencer-Driver Interaction

3.3.8 Creating the Monitor

The monitor is passive device, it's responsible for extracting data and signal information from the interface. These informations are also available to other components via standard TLM interfaces and channels.

The monitor should be limited functionality for basic monitoring. Monitor should be collect data from the interface and transfer that data to the scoreboard. This can include protocol checking-which configurable may be enabled or disabled. monitor is also used for coverage collection. Monitor has an additional high-level functionality, such as scoreboards, should be implemented separately on top of the monitor.

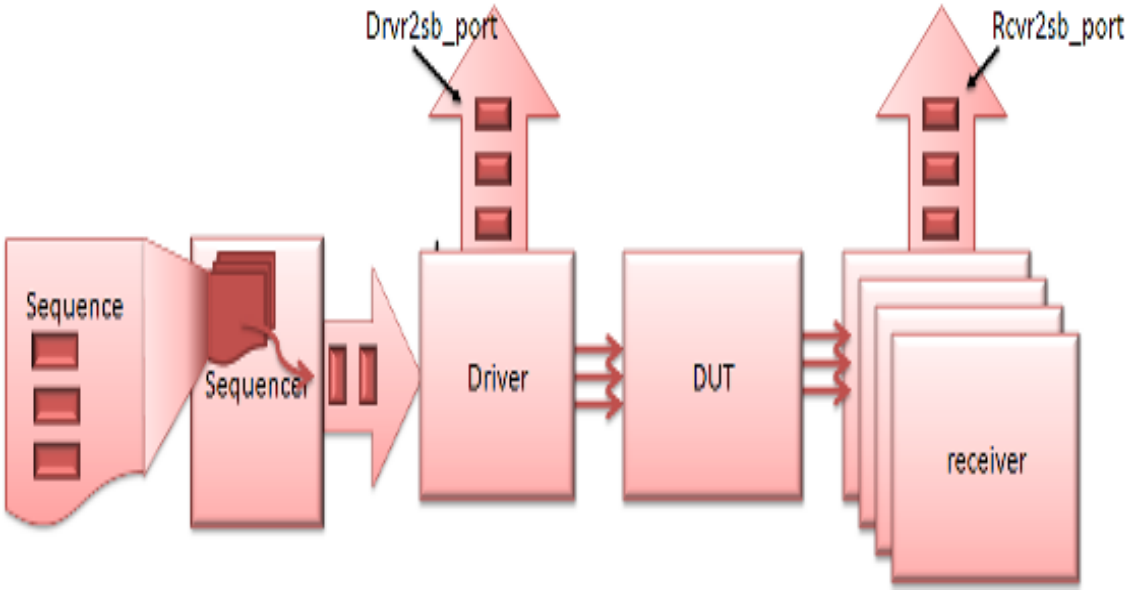

Figure 3.10: monitor receive data from DUT

The monitor has the following functionality:

- The monitor component collects information from a virtual interface.
- The monitor collect data and it's used in coverage collection and checking.
- The collected data is send on an analysis port(item_collected_port).

3.3.9 Sequence Item Flow

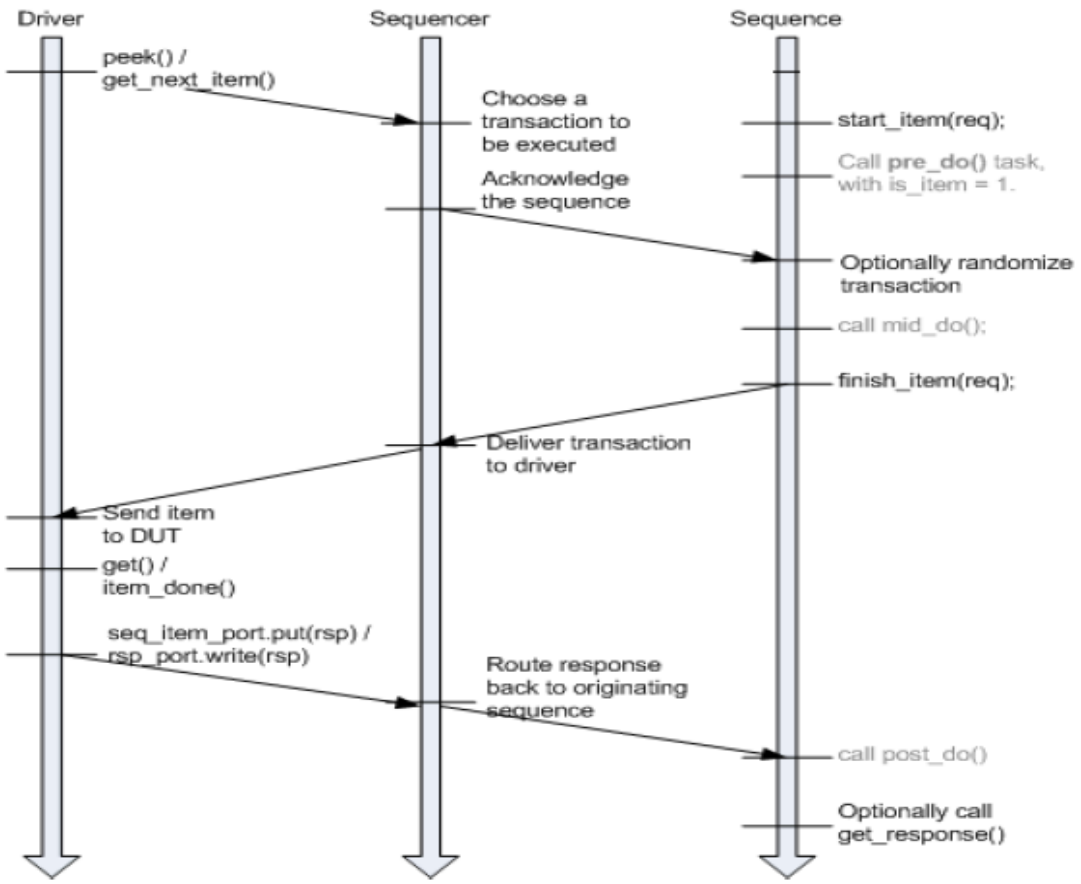

Figure 3.11: Sequence Item Flow

3.3.10 Scoreboard

Scoreboard class is extended by uvm_scorboard. The scoreboard has 2 TLM analysis ports. One is for driver, it's used for getting the packets from the driver and another for the receiver, getting the packets from the monitor. Then both the packets are compared and if they don't match, then error is occurs. For comparison, used compare () method of the Packet class.

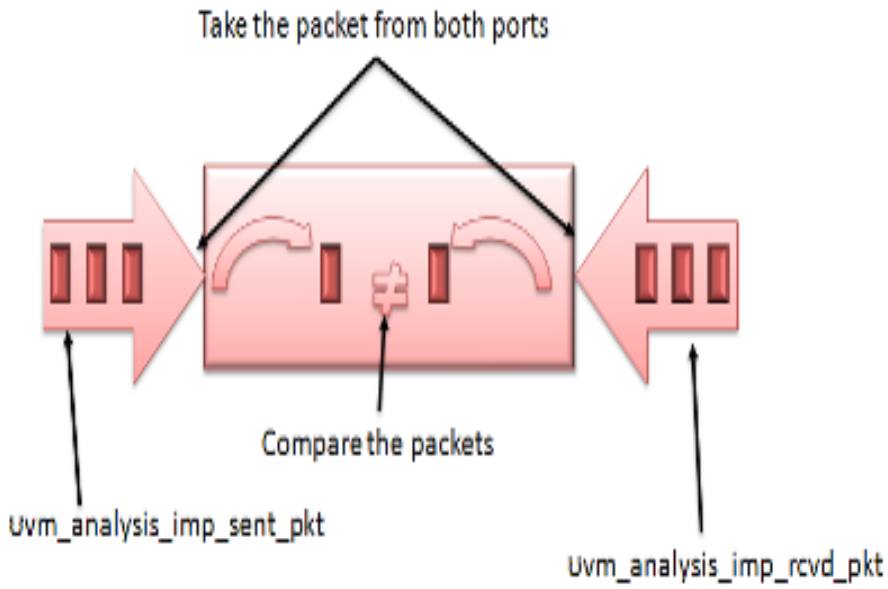

Figure 3.12: Function of scoreboard

3.4 Summary

In this chapter studied about universal verification methodology, why more used in companies. Direct communicate with testing and verification engineers. in uvm, also have user define function, so programming is more easy using this.

In uvm, testbench is more important, all other components are under top level module(testbench), environment class is under top level, all other components are in environment class. Agent have 3 components, driver, monitor and sequencer. Data is verify by using DUT(devise under test). Driver drives data to DUT and monitor receive the data from DUT by using interface.

Uvm_env, uvm_agent, uvm_driver, uvm_monitor, uvm_sequencer etc. User define classes, which are more useful for programming, also have inbuilt macros, by using Systemverilog with uvm, programming is easy.

Chapter 4

Interlaken Protocol

4.1 Introduction

Interlaken is networking protocol with configuration of a narrow, high-speed chip-to-chip interface. Interlaken Support up to 256 communications channels, and also extended up to 64K by using multiple use bits. A Meta Frame of programmable frequency synchronization, scrambler, clock compensation(skip word) from the number of SerDes lanes and SerDes rates. With a simple Xon-Xoff mechanism control both out-of-band and in-band per-channel flow control options.

In the interlaken protocol, there are main 2 structures - the data transmission format(protocol layer) and the Meta Frame. these are 2 main layer for full interlaken protocol. The data transmission format is similar concepts of SPI4.2. Data sent to the interface is divided into number of bursts, which are the subsets of the original packet burst data. Every bursts are bounded by 2 control words, 1 control word before burst data and 1 after burst data. sub fields between these control words is affect either the data following them for functions like end-of-packet(eop), start-of-packet(sop), word-lock, error bit detection, crc24, crc4 and some others. Each burst is declared with a logical channel. Channel can declare a physical networking port in

the system or some other logically connected data and also define data rate based on logical channel numbers.

MetaFrame is support the transmission of the data. MetaFrame is a set of four unique control words, which are defined as a lane alignment(synchronization word), scrambler word, skip word (clock compensation) and diagnostic word functions. The MetaFrame runs in-band flow control with the data transmissions, using the specific formatting of the control words to distinguish it from the data.

4.2 Alternatives

The two interface protocol with high-speed chip-to-chip for networking applications are **XAUI** and **SPI4.2**. SPI4.2 has important advantages in channelization and programmable burst sizes. SPI4.2 has 16 lanes but bandwidth is only 700-800 Mbps. In XAUI is a narrow 4-lane interface, but bandwidth is better than SPI4.2, its bandwidth is 3.125 Gbps, and it has a varies implementations: cable, FR4 on PCB and backplanes. Both protocols offer limited configurations, limiting the ability of the designer the interface capacity to the application.

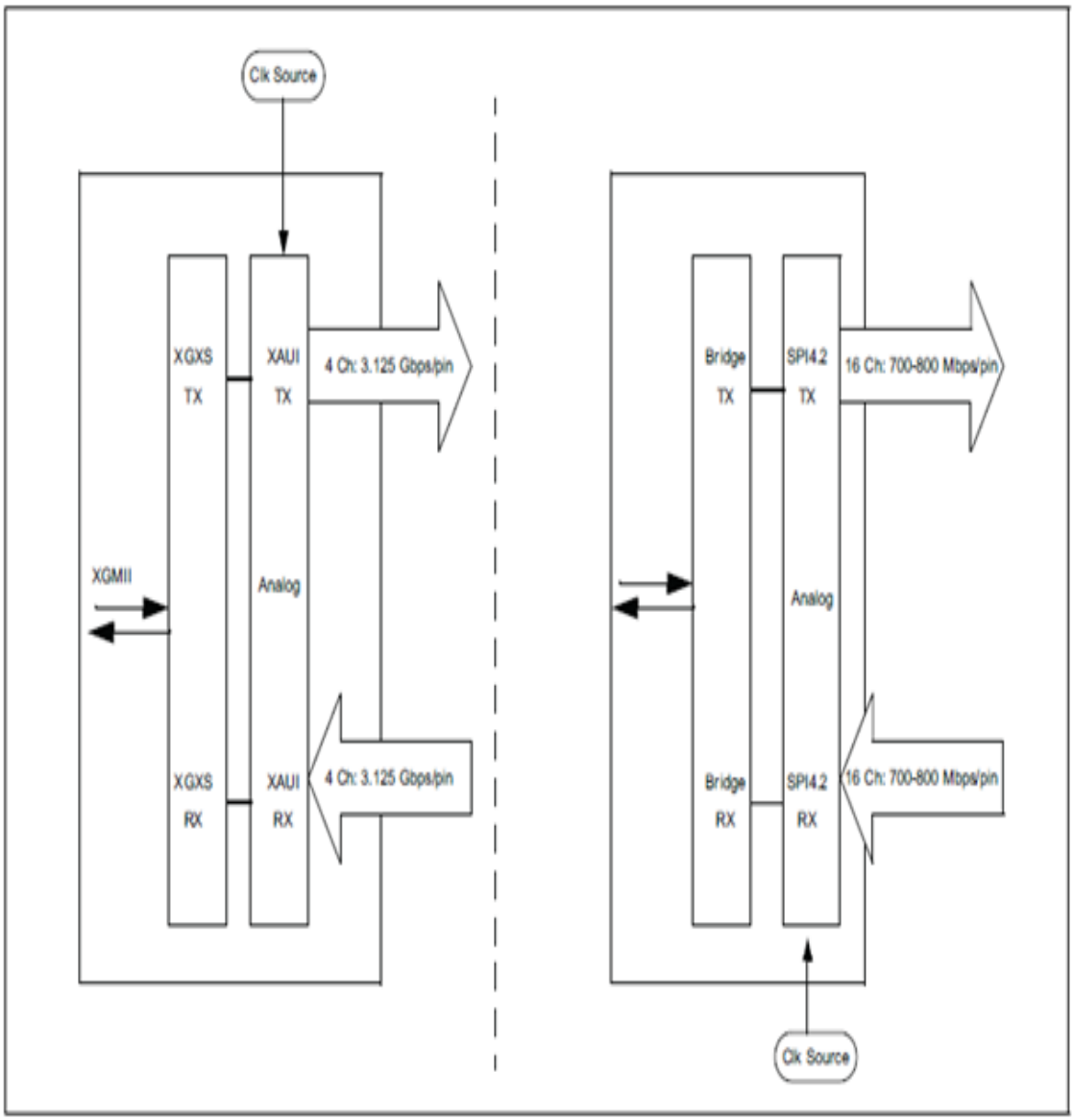

Figure 4.1: XAUI Versus SPI4.2 Interfaces

4.3 Protocol Layer

4.3.1 Transmission Format

Data burst is transmitted to the Interlaken interface via SerDes lanes. A SerDes lane is a simple serial link between two ICs. This protocol is operate with any number of lanes like 4,5,8,16,32.., including only one, with no any maximum limit. Actual implementations may choose to fix any number of lanes, because there is not any support for a variable number of lane at runtime.

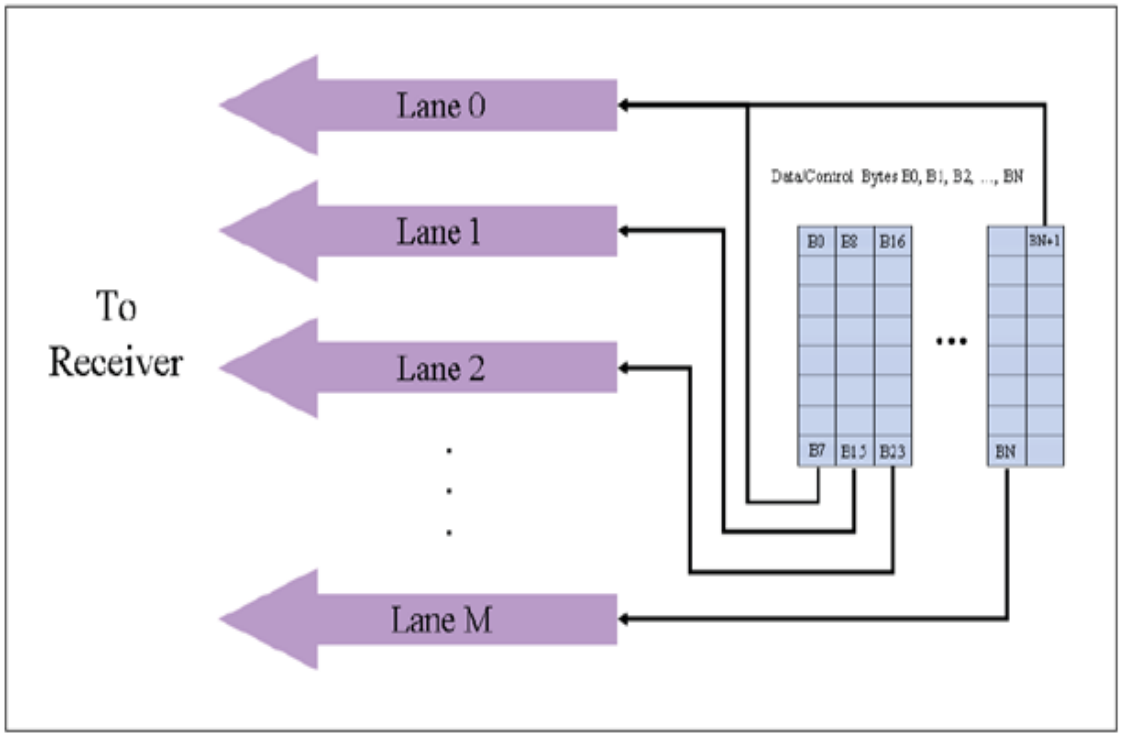

Figure 4.2: Lane Striping

Data burst sent on the interface lane is an 8-byte word means word created by 64 bits. 8 byte is chosen for the 64B/67B encoding selected for the protocol, and the size of the control word is also 67 bits used to bursts. The transfer unit is equivalent to the control word size it becomes easy to adjust the width of the interface. Control word

and data words are transmitted to the lanes sequentially, starting with lane 0, and ending at lane M, and repeating the lane for the next block of data. Transmitting data burst in all lanes is like a round robin manner, means after data transmit in lane M, repeat from lane 0. but before transmiting data burst, Synch word, scrambler word, diagnostic word is transmitted in all lanes. 64B/67B encoding occurs on each lane individually. 2 fundamental word for transport burst: Data Words and Idle/Burst Control Words.

4.3.2 Burst Structure

The Interlaken interface bandwidth is divided into data bursts with respect to the number of channels. Data packets are transferred to the interface of one or more bursts, with the bursts by means of one or more Control Words and control word is 67 bits word.

Typically the interface is operates by sending a data burst of maximum BurstMax length, with Control Word. The interface is a end-of-packet(eop) may occur a very small amount of remaining data on each channel and for that small amount of data, transmit full packet. this end of packet frame, after finish small amount of data, it pad ideal bits. Transmitter and receiver may be designed ideally with a large datapath. The BurstShort is parameter which is a minimum separation between any two successive Burst Control Words. The minimum BurstShort width is 32 bytes, also larger values possible in multiple of 8 bytes, values like 32, 40,48,56,64...

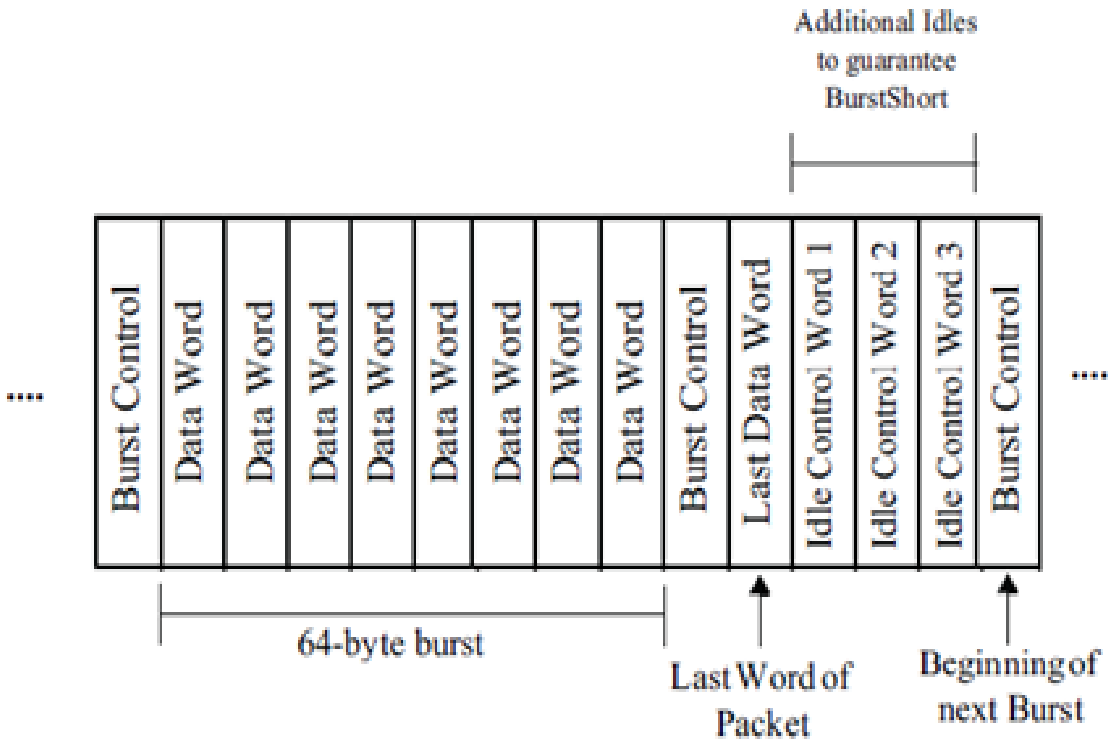

Figure 4.3: BurstShort Illustration

Figure illustrates the minimum interval required by BurstShort. If not enough bit is there then for BurstShort, adding extra Idle Words before the next Control Word is come, shown in fig 4.3.

4.3.3 Optional Scheduling Enhancement

Simple scheduling described was used for transmit packet, but using this Loss some unused bandwidth at the end of the packet for certain combination of packet length and BurstMax. When the BurstMax size is small, then small amount of data will remain at the last packet. So add some few idle bits for completing BurstShort size. In the worst case, maximum unused bandwidth is (BurstShort - 1) bytes per packet. One more efficient scheduling is possible: it's an optional scheduling enhancement.

data_packet_remainder = pkt_length

for (x=1; x <= i; x++) {
if (data_packet_remainder >= BurstMax + BurstMin) then
data_transfer = BurstMax
else
if (pkt_length MOD BurstMax < BurstMin) && (data_packet_remainder > Burst-Max) then
data_transfer = BurstMax - BurstMin
else
data_transfer = data_packet_remainder
data_packet_remainder = data_packet_remainder - data_transfer
}

This optional algorithm is implementation towards efficient mechanism of transporting data burst. There is no additional burden placed on the receiving logic. In the optional algorithm logic, there is no any requirement to add extra idle bytes. So, it's more efficient for data burst transfer.

4.3.4 Control Word Format

Bursts are divided in different 8-byte Control Word. The Control Word is identified in the data by using the '10' control code for bits[65:64], bits[65:64] are '01' for data word and other combinations are invalid. Bit[63] = '1' for the Burst and Idle Control Word.

Burst Control Words means type = '1' identify the beginning of a data burst. Every data burst transfer start with a Burst Control Word. The start-of-packet(sop) and Channel Number bits indicate beginning of data burst. When the Burst Control Word is required data bursts, the End-of-packet bits and CRC24 bits apply to the data.

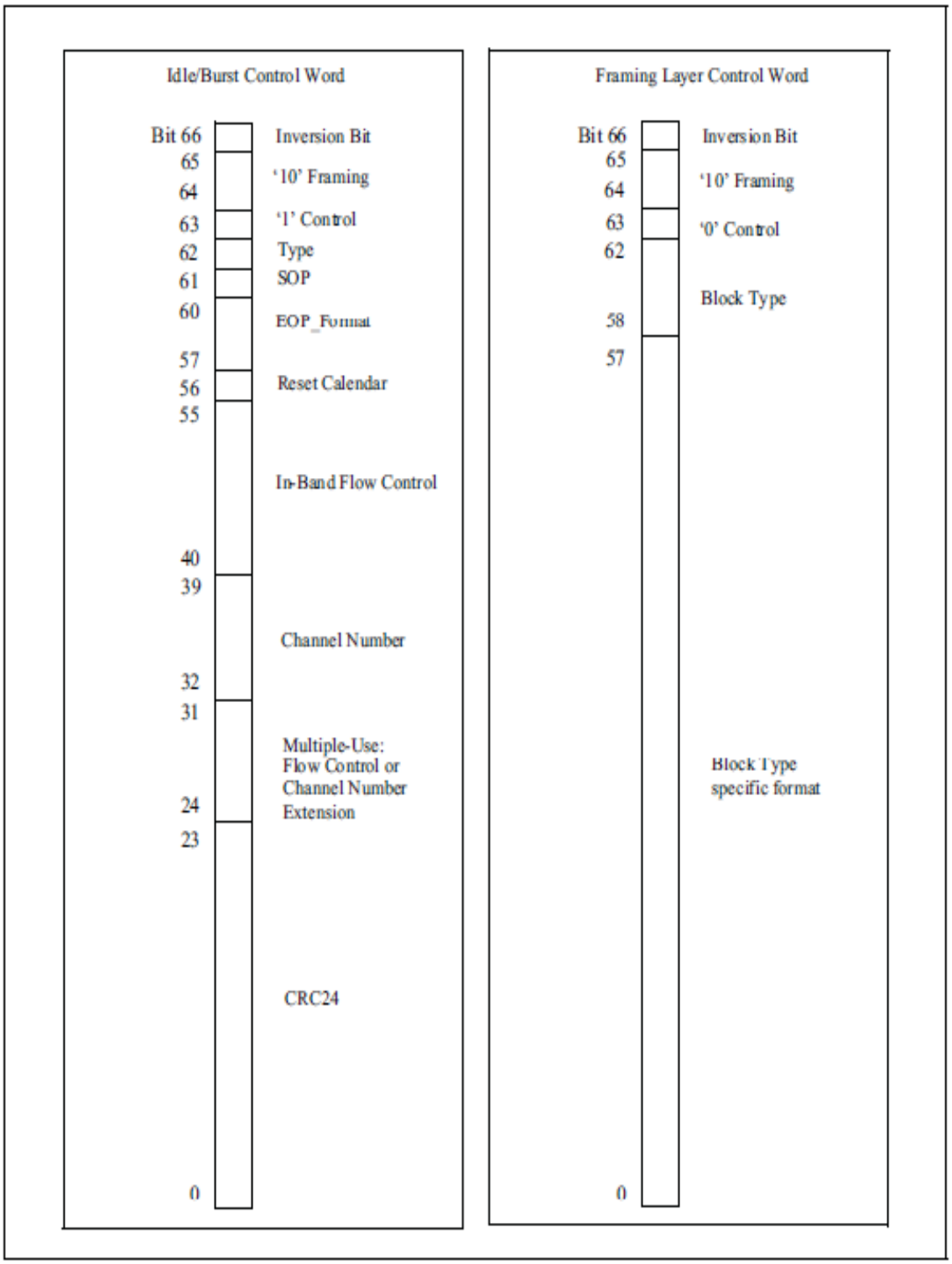

Figure 4.4: Control Word Format

Idle Control Words means Type ='0' transmit only idle words. It's always transmitted when there is no any data burst available. When data finished, idle data word will be passed. Because sent any burst to the receiver side until packet is sent. so the flow control information must be sent to the receiving device, the flow control fields are valid in both types of Control Words.

Field Name	**Position of Bits**	**Functions**
Inversion Bit	66	Bits [63:0] inversion
Framing	65:64	Check word is control word or data word
Control	63	'1' for idle-control word, '0' for framing word.
Type	62	Set to '1', channel no and sop field are valid.
Sop	61	Start of packet
Eop_Format	60:57	If '1xxx' end of packet
Reset calender	56	'1' indicates the in-band flow control
In band flow control	55:40	'1' for XON,'0'for XOFF.
bits for Channel number	39:32	Channel defined for the data burst.
Multiple use bits	31:24	This bits may used for multiple purposes
CRC24	23:0	CRC24 used for error check for data word

Table 4.1: Idle/Burst Control Word Format

Sop bit is '1', means start of packet, before data is started, and sop bit value is '1' of previous control word. After this, all times sop bit value is '0'. Eop_format have 4 bits for indication of end of packet. Bit positions [60:57] are eop_format, 1st bit indicate that packet is end or not, if 1st bit '1', then there is a last packet. Other 3 bits for how many valid bytes are there in this packet. Bytes that are invalid are discarded by the monitor.

For a size of Channel, it reserves 8 bits, so we have used up to 256 channel size. Advantage of Interlaken protocol is we used more channel as compare to others. Also we have multi-use 8 bits for channel number, so we should be used 16 bits for declare channel length. Last 8 bits is reserves for CRC24 (cyclic redundancy code). Crc24 is check error that covers control word and previous data word.

4.3.5 Flow Control

The ability to communicate per-channel backpressure is a key feature of Interlaken protocol. for this function, two options are there- an out-of-band(OOB) flow control interface and an in-band(IB) flow control interface channel. Flow control uses a simple on-off mechanism to signal for transmit on a particular channel number.

By using single bit of status the on-off flow control status is communicated for each supported channel. If a status is '1' then identify the 'XON' state, it's indicate permission for the transmitting data on that channel. A status is '0' then identifies the 'XOFF' state, it's indicating that the transmitter stop to sending data on that channel.
In this protocol, once a channel is indicated 'XON' status, than the transmitter is send as much data as it chooses on that channel until the status will change from 'XON' to 'XOFF'.

- **Out-of-Band Flow Control**
  For out of band systems that require only simplex operation. OOB flow control has a 3 signals, FC_CLK, FC_DATA and FC_SYNC signal. it is specified as follow:
  - FC_CLK : The flow control data is synchronized by clock
  - FC_DATA : Used for FC status information as a single bit
  - FC_SYNC : A sync signal is used to for beginning of the flow control calendar

  Each of these signals are either LVDS or LVCMOS. The logical timing relationship of these signals is shown in below figure-4.5:

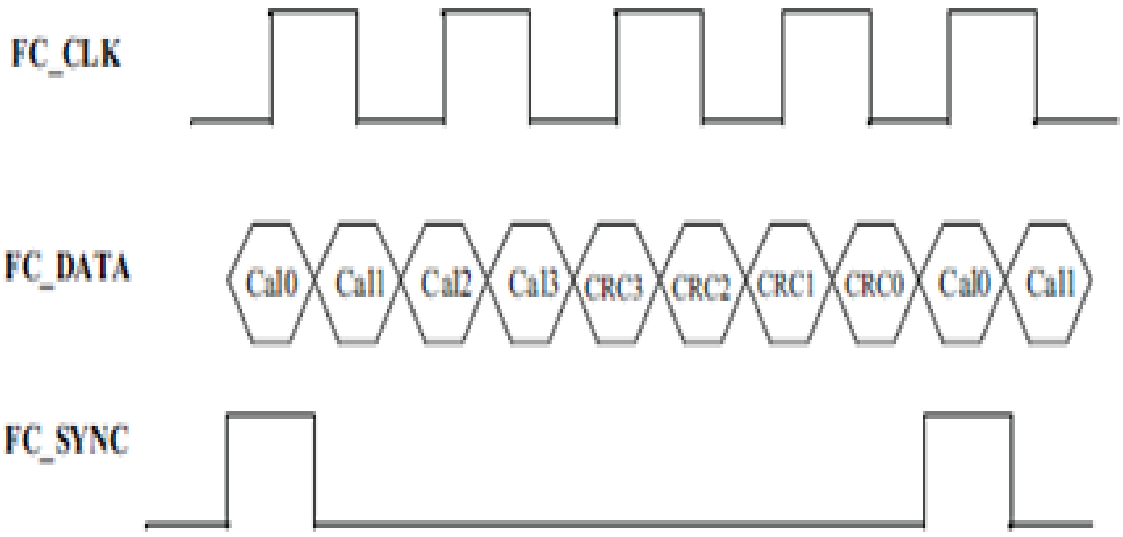

Figure 4.5: Out-of-Band Logical Timing Diagram

The OOB flow control ch is protected with a 4-bit CRC calculation that covers up to 64 bits of flow control data. Based upon the recommendations in [3], the CRC4 polynomial is:

$$x^4 + x + 1 \tag{4.1}$$

When the number of channels is 64 or less, the CRC4 checksum occurs the last calendar slot, and is followed by the flow control status of calendar slot 0.

- **In-Band Flow Control**

  When used IB flow control interface, the receiver makes use of in-band flow control status transmitted in the Control Words. Control words sent across the interface as part of the data transfer.

  The Flow Control field has a 16 bits of the Control Word, and it's located in bit positions bits[55:40] of control word format. Multiple use bits[31:24] is also be used for more 8 bits of Flow Control, so total 24 bits for flow control. These 16 status bits represent the ON-OFF flow control status for every calendar channel,

start calendar entry with X at bit [55], calendar entry X+1 at bit [54], calendar entry X+2 at bit [53], and so on.

4.4 Framing Layer

Interlaken has a multifunction framing method for achieving simple transport, which consists of the following components:

Function	Purposes
64B/67B encoding	control and data words with 8 byte word boundaries.
Synchronous scrambler	used for eliminate error multiplication
Lane alignment	All the lanes will align within a bundle.
Diagnostic	Diagnostics word checks error in metaframe per lane.
Skip	Clock Compensation for differential.

Table 4.2: Overview of Framing Layer

The framing layer uses Framing Layer Control Words. Bits [63:58] are identify which type of control word, where bit[63] set to zero and bits [62:58] indicate the Block Type.

4.4.1 64B/67B Encoding

An 64/67B encoding method is need for a serial interface to word boundaries, provide randomness to generated by the electrical transitions. 64 bit word is used for word lock, data burst will be passed after word is locked.
There is also two additional bits prepended in each 64-bit data or control word. For a data word, these sync bits are '01', and if these sync bits are '10' they identify a control word, and all other combinations like '00' and '11' are not allowed. The valid patterns searching in the received data stream, word boundary is locked after 64 correct matches.

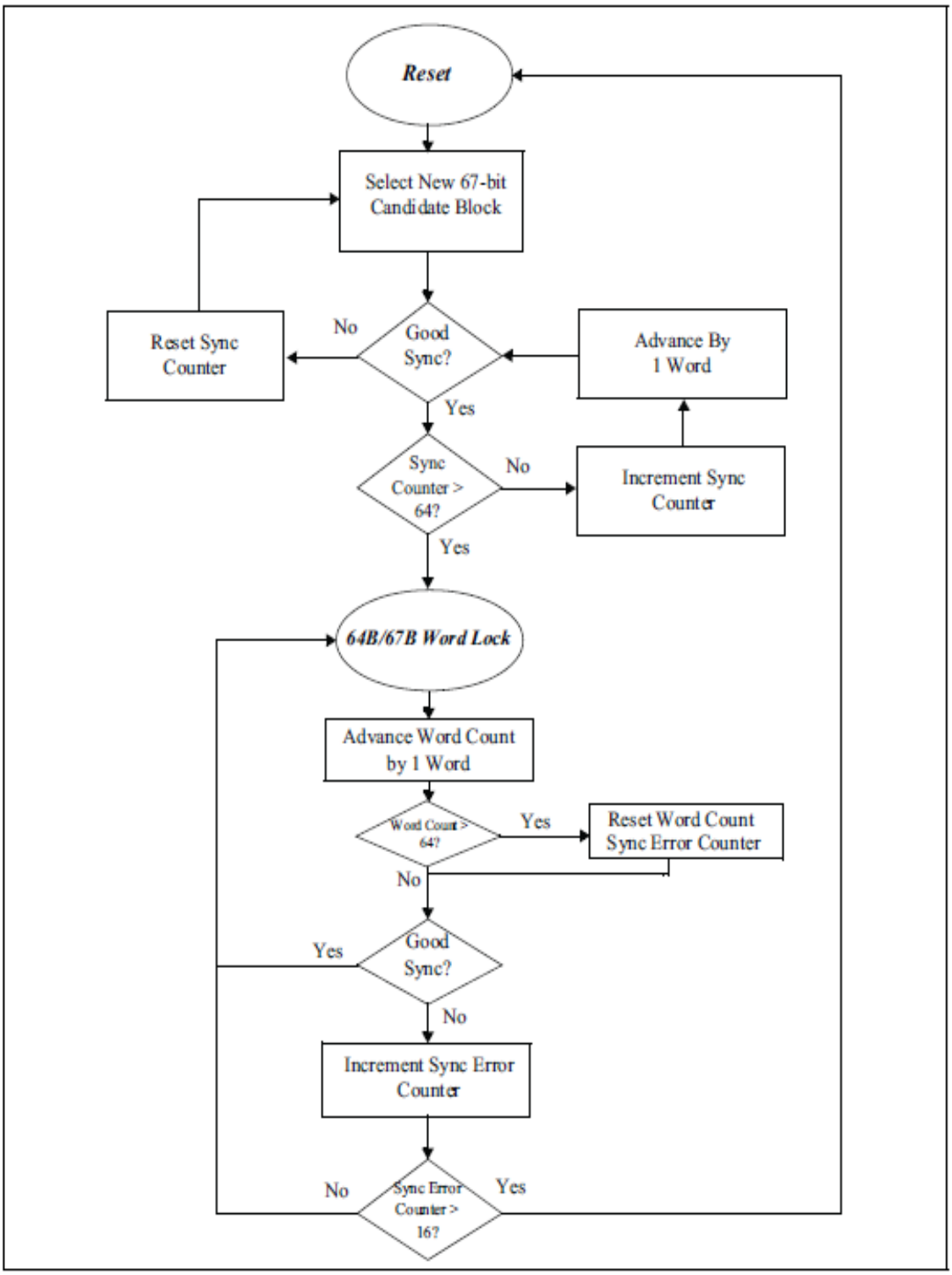

Figure 4.6: 64B/67B Word Boundary Lock

In the 64B/67B encoding, it's added 3 sync bit, so total 67 bit in each data or control word. but only 50% of the combinations are possible, it makes total 8 combination, but only 4 combinations is legal, the same as 64B/66B. Achieve word lock with low probability of an incorrect sync pattern, 64 consecutive legal sync patterns observed by the receiver, if it is achieved then word is locked.

4.4.2 Meta Frame

The framing method is also a concept of a MetaFrame. Structure of the MetaFrame is the per-lane set of the Synchronization word, Scrambler State word, Skip word, and Diagnostic words with the payload data (data burst and control word) carried on each lane. Synchronization, scrambler and skip word are transmit in starting of the metaframe, when diagnostic word is at last. skip word is optional, it is used if required for clock compensation.

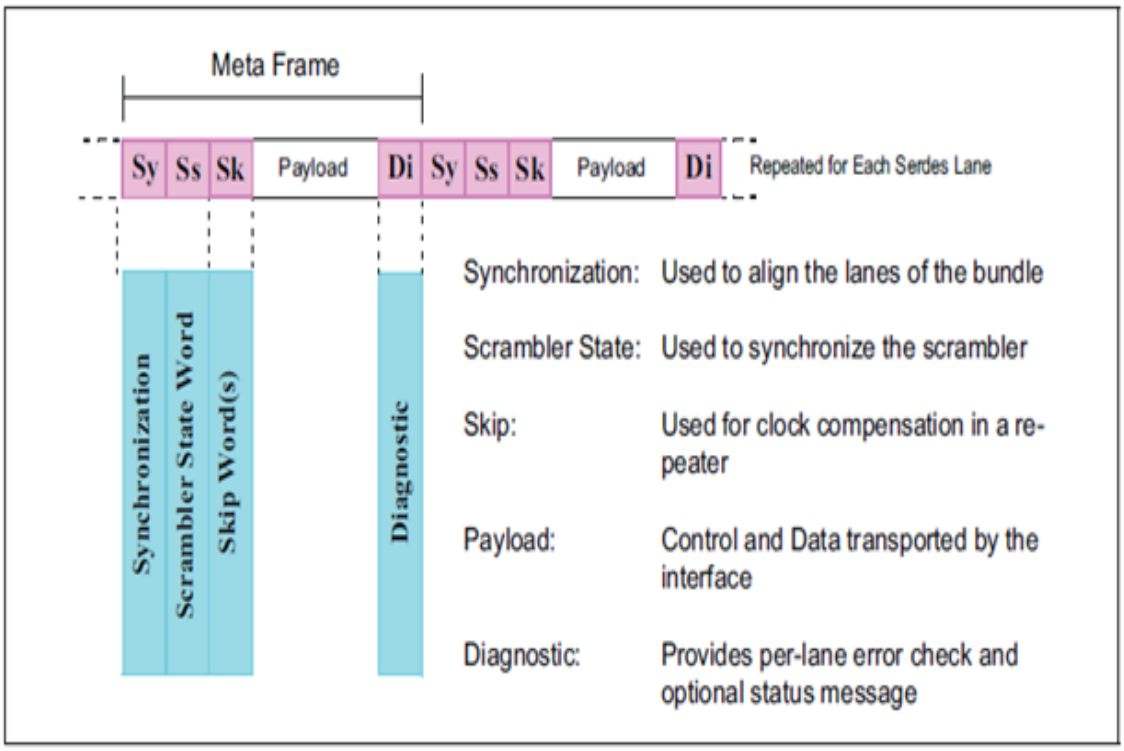

Figure 4.7: Meta Frame Structure

The size of the Meta Frame is not define, it's depend on the packet length, Meta Frame Length, that applies to all lanes of the bundle. MetaFrame length represents the sum of the control word, the data words and set of Synchronization word, Scrambler State word, Skip word, and Diagnostic word. It's structure is orthogonal for the transmissions of data burst. A Skip Word is defined to provide clock compensation. Diagnostic word is used for security purpose of MetaFrame.

4.4.3 Synchronous Scrambler

Self-synchronous scrambler is a part of the metaframe. synchronization word is the advantage of not requiring any other synchronization. The scrambler state word is the received data and can be recovered after length of the scrambler words are received. But scrambler word uses two feedback, and it has the functionality of repeat errors twice, means if single-bit error on the line than it's becomes three single-bit errors at the receiver side.

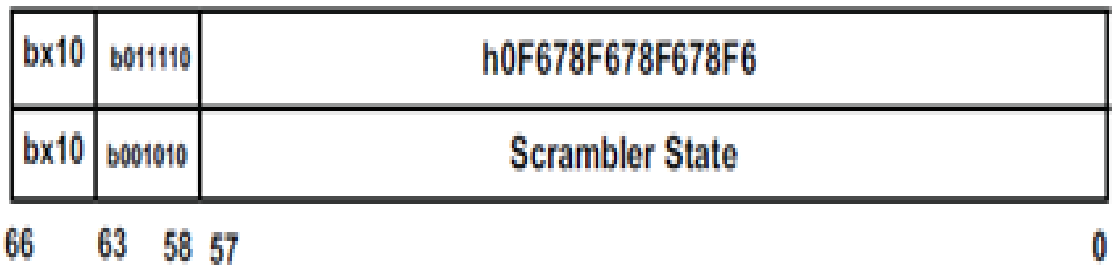

Figure 4.8: Synchronization and Scrambler State Words

There is no pass same scrambler state on each lane, not any requirement for same scrambler word and to minimize cross-talk between all lanes, the problem is that the scrambler will never be reset to all zeroes. The scrambler word is forwarded in the datapath, there is no need for the receive side of the interface to know to what value the transmit scrambler was reset. In the scrambler word, first 6 bit from bit[63:58] is reserve for identify the scrambler word and other bits are random, it's also all 0's.

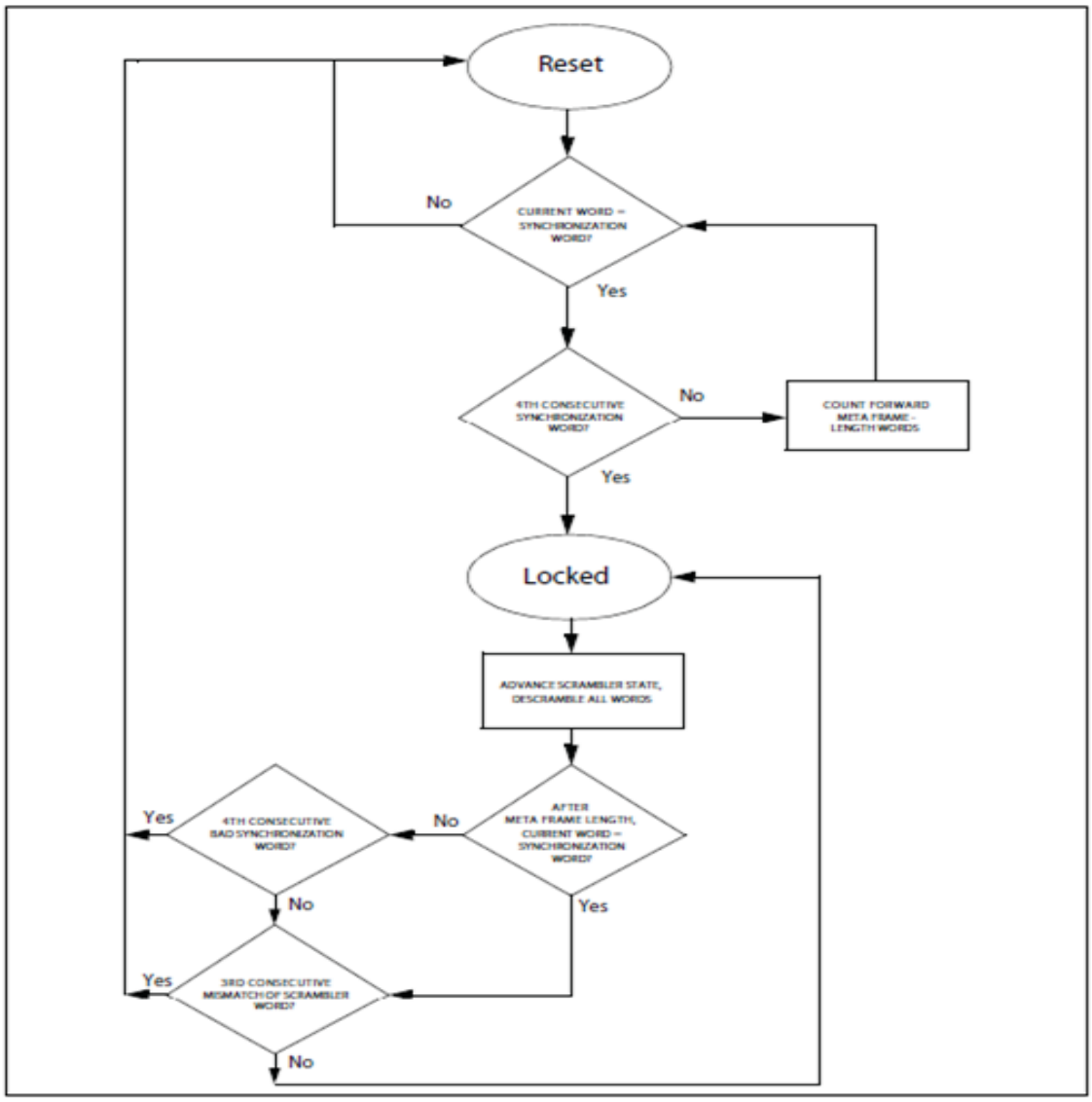

Figure 4.9: Scrambler Synchronization State Diagram

Synchronization word has a fixed pattern. bit[63:58] is for sync word identifier and other 58 bit has a fixed pattern. synchronization is a first word of the Metaframe. all new Metaframe will start from sync word. Once synchronization word is achieved, the interface uses the recovered value of the Scrambler State Word to seed the descrambler. All data word and control words exception of Sync and Scrambler words, are scrambled from bits [63:0], only 64 bit is scrambled, framing bits [66:64] are never

scrambled. All lanes should verifies that the each Scrambler State Word received after synchronization word.

MetaFrameLength means The size of the Meta Frame. Interlaken provides for the remove or addition of a Skip Word to manage clock compensation. Also use the repeater for adjust the position of the Synchronization Word relative to how it was originally transmitted.

4.4.4 Lane Alignment

Lane alignment is required for interlaken. Once the word-lock is identified and the scrambler properly reset, the lane must be aligned. Synchronization words are sent across the interface at a fixed frequency to regularly align the datapath SerDes lanes. For achieving alignment, first the Synchronization Word is transmitted in all lanes. So at receiver side, the monitor identifies these words. For lane alignment and security of sync word, transmit skew word before sync word. At receiver side, remove skew word and identify metaframe from the sync word. Also measures the skew between them across the lanes of the bundle, and adjusts its internal skew compensation logic.

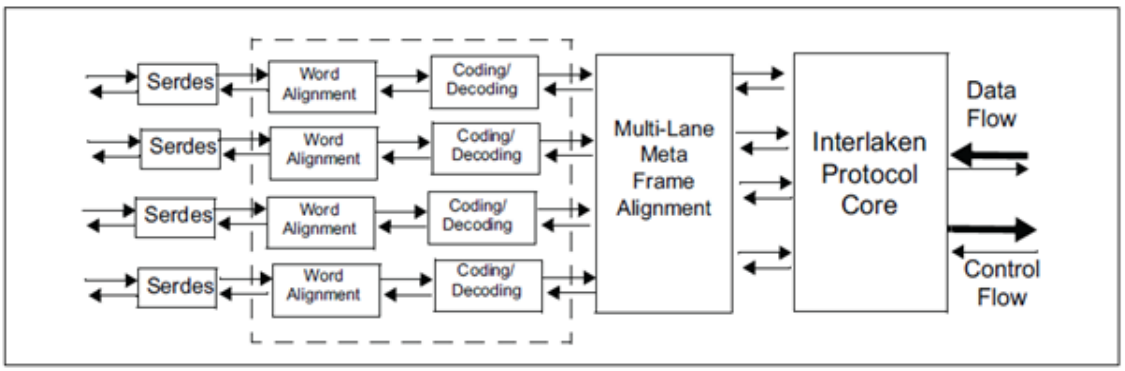

Figure 4.10: Interlaken Lane Alignment Segmentation

The transmission frequency of Synchronization Words is defined by the **MetaFrame-Length**.

4.4.5 Lane Diagnostics

The Diagnostic Word is 64 bit word, identified with the 6 bit[63:58], and it's value is **0b011001**.

The Diagnostic Word has assigned two functions - per-lane error detection and a lane Status Message. The bit position [33:32], 2-bit Status bit, which defines a place for a per-lane status message that is sent from receiver to transmitter. other bit[57:34] is fixed bit with value '0', and last 32 bit is for crc32 calculation.

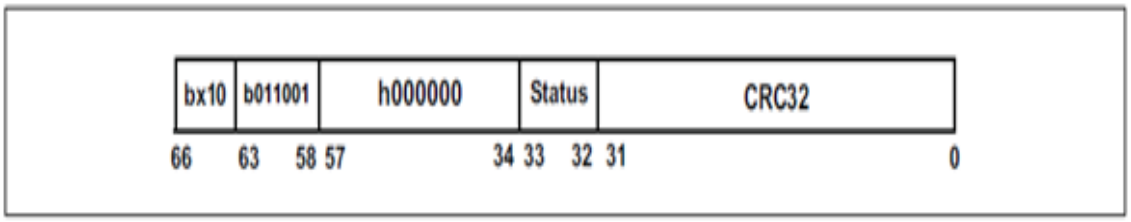

Figure 4.11: Diagnostic Word

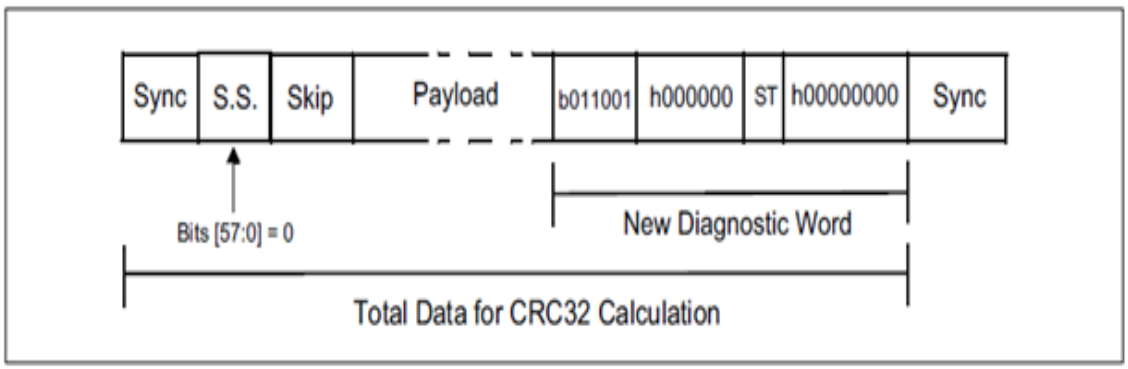

Figure 4.12: CRC32 Calculation Illustration

The CRC32 is provided calculation of Metaframe as a per-lane basis, so errors on the interface may be detected to an individual lane. It is calculated over all the

words transmitted within the Meta Frame, except 64B/67B bit., but including the Diagnostic Word itself with bits [63:0], with the CRC32 field padded to all 0's.

4.5 Applications

Interlaken is used in a lots of applications:

- **Framer/Ethernet MAC to NPU or L2/L3 switch interface**

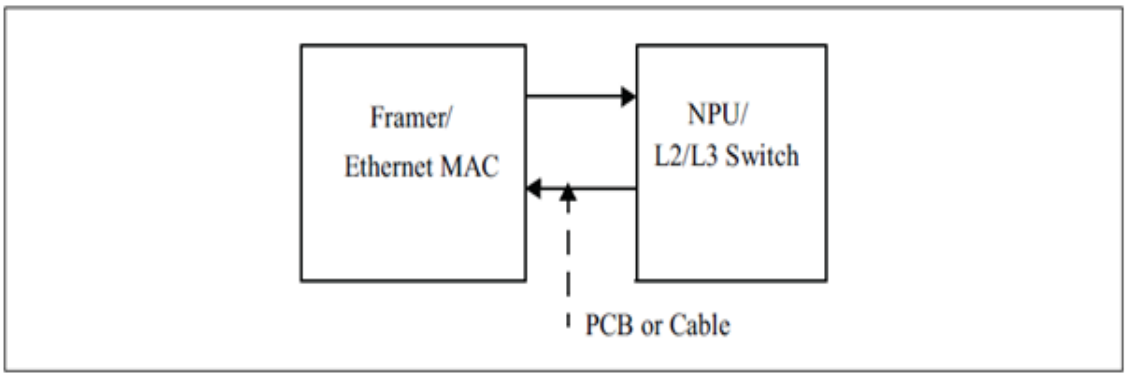

Figure 4.13: Framer/MAC to NPU/L2 or L3 Switch

- **Line Card to Switch Fabric Interface**

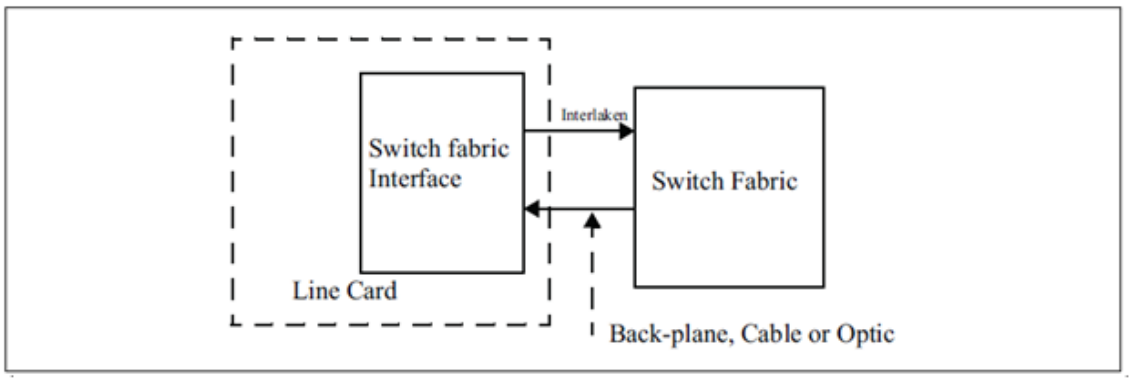

Figure 4.14: Line card to switch Fabric Interface

Interlaken can also run on multiple devices : cable, FR4 (PCB) or backplanes. Interlaken is a narrow, high-speed chip-to-chip interface protocol.

4.6 Summary

In this chapter we studied about Interlaken protocol and its alternatives, how Interlaken is more better than **xaui** and **spi4.2**. Advantage of protocol, protocol layer, framing layer, and structure of protocol layer, framing layer, metaframe.

Studied about metaframe lane size, different word in metaframe, synchronization word, scrambler word, diagnostic word structure. 64/67 bit encoding data . Also studied application of Interlaken protocol.

Chapter 5

Verified Logic Flow

5.1 Introduction

In verification, Interlaken protocol is design by design engineers, we can verify features of Interlaken protocol. In this protocol, I have verified word lock logic, skew added logic, disparity logic, CRC24 logic, lane alignment logic, multiple use bits logic, eop error logic, bad synch word logic, bad scrambler word logic, out of band flow control logic(crc4 error).

5.2 Skew added & Lane alignment Logic

Skew is added at starting of metaframe, 1st word of MetaFrame is skew, but skew size is not define, its random bit generated. In our logic, we keep skew size more than 80 bits, but it's not same size in all lanes. So 1st synchronization word is not start in same time in all lanes.

1st word after skew word will not start at same time in all lanes, so we checked skew size and arranged as a same size in all lanes for starting 1st word of lanes in same time. So at a driver size, before transmit data on lane, adjust skew size. In receiver size, skew word will be removed before descrambled data. Use of skew word is not loosing 1st word of MetaFrame.

In synchronous word there are random bits before it called as skew word. It will create problem for monitor to identify from where actually synchronous word started. To remove this problem lane alignment is used which applied same synchronous bits i.e. skew word before synchronous word.

5.3 Word lock logic

After passing 64 times control word or data words, word is locked. For this synchronization counter is used. For this continues '01' or '10' pattern is checked if fail then counter becomes reset. If word becomes locked i.e. required pattern is matched then synchronous word passed. In this logic check, 50% probability for correct pattern, check 2 bits, so it has 4 possibilities, but '00' and '11' are undefined. '01' is for data word and '10' is for control word.
64B/66B code defines a procedure for locking to the sync bits. The receiver searches for a transition from high to low or low to high (the only legal sync codes). In the next framing bit position, the receiver again looks for one of the legal patterns; if a legal pattern occurs again it repeats this procedure, and if it does not it resets its state and searches for another legal pattern from the starting. In order to declare lock the receiver must observe 64 consecutive legal sync patterns.

5.4 Disparity logic

In disparity logic, it checks that in continues metaframe if first bit of word is same as previous word's first bit then second word is inverted. If first bit of word in metaframe is not same as previous word's first bit then it will pass as it is. Also inverted logic is implementing in this. If inverted bit [66] is same, only [63:0] bits inverted, not [65:54]. Next word is inverting because of saving power. If previous word have a more '1' as compare to '0', then it's 1st bit is '1', so now if next word's 1st bit is '1', it means in this word also more number of '1' as compare to '0', invert 1st bit and inverting

bits[63:0]. In the receiving side, check the 1st bit of all the words and rearrange words. In receiving size, 2 consecutive word's 1st bit is same then gives error.

5.5 Multiple use bit logic

This multiple use bit may serve multiple purposes, depending on the application. These 8 bits may be used as a Channel Number Extension, the 8 least significant bits of the Channel Number. If additional in-band flow control bits are required, these bits may be used to represent the flow control status for the 8 calendar entries following the 16 calendar entries represented in bits[55:40]. These bits may be reserved for application specific purposes beyond the scope of this specification.
There are 8 bits used for logical channel which create 256 combinations for logical channel. There is a 8 bits for multiple uses like channel number, By using multiple use bit logic, it will add 8 another bits which make 64k combinations for logical channel. In this logic, I added these 8 multiple use bits to the channel number bits, so we have 16 bits for representing channel numbers. So we should make 64k combination for channel numbers.

5.6 CRC24 logic

The CRC4 used in the out-of-band flow control, the CRC24 used in the Burst/Idle Control Word, and the CRC32 used on each lane. CRC32 calculated on full MetaFrame, its covered sync word, scrambler word, control word, data burst and diagnostic word. Last 32 bits of diagnostic word is reserved for CRC32 calculation.
A CRC24 error check that covers the previous data burst and this control word. Last 24 bits of control word is reserve for CRC24 check. Data and control integrity is secured by means of the 24-bit CRC. The CRC24 is calculated against all data in the

burst and all the fields in the Control Word. The CRC24 polynomial is selected from

$$x^{24} + x^{21} + x^{20} + x^{17} + x^{15} + x^{11} + x^9 + x^8 + x^6 + x^5 + x + 1 \tag{5.1}$$

in this logic, driver is already drive data with calculating CRC24 bits, I implement logic in receiver side for recheck CRC24 on received data from the interface, and compare CRC24 bits of driver's data and CRC24 bits of receiver's data. If it's matched, then received data burst is correct, otherwise some problem with transmission.

5.7 Error logic

Implement some logic for checking correct data burst; I implemented some logic for error scenarios, like EOP error, sync word error, scrambler word error, word lock error, CRC4 error.

- **EOP Error Logic**

  EOP means End Of Packet. Eop has 4 bits of control word for last packet information. Bit[60:57] are for EOP bits. 1st bit of EOP is for end of packet, if it's '1', then its last packet otherwise not. Other 3 bits is for how many bytes are valid in this packet.

  '1xxx' - End-of-Packet, with bits[59:57] defining the number of valid bytes in the last 8-byte word in the burst. Bits[59:57] are encoded such that '000' means 8 bytes valid, '001' means 1 byte valid, '010' means 2 bytes valid, etc., with '111' meaning 7 bytes valid.

  '0000' means no End-of-Packet, no ERR, '0001' means Error and End-of-Packet, and all other combinations are left undefined.

  I changed some bits value for some particular time for generating error, and then recover this error. So also implement checker for eop error. Logic is implemented for checking eop bits properly worked or not.

- **Sync & Scrambler Word Error Logic**

  1st word of the MetaFrame is sync word, and then scrambler word. Sync word and scrambler word has some particular pattern of 67 bits. So it transmits with same word format. As shown in EOP error logic, same logic has implemented in these words. I changed some bits of both word for generating error for some time and then recover this error after some particular time.

- **Word Lock Error Logic**

  In the word lock logic, word is locked after consecutive 64 patterns passed. So I transmit any one wrong pattern before 64 consecutive patterns passed, and error will be generated. After some time, recover error.

- **CRC4 Error Logic**

  CRC4 is for out-of-band flow control. The out-of-band flow control is protected with a 4 bit CRC calculation that covers up to 64 bits of flow control data. The CRC4 polynomial is:

  $$x^4 + x + 1 \tag{5.2}$$

  In the crc4 error, for some time change the crc4 bit for create error scenario, and after some time, recover error.

5.8 Functional Coverage

Functional verification has a large portion of the resources required to design and validate a complex system. To minimize wasted effort, coverage is used as a guide for directing verification code by identifying tested and untested portions of the design. Coverage is defined as the percentage of verification objectives that have been done. There are two types of coverage metrics: code coverage and functional coverage. Code coverage is automatically extracted from the design code. Functional coverage is user specified in order to tie the verification environment to the design functionality.
Functional coverage is a user-defined metric that measures how much of the design

specification, as enumerated by features in the test plan, has been exercised. It can be used to measure whether interesting scenarios, corner cases, specification invariants, or other applicable design conditions captured as features of the test plan have been observed, validated and tested.

For functional coverage, create covergroup for finding coverage, first create functional coverage class, and create 4 different cover groups for coverpoints. Also create cross between coverage points. A covergroup can contain one or more coverage points. A coverage point can be a variable or an expression. Each coverage point includes a set of bins associated with its sampled values.

If coverage will get more than 90% then its good verification, verified other corner cases separately, and verify 100% functional coverage. In this project, regression is still running and it is required more time for run, because it has more number of testcases and more coverpoints. So required more time for cover all scenarios and get more verified result.

In the figure, snapshot of coverage report of only one test. So it's gives only 37% coverage. Regression is still running for full coverage.

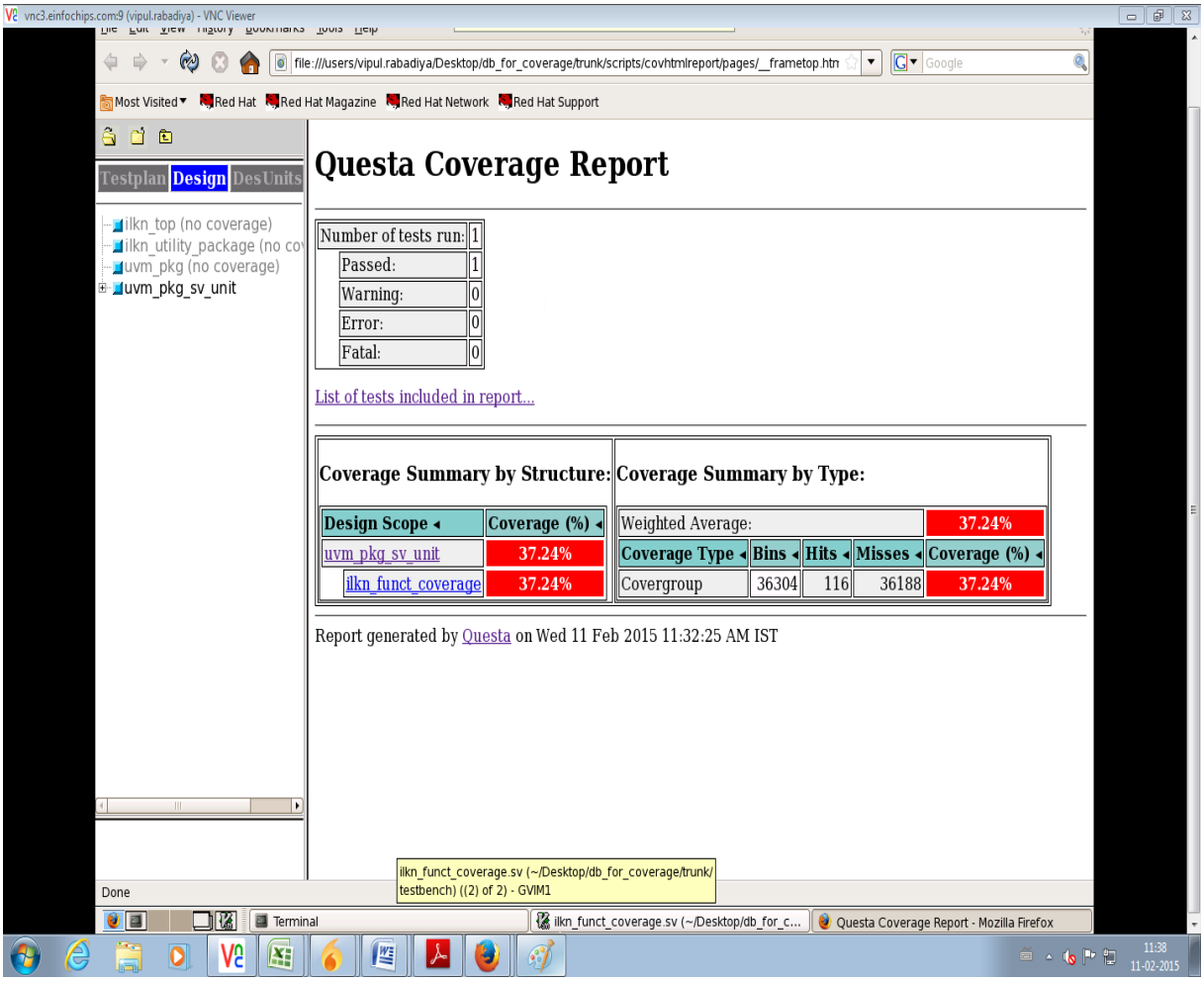

Figure 5.1: Coverage Report

Chapter 6

Conclusion

In this project we can transfer data in high speed using Interlaken protocol, by using this networking protocol; we can transmit data up to 100 Gbps data rate. Multi lane transmission is available. Interlaken protocol is verified by using universal verification methodology with system verilog programming.

When compared to available interconnect protocols, Interlaken offers many advantages in scalability, reduced pin count, and data integrity. Its channelization, flow control, and burst interleaving features make it appropriate for a wide variety of applications. Finally, the availability of a third party IP core minimizes the cost of adopting the new technology and makes Interlaken the obvious choice for next-generation communications equipment.

Interlaken protocol is verified by using system verilog logic and uvm methodology, uvm has a inbuilt class of all components, so verification make easy by using uvm methods. Simply in verification, when driver drive data to the interface, this data also caught by input monitor, and receiver side, received data also caught by output monitor, then both data compare in scoreboard. If matched, passed data is correct, otherwise failed.

I has done phase -1 programming in 3rd semester, in that, I has studied test plan and programming logic and then debug the word lock logic, disparity logic, crc24 logic. Solve the errors which were shouting in programming logic which done by design engineers. And also implement lane alignment logic, disparity logic.

After completing this, in 4th semester, complete the phase-2 and phase-3. In phase -2, implemented in-band flow control logic, skew injection, out-of-band logic and in phase-3, implemented error injection in sync word logic, crc24 logic, lane logic, word lock logic, disparity logic, synch word error, scrambler word error logic.

References

[1] $www.testbench.in$

[2] $www.asic-world.com$

[3] $www.accellera.org$

[4] $SystemVerilog3.1a: LanguageReferenceManual$

[5] $UniversalVerificationMethodology(UVM)1.1UsersGuide$

[6] $uvm\_users\_guide\_1.1$

[7] $www.verificationacademy.com$

[8] $interlakenalliance.com$

[9] $Interlaken\_Protocol\_Definition\_v1.2$