Task 1 - Mandatory

Create a UVM_AGENT that implements data transmission over a UART interface. The interface should contain only the TX and RST signals.

- The driver must send transactions on the bus without any delays between transmissions.
- After a reset, the TX signal should be 1. Before the reset, the TX signal should be X.
- The agent must have a configuration object containing a parameter for data transmission speed (supported speeds are 19200 and 115200 bits/s).
- A monitor should be implemented to collect the transactions, send them to a TLM port (the port does not need to be connected), and print the contents to the console.
- The transaction must include:
    - a byte data field,
    - fields for start and stop bits, and
    - a one-bit field for the parity bit.

All fields must be declared random, and constraints for the start, stop bits, and parity bit fields should be implemented.

Implement sequences:

1. seq1: Randomly sends 10 transactions.
2. seq2: Sends 10 transactions with random errors in the parity, start, and stop bits (errors mean the bits take incorrect values).

Create tests:

- Create a base test with an instance of the agent.
- Create an inherited test to run seq1, then pause for 10 μs, then run seq2.
- Create a top module running the test.
- The simulation result log must contain 10 error-free transactions and some erroneous transactions captured by the monitor.

All components must be implemented in SystemVerilog using the UVM methodology. Use https://www.edaplayground.com/ for implementation and testing. The interface description can be found on the Wiki page for Universal Asynchronous Receiver-Transmitter (UART).

Once task is completed, please send a link to the testbench on edaplayground.com.

Task 2.1 - 2.4 (Optional, after Task 1):

Task 2.1:

- Add register model usage to the testbench from Task 1.
- Create 2 UVM registers, each with a size of 4 bits. Field in the register should have the same name as the registers and occupy the entire register.
- Create a uvm_reg_block and place the two registers inside it:

Register	Size	Address offset	Field name	Field size	Access type	Value after reset
R1	4 bits	0x0	R1	4 bit	RW	0x0
R2	4 bits	0x1	R2	4 bit	RO	0xA

- Create object uvm_reg_block in uvm_env
- Create adapter class and implement its methods reg2bus\bus2reg
- Pass the sequencer and adapter to uvm_reg_block.
- Create a test that issues two consecutive write commands to addresses 0x0 and 0x1 with random data values.
- A register write means 1 UART transaction on TX:
    - Bit 0 specifies the operation type (0-read, 1-write).
    - Bits [3:1] contain the address.
    - Bits [7:4] hold the data.
- Add verification that the written data matches the value returned by get_mirrored_value().

Task 2.1*:

- Extend Task 2.1 with registers model usage.
- Create 2 UVM registers, each with a size of 8 bits. Field in the register should have the same name as the registers and occupy the entire register.
- Create a uvm_reg_block and place the two registers inside it:

Register	Size	Address offset	Field name	Field size	Access type	Value after reset
R1	8 bits	0x0	R1	8 bit	RW	0x0
R2	8 bits	0x1	R2	8 bit	RO	0xA

- Create object uvm_reg_block in uvm_env

- Create adapter class and implement its methods reg2bus\bus2reg
- Pass the sequencer and adapter to uvm_reg_block.
- Create a test that issues two consecutive write commands to addresses 0x0 and 0x1 with random data values.
- A register write means 3 UART transaction on TX:
    - “write” command is to be transmitted in the first uart tx message and occupy in the least bit(0-read, 1 -write)
    - Address is to be set in the second uart tx message
    - Data ia to be transmitted in the 3rd uart tx message
- After 3 steps above data will get to the register in the DUT
- Add verification that the written data matches the value returned by get_mirrored_value().

Task 2.2:

Improve task 1

- Create UVM_AGENT, which implements data transmission and reception via UART interface.
- The interface must contain TX, RX, RST signals.
- The driver must be able to send data via TX line.
- The direction field must be added to the transaction.
- Add RX signal to the interface
- The monitor must collect data on both RX and TX lines. If the contract is collected on the TX line, the direction field has a value of 1, if on RX, then 0.
- Create 2 agents (A1 and A2) in env and connect the interface signals in the top module as follows A1.RX↔A2.TX A2.RX↔A1.TX
- Run 3 uart transactions simultaneously on each of the agents.
- The log must contain information in the form of a message for quick verification:

  <who sent>.<what command>.<at what address>.<what data>

Task 2.3:

Improve tasks 2.1 and 2.2

- Create a test that runs 2 consecutive write commands at address 0x0, 0x1, and two read commands 0x0, 0x1.
- Reading a register is 2 consecutive uart transactions on TX and one on RX collected by the monitor
    - The read command is transmitted in the first tx uart message and is located in the least significant bit (0 - reading, 1 - writing)
    - The address is transmitted in the second tx uart message and is located in the least significant bit (0 - reading, 1 - writing)
    - Data is received via rx after the two previous steps via the rx line
- After these steps, the register model will receive data from the device.
- It is recommended to use 2 identical agents (from part 1) as a receiver and transmitter, whose interfaces are connected rx<->tx tx↔rx

Task 2.4:

Improve tasks 2.1 and 2.2 and 2.3

- Improve the sequence of sending data via tx in the case of a register read command so that it saves all data received by the monitor at the address and sends them back in the case of a read command.
- It is allowed to use two register models, if necessary.
- Add to the scoreboard env, in which to implement a check that the written data matches the read data

Materials for independent study:

On SV - the standard and examples from the Internet. There ais a lot of information on the Internet. https://fpga.mit.edu/6205/_static/F23/documentation/1800-2017.pdf

On UVM: Take as a basis examples from the "UVM cookbook" uvm-cookbook.pdf, which can be downloaded here:
https://verificationacademy.com/resource/68003?download_refer=/cookbook

And here is the standard for UVM:
https://www.accellera.org/images/downloads/standards/uvm/uvm_users_guide_1.1.pdf

Additionally:

The UVM Primer - Salemy Ray

A Practical Guide to Adopting the Universal Verification Methodology - Sharon Rosenberg

Coverage Cookbook - Verification Academy

Writing Testbenches using SystemVerilog - Janick Bergeron

General questions to work through while completing the test task:

1. Explain the use of OOP in the test task - examples of inheritance, polymorphism, etc.

2. Show all examples of static and dynamic scope in the testbench

3. How the separation of visibility areas is implemented in the testbench

4. Give examples of parameterization of object types in the testbench

5. Explain how static and dynamic scopes interact in the testbench

6. How various data of various types is transferred when calling a function or task

7. How the scheduler works in SystemVerilog

8. Tell about the methods of synchronization and control of threads/processes

9. Tell about the OOP design patterns used in UVM.

10. Tell about the use of randomization in verification. Types of randomization in SystemVerilog

11. Tell about the UVM phases of testbench creation and execution

12. Tell about the possible mechanisms of interaction sequence-driver-sequencer, monitor-scoreboard