Feature	Assertion
Checks that when rst_n is asserted, MISO & rx_valid & rx_data & received_address are all low and cs turns back to IDLE state	`@(posedge clk) (!rst_n) |=> (!MISO && !rx_valid && (rx_data == 10'b00_0000_0000) && (cs == IDLE) && !received_address)`
Checks that when SS_n is deasserted and after 2 clk cycles MOSI is low, the MOSI should be low after another one cycle to have a right sequence of MOSI that drives the SPI to function a write operation correctly with RAM	`@(posedge clk) disable iff (!rst_n) (SS_n == 1 ##2 MOSI == 0) |=> MOSI == 0`
Checks that when SS_n is deasserted and after 2 clk cycles MOSI is high, the MOSI should be high after another one cycle to have a right sequence of MOSI that drives the SPI to function a read operation correctly with RAM	`@(posedge clk) disable iff (!rst_n) (SS_n == 1 ##2 MOSI == 1) |=> MOSI == 1`
Checks that when SS_n is deasserted and is followed by right sequence of MOSI, the rx_valid is asserted after 10 clk cycles to transmit the data to RAM and SS_n is asserted eventually after that to end communication with the master	`sequence write1_seq_s;` `SS_n == 1 ##2 MOSI == 0 ##1 MOSI == 0 ##1 MOSI == 0;` `endsequence` `sequence read1_seq_s;` `SS_n == 1 ##2 MOSI == 1 ##1 MOSI == 1 ##1 MOSI == 0;` `endsequence` `sequence write2_seq_s;` `SS_n == 1 ##2 MOSI == 0 ##1 MOSI == 0 ##1 MOSI == 1;` `endsequence` `sequence read2_seq_s;` `SS_n == 1 ##2 MOSI == 1 ##1 MOSI == 1 ##1 MOSI == 1;` `endsequence` `sequence rx_ss_n_s;` `##10 rx_valid ##[1:$] SS_n;` `endsequence` `@(posedge clk) disable iff (!rst_n) (write1_seq_s or read1_seq_s or write2_seq_s or read2_seq_s) |-> rx_ss_n_s`
Checks that when SS_n is deasserted, the current state turns back to IDLE state to end communication and be ready for a new one	`@(posedge clk) disable iff (!rst_n) (SS_n) |=> (cs == IDLE)`
Checks that when SS_n is asserted & the current state is IDLE, the next current state is CHK_CMD	`@(posedge clk) disable iff (!rst_n) (cs == IDLE && !SS_n) |=> (cs == CHK_CMD)`

Checks that when SS_n is asserted & the current state is IDLE, if the MOSI == 0 then the next current state is WRITE	`@(posedge clk) disable iff (!rst_n) (cs == CHK_CMD && !SS_n && !MOSI) |=> (cs == WRITE)`
Checks that when SS_n is asserted & the current state is IDLE, if the MOSI == 1 and the received_address is high then the next current state is READ_DATA	`@(posedge clk) disable iff (!rst_n) (cs == CHK_CMD && !SS_n && MOSI && received_address) |=> (cs == READ_DATA)`
Checks that when SS_n is asserted & the current state is IDLE, if the MOSI == 1 and the received_address is low then the next current state is READ_ADD	`@(posedge clk) disable iff (!rst_n) (cs == CHK_CMD && !SS_n && MOSI && !received_address) |=> (cs == READ_ADD)`
Checks that when SS_n is deasserted & the current state is WRITE, the current state stays WRITE for the next cycle	`@(posedge clk) disable iff (!rst_n) (cs == WRITE && !SS_n) |=> (cs == WRITE)`
Checks that when SS_n is deasserted & the current state is READ_ADD, the current state stays READ_ADD for the next cycle	`@(posedge clk) disable iff (!rst_n) (cs == READ_ADD && !SS_n) |=> (cs == READ_ADD)`
Checks that when SS_n is deasserted & the current state is READ_DATA, the current state stays READ_DATA for the next cycle	`@(posedge clk) disable iff (!rst_n) (cs == READ_DATA && !SS_n) |=> (cs == READ_DATA)`
Checks that when the current state is CHK_CMD, the counter resets to 10 to start a new operation	`@(posedge clk) disable iff (!rst_n) (cs == CHK_CMD) |=> (counter == 4'b1010)`
Checks that when the current state is a normal operation, the counter should decrement every clk cycle	`@(posedge clk) disable iff (!rst_n) ((cs == WRITE || cs == READ_ADD || cs == READ_DATA) && counter > 0) |=> (counter == $past(counter) - 1'b1)`
Checks that when the current state is a READ_ADD & the counter is zero, the received_address should be high to make the next read state READ_DATA	`@(posedge clk) disable iff (!rst_n) (cs == READ_ADD && counter == 0) |=> (received_address)`
Checks that when the current state is a READ_DATA & the counter is zero & tx_valid is assertes, the received_address should be low to make the next read state READ_ADD	`@(posedge clk) disable iff (!rst_n) (cs == READ_DATA && tx_valid && counter == 0) |=> (!received_address)`