**Data Analytics and Artificial Intelligence**
**Vol: 3(6), 2023**
**REST Publisher; ISBN: 978-81-948459-4-2**
**Website: http://restpublisher.com/book-series/daai/**

# 10 GE MAC CORE Verification Driver-Module

***S.P.Velmurugan, Govardhan Bandi, Tejaswee Avulapati, Venkata sai Dakshinapu, Umesh Reddy Venkatathathagari**
Kalasalingam Academy of Research and Education, Krishnankoil,India
Corresponding Author Email: s.p.velmurugan@klu.ac.in

***Abstract.***The scope of this project is to build a UVM testbench – driver module by following the Objected Oriented Programming (OOP) paradigm. Under the OOP paradigm, the testbench is comprised of different objects which are designed to handle one particular task of the verification flow. The design under verification is a 10GE MAC Core. Even though the design is distributed with a rudimentary test case that allows to run some basic sanity check, such testbench would be extremely difficult to scale and/or maintain. Hence main theme of this project is providing verification environment of driver module which is easy to understand. It will be helpful while dealing with verifications of complex digital systems such as the ones that are found today in any application.

***Keywords:*** Ethernet, driver, 10 GE Mac core, OOPs concept, UVM

## 1. INTRODUCTION

According to IEEE 802.3, a data frame is made up of seven fields which are different. The fields are combined to form a single data frame. It has 7 fields they are: Preamble, start-of-frame delimiter, destination address, source address, length, data and frame check sequence which are shown in below table 1.

**TABLE 1.**Ethernet frame

| PRE | SOF | DA | SA | LENGTH | DATA | CRC |
|---|---|---|---|---|---|---|
| 7 | 1 | 6 | 6 | 2 | 46-1500 | 4 |

The DUV is intentionally designed so that it can be easily integrated with custom logic. It is also designed with a limited feature set to guarantee a small gate footprint. Even though the design features both transmit and receive XGMII (10 Gigabit Media Independent Interface) interfaces, this verification plan will not make any attempt to verify the correctness of these interfaces. The DUV will be connected in loopback mode, i.e., the xgmii_rx_* interfaces will be directly connected to the corresponding xgmii_tx_* interfaces.The project will delve into the specifics of each one of the testbench building blocks and will provide details on how they fit into the verification flow. A top level testbench diagram illustrates the existing relationships among these components and shows the mechanism that is used in order to translate testbench stimulus into RTL signals. In this project the driver module has to be created which drives stimuli to the DUV.There are different test cases that have to be created and constrained randomization is used in order to generate different types of stimuli that are meant to exercise different logic inside the DUV.

## 2. LITERATUTE SURVEY

In this article, we examine major elements of the IEEE 802.3av Standards, including the task force's debate, and present the most recent studies on the development of 10G-EPON systems [1].In this study, the authors offer a thorough examination of the UVM overhead and its main causes. We also examine the practical effects of NVIDIA's prefetching and oversubscription on various workloads and tie performance to the prefetching technique and driver implementation [2].A functional coverage model is constructed in this research to determine whether or not the verification achieves the desired coverage. System Verilog is used for the coding, and Questa sim is used for the simulation. [3]This study examines the Universal Verification Methodology (UVM) by looking at how it was used to create two testbeds for unit verification. The first one uses all the fundamental UVM components and is directed at a First Input-First Output (FIFO) buffer module. [4]The synchronous FIFO subsystem is used as a test case in this research to investigate the utilization of UVM in test bench creation. Almost all SoCs contain FIFOs as a fundamental component. In data applications, they are frequently

employed as buffers, queues, flow controllers, etc. For DUV verification, the Design Under Verification is put to the test and coverage models are used. [5]

## 3. PROPOSED SYSTEM

***Design of 10 GE MAC core:***The 10GE MAC core is made to integrate quickly with exclusive custom logic. It has a management interface that complies with Wishbone standards and a data path interface that is similar to POS-L3. The core's constrained feature set and tiny gate footprint were design goals.

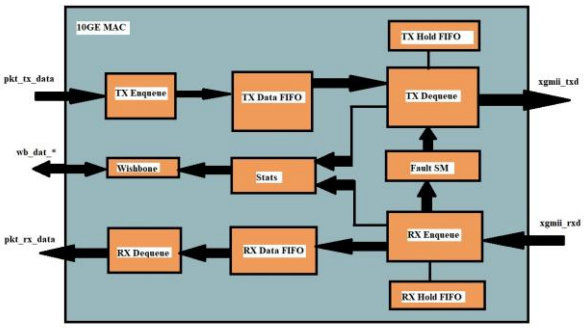


**FIGURE 1.**10GE MAC Block Diagram

Frames from the user's core logic are received by the TX Enqueue Engine, it saves in FIFO. Moreover, it gives the core the FIFO fill status.By default, the TX FIFO and RX FIFO comprises 128 entries with a total of 64 bits of data and 8 bits of status. Due to the FIFO's limited capacity of 512 bytes (64 64-bit), the MAC must operate in flow-through mode. TX Dequeue Engine uses the status information in bits 71–64 of the FIFO at the top to keep the frames aligned.Two state machines are present in the TX Dequeue Engine. Except for the last word of the frame, the CRC logic operates on 64-bit data. At the end of the frame, the CRC logic switches to 8-bit mode because the final word has 1 to 8 valid bytes.Holding FIFO's 8-word delay results from the possibility that the calculation could take up to 8 cycles to complete.The Encoding State-Machine then gets data from the Holding FIFO after adding the Ethernet preamble. The state machine produces flags that indicate when 32-bit is necessary in order to accomplish the necessary IFG and sets entire data on 64-bit borders in order to decrease logic. IFG is calculated using the current frame and the cumulative Deficit Idle Count (DIC). The Barrel Shifter receives flags, which completes the 32-bit alignment.When a local problem is discovered, the RC Layer inserts remote fault messages while monitoring link status signals from the Fault State-Machine. Moreover, the Encoding State-Machine receives the fault signal and pauses transmitting packets when there is a fault.In the event of a problem at the XGMII interface, the RC layer in the RX Enqueue Engine notifies the Fault State Machine and reports it. The decoding state machine has a tough time defining boundaries between frames and recognising incorrect frames, such as fragments and runts. When it decodes the data, a copy is written to the holding FIFO and CRC logic. The major purpose of the holding FIFO is to compensate for the slowness of the CRC computation. The CRC logic works with 64-bit data, with the exception of the final word of the frame. The CRC logic changes to 8-bit mode at the conclusion of the frame since the final word may contain 1 to 8 bytes. The Holding FIFO's 8-word delay is due to the fact that the computation might take up to 8 cycles to finish. Yet, the next FIFO item has an error flag set.SOP, EOP, and other flags are written to the RX Data FIFO together with the data as it exits the Holding FIFO. CRC logic indicates the error when an error flag is put in the FIFO.The RX Enqueue Engine's fault messages are recorded and monitored by the Fault State-Machine. Its function is to transition the fault state in accordance with the 802.3ae specification and debounce fault indications. A local fault or a remote fault can be declared by the state machine. You can keep track of both fault types by reading the interrupt status register.The Stats Engines collect various packet and byte counters and keep track of frame indication signals from the TX Dequeue and RX Enqueue Engines. Moreover, it is used to move between the faster clock domains and the slower processor interface clock domain.A deep dive into the 10 GE MAC core is not necessary because it's a complex design. Understanding the terms and functions of terms included in architecture is enough for writing the test bench.

***Testbench:***The Universal Verification Methodology (UVM), a verification approach based on the system Verilog language, is implemented into a test bench for evaluating complicated systems.The UVM testbench architecture serves as a guide for creating testbench scripts for designs that include testbench elements such as Top, Test, Environment, Driver, Scoreboard, and Monitor, among others. Each component performs a particular role, such as driving the signals to the DUV in the case of the driver.

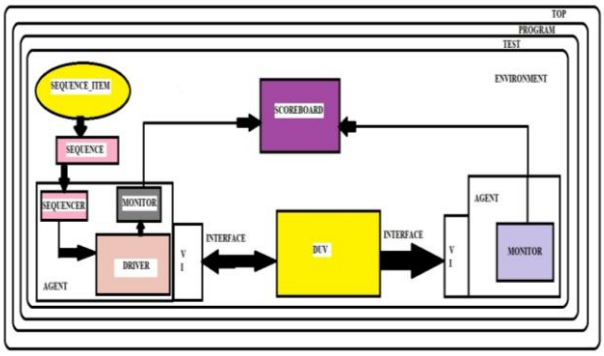


**FIGURE 2**. UVM testbench architecture

The sequence item class is derived from the base class uvm_sequence_item. First, Ethernet properties should be declared as random variables with suitable constraints (rand) and we will register them with the factory. Write constraints according to the ethernet properties. Define a constructor for the data item. Here the data is generated, which will go into sequence.Sequence is an object that contains a behavior for generating stimulus. Sequence is extended from uvm_sequence. Sequence sends transactions to the driver using a sequencer. Ethernet functional code resides in the body() task, which includes the creation of req using uvm_create and the start() method. Sequence executes when its start() method is called, and the start() method calls the body() method. We have generated 10 data packets in the sequence using random functions.The agent's sequencer regulates the movement of request and answer sequence items between the sequences and the driver. From uvm sequencer, the sequencer is an extension. The TLM interface is used by the sequencer and the driver to exchange transactions. With the statement driver.seq item port.connect(sequencer.seq item export), we will link the sequencer and driver during the connect phase of the uvm sequencer.

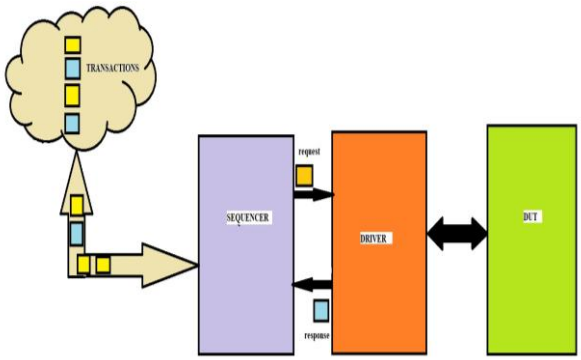


**FIGURE 3.**UVM Driver

The UVM Driver operates the DUV Interface by taking individual UVM Sequence Item transactions from the UVM Sequencer. To link the driver and the DUV, a virtual interface must be declared. All drivers should either directly or indirectly extend the uvm driver.The UVM driver pulls transaction level objects from the sequencer and uses an interface handle to drive them to the design. A certain sort of transaction object can be driven by a parameterized class called UVM driver. A TLM port of type uvmseq item pull port is present on the driver. This allows the uvm sequencer to pass it a parameterized request object. It can also give the sequencer a response object, and typically both the request and answer objects belong to the same class type. They may, however, differ if specifically stated. It essentially consists of three different sorts of methods. The get_next_item method blocks until the sequencer can fulfil a request for an item. Item_done call should come after this to complete the handshake. If a request object is not available from the sequencer, try _next_ item it Non-blocks will return null. If not, a pointer to the item is returned. The final item done also includes non-blocks, completing the handshake between the driver and sequencer. This ought to be used following a call to try_ next_item or get_next_item. It is in charge of transferring the packet level data from the sequence item to the pin level for the DUV. Figure 3 illustrates the flow of data packets from the sequencer to the driver and from the driver to the DUV.The scoreboard compares the expected output with the actual output from the output monitor. It reports the difference between actual and expected output. Scoreboard,driver,sequencer, sequence item, and monitors are executed in the environment module. In the Test module, we define phases, assign a virtual interface, and include the Environment module. Program module is for executing the random test case.In the Top module, we generate clocking, declare a virtual interface, instantiate the interface, and instantiate the DUT (design under test).

## 4. IMPLEMENTATION

Design and Testbench codes of 10 GE mac core was implemented in the Eda playground using SynopsysVCS 2021.09 tool in UVM 1.1d.

## 5. RESULT AND CONCLUSION

Testbench – driver module of 10 GE MAC core is coded in System Verilog language and UVM libraries using OOP'S concepts. Random test cases are built to verify the functionality of the 10 GE MAC core. Here, the size of the data packets is ten, so the sequence item generates 10 data packets.

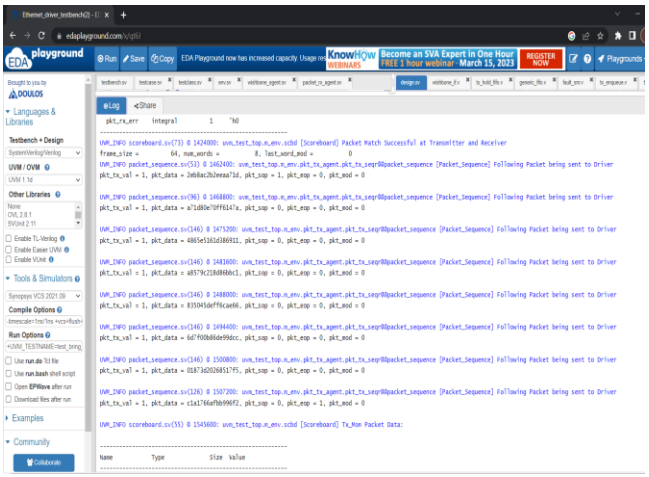

**FIGURE 4.**Data packet sent to driver

Figure 4 comprises one of the data packets that is sent to the driver by the sequencer. Likewise, ten data packets were sent to the driver. The driver sends the received data packets to the DUT according to the driver module code. Now the scoreboard compares expected output and actual output of design.

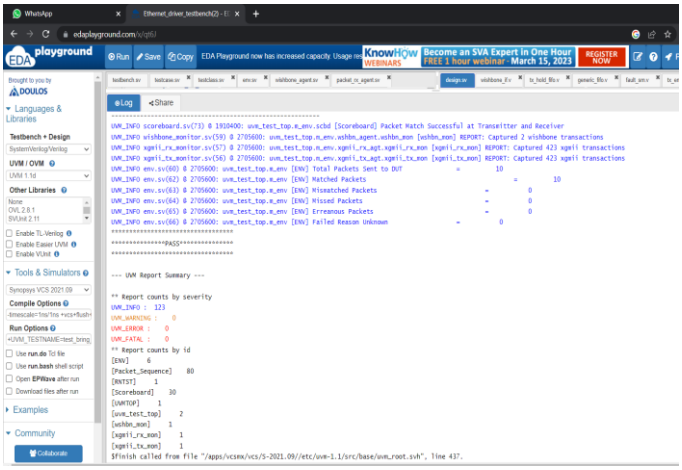

**FIGURE 5.**Output report

Figure 5 shows that the scoreboard verifies that data packets at the transmitter and receiver sides are the same, so there are no errors in our test bench. It is hereby concluded that the complex testcases of the testbench – driver module of the 10 GE MAC core is fully verified.

## 6. FUTURE WORK

10 GE MAC core Ethernet verification of driver is implemented in Eda Playground SynopsysVCS 2021.09 Tool which is open access and entry level tool for verifying the diver module of ethernet. With use of industry standard tools like Synopsys and Cadence tool, ethernet coverage of driver module will be more precise. Further, full ethernet component including the entire architecture will be implemented in hardware to verifying the functioning of ethernet.

## REFERENCES


1. K. Tanaka, A. Agata and Y. Horiuchi, "IEEE 802.3av 10G-EPON Standardization and Its Research and Development Status," in *Journal of Lightwave Technology*, vol. 28, no. 4, pp. 651-661, Feb.15, 2010, doi: 10.1109/JLT.2009.2038722.
2. T. Allen and R. Ge, "Demystifying GPU UVM Cost with Deep Runtime and Workload Analysis," *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Portland, OR, USA, 2021, pp. 141-150, doi: 10.1109/IPDPS49936.2021.00023.
3. M. Barnasconi *et al*., "UVM-SystemC-AMS Framework for System-Level Verification and Validation of Automotive Use Cases," in *IEEE Design & Test*, vol. 32, no. 6, pp. 76-86, Dec. 2015, doi: 10.1109/MDAT.2015.2427260.
4. C. Elakkiya, N. S. Murty, C. Babu and G. Jalan, "Functional Coverage - Driven UVM Based JTAG Verification," *2017 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*, Coimbatore, India, 2017, pp. 1-7, doi: 10.1109/ICCIC.2017.8524556.
5. J. Francesconi, J. Agustin Rodriguez and P. M. Julián, "UVM based testbench architecture for unit verification," *2014 Argentine Conference on Micro-Nanoelectronics, Technology and Applications (EAMTA)*, Mendoza, Argentina, 2014, pp. 89-94, doi: 10.1109/EAMTA.2014.6906085.
6. T. M. Pavithran and R. Bhakthavatchalu, "UVM based testbench architecture for logic sub-system verification," *2017 International Conference on Technological Advancements in Power and Energy ( TAP Energy)*, Kollam, India, 2017, pp. 1-5, doi: 10.1109/TAPENERGY.2017.8397323.
7. Y. Zheng and X. Sun, "Dual MAC Based Hierarchical Optical Access Network for Hyperscale Data Centers," in *Journal of Lightwave Technology*, vol. 38, no. 7, pp. 1608-1617, 1 April1, 2020, doi: 10.1109/JLT.2019.2959882.
8. R. A. Arenas, J. M. Finochietto and L. M. Rocha, "Design and implementation of packet switching capabilities on 10GbE MAC core," 2010 VI Southern Programmable Logic Conference (SPL), Ipojuca, Brazil, 2010, pp. 141-146, doi: 10.1109/SPL.2010.5483024.
9. S. Chitti, P. Chandrasekhar and M. Asha Rani, "Gigabit Ethernet verification using efficient verification methodology," 2015 International Conference on Industrial Instrumentation and Control (ICIC), Pune, 2015, pp. 1231-1235, doi: 10.1109/IIC.2015.7150935.
10. Young-Nam Yun;, "Beyond UVM for practical SoC verification", SoC Design Conference (ISOCC), 2011 International, pp158-162, 2011.
11. Creating a reusable testbench using cadance'stestbuilder and AMBA TVM Prakash Rashinkar, Peter Paterson and Leena Singh, SYSTEM-ON-A-CHIP VERIFICATION Boston: Kluwer Academic Publishers, 2001.
12. M. H. Assaf, S. R. Das, W. Hermas, E. M. Petriu and S. Biswas, "Verification of Ethernet IP Core MAC Design Using Deterministic Test Methodology," 2008 IEEE Instrumentation and Measurement Technology Conference, Victoria, BC, Canada, 2008, pp. 1669-1674, doi: 10.1109/IMTC.2008.4547312.
13. "IEEE Standard for Ethernet," in IEEE Std 802.3-2018 (Revision of IEEE Std 802.3-2015) , vol., no., pp.1-5600, 31 Aug. 2018, doi: 10.1109/IEEESTD.2018.8457469.
14. H. -T. Lim, D. Herrscher and F. Chaari, "Performance comparison of IEEE 802.1Q and IEEE 802.1 AVB in an Ethernet-based in-vehicle network," 2012 8th International Conference on Computing Technology and Information Management (NCM and ICNIT), Seoul, Korea (South), 2012, pp. 1-6.
15. S. Rabii et al., "An integrated VCSEL driver for 10Gb ethernet in 0.13/spl mu/m CMOS," 2006 IEEE International Solid State Circuits Conference - Digest of Technical Papers, San Francisco, CA, USA, 2006, pp. 930-939, doi: 10.1109/ISSCC.2006.1696134.
16. L. Leonardi, L. L. Bello and G. Patti, "Performance assessment of the IEEE 802.1Qch in an automotive scenario," 2020 AEIT International Conference of Electrical and Electronic Technologies for Automotive (AEIT AUTOMOTIVE), Turin, Italy, 2020, pp. 1-6, doi: 10.23919/AEITAUTOMOTIVE50086.2020.9307422.