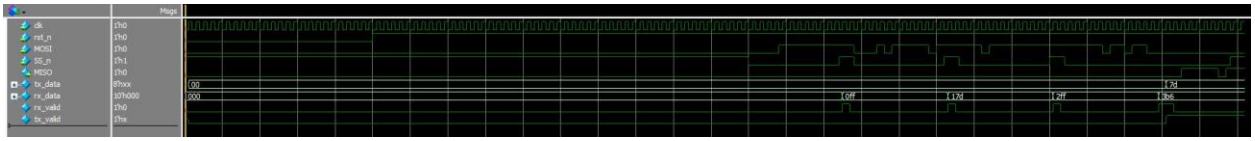
UVM Project

Objective: The project is divided into three main parts, with each part focusing on building and extending UVM environments for the individual components and the complete system. The final integration will demonstrate reuse of UVM environments.

Important Notes:

- **Options**: There are two options either to do only part 1 of this project as individual project or do the full 3 parts as a team project.
- **Design Specifications:** explained in details in the link here.
- **Golden Model implementation:** Write your golden model as a Verilog design module to make it easier for you and practice writing the design as well (Reach out to your assistant in case you are not sure how the scoreboard will check the DUT output against yours. Also, reach out to your assistant in case you implement the golden model as a task)
- **Team Size:** 3 and you must insert your team members here.
- **Here is the snippet of expected waveform**:

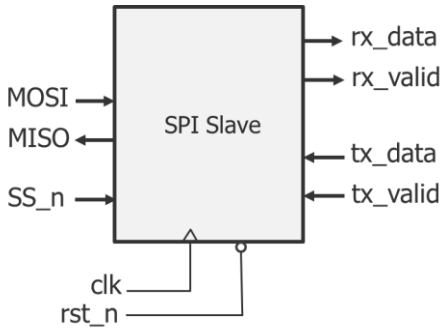


The project will be executed in three distinct phases:

- RAM Verification.
- SPI Slave Verification.
- SPI Wrapper Verification re-using the RAM and SPI Slave environments.

Part 1: UVM Environment for SPI-Slave

In this part, you will create a full UVM environment to verify the **SPI slave design**.



Sequences Requirements:

Split the sequences into the following sequences. You can switch on and off the constraints or use inline constraints to control the sequence item generated in the sequence.

- reset_sequence
- main_sequence

Constraint Requirements:

1- The reset signal (rst_n) shall be deasserted most of the time.

2- The SS_n signal to be high for one cycle every 13 cycles for all cases except read data to be high for one cycle every 23 cycles.

3- Declare a randomized array to drive bit by bit the MOSI and the first 3 bits sent serially to the MOSI after the SS_n falls and reach chk_cmd are valid combinations only (000, 001, 110, 111).

4- The tx_valid signal to be high in case of read data.

(Hint: You can use pre_randomize, post_randomize and inline constraints)

Functional Coverage Requirements:

1- Add coverpoints on rx_data[9:8] to take all possible values and all possible transitions.

2- Add coverpoints on SS_n to capture:

- Check full transaction duration: 1 → 0 [*13] → 1 for normal operations.
- Check extended transaction: 1 → 0 [*23] → 1 for READ_DATA.

3- Add coverpoints on MOSI to validate correct transitions:

- 000 (Write Address).
- 001 (Write Data).
- 110 (Read Address).
- 111 (Read Data).

4- Cross coverage between SS_n and MOSI bins (Exclude irrelevant bins to focus on legal operation scenarios).

Assertions Requirements:

1- An assertion ensures that whenever reset is asserted, the outputs (MISO, rx_valid, and rx_data) are all low.

2- Write assertions to check that after any valid command sequence (write_add_seq (000), write_data_seq (001), read_add_seq (110), or read_data_seq (111)), the rx_valid signal must assert exactly after 10 cycles and the SS_n should eventually after the 10 cycles to close communication.

3- Add the following assertions in the design to check correct FSM transitions and guard the assertions using conditional compilation with the `ifdef directive with macro named "SIM", and then include the macro in the vlog command +define+SIM option in the vlog command of your do file. Refer to this link to learn more about conditional compilation.

- IDLE → CHK_CMD
- CHK_CMD → WRITE or READ_ADD or READ_DATA
- WRITE → IDLE
- READ_ADD → IDLE
- READ_DATA → IDLE

Note: You are free to add assertions, covergroups or constraints to enrich your verification.

Part 2: UVM Environment for Single-Port RAM

In this part, you will create a full UVM environment to verify the **RAM design** with default parameters (MEM_DEPTH: 256, ADDR_SIZE: 8)

Interface:

Name	Type	Size	Description
din	Input	10 bits	Data Input
clk		1 bit	Clock
rst		1 bit	Active low synchronous reset
rx_valid		1 bit	If HIGH: accept din[7:0] to save the write/read address internally or write a memory word depending on the most significant 2 bits din[9:8]
dout	Output	8 bits	Data Output
tx_valid		1 bit	Whenever the command is memory read the tx_valid should be HIGH

Sequences Requirements:

Split the RAM sequences into the following sequences. You can switch on and off the constraints or use inline constraints to control the sequence item generated in the sequence

- reset_sequence
- write_only_sequence
- read_only_sequence
- write_read_sequence

Constraint Requirements:

1- The reset signal (rst_n) shall be deasserted most of the time.

2- The rx_valid signal shall be asserted most of time.

3- For a *write-only sequence*, every Write Address operation shall always be followed by either Write Address or Write Data operation.

4- For a *read-only sequence*, every Read Address operation shall always be followed by Read Data. After a Read Data operation shall always be followed by Read Address.

5- For a ***randomized read/write sequence***, the following ordering rules shall be enforced:

- Every Write Address operation shall always be followed by either Write Address or Write Data operation.
- After a Write Data, the next operation shall be chosen with the following probability distribution: 60% → Read Address & 40% → Write Address.
- Every Read Address operation shall always be followed by Read Data operation.
- After a Read Data, the next operation shall be chosen with the following probability distribution: 60% → Write Address & 40% → Read Address.

(Hint: You can use pre_randomize, post_randomize and inline constraints)

Functional Coverage Requirements:

1- Write Coverpoint to check transaction ordering for din[9:8]:

- Check din[9:8] takes 4 possible values
- Check write data after write address
- Check read data after read address
- Check write address => write data => read address => read data

2- Cross coverage:

- Between all bins of din[9:8] and rx_valid signal when it is high
- Between din[9:8] when it equals read data and tx_valid when it is high

Assertions Requirements: (*Add the assertions file and bind it in the top module)*

1- An assertion ensures that whenever reset is asserted, the output signals (tx_valid and dout) are low

2- An assertion checks that during address or data input phases (write_add_seq, write_data_seq, read_add_seq), the tx_valid signal must remain deasserted.

3- An assertion checks that after a read_data_seq occurs, the tx_valid signal must rise to indicate valid output and after it rises by one clock cycle, it should eventually fall.

4- An assertion checks that every **Write Address** operation must be eventually followed by a **Write Data** operation.

5- An assertion checks that every **Read Address** operation must be eventually followed by a **Read Data** operation.

Note: You are free to add assertions, covergroups or constraints to enrich your verification.

Part 3: UVM Environment for SPI Wrapper

In this part, you will build a complete UVM environment to verify the end to end functionality of the **SPI wrapper** design. The environment should integrate both the **Single-Port RAM** and the **SPI Slave** environments developed previously to reuse them but with **passive** agents (check the last extra assignment to learn how to do it).

Constraint Requirements:

1- The reset signal (rst_n) shall be deasserted most of the time.

2- The SS_n signal to be high for one cycle every 13 cycles for all cases except read data to be high for one cycle every 23 cycles.

3- Declare a randomized array to drive bit by bit the MOSI and the first 3 bits sent serially to the MOSI after the SS_n falls and reach chk_cmd are valid combinations only (000, 001, 110, 111)

4- For a *write-only sequence*, every Write Address operation shall always be followed by either Write Address or Write Data operation.

5- For a *read-only sequence*, every Read Address operation shall always be followed by Read Data. After a Read Data operation shall always be followed by Read Address.

6- For a ***randomized read/write sequence***, the following ordering rules shall be enforced:

- Every Write Address operation shall always be followed by either Write Address or Write Data operation.
- After a Write Data, the next operation shall be chosen with the following probability distribution: 60% → Read Address & 40% → Write Address.
- Every Read Address operation shall always be followed by Read Data operation.
- After a Read Data, the next operation shall be chosen with the following probability distribution: 60% → Write Address & 40% → Read Address.

(Hint: You can use pre_randomize, post_randomize and inline constraints)

Assertions Requirements:

1- An assertion ensures that whenever reset is asserted, the output (MISO) is inactive.

2- An assertion to make sure that the MISO remains with a stable value eventually as long as it is not a read data operation

>> Bind the assertion modules for the RAM and SPI Slave and SPI wrapper in the top module

Sequences Requirements:

Split the sequences into the following sequences. You can switch on and off the constraints or use inline constraints to control the sequence item generated in the sequence

- reset_sequence
- write_only_sequence
- read_only_sequence
- write_read_sequence

Note: You are free to add assertions, covergroups or constraints to enrich your verification.

Requirements:

- Verification plan
- Draw your UVM testbench showing the UVM structure using PPT, draw.io or MS Visio
  - Write a section where you will describe in detail how your UVM testbench work.
- Code snippets
- Code Coverage report
  - 100% (You can exclude the 2-D RAM declaration from the toggle coverage in the RAM design)
- Functional Coverage report, and Sequential Domain Coverage report
- Bug report
- Sections with QuestaSim snippets to for each UVM sequence and how the interface is driven in each sequence using the waveform snippets
- Also, provide a table showing the assertions used as follows:

Feature	Assertion
Whenever the rst is asserted, MISO is low	@(posedge clk) (!rst_n |-> ~MISO)

Submission File:

.rar file (<your_name>_Project2 for example Kareem_Waseem_Project2) having:

1- 3 Folders (Folder for each part containing the design files, UVM Codes, Do File, and the coverage report)
2- PDF file **has the requirements above** with your names on the first page. **Please use the same certificate names here**.