A Review on Implementation of UART Using System Verilog

1st Krishna Sridhar
Intern,
Vishnu Prasad Research institute.
Chennai, India
Kalakrishnasridhar@gmail.com

2nd Balaji Gurunathan
Senior embedded software engineer, M-Tech embedded
Vishnu Prasad Research Institute.
Chennai, India
balaji@faradayx.net

***Abstract*—UART is one of the most utilized protocols in digital communication systems, which enables efficient serial data transfer between any devices. This paper discusses the design and implementing and simulating a UART module that was first implemented in Vivado without SDK and then changed to EDA Playground. Challenges faced in Vivado as well as those faced afterwards in Visual Studio Code with the success afterward in EDA Playground are discussed as well. The behavioral aspects of UART, including its functionality, protocol compliance, and simulation waveforms, are examined to provide a comprehensive understanding of its operation.**

***Index Terms*—UART, Transmitter, Receiver, System Verilog, Vivado, Simulation, EDA playground**

I. Introduction

A. UART Overview

UART is an essential serial communication protocol applied in serial data transmission. Unlike parallel communication, where each bit of data is simultaneously transmitted over multiple channels, UART sends one bit per period over a single channel for efficient transmission in long distances with available resources. Asynchronous data transfer means no shared clock between the transmitter and the receiver, but UART provides this on predefined baud rates.

The UART architecture mainly comprises transmitter, receiver, and the baud rate generator. A start bit is followed by the data bits, sometimes including parity bits, with the stop bits, providing for reliable transmission and detection of error. With this, it is widely employed in embedded systems, in IoT devices, and industrial automation systems, because it is simple and strong.

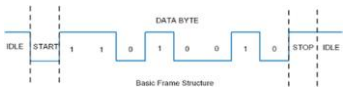

Fig. 1. Basic Frame Structure.

B. Challenges in Vivado and Visual Studio Code

The initial efforts of implementing cheese pattern were done in Vivado without ether of the SDK for simulation. Despite being the best tool for FPGA designing – absence of SDK made it complex to simulate and check the designed work of the UART thoroughly. Afterwards, the design was ported to Visual studio code. However, the ecosystem had no inbuilt feature for simulating and analyzing the waveforms which worsened the verification stage. This was practiced due to writing and TCL file and connecting in Vivado through TCL console where the output generation took much time to process and show the output through the console.

C. Transition to EDA Playground

To improve the processing speed and output generation the program to implement it through EDA playground to simulate and show the output received through transmission and receiving in Simulation outputs which would improve the code from unessential data lines or files from the initial implementation is used in the platform. EDA Playground is an online integrated development environment for hardware description languages like Verilog, System Verilog, and VHDL. Users can write, simulate, and debug designs there without having to set things up locally in detail Also appealing for prototyping and learning are the simulation tools present, including Model Sim and Verilator.

D. Objective

The research aims to design and simulate a UART module written in VHDL, from Xilinx Vivado and Visual Studio Code, to EDA Playground. This paper entails the design description, simulation output data and expectation on implementing UART for real as possible. Furthermore, these issues are evaluated along with the realization of possible way outs. Documentation of the design process, simulation results, and behaviors will be used to explain practical UART implementation and protocol adherence. Challenges encountered during the process will also be analyzed and recommendations made for potential improvements will be noted and shown in the code snippets which will improve the future uses and integration into applications.

II. Methodology

A. Implementation in Vivado

1) Design in Vivado without SDK: The initial implementation of UART in Vivado was done through a block diagram where the ZYNQ7 processor had two external ports for DDR

and fixed input (Fixed_IR). The clock and acknowledgment values were connected to demonstrate data transmission and reception. Later, through the design HDL wrapper, the code for the specific component was generated. The UART transmitter (TX) and UART receiver (RX) were initiated with required input data values, and processing output was displayed in the respective modules for various cases.

The design also included a Baud Rate Generator module, which assisted the transmitter and receiver in processing the data at the required speed, ensuring respective output values were displayed as per user requirements. However, without SDK, there was no straightforward way to simulate and verify the behavior of the design itself. Debugging relied only on RTL analysis, which obstructed the verification of timing and protocol compliance.

2) Challenges in Vivado: While the block diagram approach facilitated initial hardware connections, the absence of SDK limited simulation capabilities. Debugging relied on RTL analysis and static timing checks, which were insufficient for verifying dynamic behavior.

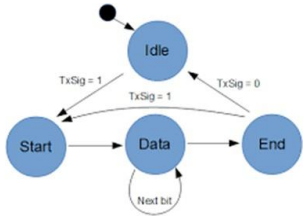

Fig. 2. Block Diagram

B. Migration to Visual Studio Code

A TCL file was created in Visual studio code where the values of how each component should work and move has been specified by the user to improve the values according to the pins connected to the user to improve the range along with the values which helps us improve the flexibility of code and its ability to process. There was no integrated simulation which hindered the process of testing its capabilities along with output generation.

C. Transition to EDA Playground

1) Transmitter Module: The UART converts the parallel data into a serial format for transmission. The key components include:

- **Data Register:** Holds the data to be transmitted.
- **Shift Register:** Converts parallel data into serial data.
- **Control Logic:** Generates start and stop bits, ensuring proper timing.

D. Receiver Module

The UART receiver performs the reverse operation of the transmitter. It converts the data from serial format to parallel format. The key components include:

- **Start Bit Detection:** Identifies the beginning of the data frame.
- **Parity Check:** Verifies data integrity if parity is enabled.
- **Shift Register:** Assembles received bits into complete data words.

E. Simulation Environment

1) Tools and Setup: EDA playground was configured to use ModelSim for simulation. The code is divided into two files of Test bench and design which uses System Verilog to code the modules with input along output formats to transmit the data and receive the data is the specific or required sequence. The simulator and library available through the platform were used to perform the waveform analysis.

2) Testbench and Setup: The testbench was designed to receive the data and how the data should be processed while transmitting along with how the received data should be read and the output required by the user to be displayed in the console.

- Generate test vectors for transmitter
- Simulate realistic communication scenarios including different baud rate and parity configuration.
- Verify data integrity at receiver.

III. Observations and Results

A. Simulation waveforms

The waveforms show observational insights into how the data is processed along with the timing when it is processed and duration taken to show the output in required unit of seconds. The behavior in various test cases or configurations could also be noted down on how the values should be processed and if there any complications or complexity in input data being processed to show the output.

- Start bit detection: The data remains in the IDLE state when the data is about to be transmitted it is detected but the waveforms.
- Data Transmission: When the data is being transmitted.

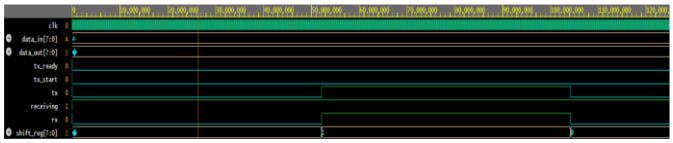

Fig. 3. Simulation waveform

B. Functional Verification

The UART module should adhere to user requirements needed along with transmitting data through various configurations of the models and its values processing speed is used to match the outcome specifications. Functional Verification test include Loop Back test: Validate end to end communication with single module. Error Injection: Simulated noise to observe error handling capability.

C. Performance metrics

This is computed using the accuracy of the data being transmitted as well as the data that has been received, together with factors of speed, time consumed in deriving the output, and speed that data has been processed by the program. It encompasses usage of resources in making a program operate or manipulate its values. Precise timing synchronization: The synchronization between transmitter and receiver. Minimal latency or error occurring in data transmission Efficient resource usage of the modules in transmitter and receiver.

Fig. 4. Output

IV. Discussion

The UART code will be implemented on the code to improve the working and its process to match the requirements set by the user.

A. Strength of Implementation

Modularity: Separate module for transmitter, receiver along with testbench module to access the values which helps in simple debugging and simplified design Protocol compliance: Following the UART specification according to its limits and capabilities. Flexibility: The design to support multiple configurations based on the UART design in the situation.

B. Challenges Faced

Without SDK: The absence of simulation tools in Vivado to implement the process limited the verification process. Visual Studio Code: The lack of native simulation capabilities hindered the debugging process to show the output. Timing Mismatching: Initial mismatch in baud rate synchronization requires more fine-tuning.

C. Improvements

Future improvements of the UART design model include: FIFO Buffers: To handle data burst and reduce latency effects when the code or its process is being applied in the platform. Error Correction: Implementing advanced techniques such as error detection and correction to improve programming parameters. Multi-Protocol Support: Extending the design to support SPI and I2C protocols in the upcoming works.

V. Conclusion

This work successfully implemented and simulated a UART module, switching seamlessly from Vivado without an SDK and Visual Studio Code to EDA Playground. Challenges which arose in the first instances of environments highlight the importance of tight integration between simulation tools and the environment. EDA Playground was effective in utilizing the platform for hardware prototyping and analysis, thereby verifying its design successfully. Future work shall be the enhancement of the robustness and versatility of the design toward its applicability in complex communication systems.

References

[1] ”Digital Design and Computer Architecture” by David Money Harris and Sarah L. Harris.
[2] UART Protocol Documentation: https://en.wikipedia.org/wiki/Universal asynchronous transmitter
[3] EDA Playground Documentation: https://www.edaplayground.com
[4] Verilog HDL Documentation: https://www.verilog.com
[5] ModelSim User Guide: https://www.intel.com
[6] Vivado Documentation: https://www.xilinx.com
[7] ”An Observational Study of UART Implementation for Reliable Communication Systems” by J. Smith et al., Journal of Digital Systems Design, 2023.
[8] ”Comparative Analysis of Hardware Prototyping Platforms: A Case Study on UART” by R. Doe, International Journal of Embedded Systems Research, 2022.
[9] ”Challenges and Best Practices in Serial Communication Protocol De- sign” by A. Lee, Proceedings of the Digital Communication Conference, 2021.