IEEE
CUSB

(Final Project)

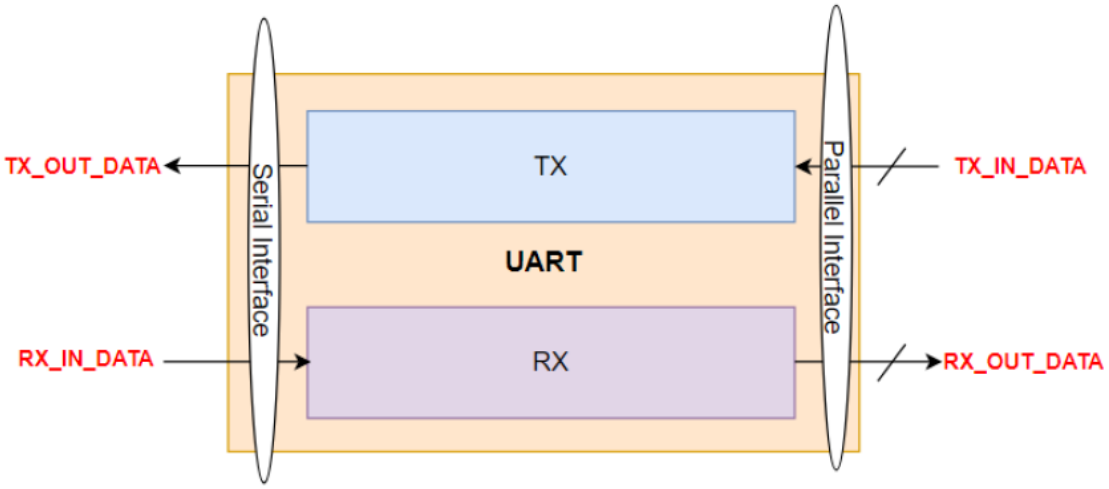
Verification Workshop
2024 / 2025

IEEE

Verification of UART Transmitter

Introduction :-

- There are many serial communication protocol as I2C, UART and SPI.
- A **U**niversal **A**synchronous **R**eceiver/**T**ransmitter (UART) is a block of circuitry responsible for implementing serial communication.
- UART is Full Duplex protocol (data transmission in both directions simultaneously)



- **Transmitting UART** converts parallel data from the master device (eg. CPU) into serial form and transmit in serial to receiving UART.
- **Receiving UART** will then convert the serial data back into parallel data for the receiving device.

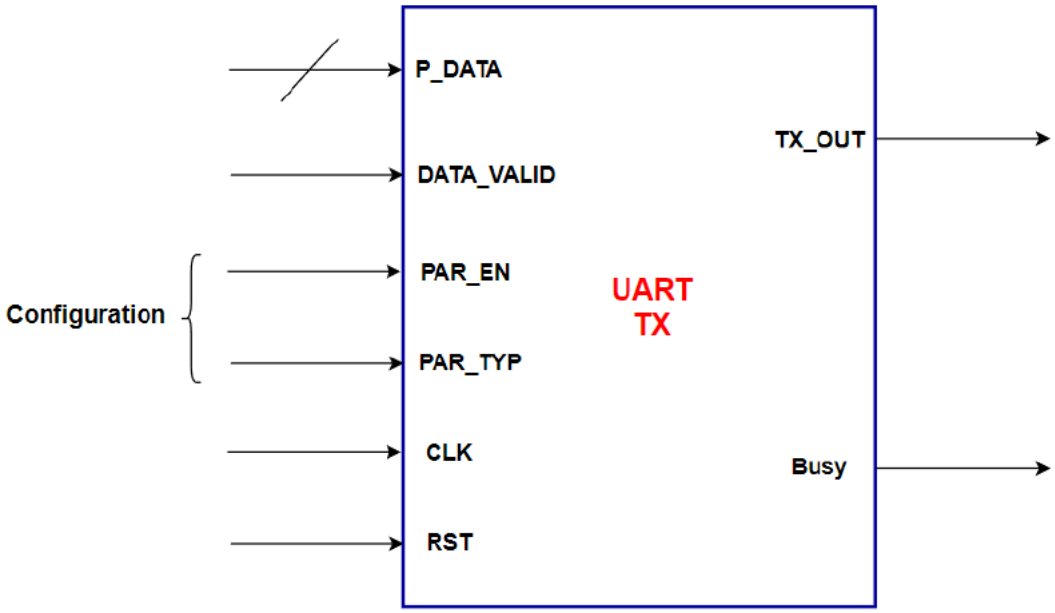
We will use a simplified version of the UART_TX

Design code:-

```
module uart_tx (
    input wire clk,
    input wire reset,
    input wire [7:0] P_DATA,
    input wire PAR_EN,
    input wire PAR_TYP,
    input wire DATA_VALID,
    output reg TX_OUT,
    output reg Busy
);

    parameter IDLE = 0, START = 1, DATA = 2, PARITY = 3, STOP = 4;

    reg [3:0] state = IDLE;
    reg [3:0] counter = 0;
    reg [7:0] data_reg;
    reg parity_bit;

    always @(posedge clk or negedge reset) begin
        if (!reset) begin
            state <= IDLE;
            TX_OUT <= 1'b1;
            Busy <= 1'b0;
            counter <= 0;
        end else begin
            case (state)
                IDLE: begin
                    TX_OUT <= 1'b1;
                    Busy <= 1'b0;
                    counter <= 0;
                    if (DATA_VALID) begin
                        data_reg <= P_DATA;
                        parity_bit <= (PAR_TYP == 0) ? ~^P_DATA : ^P_DATA;
                        state <= START;
                        Busy <= 1'b1;
                    end
                end
```

```
                START: begin
                    TX_OUT <= 1'b0;
                    state <= DATA;
                    counter <= 0;
                end

                DATA: begin
                    TX_OUT <= data_reg[counter];
                    counter <= counter + 1;
                    if (counter == 7)
                        state <= (PAR_EN) ? PARITY : STOP;
                end

                PARITY: begin
                    TX_OUT <= parity_bit;
                    state <= STOP;
                end

                STOP: begin
                    TX_OUT <= 1'b1;
                    state <= IDLE;
                end
            endcase
        end
    end

endmodule
```

Block interface :-



Signals Explanation:

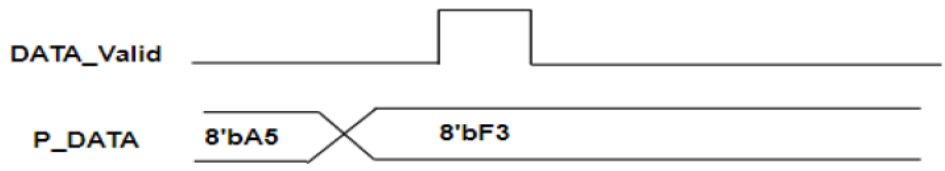
Port	Width	Description
CLK	1	UART TX Clock Signal
RST	1	Synchronized reset signal
PAR_TYP	1	Parity Type
PAR_EN	1	Parity Enable
P_DATA	8	Input data byte
DATA_VALID	1	Input data valid signal
TX_OUT	1	Serial Data OUT
Busy	1	High signal during transmission, otherwise low

Specifications:-

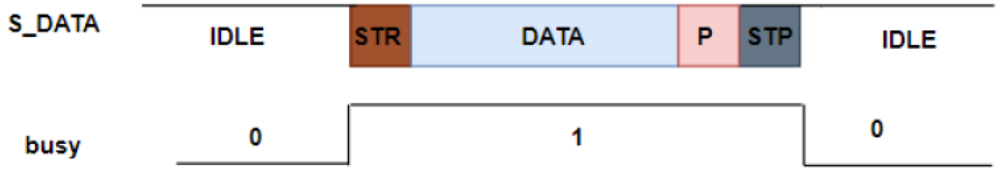
- UART TX receive the new data on **P_DATA** Bus only when **Data_Valid** Signal is **high**.
- Registers are cleared using asynchronous active low reset
- **Data_Valid** is high for only 1 clock cycle
- **Busy** signal is **high** as long as UART_TX is transmitting the frame, otherwise **low**.
- UART_TX couldn't accept any data on **P_DATA** during UART_TX processing, however **Data_Valid** get high.
- **S_DATA** is high in the **IDLE** case (No transmission).
- **PAR_EN** (Configuration)
    - 0: To disable frame parity bit
    - 1: To enable frame parity bit
- **PAR_TYP**
    - 0: Even parity bit
    - 1: Odd parity bit

Waveforms:-

Expected Input: -



Expected Output: -



PART 1 (Verification Using SV Interface) :-

I- SV Test Bench Structure

- Draw your UVM testbench showing the SV structure.
- The files needed are:
  Top, Testbench, DUT, Assertions, Monitor, Interface.
- Other needed files (Not to be drawn in the Structure):
  Do File, Shared Package(If any), Package, Golden Model(If any) and Coverage reports.

II- Verification Plan

This part outlines the verification strategy, including the methods used to validate each requirement. In the Functionality Check column, you can indicate whether the verification is performed using a golden/reference model, assertions, or a combination of both.

For reference, you can find an example of this document below.

Label	Description	Stimulus Generation	Functional Coverage	Functionality Check
LABEL1	When the rst_n is asserted, All flags & internal signals should equal 0	Randomized less often 95 off & 5% on during the simulation	-	Immediate assertion to check async rst_n functionality as it is async
LABEL2	When rst_n is deactivated & the FIFO is full of elements (conut=depth), The full flag should be high	Randomized, Write enable to be high with distribution of the value WR_EN_ON_DIST and to be low with 100-WR_EN_ON_DISTduring the simulation	Cross coverage: WRITE_READ_FULL_CROSS, covers write & read enables and full flag	Immediate assertion to check full flag functionality as it is combinational
LABEL3	When rst_n is deactivated & the FIFO has one element left(conut=depth-1), The almostfull flag should be high	Randomized, Write enable to be high with distribution of the value WR_EN_ON_DIST and to be low with 100-WR_EN_ON_DISTduring the simulation	Cross coverage: WRITE_READ_ALMOST_FULL_CROSS, covers write & read enables and almostfull flag	Immediate assertion to check almostfull flag functionality as it is combinational
LABEL4	When rst_n is deactivated & the FIFO has no element inside (conut=0), The empty flag should be high	Randomized, Read enable to be high with distribution of the value RD_EN_ON_DIST and to be low with 100-RD_EN_ON_DIST during the simulation	Cross coverage: WRITE_READ_EMPTY_CROSS, covers write & read enables and empty flag	Immediate assertion to check empty flag functionality as it is combinational
LABEL5	When rst_n is deactivated & the FIFO has only one element inside (conut=1), The empty flag should be high	Randomized, Read enable to be high with distribution of the value RD_EN_ON_DIST and to be low with 100-RD_EN_ON_DIST during the simulation	Cross coverage: WRITE_READ_ALMSOT_EMPTY_CROSS, covers write & read enables and almostempty flag	Immediate assertion to check almostempty flag functionality as it is combinational

III- Code Coverage

You should reach 100% coverage for Condition, Branch, statement and toggle coverage reports. Please add snippets from both QuestaSim and coverage reports showing your coverage with justification if you could not reach 100% coverage for the design and add code coverage exclusions to reach 100% coverage.

IV- Functional Coverage

Constraint Blocks required:

I- Assert reset to be active 3 % from the simulation time.
II- Assert PAR_TYP to toggle with duty 50%
III- Assert PAR_EN to be active 25% from the simulation time

IV- Assert DATA_VALID to be active most of simulation time

V- Assert P_DATA with the following specs:

- LSB to equal 1 80 of simulation time
- Use inside operator to take the values (11111111, 00000000, 10101010) 4% only during simulation time.

Coverpoints & Covergroup required:

Make a coverpoint for each individual signal, Make the following Cross coverpoints:

- PAR_EN & PAR_TYPE
- P_DATA & P_VALID
- TX_OUT & BUSY

You should reach 100% coverage for functional coverage report. Please add snippets from both QuestaSim and coverage reports showing your coverage with justification if you could not reach 100% coverage for the design and ignore the un-hit values using ignore_bins to reach 100% coverage.

V- Assertions

Add assertions and bind it in the top module. Assertions mustn't be less than 12 to experience all possible conditions.

Provide a table in your PDF file showing the assertions used as follows:

Feature	Assertion

Whenever the rst_n is active, All sequential flags and internal signals should be low.	`always_comb begin`
	`if(!FIFO_IF.rst_n)`
	`assert final ((!FIFO_IF.wr_ack)&&(!FIFO_IF.overflow)&&(!FIFO_IF.underflow)&&(!FIFO.wr_ptr )&&(!FIFO.rd_ptr)&&(!FIFO.count));`
	`cover final ((!FIFO_IF.wr_ack)&&(!FIFO_IF.overflow)&&(!FIFO_IF.underflow)&&(!FIFO.wr_ptr)&&(!FIFO.rd_ptr)&&(!FIFO.count));`
	`end`
	
	`@(posedge FIFO_IF.clk or negedge FIFO_IF.rst_n)`
	`(!FIFO_IF.rst_n) |-> ((!FIFO_IF.wr_ack)&&(!FIFO_IF.overflow)&&(!FIFO_IF.underflow)&&(!FIFO.wr_ptr)&&(!FIFO.rd_ptr)&&(!FIFO.count));`

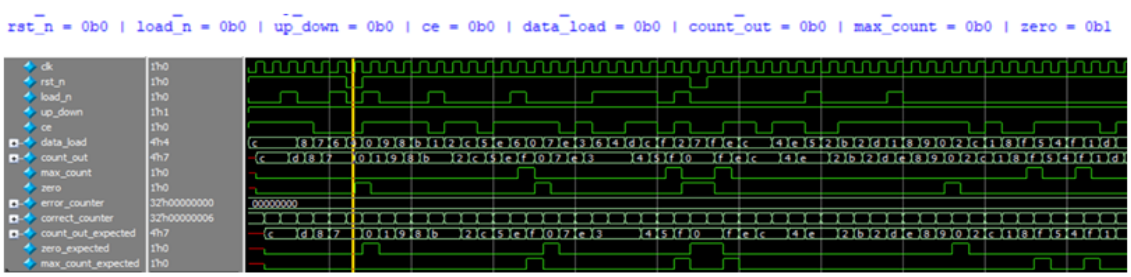
Whenever the rst_n is deactivated & number of FIFO elements equal FIFO maximum depth, full flag should be high.	`always_comb begin`
	`if((FIFO_IF.rst_n)&&(FIFO.count == FIFO_IF.FIFO_DEPTH))`
	`full_assertion: assert final (FIFO_IF.full);`
	`full_cover: cover (FIFO_IF.full);`
	`end`

Requirements:

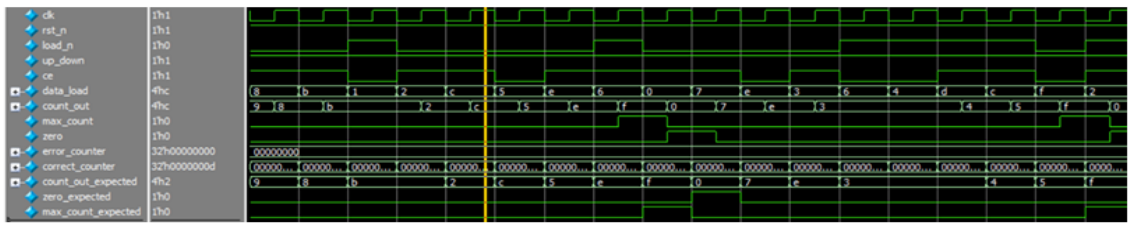
One PDF file has the following:

- Testbench code
- Top module code
- SVA module code
- Package code
- Design code
- Golden Model (If any)
- Shared Package (If any)
- Do File & Snippet to your verification requirement document
- Code Coverage, Functional Coverage and assertion coverage report snippets
- QuestaSim Simulation snippets showing labels, take this as a reference:

LABEL1 (When the rst_n is asserted, the output value should be low):



LABEL2 (When rst_n is deactivated & load_n is active count_out should equal data_load):



First Submission file:

Folder named “SV PART” containing the following:

- PDF file having the requirements for both parts with screenshots showing your work
- Testbench and design files.
- Do file to run simulation.

PART 2 (Verification Using UVM) :-

I- UVM Test Bench Structure

- Draw your UVM testbench showing the UVM structure.
- The files needed are:

  Top, Test, DUT, Assertions, Monitor, Interface, Agent, Driver, Environment, Scoreoard, Coverage, Sequencer, Sequences and sequence items.
- Other needed files (Not to be drawn in the Structure):

  Do File, Shared Package (If any), Golden Model(If any), Coverage reports and List Files(If any).
- Sequences must be at least 2:

  Reset sequence and Main sequence.

  You can add more sequence and split your work into more than 2 sequences based on your verification plan if you want.

II- Steps

Create a full UVM Environment for UART_TX then:

- Add constraints in the sequence item class.
- Add covergroups, coverpoints in the coverage collector class.
- Add assertions and bind it in the top module.

Requirements:

One PDF file has the following:

- Design
- SV Assertions
- Interface
- Top module
- Test
- Environment
- Agent
- Driver
- Monitor
- Sequence Items
- Sequencer
- Sequences
- Config. Object
- Scoreboard
- Coverage
- Do File
- List of files
- Transcricpt snippet showing UVM output.

Gentle Notes:

- If you attached the following snippets in the first part Code Coverage, Functional Coverage, Assertion coverage report snippets and QuestaSim Simulation, there is no need to attach them again in the second part.

Second Submission file:

Folder named “UVM Part” containing the following:

- PDF file having the requirements for both parts with screenshots showing your work
- Testbench and design files.
- Do file to run simulation.

Final Submission file:

Compressed file containing the two folders SV Part and UVM Part.

اللهم إنا نسألك لـ أهل غزة النّصر على من عاداهم، عاجلاً غير آجل يا رب العالمين

اللهم نستودعك أهالي غزّة وفلسطين فانصرهم واحفظهم بعينك التي لا تنام، واربط على قلوبهم وأمدهم بجُندك وأنزل عليهم سكينتك وسخر لهم الأرض ومن عليها

اللهم إنا نسألك باسمك القّهار أنْ تقهر من قهر إخواننا في غزة وفلسطين، ونسألك أن تنصرهم على القوم المجرمين

اللهم بارك جهاد المجاهدين في غزة و فلسطين، وأيدهم بجنودك ونصرك يا قوي يا كريم

اللهم انصر إخواننا المجاهدين في غزة وفلسطين، اللهم كن لهم ولا تكن عليهم

اللهم انتقم من أعدائهم، واقذف الرعب في قلوبهم وردّ كيدهم في نحورهم

اللهم انصر المرابطين المستضعفين من أهل غزة وفلسطين

اللهم إنا نسألك باسمك الأعظم أن تجعل المسجد الأقصى في حفاظتك؛ فلا يدنّسه غاصب، ولا يعتدي حرمته ظالم يا جبار السموات والأرض يا رب العالمين

اللهم اجعل نار أعدائهم عليهم بردا وسلاما يا أرحم الرّاحمين