VIETNAM NATIONALUNIVERSITY – HO CHI MINH CITY

UNIVERSITY OF TECHNOLOGY

FACULTY OF ELECTRICAL AND ELECTRONICS ENGINEERING

DEPARTMENT OF ELECTRONICS ENGINEERING

------o0o-----

INTERNSHIP REPORT

Advisor:	MSc. Phan Võ Kim Anh
Student:	Nguyễn Duy Ngọc
Student ID:	2251036

Ho Chi Minh City, August 2025

ACKNOWLEDGMENT

First and foremost, I would like to express my sincere gratitude to the Department of Electronics Engineering, Faculty of Electrical and Electronics Engineering, Ho Chi Minh City University of Technology, for providing me with the opportunity to undertake this internship as part of my academic program.

I am deeply thankful to my academic advisor, **MSc. Phan Võ Kim Anh**, for her valuable guidance and support throughout the internship period.

I would like to extend my sincere appreciation to the leadership of Marvell Technology Vietnam LLC for providing the opportunity and environment that enabled me to participate in this internship program. Their support has been essential in facilitating my learning and professional development.

My heartfelt thanks go to **Mr. Dien Luong**, Director of the Design Verification – Connectivity at Marvell Technology, for granting me the opportunity to join the team as a Design Verification Intern. His leadership and support have been greatly appreciated.

I am especially grateful to **Ms. Quyen Luong**, my direct mentor, for her dedicated guidance, continuous encouragement, and for generously sharing her expertise throughout the internship. Her mentorship has been instrumental in helping me grow both technically and professionally.

Finally, I would like to thank my family and friends for their unwavering support and encouragement during this internship journey.

Ho Chi Minh City, August 11, 2025

Student

Nguyễn Duy Ngọc

TABLE OF CONTENTS

TABLE OF FIGURES

1. INTRODUCTION

1.1. Company Introduction

Marvell Technology, Inc. is a leading American semiconductor company that designs and develops data infrastructure technology. Founded in 1995 and headquartered in Santa Clara, California, Marvell provides a broad portfolio of silicon solutions that power the world's data centers, enterprise networks, cloud infrastructure, and carrier architectures [1].

Marvell's mission is to move, store, process, and secure the world's data faster and more reliably than ever before. The company focuses on high-performance, low-power semiconductor solutions that are essential for enabling the next generation of data-driven applications and services [1]. The company serves a wide range of markets, including cloud data centers, enterprise networking, automotive, and carrier infrastructure [2].

With a global workforce and engineering teams located in major technology hubs around the world, including Vietnam, Marvell fosters a collaborative and inclusive environment that supports innovation and continuous learning [3].

MARVELL™

Essential technology, done right™

Figure 1. Logo and Tagline of Marvell Technology, Inc.

1.2. Assigned Tasks

During my internship at Marvell Technology, I was assigned to the Design Verification, Connectivity team. My primary role was to study the verification of high-speed data communication systems, particularly those aligned with the Institute of Electrical and Electronics Engineers (IEEE) 802.3 standard for 1.6Tbps Ethernet.

The main tasks assigned to me included:

- Learning and applying Verilog and SystemVerilog for hardware verification.
- Studying the Universal Verification Methodology (UVM) framework.
- Using QuestaSIM for simulation and waveform analysis.
- Understanding key concepts in Physical Coding Sublayer (PCS) and Forward Error Correction (FEC) as defined in the IEEE 802.3 standard.
- Investigating the 1.6T data sequence generator and monitor in the internal design module (IDM) common library by executing data path tests on a production-level project.

- Running and analyzing test pattern generator/monitor (TP_GEN/TP_MON) testcases in the project.
- Reviewing test plans and verifying the correctness of test implementations.

1.3. Internship Timeline

The internship spanned 13 weeks, from June to August 2025, and was structured as follows:

- Week 1–2:
  - Learned the basics of SystemVerilog for verification.
  - Studied UVM fundamentals and its application in testbench development.
  - Gained hands-on experience using QuestaSIM for simulation and waveform debugging.
- Week 3–4:
  - Studied the PCS and FEC layers in the context of high-speed Ethernet.
  - Reviewed relevant sections of the IEEE 802.3 standard to understand their role in data integrity and transmission.
- Week 5–6:
  - Focused on clauses related to 1.6Tbps Ethernet in the IEEE 802.3 standard.
  - Analyzed the technical specifications and requirements for high-speed data communication.
- Week 7–9:
  - Investigated the 1.6T Data Sequence Generator and Monitor in the IDM common library.
  - Executed data path tests on a production-level project to understand how data is generated and verified.
  - Explored how the monitor validates received data against expected sequences.
- Week 10–12:
  - Executed TP_GEN/TP_MON testcases for the in the project.
  - Analyzed simulation results by checking sim.log files and dumping waveforms for debugging.
- Week 13:
  - Reviewed the TP_GEN/TP_MON testcases.
  - Cross-checked the implementation against the test plan to ensure completeness and correctness.

2. INTERNSHIP CONTENT

2.1. ASIC Design Flow and Design Verification

The Application Specific Integrated Circuit (ASIC) design process includes the key stages as shown in *Figure 2*.

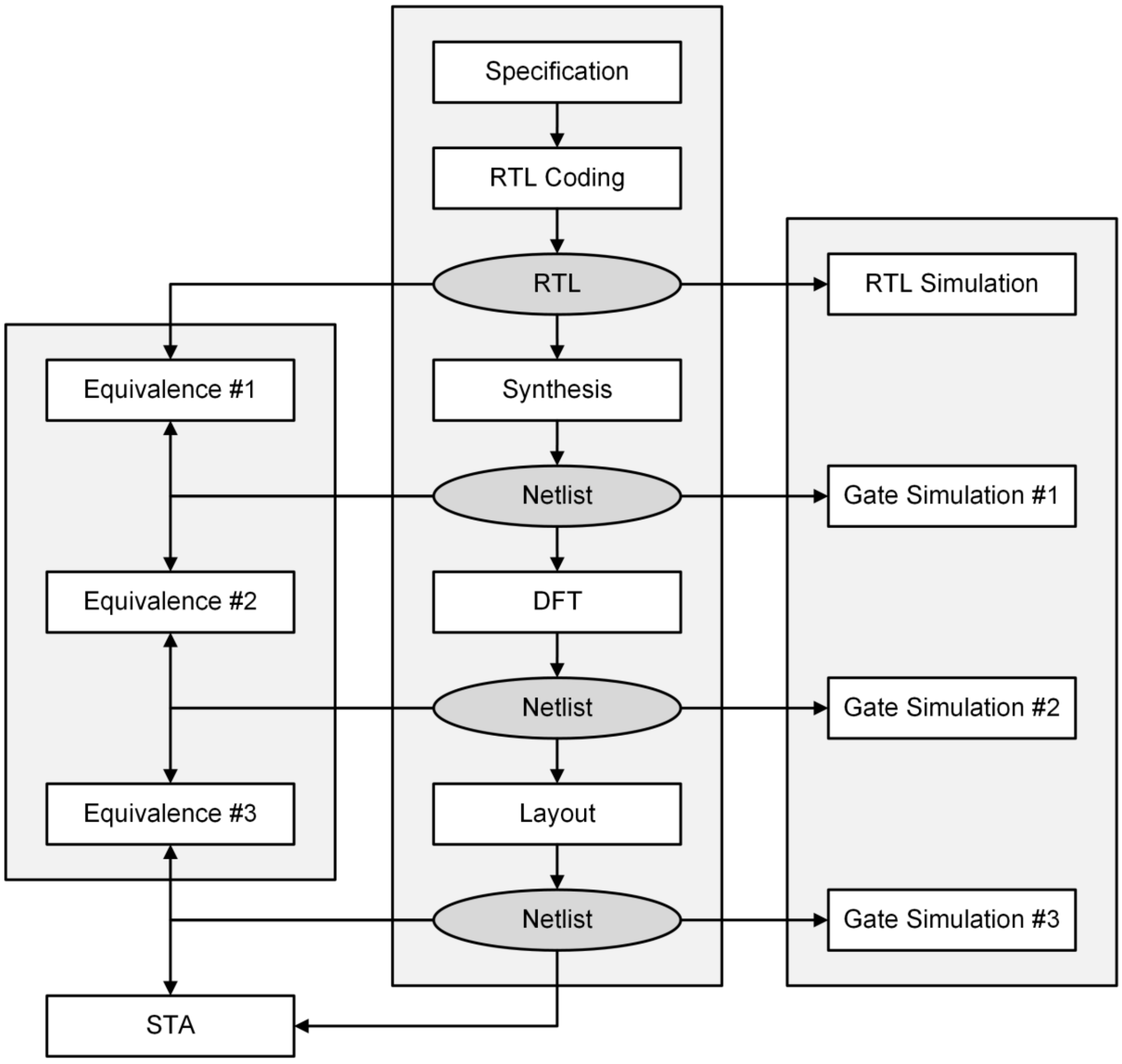


Figure 2. Block Diagram of ASIC Design Flow [4]

My internship focused on the design verification (DV) stage, particularly the Register Transfer Level (RTL) Simulation phase. Thus, I studied and applied SystemVerilog and UVM to analyze testbenches for verifying digital designs.

2.2. SystemVerilog and UVM Study

2.2.1. SystemVerilog Study

SystemVerilog is an extension of Verilog, designed to support both hardware design and verification. It introduces advanced features such as new data types, object-oriented programming, constrained randomization, assertions, and coverage, which are essential for building scalable and reusable testbenches.

2.2.1.1. Data Types and Casting

New data types are introduced:

- Two-state literals (only 0 or 1): bit, byte, shortint, int, longint
- Four-state literals (0, 1, X, Z): reg, logic, integer
- Non-integer types: time, shortreal, real, realtime
- Compound types: dynamic arrays, queues, associative arrays
- User-defined types: typedef, enum, struct, and union

For type casting, SystemVerilog supports either dynamic or static casting to convert between types with or without runtime checks.

2.2.1.2. Operators

Supported operators are:

- Arithmetic: +, -, *, /, %
- Logical: !, &&, ||
- Relational: >, <, >=, <=, ==, !=
- Bitwise: ~, &, |, ^, ~^
- Shift: <<, >>
- Concatenation: {} for combining bits or arrays
- Conditional: ? : for inline if-else logic
- Stream: {<<{}}, {>>{}} for bit packing and unpacking

2.2.1.3. Function, Task, and Multi-thread Execution

Functions and tasks are distinguished. Arguments can be passed by using ref.

- Functions: Do not consume simulation time; used for combinational logic.
- Tasks: May include delays; used for sequential operations.

Multi-thread execution is supported using constructs: fork...join, fork...join_any, and fork...join_none.

2.2.1.4. Interface

Interfaces group related signals and methods into a single construct in order to:

- Simplify connections between device under test (DUT) and testbench.
- Support encapsulation of protocol logic.

2.2.1.5. Object-Oriented Programming (OOP)

Object-Oriented Programming (OOP) features are supported to promote modularity and reuse:

- Class definitions with variables and methods
- Inheritance using extends

- Parameterized classes for generic design
- Handles for dynamic object management

2.2.1.6. Package

Packages are used to encapsulate and share definitions, which:

- Include classes, typedefs, enums, and other declarations.
- Can be imported using the import statement.

2.2.1.7. Randomization and Constraints

Constrained random stimulus generation is newly introduced:

- Variables declared with rand or randc are randomized.
- Constraints restrict the range and relationships of values.
- Advanced features include inside, dist, solve...before, and constraint_mode().

2.2.1.8. Assertions and Coverage

Assertions are used to validate design behavior during simulation. Immediate and concurrent assertions check protocol compliance and detect violations.

Functional coverage is implemented using covergroup:

- Tracks which scenarios have been exercised.
- Guides test completeness and verification quality.

SystemVerilog's rich feature set forms the foundation for UVM and modern verification practices. Its capabilities in modeling, abstraction, and automation are crucial for building effective and reusable verification environments [5].

2.2.2. Universal Verification Methodology (UVM) Study

2.2.2.1. UVM Fundamentals

Universal Verification Methodology (UVM) was standardized by Accellera to support reusable, scalable, and modular verification environments using SystemVerilog.

A typical UVM testbench includes the following components, as shown in *Figure 3*:

- uvm_env is a container for agents, scoreboards, and other components. It promotes modularity and reuse by grouping related functionality.
- uvm_agent typically includes a sequencer, driver, and monitor. It can operate in active mode (generating stimulus) or passive mode (monitoring only).
- uvm_driver receives transactions from the sequencer and drives them to the DUT using the interface.
- uvm_monitor observes DUT signals and converts them into transactions for analysis or checking.

- uvm_scoreboard compares actual DUT outputs with expected results, often generated by a reference model or golden logic.
- uvm_sequencer manages the flow of transactions from sequences to the driver. It supports arbitration and synchronization between multiple sequences.

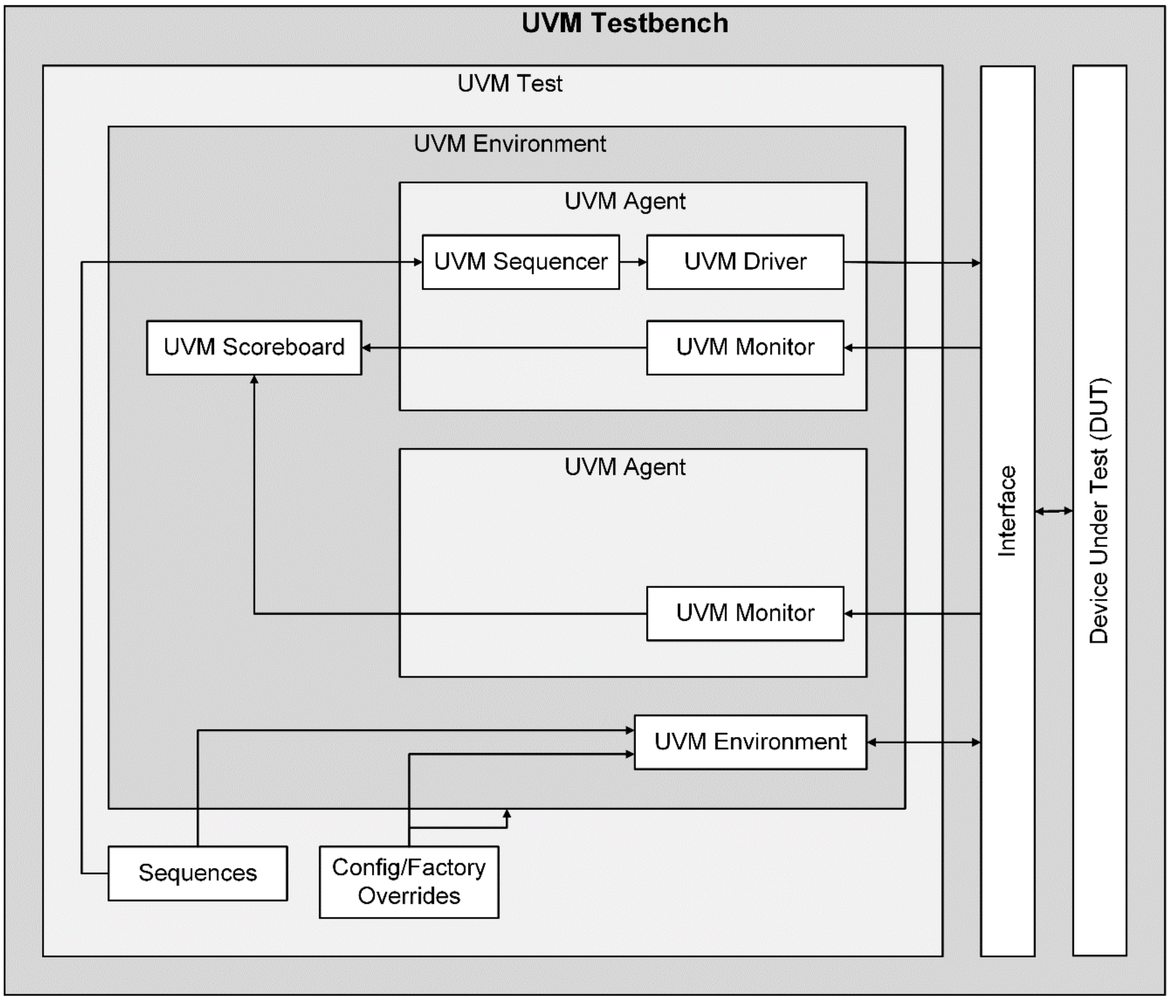


Figure 3. Typical UVM Testbench Architecture [6]

2.2.2.2. UVM Component Hierarchy

The UVM class hierarchy is organized into two main branches: the component branch and the sequence branch. All classes are ultimately derived from the base class uvm_object, as shown in image.

- uvm_object is the root class of the UVM class hierarchy, providing printing, copying, comparing, and recording utilities.

Component branch includes:

- uvm_report_object extends uvm_object and adds standardized reporting capabilities.
- uvm_component extends uvm_report_object and introduces simulation phase control.

Sequence branch includes:

- uvm_transaction extends uvm_object and serves as the base for all transaction-level data items. It includes timing and recording features for transaction analysis.
- uvm_sequence extends uvm_sequence_item and is used to generate and control sequences of transactions.

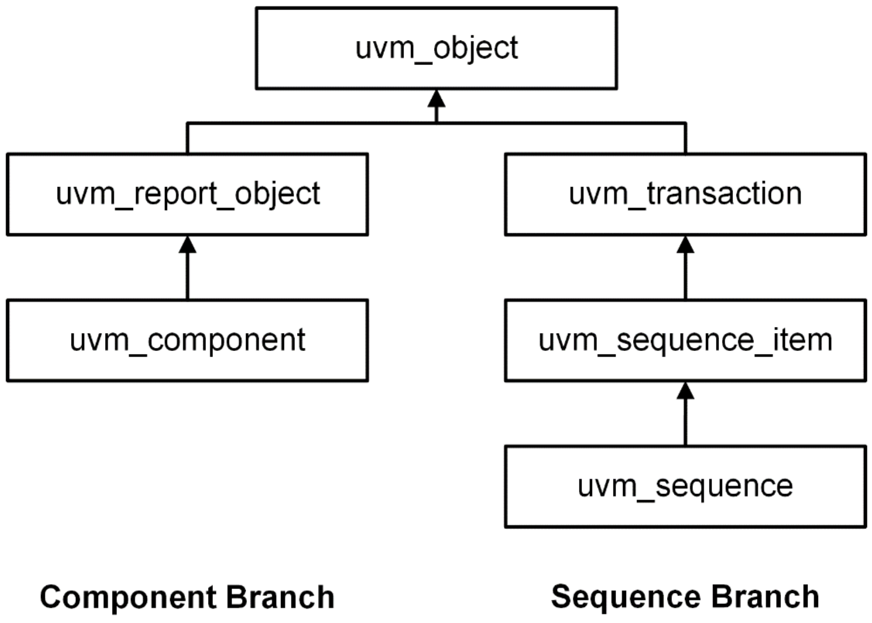


Figure 4. Simplified UVM Inheritance Diagram [7]

2.2.2.3. UVM Phases

The UVM simulation flow is divided into multiple phases. Each phase serves a specific purpose in constructing, executing, and finalizing the testbench. As shown in figure, the phases are grouped into three categories: build, run, and cleanup.

Build phases include:

- build: Testbench components are constructed using the UVM factory.
- connect: Transaction-level modeling (TLM) ports and exports are connected.
- end_of_elaboration: Final adjustments are made before simulation starts.

Run phases include:

- start_of_simulation: Testbench topology and configuration are displayed.
- run: Main simulation activity occurs. It is divided into structured sub-phases, each grouped into three parts: pre_phase, phase, and post_phase.
    - Reset: DUT and interfaces are reset to default states.
    - Configuration: DUT and memories are programmed for test readiness.
    - Main Test: Main stimulus is generated and applied to the DUT.
    - Shutdown: Ensures stimulus effects have propagated and DUT is idle.

Cleanup phases include:

- extract: Data is retrieved from scoreboards and coverage monitors for analysis.
- check: DUT behavior is verified against expected results.
- report: Simulation results are displayed or written to file.
- final: Outstanding actions are completed before simulation ends.

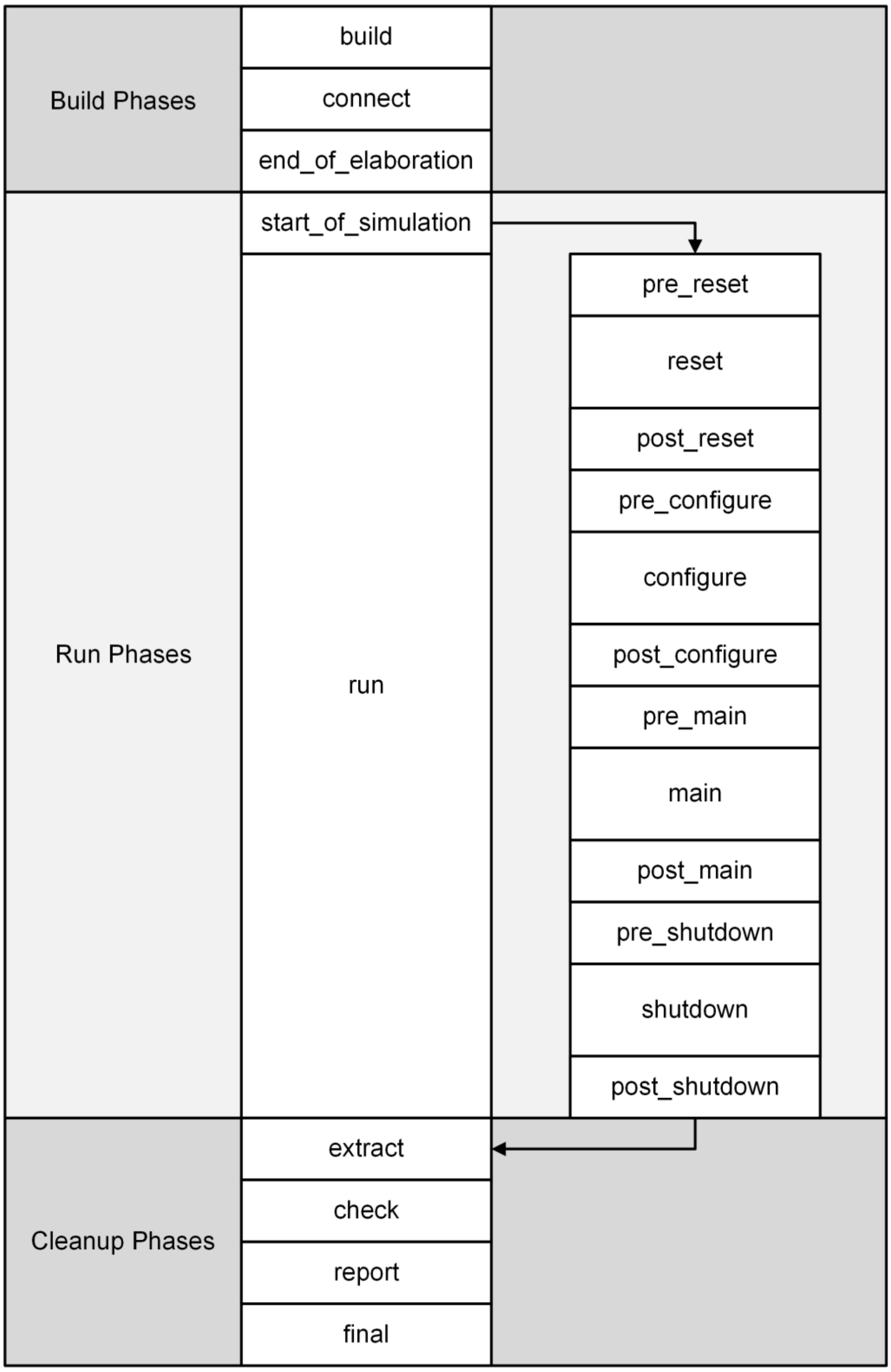


Figure 5. Standard UVM Phases [6]

2.2.2.4. Implementing UVM in the Verification Flow

Design verification (DV) is the process of ensuring that an RTL design behaves according to its specification before fabrication. Functional correctness is validated through simulation, using testbenches that apply stimulus, monitor responses, and check results. DV plays a critical role in identifying design bugs early, reducing the risk of costly rework.

UVM provides a standardized, reusable, and scalable framework to support DV. By using UVM, verification environments can be constructed in a modular and configurable way, enabling efficient test development and maintenance across projects.

The following steps describe how UVM is applied in a practical verification flow:

- Create a Top-Level Environment
- Instantiate Verification Components
- Create Test Classes
- Configure Verification Components
- Create and Select a User-Defined Test.
- Create Meaningful Test
- Create Virtual Sequences
- Check DUT Correctness
- Create and Configure Scoreboard
- Implement Coverage Model

2.3. IEEE 802.3 Ethernet Standards – Key Concepts Study

2.3.1. OSI Reference Model and PHY Layer

The OSI (Open Systems Interconnection) reference model is a framework that standardizes the functions of a communication system into seven distinct layers, as shown in *Figure 6*. These layers, from top to bottom, are: Application, Presentation, Session, Transport, Network, Data Link, and Physical.

In Ethernet systems, the Physical Layer (PHY) is responsible for the transmission and reception of raw bitstreams over a physical medium. To handle different aspects of physical transmission, the PHY is further divided into multiple sublayers, including:

- Physical Coding Sublayer (PCS)
- Physical Medium Attachment (PMA)
- Physical Medium Dependent (PMD)

This study focuses on the evolution of PCS and PMA functions across various Ethernet generations as defined in the IEEE 802.3 standard.

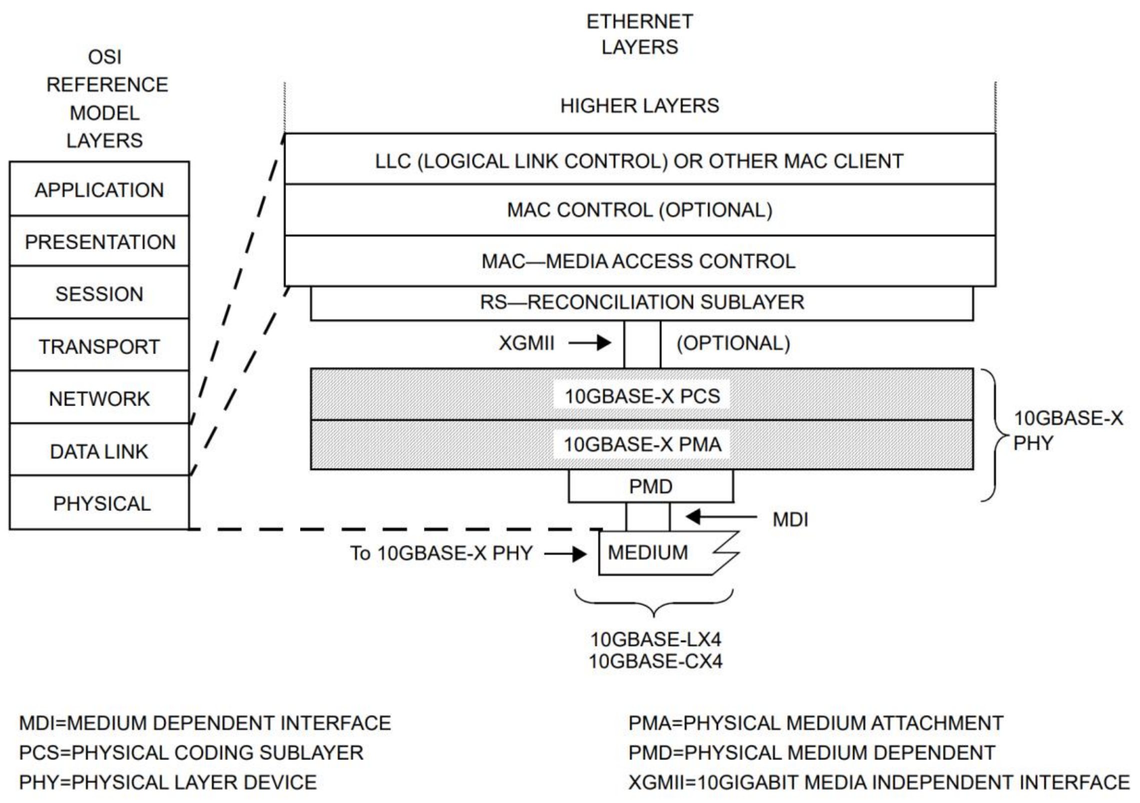

ETHERNET LAYERS
OSI REFERENCE MODEL LAYERS
APPLICATION
PRESENTATION
SESSION
TRANSPORT
NETWORK
DATA LINK
PHYSICAL
HIGHER LAYERS
LLC (LOGICAL LINK CONTROL) OR OTHER MAC CLIENT
MAC CONTROL (OPTIONAL)
MAC—MEDIA ACCESS CONTROL
RS—RECONCILIATION SUBLAYER
XGMII
(OPTIONAL)
10GBASE-X PCS
10GBASE-X PMA
10GBASE-X PHY
PMD
MDI
To 10GBASE-X PHY
MEDIUM
10GBASE-LX4
10GBASE-CX4
MDI=MEDIUM DEPENDENT INTERFACE
PCS=PHYSICAL CODING SUBLAYER
PHY=PHYSICAL LAYER DEVICE
PMA=PHYSICAL MEDIUM ATTACHMENT
PMD=PHYSICAL MEDIUM DEPENDENT
XGMII=10GIGABIT MEDIA INDEPENDENT INTERFACE

Figure 6. 10GE PCS Relationship to the OSI Model and IEEE 802.3 Model [8]

2.3.2. PCS Functions

PCS is positioned directly below the Reconciliation Sublayer (RS) and above the PMA, ensuring that data is formatted and prepared for reliable transmission over multiple lanes.

This section summarizes the key concepts and mechanisms implemented in PCS across various IEEE 802.3 clauses, including clause 49, 82, 91, 119, etc.

2.3.2.1. MAC Frame and MII

The Media Access Control (MAC) frame is the data unit generated by the MAC layer. Each frame ranges from 81 to 1,541 bytes in size, as shown in *Figure 7*, and follows the structure below:

- Preamble (7 bytes): Idle bytes used to establish timing.
- Start Frame Delimiter (SFD) (1 byte): Indicates the beginning of the frame.
- Destination Address (6 bytes): Specifies the MAC address of the receiving device.
- Source Address (6 bytes): Specifies the MAC address of the transmitting device.
- Length (2 bytes): Indicates the size of the data field.
- Data (46 – 1,500 bytes): Contains the Logical Link Control (LLC) header and payload.

- Frame Check Sequence (FCS) (4 bytes): Provides a Cyclic Redundancy Check (CRC) remainder for error detection.
- Inter-Packet Gap (IPG) (9 – 15 bytes): Idle bytes inserted to separate consecutive frames.

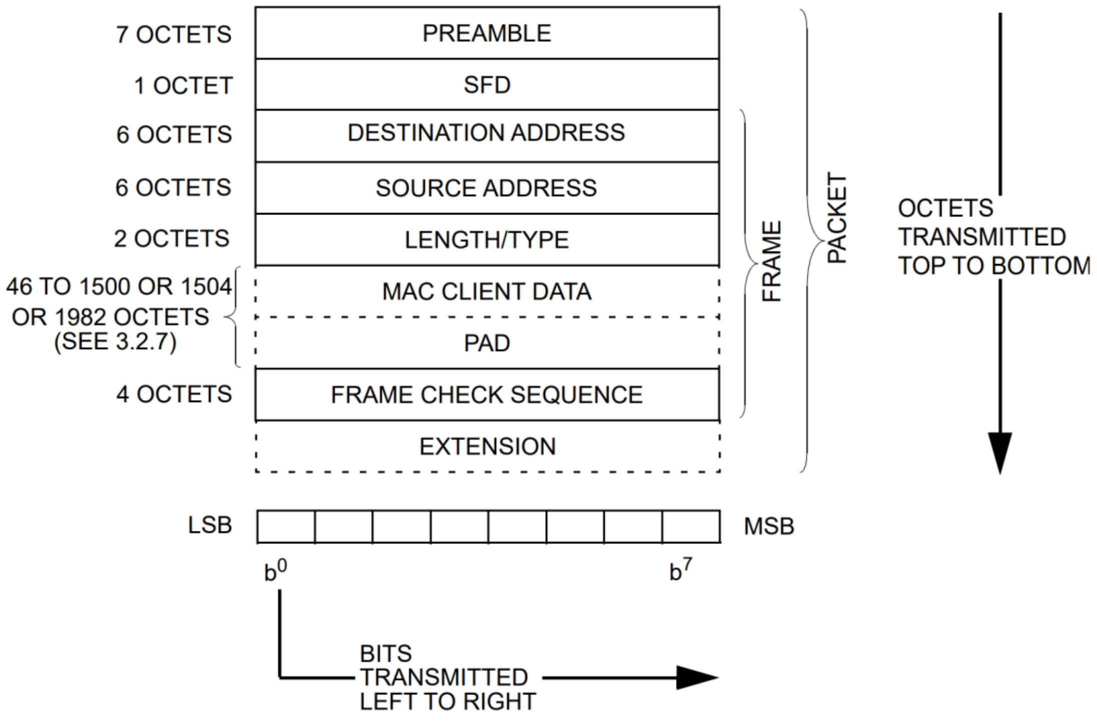


Figure 7. MAC Frame Format [8]

The Media Independent Interface (MII) is used to transfer MAC frames to the PCS for encoding and transmission. Every two consecutive MII transfers are combined by the PCS to form a 64-bit data block, as illustrated in *Figure 8*.

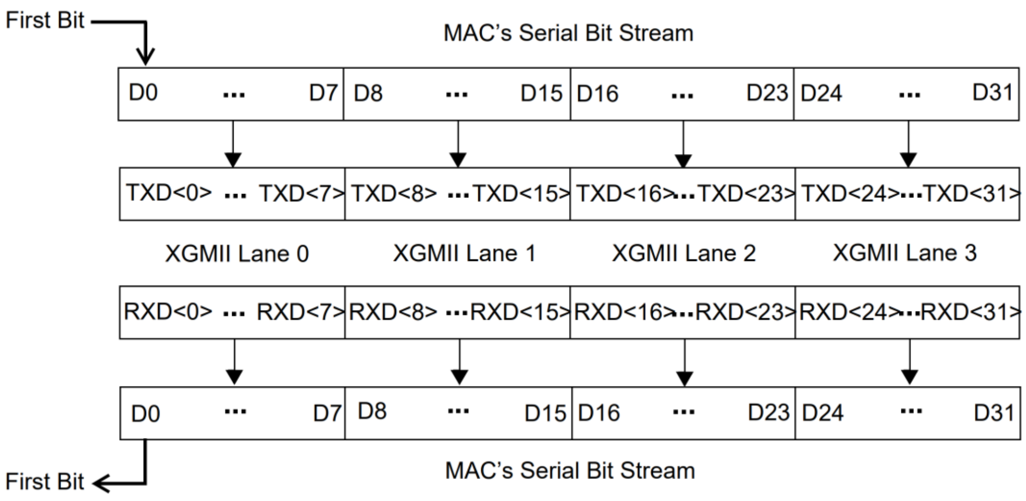


Figure 8. Relationship of MII Data Lanes to MAC Serial Bit Stream [8]

2.3.2.2. 64b/66b Encoding

A 64b/66b encoding scheme is applied to convert each 64-bit data block into a 66-bit block by prefixing a 2-bit synchronization (sync) header, as illustrated in *Figure 9*. This header is used to indicate whether the corresponding block contains control characters.

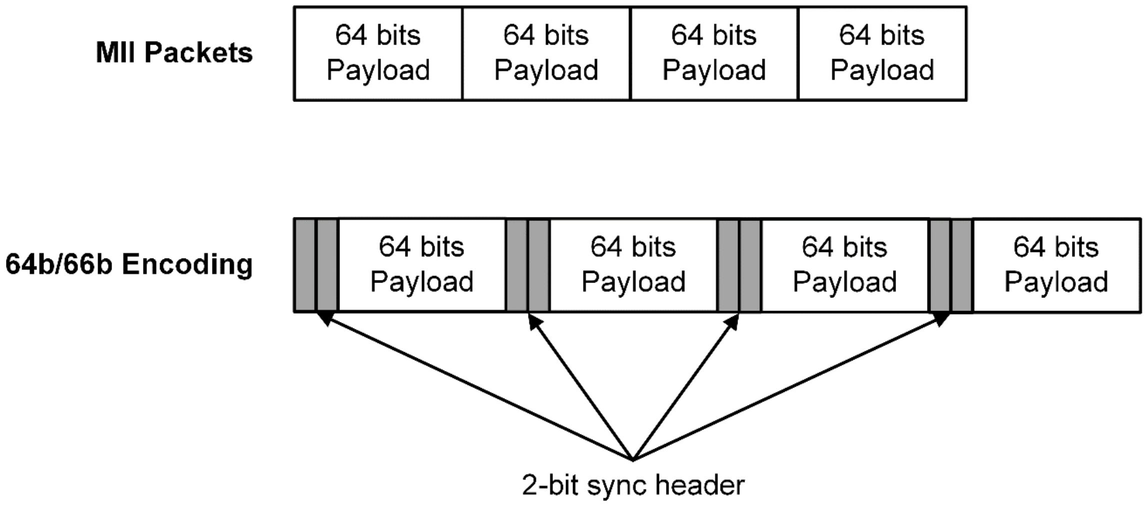


Figure 9. 64b/66b Encoding Illustration [9]

2.3.2.3. 256b/257b Transcoding

A 256b/257b transcoding scheme is to reduce overhead while preserving alignment.

In this method, each 256-bit data block is mapped into a 257-bit code block by appending a 1-bit sync header, which is formed from the original four 2-bit sync headers, to every group of four 64-bit blocks, as illustrated in *Figure 10*.

This approach results in a savings of 7 bits per group of four 64-bit blocks, or 140 bits across 20 such groups – resources that are later utilized for Reed-Solomon Forward Error Correction (RS-FEC) encoding.

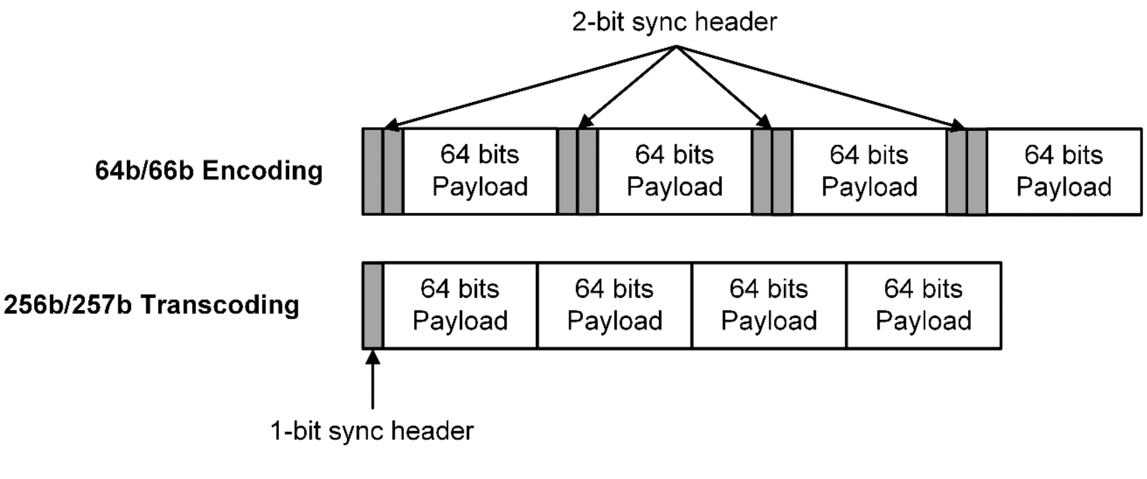


Figure 10. Examples of 256b/257b Transcoding [9]

2.3.2.4. Scrambling

Scrambling is applied to randomize bit patterns, as illustrated in *Figure 11*. This technique maintains Direct Current (DC) balance, minimizes Electromagnetic Interference (EMI), and prevents long sequences of identical bits.

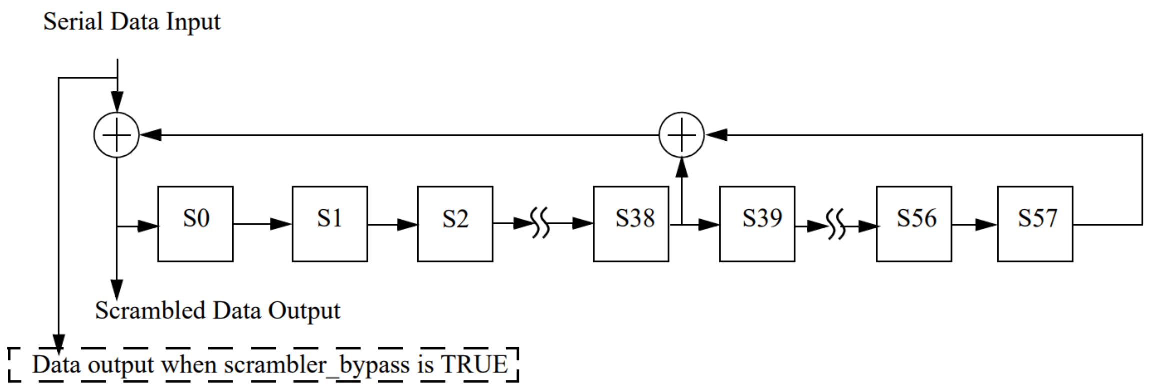


Figure 11. Scrambler Model [8]

2.3.2.5. RS-FEC Encoding

RS-FEC is implemented as a sublayer within the PCS. It performs Reed-Solomon encoding over the Galois Field GF (2^{10}), where each symbol consists of 10 bits, as illustrated in *Figure 12*. A Reed-Solomon encoder processes k message symbols to generate 2t parity symbols. The parity symbols are appended to the message to form a codeword of n = k + 2t symbols.

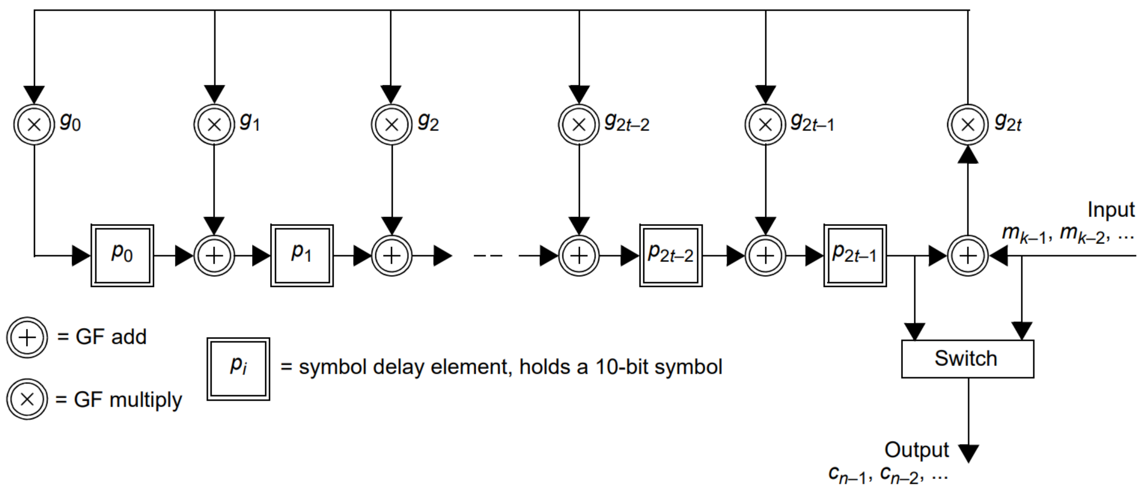


Figure 12. Reed-Solomon Encoder Model [8]

Two RS-FEC configurations are defined:

- RS (528, 514):
    - Generates 14 parity symbols (140 parity bits)
    - Allows correction of up to 7 error symbols per codeword

- RS (544, 514):
    - Generates 30 parity symbols (300 parity bits)
    - Allows correction of up to 15 error symbols per codeword

Each message of twenty 257-bit blocks is appended with either 140 or 300 parity bits, forming a 5280-bit or 5440-bit RS-FEC codeword, respectively, as illustrated in *Figure 13*.

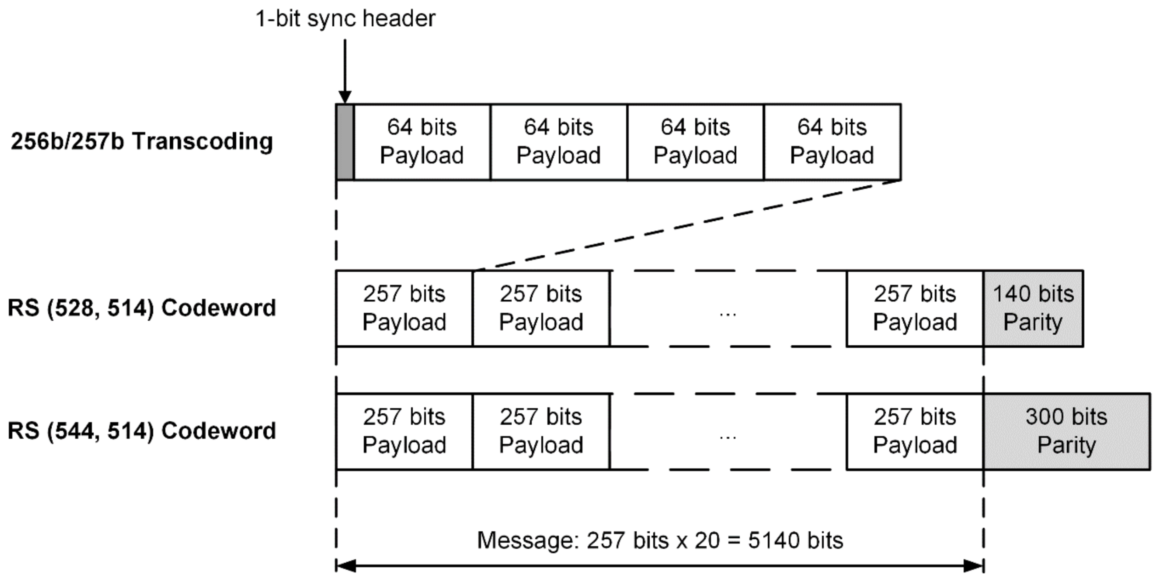


Figure 13. Illustration of RS-FEC Codeword Formation

In RS (528, 514), the 140-bit parity fits within the 140-bit overhead available from twenty 256b/257b blocks. However, in RS (544, 514), the 300-bit parity exceeds the available overhead, requiring a bit rate increase by 3.033% per lane.

2.3.2.6. PCS Lanes – Block and Symbol Distribution

PCS lanes (PCSLs) were initially introduced as virtual lanes in early Ethernet standards (e.g., Clause 82), where they served to distribute data across multiple distribution paths. In later standards (e.g., Clause 91), they were referred to as FEC lanes (FECLs), representing distribution paths that carry portions of the encoded data. Despite the variation in terminology, the currently adopted term is PCS lanes.

Block distribution assigns either 64b/66b, which is used in PCS without RS-FEC, or 256b/257b encoded data blocks to PCSLs in a round-robin manner, as shown in *Figure 15*.

Symbol distribution, which is used in PCS with RS-FEC, performs a similar operation but operates at the symbol level, typically distributing 10-bit symbols across lanes, as shown in *Figure 14*.

Symbol interleave distributes symbols from codewords from different FEC encoders across PCSLs to improve error resilience, as shown in *Figure 14*.

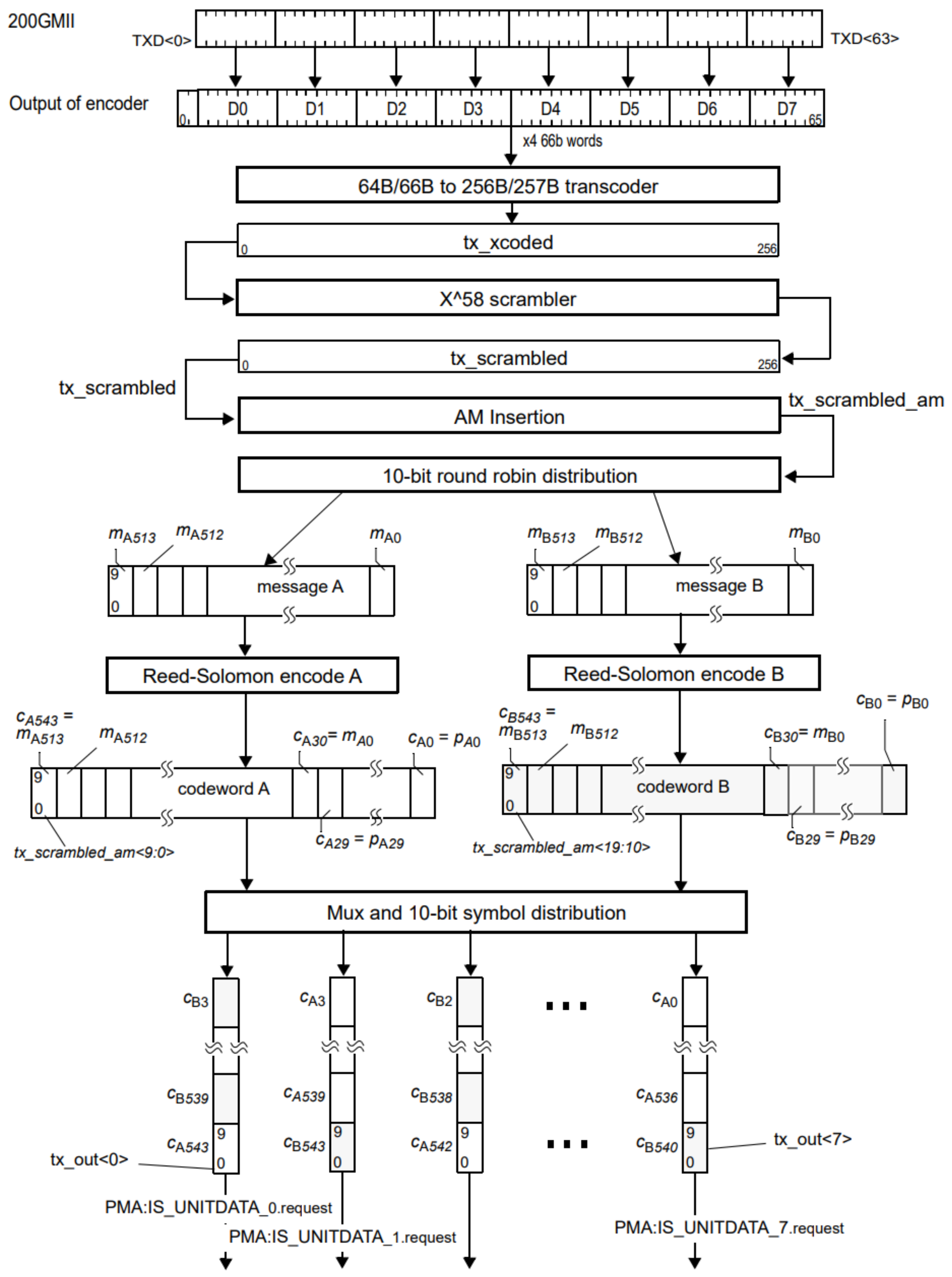


Figure 14. Distribution and Interleaving on 200GE PCS Lanes (Clause 119) [8]

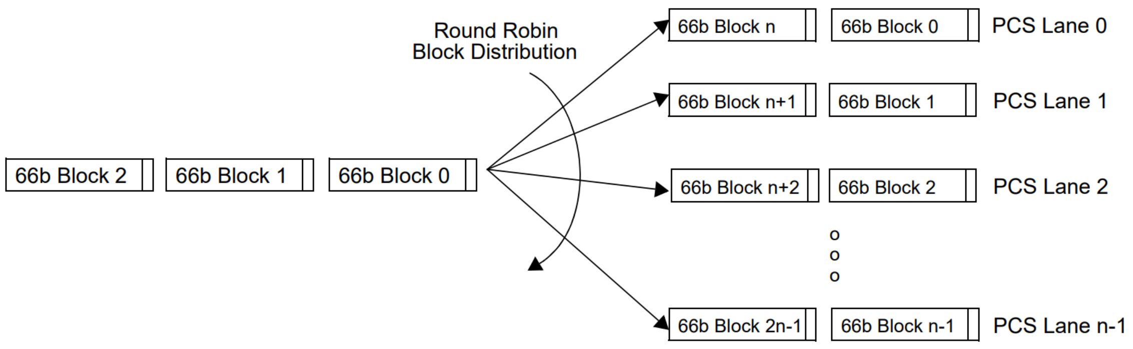


Figure 15. Block Distribution on 100GE PCS Lanes (Clause 82) [8]

2.3.2.7. Alignment Marker Insertion

To support deskew and reordering of individual PCSLs at the receive PCS, alignment markers (AMs) are periodically inserted into each PCSL, as specified in early standards (e.g., Clause 82), as illustrated in *Figure 16*.

- AMs are defined as special 66-bit blocks, which is used in PCS without RS-FEC, with a control block sync header.
- AMs are inserted simultaneously across all PCSLs.
- IPG bytes are removed from the MII data stream to accommodate AMs.
- AMs are inserted after encoding and are not scrambled.
- AMs are designed to maintain DC balance and high transition density, eliminating the need for scrambling.
- In clause 82, one AM is inserted every 16,383 66-bit blocks on each PCSL, which is also called AM distance, as shown in *Figure 17*.

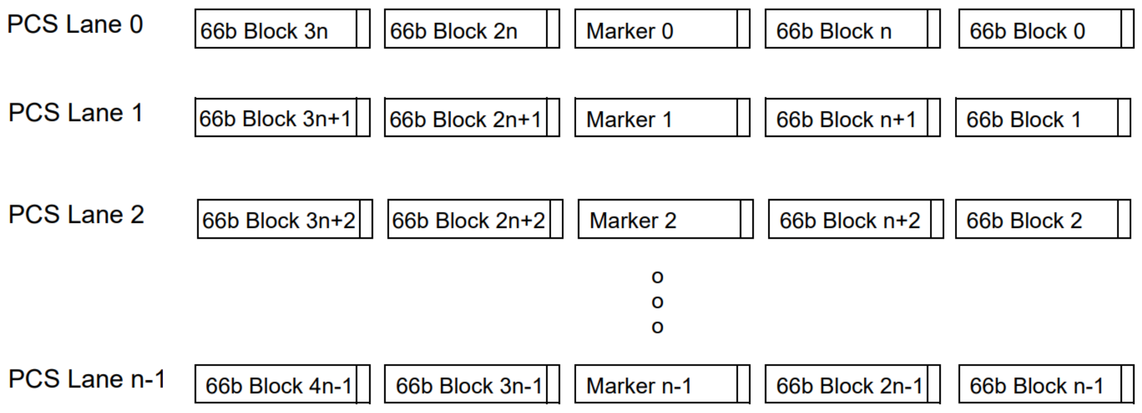


Figure 16. Alignment Marker Insertion for 100GE PCS (Clause 82) [8]

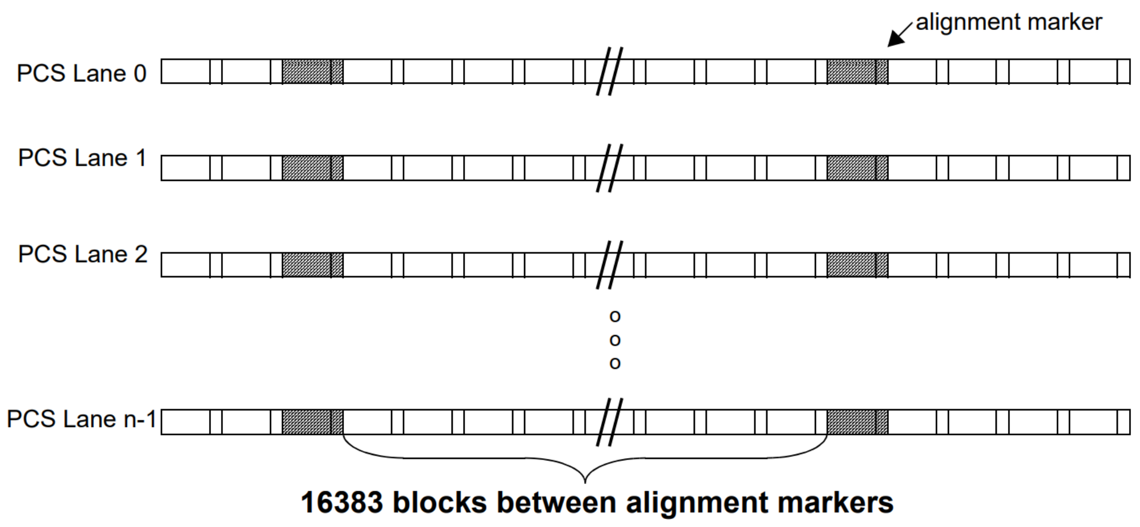


Figure 17. Alignment Marker Distance for 100GE PCS (Clause 82) [8]

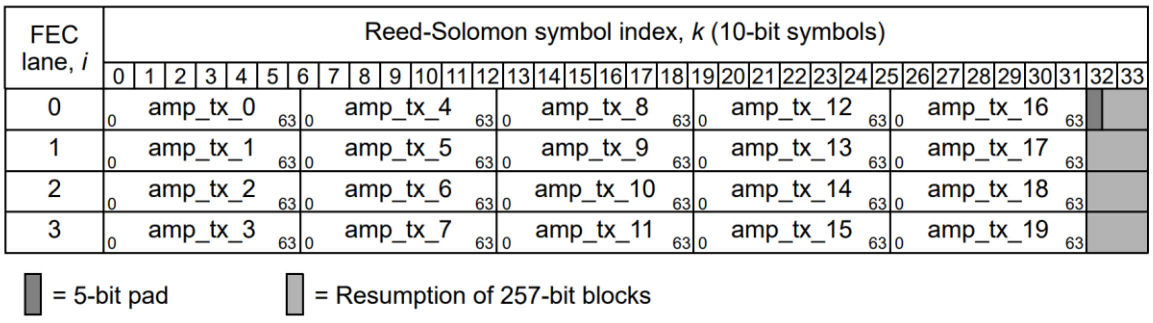
In later standards (e.g., Clause 91), in which PCS has RS-FEC, due to the adoption of 256b/257b transcoding, the AMs defined in earlier standards are removed and mapped into a new format.

- After deskew and alignment, PCSLs are de-multiplexed to reconstruct the original block stream.
- AMs are removed from the data stream once alignment is achieved.

After that, AMs are re-inserted after processing by the AM mapping function. The mapping format is defined differently across clauses, depending on the bit rate and the number of PCSLs.

For instance, the AM mapping function is described in Clause 91 as below.

- A group of 20 AMs, which are aligned and reordered from 20 virual lanes as defined in earlier standards, are mapped as illustrated in *Figure 18*.
- A 5-bit pad is appended to form five 257-bit blocks.
- One group of mapped AMs is inserted every 20 × 16,384 66-bit blocks, corresponding to 4,096 RS-FEC codewords, as specified in Clause 91.



FEC lane, *i*	Reed-Solomon symbol index, *k* (10-bit symbols) 0–6	6–12	12–19	19–25	25–32	32–33
0	0 amp_tx_0 63	0 amp_tx_4 63	0 amp_tx_8 63	0 amp_tx_12 63	0 amp_tx_16 63	▮ ▒
1	0 amp_tx_1 63	0 amp_tx_5 63	0 amp_tx_9 63	0 amp_tx_13 63	0 amp_tx_17 63	▒
2	0 amp_tx_2 63	0 amp_tx_6 63	0 amp_tx_10 63	0 amp_tx_14 63	0 amp_tx_18 63	▒
3	0 amp_tx_3 63	0 amp_tx_7 63	0 amp_tx_11 63	0 amp_tx_15 63	0 amp_tx_19 63	▒

▮ = 5-bit pad ▒ = Resumption of 257-bit blocks

Figure 18. Alignment Marker Mapping for 100GE PCS (Clause 91) [8]

Moreover, Clause 119 describes the AM mapping function as below.

- A set of eight 120-bit AMs is created.
- An additional 65-bit pad and a 3-bit status field are combined to form an AM group of four 257-bit blocks, as shown in *Figure 19*.
- One group of mapped AMs is inserted into the data stream 81 920 × 257-bit blocks.

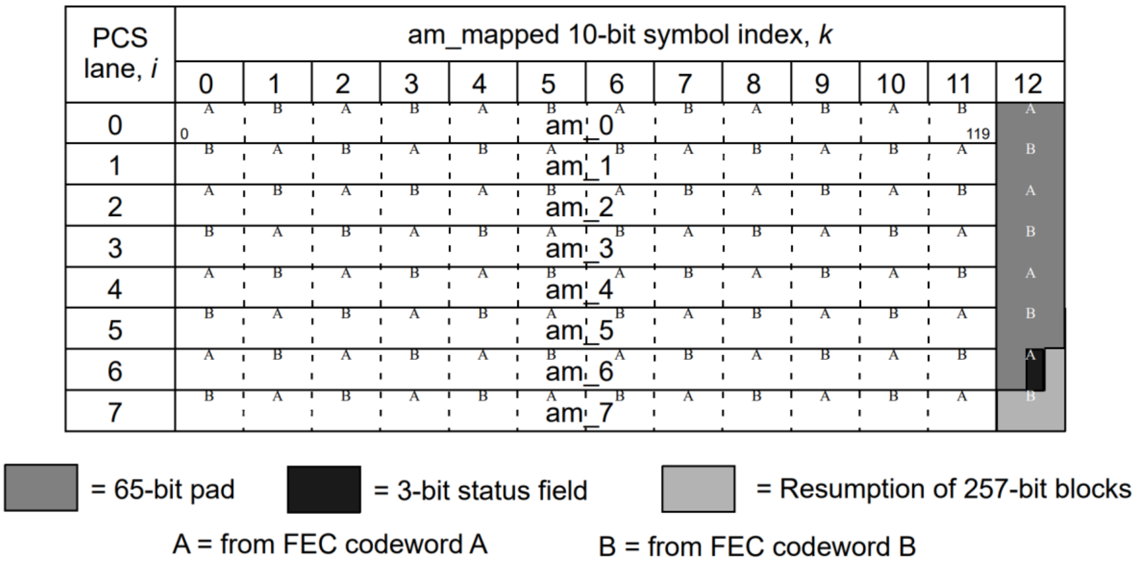


Figure 19. Alignment Marker Mapping for 200GE PCS (Clause 119) [8]

The receive functions of the PCS perform the reverse operations of the transmit direction, as shown in *Figure 20*. These include:

- Alignment lock is established.
- Lane deskew is performed.
- Lane reorder and de-interleave are applied.
- Post-FEC interleave is executed.
- Alignment markers (AMs) are removed.
- Descrambling is applied.
- Reverse transcoding is performed.
- Decoding and rate matching are completed.

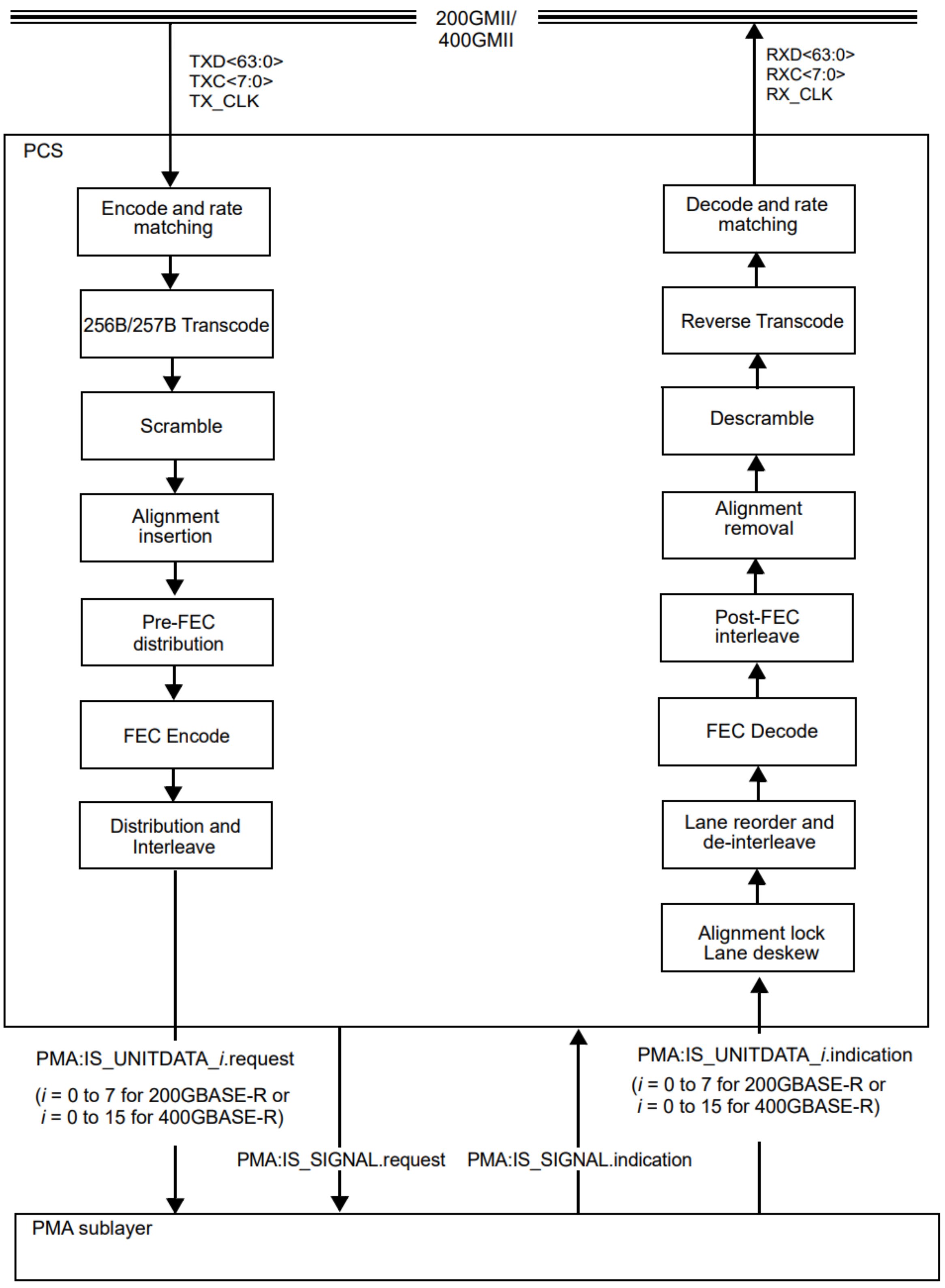


Figure 20. Functional Block Diagram for 200GE/400GE PCS (Clause 119) [8]

2.4. Latest 1.6Tbps Ethernet Standard Investigation

The latest standard regarding to 1.6TE PCS, which is currently under discussion by IEEE, is described in Clause 175. This clause introduces several enhancements to support 1.6 Tbps operation.

2.4.1. Functional Overview

The transmit functions, which is illustrated in *Figure 21* and *Figure 22* defined in Clause 175 are listed as follows:

- 64b/66b encoding is applied to convert 64-bit data into 66-bit blocks.
- Rate matching is performed to compensate for clock domain differences and AM insertion.
- 256b/257b transcoding is executed to reduce line coding overhead.
- Block distribution is applied to alternate 257-bit blocks between two parallel flows (flow 0 and flow 1).
- Scrambling is performed independently on each flow.
- AM mapping and insertion is executed periodically in each flow, as shown in *Figure 23* and *Figure 24*.
- Pre-FEC distribution is applied to organize scrambled blocks into 5140-bit structures for FEC encoding.
- RS encoding is performed on each 5140-bit blockUsing RS (544,514) to generate four codewords.
- Symbol distribution is applied to interleave symbols from four codewords across PCSLs.

2.4.2. Key Innovations

Clause 175 introduces several enhancements that set the 1.6Tbps Ethernet standard apart from previous generations:

- Higher PCS Lane Bandwidth: The standard increases the lane rate from 25Gbps to 100Gbps per PCS Lane (PCSL), significantly reducing the number of lanes required and enabling more efficient data transmission.
- Dual Flow Architecture: After transcoding, data is distributed into two parallel flows. Each flow undergoes independent scrambling, alignment marker (AM) insertion, and RS-FEC encoding. This structure enables the use of four RS-FEC codewords, enhancing burst error correction and improving overall transmission reliability.
- AM Mapping: Alignment markers are inserted and mapped across both flows to maintain synchronization and support deskewing at the receiver. The mapping format is optimized for high-speed operation and ensures robust lane alignment.

- Symbol-Quartet Multiplexing in PMA: The Physical Medium Attachment (PMA) layer adopts a 40-bit symbol multiplexing format, which aligns with the increased lane rate of 200Gbps, as shown in *Figure 25*. This innovation facilitates compatibility with 100Gbps PCSLs and supports efficient symbol distribution across physical lanes.

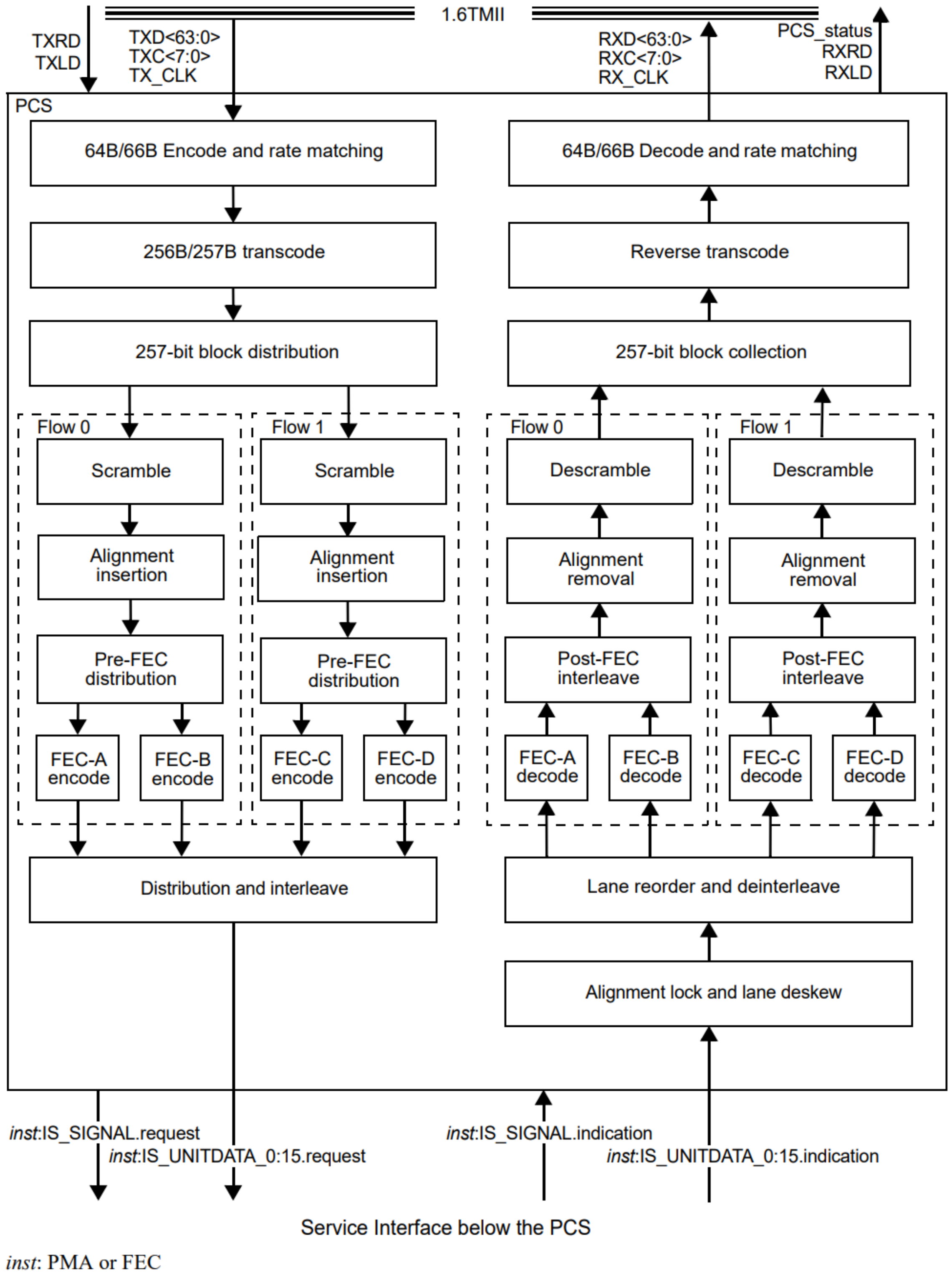


Figure 21. Functional Block Diagram of 1.6TE PCS (Clause 175) [8]

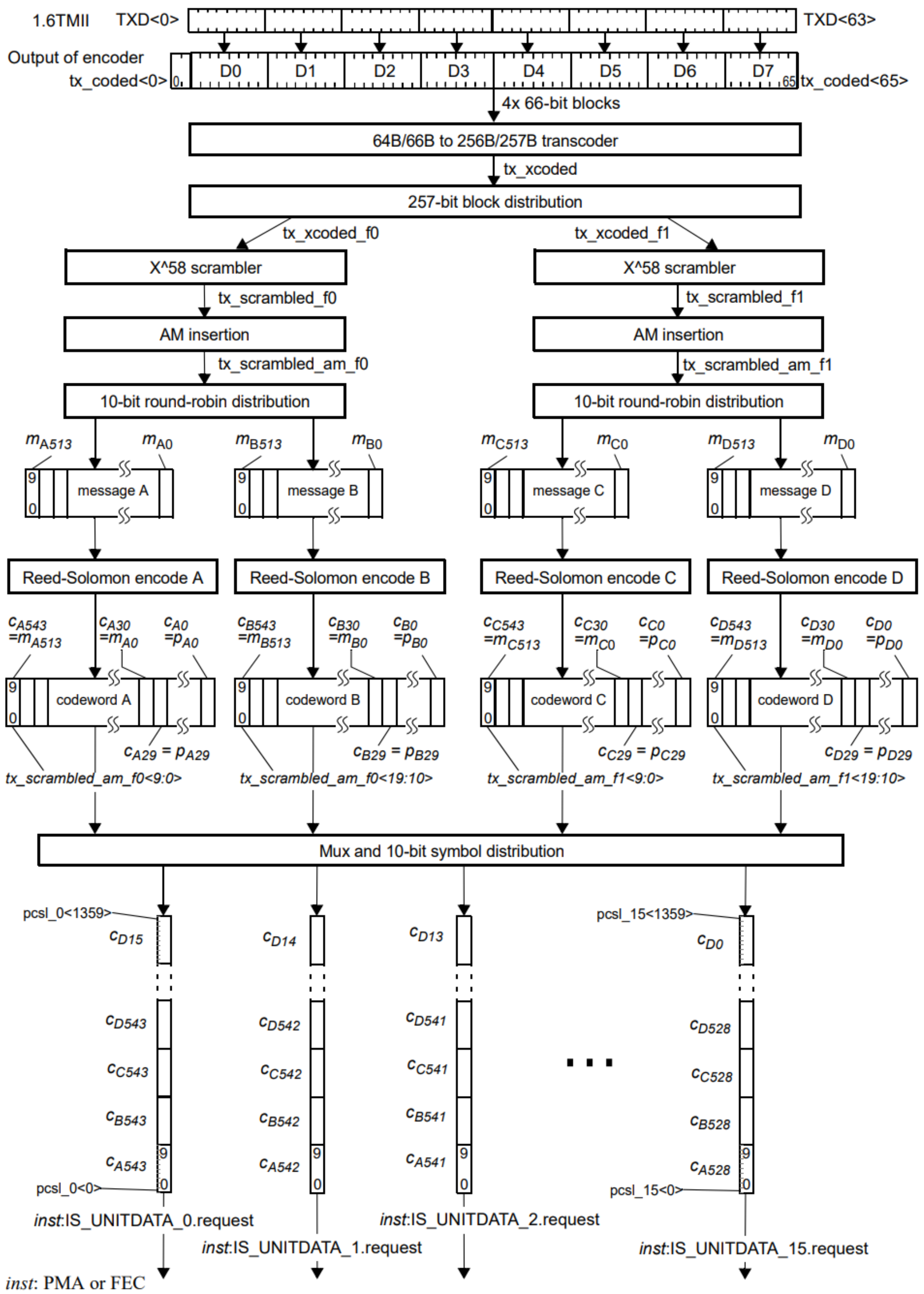


Figure 22. Transmit Data Distribution of 1.6TE PCS (Clause 175) [8]

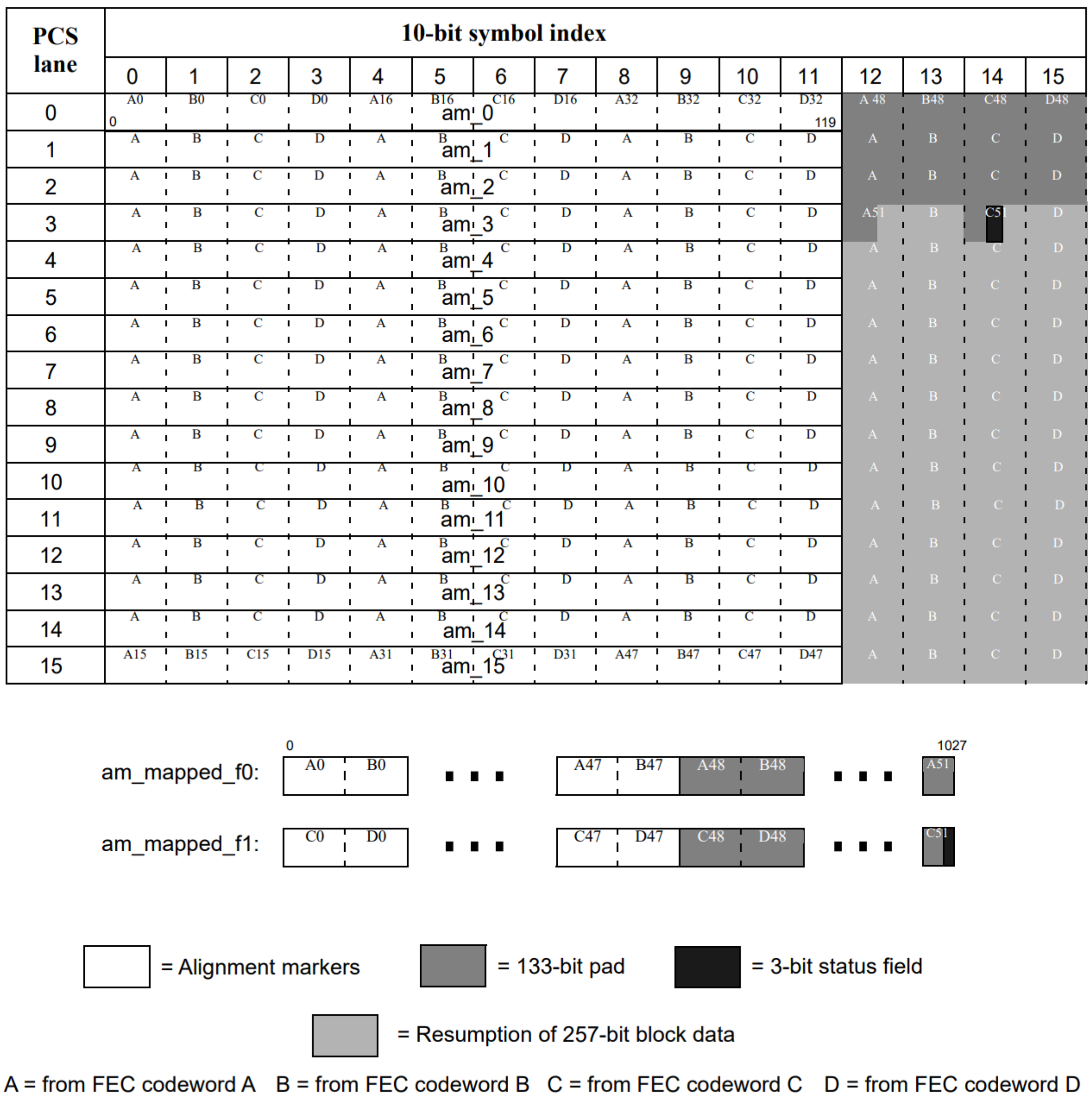

PCS lane
10-bit symbol index
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
A0 B0 C0 D0 A16 B16 C16 D16 A32 B32 C32 D32 A 48 B48 C48 D48
am_0
am_1
am_2
am_3
am_4
am_5
am_6
am_7
am_8
am_9
am_10
am_11
am_12
am_13
am_14
am_15
A15 B15 C15 D15 A31 B31 C31 D31 A47 B47 C47 D47
0
119
A51
C51
am_mapped_f0:
am_mapped_f1:
1027
A0 B0 A47 B47 A48 B48 A51
C0 D0 C47 D47 C48 D48 C51
= Alignment markers
= 133-bit pad
= 3-bit status field
= Resumption of 257-bit block data
A = from FEC codeword A B = from FEC codeword B C = from FEC codeword C D = from FEC codeword D

Figure 23. Alignment Marker Mapping of 1.6TE PCS (Clause 175) [8]

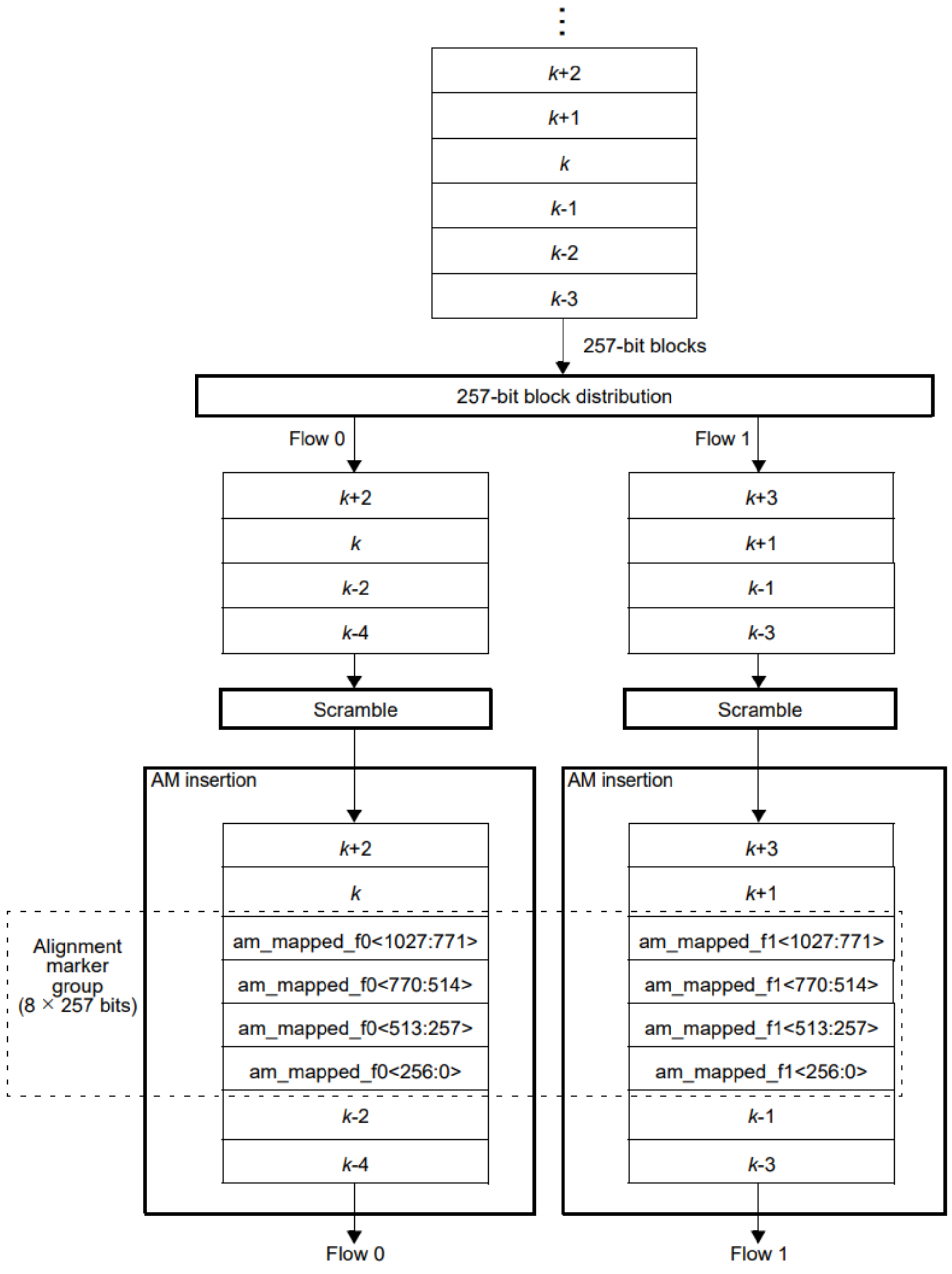


Figure 24. Alignment Marker Insertion of 1.6TE PCS (Clause 175) [8]

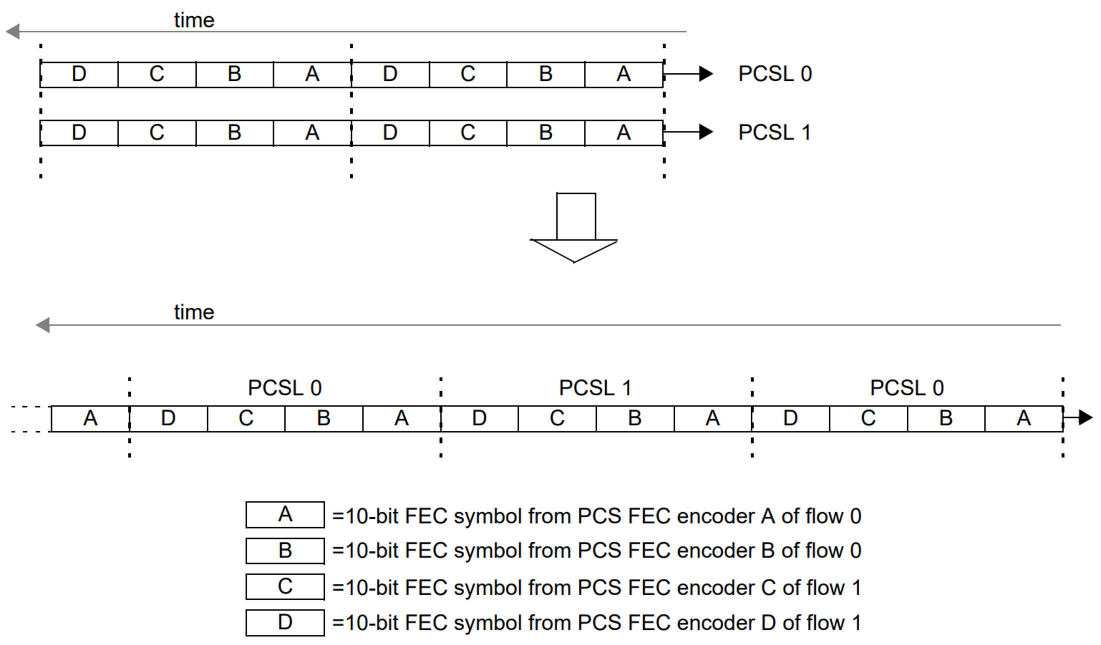

time
D C B A D C B A PCSL 0
D C B A D C B A PCSL 1
time
PCSL 0 PCSL 1 PCSL 0
A D C B A D C B A D C B A
A =10-bit FEC symbol from PCS FEC encoder A of flow 0
B =10-bit FEC symbol from PCS FEC encoder B of flow 0
C =10-bit FEC symbol from PCS FEC encoder C of flow 1
D =10-bit FEC symbol from PCS FEC encoder D of flow 1

Figure 25. Symbol-Quartet Multiplexing used in 1.6TE PCS (Clause 176) [8]

2.5. ***Project Work***

2.5.1. Project Introduction

During the internship, I participated in the verification of a high-speed networking chip integrated into optical modules used in data center environments. The chip supports data transmission rates up to 1.6 Tbps and is designed to meet the requirements defined in the IEEE 802.3 Ethernet standard. My work focused on simulation-based verification of the data path using UVM methodology.

2.5.2. Optical Module

In high-speed data center networks, optical modules serve as the interface between electrical switching systems and optical fiber transmission. These modules perform signal conversion and processing to ensure reliable communication at multi-terabit rates.

The architecture of the optical interconnect system is illustrated in *Figure 26*.

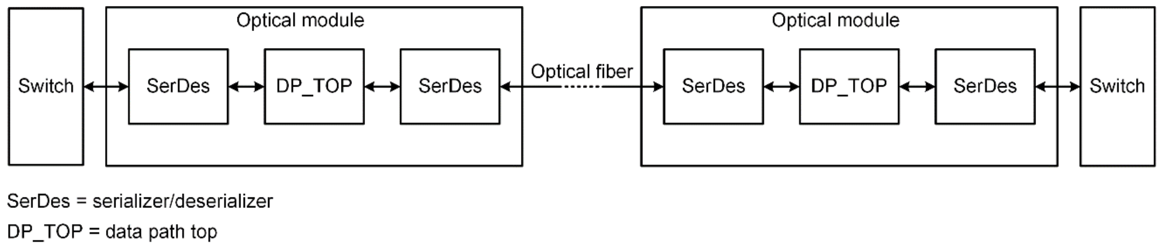


Figure 26. Optical Module Architecture

- Serializer/Deserializer (SerDes) handles the conversion between serial and parallel data formats, transforms electrical signals into optical signals and vice versa.
- Data path top (DP_TOP) is the core logic block responsible for encoding, decoding, and error correction.

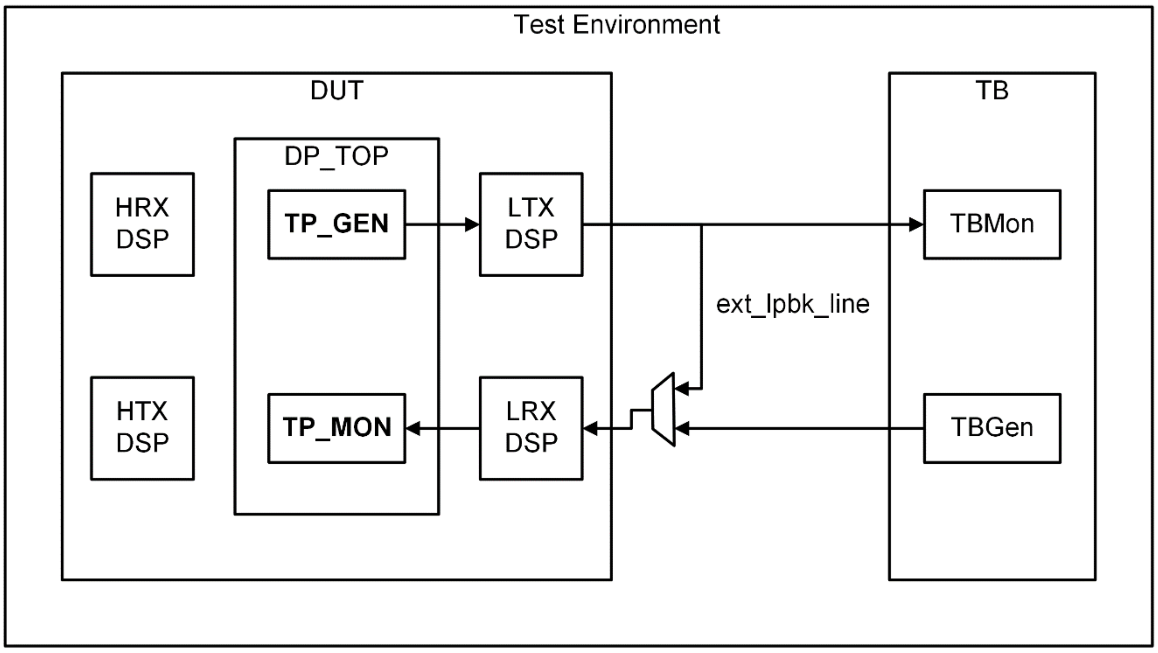
This structure aligns with the layered functions defined in the IEEE 802.3 Ethernet standard, as discussed in Sections 2.3 and 2.4, including PCS, PMA, and RS-FEC mechanisms.

2.5.3. Test Pattern Generator and Monitor

The verification of the DP_TOP block focuses on two key components:

- Test Pattern Generator (TP_GEN): Generates predefined 66-bit patterns such as idle, local fault (LF), and remote fault (RF). These patterns are generated into the egress data path to simulate transmission behavior.
- Test Pattern Monitor (TP_MON): Observes and validates incoming data patterns on the receiving path, ensuring correct reception and protocol compliance.

The general structure of the test environment is shown in *Figure 27*.



TP_GEN = test pattern generator
TP_MON = test pattern monitor
DP_TOP = data path top
HTX/HRX/LTX/LRX = host/line transmit/receive
DSP = digital signal processor
DUT = device under test
ext_lpbk_line = external line loopback
TB = testbench
TBGen = testbench generator
TBMon = testbench monitor

Figure 27. General Diagram of Test Environment

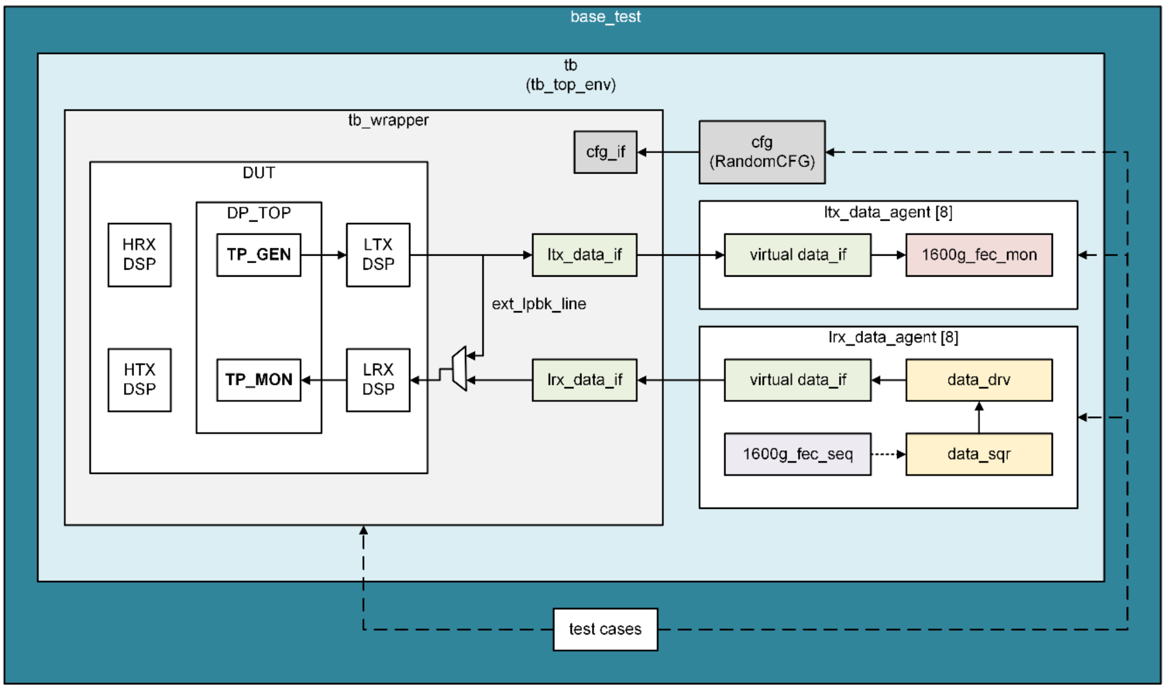
The DUT includes multiple digital signal processing blocks (DSPs), namely HRX, HTX, LRX, and LTX, which interface with the core data path logic (DP_TOP). Within DP_TOP, the TP_GEN and TP_MON blocks are responsible for injecting and validating test patterns.

The testbench consists of TBGen and TBMon, which respectively generate stimulus and monitor responses.

2.5.4. Testbench Environment

The testbench is implemented using UVM and structured to support modular verification of the DUT, as shown in *Figure 28*. It is built upon a base test (base_test) layer and includes a top-level environment (tb_top_env) that encapsulates all verification components.

Within the environment, the wrapper (tb_wrapper) contains the DUT and its associated interfaces (ltx_data_if and lrx_data_if). Configuration is managed through the cfg block (cfg), which applies randomized settings via cfg_if. Data agents (ltx_data_agent and lrx_data_agent) handle transmission and reception, each containing virtual interfaces and monitoring sequences.



TP_GEN = test pattern generator
TP_MON = test pattern monitor
DP_TOP = data path top
HTX/HRX/LTX/LRX = host/line transmit/receive
DSP = digital signal processor

DUT = device under test
ext_lpbk_line = external line loopback
if = interface
tb = testbench
cfg = configuration

mon = monitor
drv = driver
seq = sequence
sqr = sequencer
env = environment

Figure 28. Testbench Environment Overview

This environment enables controlled configuration, pattern injection, and data monitoring across multiple datapath agents, ensuring full coverage of the verification scope.

2.5.5. Test Scenarios

Three simulation scenarios were executed to verify the functionality of the TP_GEN and TP_MON blocks under different datapath configurations, as shown in *Figure 29* and *Figure 30*. Each scenario follows a structured procedure to ensure consistent setup, execution, and result validation.

2.5.5.1. Test Scenario for TP_GEN

The objective is to verify that TP_GEN correctly generates and transmits a standard pattern through the egress datapath.

- Step 1: Configuration
  - Set the DUT to pattern generation mode.
  - Configure TP_GEN to generate a predefined pattern.
  - Apply the necessary configuration settings.
  - Enable TBMon to capture and validate the transmitted pattern.
- Step 2: Execution
  - Start the simulation.
- Step 3: Verification
  - Check the captured data in TBMon.
  - Validate the status counters and pattern integrity.
  - Report the test result.

2.5.5.2. Test Scenario 1 for TP_MON (Direct Path)

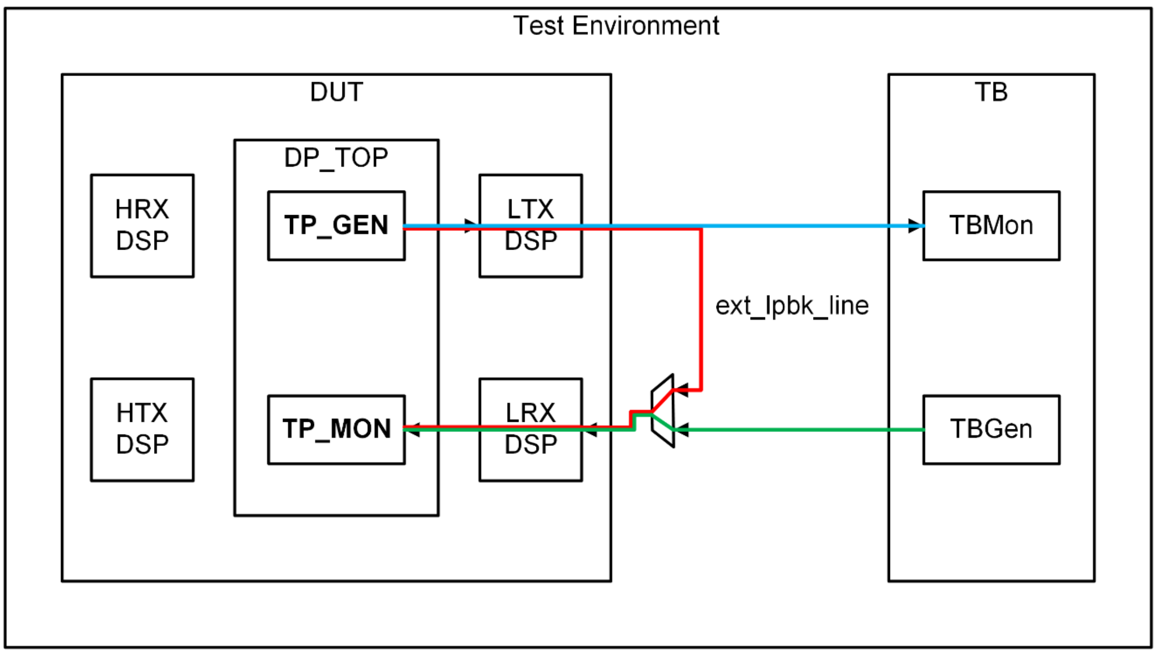
The objective is to verify that TP_MON correctly monitors and validates incoming patterns generated directly by the testbench.

- Step 1: Configuration
  - Set the DUT to pattern monitoring mode.
  - Configure TBGen to generate a predefined pattern.
  - Route the pattern through the multiplexer to TP_MON.
  - Apply the necessary configuration settings.
  - Enable TP_MON to monitor and validate the incoming data.
- Step 2: Execution
  - Start the simulation.
- Step 3: Verification
  - Check the received pattern at TP_MON.
  - Validate the status counters and pattern integrity.
  - Report the test result.

2.5.5.3. Test Scenario 2 for TP_MON (Loopback Path)

The objective is to verify that TP_MON correctly monitors patterns looped back externally from TP_GEN.

- Step 1: Configuration
  - Set the DUT to pattern generation mode and monitoring mode.
  - Configure TP_GEN to generate a predefined test pattern.
  - Enable external loopback (ext_lpbk_line).
  - Apply the necessary configuration settings.
  - Enable TP_MON to monitor the looped-back data.
- Step 2: Execution
  - Start the simulation.
- Step 3: Verification
  - Check the received pattern at TP_MON.
  - Validate the status counters and pattern integrity.
  - Report the test result.

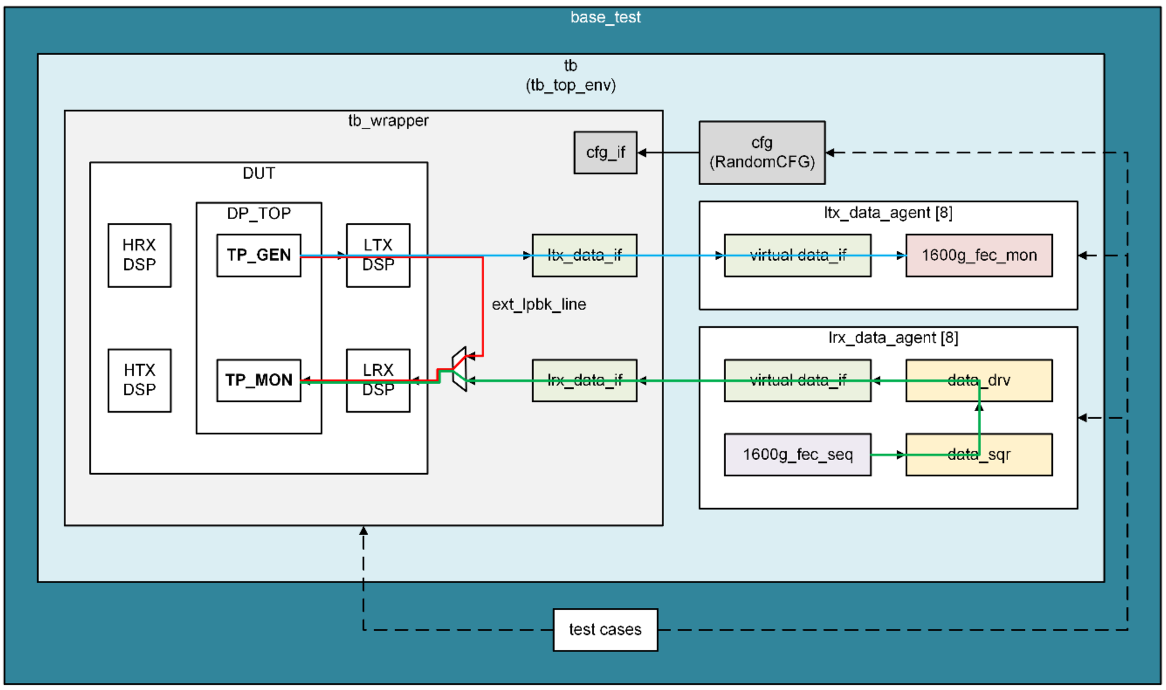


TP_GEN = test pattern generator
TP_MON = test pattern monitor
DP_TOP = data path top
HTX/HRX/LTX/LRX = host/line transmit/receive
DSP = digital signal processor

DUT = device under test
ext_lpbk_line = external line loopback
TB = testbench
TBGen = testbench generator
TBMon = testbench monitor

🟦 Test scenario for TP_GEN
🟥 Test scenario (1) for TP_MON
🟩 Test scenario (2) for TP_MON

Figure 29. General Diagram of Test Scenarios



TP_GEN = test pattern generator	DUT = device under test	mon = monitor	🟦 Test scenario for TP_GEN
TP_MON = test pattern monitor	ext_lpbk_line = external line loopback	drv = driver	🟥 Test scenario (1) for TP_MON
DP_TOP = data path top	if = interface	seq = sequence	🟩 Test scenario (2) for TP_MON
HTX/HRX/LTX/LRX = host/line transmit/receive	tb = testbench	sqr = sequencer	
DSP = digital signal processor	cfg = configuration	env = environment	

Figure 30. Test Scenarios

2.5.6. Results and Observations

All test scenarios were executed successfully. Simulation logs and waveform analysis confirmed that the DUT behaved as expected. Status counters and pattern checks passed, indicating correct implementation of TP_GEN and TP_MON functionalities.

3. INTERNSHIP SUMMARY

3.1. Results

During the 13-week internship at Marvell Technology, the following technical and professional outcomes were achieved:

- Design Verification Knowledge: Gained comprehensive understanding of the ASIC design flow, with a focus on the RTL simulation phase. Applied SystemVerilog and UVM to construct modular and reusable testbenches for high-speed digital designs.
- SystemVerilog Proficiency: Studied and implemented advanced features such as data types, casting, operators, tasks, interfaces, object-oriented programming, packages, constrained randomization, assertions, and coverage.
- UVM Methodology: Developed a complete UVM-based testbench architecture, including environment, agents, drivers, monitors, scoreboards, and sequencers. Understood UVM phases and their role in simulation control and verification.
- IEEE 802.3 Ethernet Standards: Investigated key concepts in PCS and FEC layers, including encoding schemes (64b/66b, 256b/257b), scrambling, RS-FEC encoding, PCS lane distribution, and alignment marker insertion and mapping.
- 1.6Tbps Ethernet Standard: Explored Clause 175 of the IEEE 802.3 standard, focusing on dual-flow architecture, enhanced lane bandwidth, AM mapping, and 40-bit multiplexing in PMA.
- Project Execution: Participated in the verification of a high-speed networking chip integrated into optical modules. Designed and executed test scenarios for TP_GEN and TP_MON blocks using UVM methodology.
- Simulation and Debugging: Utilized QuestaSIM for simulation and waveform analysis. Verified DUT behavior through log inspection, waveform dumping, and status counter validation.
- Successful Test Outcomes: All test scenarios were executed successfully. The DUT met expected behavior criteria, confirming the correctness of TP_GEN and TP_MON implementations.

3.2. Lessons Learned

- Technical Growth: Developed strong proficiency in SystemVerilog and UVM, enabling effective construction and debugging of verification environments.
- Standard Familiarity: Gained in-depth knowledge of IEEE 802.3 standards, particularly the PCS and FEC mechanisms critical for high-speed Ethernet communication.
- Verification Strategy: Learned the importance of structured test planning, scenario definition, and result validation in ensuring design correctness.

- Tool Utilization: Acquired hands-on experience with industry-standard tools such as QuestaSIM, enhancing simulation and debugging capabilities.
- Professional Development: Improved communication, documentation, and collaboration skills through interaction with mentors and team members in a professional engineering environment.

4. REFERENCES

[1] Marvell Technology, Inc., "About Marvell," *Marvell*, [Online]. Available: https://www.marvell.com/company/about-marvell.html (accessed Jun. 13, 2025).

[2] Marvell Technology, Inc., "Markets," *Marvell*, [Online]. Available: https://www.marvell.com/solutions.html (accessed Jun. 13, 2025).

[3] Marvell Technology, Inc., "Careers," *Marvell*, [Online]. Available: https://www.marvell.com/company/careers.html (accessed Jun. 13, 2025).

[4] Arrive Technologies, *Guideline for verification*, Feb. 3, 2020.

[5] Chris Spear and Greg Tumbush. *SystemVerilog for Verification: A Guide to Learning the Testbench Language Features*. Springer, 2012. ISBN: 978-1461407140.

[6] Accellera Systems Initiative, *Universal Verification Methodology (UVM) 1.2 User's Guide*, Oct. 8, 2015.

[7] Mentor Graphics, *UVM Cookbook*, Siemens EDA (formerly Mentor Graphics), 2014.

[8] IEEE Standard for Ethernet, IEEE Std 802.3™, IEEE Standards Association, New York, NY, USA. Available: https://standards.ieee.org/standard/802_3-2018.html

[9] R. Bierman, *Reed-Solomon FEC Protocol and Testing*, Inphi Corporation, Proprietary and Confidential, ver. 0.1, Aug. 27, 2020.