DESIGNING OF UART PROTOCOL USING SYSTEM VERILOG AND VERIFICATION USING UVM

GUNDAPPAGARI VARSHA,PALA SAI KARTHIK REDDY

May 22, 2023

Contents

1 PROBLEM STATEMENT

Here goes the brief about the problem statement: The designing of The UART protocol in System verilog and the second part follows with the verification of the design module.

2 INTRODUCTION

The Universal Asynchronous Receiver/Transmitter (UART) is a widely used communication protocol in digital systems. It provides a simple and efficient way to transmit and receive data between two devices. In this presentation, we will discuss the design of UART in SystemVerilog and its verification using Universal Verification Methodology (UVM). We will cover the basic architecture of UART, its functionality, and the challenges we faced during its designing and verification.

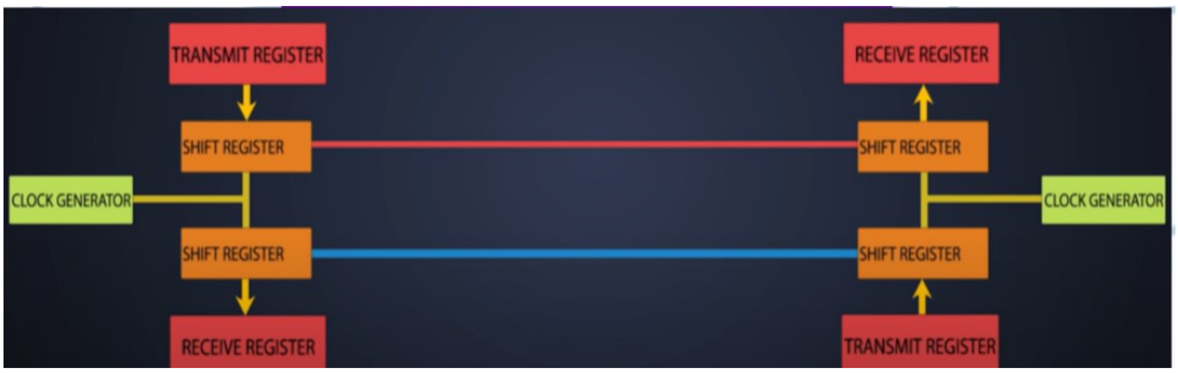
3 HOW TWO DEVICES COMMUNICATE WITH EACH OTHER?

For two devices to communicate with each other using UART protocol they should have three basic connections between them which are Tx,Rx and common terminal ground in order to make two devices work on the same level.

3.1 DATA TRANSMISSION BETWEEN TWO DEVICES

The data input is taken into the transmit register and then whole of data is loaded at once into the D-flip flops present in the shift register which send the data serially bit by bit synchronizing with the clock to the reciever device. Similarly the shift register on the reciever side recieves the data bit by bit and sends it paralelly to the recieve register.

3.2 UART PERIPHERALS



3.3 DATA FRAMING

A particular data frame conists of certain bits such as start bit which is in general high and then goes low when the communication starts and then we have the actual data which is in general of 8 or 9 bits followed by the parity bit to check whether there is an error in the data recieved and then at last there is a stop bit which indicates the end of the transmission and this bit goes low to high when it completes recieving the data.

3.4 BAUD RATE GENERATION

Why we have to use a seperate clock for Tx and Rx??

It is necessary to have separate clocks for Tx and Rx to enable the use of full-duplex communication, where both devices can transmit and receive data simultaneously. In this case, separate clocks are required to keep the two transmission and reception processes independent of each other.

4 DESIGNING UART IN SYSTEM VERILOG

4.1 DESIGN MODULE BLOCKS

The whole design module mainly consists of two blocks which are the transmitter and the reciever . then these have some input and output pins interfaced along .

4.2 UART TRANSMITTER

We have designed the module with variable clock frequency and variable baud rate hence specified them with "parameter". The input signals to the transmitter are as follows:

- **Clk and Reset**
- **send**:when the data is readily available to send it becomes high to activate the reciever.
- **dintx**: when the send signal becomes high then we send these 8 bits of data. **The output signals are:**
- **tx**: the signal is used to activate the reciever that there is data ready to be sent. **donetx**:signal which tells that the transmission is completed.

4.3 UART RECIEVER

The reciever sub-module also has some input and output signals interfaced with it.

Input signals are:

- **Clk and Reset**

- **Rx**: this signal becomes high when the reciever starts recieveing the stop bit which indicates end of transmission.

The output signals are:

- **donerx**:This signal is used to tell the user that the reception of all the 8 bits of data is completed.
- **doutrx**: this signal is used to send all the 8 bits recieved to store it in a register.

4.4 WHY SV OVER VERILOG?

- COMBINES VERILOG,VHDL AND C++,AND PROVIDES
- RESUABILITY OF THE CODE
- HIGH FUNCTIONAL COVERAGE
- THE DATA TYPES PROVISION IS MUCH HIGHER THAN IN VERILOG.

5 VERIFICATION IN SV

Verification in SystemVerilog is the process of confirming that a design or module functions correctly according to its specifications. It involves creating testbenches, which are modules or environments designed to exercise the design under test (DUT) and verify its behavior.

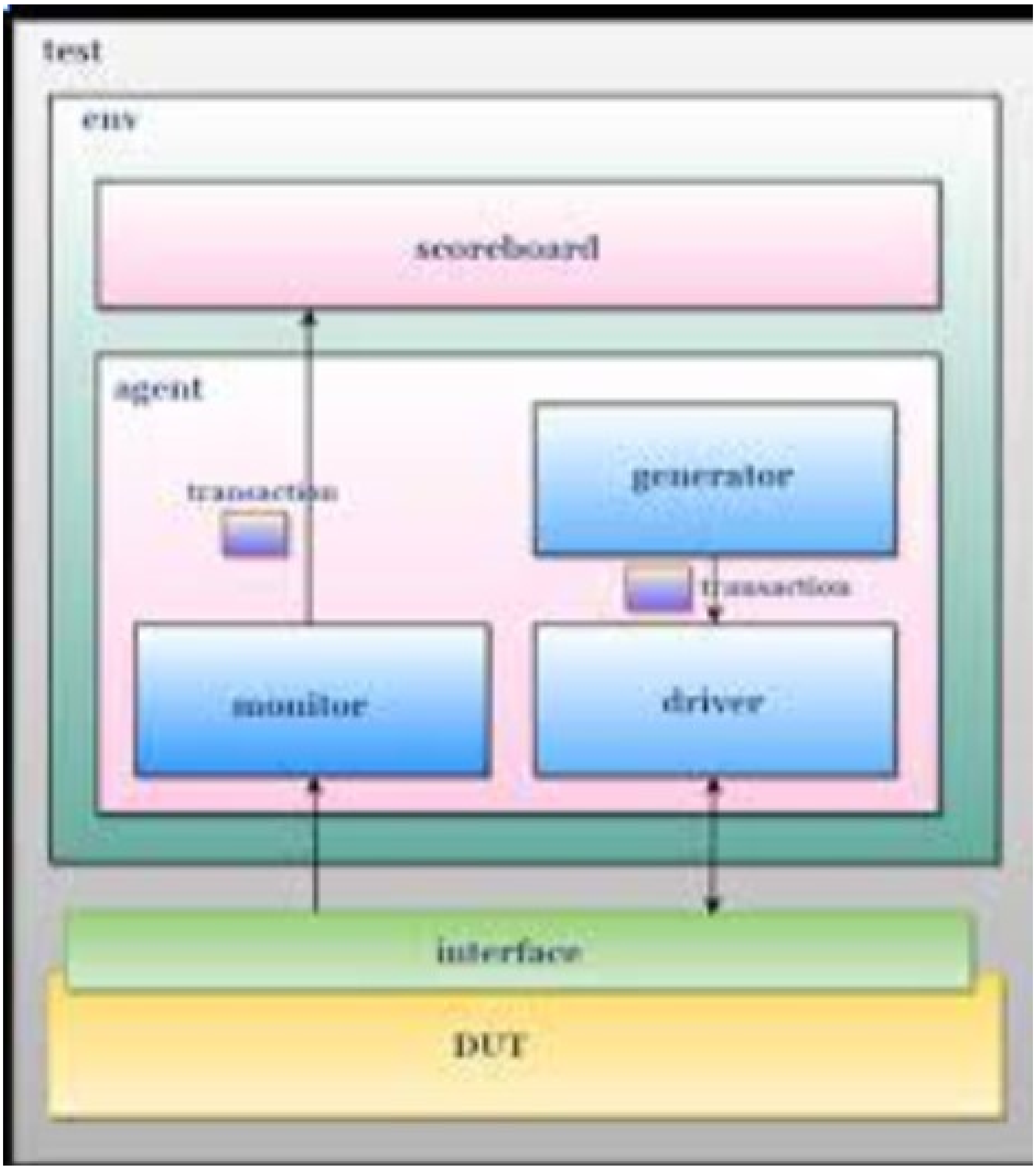
Here are the steps typically involved in the verification process using SystemVerilog:

- **Write the DUT**: Create the design module or modules that need to be verified. This could include RTL (Register Transfer Level) code or higher-level abstractions like behavioral or structural models.
- **Create a testbench**: A testbench is a module that instantiates the DUT and provides stimuli or inputs to it. It also captures the outputs of the DUT and checks them against expected values. Testbenches can be written in SystemVerilog as well.
- **Write test cases**: Define test cases that cover different aspects of the DUT's functionality. Test cases consist of input values and expected output values. They should be designed to thoroughly exercise the DUT and uncover any bugs or issues.
- **Instantiate the DUT in the testbench**: Instantiate the DUT module within the testbench and connect it to the appropriate signals and ports.

- **Apply stimuli**: Drive the test inputs to the DUT using procedural or concurrent statements in the testbench. These inputs can be generated randomly, based on specific patterns, or from a stimulus file.
- **Capture outputs**: Monitor the outputs of the DUT in the testbench and record their values for later comparison.
- **Perform assertions and checks**: Use assertions or checkers in the testbench to verify that the DUT's outputs match the expected values. Assertions are statements that assert certain properties or conditions to be true during simulation.
- **Simulate and debug**: Run the simulation using a SystemVerilog simulator, such as QuestaSim, VCS, or ModelSim. Monitor the simulation results, debug any issues, and investigate any failing assertions.
- **Coverage analysis**: Analyze the coverage metrics to ensure that the test cases are sufficiently covering different parts of the design. Coverage metrics measure the percentage of design code exercised by the testbench.
- **Functional and performance verification**: Once the functional correctness is confirmed, additional verification steps may be taken to verify the DUT's performance under various conditions, such as corner cases, stress testing, or real-world scenarios.
- **Debug and resolve issues**: If any issues or bugs are found during the verification process, debug them by analyzing waveforms, logs, and other debugging techniques. Make the necessary modifications to the design or testbench to address the issues.
- **Regression testing**: As the design or testbench is modified, run regression tests to ensure that the changes did not introduce any new issues or regressions.

5.1 WHY SV FOR VERIFICATION??

- **Integration of verification and design**: SystemVerilog allows you to seamlessly integrate your verification code with your design code. This integration enables easy access to design internals, making it simpler to drive stimuli and check the design's outputs.
- **Strong typing and compile-time checking**: SystemVerilog provides strong typing, allowing you to define and enforce data types for variables, ports, and signals. This helps catch errors at compile-time, reducing the likelihood of bugs and improving the overall reliability of the verification process.

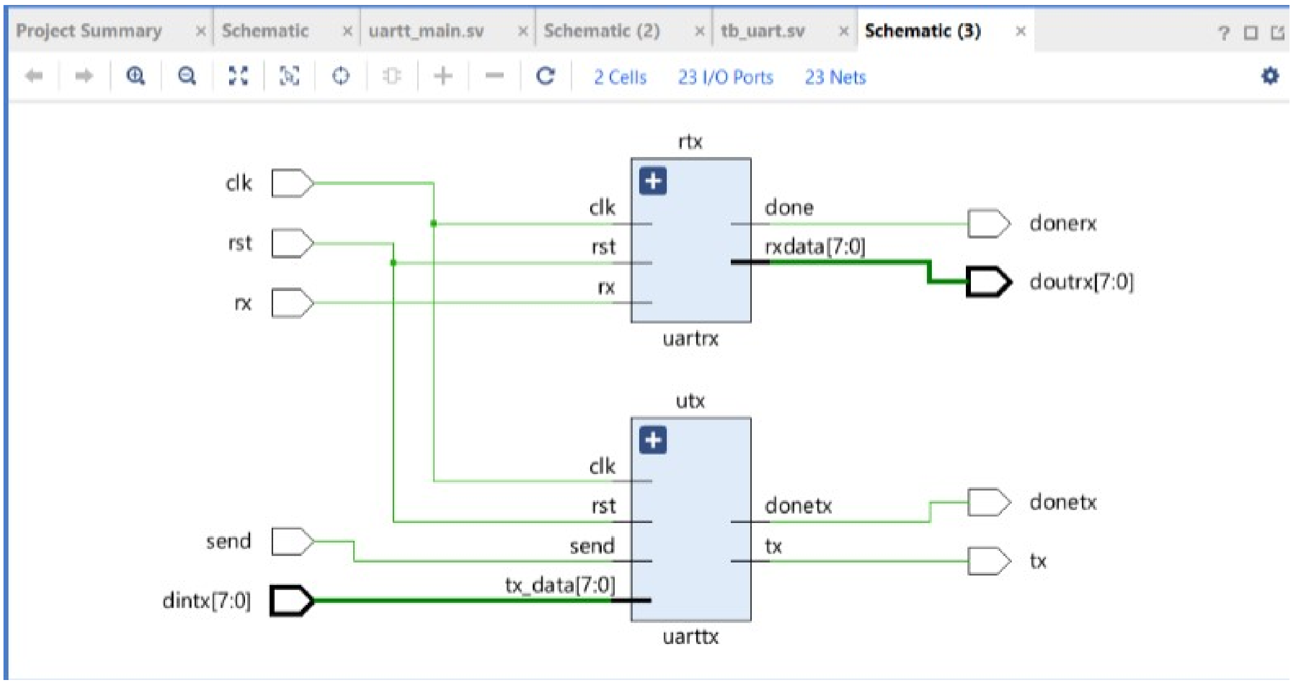


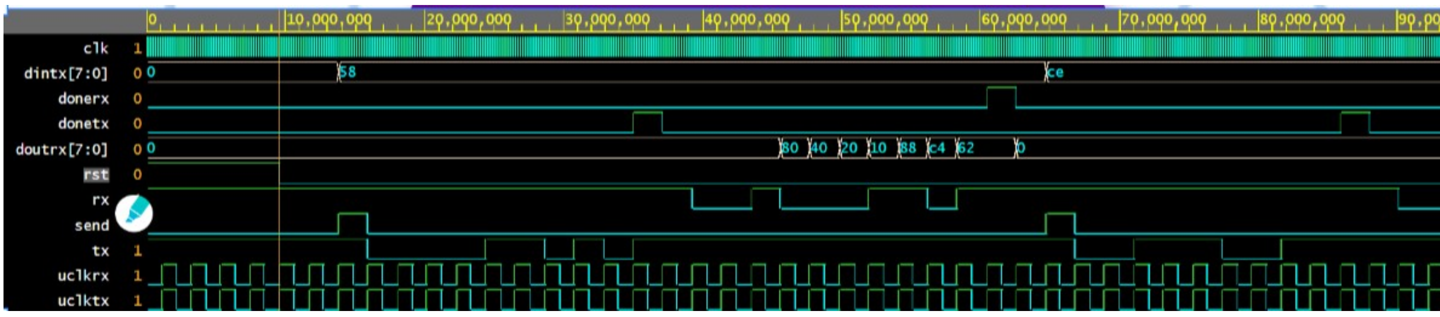
- **Constrained random stimulus generation**: SystemVerilog offers built-in constructs for random value generation. This feature is particularly useful for generating realistic and complex test scenarios, as you can define constraints on input values and let the simulator automatically generate random stimuli that adhere to those constraints. This approach enhances test coverage and reduces the effort required to write exhaustive test cases manually.
- **Coverage-driven verification**: SystemVerilog supports coverage constructs that allow you to measure the extent to which your testbench exercises the design. Coverage metrics provide insight into the completeness of your verification and help identify areas of the design that require additional testing. This enables you to achieve a higher level of confidence in

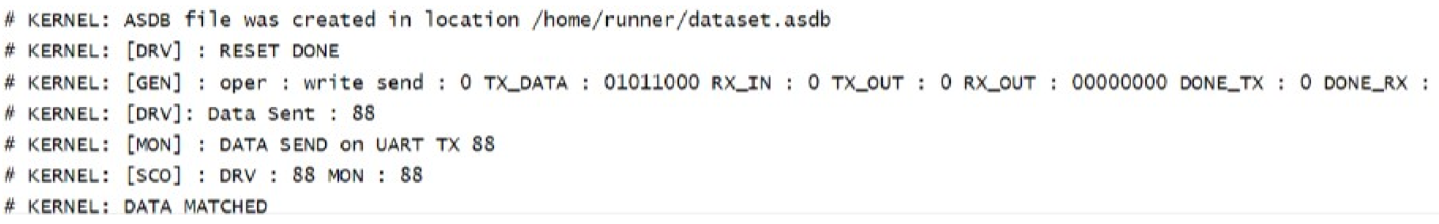
the verification results.

- **Reusability and scalability**: SystemVerilog supports modular and hierarchical design, which facilitates reusability and scalability of verification environments. You can build reusable verification components, such as testbenches, checkers, and monitors, that can be easily instantiated and connected to different designs. This modularity allows for efficient development and maintenance of verification infrastructure, saving time and effort.

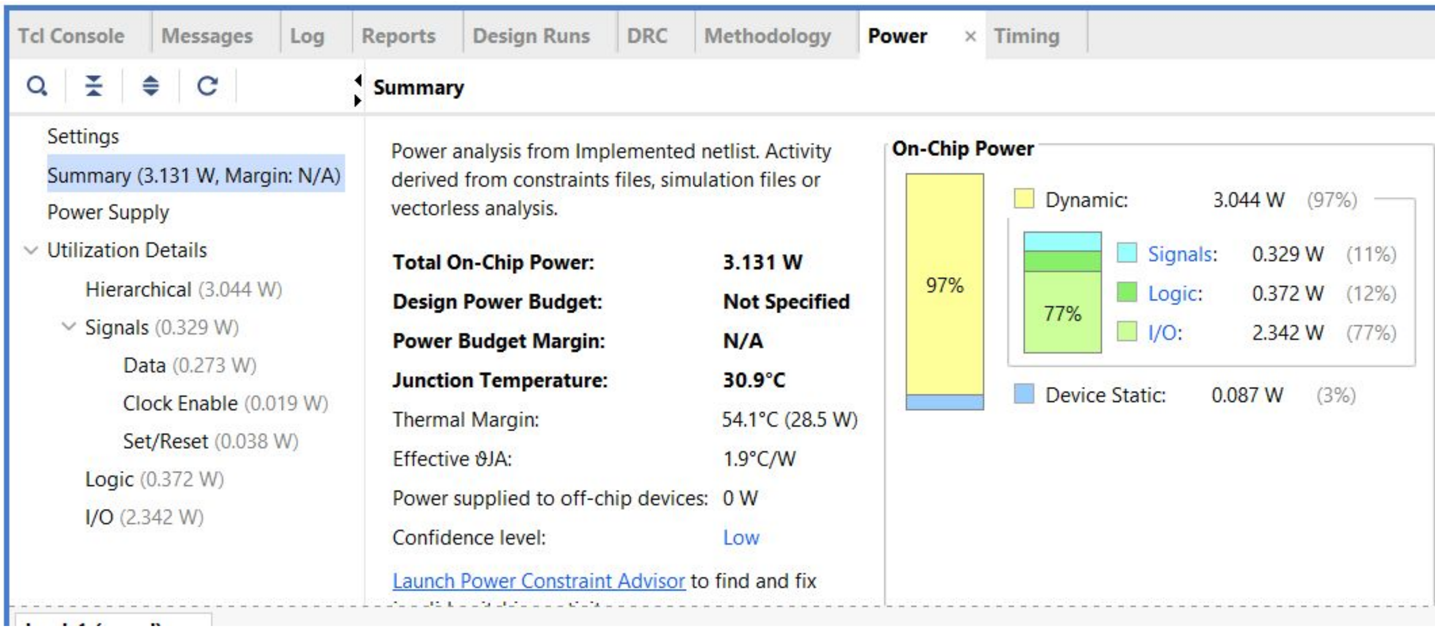
6 VERIFICATION,DESIGNING RESULTS







```
# KERNEL: ASDB file was created in location /home/runner/dataset.asdb
# KERNEL: [DRV] : RESET DONE
# KERNEL: [GEN] : oper : write send : 0 TX_DATA : 01011000 RX_IN : 0 TX_OUT : 0 RX_OUT : 00000000 DONE_TX : 0 DONE_RX :
# KERNEL: [DRV]: Data Sent : 88
# KERNEL: [MON] : DATA SEND on UART TX 88
# KERNEL: [SCO] : DRV : 88 MON : 88
# KERNEL: DATA MATCHED
```



Settings
Summary (3.131 W, Margin: N/A)
Power Supply
Utilization Details
Hierarchical (3.044 W)
Signals (0.329 W)
Data (0.273 W)
Clock Enable (0.019 W)
Set/Reset (0.038 W)
Logic (0.372 W)
I/O (2.342 W)

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**3.131 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**30.9°C**
Thermal Margin:	54.1°C (28.5 W)
Effective ϑJA:	1.9°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix

On-Chip Power

	Dynamic:	3.044 W	(97%)
	Signals:	0.329 W	(11%)
	Logic:	0.372 W	(12%)
	I/O:	2.342 W	(77%)
	Device Static:	0.087 W	(3%)

7 CHALLENGES AND FUTURE SCOPE

7.1 CHALLENGES FACED

The verification of UART poses several challenges, including the complexity of the design, the need for extensive stimulus generation, and the difficulty in debugging errors. The complexity of the design makes it challenging to ensure the correctness of all possible scenarios. The extensive stimulus generation requires significant effort to cover all possible test cases. Debugging errors can be time-consuming and require advanced skills in debugging tools and techniques.

7.2 FUTURE SCOPE

Improving the data transfer rate: The primary objective of the UART design is to transmit and receive data. By optimizing the hardware design and software algorithms, the data transfer rate can be increased. This can be achieved by implementing parallel data transmission or using faster clock rates.

Integration with other protocols like I2C and SPI protocols

The testing and verification with system verilog itself is faster but we can improve it much further by using some lagorithms like random testing , assertion based verification and formal verification.

7.3 CONCLUSION

The design of UART and its verification using system verilog provide an efficient and reliable way to transmit and receive data between two devices. While the verification process poses several challenges, the use of advanced tools and techniques can help overcome these challenges. In conclusion, the design and verification of UART play a critical role in the development of digital systems and are essential for ensuring the reliability and performance of these systems.