Design and Verification of Peripheral Component Interconnect Express (PCIe) 3.0

K U Prasad Bhat[1], Hrishikesh Ravish[2], Anirudha V[3], Prabhavathi P[4], Kumaraswamy K V[5]

[1,2,3]Student, Department of Electronics and Communication Engineering, BNMIT, Bengaluru, India
[4]Associate Professor, Department of Electronics and Communication Engineering, BNMIT, Bengaluru, India
[5]Senior Technical Manager, Trident Techlabs Pvt. Ltd, Bengaluru, India

---***---

Abstract - *PCIe (Peripheral Component Interconnect Express), is a common motherboard interface standard for hard drives, SSDs, Ethernet and Wi-Fi connections as well as graphic cards with a computing system. PCIe provides multiple direct links that allows multiple devices to communicate with each other simultaneously. Many different versions of PCIe are developed in order to increase speed, bandwidth and data width. Hence, this project aims at developing a soft IP for PCIe 3.0 standard protocol. This had been modeled in Verilog HDL, simulated in Questa Simulator and synthesized in Precision RTL. PCIe 3.0 soft IP is developed as per FPGA design methodology with a clock frequency of 100 MHz, for Xilinx Vivado device. Equivalence Check was performed using FormalPro. Finally, the proposed design was checked for DO-254 standard.*

Key Words: **PCIe 3.0, Soft IP, Verilog HDL, FPGA, Simulation, Synthesis, Equivalence Check, Gate-Level Simulation, DO-254 Standard.**

1. INTRODUCTION [1]

PCIe (Peripheral Component Interconnect Express), is a high speed computer expansion bus standard, designed to replace the older PCI, PCI-X and AGP bus standards. It is the common motherboard interface for personal computers' graphic cards, hard drives, SSDs, Ethernet and Wi-Fi hardware connections. Conceptually, the PCI Express bus is a high-speed serial replacement of the older PCI/PCI-X bus. One of the key differences between the PCI Express bus and the older PCI is the bus topology; PCI uses a shared parallel bus architecture, in which the PCI host and all devices share a common set of address, data and control lines. In contrast, PCI Express is based on point-to-point topology, with separate serial links connecting every device to the root complex (host). PCIe has numerous improvements over the older standards, including higher maximum system bus throughput, lower I/O pin count and smaller physical footprint, better performance scaling for bus devices, a more detailed error detection and reporting mechanism (Advanced Error Reporting, AER), and native hot-swap functionality. More recent revisions of the PCIe standard provide hardware support for I/O virtualization. Format specifications are maintained and developed by the PCI-SIG (PCI Special Interest Group), a group of more than 900 companies that also maintain the conventional PCI specifications.

1.1 PCI EXPRESS LINK[1]

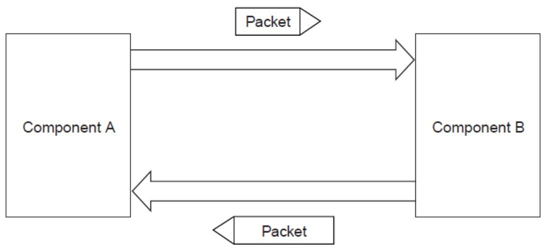


Fig-1: PCIe Link

Figure 1 represents the PCI Express link. A Link represents a dual-simplex communications channel between two components. The fundamental PCI Express Link consists of two, low-voltage, differentially driven signal pairs: a Transmit pair and a Receive pair.

1.2 PCIe 3.0 LAYERED TOPOLOGY[1][2]

PCI Express protocol communication mechanism consists of three layers.

- Transaction Layer
- Data Link Layer
- Physical Layer

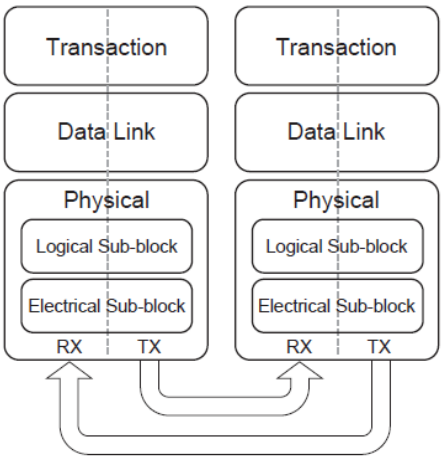


Fig-2: Overview of PCIe Protocol Layering

The communication between the target (Rx) and initiator (Tx) happens as per the layers, shown in Figure 2. PCI Express uses packets to communicate information between components. Packets are formed in the Transaction and Data Link Layers to carry the information from the transmitting component to the receiving component. As the transmitted packets flow through the other layers, they are extended with additional information necessary to handle packets at those layers. At the receiving side, the reverse process occurs and packets get transformed from their Physical Layer representation to the Data Link Layer representation and finally (for Transaction Layer Packets) to the form that can be processed by the Transaction Layer of the receiving device.

1.3 FPGA DESIGN METHEDOLOGY

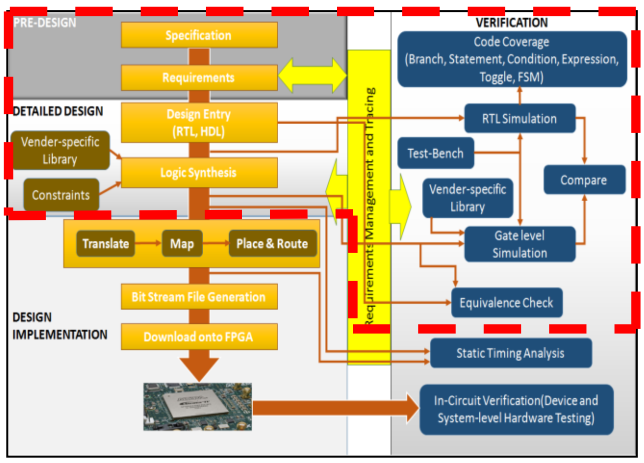


Fig -3: FPGA Design flow diagram

▬ ▬ Encloses operations performed in this project

- **Specification & Requirements**

  At First, the FPGA design functionality and power/area/speed specifications are decided. The design architecture is then created based on these specifications. The architecture would be normally partitioned into sub-modules that interact with each other to form the system level module. The FPGA development board on which the design will be uploaded should also be chosen in accordance with price budget. Table 1 gives a brief overview of the design specification considered for this project work under FPGA Design methodology.

Table -1: FPGA Design Specification for implementing PCIe 3.0

Design Implementation Methodology	FPGA	
FPGA Board Specification	**Vendor**	Xilinx
	Family	Spartan-6
	Device	6SLX16CSG324
	Speed Grade	-3
Frequency of Operation	100 MHz	
Data Input	32-bit	
Reset	Active Low	
Header	FA(base-16) for Resource Request*	
	AF(base-16) for Completion Request*	
Trailer	77(base-16)*	

[*Examples of headers and trailers are used and are for illustration purpose only]

- **Design Entry**

  Design files could be uploaded to the FPGA CAD tool in either schematic format or in HDL (Hardware Description Language) format as indicated in Figure 3. In this project the design functionality and a testbench to verify it, has been written in Verilog language.

- **RTL Simulation with Code Coverage**

  It is performed before synthesis. A simulator tool will take a Verilog source code and a testbench as inputs. Input files are compiled, optimized for all coverages and simulated. Waveform, schematics and percentage of coverage reports are the outputs of simulation process.

- **Logic Synthesis**

  Synthesis is the transformation of RTL Design to Technology specific gate level netlist considering all design and library constructs. Input to the synthesis files would be source code along with constraint file in .sdc (Synopsis Design Constraints) format. At this stage an FPGA device on which the design will be implemented is chosen and synthesis is carried out for the same device specification. Outputs of synthesis process are gate-level netlist, Technology schematic, SDC (Synopsis Design Constraints), SDF (Standard Delay Format), synthesis reports (Timing and area).

- **Equivalence Check**

  Here the functional equivalence is checked between RTL and gate-level netlist.

- **Gate-Level Simulation**

  Here the gate-level netlist is considered instead of RTL for verification. The same testbench is used as that was used during RTL simulation. Further, the simulated waveforms of RTL and gate-level simulation are compared to identify difference points.

- **Design Implementation**

  The Design Implementation process involves three steps: Translate, Map, Place&route.

- **Static Timing Analysis (STA)**

  Static Timing Analysis is carried out after the design implementation to check that the design follows the timing constraints. It checks all the possible signal propagation paths for delays.

- **Bit stream file generation**

  The implemented design must then be converted into a Bit stream, using a bit-generation tool, so that the FPGA platform can understand the design. The Bit stream file is then stored on the FPGA memory card, so it can be uploaded by the board.

- **Downloading onto FPGA**

  Now the design must be loaded onto the FPGA. After the Bit stream file is uploaded on the FPGA, in-circuit verification is carried out to ensure that correct circuit implementation has taken place. This is done using the hardware debugging IPs integrated in the FPGA board.

In this project operations till gate-level simulation have been carried out.

1.4 Requirement Tracing and DO-254 Standard rule Checks[11]

DO-254 is defined as a requirements-driven process-oriented safety standard used on commercial electronics that go into aircraft. (Conceptually speaking, this standard applies to all electronics in anything that flies or could crash and pose a hazard to the public.)

Based on their safety criticality, different parts of the aircraft are designated different Design Assurance Levels (DALs) as shown in Table 2. A system that is highly critical will receive a higher DAL, with DAL A reserved for the most critical systems. This criticality is determined by a safety assessment of the aircraft and interacting systems to determine the required target failure rate. For DO-254, the difference between meeting DAL A and DAL B is minimal, so they are frequently referred to as “DAL A/B”.

Table -2: Design Assurance Levels (DALs)

Design Assurance Level (DAL)	Description	Target System Failure Rate	Example System
Level-A (Catastrophic)	Failure causes crash, deaths	<1 x 10-9 chance of failure/flig ht-hr	Flight Controls
Level-B (Hazardous)	Failure may cause crash, deaths	<1 x 10-7 chance of failure/flig ht-hr	Braking systems
Level-C (Major)	Failure may cause stress, injuries	<1 x 10-5 chance of failure/flig ht-hr	Backup systems
Level-D (Minor)	Failure may cause inconvenience	No safety metric	Ground navigation systems
Level-E (No Effect)	No safety effect on passengers/cr ew	No safety metric	Passenger entertainme nt

As DO-254 is a process-oriented standard, it is important to understand the overall flow, shown in Figure 4.

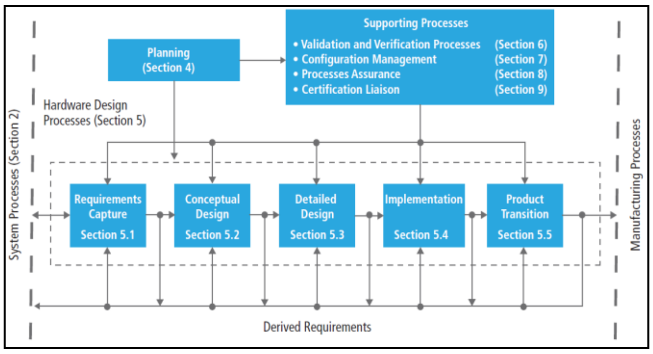

Supporting Processes
• Validation and Verification Processes (Section 6)
• Configuration Management (Section 7)
• Processes Assurance (Section 8)
• Certification Liaison (Section 9)
Planning (Section 4)
Hardware Design Processes (Section 5)
System Processes (Section 2)
Manufacturing Processes
Requirements Capture Section 5.1
Conceptual Design Section 5.2
Detailed Design Section 5.3
Implementation Section 5.4
Product Transition Section 5.5
Derived Requirements

Fig -4: DO-254 Flow diagram

- **Planning**

  Planning is a critical piece of the DO-254 certification. It is important to document project flow up-front and approach the certification official to gain their approval early in the project. Typically the high-level plans are documented in the Plan for Hardware Aspects of Certification (PHAC commonly pronounced as "pea-hack"). This plan should include all aspects of the project and how designer will meet the DO-254 requirements.

- **Requirements Capture and Validation**

  The DO-254 specification utilizes a requirements-based design and verification approach. This means that the entire hardware project revolves around a formal set of high-level requirements. Before any RTL is written, each of these requirements must be written down, given a unique reference name, and reviewed for a variety of criteria including understandability, testability, verifiability, etc.

- **Conceptual Design**

  A larger design is chunked down into a smaller and more manageable sub-blocks. This might be thought of as a high-level block diagram. (Note: For a sufficiently simple system, the conceptual design step may be skipped or merged with the Detailed Design step).

- **Detailed Design**

  This step is where the real design work takes place. For each component detailed in the conceptual design, the RTL hardware design should implement each and every requirement for that component. Each high-level requirement should be "traced" to the top-level RTL module implementing that requirement. This traceability can happen in a variety of ways, and it is up to the implementation team to determine the desired approach. On the other hand, the verification team should create verification tests to verify that each requirement has been met by the RTL, including a message to the log file showing the expected result, the actual result seen in the simulation and the result (pass/fail). Each test must also be linked to the high-level requirement, including the pass/fail criteria (all must pass).

- **Implementation**

  The implementation process is technology specific. For an RTL-based design (such as an FPGA or ASIC), the implementation step includes the synthesis process of converting RTL into actual technology-specific gates. For an FPGA, this also includes creating the programming file to load into the FPGA. For an ASIC, this step includes the backend design/verification steps. Here, the main point is to follow the process detailed in PHAC document up-front. This is due to the fact that there will be significant testing performed on the final design. Traceability analysis for all the above steps is performed as shown in Figure 5.

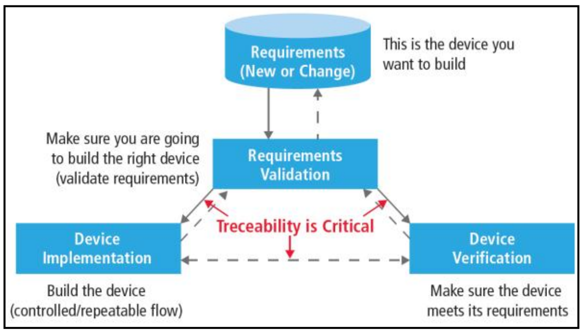

Requirements (New or Change)
This is the device you want to build
Make sure you are going to build the right device (validate requirements)
Requirements Validation
Treceability is Critical
Device Implementation
Device Verification
Build the device (controlled/repeatable flow)
Make sure the device meets its requirements

Fig -5: Requirement-driven flow, including traceability

- **Production Transition**

  This is the final stage, where design is transferred to manufacturing unit. This stage ensures aspects such as:

  - Use of correct version of the programming file during the manufacturing process.
  - Use the of correct design methodology (ASIC and FPGA).
  - Handling of errata for the device used.

- **Process Assurance**

  Along with DO-254-compliant plan, designer should also document to meet plan, typically documented in a Process Assurance or Quality Assurance plan. This plan documents who will be designated as the process assurance person or organization to double check that the PHAC and other plans are followed, and how this checking will be performed.

- **Configuration Management**

  In addition to the Process Assurance plan, designer should also create a Configuration Management (CM) plan. In this plan, documentation of how to ensure the development process and artifact generation process is repeatable. This typically includes a revision control and bug tracking systems for all design/verification files, as well as all documentation and artifact documents. The DO-254 specification refers to the importance of tracking all design artifacts throughout the design process. Certification officials understand that design and verification files will go through much iteration. However, once they are stable, designers are expected to "baseline" the design. In typical commercial electronics, this is analogous to a design freeze a point in a schedule when subsequent changes are closely controlled and documented, as shown in Figure 6.

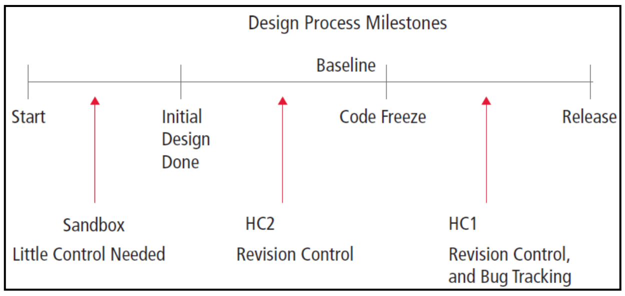


Fig -6: Example for Design process and baselines

- **Certification Liaison**

  Typically, a single person is selected as the main communication point for the certification officials. This single point of contact enables clean communication, and ensures that the certification official obtains a clear view of the overall design process. Typically, this certification liaison has previous DO-254 experience, with the skill to communicate the details in a way that the certification official can understand.

- **In-Target Testing**

  In-target testing is a critical component of the DO-254 specification, and is a required part of the overall flow. From a DO-254 perspective, all verification done in a simulator was performed on a model of the design. There is no guarantee that the model used in simulation matches the actual device as it sits on the target board that will be installed in the aircraft.

- **DO-254 Standard Rule sets[12]**

  For a design to obtain DO-254 Certificate it should satisfy all the rules mentioned in Table 3.

Table -3: DO-254 standard rule sets

Coding Practices	Design Reviews	Safe Synthesis
Assign Value Before Using	Avoid Large Design Files	Asynchronous Block
Assignment Style (Verilog) - Combinatorial Blocks Inferring Latches	Avoid Mixed Case Naming (for Differentiation)	Avoid Asynchronous Reset Release
Assignment Style (Verilog) - Pure Combinatorial Blocks	Avoid Using Tab	Avoid Clock Used As Data
Assignment Style (Verilog) - Sequential Blocks	Ensure Consistent File Header	Avoid Combinational Feedback
Avoid Duplicate Signal Assignments	Ensure Consistent Indentation	Avoid Feed troughs
Avoid Hard-Coded Numeric Values	Ensure Proper Placement of Comments	Avoid Gated Clocks
Avoid Hard-Coded Vector Assignments	Ensure Sufficient Comment Density	Avoid Implied Logic
Avoid Incorrect VHDL Type Usage	Ensure Unique Name Spaces	Avoid Initialization Assignments
Avoid Mismatching Ranges	Use Separate Declaration Style	Avoid Internally Generated Clocks
Avoid Unconnected Input Ports	Use Separate Statement Style	Avoid Internally Generated Resets
Avoid Unconnected Output Ports	Use Statement Labels	Avoid Latch Inference
Avoid Unused Declarations		Avoid Mixed Polarity Reset
Define All Design Units		Avoid Multiple Drivers
Ensure Complete Sensitivity List		Avoid Multiple Waveforms
Ensure Consistent FSM State Encoding Style		Avoid Shared Clock and Reset Signal
Ensure Proper Sub-Program Body		Avoid Snake Paths
Ensure Safe FSM Transitions		Avoid Undriven & Unused Logic

Mixed Variable Assignments		Avoid Uninitialized Deferred Constants
		Avoid Unresettable Register
		Continuous Assignments
		Ensure Consistent Vector Order
		Ensure Nesting Limits
		Ensure Proper Case Statement Specification
		Ensure Register Controllability
		Unsynthesizable Event Controls

2. PROPOSED DESIGN

The proposed design will take a 32-bit data as input from the user. Then the input data flows through all the three layers of PCIe naming transaction, data-link and physical layers and is processed at each of the above three layers present at transmitter and receiver respectively, according to their functionalities. At receiver, received data will be verified at each of the layer to ensure data integrity, data reliability and error free transmission through the physical links. Proper handshake mechanism will be used by receiver to notify the transmitter about the status of the data. Block diagram emphasizing the input and output ports of the PCIe 3.0 design is as shown in Figure 7.

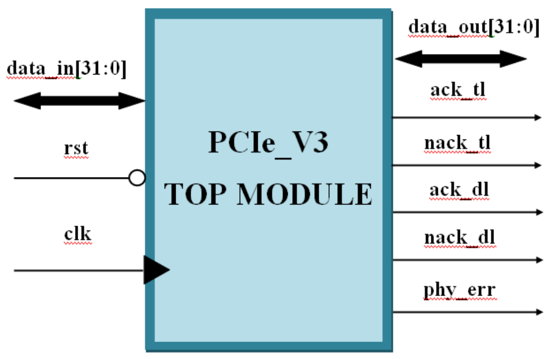


Fig -7: PCIe 3.0 Top Module Block diagram

The flow chart representing a 32-bit input data passing through all the layers and components present in PCIe 3.0 has been indicated in Figure 8.

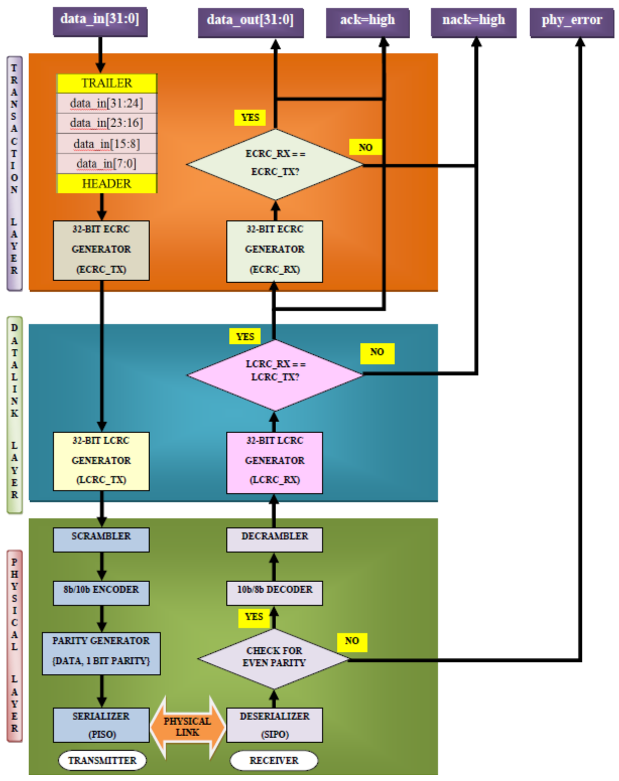


Fig -8: Data processing across all three layers of PCIe 3.0

Components present in each of the three layers are explained in sub-sections below.

2.1 TRANSACTION LAYER[1][3]

- **Header-Trailer Block[1]**

  At transmitter, when user inputs a 32-bit data, it is broken and stored in a FIFO buffer of 8-bit width and 6 stacks as shown in Figure 8. 8-bit Header and trailer values assumed in Table 1 are used only for illustration purpose.

- **32-bit ECRC (End-to-End Cyclic Redundancy Check) block[3]**

  At transmitter, for every 8-bit data a 32-bit ECRC value is computed and stored in CRC field of Transaction Layer Packet (TLP).

  Again at receiver a 32-bit ECRC value is computed for each received TLP. The received ECRC value is compared with ECRC value transmitted. If they are equal, then an ACK[5] (Acknowledge) signal is sent to the transmitter notifying data has been received with no corruption; else a NACK[5] (No Acknowledge) signal is sent to the transmitter notifying a corrupted data has been received and request to retransmit the same TLP. From this end-to-end data reliability is ensured.

2.2 DATA-LINK LAYER[1]

At Data-link layer the data packet is called Data-Link Layer Packet (DLLP). Important component present in this layer is 32-LCRC block.

32-bit LCRC (Link Cyclic Redundancy Check) block

At transmitter, for every 8-bit data a 32-bit LCRC value is computed and stored in CRC field of Transaction Layer Packet (TLP).

Again at receiver a 32-bit ECRC value is computed for each received TLP. The received LCRC value is compared with LCRC value transmitted. If they are equal, then an ACK[5] (Acknowledge) signal is sent to the transmitter notifying data has been received with no corruption; else a NACK[5] (No Acknowledge) signal is sent to the transmitter notifying a corrupted data has been received and request to retransmit the same TLP. From this, whether an error is inserted at the physical link or not can be verified and data security can be ensured.

2.3 PHYSICAL LAYER[1]

The following components are present in physical layer.

- **Scrambler and Descrambler**

  At transmitter, the input 8-bit data is scrambled with the help of LFSR and a scrambled data is produced. Similarly, at receiver, the received data is descrambled to retrieve original DLLP.

- **8b/10b Encoder and 10b/8b Decoder**

  At transmitter, input data is encoded using 8b/10b Encoder. At receiver, received data is decoded using 10b/8b Decoding mechanism.

- **Parity Generator and Checker**

  At transmitter, the parity of encoded data is found and appended with the data itself, to make data always possess even parity. At receiver, the parity of received data is found and it is checked for even parity. If the received data has even parity, then the data is sent for data-link layer for further processing; else, phy_err signal goes high indicating data has been corrupted and discard the data and request sender to retransmit the data.

- **Serializer and Deserializer (SERDES)[4]**

  At transmitter end the parallel data needs to be serialized for speedy transmission through the link. Hence a PISO (Parallel In Serial Out) converter is used to convert parallel data to serial data.

  At receiver end the serial data needs to be converted to parallel data for further processing at receiver. Hence a SIPO (Serial In Parallel Out) converter is used to convert serial data from the physical link to parallel data.

  All the components are integrated at transmitter and receiver to obtain the top-level module of PCIe 3.0. Block diagram representing the top module of PCIe 3.0 is shown in Figure 8.

3. RESULTS AND INFERENCES

The design of PCIe 3.0 was carried out in Verilog HDL. As mentioned in section 2 a modular architecture was incorporated with layering. Verilog version 2001 was used for developing the source code.

3.1 MODULE WISE SIMULATION RESULTS OF PCIe 3.0

This section consists of the simulation results of each of the module present in design hierarchy as explained in section 2.

- **32-bit ECRC module**

  It takes 32-bit data as input and computes a 32-bit ECRC (End-to-End Cyclic Redundancy Check) value as indicated in Figure 9. For example in the Figure 10 input to the ECRC block is $(fofofofo)_{16}$ and the generated ECRC value is $(6b6ec559)_{16}$.

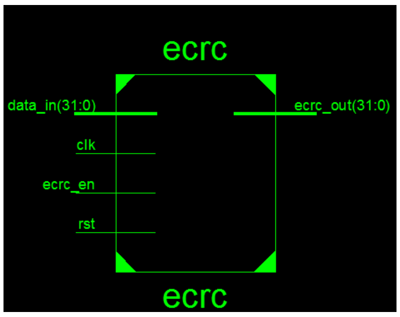


Fig -9: RTL schematic of 32-bit ECRC computation

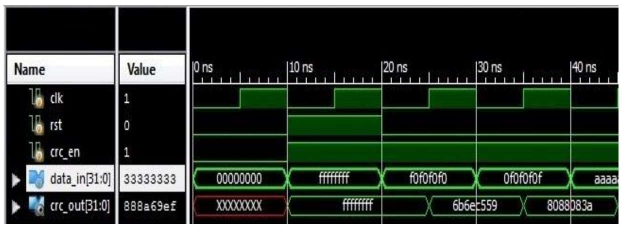


Fig -10: Simulation waveform of 32-bit ECRC computation

- **32-bit LCRC module**

It takes 32-bit data as input and computes a 32-bit LCRC (Link Cyclic Redundancy Check) value as indicated in Figure 11. For example in the Figure 12 input to the LCRC block is $(00000003)_{16}$ and the generated LCRC value is $(000000c0)_{16}$.

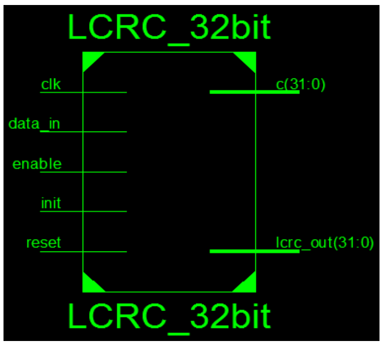


Fig -11: RTL Schematic of 32-bit LCRC

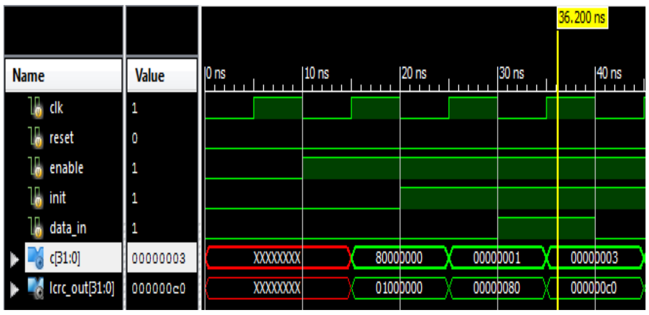


Fig -12: Simulation waveform of 32-bit LCRC computation

- **8 bit Scrambler and Descrambler**

Scrambler and Descrambler take 8 bit data as input and produce an 8-bit scrambled output as shown in Figure 13. In the waveform indicated in Figure 14, input is $(01010101)_2$ and the scrambled value is $(11100111)_2$. Similarly Figure 15 indicates the simulation waveform of Descrambler module.

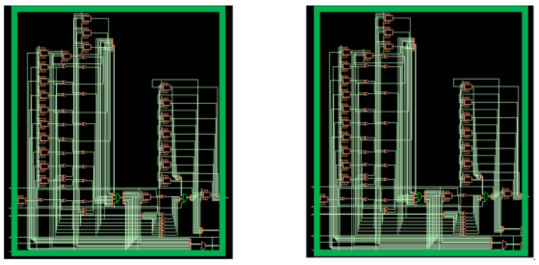


Fig -13: RTL Schematic of 8-bit Scrambler (Left) and Descrambler (Right)

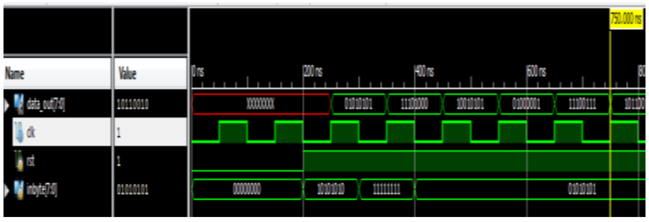


Fig -14: Simulation waveform of 8-bit Scrambler

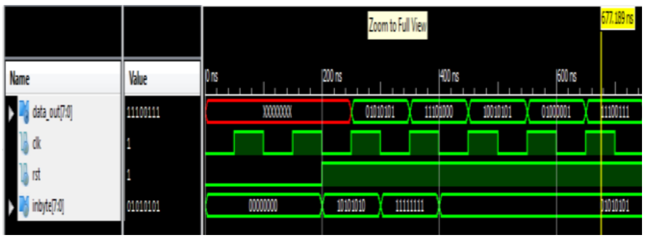


Fig -15: Simulation waveform of 8-bit Descrambler

- **8b/10b Encoder**

Figure 16 and Figure 17 indicates RTL schematic and output waveform of 8b/10b encoder respectively. An 8 bit data $(000)_{16}$ is encoded as $(0B9)_{16}$ as shown.

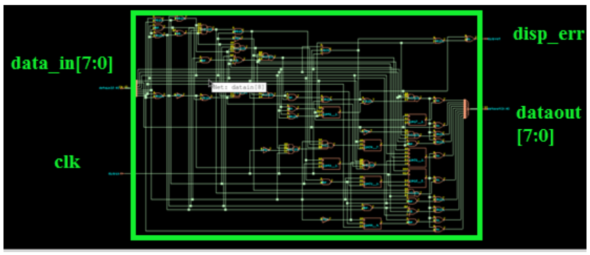


Fig -16: RTL Schematic of 8b/10b Encoder

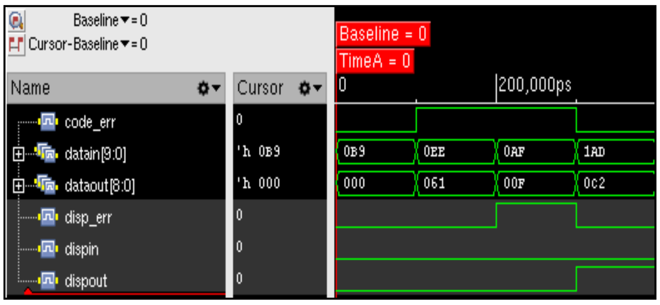


Fig -17: Simulation waveform of 8b/10b Encoder

- **10b/8b Decoder**

Figure 18 and Figure 19 indicates RTL schematic and output waveform of 10b/8b decoder. A 10 bit data $(0B9)_{16}$ is decoded as $(000)_{16}$ as shown.

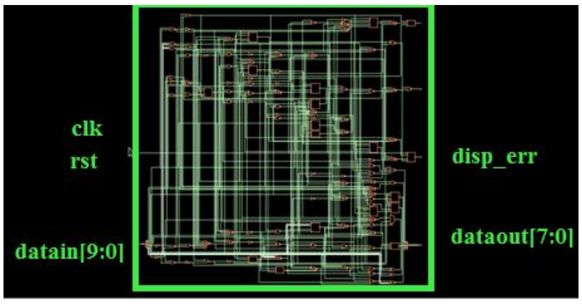


Fig -18: RTL Schematic of 10b/8b Decoder

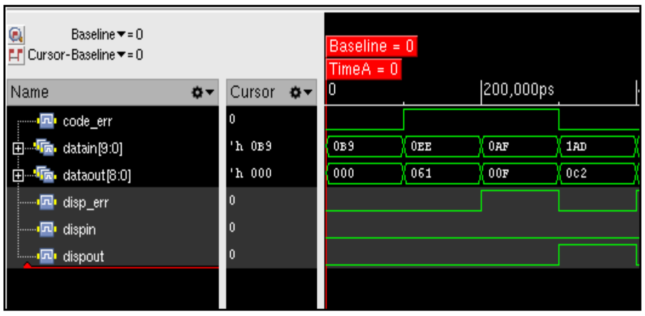


Fig -19: Simulation waveform of 10b/8b Decoder

- **Parity Generator and Checker**

Figure 20 and Figure 21 indicates RTL schematic and output waveform of Parity generator and checker block respectively. An 8 bit data $(11100101)_2$ has an odd parity which is indicated by parity_bit as shown. As parity bit is high, which indicates the data is of odd parity and hence reject data is high as shown.

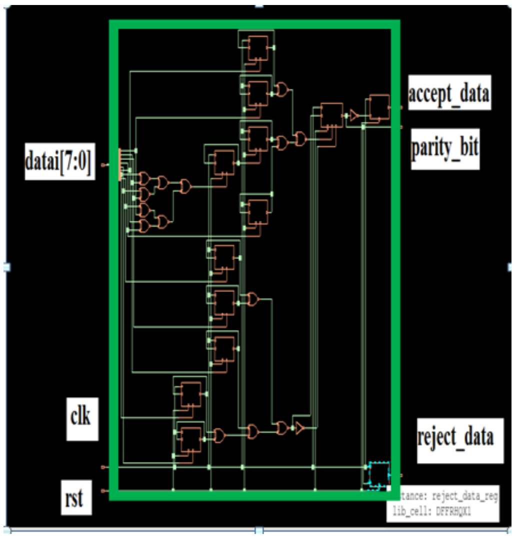


Fig -20: RTL Schematic of Parity generator and checker

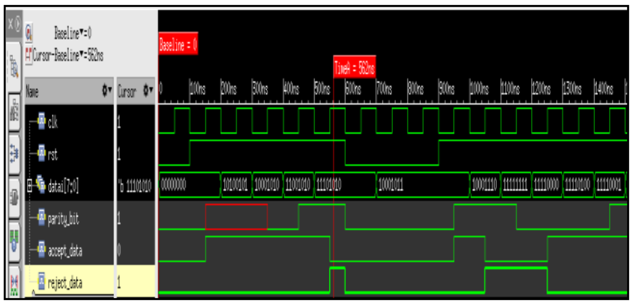


Fig -21: Simulation waveform of Parity generator and checker

- **16 bit Serializer**

Figure 22 and Figure 23 indicates RTL schematic and output waveform of 16 bit serializer respectively. It converts parallel data into serial data as shown.

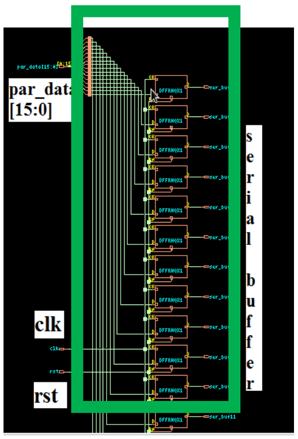


Fig -22: RTL Schematic of Serializer

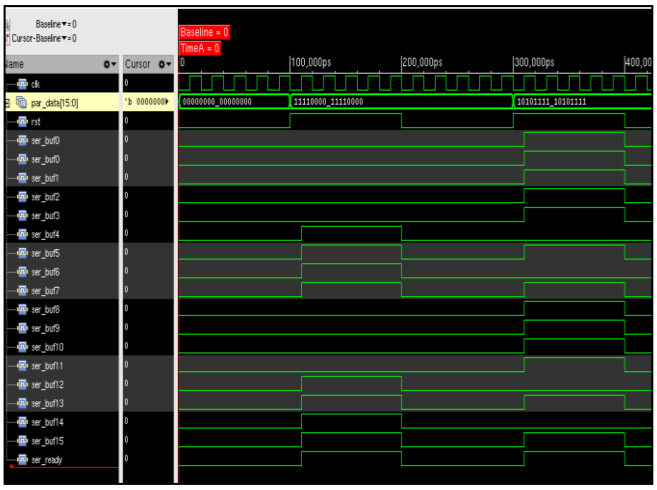


Fig -23: Simulation waveform of 16 bit Serializer

- **16 bit Deserializer**

  Figure 24 and Figure 25 indicates RTL schematic and output waveform of 16 bit Deserializer respectively. It converts serial data into parallel data as shown.

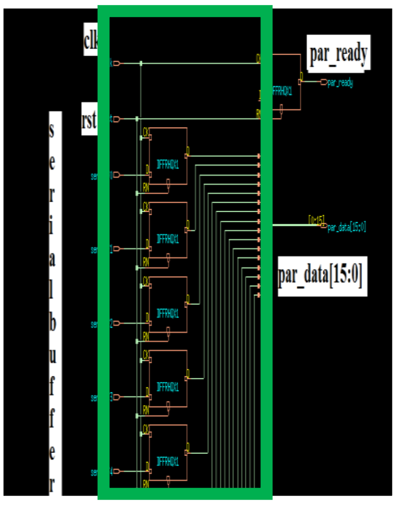


Fig -24: RTL Schematic of Deserializer

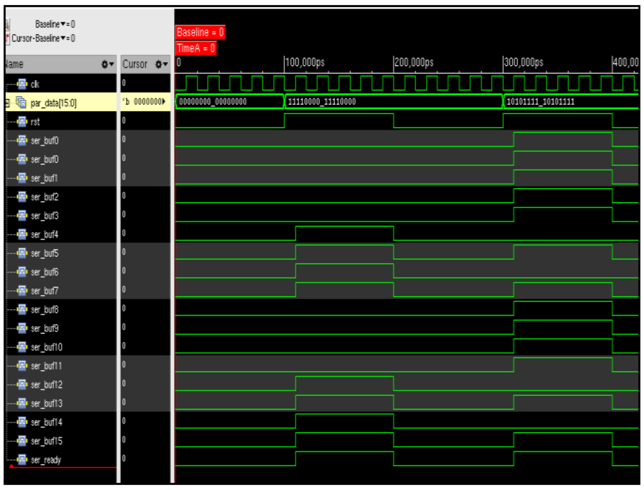


Fig -25: Simulation waveform of 16 bit Deserializer

3.2 RTL SIMULATION RESULTS OF INTEGRATED DESIGN OF PCIe 3.0 USING QUESTASIM

In this project QuestaSim[6] Tool has been used to simulate the design using a testbench and verify the same through code coverage. Figure 26 represents simulation waveform of design PCIe 3.0. It signifies the flow of data through 32-bit data input and output ports along with 8-bit header (FA(base-16) for Resource Request and AF(base-16) for Resource Completion) and trailer frames (77(base-16)) (FA, AF and 77 are used only for illustration purpose as shown in Table 1). An I/O delay of 165ns was inferred from the waveform.

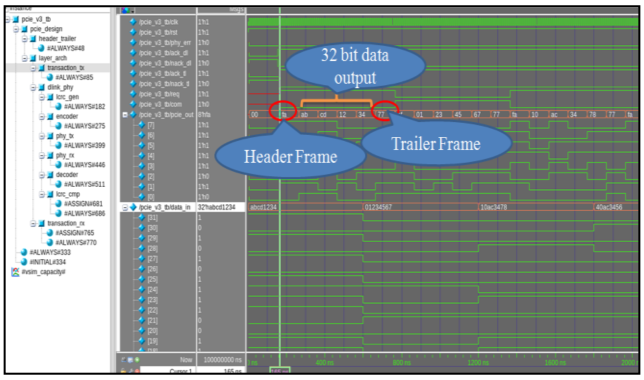
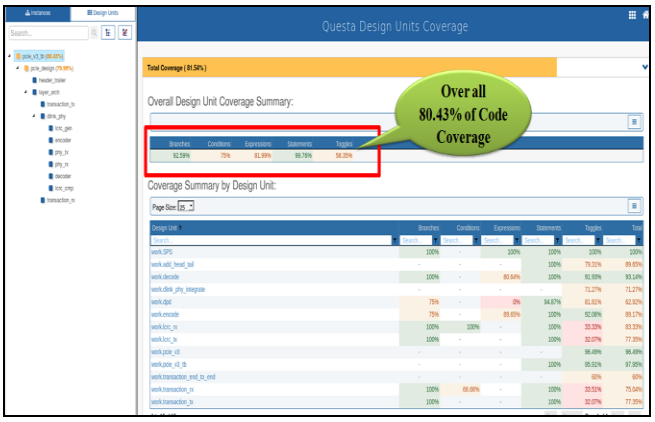


Fig -26: Simulation waveform of PCIe 3.0

The testbench was able to cover 80.43% of entire Verilog module of PCIe 3.0 as indicated by the coverage analysis report shown in Figure 27.



Questa Design Units Coverage

Total Coverage (81.54%)

Overall Design Unit Coverage Summary:

Branches	Conditions	Expressions	Statements	Toggles
92.59%	75%	81.99%	99.76%	58.35%

Coverage Summary by Design Unit:

Page Size: 25

Design Unit	Branches	Conditions	Expressions	Statements	Toggles	Total
work.SPS	100%	-	100%	100%	100%	100%
work.add_head_tail	-	-	-	100%	79.31%	89.65%
work.decode	100%	-	80.64%	100%	91.93%	93.14%
work.dlink_phy_integrate	-	-	-	-	71.27%	71.27%
work.dpd	75%	-	0%	94.87%	81.81%	62.92%
work.encode	75%	-	89.65%	100%	92.06%	89.17%
work.lcrc_rx	100%	100%	-	100%	33.33%	83.33%
work.lcrc_tx	100%	-	-	100%	32.07%	77.35%
work.pcie_v3	-	-	-	-	96.49%	96.49%
work.pcie_v3_tb	-	-	-	100%	95.91%	97.95%
work.transaction_end_to_end	-	-	-	-	60%	60%
work.transaction_rx	100%	66.66%	-	100%	33.51%	75.04%
work.transaction_tx	100%	-	-	100%	32.07%	77.35%

Fig -27: Overall Coverage report after modifying the test bench

3.3 SYNTHESIS RESULTS USING PRECISION RTL PLUS

Precision RTL Plus [7] tool was used for synthesizing PCIe 3.0 design. The FPGA device 6SLX16CSG324 belonging to Xilinx vendor, Spartan-6 family with speed grade -3 was selected for the PCIe 3.0 design implementation. Synthesis was carried out with a device clock frequency of 100 MHz and 5ns I/O delay (As per the specifications listed in Table 1).

RTL Schematic and Technology Schematic of the PCIe 3.0 are represented by Figure 28 and Figure 29 respectively as shown. The D Flip flop present at the output terminal ecrc_tx of RTL schematic shown in Figure 28 has been mapped to a technology specific entity named FDE by the synthesis tool as shown in Figure 29.

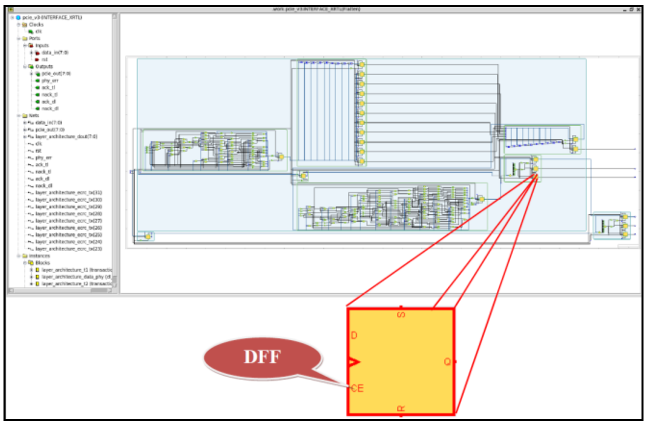


Fig -28: RTL Schematic of PCIe 3.0

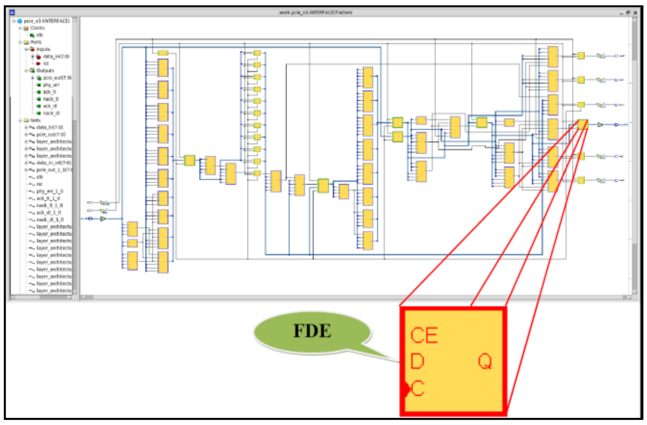


Fig -29: Technology Schematic of PCIe 3.0

Synthesis tool will also dump the timing and area reports. The summary of synthesis reports is tabulated in Table 4 as shown below.

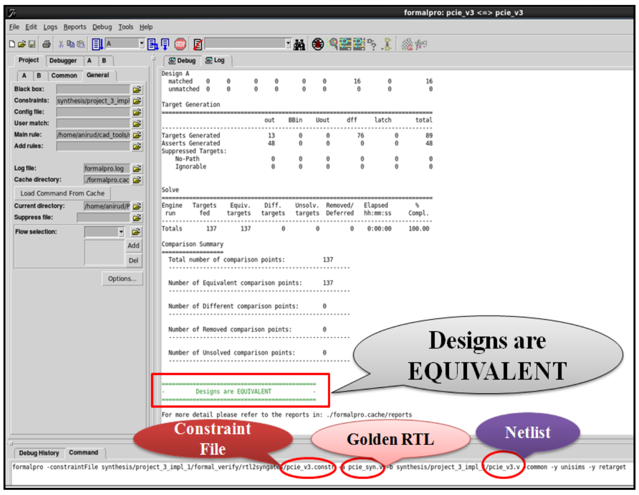
Table -4: Summary of Synthesis Reports

Synthesis Reports	Output Values	
Timing	Cell delay	79.42%
	Net delay	20.58%
	Slack	0.538 ns
Area	Total number of DFFs	68
	Total number of LUTs	53
	Total number of accumulated Instances	131

3.4 EQUIVALENCE-CHECK RESULT USING FORMALPRO

FormalPro[8] tool has been used to check the functional equivalence between RTL and gate-level netlist. Design A is chosen as RTL and Design B is chosen as gate-level netlist. Common section is loaded with FPGA reference libraries (same as that of used during synthesis).

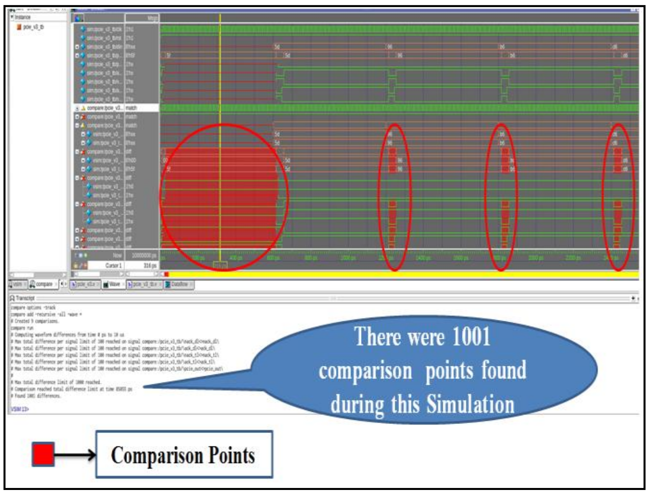
The synthesis tool will dump a constraint file as an output. This constraint file is fed to FormalPro as input along with RTL and gate-level netlist. After performing Equivalence Check both the designs were found to be equivalent as shown in Figure 30.



```
Design A
  matched    0    0     0    0     0    0      16     0      16
  unmatched  0    0     0    0     0    0       0     0       0

Target Generation
================================================================================
                              out    BBin    Uout     dff     latch      total
--------------------------------------------------------------------------------
Targets Generated              13      0       0       76       0          89
Asserts Generated              48      0       0        0       0          48
Suppressed Targets:
    No-Path                     0      0       0        0       0           0
    Ignorable                   0      0       0        0       0           0


Solve
================================================================================
Engine   Targets    Equiv.    Diff.    Unsolv.   Removed/  Elapsed       %
 run       fed     targets   targets   targets   Deferred  hh:mm:ss    Compl.
--------------------------------------------------------------------------------
Totals      137       137        0         0          0     0:00:00     100.00

Comparison Summary
==================
  Total number of comparison points:         137
  -------------------------------------------------------

  Number of Equivalent comparison points:    137
  -------------------------------------------------------

  Number of Different comparison points:     0
  -------------------------------------------------------

  Number of Removed comparison points:       0
  -------------------------------------------------------

  Number of Unsolved comparison points:      0
  -------------------------------------------------------

==============================================
-        Designs are EQUIVALENT              -
==============================================
For more detail please refer to the reports in: ./formalpro.cache/reports
```

```
formalpro -constraintFile synthesis/project_3_impl_1/formal_verify/rtl2syngates/pcie_v3.constr -a pcie_syn.v -b synthesis/project_3_impl_1/pcie_v3.v common -y unisims -y retarget
```

Fig -30: Formal Equivalence Check result after constraining the design


There were 1001 comparison points found during this Simulation
Comparison Points
Transcript
Wave
Dataflow
compare

```
compare options -track
compare add -recursive -all -wave *
# Created 9 comparisons.
compare run
# Computing waveform differences from time 0 ps to 10 us.
# Max total difference per signal limit of 100 reached on signal compare:/pcie_v3_tb/\nack_dl<>nack_dl\
# Max total difference per signal limit of 100 reached on signal compare:/pcie_v3_tb/\ack_dl<>ack_dl\
# Max total difference per signal limit of 100 reached on signal compare:/pcie_v3_tb/\nack_tl<>nack_tl\
# Max total difference per signal limit of 100 reached on signal compare:/pcie_v3_tb/\ack_tl<>ack_tl\
# Max total difference per signal limit of 100 reached on signal compare:/pcie_v3_tb/\pcie_out<>pcie_out\
#
# Max total difference limit of 1000 reached.
# Comparison reached total difference limit at time 85855 ps
# Found 1001 differences.

VSIM 13>
```

Fig -32: Output of Comparison between RTL and Gate-Level Simulation Results

3.5 GATE-LEVEL SIMULATION RESULT USING QUESTA SIMULATOR

Again Questa Simulator [6] was used for performing gate-level simulation. Here instead of RTL, gate-level netlist was simulated using testbench. The output waveform of PCIe 3.0 gate-level netlist is as shown in Figure 31.

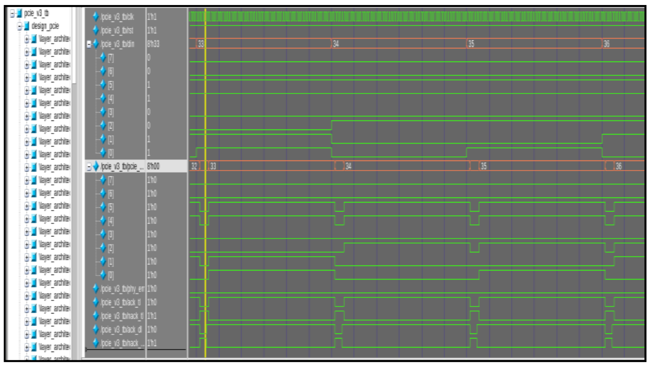
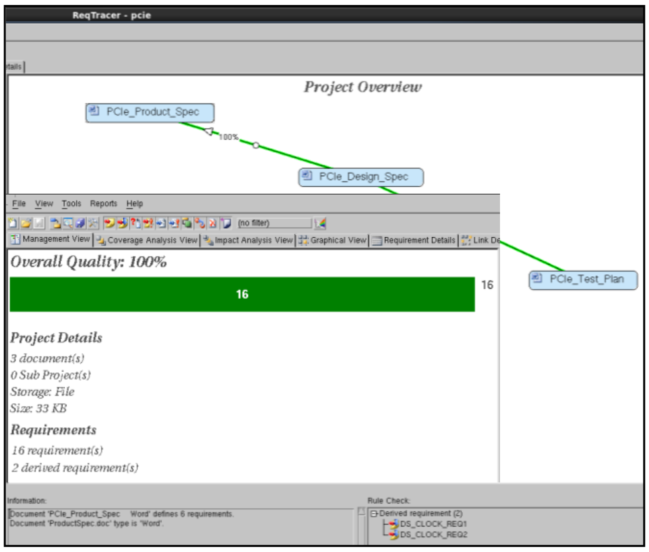


Fig -31: Simulation waveform of gate-level netlist

Comparison between RTL simulation waveform and gate-level simulation waveform was carried out with the help of Questa Simulator. Both the waveforms were fed to the tool and a waveform showing comparison points was generated as shown in Figure 32. It is inferred from the waveform that, there were 1001 comparison points in total.

3.6 REQUIREMENT TRACEABILITY ANALYSIS RESULTS USING REQTRACER

Design specification, Product specification and Test plan related to PCIe 3.0 were written in a word document(.doc) and kept in a directory. These three documents were fed to ReqTracer[9] as inputs. A logical link was created between Test plan and Design specification, and Design specification and Product specification. Traceability analysis showed 100 percent mapping among all the three as shown in Figure 33. Figure 34 represents graphical overview of the mapping among all three files.



Overall Quality: 100%

16 16

Project Details
3 document(s)
0 Sub Project(s)
Storage: File
Size: 33 KB

Requirements
16 requirement(s)
2 derived requirement(s)

Document 'PCIe_Product_Spec' 'Word' defines 6 requirements.
Document 'ProductSpec.doc' type is 'Word'.

Derived requirement (2)
DS_CLOCK_REQ1
DS_CLOCK_REQ2

Fig -33: 100 percent mapping among Design specification, Product specification and Test plan

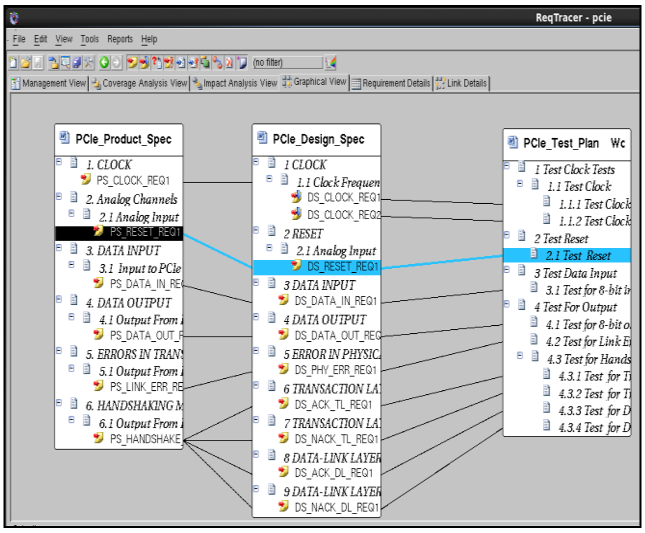


Fig -34: Graphical View of ReqTracer output

3.7 DO-254 STANDARD RULE CHECK RESULTS USING HDS

HDL Designer Series (HDS) [10] tool was used to cleanup the Verilog source code of PCIe 3.0 design. As per the DO-254 standard rule sets mentioned in Table 3. The verilog code was modified without hampering the functionality of the design. Figure 35 represents HDS environment where design file is loaded and tested for DO-254 Standard.

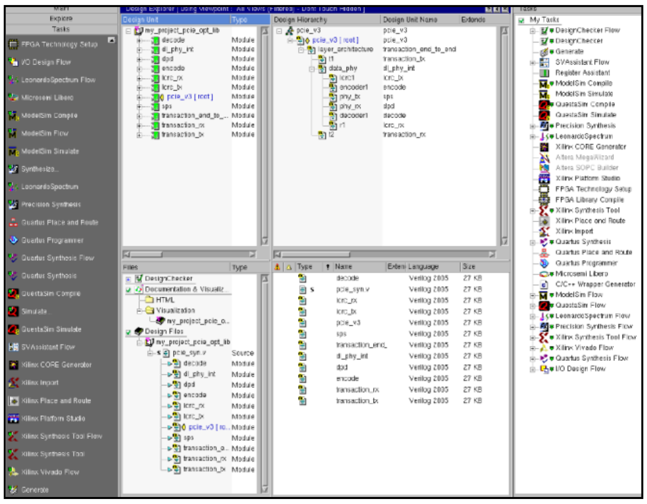
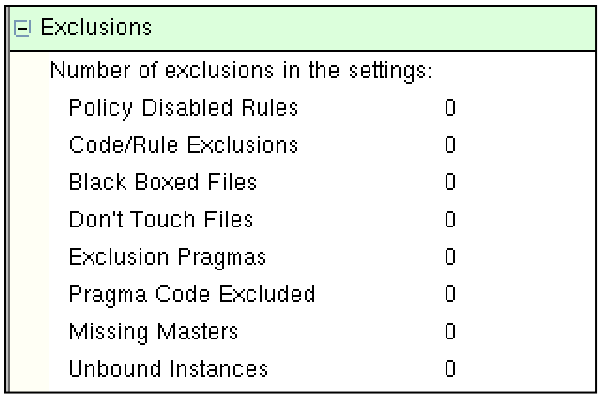


Fig -35: HDS Project Environment along with Design Checker

Design Checker application is supported by HDS which checks and validates DO-254 standard for the given design. The design checker gave a 98 percent pass result for the developed Verilog code of PCIe 3.0 as shown in Figure 36.

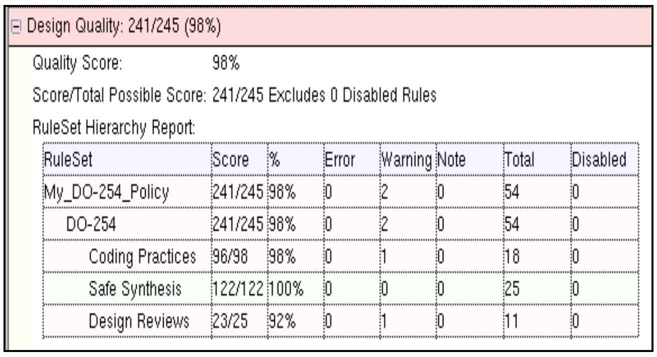


Exclusions

Number of exclusions in the settings:

Policy Disabled Rules	0
Code/Rule Exclusions	0
Black Boxed Files	0
Don't Touch Files	0
Exclusion Pragmas	0
Pragma Code Excluded	0
Missing Masters	0
Unbound Instances	0

Fig -36(a): Exclusion parameters - DO-254 Rule check



Design Quality: 241/245 (98%)

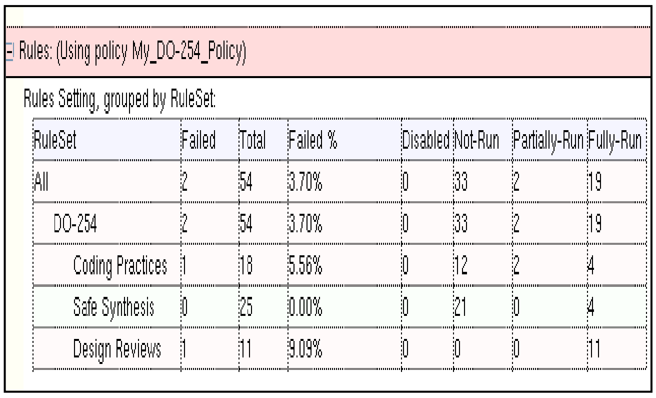
Quality Score: 98%

Score/Total Possible Score: 241/245 Excludes 0 Disabled Rules

RuleSet Hierarchy Report:

RuleSet	Score	%	Error	Warning	Note	Total	Disabled
My_DO-254_Policy	241/245	98%	0	2	0	54	0
DO-254	241/245	98%	0	2	0	54	0
Coding Practices	96/98	98%	0	1	0	18	0
Safe Synthesis	122/122	100%	0	0	0	25	0
Design Reviews	23/25	92%	0	1	0	11	0

Fig -36(b): Design Quality under DO-254 Rule check

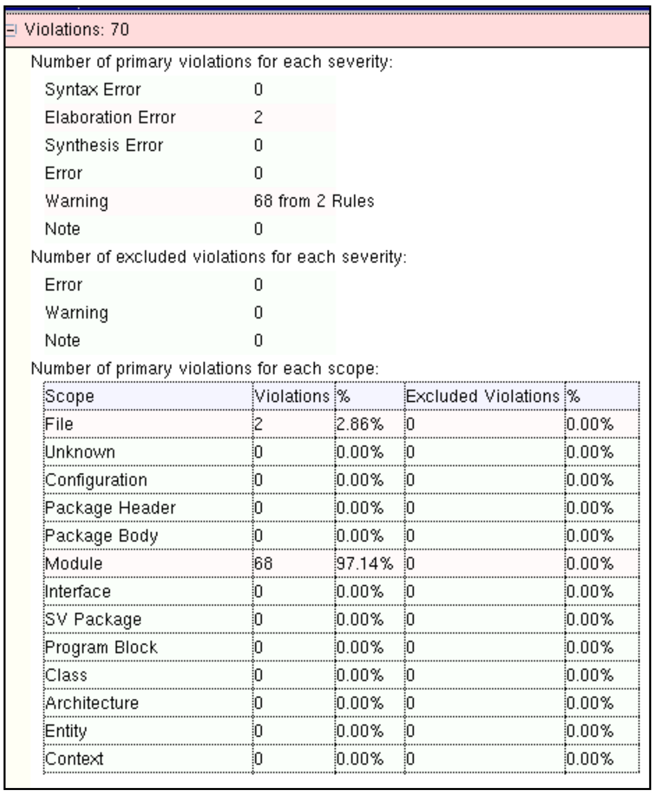


Rules: (Using policy My_DO-254_Policy)

Rules Setting, grouped by RuleSet:

RuleSet	Failed	Total	Failed %	Disabled	Not-Run	Partially-Run	Fully-Run
All	2	54	3.70%	0	33	2	19
DO-254	2	54	3.70%	0	33	2	19
Coding Practices	1	18	5.56%	0	12	2	4
Safe Synthesis	0	25	0.00%	0	21	0	4
Design Reviews	1	11	9.09%	0	0	0	11

Fig -36(c): DO-254 Rule check



⊟ Violations: 70

Number of primary violations for each severity:

Syntax Error	0
Elaboration Error	2
Synthesis Error	0
Error	0
Warning	68 from 2 Rules
Note	0

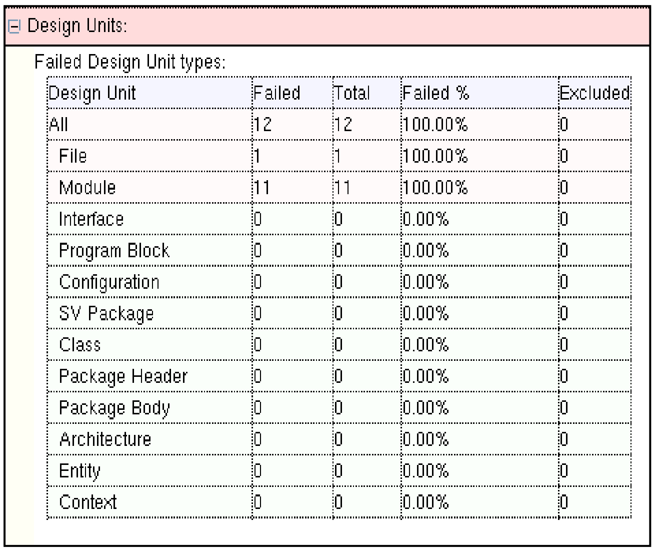
Number of excluded violations for each severity:

Error	0
Warning	0
Note	0

Number of primary violations for each scope:

Scope	Violations	%	Excluded Violations	%
File	2	2.86%	0	0.00%
Unknown	0	0.00%	0	0.00%
Configuration	0	0.00%	0	0.00%
Package Header	0	0.00%	0	0.00%
Package Body	0	0.00%	0	0.00%
Module	68	97.14%	0	0.00%
Interface	0	0.00%	0	0.00%
SV Package	0	0.00%	0	0.00%
Program Block	0	0.00%	0	0.00%
Class	0	0.00%	0	0.00%
Architecture	0	0.00%	0	0.00%
Entity	0	0.00%	0	0.00%
Context	0	0.00%	0	0.00%

Fig -36(d): DO-254 Rule Check Violations



⊟ Design Units:

Failed Design Unit types:

Design Unit	Failed	Total	Failed %	Excluded
All	12	12	100.00%	0
File	1	1	100.00%	0
Module	11	11	100.00%	0
Interface	0	0	0.00%	0
Program Block	0	0	0.00%	0
Configuration	0	0	0.00%	0
SV Package	0	0	0.00%	0
Class	0	0	0.00%	0
Package Header	0	0	0.00%	0
Package Body	0	0	0.00%	0
Architecture	0	0	0.00%	0
Entity	0	0	0.00%	0
Context	0	0	0.00%	0

Fig -36(e): Failed design units - DO-254 Rule check

After this the DO-254 Standard validated code is simulated using Questa simulator, synthesized using Precision RTL. All the tools were invoked within HDS. Figure 37 represents simulation waveform, Figure 38 and Figure 39 represents RTL schematic and Technology schematic after synthesis respectively. Table 5 gives a summary of synthesis reports.

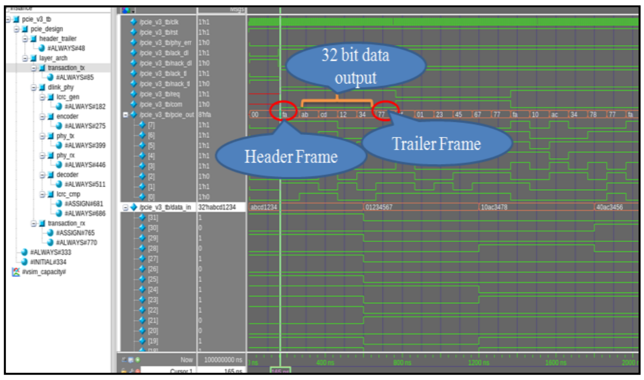


Fig -37: Simulation waveform of DO-254 Standard PCIe 3.0

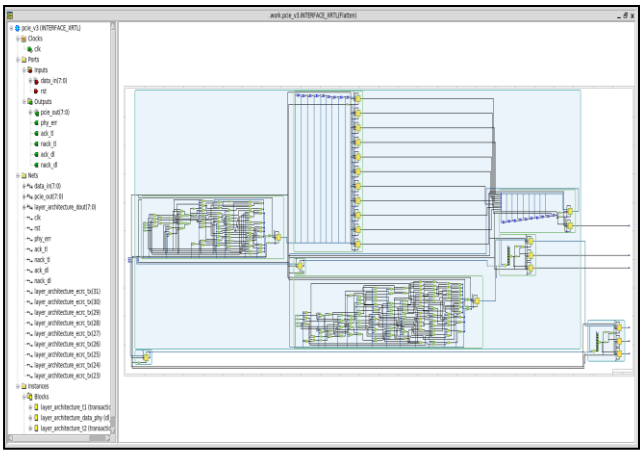


Fig -38: RTL Schematic of DO-254 Standard PCIe 3.0

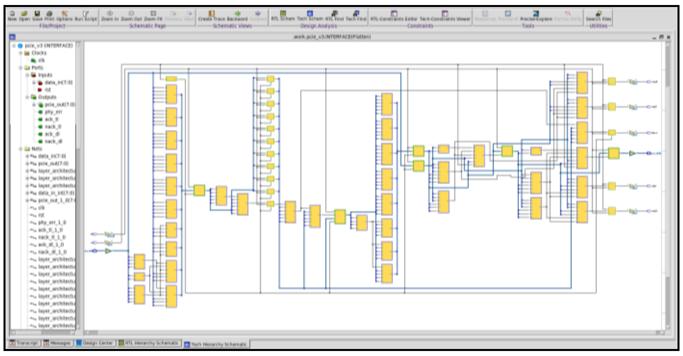


Fig -39: Technology Schematic of DO-254 Standard PCIe 3.0

Table -5: Synthesis Reports summary of DO-254 standard design

Synthesis Reports	Output Values	
Timing	Cell delay	79.42%
	Net delay	20.58%
	Slack	0.538 ns
Area	Total number of DFFs	68
	Total number of LUTs	53
	Total number of accumulated Instances	131

4. CONCLUSION

PCIe 3.0 was designed, verified and synthesized successfully with technology library and FPGA as per the standard protocol. Requirement tracing of project components such as Design specification, Product specification and Test plan was achieved. PCIe 3.0 design has validated all the rule sets present in DO-254 Standard. The soft IP of PCIe 3.0 thus developed can be taken for physical design.

5. FUTURE SCOPE

Soft IP of PCI e 3.0 can be taken through Physical Design and Physical Verification. A hard IP can then be integrated with many platforms and applications.

REFERENCES

[1] "PCI Express® Base Specification Revision 3.0", by PCI-SIG, November 2010.

[2] Hiroki Nakamura, Hirotaka Takayama, Yoshiki Yamaguchi and Taisuke Boku, "Thorough analysis of PCIe Gen3 Communication", (IEEE).

[3] Shoeb Mohammed Balabatti, Radha R, April 2014, "Calculation of ECRC FOR TLP-Packets of PCIe Protocol", (IJAREEIE).

[4] Chandana K N, Karunavathi R K, 2015, "Link Initialization and Training in MAC Layer of PCIe 3.0", (IJCSIT).

[5] Gokulakrishnan, Radhakrishnan, December 2016, "Design and Verification of ACK/ NAK Protocol of PCI Express Data Link Layer in System Verilog", (IJRASET).

[6] The Questa Verification Solution by Mentor Graphics

http://www.edmd-solutions.com/wp-content/uploads/2017/12/Questa_Datasheet.pdf

[7] Precision RTL Plus by Mentor Graphics

http://www.europractice.stfc.ac.uk/vendors/mg_rtl_plus_datasheet.pdf

[8] FormalPro by Mentor Graphics

http://www.europractice.stfc.ac.uk/vendors/mg_FormalPro_DS.pdf

[9] ReqTracer by Mentor Graphics

http://www.europractice.stfc.ac.uk/vendors/mg_reqtracer_datasheet.pdf

[10] HDL Designer Series by Mentor Graphics

https://hornad.fei.tuke.sk/predmety/ncs/FPGA_Advantage_Documentation/hds_user.pdf

[11] DO-254 Explained by Cadence

https://www.cadence.com/content/dam/cadence-www/global/en_US/documents/solutions/aerospace-and-defense/do-254-explained-wp.pdf

[12] Graeme Jessiman, May 2016, DO-254 Coding Checks for RTL Code

https://nmi.org.uk/wp-content/uploads/2016/05/Mentor-Graphics_NMI_UK_Seminar_GraemeJ_May2016.pdf