Assignment 3

Classes & CRT

Design Specs:

This is a UART transmitter, which converts an 8-bit parallel input (data_in) into a serial data stream (tx) following the UART protocol.

Port Name	Port Type	Width	Description
clk	input	1 bit	System clock used to drive the transmitter FSM and timing.
rst_n	input	1 bit	Active-low reset. When low, it resets all internal registers and state.
tx_start	input	1 bit	Start signal to trigger transmission. Should be asserted high for 1 clock cycle when ready to send data_in.
data_in	input	8 bits	The 8-bit data byte to be transmitted serially over the tx line.
parity_en	input	1 bit	Enables parity bit transmission: 1 = add parity bit, 0 = no parity.
even_parity	input	1 bit	If parity_en is 1, this decides the parity mode: 1 = even parity 0 = odd parity.
tx	output	1 bit	UART serial output line. IDLE = 1, START = 0, DATA bits (LSB first), optional parity, STOP = 1.
tx_busy	output	1 bit	Indicates transmission is in progress. High from tx_start until the stop bit is completed.

Features & Behavior:

- ### Start Bit

  - Transmission begins with a **start bit** (`tx = 0`), marking the beginning of a frame.

- ### Data Bits

  - Transmits ***8 data bits***, ***LSB first*** *(bit 0 is sent first).*
  - The data comes from *data_in*.

- ### Optional Parity Bit

  - If parity_en = 1, a parity bit is added after the data bits.
  - Parity is calculated as:
    - **Even parity**: parity_bit = ~(data_in[0] ^ data_in[1] ^ ... ^ data_in[7])
    - **Odd parity**: parity_bit = data_in[0] ^ data_in[1] ^ ... ^ data_in[7]
  - This helps the receiver detect single-bit errors.

- ### Stop Bit

  - Always sends **1 stop bit** (tx = 1) after data (and parity if enabled).
  - The stop bit marks the end of a frame and allows the receiver to recover.

- ### FSM (Finite State Machine) States

State	Action
IDLE	Wait for **tx_start**, set **tx = 1** (idle).
START	Send start bit (**tx = 0**).
DATA	Send 8 bits of **data_in**, LSB first.
PARITY	Send computed parity bit if enabled.
STOP	Send 1 stop bit (**tx = 1**), then return to **`IDLE`**.

- ### The design operates on frequency <u>100MHz.</u>

- ### Transmission Timing Overview:

Serial transmission starts on the **next clock edge** after tx_start is sampled in the design. UART output format:

<u>start_bit (1 cycle) + 8 data bits (8 cycles) + optional parity (1 cycle) + stop bit (1 cycle)</u>

Total cycles:

- **Without parity** = 10 clock cycles (1+8+1).
- **With parity** = 11 clock cycles (1+8+1+1).

Objective:

To build a SystemVerilog **Testbench environment** to verify a UART Transmitter module, focusing on the following key concepts:

- Use of **SystemVerilog classes** to model data and stimulus generators.
- Use of **constrained randomization** to generate diverse, protocol-valid transactions.
- Application of **enumerations** for protocol configuration options.

Required Structure:

Testbench.sv

Uart_packet.sv (class)

enum_pkg.sv

Helpful Steps:

- Create a Test plan to use while building your test environment and implementing the stimulus to cover all the valid testcases.

- Implement an **enum** for the parity stimulus to be used with your randomization. And can be used to control the flow of the stimulus u are sending to the design.

- Use the Randomization and Constraint Randomization to cover all testcases and corner cases like:
  - Sending full ones of data.
  - Sending full zero of data.
  - Sending different data to cover all combinations.
  - Different cases for parity **[ODD, EVEN, NO_PARITY].**

- Implement print task in your class to help you in displaying the stimulus values in the packet ill be helpful for debugging.

- Implement **generate_stimulus()** task that allocate a new object and makes randomization.

- Store the Expected data in a **golden_model()** task in which u calculate and store the expected data in **expected_queue[$].**

- Implement **drive_stim()** task that drives the randomized stimulus to DUT and wait for tx_busy to be de-asserted to send again.

- U need to assert the **tx_start** for one cycle after your **data_in** is ready and in next cycle the serial data will be out bit by bit.

- Implement a **collect_output()** task that samples the output serial data form the UART TX design. This task will collect the serial data after de-assertion of tx_start happens and convert it back to parallel then save it in actual_queue[$].

- Implement a **check_result()** task that takes both actual and expected queue and checks the output data with expected data with appropriate print statements that displays the actual and expected data.

- Class can contain control signals which not necessary to be sent to design it maybe used in your environment to control the checker or stimulus flow and these control signals can be randomized too which gives you the flexibility to control the verification process.

- This design may be a buggy one if you detected any bugs (if exists) refer to it in your documentation.

- All these recommended hints are just a blueprint you can follow, feel free to change it with keeping the structure and tasks as required.

Deliverables:

U have to deliver

- Zip Folder contains:
  - The implemented **.sv** files and the **design files** and **run.do file**.

- **Separated PDF** contains:
  - Snippets for the implemented code.
  - Snippet for the waveform shows the different testcases
  - Snippets for the logs show the all printed values.
  - Snippet for Do file you have used for automating the process.

The delivered zip file must be named like **your_name_Assignment_1.rar** for example: Hassan_Khaled_Assignment_3.rar also the PDF file Hassan_Khaled_Assignment_3.pdf

---

Good Luck