

**ECE6024 - VLSI Verification Methodologies**

**PROJECT REPORT**

**Winter Semester 2019-2020**

**Verification of Universal Asynchronous  
Receiver and Transmitter (UART)**

By

**Mahima Bhatnagar – 19MVD0039**

**Prabhu N – 19MVD0058**

**Mayur – 19MVD0041**



**VIT<sup>®</sup>**

---

**Vellore Institute of Technology**

(Deemed to be University under section 3 of UGC Act, 1956)

## **ABSTRACT**

By using a conventional directed test bench, verification of complex circuit design is difficult because creating all test cases manually is a tedious task and also not very efficient. The greater the complexity of the designs, the higher the probability of bugs appearing in the code. Increasing complexity of ICs has created a necessity for performing verification on designs with an advanced, automated verification environment.

In this project UVM (Universal Verification Methodology) is used and required output is observed in EDA playground. UVM is used to build modular reusable verification components and test benches. Our approach is to design UART is that we implement UART transmitter and receiver separately. In this baud rate used is 19200 bits per second for transmission of signal and system clock is divided according to that. Clock frequency used is 2.6 KHz In this Finite State Machine (FSM) approach is used to code both UART transmitter and receiver.

## **1. Introduction**

UART (Universal Asynchronous Receiver Transmitter). It is a integrated bus protocol which is used to transmit data serially. It also control serial device that are interfaced with computer. It consists of both receiver and transmitter. In UART, data flows from transmitter to receiver pin. The first device transmitter is connected to second device receiver. In this clock is not used for transmitting data because UART is asynchronous serial transmission. It transmits data in packets so it consists of start bit, stop bit, data bits.

In DUT transmitter and receiver is designed. The input of transmitter is clock, reset, start, 8-bit transmitter data input, output of transmitter, and done when data is transferred successfully it will become high. In case of receiver, the input will be clock, reset, receiver and receiver out. In top module of design input clock, reset, receiver, 8bit transmit data, start is used and output pin transceiver, 8 bit receiver data out, transmitter active and done signal is used. All these pins of top module are mapped to transmitter and receiver.

## **2. Limitations:**

Assertion and code coverage is not verified.

### 3. About DUT

A UART consists of a transmitter and receiver, both of which utilize a shift register that performs parallel to serial and serial to parallel conversion. The UART transmitter sets the channel interrupt status flag and the data transfer request status flag to indicate when data has been transferred from the transmit data register to the transmit shift register. The setting of these flags indicates the transmit data register is available to accept new data. When data is written to the transmit data register, the MSB of the transmit data register must be written as zero. This is a handshake signal to the UART function, indicating that new transmit data to be serially shifted out has been written. In UART data gets transferred serially in small package. The data flow like this:

**Start bit:** Start bit starts the communication by sending data in a packet. When data is not transferred at that time start bit is set at high and when data transfer begin start bit is been pulled down. The receiver will detect this change from high to low and begins reading actual data.

**Stop bit:** UART set the data line to high voltage to end the data transaction.

**Data Bit:** In these actual data is transferred from sender to receiver. In this project 8bit data is transferred. The LSB is the first bit data of data to be transmitted.



Fig1: Transmission of Data

In this paper, the UART is operated in a 2.6KHz .In UART ,the bit start is used to start the transmitter to send the data , when the start bit is one then the transmitter will send the data , after the 8 bit sent then the done bit will be one ,then the start bit will become zero , the receiver receives the data in real time ,after receiving 8 bit it will automatically stop receiving the data and make receiver bit 0 .The Data is transmitted and received in the form of Finite state machine.

## State Diagram:

### Transmitter:

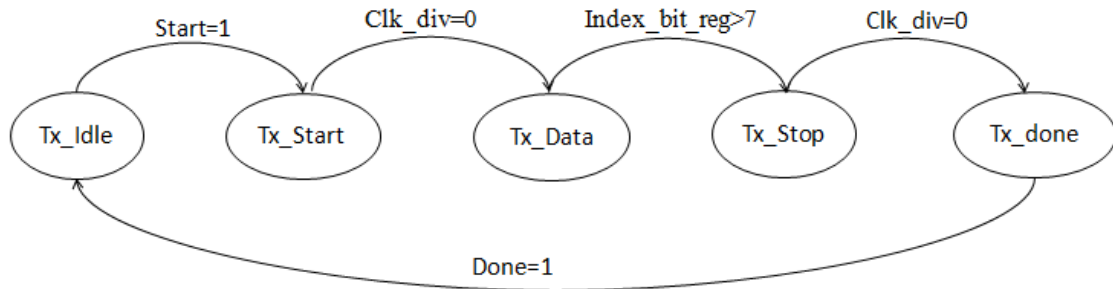


Fig2: Transmitter state representation

In the transmitter , the TX\_Idle bit will be activated only when the start is 1 , which is indicating that the transmitter is ready to send the data any , then it is moved to the Tx\_Start state , in this state when the frequency of the clock is reached then it is moved to the next state Tx\_Data . in Tx\_Data , all the 8 bit is transmitted one by one ,after transmitting all the data , it is moved to the next state Tx\_Stop .In Tx\_Stop state when the clock frequency of the UART is received then it is moved to the Idle state. The state diagram of the transmitter is shown in Fig 2.

### Receiver:

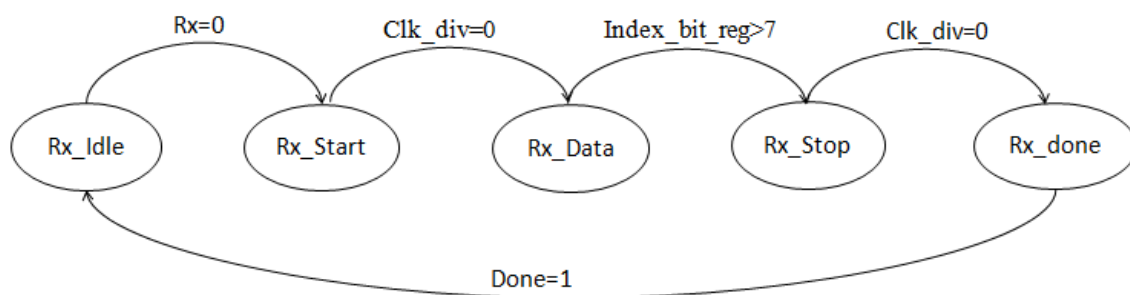


Fig3: Receiver state representation

In the receiver the data received using the FSM states. The Rx\_Start will be activated only when receiver is 0, which indicated that the receiver is not receiving any data. Then it is moved to next

state RX\_data. In Rx\_data all the 8 bit data is received one by one, after receiving all the 8 bit data then it is moved to next state Rx\_Stop. In this state, when the clock frequency of the UART is received, then it will be moved to the RX\_Idle state as shown in Fig3.

### 3.1 Port Definitions:

Port Name	Direction	Size
Clk	input	1
Reset	input	1
tx_data_in	input	8
Start	input	1
Tx	output	1
rx_data_out	output	8
tx_active	output	1
done_tx	output	1

Table1: Port Definitions

### 3.2 Block Diagram of DUT:

The UART main components are Transmitter and Receiver. Both transmitter and receiver have given Baud Rate. Baud rate decides at which rate data get transferred over serial channel. In the transmitter hold register contain the data byte to be transmitted and shift register shift data bit in a channel until the data is received. Control logic will tell when to read data or write data.

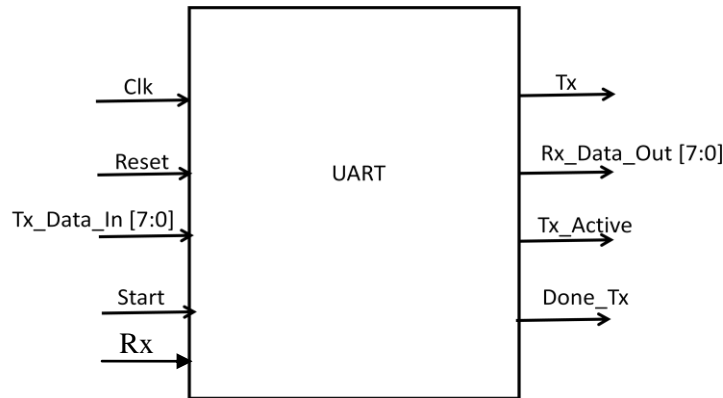


Fig 4: Block diagram of DUT

#### 4. About the Test bench:

**Agent:** The most important and basic element in UVM Architecture is the Universal Verification Component (UVC) or Agent. Because of Agent, the Test-bench of UVM is re-usable. Agent is an encapsulation of Driver, Monitor and sequencer. An UVM environment can consists of one or more agents. Agent can be configurable. There are two types of Agents-Active agent and Passive agent. If the agent is Active, then the agent will have all the driver, monitor and sequencer. But if the Agent is Passive, then the agent will have only Monitor.

**Driver:** Driver is an active entity that emulates the logic that drives the DUT. It fetches data repeatedly from the sequencer. Driver has to drive the DUT according to the protocol using the virtual interface. Driver drives the data to DUV using an interface.

**Monitor:** Monitor is a passive entity that can sample the DUT signals, but does not drives them. A monitor extracts signal information from the bus and translates the information into a transaction that can be made available to other components and to the test case writer.

**Sequencer:** The sequencer in UVM just acts as a gateway between sequence and driver. This is the only “non-virtual” class in our UVM test bench architecture. The sequencer is also parameterized by transaction class. The sequencer takes the randomized data from sequence and passes it to the driver to which it is connected, thus it connects several sequences to driver.

**Sequence:** The actual driven data is randomized in the sequence. From the sequence, this randomized data is given to driver via sequencer.

**Test:** Test is a place where we start the sequences on sequencer. In base test we will set all the parameters of configuration database class according to our requirements. We will also get the interface set .00000000in top and again set it to the interface handles in local configuration database classes. The base test also creates the environment. All these things happen in build phase of base test. The further child tests will be made from this base test class. In child test we just create the handle of virtual sequence and start it on virtual sequencer, before starting the sequence an objection is raised and this objection is dropped again after starting the sequence. If we don't raise the objections, the simulator will think that there is no run phase to execute, so the simulator can jump directly to extract phase. The total number of raised objections should be equal to the number of dropped objections.

**Environment:** The object of this class is created in test and is a most important component of UVM test bench. Environment creates agents, scoreboard, virtual sequencer etc. The environment makes connection between monitors and scoreboard. If it has virtual sequencer, here in environment we have to connect physical sequencers with the handles of physical sequencers in virtual sequencer.

## **4.1 Test plans:**

1. Checking whether, if reset is one all the outputs are set zero.
2. Checking whether the UART is working at 2.6 KHz frequency
3. If start =0, the data is not been transmitted.
4. If Tx\_Active, the data is not received by the receiver.
5. If done\_tx = 0, then checking whether the data is not received.
6. Checking whether the Tx and Rx are same.

## 4.2 Coverage

Coverage is a metric which gives us information that during simulation are all requirements and features in test plan are verified or not and also tells whether all lines of code are covered or not.

This information is carried by doing:

1. Code Coverage
2. Functional Coverage

**Code Coverage:** The simplest way to measure verification is code coverage. Here it checks the execution of number of lines called as Line Coverage, it checks each path from where the code is being executed called Path Coverage, it also measure all toggle condition called Toggle Coverage and it also measure all state and transitions of FSM condition called FSM Coverage.

**Functional Coverage:** Code coverage does not know what the design should do. If anything is missing in code then code coverage cannot find but functional coverage can tell if any functionality is missing. So, it measure how much design specification has been exercised in specification. Functional coverage is also called “Specification Coverage”.

In this project, functional coverage achieved is 100% as shown below.

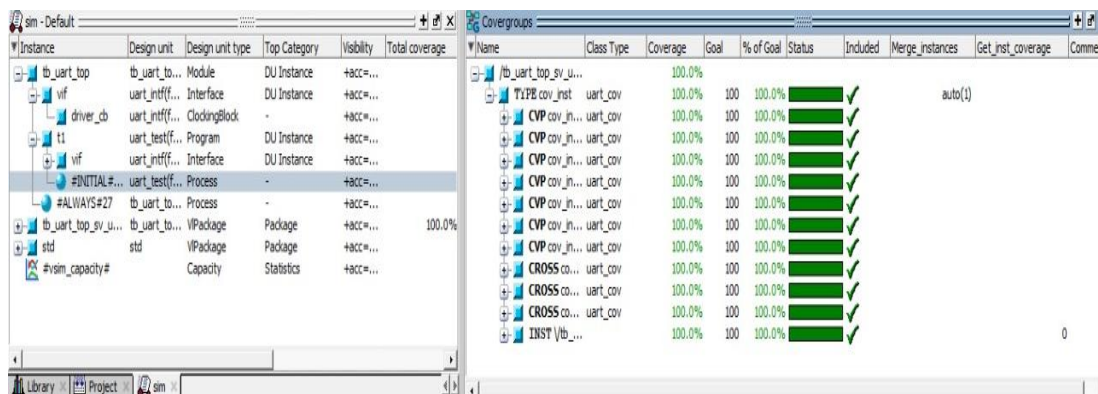


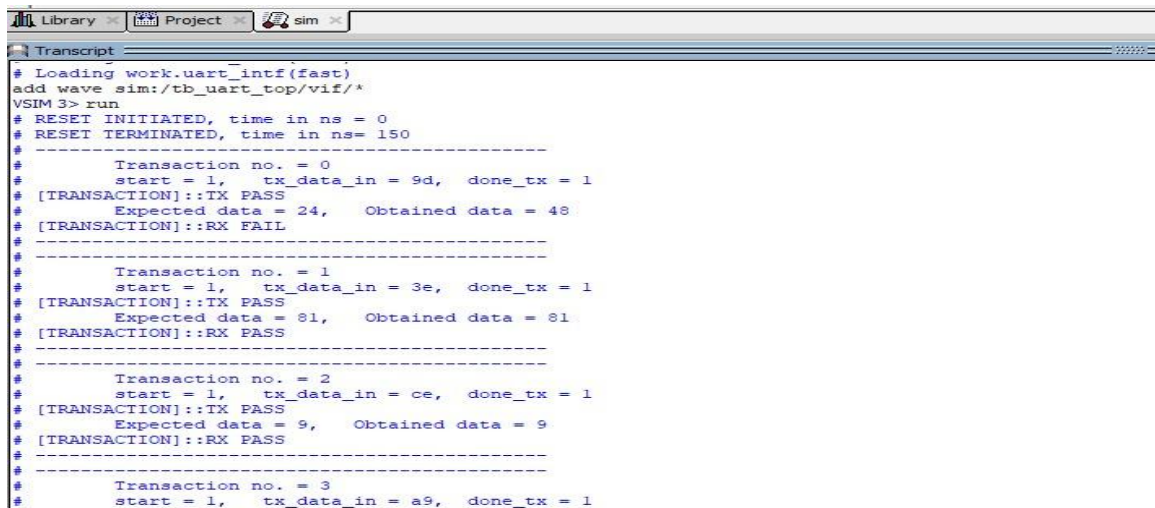
Fig 5: Functional Coverage



## 4.4.1 Simulation Results Using SV (System Verilog)

### i. Reset :

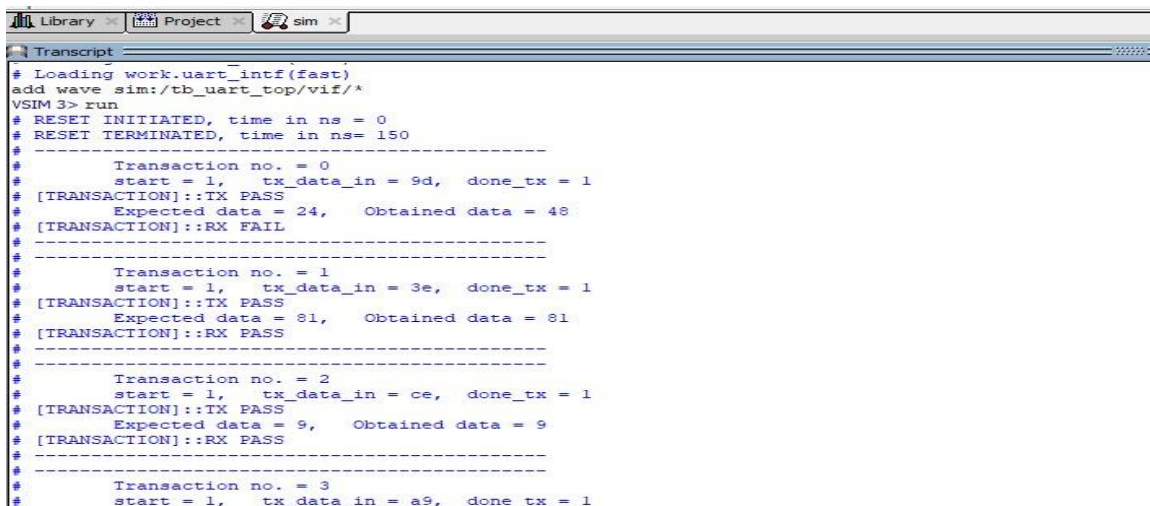
The Transcript is showing that when reset is high the data is not yet transmitted and received.



```
# Loading work. uart_intf(fast)
add wave sim:/tb_uart_top/vif/*
VSIM3> run
# RESET INITIATED, time in ns = 0
# RESET TERMINATED, time in ns= 150
# -----
# Transaction no. = 0
# start = 1, tx_data_in = 9d, done_tx = 1
# [TRANSACTION]::TX PASS
# Expected data = 24, Obtained data = 48
# [TRANSACTION]::RX FAIL
# -----
# Transaction no. = 1
# start = 1, tx_data_in = 3e, done_tx = 1
# [TRANSACTION]::TX PASS
# Expected data = 81, Obtained data = 81
# [TRANSACTION]::RX PASS
# -----
# Transaction no. = 2
# start = 1, tx_data_in = ce, done_tx = 1
# [TRANSACTION]::TX PASS
# Expected data = 9, Obtained data = 9
# [TRANSACTION]::RX PASS
# -----
# Transaction no. = 3
# start = 1, tx_data_in = a9, done_tx = 1
```

Fig 6: Reset condition using SV

### ii. Transcript Output for Test plan execution:



```
# Loading work. uart_intf(fast)
add wave sim:/tb_uart_top/vif/*
VSIM3> run
# RESET INITIATED, time in ns = 0
# RESET TERMINATED, time in ns= 150
# -----
# Transaction no. = 0
# start = 1, tx_data_in = 9d, done_tx = 1
# [TRANSACTION]::TX PASS
# Expected data = 24, Obtained data = 48
# [TRANSACTION]::RX FAIL
# -----
# Transaction no. = 1
# start = 1, tx_data_in = 3e, done_tx = 1
# [TRANSACTION]::TX PASS
# Expected data = 81, Obtained data = 81
# [TRANSACTION]::RX PASS
# -----
# Transaction no. = 2
# start = 1, tx_data_in = ce, done_tx = 1
# [TRANSACTION]::TX PASS
# Expected data = 9, Obtained data = 9
# [TRANSACTION]::RX PASS
# -----
# Transaction no. = 3
# start = 1, tx_data_in = a9, done_tx = 1
```

Fig 7: Transcript Output using SV

### iii. Output Waveform in System Verilog :

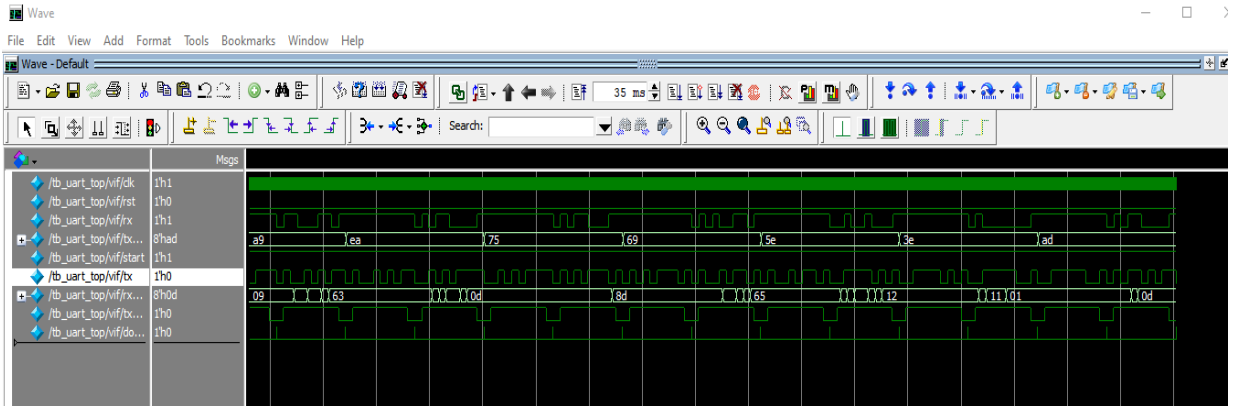


Fig 8: Output waveform using SV

## 4.4.2 Simulation Results Using UVM (Universal Verification Methodology):

### i. Reset :

The Transcript is showing that when reset is high the data is not yet transmitted and received

```
# RESET INITIATED, time in ns = 0
# RESET TERMINATED, time in ns= 150
# -----
# Transaction no. = 0
```

### ii. Transcript Output:

```
# -----
#
# UVM_INFO uart_driver.sv(32) @ 0: uvm_test_top.env.agent.driv [] -----
# UVM_INFO uart_driver.sv(33) @ 0: uvm_test_top.env.agent.driv [] Transaction No. = 0
# UVM_INFO uart_driver.sv(43) @ 261405: uvm_test_top.env.agent.driv [] start = 1, tx_data_in = 80, done_tx = 1
# UVM_INFO uart_driver.sv(44) @ 261405: uvm_test_top.env.agent.driv [] [TRANSACTION]::TX PASS
# UVM_INFO uart_driver.sv(63) @ 523815: uvm_test_top.env.agent.driv [] Expected data = 24, Obtained data = 24
# UVM_INFO uart_driver.sv(66) @ 523815: uvm_test_top.env.agent.driv [] [TRANSACTION]::RX PASS
# UVM_INFO uart_driver.sv(67) @ 523815: uvm_test_top.env.agent.driv [] -----
# UVM_INFO uart_driver.sv(32) @ 523815: uvm_test_top.env.agent.driv [] Transaction No. = 1
# UVM_INFO uart_driver.sv(33) @ 523815: uvm_test_top.env.agent.driv [] start = 1, tx_data_in = a5, done_tx = 1
# UVM_INFO uart_driver.sv(43) @ 784225: uvm_test_top.env.agent.driv [] [TRANSACTION]::TX PASS
# UVM_INFO uart_driver.sv(44) @ 784225: uvm_test_top.env.agent.driv [] Expected data = 81, Obtained data = 81
# UVM_INFO uart_driver.sv(63) @ 1046635: uvm_test_top.env.agent.driv [] [TRANSACTION]::RX PASS
# UVM_INFO uart_driver.sv(66) @ 1046635: uvm_test_top.env.agent.driv [] -----
# UVM_INFO uart_driver.sv(67) @ 1046635: uvm_test_top.env.agent.driv [] Transaction No. = 2
# UVM_INFO uart_driver.sv(32) @ 1046635: uvm_test_top.env.agent.driv [] start = 1, tx_data_in = 2d, done_tx = 1
# UVM_INFO uart_driver.sv(33) @ 1307045: uvm_test_top.env.agent.driv [] [TRANSACTION]::TX PASS
# UVM_INFO uart_driver.sv(43) @ 1307045: uvm_test_top.env.agent.driv [] Expected data = 9, Obtained data = 9
# UVM_INFO uart_driver.sv(44) @ 1307045: uvm_test_top.env.agent.driv [] [TRANSACTION]::RX PASS
# UVM_INFO uart_driver.sv(63) @ 1569455: uvm_test_top.env.agent.driv [] -----
# UVM_INFO uart_driver.sv(66) @ 1569455: uvm_test_top.env.agent.driv [] Transaction No. = 3
# UVM_INFO uart_driver.sv(67) @ 1569455: uvm_test_top.env.agent.driv [] start = 1, tx_data_in = 3d, done_tx = 1
# UVM_INFO uart_driver.sv(32) @ 1569455: uvm_test_top.env.agent.driv [] [TRANSACTION]::TX PASS
# UVM_INFO uart_driver.sv(33) @ 1569455: uvm_test_top.env.agent.driv [] Expected data = 3d, Obtained data = 3d
# UVM_INFO uart_driver.sv(43) @ 1829865: uvm_test_top.env.agent.driv [] [TRANSACTION]::RX PASS
```

Fig 9: Transcript Output using UVM

## Waveform Output:

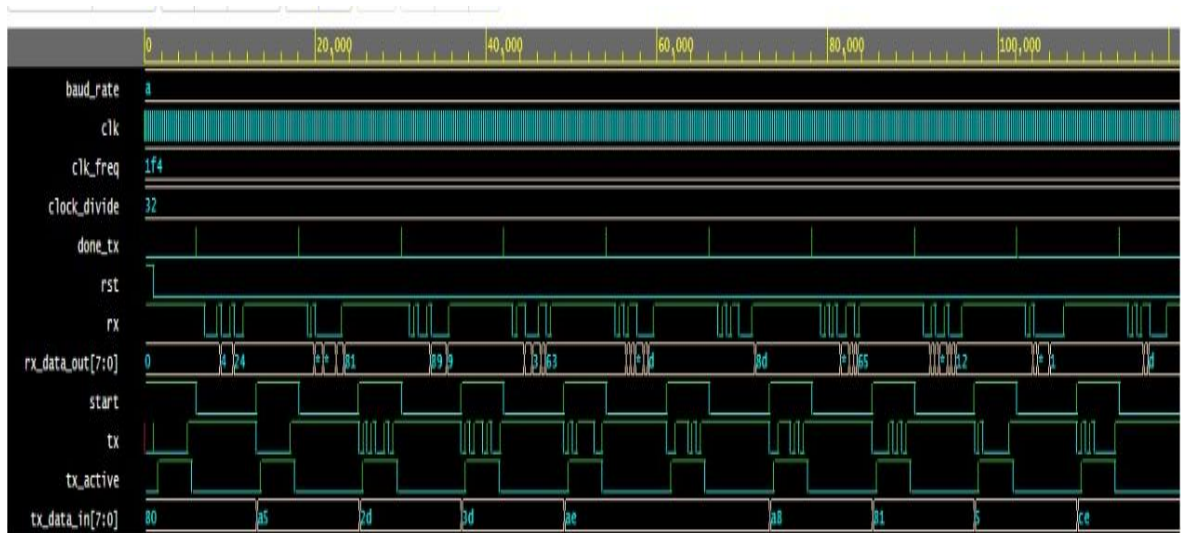


Fig 10: Waveform Output using UVM

### ii. UVM Report Summary

```
# UVM_INFO verilog_src/uvm-1.2/src/base/uvm_object.svh(1270) @ 121195: reporter [TEST_DONE] 'run' phase is ready to proce
# UVM_INFO verilog_src/uvm-1.2/src/base/uvm_report_server.svh(847) @ 121195: reporter [UVM/REPORT/SERVER]
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO : 76
# UVM_WARNING : 0
# UVM_ERROR : 0
# UVM_FATAL : 0
# ** Report counts by id
# [] 71
# [Questa UVM] 2
# [RNTST] 1
# [TEST_DONE] 1
# [UVM/RELNOTES] 1
#
# ** Note: $finish : /usr/share/questa/questasim/linux_x86_64/./verilog_src/uvm-1.2/src/base/uvm_root.svh(517)
# Time: 121195 ps Iteration: 78 Instance: /tb_uart_top
# End time: 13:06:00 on May 09,2020, Elapsed time: 0:00:07
# Errors: 0, Warnings: 7
Done
```

Fig 10: UVM Report Summary of UART

**5. Conclusion :** The functionality of UART has been verified successfully with UVM and System Verilog. Different test case has been checked. The functional coverage obtained for UART protocol is 100%.

