RTL to GDS Implementation and Verification of UART using UVM and OpenLane

A Project Report Presented to
The faculty of the Department of Electrical Engineering
San José State University

In Partial Fulfillment of the Requirements for the Degree
Master of Science

By

Purav Bhatt

SJSU ID# 015264041

EE 297B Section # 03 , Spring 2023

purav.bhatt@sjsu.edu

Phone # 408-816-9925

Dharmik Vijay Joshi

SJSU ID# 015216240

EE 297B Section # 05 , Spring 2023

dharmikvijay.joshi@sjsu.edu

Phone # 669-588-7170

Department Approval

Shrikant Jadhav
Project Advisor

: DocuSigned by: Shrikant Jadhav
shrikant.jadhav@sjsu.edu

Date : *5/15/2023*

Birsen Sirkeci
Graduate Coordinator

: DocuSigned by: Birsen Sirkeci
birsen.sirkeci@sjsu.edu

Date : *5/15/2023*

Department of Electrical Engineering
Charles W. Davidson College of Engineering
San José State University
San Jose, CA 95192-0084

Upload Files

Project Report	Turn-it-in report	Project Poster
		

Submission Date:*5/15/2023*

Purav Bhatt

5/15/2023

DocuSigned by:
9439AD5F808B49A...

Dharmik Vijay Joshi

5/15/2023

DocuSigned by:
Dharmik Vijay Joshi
14ABF3845580425...

TABLE OF CONTENTS

1) Abstract…..…….…… 2

2) Introduction……..…….……..... 2

3) Literature review……...…..……..….... 4

4) UART RTL Design…….……...… 5

5) UART Verification……. 11

6) UART Physical Design……..……. 21

7) Conclusion…….. 29

8) References…….. 30

RTL to GDS Implementation and Verification of UART using UVM and OpenLane

Dharmik Vijay Joshi: dharmikvijay.joshi@sjsu.edu; Purav Bhatt: purav.bhatt@sjsu.edu

Shrikant Jadhav: shrikant.jadhav@sjsu.edu

1 Abstract

The paper presents an ASIC design flow implementation for Universal Asynchronous Receiver and Transmitter (UART). It starts with the RTL implementation using Verilog HDL. It's then verified using UVM, including test benches, sequences, and monitors. Finally, the paper discusses the use of OpenLane for the physical design implementation of the verified design from synthesis up to GDS (Graphic Data System). The goal is to demonstrate how a sophisticated digital circuit is designed and verified. The results of the verification are shown via waveforms for test cases. The results of the backend implementation include layouts and QoR (Quality of Results) report tables. The emphasis is also on design rule checking (DRC) and layout versus schematic (LVS) to ensure accuracy and manufacturability. The paper also mentions the difficulties encountered while working on the project and offers suggestions for future developments.

2 Introduction

With the increase of IoT applications, HPC, and Big Data solutions, it becomes quite crucial to have a communication protocol to match those demands. A reliable, reusable, and plug-and-play solution can help with the sharp rise in data communication. In real-time systems, a communication protocol becomes center stage for peripherals to interact with the processor so that the propagation delay would be as low as possible. If the communication is not reliable, the real-time system would essentially be a failure. For serial communication through a computer or a peripheral device, UART is often an Integrated Circuit (IC).

Figure 1 explains the functionality of UART. 2 modules are connected in such a way that the transmitter of one is connected to the receiver of the other. UART1 transmits data and UART2 receives data. Today, UARTs are frequently seen in microcontrollers. The universal asynchronous receiver/transmitter (UART) receives the data as bytes and sends the bits sequentially. The received bits are put back together into whole bytes at the destination side using a second UART. A shift register is present in each UART and is essentially used to translate serial data into parallel data. Compared to parallel transmission across several wires, serial transfer of digital information (bits) using a single wire or other media is less expensive.

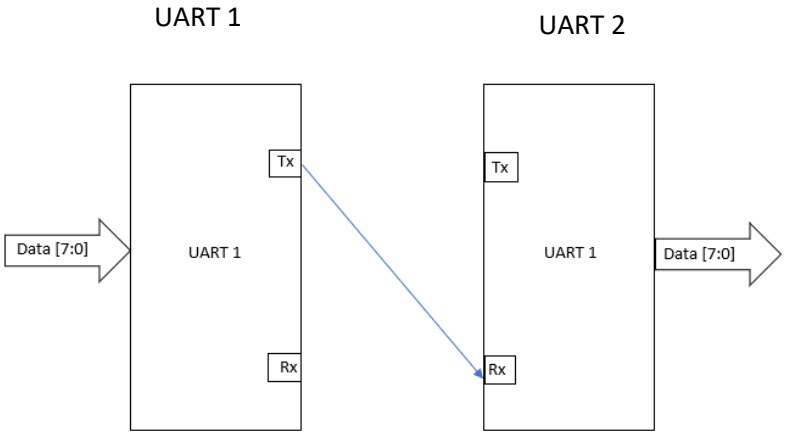

Figure 1: Traditional UART communication

The paper encompasses the whole ASIC design flow. The ASIC (Application-Specific Integrated Circuit) design pipeline is a structured and systematic process used to create custom integrated circuits tailored for specific purposes. It involves multiple phases, starting from concept and specification and extending to manufacturing and testing. The initial stage focuses on establishing the criteria and specifications for the ASIC, including its intended function, desired functionality, and performance objectives. In the architectural design stage, the overall organization and structure of the ASIC are determined. This entails selecting the system design, dividing the functionality into components, and defining their interactions. The next step is the register transfer level (RTL) design, where the high-level architectural description is transformed into a digital representation using hardware description languages (HDLs) like Verilog or VHDL. This involves designing and describing the behavior and connections of individual digital components. Once the RTL design is completed, it undergoes rigorous functional verification to ensure that it operates as intended. Techniques such as hardware emulation, formal verification, and simulation are employed to verify compliance with the requirements. In the physical design stage, the gate-level representation is transformed into a physically feasible layout. Tasks like floorplanning, placement, clock tree synthesis, routing, and optimization are performed to meet time, power, and space constraints. Once the final layout database is created, containing all the necessary design data, it is ready for manufacturing after it has been thoroughly reviewed and satisfies all relevant requirements. The manufacturing process involves sending this database to a semiconductor foundry for fabrication.

The verification process plays a critical role in the development of VLSI chips. It involves running numerous test cases to ensure that the design functions as intended. Verification and performance measurement of integrated circuits are demanding tasks that require significant time and effort. As the complexity of the device design increases, verification becomes even more challenging. Traditional verification approaches often lack reusability, and their time to market is typically lengthy. The Universal Verification Methodology (UVM) library addresses these limitations by providing reusable components and testing environments using System Verilog. UVM is a widely adopted methodology for verifying integrated circuit designs, offering advantages such as reusability, scalability, and interoperability. Employing UVM enables the realization of these qualities and brings multiple benefits to the verification process.The physical design of the UART (Universal Asynchronous Receiver-Transmitter) is facilitated using the OpenLane toolsets. OpenLane is a leading open-source application

for semiconductor digital design. It aims to promote open access, expertise, rapid innovation, and accelerated design turnaround by eliminating barriers related to cost, risk, and uncertainty in hardware design. OpenLane offers a flexible flow control by providing an API with Tcl and Python bindings. It is an integral part of open-source digital design flows, exemplified by OpenLane-flow-scripts.

In this paper, the physical design is also demonstrated. Physical design plays a vital role in creating a wide range of products, systems, and structures, encompassing architecture, industrial design, integrated circuit layouts, and urban planning. Its purpose is to transform intangible requirements, ideas, and concepts into tangible and practical forms. The goal of physical design is to optimize the placement of elements and physical features to achieve effectiveness, efficiency, and user satisfaction. Partitioning is a crucial step in physical design, where a complex circuit is divided into manageable subsystems or sub-blocks with minimal interconnections. During this phase, the shape and approximate location of each block on the silicon chip are determined. The placement of pins for each block is also decided, ensuring easy routing of connections between these blocks in the future. Placement involves precisely positioning the sub-blocks to allow sufficient space between them for routing Vdd and Ground supply lines, as well as connector wires. Static timing analysis is then performed on the final routed netlist, which represents or simulates the chip's actual layout to validate its timing performance. To ensure the design's functionality remains unchanged, Layout Vs Schematic (LVS) equivalency verification and physical design implementation Layout should be conducted. These steps confirm that the design's intended functionality is preserved throughout the physical design process.

3 Literature Review

The verification approach can be followed to verify the UART protocol using UVM. This verification method includes setting up a UART configuration type in which all the UART protocol parameter information is packaged and sent the UART protocol parameter information to related platform parts like a driver, a monitor, and a grade recording board through a config-db mechanism provided by the UVM [1]. This process involves finishing an overall framework of UART protocol verification using a UVM verification platform structure and a verification idea. This article signifies the importance and the need for UVM to verify the UART protocol as it can make the verification of a block simpler and more efficient concerning other conventional verification methods [2]. It also gives a detailed overview of how UART works and how the verification environment should be set up to check for communication with the same. The purpose of this study is to use SystemVerilog to develop a UART and a UVM-based testbench to validate the design. One of the most popular serial communication protocols that can be used without a clock stimulation is UART [3]. The Baud rate generator, control, bus interface, interrupt control, and receiver-transmitter FIFO blocks are all included in the architecture [4]. System bus parallel data are serialized, transported, and converted back to parallel data at the receiving end. A reusable and adaptable verification environment reduces complexity and processing time. This paper details the SV hardware description language design for the UART and SPI functional modules. The Universal Verification Methodology is used to undertake functional verification of the UART and SPI [5]. By comparing the collected response to the intended response via the scoreboard mechanism, the reusable UVM testbench architecture is meant to push the randomized stimuli to the unit under test to assess the6functional correctness. The overall execution time will be shortened by the stimulus's reusability [6]. This paper describes the design of a UART in Verilog and UVM verification. This paper's verification demonstrates the full functionality of UART, which may be validated using a UART device. The UVM-based verification will cut down on the total amount of time the logic gates need to operate. The overall execution time will be reduced since the tests of sequences are kept separate from the original test bench hierarchy and the reusability of stimulus [7]. This paper's primary goal is to use

System Verilog to design and test a full duplex UART module (SV). It is a serial communication protocol that allows devices to communicate without the use of a clock signal. It sends parallel data that has been converted to serial format [8]. Data that is received in serial format is transformed into parallel format. The baud rate generator, receiver, transmitter, interrupt, and FIFO modules are all designed as part of the UART design process. Verification entails checking the design by establishing a verification environment that permits testbench reuse and minimizes code complexity. test the difficult-to-reach corner situations, randomization is applied [9]. 100% functional coverage and 100% assertion are attained. The design and verification of UART are the subjects of this study . UARTs are now included in microcontrollers quite frequently. A UART is typically an exclusive integrated circuit (IC) or a component of an IC that is used for serial communication on a serial port on a computer or peripheral device [10]. It works flawlessly when connected to a PC's serial port for data transfer with customized electronics. Today, UARTs are frequently included in microcontrollers. A full duplex transmitter or receiver is a UART.

The paper highlights the architecture that uses the shift register for both the transmitter and the receiver. The transmitter has FSM that has 5 stages, which makes the data-frame which is of 10 bits. Also, the receiver has 3 stages which are used to read the data-frame and catch the parity bit for errors. The paper also demonstrates the implementation of the physical design part which lately has not been done as part of the full ASIC design flow. The flow stops at verification or validation on the FPGA board. But the backend implementation is not performed which can be very useful to optimize the design as per power, area and performance.

4 UART Design

The UART design has a transmission FIFO, receiver FIFO, a couple of shift registers, a baud rate generator, a control unit, and a wishbone interface. The baud rate generator produces various baud rates for transmission. It then gives that information to the control unit which then produces various reads, writes, and other control signals. The shift register is used to link the data packet generation to the FIFOs. The BRG is a flexible and programmable unit that produces a baud clock. The shift register helps to hold the data packet by adding parity bits and BRG bits. The FIFO helps to transmit and receive the data serially. The control unit helps to know when to read or write data. The IP core has a wishbone interface which helps to make the design more industry-standard. Also, the design has a register bank, especially for the control registers and interrupt registers.

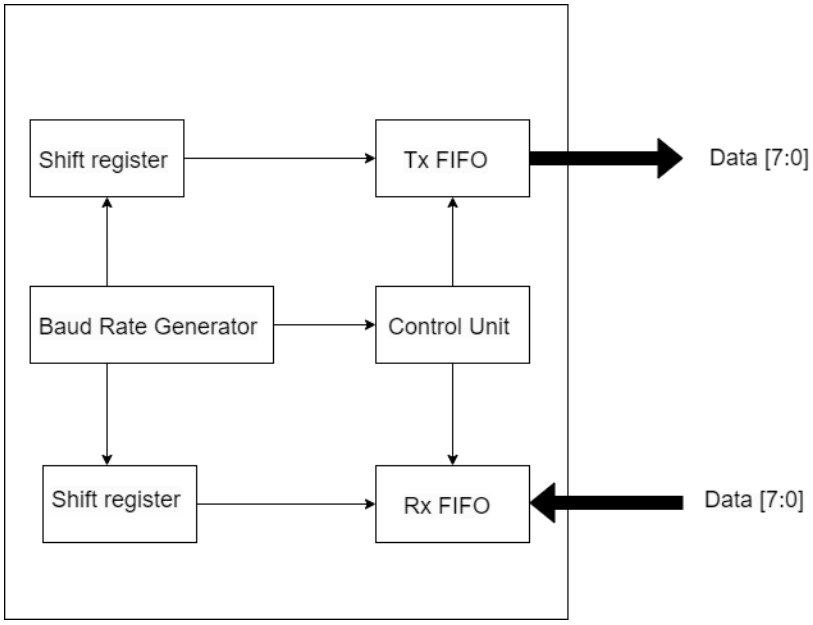

Figure 2: Block diagram overview of proposed UART

Figure 2 explains the block diagram of UART. The baud rate generator develops the baud rate as per the user. The control unit is essentially a finite state machine that changes the control bits for both the transmitter and receiver. The shift register essentially is used to roll in data bit by bit. The FIFO is used to streamline the data. In asynchronous communication, transmission speed is determined by the Baud rate generator. The 16-bit timer that creates the clocking mechanism is used.

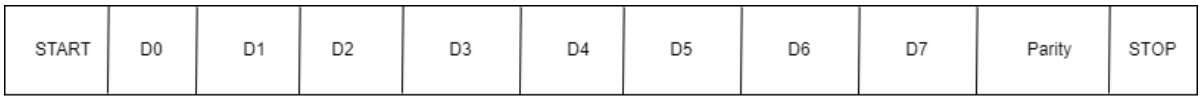

Figure 3: UART data format

Figure 3 explains the data frame of the UART. The Start bit is inserted at the start of a word when it is transferred to the UART for asynchronous transmissions. Therefore, the Start bit is first utilized to signal to the receiver that a word of data is about to be sent, putting pressure on the receiver's clock to match the transmitter's clock. The word of data's constituent bits is delivered after the Start bit, starting with the Least Significant Bit (LSB). An optional Parity bit is then delivered to the transmitter once the data has been sent in its entirety. The receiver often uses this bit to carry out basic error checking. To signal the end of transmission, a Stop bit will be transmitted at the conclusion. A preset sequence is applied by the verification environment to the design, and the outcome is compared. The intended UART module is represented by the DUT. The test bench can be used again because the environment can contain one or more agents. Two distinct DUTs stand in for two UART modules, and each DUT has a separate verification agent. Monitor, driver, and sequencer functions are all contained within the agent. To ensure full duplex operation, two agents are created. To test all the random conditions,

randomization is employed to randomize the sequence. The randomize feature is used to create random test scenarios that will examine all of the functions' hard-to-reach corner circumstances. Since tests for design verification are automatically generated through random testing, it is more effective than other methods.

4.1Baud Rate Generator

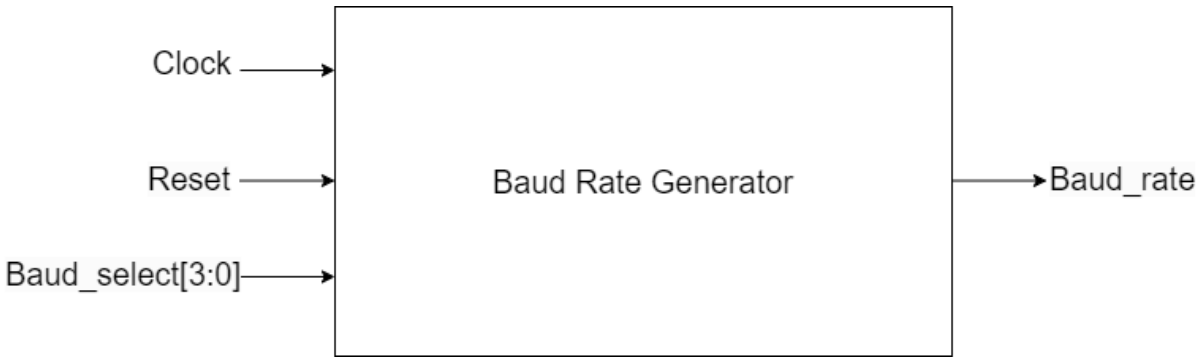

Figure 4: BRG Implementation with variable rates

Figure 4 explains the baud rate generatorThe baud generator on the UART is programmable. making a baud clock. A divisor that is kept in a divisor latch is used by BGR to split an input clock from the processor clock generator (BCLK). 16 baud clock cycles, or sixteen times the baud rate, are used to send data. The baud rate frequency factor can be calculated using the requested baud rate and the system clock frequency (sometimes referred to as oscillator clock). The receiver should accurately reflect the asynchronous serial data. The BRG module has the inputs: Clock, reset, and selection bits for the baud rate. The reset pin disables the operations in the circuit. The module provides 6 different baud rates to the user as per the requirement. These are 4800 bps, 9600 bps, 19200 bps, 38400 bps, 57600 bps, and 115200 bps. As a result, the module provides 4 selection bits to the user. It uses a 32-bit counter that generates various delays so that the user can get a particular baud rate. The 4-bit selection pins are connected to an internal multiplexer to select the baud rate. The UART utilizes a divisor to scale down the frequency of the clock signal to produce the desired baud rate because the frequency of the clock signal is typically significantly higher than the baud rate.

$$Divisor = \frac{Clock\ Frequency}{16\ *\ Desired\ Baud\ Rate}$$

Equation 1

Equation 1 refers to the calculation of the divisor for a particular baud rate. The module provides 6 different baud rates to the user as per the requirement. These are 4800 bps, 9600 bps, 19200 bps, 38400 bps, 57600 bps, and 115200 bps. So, the corresponding divisor is calculated from the previously mentioned baud rates. These divisors are used in the counters to get the baud rate.

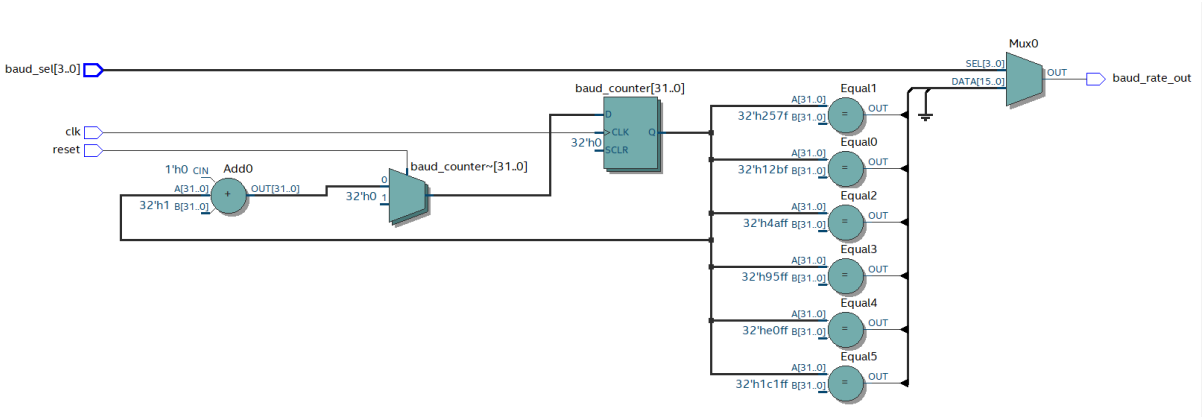

Figure 5: Implementation of Baud Rate Generator

Figure 5 shows the internal circuitry of the baud rate generator. This design is implemented using reference UART models and integrating them according to our needs where the option of variable baud rates and the above-mentioned formula play a vital role in determining the speed of the transmission. The design is generated using the physical design CAD tools available using the reference Verilog module that gets fed to the software to generate the behavioral and pin-compatible circuit.

4.2 Transmitter

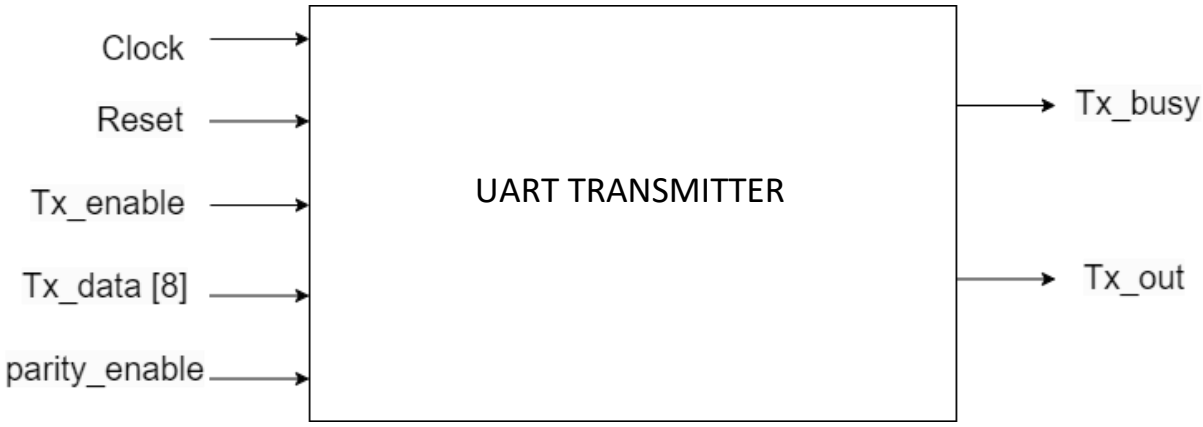

Figure 6: UART Transmitter with defined pins

Figure 6 explains the transmitter block diagram. The transmitter module takes inputs which are: clk, reset, enable, data and the parity enable bit and produces data to transmit and busy bit as outputs. It uses an internal shift register to transmit the data, a counter, that counts the number of bits that are to be transmitted as well as the register for keeping the state machine information. The transmitter is in charge of using a communication channel to transmit data in the form of binary signals, one bit at a time, to a receiving device. It changes the parallel data coming from the microcontroller into serial data that can be sent through a serial port or a bus of communications. The UART transmitter operates in an asynchronous mode, which means that data transmission synchronization is accomplished without the

use of a clock signal. To frame each data byte and guarantee proper synchronization, it instead uses start and stop bits. Many embedded systems and communication protocols depend on the UART transmitter as a key component.

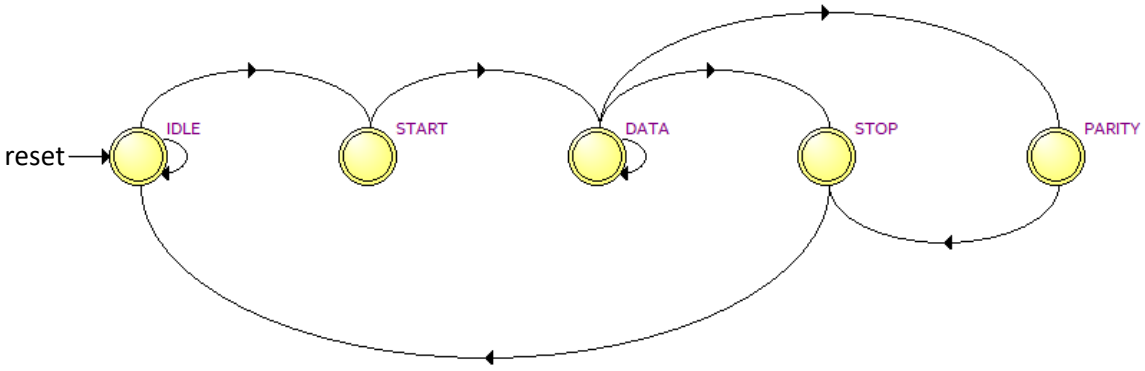

Figure 7: Software-generated Transmitter State Machine

Figure 7 shows the state flow of the transmitter. It has five states, IDLE, START, DATA, PARITY, STOP. By default, it starts with the IDLE state before the Tx_enable signal is switched ON. Then it goes to the START state. Here, the transmitter adds the start bit to the shift register and shifts the data left. Then it goes into the DATA state. Here, the transmitter shifts data left until all 8 bits have been transmitted. If the user does not want to add the parity bit to the data frame, the parity_enable signal is not turned ON and finally, the FSM enters the last stage which is the STOP bit. If the parity_enable signal is turned ON, the transmitter adds a parity bit before entering the STOP bit. Then the data frame is given out as an output from the pin Tx_out. During all the operations are happening, the busy signal is set to 1 to indicate that the next data frame has to wait till the current operations are finished.

4.3 Receiver

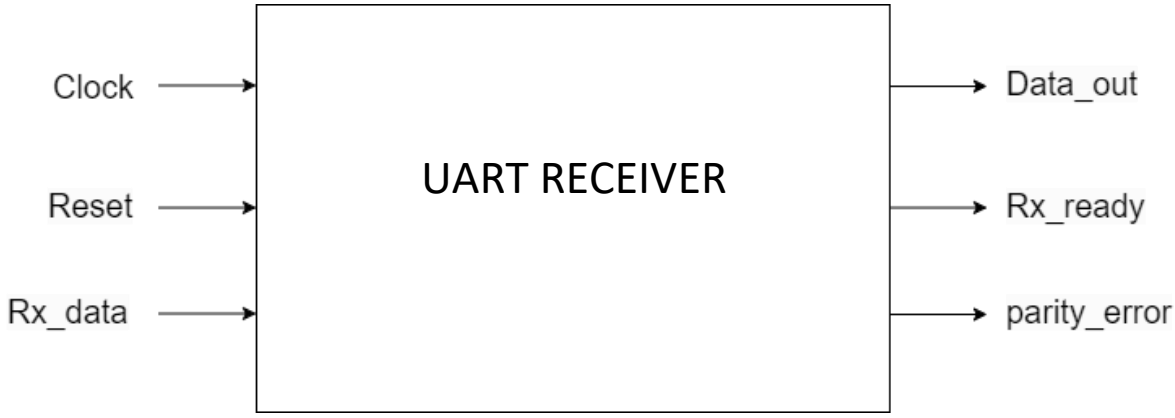

Figure 8: Receiver Implementation

Figure 8 shows the UART receiver. The receiver module takes in 3 inputs and produces 3 outputs. Inputs are: clock, reset, and data, outputs are: data_out, ready, and parity signal. The reset signal shuts off all the operations of the circuit. The clock signal is used to transit from one state to another. It has various internal registers such as shift register, state register, bit-count register, and received-data register. The shift register is used to shift the data frame coming from the transmitter. The state register has information about the 3 states of the receiver. Th bit-count register is used to set as a base for the counter to go through the state frame. The received data register is used to store the data coming from the transmitter. The rx_data signal indicates whether the data is received and stored in the internal register or not. The rx_ready signal indicates if the receiver is ready or not, i.e. if the operations are ongoing, then it cannot accept the next set of data frames. Parity_error signal indicates if the parity error is caught by the receiver or not.

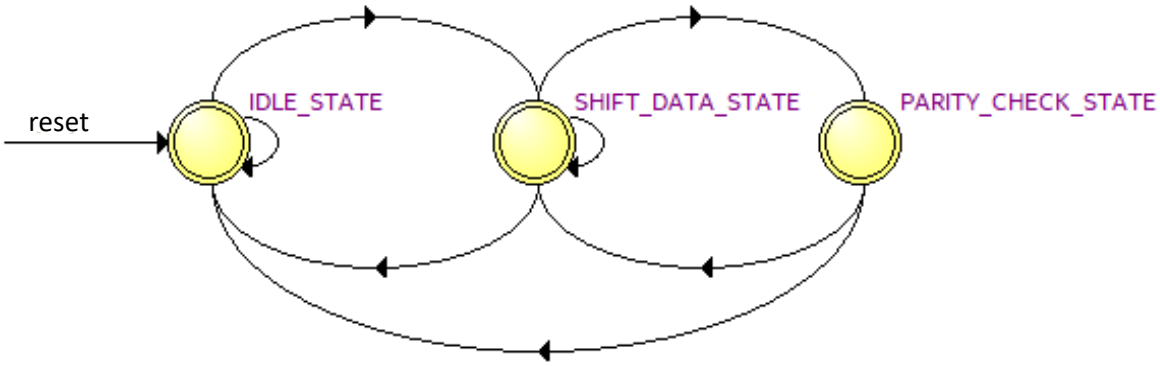

Figure 9: Software generated UART Receiver State Machine

Figure 9 shows the finite state machine of the receiver. It has 3 states namely, IDLE_STATE, SHIFT_DATA_STATE, and PARITY_CHECK_STATE. The state machine starts by waiting for the start bit to be received in the IDLE_STATE. The state machine switches to the SHIFT_DATA_STATE after detecting the start bit, where it inserts incoming data bits one at a time until all data bits and the parity bit are received. The state machine switches to PARITY_CHECK_STATE after receiving all the data bits and the parity bit, where it checks the parity bit for parity mistakes.

5 UART Verification

A popular digital system communication protocol that enables the sending and receiving of serial data between two devices is called UART (Universal Asynchronous Receiver Transmitter). The performance and functionality of UART must be tested in contemporary designs to guarantee that it functions as intended. A common verification methodology for digital designs called the Universal Verification Methodology (UVM) offers verification engineers a strong and effective framework for creating and managing intricate verification environments. Engineers can construct efficient and reusable testbeds that can more methodically and effectively verify the functionality of UART by using UVM. The UVM verification methodology can provide better control over the verification process, improve simulation performance, and enable the creation of complex scenarios for testing the functionality of UART.UVM can be used to verify UART in a way that drastically cuts down on both the time and cost of the verification process while also improving design quality. Early on in the verification process, it can assist in finding and fixing design bugs, resulting in a more effective and reliable design. UVM also enables the development of testbenches that can generate a wide variety of test scenarios, including various corner cases, error conditions, and boundary conditions that can help guarantee the correct functionality of UART under various operating conditions. In order to ensure that UART functions correctly and dependably in digital designs, UVM verification methodology must be used to verify UART.

The steps performed by the team in verifying the UART DUT in UVM can be divided into the following steps:

5.1Develop a test plan

The design's functional, performance, and interface requirements should all be specified in the test plan. Goals for coverage and test scenario identification should also be included. The test strategy should be thorough, encompassing all potential uses of the UART DUT, and it should be specific, including a list of test scenarios, test methods, and anticipated outcomes. The test cases should cover all of the design's functional needs, including data transfer, error detection and repair, and flow control methods. Additionally, the verification environment has to be reusable for multiple test case scenarios in the same integration environment.

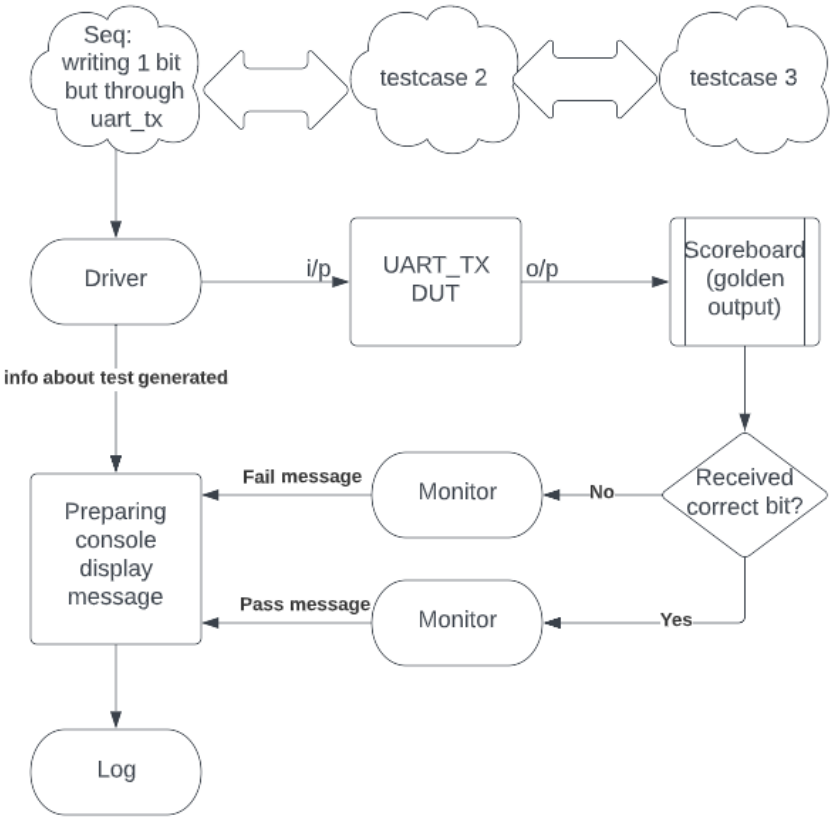

Figure 10: Test Plan Flowchart

The flow chart defined in Figure 10 depicts the verification methodology defined by the team. It is possible to divide the UVM verification of UART into several steps. The first step is to build a UVM testbench that has a scoreboard, a monitor, and a test sequence generator. The test sequence generator generates a stimulus that activates the UART module, which is the design under test (DUT) in this instance. The monitor gathers data from the signals being sent and received at the DUT's interface and produces transactions that are sent to the scoreboard. The scoreboard analyzes any discrepancies between the transactions produced by the monitor and the expected transactions and reports them.

A UVM testbench environment with a driver, a sequencer, and a virtual interface must be created next. The virtual interface is used by the driver to communicate the transactions produced by the test sequence generator with the DUT. The transaction flow is managed by the sequencer, who also makes sure that the transactions are sent in the right order. In addition to isolating the testbench from the DUT's implementation specifics, the virtual interface offers a transaction-level interface between the two. The UVM testbench must be run as the last step using a simulation program like QuestaSim or VCS. Here we have used an open-source simulation platform edaplayground to run our simulations and waveforms. The test sequence generator generates a stimulus that is sent to the DUT through the driver and the virtual interface during the simulation. The monitor gathers data from the signals being sent and received at the DUT's interface and produces transactions that are sent to the scoreboard. The scoreboard analyzes any discrepancies between the transactions produced by the monitor and the expected transactions and reports them. In the event that there are any discrepancies, the testbench can be

troubleshot to determine the root of the issue. UVM can be used to automate, standardize, and streamline the verification of UART, producing designs with higher quality and faster time to market.

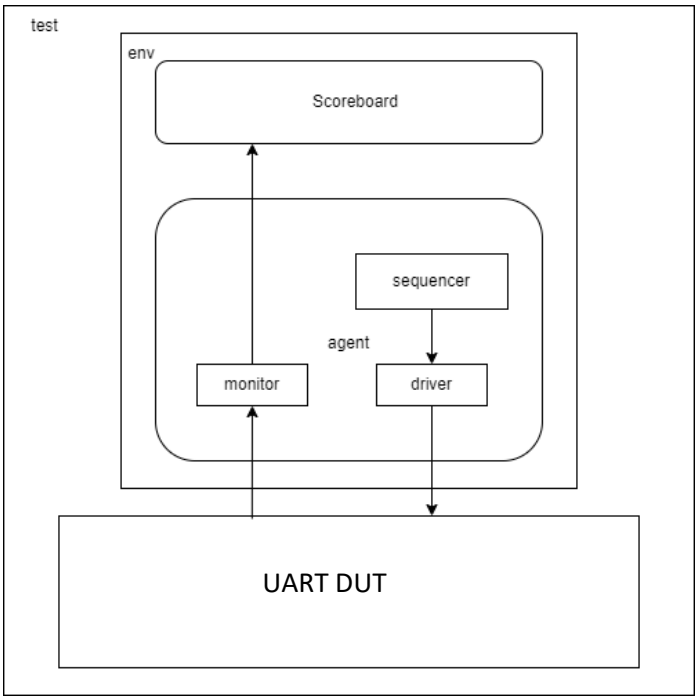

Figure 11: UVM components

As depicted in the figure 11, The sequence is the data being transmitted, while the Sequencer merely acts as a bridge between the Sequence and the Driver. The driver receives the randomized data from the sequencer. The driver sends this info to DUT following protocol. A monitor keeps track of the communication data and feeds it to the scoreboard. The output of the DUT is compared to the sequence on the scoreboard. The term coverage is used to gauge the verification's progress. Functional coverage tracks how the test plan is being executed. It compiles the simulation data to produce the progress report. It keeps track of the extent to which other features, such as important sets of values and boundary conditions, are covered. Knowing what set of values or characteristics has been tested for is a crucial component of testing. To improve the effectiveness of the design, the test cases are adjusted or new test cases are introduced using the progress report. The user defines the coverage points for the functions in a DUT. All of the functions in the test plan have been properly tested, according to 100% coverage. The test procedures should include the expected outcomes for each test case as well as step-by-step instructions for carrying out the test cases. The intended outcomes must be specified precisely, describing how the UART DUT is supposed to behave. Goals for coverage should be included in the test strategy as well. The functional requirements of the design should serve as the basis for defining the coverage targets, which should encompass both functional and code coverage. Statement, branch, and condition coverage are a few examples of the coverage metrics that should be included in the test plan.

5.2UVM Environment Creation

To drive the UART DUT with the test stimulus and record the response, the UVM environment is essential to the verification process. The UVM environment is built using the Universal Verification Methodology (UVM) library, which is a standardized methodology for developing reusable verification environments. Several UVM elements, such as the agent, monitor, scoreboard, and driver, make up the UVM environment. The agent is in charge of applying the test stimulus to the UART DUT and recording the response. It includes a sequence generator that creates test sequences using the test plan created earlier. The agent also includes a driver, which transforms the test sequences into signals sent to the UART DUT, and a monitor, which records the UART DUT's response. Verifying the accuracy of the UART DUT's response is the responsibility of the scoreboard. When there is a discrepancy between the response from the monitor and the anticipated outcomes specified in the test plan, an error is generated. The UVM environment created by the verification has been shown in figure 12.

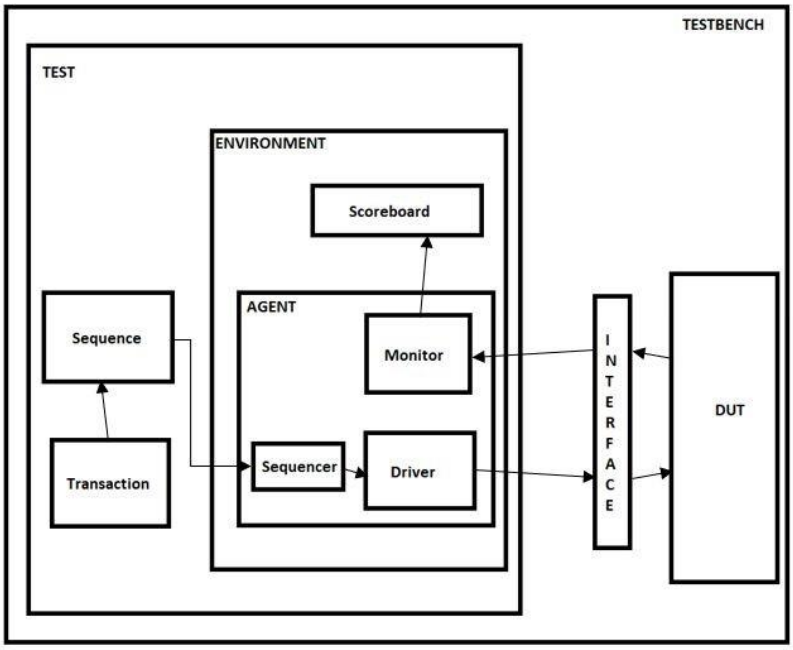

Figure 12: UVM Environment

Figure 12 explains the UVM environment. A configuration object, which is used to pass parameters to the various UVM components, as well as a test bench, which organizes the execution of the test sequences, are also included in the UVM environment in addition to the UVM components. A UVM testbench environment with a driver, a sequencer, and a virtual interface must be created next. The virtual interface is used by the driver to communicate the transactions produced by the test sequence generator with the DUT.

5.3 UVM Test Case Development

Defining the test scenarios and making the appropriate test sequences that will apply the required stimulus to the UART DUT are required steps in developing UVM test cases. The purpose of these test cases should be to confirm that the UART DUT complies with the specifications listed in the test plan. To test the DUT's fault-handling and error-recovery mechanisms, the UVM test cases should also include a variety of error scenarios. The test cases might, for instance, involve scenarios in which the UART DUT receives erroneous data, encounters buffer overflow or underflow, or encounters communication errors as a result of noise or interference. The sequences, drivers, and monitors from the UVM test bench should be used to create the UVM test cases. These elements are in charge of creating the necessary stimulus and recording the responses from the DUT. Following the development, the test cases are added to the UVM test bench and executed using a UVM test runner. The UVM environment records the results as the test cases are carried out, creates a test coverage report, and calculates the proportion of the design that has been tested using this information. It may be necessary to add more test cases to increase the test coverage if any design elements are not completely covered.

Testcases checked:

Table 1: Test Case Coverage

	Test Name	**Test case description**	**Status: Pass/Fail/Unchecked**
1	transmitter check	Correct data to Tx check	Pass
2	transmitter address allocation check	Whether the Tx data is allocated to the right address	Pass
3	receiver check	Correct data to Rx check	Pass
4	receiver address allocation check	Whether the Rx data is allocated to the right address	Pass
5	Are the transmitter and receiver both working parallelly or not	Full duplex check	Pass
6	baud rate check	Whether the system operates at the right baud rate	Pass
7	baud rate change check	Toggling baud rates	Unchecked
8	transmission with correct baud rate check	Whether the tx baud rate is accurately generated as mentioned in the spec	Pass
9	receiving with correct baud rate check	Whether the rx baud rate is accurately captured as mentioned in the spec	Pass
10	variable baud rate Tx and rx check	Operating tx and rx at different speeds	Fail
11	parity check	Checking the parity of the signals	Pass
12	data conversion with correct parity check	Checking the tx and rx as a whole after adding parity bits	Pass

5.4 Functional Coverage of the System

To make sure that the test cases have covered every aspect of the UART DUT's functionality, functional coverage is a crucial metric. It assists in determining how thoroughly the verification process has been carried out. Functional coverage points must be established to accomplish this; these points represent particular functional features or features that must be combined and put to use by the test cases. The UVM environment's coverage collector component can be used to gather the coverage results after these coverage points have been defined using SystemVerilog cover groups or cover points. Once functional gaps in the verification have been found, the coverage data can be examined to determine which areas need further testing. To make sure that all facets of the verification process have been covered, it is crucial to monitor additional coverage metrics in addition to functional coverage, including code coverage, assertion coverage, and protocol coverage. Doing so makes it easier to spot any potential problems that might have gone unnoticed during the verification process and guarantees that the UART DUT satisfies the necessary functional and performance requirements.

5.5 Protocol Checking

Protocol checking involves confirming that the DUT complies with the UART protocol specifications in the context of UART verification using UVM. To accomplish this, the output of the DUT is typically compared to the expected behavior specified in the UART protocol specification.

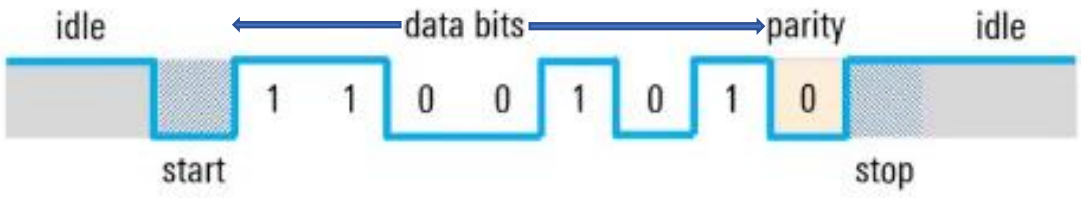

Figure 13: Protocol Reference

Figure 13 explains the protocol reference. In serial communication between two devices, the UART (Universal Asynchronous Receiver/Transmitter) protocol is frequently used. A start bit, data bits, optional parity bits, and stop bits are all included in the format of the data that is transmitted. The following describes the UART's data format:

1. Start Bit: When a UART transmits a bit, the start bit is always the very first one. It serves as a notification to the receiver that a new byte of data has begun. In most cases, the start bit is one bit-time long and always low ((0)).

2. Data Bits: Following the start bit, the data bits themselves are transmitted. Depending on the application, there may be 5, 6, 7, or 8 bits of data transmitted for each byte. The most typical data format consists of eight data bits, one stop bit, and no parity.
3. Parity Bit: To check for errors during data transmission, the parity bit is a supplementary bit that is optionally included. The total number of bits transmitted (including the parity bit) is made even or odd by setting the parity bit to either 0 (even parity) or 1 (odd parity). Once the receiver has verified that the total number of bits transmitted is accurate, it checks the parity bit.
4. The stop bit, which the UART always transmits as the final bit, is known as the stop bit. It is employed to inform the receiver that the last data byte has been received. In accordance with the application, the stop bit is always a high (1) bit and is typically one or two bit-times long.
5. UART makes sure that the transmitted data is received precisely and without errors by adhering to this data format.

Verifying compliance with protocols is the responsibility of the UVM environment's protocol checker component. It receives the output from the monitor component, which records the interaction between the DUT and the testbench, and evaluates it in comparison to the predicted behavior. Errors are reported for any differences between the actual behavior and the expected behavior. To make sure that the DUT complies with the necessary communication standards and that it can communicate with other devices that use the same protocol, it is crucial to perform the protocol-checking step. Additionally, early detection of any potential design flaws or problems during the verification process is helpful.

5.6 Performance Analysis for the DUT

A UART DUT's performance analysis is an essential part of the verification process. To confirm that the UART DUT satisfies the necessary performance requirements, the performance of the device is being examined. Throughput, latency, and power consumption are among the performance metrics that are frequently verified. The amount of data that can be transmitted in a given amount of time is referred to as throughput. Different test cases are created to transmit various amounts of data at various speeds to verify throughput. To confirm that the data transfer rate satisfies the necessary throughput requirements, a measurement is then taken. The time interval between data transmission and data reception by the receiving device is referred to as latency. Test cases are created to transmit data at various rates, and the time between transmission and reception is measured to confirm that it complies with the necessary latency requirements. Another crucial performance metric that is confirmed during performance analysis is power consumption. The UART DUT is tested under various operating conditions and its power consumption is measured using power analysis tools to confirm power consumption. To confirm that the power consumption complies with the necessary power specifications, it is then compared to those specifications. Overall, performance analysis makes sure the UART DUT complies with the necessary performance requirements and is prepared for integration into the finished product.

5.7 Corner Cases Verification

Corner case testing is a crucial step in determining the UART DUT's robustness. It entails testing conditions that are on the periphery of the typical operating range, including minimum and maximum values, overflow and underflow scenarios, error scenarios, and other uncommon cases. With the aid of these tests, it is possible to confirm that the UART DUT can successfully handle a variety of challenging circumstances. Corner cases include, for instance, testing the UART DUT at either the maximum or minimum baud rate, with both the maximum and minimum number of data bits, and with both the maximum and minimum values of stop bits and parity bits. To make sure that the UART DUT handles errors gracefully, it is crucial to test error scenarios like invalid input values or unexpected communication interruptions. You can make sure the UART DUT is durable and dependable under a wide range of operating conditions by testing these edge cases.

5.8 Coverage Report Generation

To make sure that the verification team has adhered to the objectives and specifications of the test plan, a coverage report is an essential tool in UVM verification. The functional and protocol coverage that the test cases were able to achieve is summarized in the coverage report. The verification team can use the coverage report to pinpoint areas where the test cases might not have satisfied the desired coverage objectives. After identifying these areas, the verification team can then develop new test cases to fill in the gaps. The percentage of code that has been used by the test cases is known as functional coverage. To measure the functional coverage of the UART DUT's required testing, the verification team can define functional coverage points. The UART DUT is guaranteed to comply with the required communication standards by protocol coverage. Measured by protocol coverage points are particular facets of the UART protocol that the verification team must check. The functional and protocol coverage goals and the degree to which the test cases have achieved them should be summarized in the coverage report. Any components of the UART DUT or UART protocol that were not sufficiently covered by the test cases should be noted in the coverage report. The verification team can subsequently develop new test cases to fill in these gaps and increase the overall test coverage.

5.9 Results:

The next step is to review the test results and troubleshoot any issues or errors that were discovered after creating the coverage report. Waveform viewers, log files, and debuggers are just a few examples of the tools and methods that can be used for the analysis. Finding the problems' underlying causes and fixing them are both parts of the debugging process. It might be necessary to change the DUT code or test cases. These tests are the overall accumulation of all the tests performed as mentioned above on the transmitter and the receiver side of the design. These tests are combined in the waveform to provide an imperative idea of the operations of the protocol. In individual tests conducted or the waveforms would not be feasible under general supervision.

For the implementation of this project, we have used Quartus Prime 18.1 for the design and to get the FSM for both the transmitter and the receiver and to check the validity of the code. The design is verified on edaplayground and we used synopsys vcs compiler to compile and used UVM 1.2. For the physical

design part, we have used openLane which is on top of the Windows subsystem for Linux (WSL 2). We have also used docker version 2.36, git version 2.36, python 3. WSL has ubuntu 2.

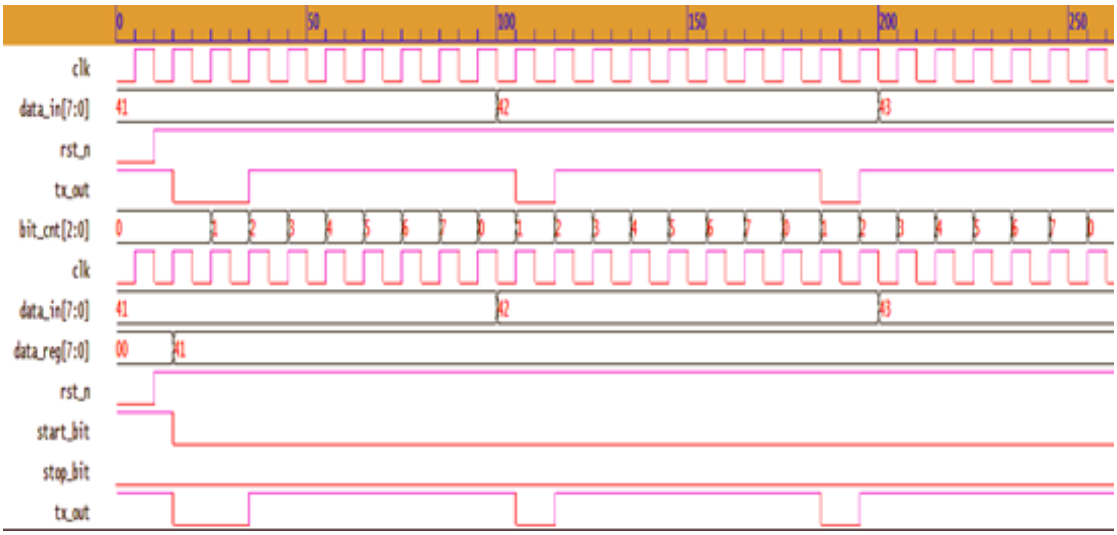

Figure 14: Transmitter Simulation

Figure 14 depicts the way uart design reacts to the stimulus provided by the testbench. The UART TX (transmitter) begins transmitting data when data are written to it. A start bit, which is always sent by the transmitter at a low logic level to indicate the beginning of a new data frame, is sent first. Following that, it sends the data bits one at a time, beginning with the least significant bit (LSB) and concluding with the most significant bit (MSB). In most cases, depending on the configuration, there are 7 or 8 data bits. The transmitter optionally sends a parity bit for error detection after the data bits to help with transmission. It is based on the data bits to calculate the parity bit, which can be even, odd, or none. As a final signal that the data frame has come to an end, the transmitter sends a stop bit, which is always sent at a high logic level. The UART TX responds to a number of events that occur during transmission, including receiving new data bytes, completing data frames, and spotting errors. Interrupt creation, error reporting, and state modifications are just a few of the possible responses. As an illustration, when the transmitter receives a fresh data byte, it buffers it and waits for the subsequent transmission cycle. The transmitter signals the end of transmission and awaits the arrival of the following data byte after completing a data frame. The transmitter reports any errors it finds and tries the transmission again when it finds one, such as a parity or framing error. The clock signal is generated inside the testbench for reference using the baud rate calculations provided in the report where data_in is an 8-bit data coming into the transmitter end of the design. The device defined here is an active low reset design so as soon as the rst_n signal is de-asserted, the design accepts the incoming data and processes it further. Then the data is sampled and shifted and takes 8 clock cycles depicted by a variable bit_cnt in the waveform to keep track of whether the data coming in is in multiples of 8 or not which is the required data frame in the design and then stores it in the data_reg in the design. Whenever there is a out on the transmitter end, the tx_out signal is asserted and de-asserted respectively to match the transmission and processing of the design. The start and stop bits show the starting and stop of the transmission in the design. This snippet is a partial waveform and has a stop_bit signal asserted at the end of the simulation.

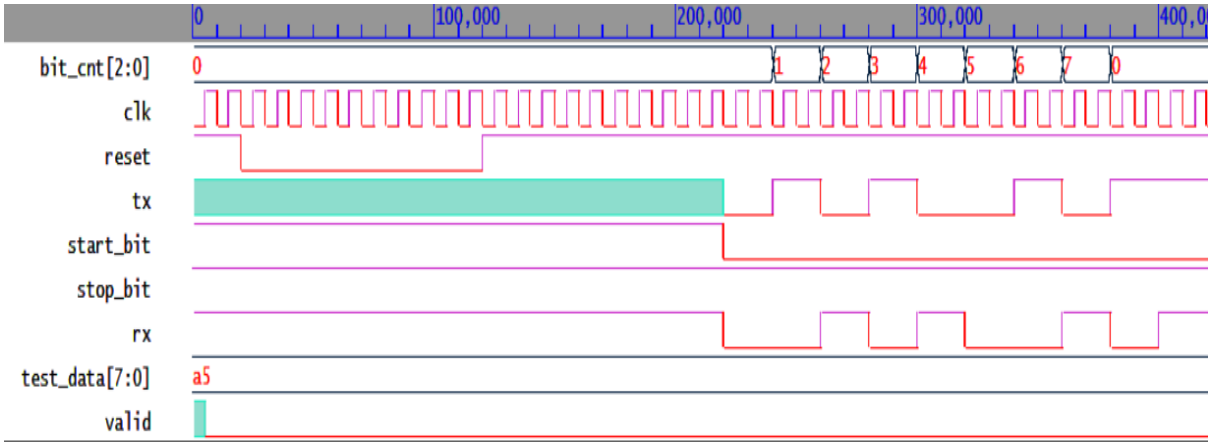

Figure 15: Receiver Simulation

Figure 15 explains the receiver waveform. The clock and reset signals, as well as the signals for sending and receiving data, are defined at the beginning of the code. Additionally, it specifies the validity and signals for the data received. The correct signal connections are then made when the DUT is first created. There are several parameters that are defined, including the simulation time, baud rate, number of data bits, and stop bits. The testbench variables and signals are initialized, and the test data to be transmitted is specified as a vector with the logic value [7:0]. The testbench variables and signals are initialized after the testbench clock is created using an "always" block by the code. Prior to sending the test data, the DUT is reset and given time to stabilize. By using the start bit, data bits, and stop bits signals to create the defined baud rate, data bits, and stop bits signals, the transmission of the data is made possible. After comparing the transmitted and received data, the transmitted data is declared valid. If the transmitted and received data are identical and the data is marked as valid, the test is successful. Upon reaching a certain number of clock cycles, the simulation is over.It's critical to have a thorough understanding of the design and testbench before you can debug something effectively. This entails being aware of the timing and order of signals, the expected DUT behavior, and the operation of the UVM components. To work effectively with other team members who may be in charge of various facets of the design or testbench, it is also critical to have strong communication and collaboration skills. The tests should be rerun to ensure that no new problems have been introduced by the changes after the problems have been located and fixed. To reflect any additional coverage that the modified tests may have produced, the coverage report should be updated. Until the desired coverage and functionality objectives are met and the UART DUT is prepared for integration into the larger system, this iterative process is continued.

6 UART Physical Design

The Physical design,i.e. The backend is also implemented for the UART design using OpenLane. Physical design simply means generating the GDS format of the layout from the RTL code which then can be used by the foundry, for example, TSMC or IFS for chip fabrication. Generating GDS has multiple intermediary steps. It starts with synthesis, then partitioning, floorplanning, placement, clock-tree synthesis, routing, timing closure, and then sign-off. All the mentioned steps have their inputs and outputs and their intermediary steps as well. Upon sign-off, the GDS of the design is generated. Proper implementation of the physical design of a particular design yields better performance, area, power, and ultimately cost which directly impacts the sale of the chip for that company. A product or system's physical design makes sure it performs well and accomplishes its intended function. Performance, reliability, and efficiency are directly impacted by factors including component placement, interconnections, and layout optimization. A well-thought-out physical design facilitates the smooth fusion of many components, maximizing functionality and improving the overall user experience. It is fundamental to producing structures and goods that are usable, aesthetically pleasing, safe, efficient, and sustainable. It combines technological know-how with aesthetics, usability, and manufacturing concerns to produce successful results that satisfy customer wants and market expectations.

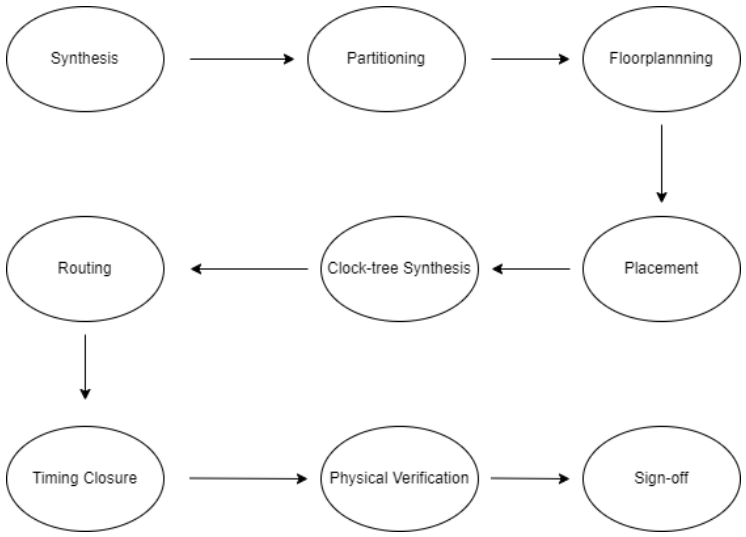

Figure 16: Physical Design Flow

Figure 16 explains the backend flow. In this project, the backend implementation is done using OpenLane, which is an open-source tool for RTL to GDS flow. The flow starts with the synthesis. It takes inputs such as HDL files (.v, .sv), Libraries, and constraints (.sdc) and generates a netlist as well as for, area, and timing reports. Here, we have used Skywater 130 nm library. Then, the partitioning takes the outputs which synthesis generates along with physical libraries (.lef file) to get the portion of the design. The floorplan takes the inputs and generates a .def file. Then the placement stage places the cells in their unique position as per the floorplan. The clock tree is then generated using RC delay models. Then the timing tool performs a timing analysis of the design and generates a timing report. After that, the DRC and LVS are performed to see the violations of the design and then finally, GDS is

generated. OpenLane has a separate tool for every stage. Synthesis uses Yosys and ABC, floorplan uses Placer and PDN, and placement uses RePLace, Resizer, and OpenDP. Clock tree synthesis uses TritionCTS, Routing uses FastRoute, and tools like Magic, and Klayout are used for checks and GDSII generation. The flow also performs DRC, LVS, and Antenna checks and the circuit validity checker. These checks are related to the foundry rules, for example, the minimum spacing between 2 nets, the thickness of the metal layer, etc. The flow also generates logs and reports which are used to analyze the performance of the design, For example, the timing reports tell the slack value. The placement report tells the utilization of macros and standard cells. The routing stage gives the congestion report which tells the amount of congestion between 2 metal layers. These metrics are used to analyze the design performance. The OpenLane architecture is shown in the figure below.

Synthesis:

In the VLSI design process, synthesis occurs between the RTL Design & Verification and Physical Design phases. The conversion of one notion level into another is the definition of synthesis. Front-end engineers write the code for RTL Design in a variety of languages, including Verilog, system Verilog, VHDL, etc., and then validate the codes (verification stage). Once the RTL code has been validated, we move on to the Synthesis step of VLSI Design. We now translate logic codes into the circuit. Numerous goals are served via synthesis. One of them is getting a Gate-level Netlist. Clock gate insertion, DFT logic insertion, and logic optimization are additional crucial goals.

Table 2 QoR report metrics of synthesis stage.

Metric	**Number**
Number of wires	315
Number of wire bits	404
Number of public wires	18
Number of public wire bits	68
Number of memories	0
Number of memory bits	0
Number of processes	0
Number of cells	371
Chip area	2772.659200

Table 2 lists various metrics which are observed in various QoR reports of the synthesis stage. These numbers are useful when an improvement happens and the designer then compares a better and optimized netlist to the previous. Sometimes, the front-end team makes an addition to the Verilog code, so the new netlist can be compared to the previous one to see the new cells added by the tool because of the changes in the Verilog code. The synthesis stage also performs a rough timing analysis for the designer to visualize the timing. This operation is not a very good estimation but gives the designer an idea of the improvement the front-end team can make to achieve a good slack. It has the following information:

Table 3 Power report of the synthesis stage.

Type of group	Power (microwatts)	Percentage
Sequential group	202	68.1 %
Combinational group	94.6	31.9 %
Macro	0	0%
Pad	0	0 %
Total	297	100 %

Table 3 lists various power related metrics of the synthesis stage. The sequential group (flips flops and registers) contributed 68.1 % of the whole power which is 202 microwatts. This is because the clock switching generally takes more consumption that the logical or combinational circuits which contribute 32 % ot 94.6 microwatts of power. The total power consumption is roughly 297 microwatts of the entire design.

Floorplan

The key to a successful physical design lies in the floor plan, which determines the layout of an ASIC design. By carefully considering the arrangement of I/O pads, macros, power, and ground structures, an optimized ASIC design with improved performance can be achieved. It is crucial to ensure that the data used for the physical design is properly prepared before proceeding with the actual floorplanning process. Throughout the entire physical design workflow, iterative modifications of the representation are made, with each step involving the construction and evaluation of a new ASIC representation. These iterative refinements are necessary to meet the requirements of the system. Physical designers often encounter issues related to breaches of physical design rules during ASIC design verification. If such breaches are identified, the physical design processes must be redone to gradually enhance the routing procedures and meet the design timing requirements. Effective floor planning significantly impacts an IC's performance. By strategically placing components and functional blocks, designers can reduce signal delays, minimize distances between critical routes, and improve overall timing characteristics. Proper floor planning allows for optimization of routing, signal integrity, and power distribution, resulting in increased speed, reduced power consumption, and improved overall performance. Moreover, it enhances overall production efficiency, shortens time to market, and facilitates testing and debugging during manufacturing and testing stages. This, in turn, ensures the long-term reliability and functionality of the IC by enabling efficient power delivery, minimizing power usage, and preventing overheating.

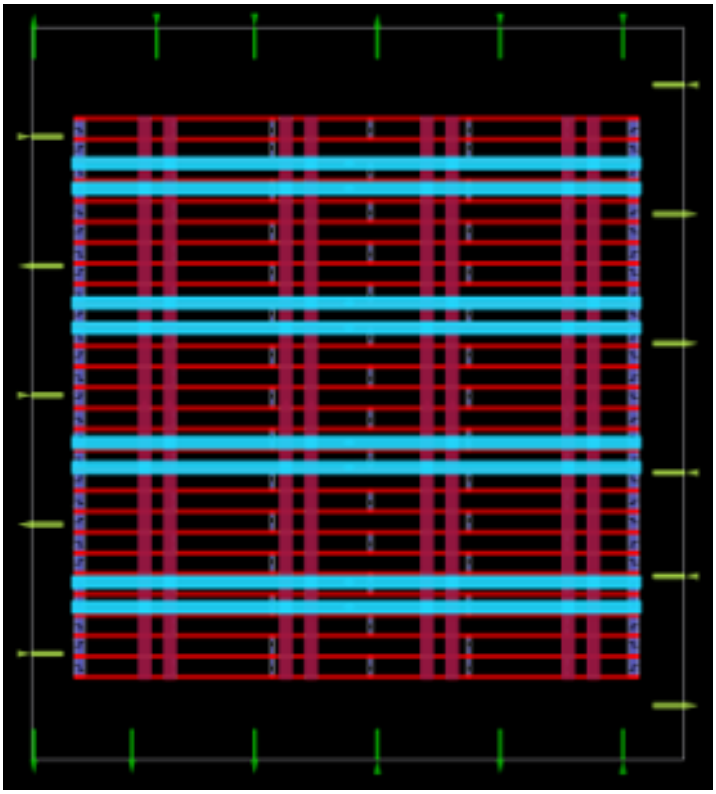

Figure 17: Floorplan Layout

Figure 17 shows the output layout of the floorplan stage. The yellow lines on the boundary are the input pins of the chip. The green pins are the output pins of the chip. The red and the blue lines are Vdd and the ground layers. They provide specific voltage levels to the cells. The purple line between 2 metal layers is the tap cells which are used to block the placement of particular cells.

Placement:

Placement is a crucial step in the physical design process, involving the strategic positioning of modules within the designated area to optimize performance and minimize wire length. The primary objective is to achieve timing closure, ensuring that the circuit meets its timing constraints. By placing critical lines and related components close together, placement reduces clock domain skew and minimizes signal delays. This leads to improved timing constraints and simplifies the overall clock distribution network, resulting in enhanced performance and the attainment of desired frequency targets. Placement is an iterative process integrated with other physical design stages such as floor planning, routing, and optimization. To strike a balance between various design constraints and achieve the best results, multiple iterations of placement are often necessary. Designers can explore different placement options and make adjustments based on time, power, and area requirements until an optimal solution is found. The placement directly impacts the utilization of chip area, and efficient placement aims to satisfy all design constraints while minimizing the overall chip area required. By optimizing the arrangement of components and macros, placement maximizes the utilization of available space, leading to a more

compact and cost-effective design. Effective area utilization enables higher chip density, reducing manufacturing costs and allowing for the integration of additional functionality within a given chip size. Furthermore, placement plays a role in reducing electromagnetic interference (EMI), crosstalk, and signal distortion by minimizing the length and impedance of interconnects between components. This contributes to accurate data transmission and reception while reducing the likelihood of errors or signal integrity issues, thereby ensuring reliable signal quality.

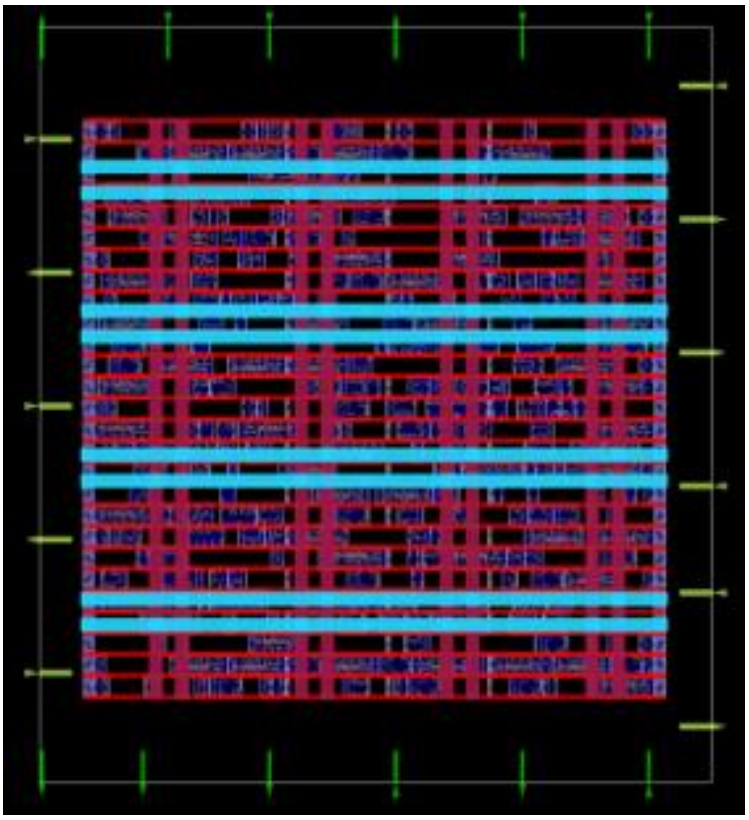

Figure 18: Placement Layout

Figure 18 shows the output layout of the placement stage. The yellow lines on the boundary are the input pins of the chip. The green pins are the output pins of the chip. The red and the blue lines are Vdd and the ground layers. They provide specific voltage levels to the cells. The purple line between 2 metal layers is the tap cells which are used to block the placement of particular cells. The little purple blocks in between the lines are the standard cells that are placed by the tool. Some tools also place filler cells which are used in case there is a modification of any addition to the RTL code.

Routing:

Routing is a crucial step in chip design where the previously placed components are connected using wires to establish physical pathways for data flow. It directly affects the timing characteristics of the design. By eliminating signal delays and optimizing route paths, routing contributes to achieving timing closure. Techniques like clock tree routing and skew management are employed to ensure that critical paths meet timing specifications and enable the desired operating frequency and performance. Efficient

routing facilitates the best possible utilization of the IC's area. It accomplishes this by strategically planning and arranging interconnects, which reduces overall wire length and the chip area required. This compact design allows for more cost-effective chip fabrication and higher integration densities. Proper routing is essential for maintaining signal integrity within the IC. It minimizes signal distortions, delays, and electromagnetic interference (EMI) by maximizing routing paths and considering factors such as cable length, impedance management, and noise coupling. Routing significantly influences the distribution of power and ground signals across the IC. By effectively routing power and ground nets, it reduces power supply noise, voltage dips, and coupling effects. This ensures a steady and reliable power distribution to all components, preventing power-related issues and maintaining proper operation. Employing appropriate routing strategies addresses manufacturability concerns such as wire density, lithography limitations, and viable via locations. Routing ensures that the layout adheres to design guidelines and restrictions, making it manufacturable and allowing for proper and reliable manufacturing processes

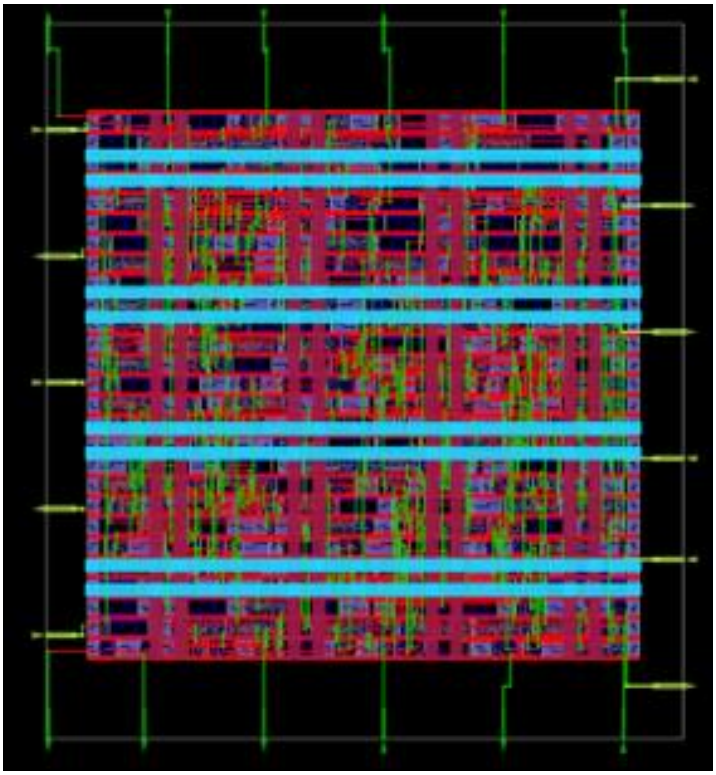

Figure 19: Routing Layout

Figure 19 shows the output of the routing stage. The lines connecting 2 standard cells are the metal layers. There are 5 metal layers and each layer is connected to a nearby layer with vias. Just like 2 floors of a building are connected via pillars, similarly, all the layers are connected through vias.

Clock-Tree Synthesis:

In the process of designing and optimizing clock distribution networks for integrated circuits (ICs), Clock Tree Synthesis (CTS) plays a vital role. Its primary objective is to ensure effective and reliable delivery of clock signals to all sequential elements, such as flip-flops, within the IC layout. Clock signals are necessary to synchronize the operation of consecutive elements in an IC. CTS is crucial for achieving timing closure as it enhances the efficiency of the clock distribution network. By minimizing clock skew, which refers to variations in the timing of clock signals as they reach different locations, CTS ensures that the clock reaches the flip-flops within their designated setup and hold time windows. CTS enables the fulfillment of timing constraints, allowing the IC to achieve the desired operating frequency and performance by ensuring balanced clock routes and minimizing clock skews. In addition to timing considerations, CTS also contributes to power efficiency in IC designs. By optimizing the routing paths of clock signals and considering factors such as wire length, capacitance, and power supply noise, CTS reduces power consumption. This is achieved by minimizing unnecessary switching activity and power dissipation in the clock network. The result is improved power efficiency and overall reduced power consumption.

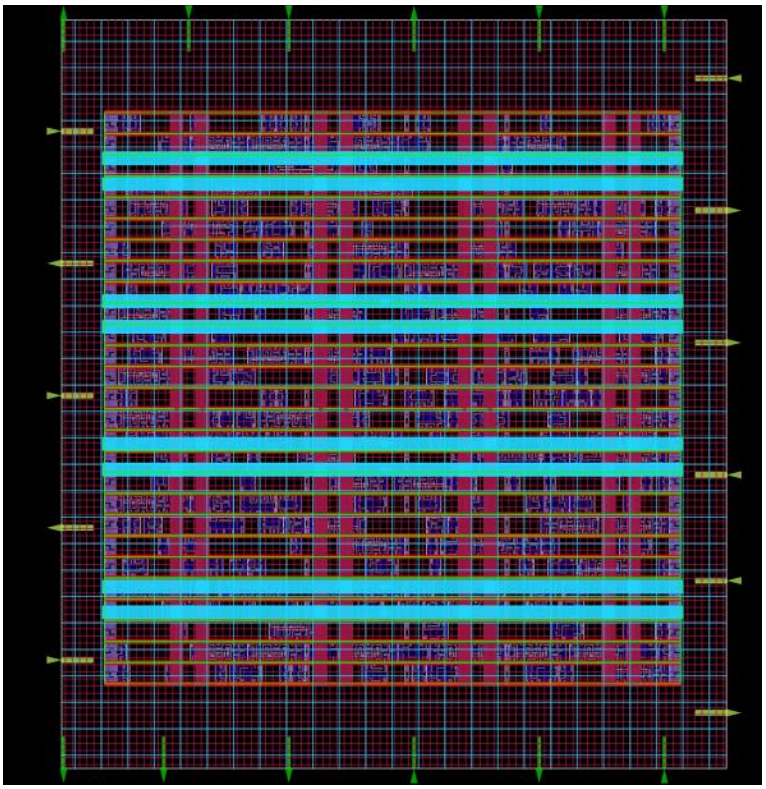

Figure 20: CTS Layout

Figure 20 shows the layout of the cts stage. In the CTS stage, the clock network is generated throughout the design. As the UART is asynchronous, the baud rate is something that keeps the transmitter and receiver synchronized. Here, the baud rate is generated from the clock using a counter. The figure below shows the zoomed picture of the CTS output which is the metal tracks along the routed paths.

Physical Verification:

For the physical verification, the flow generates a manufacturability report that has DRC and LVS information. There are no LVS violations, so the design is LVS clean, but there are 27256 DRC violations that can be cleaned and brought further down. Typically, the DRC violations are in the 10,000s but can be lowered when timing improvement is done. Also, the design is antenna clean, which means there are no antenna violations. Some of the other violations are mentioned in the table.

Table 4 DRC violations

Type	**Number**
Triton route violation	0
Short violations	0
Metal spacing violations	0
Off-grid violations	0
Minimum hold violation	0
Magic violations	0
Pin violations	0
Net violations	0
Lvs total violations	0
Cvc total violations	1
Klayout violations	4549
Total	**4550**

Table 4 explains the various design check and rules that are part of the physical verification stage. As per the table, the design does not violate the rules apart from the CVC and the klayout violations. The minimum width and spacing between various layout elements, such as metal lines, contacts, or vias, are checked by CVC. It marks them as violations if these dimensions are less than the set design guidelines. KLayout checks to see if the minimum distance between layout elements like wires, vias, or pads is being met. KLayout checks to see if the area complies with the requirements by looking at the density of the features there. If the density is either too high or too low, it can signal infractions. To make sure layout features, like metal lines or transistors, adhere to the established standards, CVC evaluates their density. If there are too many or too few features in a specific region, density violations may happen.

Signoff:

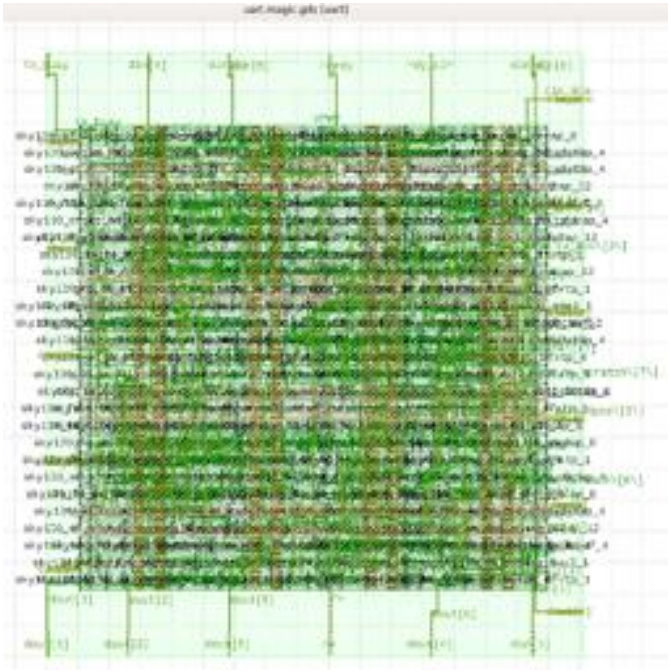

Figure 21: GDS Signoff

Figure 21 shows the final GDS picture of the design. It contains the shape, size, and location of every cell along with the routing. The green lines are the metal layers and the square shapes are the interconnections between them. The figure below shows the zoomed picture of the GDS with the metal interconnections between two layers. This file is used by the verification tools and the foundries example Intel Foundry Services or TSMC for chip fabrication.

7.Conclusion:

In this paper, a thorough and methodical methodology was used across all stages of the design process, , RTL design, verification, synthesis, and physical implementation, to guarantee a solid and dependable design. The RTL design stage had the translation of the specifications into Verilog code. The verification stage was important to validate the functionality of the design. The physical design stage was important to visualize and look for the metrics to optimize the current design. In conclusion an ASIC design cycle is implemented in this project to demonstrate the importance of each and every step of the cycle right from RTL to GDS generation. The future work includes optimizing the design at RTL level, maybe including pipelining. This could make the design work at greater frequencies. Also, adding more test cases. Finally, optimizing the design at the backend level to reach a more optimal value of performance, power and area.

8.References:

[1] Xi Jianbo, Xia Jijin, Luo Chuanhui, Deng Qingyong, and Zhu Peng, “Verification method of FPGA universal configurable UART protocol based on UVM”, Date published: 06/29/2016.

[2] Bidisha Kashyap and V Ravi, “Universal Verification Methodology Based Verification of UART Protocol”, Date published: 05/12/2020.

[3] M Srinath and Sujatha Hiremath, “Verification of Universal Asynchronous Receiver and Transmitter (UART) using System Verilog”, Date published: 07/07/2022.

[4] Vivekananda T and Mahesh Kumar N, “Design and Verification of the UART and SPI protocol using UVM”, Date published: 09/2022.

[5] B. Priyanka, M. Gokul, A. Nigitha and J.T Poomica, “Design of UART Using Verilog And Verifying Using UVM”, Date published: 03/19/2021.

[6] Yamini R and Ramya M V, “Design and Verification of UART using System Verilog”, Date published: 06/05/2020.

[7] Kumari Amrita and Avantika Kumari, “DESIGN AND VERIFICATION OF UART USING VERILOG HDL”, Date published: 01/2018.

[8] W.Elmenrwicb, M.Delvai, Time Triggered Communication with UARTs. In Proceedins of the 4’h IEEE International Workshop Communication Systems, Aug.2002.

[9] Nithin Patel, Naresh Patel, . VHDL Implementation of UART with BIST Capability, Fourth International Conference on computing, Communications, and Networking Technologies, pp 1-5, July, 2013.

[10] Kashyap B, Ravi V. Universal Verification Methodology Based Verification of UART Protocol. In Journal of Physics: Conference series 2020. Dec 1 (Vol.1716, No. 1, p. 012040).

Final_paper.pdf

by Purav Bhatt

Submission date: 15-May-2023 11:54AM (UTC-0700)
Submission ID: 2093993856
File name: 0515-34084-1yejvh2_attachment_7310077020230515-34084-1bqzxt1.pdf (897.33K)
Word count: 10293
Character count: 54519

TABLE OF CONTENTS

1) Abstract…..…….…… 2

2) Introduction…….…….....…. 2

3) Literature review……...…..…….…..….. 4

4) UART RTL Design……..……...… 5

5) UART Verification……. 11

6) UART Physical Design……..…… 21

7) Conclusion…….. 29

8) References…….. 30

RTL to GDS Implementation and Verification of UART using UVM and OpenLane

Dharmik Vijay Joshi: dharmikvijay.joshi@sjsu.edu; Purav Bhatt: purav.bhatt@sjsu.edu

Shrikant Jadhav: shrikant.jadhav@sjsu.edu

1 Abstract

The paper presents an ASIC design flow implementation for Universal Asynchronous Receiver and Transmitter (UART). It starts with the RTL implementation using Verilog HDL. It's then verified using UVM, including test benches, sequences, and monitors. Finally, the paper discusses the use of OpenLane for the physical design implementation of the verified design from synthesis up to GDS (Graphic Data System). The goal is to demonstrate how a sophisticated digital circuit is designed and verified. The results of the verification are shown via waveforms for test cases. The results of the backend implementation include layouts and QoR (Quality of Results) report tables. The emphasis is also on design rule checking (DRC) and layout versus schematic (LVS) to ensure accuracy and manufacturability. The paper also mentions the difficulties encountered while working on the project and offers suggestions for future developments.

2 Introduction

With the increase of IoT applications, HPC, and Big Data solutions, it becomes quite crucial to have a communication protocol to match those demands. A reliable, reusable, and plug-and-play solution can help with the sharp rise in data communication. In real-time systems, a communication protocol becomes center stage for peripherals to interact with the processor so that the propagation delay would be as low as possible. If the communication is not reliable, the real-time system would essentially be a failure. For serial communication through a computer or a peripheral device, UART is often an Integrated Circuit (IC).

Figure 1 explains the functionality of UART. 2 modules are connected in such a way that the transmitter of one is connected to the receiver of the other. UART1 transmits data and UART2 receives data. Today, UARTs are frequently seen in microcontrollers. The universal asynchronous receiver/transmitter (UART) receives the data as bytes and sends the bits sequentially. The received bits are put back together into whole bytes at the destination side using a second UART. A shift register is present in each UART and is essentially used to translate serial data into parallel data. Compared to parallel transmission across several wires, serial transfer of digital information (bits) using a single wire or other media is less expensive.

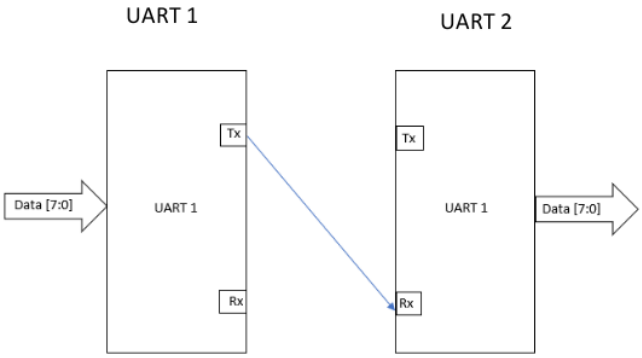

Figure 1: Traditional UART communication

The paper encompasses the whole ASIC design flow. The ASIC (Application-Specific Integrated Circuit) design pipeline is a structured and systematic process used to create custom integrated circuits tailored for specific purposes. It involves multiple phases, starting from concept and specification and extending to manufacturing and testing. The initial stage focuses on establishing the criteria and specifications for the ASIC, including its intended function, desired functionality, and performance objectives. In the architectural design stage, the overall organization and structure of the ASIC are determined. This entails selecting the system design, dividing the functionality into components, and defining their interactions. The next step is the register transfer level (RTL) design, where the high-level architectural description is transformed into a digital representation using hardware description languages (HDLs) like Verilog or VHDL. This involves designing and describing the behavior and connections of individual digital components. Once the RTL design is completed, it undergoes rigorous functional verification to ensure that it operates as intended. Techniques such as hardware emulation, formal verification, and simulation are employed to verify compliance with the requirements. In the physical design stage, the gate-level representation is transformed into a physically feasible layout. Tasks like floorplanning, placement, clock tree synthesis, routing, and optimization are performed to meet time, power, and space constraints. Once the final layout database is created, containing all the necessary design data, it is ready for manufacturing after it has been thoroughly reviewed and satisfies all relevant requirements. The manufacturing process involves sending this database to a semiconductor foundry for fabrication.

The verification process plays a critical role in the development of VLSI chips. It involves running numerous test cases to ensure that the design functions as intended. Verification and performance measurement of integrated circuits are demanding tasks that require significant time and effort. As the complexity of the device design increases, verification becomes even more challenging. Traditional verification approaches often lack reusability, and their time to market is typically lengthy. The Universal Verification Methodology (UVM) library addresses these limitations by providing reusable components and testing environments using System Verilog. UVM is a widely adopted methodology for verifying integrated circuit designs, offering advantages such as reusability, scalability, and interoperability. Employing UVM enables the realization of these qualities and brings multiple benefits to the verification process.The physical design of the UART (Universal Asynchronous Receiver-Transmitter) is facilitated using the OpenLane toolsets. OpenLane is a leading open-source application

for semiconductor digital design. It aims to promote open access, expertise, rapid innovation, and accelerated design turnaround by eliminating barriers related to cost, risk, and uncertainty in hardware design. OpenLane offers a flexible flow control by providing an API with Tcl and Python bindings. It is an integral part of open-source digital design flows, exemplified by OpenLane-flow-scripts.

In this paper, the physical design is also demonstrated. Physical design plays a vital role in creating a wide range of products, systems, and structures, encompassing architecture, industrial design, integrated circuit layouts, and urban planning. Its purpose is to transform intangible requirements, ideas, and concepts into tangible and practical forms. The goal of physical design is to optimize the placement of elements and physical features to achieve effectiveness, efficiency, and user satisfaction. Partitioning is a crucial step in physical design, where a complex circuit is divided into manageable subsystems or sub-blocks with minimal interconnections. During this phase, the shape and approximate location of each block on the silicon chip are determined. The placement of pins for each block is also decided, ensuring easy routing of connections between these blocks in the future. Placement involves precisely positioning the sub-blocks to allow sufficient space between them for routing Vdd and Ground supply lines, as well as connector wires. Static timing analysis is then performed on the final routed netlist, which represents or simulates the chip's actual layout to validate its timing performance. To ensure the design's functionality remains unchanged, Layout Vs Schematic (LVS) equivalency verification and physical design implementation Layout should be conducted. These steps confirm that the design's intended functionality is preserved throughout the physical design process.

3 Literature Review

The verification approach can be followed to verify the UART protocol using UVM. This verification method includes setting up a UART configuration type in which all the UART protocol parameter information is packaged and sent the UART protocol parameter information to related platform parts like a driver, a monitor, and a grade recording board through a config-db mechanism provided by the UVM [1]. This process involves finishing an overall framework of UART protocol verification using a UVM verification platform structure and a verification idea. This article signifies the importance and the need for UVM to verify the UART protocol as it can make the verification of a block simpler and more efficient concerning other conventional verification methods [2]. It also gives a detailed overview of how UART works and how the verification environment should be set up to check for communication with the same. The purpose of this study is to use SystemVerilog to develop a UART and a UVM-based testbench to validate the design. One of the most popular serial communication protocols that can be used without a clock stimulation is UART [3]. The Baud rate generator, control, bus interface, interrupt control, and receiver-transmitter FIFO blocks are all included in the architecture [4]. System bus parallel data are serialized, transported, and converted back to parallel data at the receiving end. A reusable and adaptable verification environment reduces complexity and processing time. This paper details the SV hardware description language design for the UART and SPI functional modules. The Universal Verification Methodology is used to undertake functional verification of the UART and SPI [5]. By comparing the collected response to the intended response via the scoreboard mechanism, the reusable UVM testbench architecture is meant to push the randomized stimuli to the unit under test to assess the6functional correctness. The overall execution time will be shortened by the stimulus's reusability [6]. This paper describes the design of a UART in Verilog and UVM verification. This paper's verification demonstrates the full functionality of UART, which may be validated using a UART device. The UVM-based verification will cut down on the total amount of time the logic gates need to operate. The overall execution time will be reduced since the tests of sequences are kept separate from the original test bench hierarchy and the reusability of stimulus [7]. This paper's primary goal is to use

System Verilog to design and test a full duplex UART module (SV). It is a serial communication protocol that allows devices to communicate without the use of a clock signal. It sends parallel data that has been converted to serial format [8]. Data that is received in serial format is transformed into parallel format. The baud rate generator, receiver, transmitter, interrupt, and FIFO modules are all designed as part of the UART design process. Verification entails checking the design by establishing a verification environment that permits testbench reuse and minimizes code complexity. test the difficult-to-reach corner situations, randomization is applied [9]. 100% functional coverage and 100% assertion are attained. The design and verification of UART are the subjects of this study . UARTs are now included in microcontrollers quite frequently. A UART is typically an exclusive integrated circuit (IC) or a component of an IC that is used for serial communication on a serial port on a computer or peripheral device [10]. It works flawlessly when connected to a PC's serial port for data transfer with customized electronics. Today, UARTs are frequently included in microcontrollers. A full duplex transmitter or receiver is a UART.

The paper highlights the architecture that uses the shift register for both the transmitter and the receiver. The transmitter has FSM that has 5 stages, which makes the data-frame which is of 10 bits. Also, the receiver has 3 stages which are used to read the data-frame and catch the parity bit for errors. The paper also demonstrates the implementation of the physical design part which lately has not been done as part of the full ASIC design flow. The flow stops at verification or validation on the FPGA board. But the backend implementation is not performed which can be very useful to optimize the design as per power, area and performance.

4 UART Design

The UART design has a transmission FIFO, receiver FIFO, a couple of shift registers, a baud rate generator, a control unit, and a wishbone interface. The baud rate generator produces various baud rates for transmission. It then gives that information to the control unit which then produces various reads, writes, and other control signals. The shift register is used to link the data packet generation to the FIFOs. The BRG is a flexible and programmable unit that produces a baud clock. The shift register helps to hold the data packet by adding parity bits and BRG bits. The FIFO helps to transmit and receive the data serially. The control unit helps to know when to read or write data. The IP core has a wishbone interface which helps to make the design more industry-standard. Also, the design has a register bank, especially for the control registers and interrupt registers.

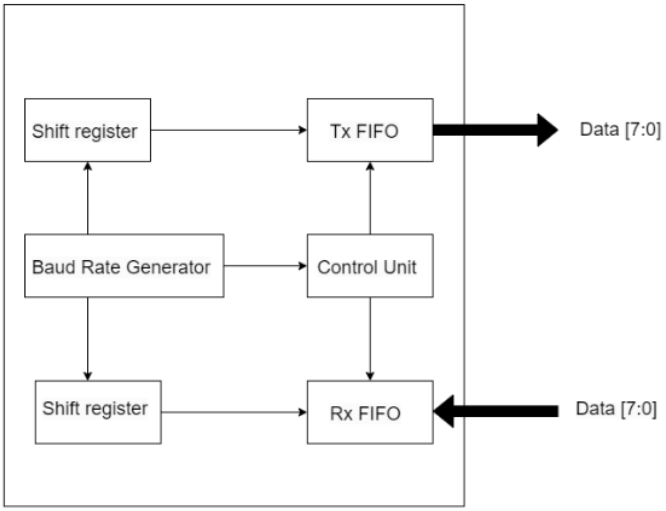

Figure 2: Block diagram overview of proposed UART

Figure 2 explains the block diagram of UART. The baud rate generator develops the baud rate as per the user. The control unit is essentially a finite state machine that changes the control bits for both the transmitter and receiver. The shift register essentially is used to roll in data bit by bit. The FIFO is used to streamline the data. In asynchronous communication, transmission speed is determined by the Baud rate generator. The 16-bit timer that creates the clocking mechanism is used.

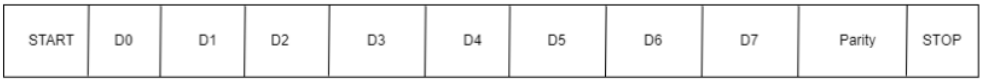

Figure 3: UART data format

Figure 3 explains the data frame of the UART. The Start bit is inserted at the start of a word when it is transferred to the UART for asynchronous transmissions. Therefore, the Start bit is first utilized to signal to the receiver that a word of data is about to be sent, putting pressure on the receiver's clock to match the transmitter's clock. The word of data's constituent bits is delivered after the Start bit, starting with the Least Significant Bit (LSB). An optional Parity bit is then delivered to the transmitter once the data has been sent in its entirety. The receiver often uses this bit to carry out basic error checking. To signal the end of transmission, a Stop bit will be transmitted at the conclusion. A preset sequence is applied by the verification environment to the design, and the outcome is compared. The intended UART module is represented by the DUT. The test bench can be used again because the environment can contain one or more agents. Two distinct DUTs stand in for two UART modules, and each DUT has a separate verification agent. Monitor, driver, and sequencer functions are all contained within the agent. To ensure full duplex operation, two agents are created. To test all the random conditions,

randomization is employed to randomize the sequence. The randomize feature is used to create random test scenarios that will examine all of the functions' hard-to-reach corner circumstances. Since tests for design verification are automatically generated through random testing, it is more effective than other methods.

4.1Baud Rate Generator

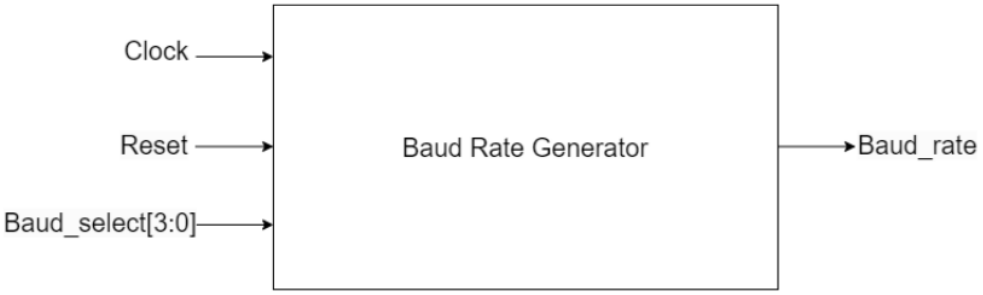

Figure 4: BRG Implementation with variable rates

Figure 4 explains the baud rate generatorThe baud generator on the UART is programmable. making a baud clock. A divisor that is kept in a divisor latch is used by BGR to split an input clock from the processor clock generator (BCLK). 16 baud clock cycles, or sixteen times the baud rate, are used to send data. The baud rate frequency factor can be calculated using the requested baud rate and the system clock frequency (sometimes referred to as oscillator clock). The receiver should accurately reflect the asynchronous serial data. The BRG module has the inputs: Clock, reset, and selection bits for the baud rate. The reset pin disables the operations in the circuit. The module provides 6 different baud rates to the user as per the requirement. These are 4800 bps, 9600 bps, 19200 bps, 38400 bps, 57600 bps, and 115200 bps. As a result, the module provides 4 selection bits to the user. It uses a 32-bit counter that generates various delays so that the user can get a particular baud rate. The 4-bit selection pins are connected to an internal multiplexer to select the baud rate. The UART utilizes a divisor to scale down the frequency of the clock signal to produce the desired baud rate because the frequency of the clock signal is typically significantly higher than the baud rate.

$$Divisor = \frac{Clock\ Frequency}{16\ *\ Desired\ Baud\ Rate}$$

Equation 1

Equation 1 refers to the calculation of the divisor for a particular baud rate. The module provides 6 different baud rates to the user as per the requirement. These are 4800 bps, 9600 bps, 19200 bps, 38400 bps, 57600 bps, and 115200 bps. So, the corresponding divisor is calculated from the previously mentioned baud rates. These divisors are used in the counters to get the baud rate.

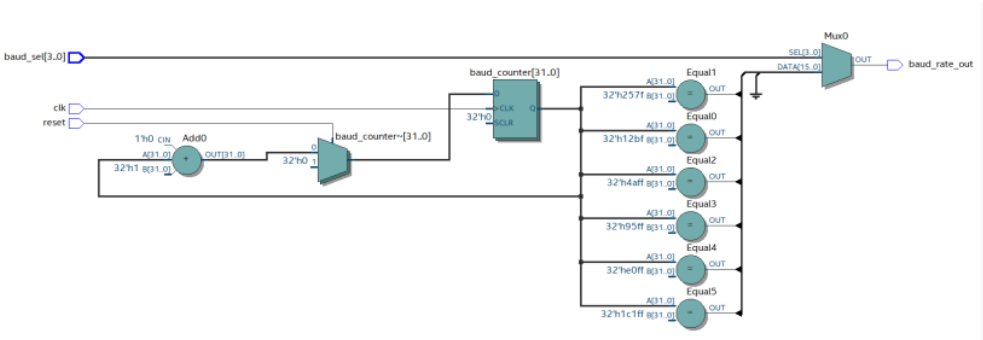

Figure 5: Implementation of Baud Rate Generator

Figure 5 shows the internal circuitry of the baud rate generator. This design is implemented using reference UART models and integrating them according to our needs where the option of variable baud rates and the above-mentioned formula play a vital role in determining the speed of the transmission. The design is generated using the physical design CAD tools available using the reference Verilog module that gets fed to the software to generate the behavioral and pin-compatible circuit.

4.2 Transmitter

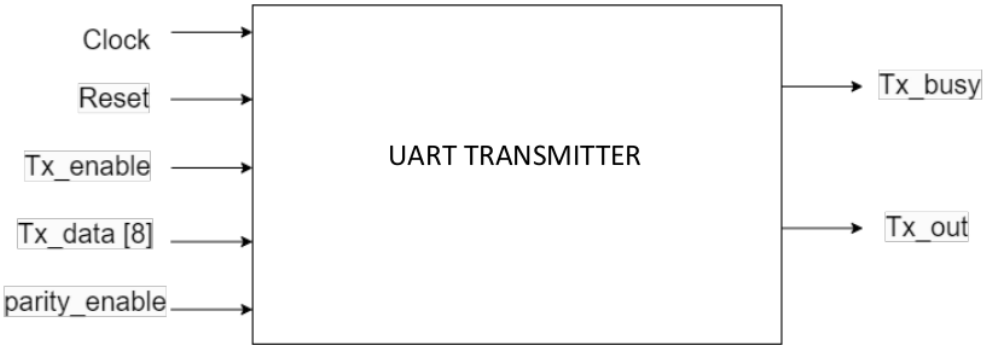

Figure 6: UART Transmitter with defined pins

Figure 6 explains the transmitter block diagram. The transmitter module takes inputs which are: clk, reset, enable, data and the parity enable bit and produces data to transmit and busy bit as outputs. It uses an internal shift register to transmit the data, a counter, that counts the number of bits that are to be transmitted as well as the register for keeping the state machine information. The transmitter is in charge of using a communication channel to transmit data in the form of binary signals, one bit at a time, to a receiving device. It changes the parallel data coming from the microcontroller into serial data that can be sent through a serial port or a bus of communications. The UART transmitter operates in an asynchronous mode, which means that data transmission synchronization is accomplished without the

use of a clock signal. To frame each data byte and guarantee proper synchronization, it instead uses start and stop bits. Many embedded systems and communication protocols depend on the UART transmitter as a key component.

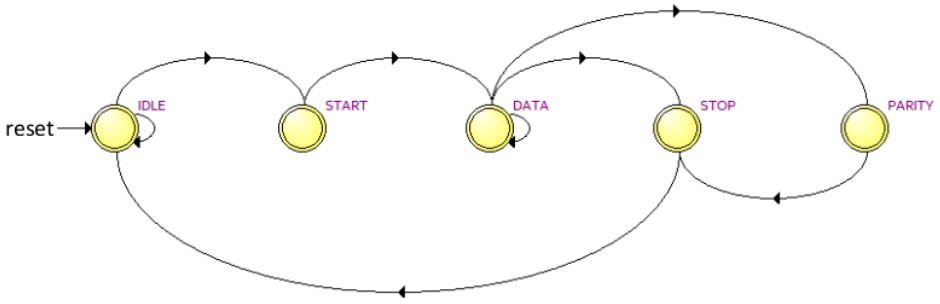

Figure 7: Software-generated Transmitter State Machine

Figure 7 shows the state flow of the transmitter. It has five states, IDLE, START, DATA, PARITY, STOP. By default, it starts with the IDLE state before the Tx_enable signal is switched ON. Then it goes to the START state. Here, the transmitter adds the start bit to the shift register and shifts the data left. Then it goes into the DATA state. Here, the transmitter shifts data left until all 8 bits have been transmitted. If the user does not want to add the parity bit to the data frame, the parity_enable signal is not turned ON and finally, the FSM enters the last stage which is the STOP bit. If the parity_enable signal is turned ON, the transmitter adds a parity bit before entering the STOP bit. Then the data frame is given out as an output from the pin Tx_out. During all the operations are happening, the busy signal is set to 1 to indicate that the next data frame has to wait till the current operations are finished.

4.3 Receiver

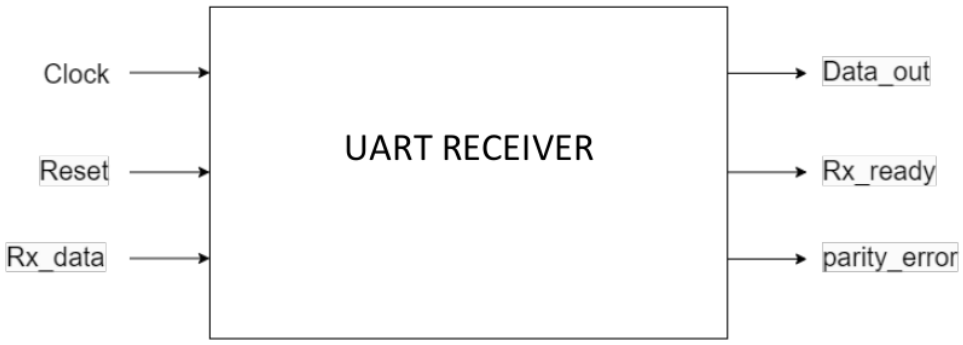

Figure 8: Receiver Implementation

Figure 8 shows the UART receiver. The receiver module takes in 3 inputs and produces 3 outputs. Inputs are: clock, reset, and data, outputs are: data_out, ready, and parity signal. The reset signal shuts off all the operations of the circuit. The clock signal is used to transit from one state to another. It has various internal registers such as shift register, state register, bit-count register, and received-data register. The shift register is used to shift the data frame coming from the transmitter. The state register has information about the 3 states of the receiver. Th bit-count register is used to set as a base for the counter to go through the state frame. The received data register is used to store the data coming from the transmitter. The rx_data signal indicates whether the data is received and stored in the internal register or not. The rx_ready signal indicates if the receiver is ready or not, i.e. if the operations are ongoing, then it cannot accept the next set of data frames. Parity_error signal indicates if the parity error is caught by the receiver or not.

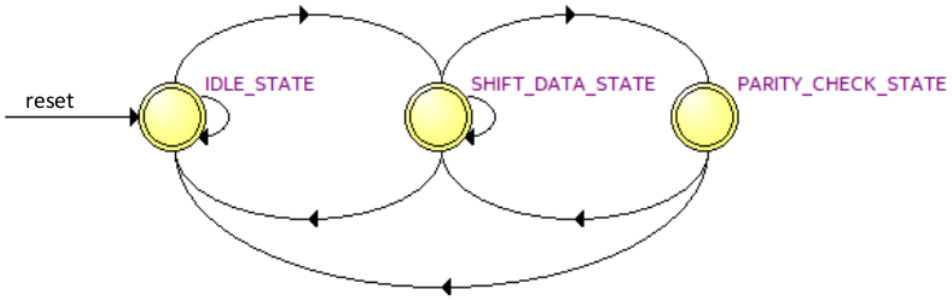

Figure 9: Software generated UART Receiver State Machine

Figure 9 shows the finite state machine of the receiver. It has 3 states namely, IDLE_STATE, SHIFT_DATA_STATE, and PARITY_CHECK_STATE. The state machine starts by waiting for the start bit to be received in the IDLE_STATE. The state machine switches to the SHIFT_DATA_STATE after detecting the start bit, where it inserts incoming data bits one at a time until all data bits and the parity bit are received. The state machine switches to PARITY_CHECK_STATE after receiving all the data bits and the parity bit, where it checks the parity bit for parity mistakes.

5 UART Verification

A popular digital system communication protocol that enables the sending and receiving of serial data between two devices is called UART (Universal Asynchronous Receiver Transmitter). The performance and functionality of UART must be tested in contemporary designs to guarantee that it functions as intended. A common verification methodology for digital designs called the Universal Verification Methodology (UVM) offers verification engineers a strong and effective framework for creating and managing intricate verification environments. Engineers can construct efficient and reusable testbeds that can more methodically and effectively verify the functionality of UART by using UVM. The UVM verification methodology can provide better control over the verification process, improve simulation performance, and enable the creation of complex scenarios for testing the functionality of UART.UVM can be used to verify UART in a way that drastically cuts down on both the time and cost of the verification process while also improving design quality. Early on in the verification process, it can assist in finding and fixing design bugs, resulting in a more effective and reliable design. UVM also enables the development of testbenches that can generate a wide variety of test scenarios, including various corner cases, error conditions, and boundary conditions that can help guarantee the correct functionality of UART under various operating conditions. In order to ensure that UART functions correctly and dependably in digital designs, UVM verification methodology must be used to verify UART.

The steps performed by the team in verifying the UART DUT in UVM can be divided into the following steps:

5.1Develop a test plan

The design's functional, performance, and interface requirements should all be specified in the test plan. Goals for coverage and test scenario identification should also be included. The test strategy should be thorough, encompassing all potential uses of the UART DUT, and it should be specific, including a list of test scenarios, test methods, and anticipated outcomes. The test cases should cover all of the design's functional needs, including data transfer, error detection and repair, and flow control methods. Additionally, the verification environment has to be reusable for multiple test case scenarios in the same integration environment.

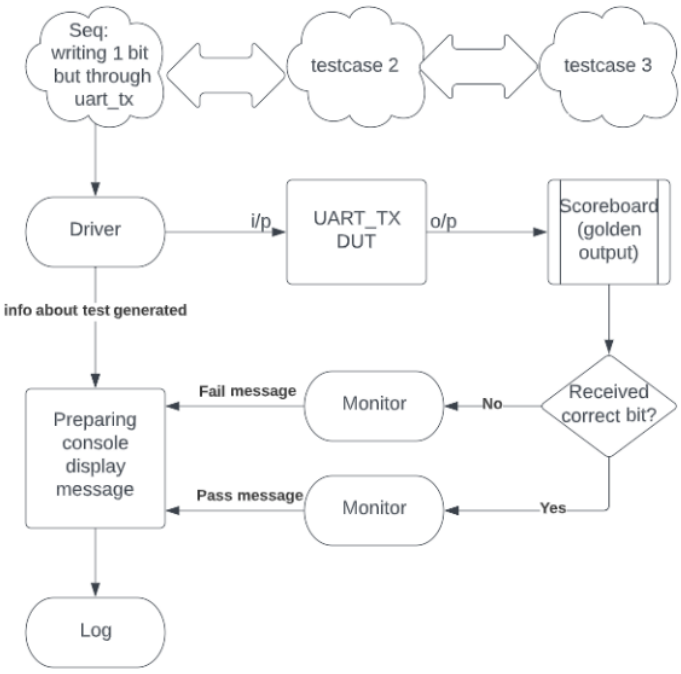

Figure 10: Test Plan Flowchart

The flow chart defined in Figure 10 depicts the verification methodology defined by the team. It is possible to divide the UVM verification of UART into several steps. The first step is to build a UVM testbench that has a scoreboard, a monitor, and a test sequence generator. The test sequence generator generates a stimulus that activates the UART module, which is the design under test (DUT) in this instance. The monitor gathers data from the signals being sent and received at the DUT's interface and produces transactions that are sent to the scoreboard. The scoreboard analyzes any discrepancies between the transactions produced by the monitor and the expected transactions and reports them.

A UVM testbench environment with a driver, a sequencer, and a virtual interface must be created next. The virtual interface is used by the driver to communicate the transactions produced by the test sequence generator with the DUT. The transaction flow is managed by the sequencer, who also makes sure that the transactions are sent in the right order. In addition to isolating the testbench from the DUT's implementation specifics, the virtual interface offers a transaction-level interface between the two. The UVM testbench must be run as the last step using a simulation program like QuestaSim or VCS. Here we have used an open-source simulation platform edaplayground to run our simulations and waveforms. The test sequence generator generates a stimulus that is sent to the DUT through the driver and the virtual interface during the simulation. The monitor gathers data from the signals being sent and received at the DUT's interface and produces transactions that are sent to the scoreboard. The scoreboard analyzes any discrepancies between the transactions produced by the monitor and the expected transactions and reports them. In the event that there are any discrepancies, the testbench can be

troubleshot to determine the root of the issue. UVM can be used to automate, standardize, and streamline the verification of UART, producing designs with higher quality and faster time to market.

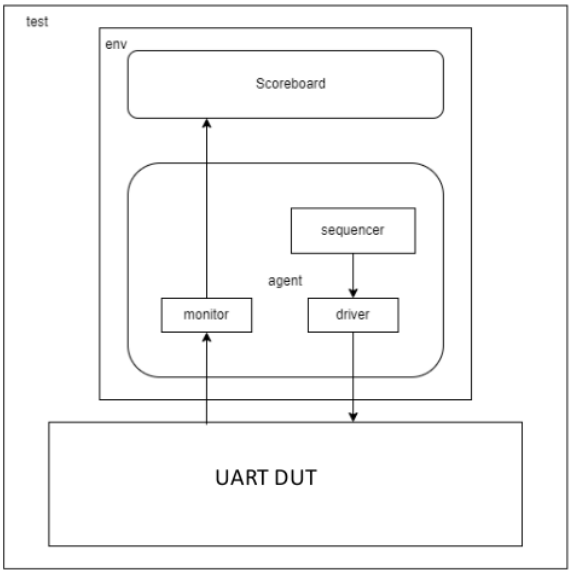

Figure 11: UVM components

As depicted in the figure 11, The sequence is the data being transmitted, while the Sequencer merely acts as a bridge between the Sequence and the Driver. The driver receives the randomized data from the sequencer. The driver sends this info to DUT following protocol. A monitor keeps track of the communication data and feeds it to the scoreboard. The output of the DUT is compared to the sequence on the scoreboard. The term coverage is used to gauge the verification's progress. Functional coverage tracks how the test plan is being executed. It compiles the simulation data to produce the progress report. It keeps track of the extent to which other features, such as important sets of values and boundary conditions, are covered. Knowing what set of values or characteristics has been tested for is a crucial component of testing. To improve the effectiveness of the design, the test cases are adjusted or new test cases are introduced using the progress report. The user defines the coverage points for the functions in a DUT. All of the functions in the test plan have been properly tested, according to 100% coverage. The test procedures should include the expected outcomes for each test case as well as step-by-step instructions for carrying out the test cases. The intended outcomes must be specified precisely, describing how the UART DUT is supposed to behave. Goals for coverage should be included in the test strategy as well. The functional requirements of the design should serve as the basis for defining the coverage targets, which should encompass both functional and code coverage. Statement, branch, and condition coverage are a few examples of the coverage metrics that should be included in the test plan.

5.2UVM Environment Creation

To drive the UART DUT with the test stimulus and record the response, the UVM environment is essential to the verification process. The UVM environment is built using the Universal Verification Methodology (UVM) library, which is a standardized methodology for developing reusable verification environments. Several UVM elements, such as the agent, monitor, scoreboard, and driver, make up the UVM environment. The agent is in charge of applying the test stimulus to the UART DUT and recording the response. It includes a sequence generator that creates test sequences using the test plan created earlier. The agent also includes a driver, which transforms the test sequences into signals sent to the UART DUT, and a monitor, which records the UART DUT's response. Verifying the accuracy of the UART DUT's response is the responsibility of the scoreboard. When there is a discrepancy between the response from the monitor and the anticipated outcomes specified in the test plan, an error is generated. The UVM environment created by the verification has been shown in figure 12.

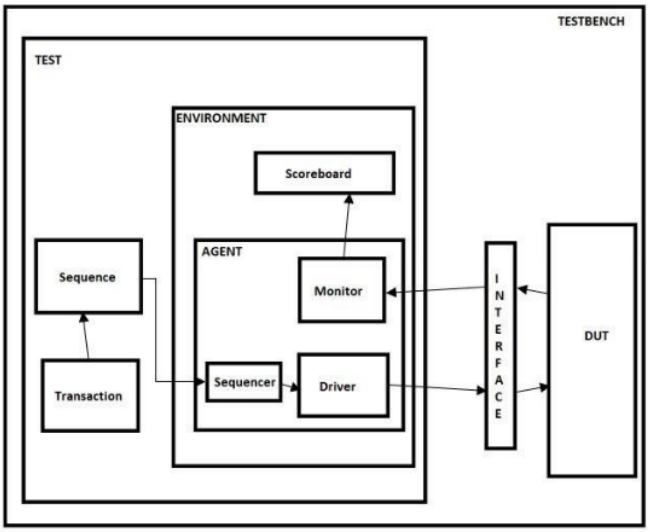

Figure 12: UVM Environment

Figure 12 explains the UVM environment. A configuration object, which is used to pass parameters to the various UVM components, as well as a test bench, which organizes the execution of the test sequences, are also included in the UVM environment in addition to the UVM components. A UVM testbench environment with a driver, a sequencer, and a virtual interface must be created next. The virtual interface is used by the driver to communicate the transactions produced by the test sequence generator with the DUT.

5.3 UVM Test Case Development

Defining the test scenarios and making the appropriate test sequences that will apply the required stimulus to the UART DUT are required steps in developing UVM test cases. The purpose of these test cases should be to confirm that the UART DUT complies with the specifications listed in the test plan. To test the DUT's fault-handling and error-recovery mechanisms, the UVM test cases should also include a variety of error scenarios. The test cases might, for instance, involve scenarios in which the UART DUT receives erroneous data, encounters buffer overflow or underflow, or encounters communication errors as a result of noise or interference. The sequences, drivers, and monitors from the UVM test bench should be used to create the UVM test cases. These elements are in charge of creating the necessary stimulus and recording the responses from the DUT. Following the development, the test cases are added to the UVM test bench and executed using a UVM test runner. The UVM environment records the results as the test cases are carried out, creates a test coverage report, and calculates the proportion of the design that has been tested using this information. It may be necessary to add more test cases to increase the test coverage if any design elements are not completely covered.

Testcases checked:

Table 1: Test Case Coverage

	Test Name	**Test case description**	**Status: Pass/Fail/Unchecked**
1	transmitter check	Correct data to Tx check	Pass
2	transmitter address allocation check	Whether the Tx data is allocated to the right address	Pass
3	receiver check	Correct data to Rx check	Pass
4	receiver address allocation check	Whether the Rx data is allocated to the right address	Pass
5	Are the transmitter and receiver both working parallelly or not	Full duplex check	Pass
6	baud rate check	Whether the system operates at the right baud rate	Pass
7	baud rate change check	Toggling baud rates	Unchecked
8	transmission with correct baud rate check	Whether the tx baud rate is accurately generated as mentioned in the spec	Pass
9	receiving with correct baud rate check	Whether the rx baud rate is accurately captured as mentioned in the spec	Pass
10	variable baud rate Tx and rx check	Operating tx and rx at different speeds	Fail
11	parity check	Checking the parity of the signals	Pass
12	data conversion with correct parity check	Checking the tx and rx as a whole after adding parity bits	Pass

5.4 Functional Coverage of the System

To make sure that the test cases have covered every aspect of the UART DUT's functionality, functional coverage is a crucial metric. It assists in determining how thoroughly the verification process has been carried out. Functional coverage points must be established to accomplish this; these points represent particular functional features or features that must be combined and put to use by the test cases. The UVM environment's coverage collector component can be used to gather the coverage results after these coverage points have been defined using SystemVerilog cover groups or cover points. Once functional gaps in the verification have been found, the coverage data can be examined to determine which areas need further testing. To make sure that all facets of the verification process have been covered, it is crucial to monitor additional coverage metrics in addition to functional coverage, including code coverage, assertion coverage, and protocol coverage. Doing so makes it easier to spot any potential problems that might have gone unnoticed during the verification process and guarantees that the UART DUT satisfies the necessary functional and performance requirements.

5.5 Protocol Checking

Protocol checking involves confirming that the DUT complies with the UART protocol specifications in the context of UART verification using UVM. To accomplish this, the output of the DUT is typically compared to the expected behavior specified in the UART protocol specification.

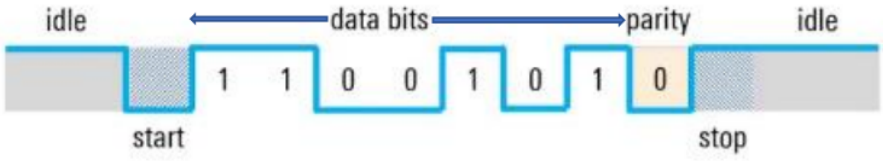

Figure 13: Protocol Reference

Figure 13 explains the protocol reference. In serial communication between tw devices, the UART (Universal Asynchronous Receiver/Transmitter) protocol is frequently used. A start bit, data bits, optional parity bits, and stop bits are all included in the format of the data that is transmitted. The following describes the UART's data format:

1. Start Bit: When a UART transmits a bit, the start bit is always the very first one. It serves as a notification to the receiver that a new byte of data has begun. In most cases, the start bit is one bit-time long and always low ((0)).

2. Data Bits: Following the start bit, the data bits themselves are transmitted. Depending on the application, there may be 5, 6, 7, or 8 bits of data transmitted for each byte. The most typical data format consists of eight data bits, one stop bit, and no parity.
3. Parity Bit: To check for errors during data transmission, the parity bit is a supplementary bit that is optionally included. The total number of bits transmitted (including the parity bit) is made even or odd by setting the parity bit to either 0 (even parity) or 1 (odd parity). Once the receiver has verified that the total number of bits transmitted is accurate, it checks the parity bit.
4. The stop bit, which the UART always transmits as the final bit, is known as the stop bit. It is employed to inform the receiver that the last data byte has been received. In accordance with the application, the stop bit is always a high (1) bit and is typically one or two bit-times long.
5. UART makes sure that the transmitted data is received precisely and without errors by adhering to this data format.

Verifying compliance with protocols is the responsibility of the UVM environment's protocol checker component. It receives the output from the monitor component, which records the interaction between the DUT and the testbench, and evaluates it in comparison to the predicted behavior. Errors are reported for any differences between the actual behavior and the expected behavior. To make sure that the DUT complies with the necessary communication standards and that it can communicate with other devices that use the same protocol, it is crucial to perform the protocol-checking step. Additionally, early detection of any potential design flaws or problems during the verification process is helpful.

5.6 Performance Analysis for the DUT

A UART DUT's performance analysis is an essential part of the verification process. To confirm that the UART DUT satisfies the necessary performance requirements, the performance of the device is being examined. Throughput, latency, and power consumption are among the performance metrics that are frequently verified. The amount of data that can be transmitted in a given amount of time is referred to as throughput. Different test cases are created to transmit various amounts of data at various speeds to verify throughput. To confirm that the data transfer rate satisfies the necessary throughput requirements, a measurement is then taken. The time interval between data transmission and data reception by the receiving device is referred to as latency. Test cases are created to transmit data at various rates, and the time between transmission and reception is measured to confirm that it complies with the necessary latency requirements. Another crucial performance metric that is confirmed during performance analysis is power consumption. The UART DUT is tested under various operating conditions and its power consumption is measured using power analysis tools to confirm power consumption. To confirm that the power consumption complies with the necessary power specifications, it is then compared to those specifications. Overall, performance analysis makes sure the UART DUT complies with the necessary performance requirements and is prepared for integration into the finished product.

5.7 Corner Cases Verification

Corner case testing is a crucial step in determining the UART DUT's robustness. It entails testing conditions that are on the periphery of the typical operating range, including minimum and maximum values, overflow and underflow scenarios, error scenarios, and other uncommon cases. With the aid of these tests, it is possible to confirm that the UART DUT can successfully handle a variety of challenging circumstances. Corner cases include, for instance, testing the UART DUT at either the maximum or minimum baud rate, with both the maximum and minimum number of data bits, and with both the maximum and minimum values of stop bits and parity bits. To make sure that the UART DUT handles errors gracefully, it is crucial to test error scenarios like invalid input values or unexpected communication interruptions. You can make sure the UART DUT is durable and dependable under a wide range of operating conditions by testing these edge cases.

5.8 Coverage Report Generation

To make sure that the verification team has adhered to the objectives and specifications of the test plan, a coverage report is an essential tool in UVM verification. The functional and protocol coverage that the test cases were able to achieve is summarized in the coverage report. The verification team can use the coverage report to pinpoint areas where the test cases might not have satisfied the desired coverage objectives. After identifying these areas, the verification team can then develop new test cases to fill in the gaps. The percentage of code that has been used by the test cases is known as functional coverage. To measure the functional coverage of the UART DUT's required testing, the verification team can define functional coverage points. The UART DUT is guaranteed to comply with the required communication standards by protocol coverage. Measured by protocol coverage points are particular facets of the UART protocol that the verification team must check. The functional and protocol coverage goals and the degree to which the test cases have achieved them should be summarized in the coverage report. Any components of the UART DUT or UART protocol that were not sufficiently covered by the test cases should be noted in the coverage report. The verification team can subsequently develop new test cases to fill in these gaps and increase the overall test coverage.

5.9 Results:

The next step is to review the test results and troubleshoot any issues or errors that were discovered after creating the coverage report. Waveform viewers, log files, and debuggers are just a few examples of the tools and methods that can be used for the analysis. Finding the problems' underlying causes and fixing them are both parts of the debugging process. It might be necessary to change the DUT code or test cases. These tests are the overall accumulation of all the tests performed as mentioned above on the transmitter and the receiver side of the design. These tests are combined in the waveform to provide an imperative idea of the operations of the protocol. In individual tests conducted or the waveforms would not be feasible under general supervision.

For the implementation of this project, we have used Quartus Prime 18.1 for the design and to get the FSM for both the transmitter and the receiver and to check the validity of the code. The design is verified on edaplayground and we used synopsys vcs compiler to compile and used UVM 1.2. For the physical

design part, we have used openLane which is on top of the Windows subsystem for Linux (WSL 2). We have also used docker version 2.36, git version 2.36, python 3. WSL has ubuntu 2.

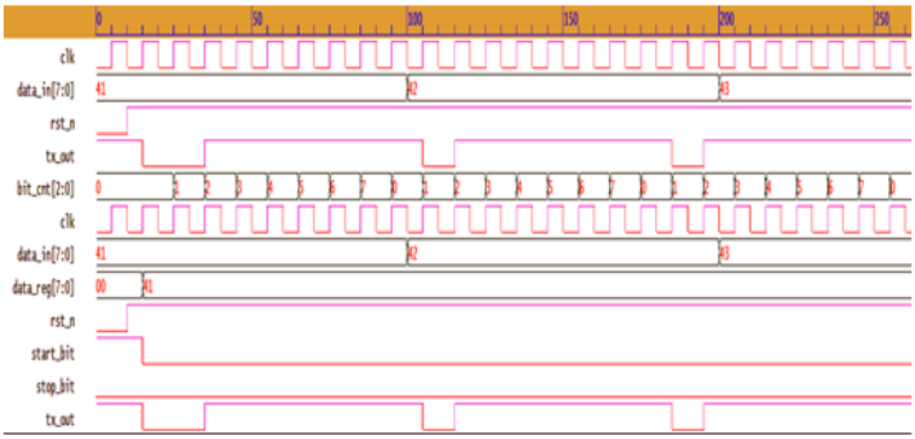

Figure 14: Transmitter Simulation

Figure 14 depicts the way uart design reacts to the stimulus provided by the testbench. The UART TX (transmitter) begins transmitting data when data are written to it. A start bit, which is always sent by the transmitter at a low logic level to indicate the beginning of a new data frame, is sent first. Following that, it sends the data bits one at a time, beginning with the least significant bit (LSB) and concluding with the most significant bit (MSB). In most cases, depending on the configuration, there are 7 or 8 data bits. The transmitter optionally sends a parity bit for error detection after the data bits to help with transmission. It is based on the data bits to calculate the parity bit, which can be even, odd, or none. As a final signal that the data frame has come to an end, the transmitter sends a stop bit, which is always sent at a high logic level. The UART TX responds to a number of events that occur during transmission, including receiving new data bytes, completing data frames, and spotting errors. Interrupt creation, error reporting, and state modifications are just a few of the possible responses. As an illustration, when the transmitter receives a fresh data byte, it buffers it and waits for the subsequent transmission cycle. The transmitter signals the end of transmission and awaits the arrival of the following data byte after completing a data frame. The transmitter reports any errors it finds and tries the transmission again when it finds one, such as a parity or framing error. The clock signal is generated inside the testbench for reference using the baud rate calculations provided in the report where data_in is an 8-bit data coming into the transmitter end of the design. The device defined here is an active low reset design so as soon as the rst_n signal is de-asserted, the design accepts the incoming data and processes it further. Then the data is sampled and shifted and takes 8 clock cycles depicted by a variable bit_cnt in the waveform to keep track of whether the data coming in is in multiples of 8 or not which is the required data frame in the design and then stores it in the data_reg in the design. Whenever there is a out on the transmitter end, the tx_out signal is asserted and de-asserted respectively to match the transmission and processing of the design. The start and stop bits show the starting and stop of the transmission in the design. This snippet is a partial waveform and has a stop_bit signal asserted at the end of the simulation.

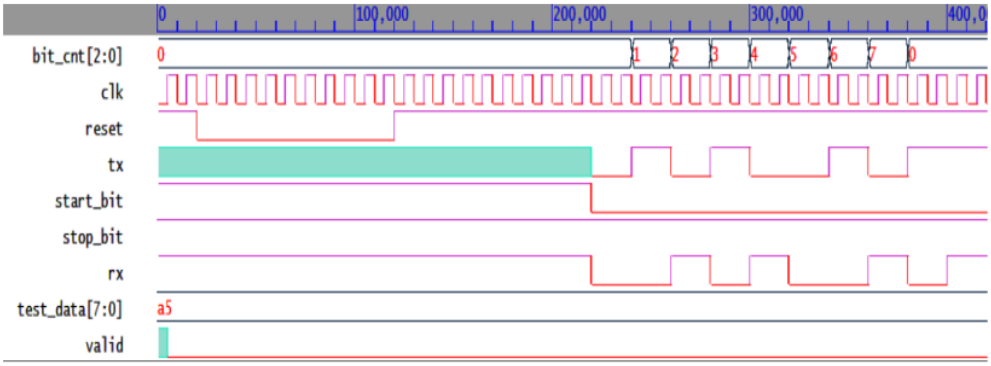

Figure 15: Receiver Simulation

Figure 15 explains the receiver waveform. The clock and reset signals, as well as the signals for sending and receiving data, are defined at the beginning of the code. Additionally, it specifies the validity and signals for the data received. The correct signal connections are then made when the DUT is first created. There are several parameters that are defined, including the simulation time, baud rate, number of data bits, and stop bits. The testbench variables and signals are initialized, and the test data to be transmitted is specified as a vector with the logic value [7:0]. The testbench variables and signals are initialized after the testbench clock is created using an "always" block by the code. Prior to sending the test data, the DUT is reset and given time to stabilize. By using the start bit, data bits, and stop bits signals to create the defined baud rate, data bits, and stop bits signals, the transmission of the data is made possible. After comparing the transmitted and received data, the transmitted data is declared valid. If the transmitted and received data are identical and the data is marked as valid, the test is successful. Upon reaching a certain number of clock cycles, the simulation is over.It's critical to have a thorough understanding of the design and testbench before you can debug something effectively. This entails being aware of the timing and order of signals, the expected DUT behavior, and the operation of the UVM components. To work effectively with other team members who may be in charge of various facets of the design or testbench, it is also critical to have strong communication and collaboration skills. The tests should be rerun to ensure that no new problems have been introduced by the changes after the problems have been located and fixed. To reflect any additional coverage that the modified tests may have produced, the coverage report should be updated. Until the desired coverage and functionality objectives are met and the UART DUT is prepared for integration into the larger system, this iterative process is continued.

6 UART Physical Design

The Physical design,i.e. The backend is also implemented for the UART design using OpenLane. Physical design simply means generating the GDS format of the layout from the RTL code which then can be used by the foundry, for example, TSMC or IFS for chip fabrication. Generating GDS has multiple intermediary steps. It starts with synthesis, then partitioning, floorplanning, placement, clock-tree synthesis, routing, timing closure, and then sign-off. All the mentioned steps have their inputs and outputs and their intermediary steps as well. Upon sign-off, the GDS of the design is generated. Proper implementation of the physical design of a particular design yields better performance, area, power, and ultimately cost which directly impacts the sale of the chip for that company. A product or system's physical design makes sure it performs well and accomplishes its intended function. Performance, reliability, and efficiency are directly impacted by factors including component placement, interconnections, and layout optimization. A well-thought-out physical design facilitates the smooth fusion of many components, maximizing functionality and improving the overall user experience. It is fundamental to producing structures and goods that are usable, aesthetically pleasing, safe, efficient, and sustainable. It combines technological know-how with aesthetics, usability, and manufacturing concerns to produce successful results that satisfy customer wants and market expectations.

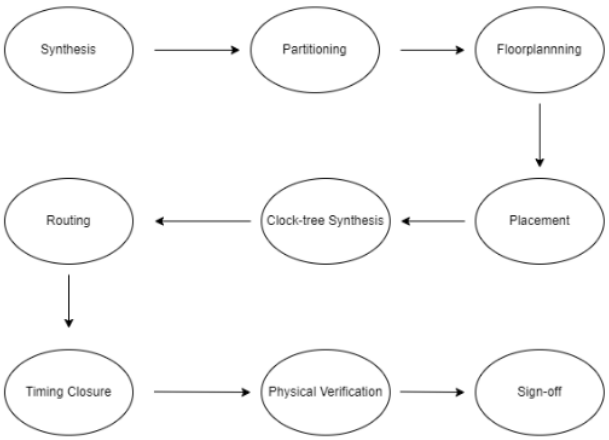

Figure 16: Physical Design Flow

Figure 16 explains the backend flow. In this project, the backend implementation is done using OpenLane, which is an open-source tool for RTL to GDS flow. The flow starts with the synthesis. It takes inputs such as HDL files (.v, .sv), Libraries, and constraints (.sdc) and generates a netlist as well as for, area, and timing reports. Here, we have used Skywater 130 nm library. Then, the partitioning takes the outputs which synthesis generates along with physical libraries (.lef file) to get the portion of the design. The floorplan takes the inputs and generates a .def file. Then the placement stage places the cells in their unique position as per the floorplan. The clock tree is then generated using RC delay models. Then the timing tool performs a timing analysis of the design and generates a timing report. After that, the DRC and LVS are performed to see the violations of the design and then finally, GDS is

generated. OpenLane has a separate tool for every stage. Synthesis uses Yosys and ABC, floorplan uses Placer and PDN, and placement uses RePLace, Resizer, and OpenDP. Clock tree synthesis uses TritionCTS, Routing uses FastRoute, and tools like Magic, and Klayout are used for checks and GDSII generation. The flow also performs DRC, LVS, and Antenna checks and the circuit validity checker. These checks are related to the foundry rules, for example, the minimum spacing between 2 nets, the thickness of the metal layer, etc. The flow also generates logs and reports which are used to analyze the performance of the design, For example, the timing reports tell the slack value. The placement report tells the utilization of macros and standard cells. The routing stage gives the congestion report which tells the amount of congestion between 2 metal layers. These metrics are used to analyze the design performance. The OpenLane architecture is shown in the figure below.

Synthesis:

In the VLSI design process, synthesis occurs between the RTL Design & Verification and Physical Design phases. The conversion of one notion level into another is the definition of synthesis. Front-end engineers write the code for RTL Design in a variety of languages, including Verilog, system Verilog, VHDL, etc., and then validate the codes (verification stage). Once the RTL code has been validated, we move on to the Synthesis step of VLSI Design. We now translate logic codes into the circuit. Numerous goals are served via synthesis. One of them is getting a Gate-level Netlist. Clock gate insertion, DFT logic insertion, and logic optimization are additional crucial goals.

Table 2 QoR report metrics of synthesis stage.

Metric	Number
Number of wires	315
Number of wire bits	404
Number of public wires	18
Number of public wire bits	68
Number of memories	0
Number of memory bits	0
Number of processes	0
Number of cells	371
Chip area	2772.659200

Table 2 lists various metrics which are observed in various QoR reports of the synthesis stage. These numbers are useful when an improvement happens and the designer then compares a better and optimized netlist to the previous. Sometimes, the front-end team makes an addition to the Verilog code, so the new netlist can be compared to the previous one to see the new cells added by the tool because of the changes in the Verilog code. The synthesis stage also performs a rough timing analysis for the designer to visualize the timing. This operation is not a very good estimation but gives the designer an idea of the improvement the front-end team can make to achieve a good slack. It has the following information:

Table 3 Power report of the synthesis stage.

Type of group	Power (microwatts)	Percentage
Sequential group	202	68.1 %
Combinational group	94.6	31.9 %
Macro	0	0%
Pad	0	0 %
Total	297	100 %

Table 3 lists various power related metrics of the synthesis stage. The sequential group (flips flops and registers) contributed 68.1 % of the whole power which is 202 microwatts. This is because the clock switching generally takes more consumption that the logical or combinational circuits which contribute 32 % ot 94.6 microwatts of power. The total power consumption is roughly 297 microwatts of the entire design.

Floorplan

The key to a successful physical design lies in the floor plan, which determines the layout of an ASIC design. By carefully considering the arrangement of I/O pads, macros, power, and ground structures, an optimized ASIC design with improved performance can be achieved. It is crucial to ensure that the data used for the physical design is properly prepared before proceeding with the actual floorplanning process. Throughout the entire physical design workflow, iterative modifications of the representation are made, with each step involving the construction and evaluation of a new ASIC representation. These iterative refinements are necessary to meet the requirements of the system. Physical designers often encounter issues related to breaches of physical design rules during ASIC design verification. If such breaches are identified, the physical design processes must be redone to gradually enhance the routing procedures and meet the design timing requirements. Effective floor planning significantly impacts an IC's performance. By strategically placing components and functional blocks, designers can reduce signal delays, minimize distances between critical routes, and improve overall timing characteristics. Proper floor planning allows for optimization of routing, signal integrity, and power distribution, resulting in increased speed, reduced power consumption, and improved overall performance. Moreover, it enhances overall production efficiency, shortens time to market, and facilitates testing and debugging during manufacturing and testing stages. This, in turn, ensures the long-term reliability and functionality of the IC by enabling efficient power delivery, minimizing power usage, and preventing overheating.

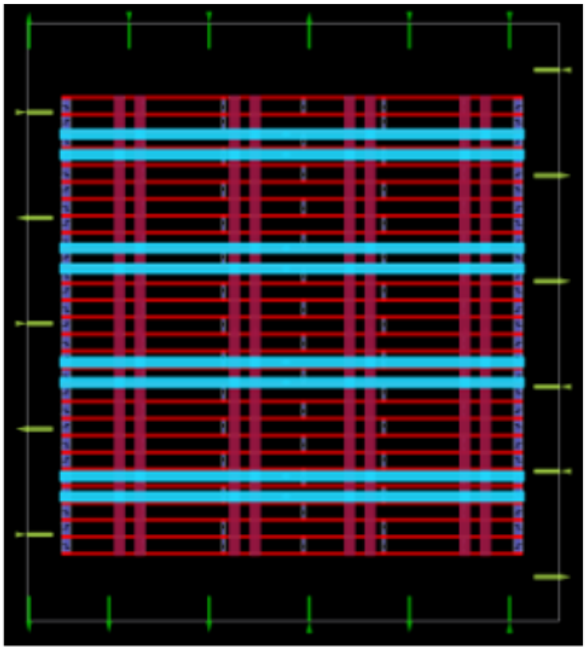

Figure 17: Floorplan Layout

Figure 17 shows the output layout of the floorplan stage. The yellow lines on the boundary are the input pins of the chip. The green pins are the output pins of the chip. The red and the blue lines are Vdd and the ground layers. They provide specific voltage levels to the cells. The purple line between 2 metal layers is the tap cells which are used to block the placement of particular cells.

Placement:

Placement is a crucial step in the physical design process, involving the strategic positioning of modules within the designated area to optimize performance and minimize wire length. The primary objective is to achieve timing closure, ensuring that the circuit meets its timing constraints. By placing critical lines and related components close together, placement reduces clock domain skew and minimizes signal delays. This leads to improved timing constraints and simplifies the overall clock distribution network, resulting in enhanced performance and the attainment of desired frequency targets. Placement is an iterative process integrated with other physical design stages such as floor planning, routing, and optimization. To strike a balance between various design constraints and achieve the best results, multiple iterations of placement are often necessary. Designers can explore different placement options and make adjustments based on time, power, and area requirements until an optimal solution is found. The placement directly impacts the utilization of chip area, and efficient placement aims to satisfy all design constraints while minimizing the overall chip area required. By optimizing the arrangement of components and macros, placement maximizes the utilization of available space, leading to a more

compact and cost-effective design. Effective area utilization enables higher chip density, reducing manufacturing costs and allowing for the integration of additional functionality within a given chip size. Furthermore, placement plays a role in reducing electromagnetic interference (EMI), crosstalk, and signal distortion by minimizing the length and impedance of interconnects between components. This contributes to accurate data transmission and reception while reducing the likelihood of errors or signal integrity issues, thereby ensuring reliable signal quality.

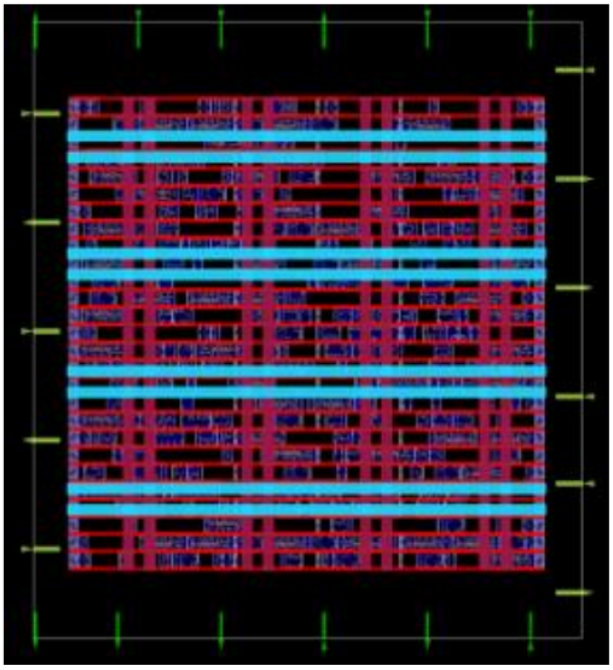

Figure 18: Placement Layout

Figure 18 shows the output layout of the placement stage. The yellow lines on the boundary are the input pins of the chip. The green pins are the output pins of the chip. The red and the blue lines are Vdd and the ground layers. They provide specific voltage levels to the cells. The purple line between 2 metal layers is the tap cells which are used to block the placement of particular cells. The little purple blocks in between the lines are the standard cells that are placed by the tool. Some tools also place filler cells which are used in case there is a modification of any addition to the RTL code.

Routing:

Routing is a crucial step in chip design where the previously placed components are connected using wires to establish physical pathways for data flow. It directly affects the timing characteristics of the design. By eliminating signal delays and optimizing route paths, routing contributes to achieving timing closure. Techniques like clock tree routing and skew management are employed to ensure that critical paths meet timing specifications and enable the desired operating frequency and performance. Efficient

routing facilitates the best possible utilization of the IC's area. It accomplishes this by strategically planning and arranging interconnects, which reduces overall wire length and the chip area required. This compact design allows for more cost-effective chip fabrication and higher integration densities. Proper routing is essential for maintaining signal integrity within the IC. It minimizes signal distortions, delays, and electromagnetic interference (EMI) by maximizing routing paths and considering factors such as cable length, impedance management, and noise coupling. Routing significantly influences the distribution of power and ground signals across the IC. By effectively routing power and ground nets, it reduces power supply noise, voltage dips, and coupling effects. This ensures a steady and reliable power distribution to all components, preventing power-related issues and maintaining proper operation. Employing appropriate routing strategies addresses manufacturability concerns such as wire density, lithography limitations, and viable via locations. Routing ensures that the layout adheres to design guidelines and restrictions, making it manufacturable and allowing for proper and reliable manufacturing processes

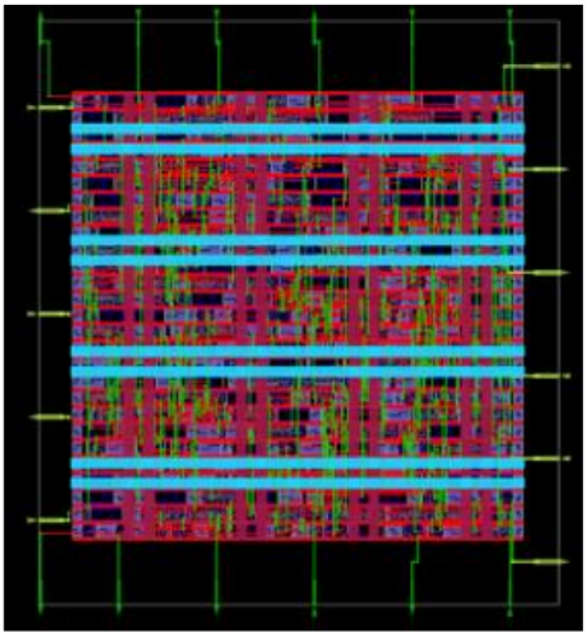

Figure 19: Routing Layout

Figure 19 shows the output of the routing stage. The lines connecting 2 standard cells are the metal layers. There are 5 metal layers and each layer is connected to a nearby layer with vias. Just like 2 floors of a building are connected via pillars, similarly, all the layers are connected through vias.

Clock-Tree Synthesis:

In the process of designing and optimizing clock distribution networks for integrated circuits (ICs), Clock Tree Synthesis (CTS) plays a vital role. Its primary objective is to ensure effective and reliable delivery of clock signals to all sequential elements, such as flip-flops, within the IC layout. Clock signals are necessary to synchronize the operation of consecutive elements in an IC. CTS is crucial for achieving timing closure as it enhances the efficiency of the clock distribution network. By minimizing clock skew, which refers to variations in the timing of clock signals as they reach different locations, CTS ensures that the clock reaches the flip-flops within their designated setup and hold time windows. CTS enables the fulfillment of timing constraints, allowing the IC to achieve the desired operating frequency and performance by ensuring balanced clock routes and minimizing clock skews. In addition to timing considerations, CTS also contributes to power efficiency in IC designs. By optimizing the routing paths of clock signals and considering factors such as wire length, capacitance, and power supply noise, CTS reduces power consumption. This is achieved by minimizing unnecessary switching activity and power dissipation in the clock network. The result is improved power efficiency and overall reduced power consumption.

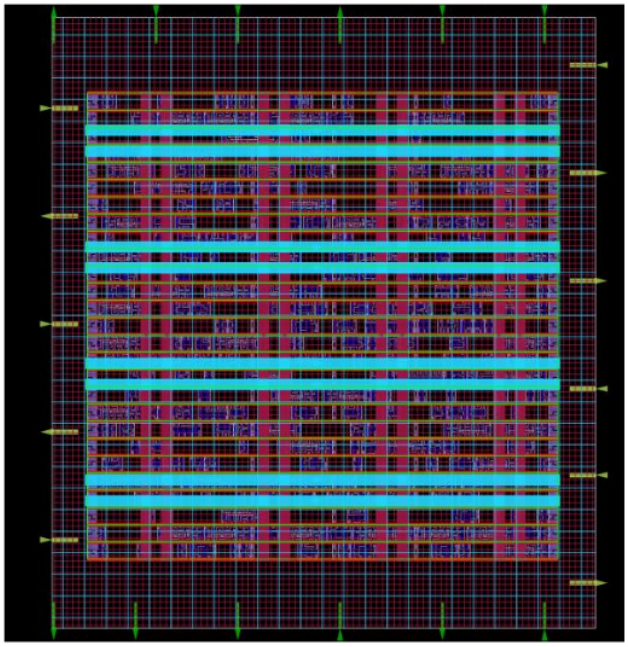

Figure 20: CTS Layout

Figure 20 shows the layout of the cts stage. In the CTS stage, the clock network is generated throughout the design. As the UART is asynchronous, the baud rate is something that keeps the transmitter and receiver synchronized. Here, the baud rate is generated from the clock using a counter. The figure below shows the zoomed picture of the CTS output which is the metal tracks along the routed paths.

Physical Verification:

For the physical verification, the flow generates a manufacturability report that has DRC and LVS information. There are no LVS violations, so the design is LVS clean, but there are 27256 DRC violations that can be cleaned and brought further down. Typically, the DRC violations are in the 10,000s but can be lowered when timing improvement is done. Also, the design is antenna clean, which means there are no antenna violations. Some of the other violations are mentioned in the table.

Table 4 DRC violations

Type	**Number**
Triton route violation	0
Short violations	0
Metal spacing violations	0
Off-grid violations	0
Minimum hold violation	0
Magic violations	0
Pin violations	0
Net violations	0
Lvs total violations	0
Cvc total violations	1
Klayout violations	4549
Total	**4550**

Table 4 explains the various design check and rules that are part of the physical verification stage. As per the table, the design does not violate the rules apart from the CVC and the klayout violations. The minimum width and spacing between various layout elements, such as metal lines, contacts, or vias, are checked by CVC. It marks them as violations if these dimensions are less than the set design guidelines. KLayout checks to see if the minimum distance between layout elements like wires, vias, or pads is being met. KLayout checks to see if the area complies with the requirements by looking at the density of the features there. If the density is either too high or too low, it can signal infractions. To make sure layout features, like metal lines or transistors, adhere to the established standards, CVC evaluates their density. If there are too many or too few features in a specific region, density violations may happen.

Signoff:

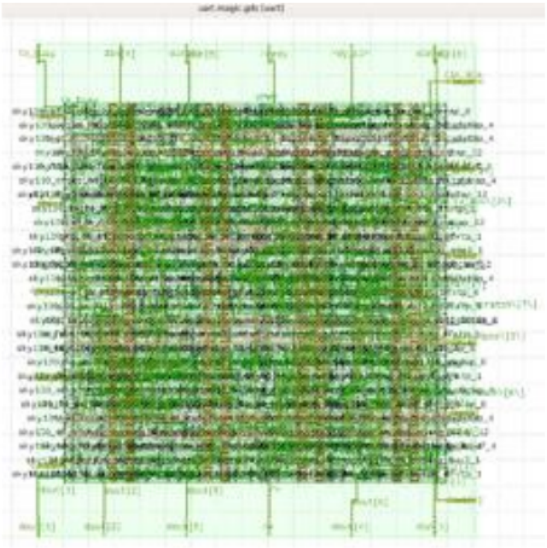

Figure 21: GDS Signoff

Figure 21 shows the final GDS picture of the design. It contains the shape, size, and location of every cell along with the routing. The green lines are the metal layers and the square shapes are the interconnections between them. The figure below shows the zoomed picture of the GDS with the metal interconnections between two layers. This file is used by the verification tools and the foundries example Intel Foundry Services or TSMC for chip fabrication.

7.Conclusion:

In this paper, a thorough and methodical methodology was used across all stages of the design process, , RTL design, verification, synthesis, and physical implementation, to guarantee a solid and dependable design. The RTL design stage had the translation of the specifications into Verilog code. The verification stage was important to validate the functionality of the design. The physical design stage was important to visualize and look for the metrics to optimize the current design. In conclusion an ASIC design cycle is implemented in this project to demonstrate the importance of each and every step of the cycle right from RTL to GDS generation. The future work includes optimizing the design at RTL level, maybe including pipelining. This could make the design work at greater frequencies. Also, adding more test cases. Finally, optimizing the design at the backend level to reach a more optimal value of performance, power and area.

8.References:

[1] Xi Jianbo, Xia Jijin, Luo Chuanhui, Deng Qingyong, and Zhu Peng, “Verification method of FPGA universal configurable UART protocol based on UVM”, Date published: 06/29/2016.

[2] Bidisha Kashyap and V Ravi, “Universal Verification Methodology Based Verification of UART Protocol”, Date published: 05/12/2020.

[3] M Srinath and Sujatha Hiremath, “Verification of Universal Asynchronous Receiver and Transmitter (UART) using System Verilog”, Date published: 07/07/2022.

[4] Vivekananda T and Mahesh Kumar N, “Design and Verification of the UART and SPI protocol using UVM”, Date published: 09/2022.

[5] B. Priyanka, M. Gokul, A. Nigitha and J.T Poomica, “Design of UART Using Verilog And Verifying Using UVM”, Date published: 03/19/2021.

[6] Yamini R and Ramya M V, “Design and Verification of UART using System Verilog”, Date published: 06/05/2020.

[7] Kumari Amrita and Avantika Kumari, “DESIGN AND VERIFICATION OF UART USING VERILOG HDL”, Date published: 01/2018.

[8] W.Elmenrwicb, M.Delvai, Time Triggered Communication with UARTs. In Proceedins of the 4’h IEEE International Workshop Communication Systems, Aug.2002.

[9] Nithin Patel, Naresh Patel, . VHDL Implementation of UART with BIST Capability, Fourth International Conference on computing, Communications, and Networking Technologies, pp 1-5, July, 2013.

[10] Kashyap B, Ravi V. Universal Verification Methodology Based Verification of UART Protocol. In Journal of Physics: Conference series 2020. Dec 1 (Vol.1716, No. 1, p. 012040).

Final_paper.pdf

ORIGINALITY REPORT

5%	3%	2%	1%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	technodocbox.com Internet Source	1%
2	chipedge.com Internet Source	1%
3	Kalaiyarasi M, Kaushik S, Saravanan R. "Infant Health Monitoring and Security System using IoT", 2021 Smart Technologies, Communication and Robotics (STCR), 2021 Publication	<1%
4	www.ijeat.org Internet Source	<1%
5	www.mail-archive.com Internet Source	<1%
6	K. Jayalakshmi. "Real-time simulation of electrical machines on FPGA platform", 2006 India International Conference on Power Electronics, 12/2006 Publication	<1%
7	Pong P. Chu. "FPGA Prototyping by VHDL Examples", Wiley, 2008	<1%

Publication

8	Submitted to CSU, San Jose State University Student Paper	<1 %
9	Submitted to Universiti Sains Malaysia Student Paper	<1 %
10	Submitted to CSU Northridge Student Paper	<1 %
11	kb.osu.edu Internet Source	<1 %
12	Cheng-Wen Wu. "Hierarchical testing using the IEEE Std 1149.5 module test and maintenance slave interface module", Proceedings of the Fifth Asian Test Symposium (ATS 96) ATS-96, 1996 Publication	<1 %
13	T Jain, P Bansod, C B S Kushwah, M Mewara. "Reconfigurable Hardware for Median Filtering for Image Processing Applications", 2010 3rd International Conference on Emerging Trends in Engineering and Technology, 2010 Publication	<1 %
14	s3.cern.ch Internet Source	<1 %
15	Homayoun Shahri. "Blurring Lines Between Hardware and Software", Queue, 2003 Publication	<1 %

16	dc.library.okstate.edu Internet Source	<1%
17	designengineering.jobs Internet Source	<1%
18	www.researchgate.net Internet Source	<1%
19	A. Steininger. "Revision and verification of an enhanced UART", IEEE International Workshop on Factory Communication Systems 2004 Proceedings WFCS-04, 2004 Publication	<1%
20	Jing Xu, Tao Lu, Lingling Gao. "Design and Application of In-Vehicle Terminal for Car Network System Based on ARM9", 2008 International Workshop on Education Technology and Training & 2008 International Workshop on Geoscience and Remote Sensing, 2008 Publication	<1%
21	Lahtinen, V.. "Interconnection scheme for continuous-media systems-on-a-chip", Microprocessors and Microsystems, 20020406 Publication	<1%
22	Syed Fasiuddin. "GDRIU Digital Section Implementation Using FPGA", 2017 International Conference on Recent Advances	<1%

	in Electronics and Communication Technology (ICRAECT), 2017 Publication	
23	www.springerprofessional.de Internet Source	<1 %
24	Pong P. Chu. "Embedded SoPC Design with Nios II Processor and Verilog Examples", Wiley, 2012 Publication	<1 %

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		

Electrical Engineering Department

RTL to GDS implementation and verification of UART using UVM and OpenLane

Bhatt, Purav (MS Electrical Engineering)
Joshi, Dharmik Vijay (MS Electrical Engineering)

Project Advisor: Shrikant Jadhav

Introduction

- The universal asynchronous receiver/transmitter (UART) receives the data as bytes and sends the bits sequentially.
- The received bits are put back together into whole bytes at the destination side using a second UART. A shift register is present in each UART and is essentially used to translate serial data into parallel data.

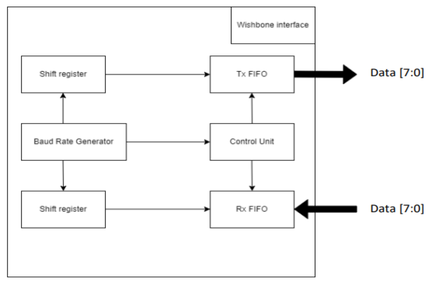

Figure 1: Proposed UART Design

- To avoid race circumstances, UVM offers systematic, orderly control over simulation behavior. This control will be carried out in sequence.

Methodology

Designing of UART:

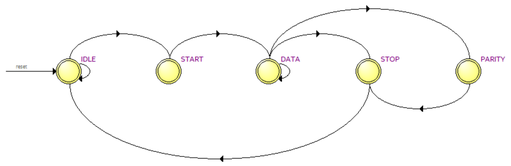

Figure 2: Transmitter FSM

- By default, it starts with the IDLE state before the Tx_enable signal is switched ON.
- Then it goes to the START state. Here, the transmitter adds the start bit to the shift register and shifts the data left.
- Then it goes into the DATA state.
- Finally, the FSM enters the last stage which is the STOP bit.
- If the parity_enable signal is turned ON, the transmitter adds a parity bit before entering the STOP bit.

Methodology

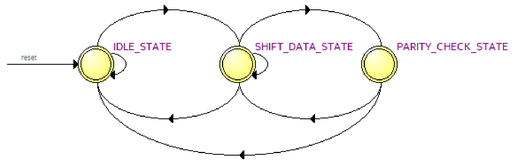

Figure 3: Receiver FSM

- The state machine starts by waiting for the start bit to be received in the IDLE_STATE.
- The state machine switches to the SHIFT_DATA_STATE after detecting the start bit.
- The state machine switches to PARITY_CHECK_STATE after receiving all the data bits and the parity bit.

Verification of UART:

1. Test Bench: It replicates the environment. The test bench verifies whether the RTL fits the design specs..
2. Test Cases: Test cases are generated for several scenarios that encompass the functionality and corner cases.

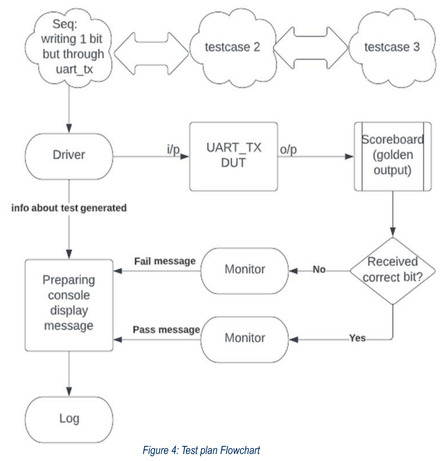

Figure 4: Test plan Flowchart

3. DUT: The hierarchal layering of the verification facilitates the maintenance and reuse of the DUT.

4. Virtual Interface: It offers a technique for separating abstract models and test programs from the actual signals that comprise the design.

5. Scoreboard: The data and control information at the output of the DUT are collected and compared to those at the output on the scoreboard.

Analysis and Results

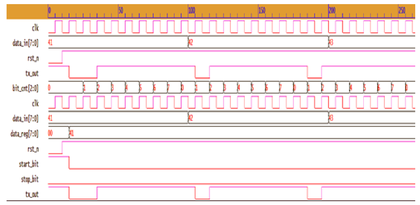

Figure 5: Simulation Results of Tx

- After running the tests and simulations on the DUT on both the transmitter and receiver side, the results have been satisfactory for ideal conditions.
- In some corner cases where the design is not expected to return any value, the DUT produces garbage values. Producing garbage values is expected as the spec that is defined by the team is pretty basic and just for educational purposes.
- In that case, the DUT and the verification environment work in perfect sync and show no anomalies during the runtime simulations.

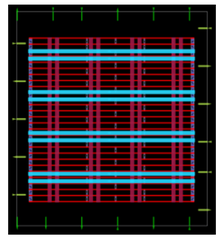

Figure 6:: Floor-planning

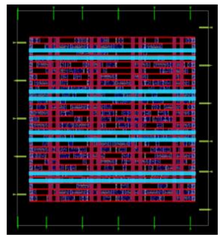

Figure 7: Placement

- Floor-planning is the process of allocating space for blocks/modules in a chip. Placement is the process of positioning the modules on the allocated space to optimize performance and reduce wirelength.

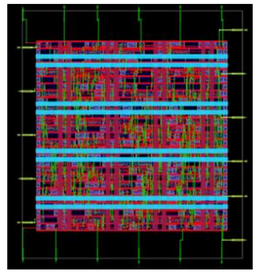

Figure 8: Routing

- Routing is the process of connecting the previously placed components in a chip design with wires to create a physical path for data flow.

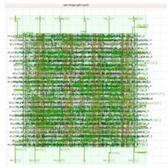

Figure 9: Sign-off

- Signoff refers to the final stage of a chip design where the design is checked against specifications and criteria to ensure functionality and manufacturability. GDSII is the file format used to transfer the finalized design data to a foundry for manufacturing.

Summary/Conclusions

- UART is a serial communication protocol that allows devices to communicate without the use of a clock signal. It sends parallel data that has been converted to serial format.
- It is a full duplex system that can transmit and receive data at the same time.
- This project proposes a methodology for the Design and Verification of a UART block along with physical implementation from the RTL to GDS flow using Open-Lane.

Key References

[1] Xi Jianbo, Xia Jijin, Luo Chuanhui, Deng Qingyong and Zhu Peng, “Verification method of FPGA universal configurable UART protocol based on UVM”, Date published: 06/29/2016. [Online]. Available: https://patents.google.com/patent/CN105718344A/en

[2] B. Priyanka, M. Gokul, A. Nigitha and J.T Poomica, “Design of UART Using Verilog And Verifying Using UVM”, Date published: 03/19/2021. [Online]. Available: https://www.semanticscholar.org/paper/Design-of-UART-Using-Verilog-And-Verifying-Using-Priyanka-Gokul/cc91fcab1360dc0b5b79974b6e1f99724ee52151

[3] Bidisha Kashyap and V Ravi, “Universal Verification Methodology Based Verification of UART Protocol”, Date published: 05/12/2020. [Online]. Available: https://iopscience.iop.org/article/10.1088/1742-6596/1716/1/012040

Acknowledgements

We would like to thank Prof. Shrikant Jadhav for his continuous encouragement and for believing in the team for going the extra mile and helping in the successful implementation of the project. We would also like the thank Prof. Morris Jones for his guidance related to the verification of the design in UVM.