I2C protocol

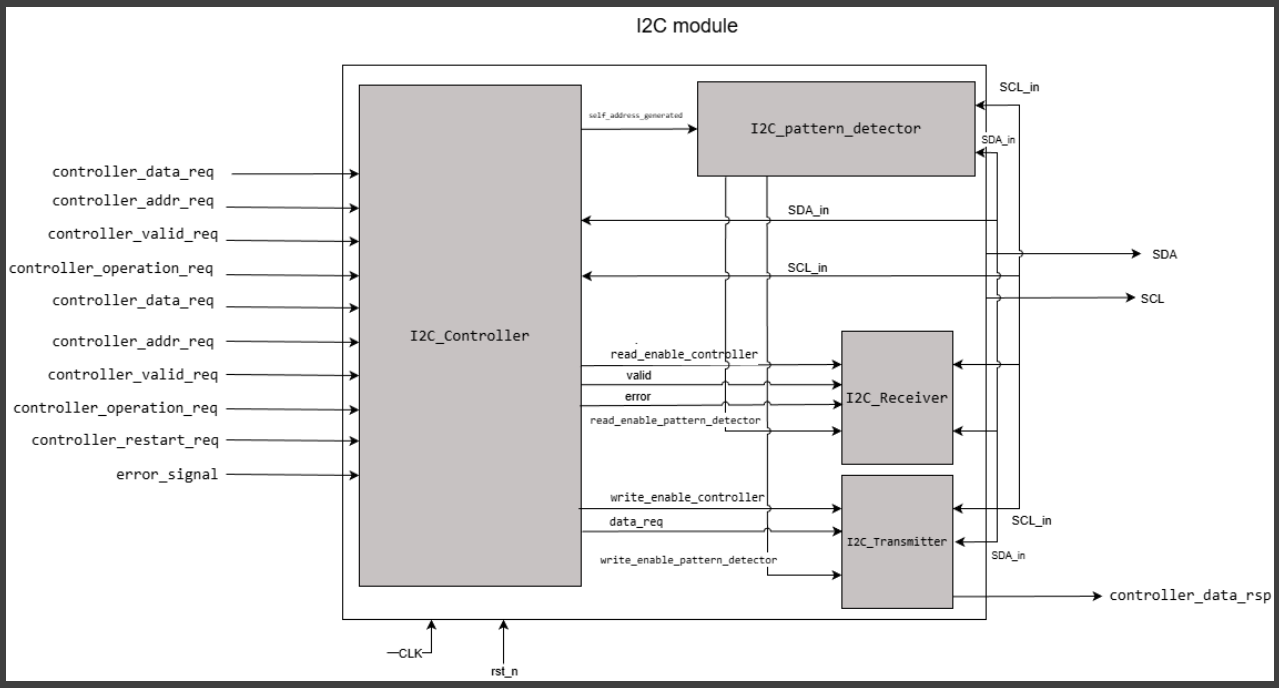
topics

1. Overview of I2C protocol

1.1 Port Interfaces

Name	Type	Width	Description
clk	Input	1	System clock.
rst_n	Input	1	System reset.
controller_data_req	input	DATA_WIDTH	The data given from the controller to be sent.
controller_addr_req	input	ADDR_WIDTH	The address given from the controller to which the data will be sent.
controller_valid_req	input	1	Indication from the controller that a valid transaction has to happen
controller_operation_req	input	1	Indication from the controller to which operation will happen.
controller_restart_req	input	1	Indication from the controller to restart the transaction.
error_signal	input	1	Indication from the controller that an error happened
controller_data_rsp	output	DATA_WIDTH	The data received from the other I2C module.
SDA	inout	1	Common bus on which the data is sent
SCL	inout	1	Common bus on which the synchronization between the I2C modules happen.

1.2 Block Diagram



1.3 Functional Description

- I2C is a communication protocol that send the data from one/multiples controllers into one/multiple targets where each target has a different address.
- Each address must detect the address for which the transaction target so it can start sampling and responding to this transaction.
- The data is sent through the SDA line, and the synchronization happens through the SCL line.
- The SDA and the SCL lines are a pull-up wires means that when no one is forces a zero on them the value on them will be 1.
- The controller and the target both are sharing the same SDA and SCL lines to communicate between each other.
- In case of multiple controllers, clock synchronization and arbitration have to be used to avoid multiple masters to drive the SCL and the SDA lines.
- SDA has to be stable while the SCL line is 1 and start changing when the SCL line becomes 0 (see Figure 1)

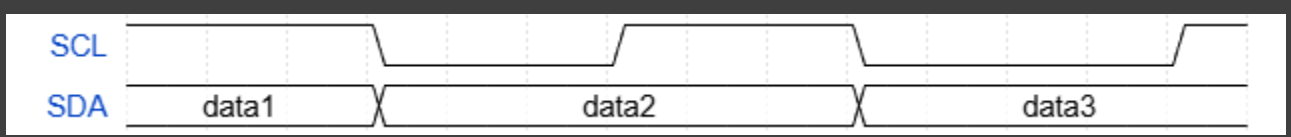
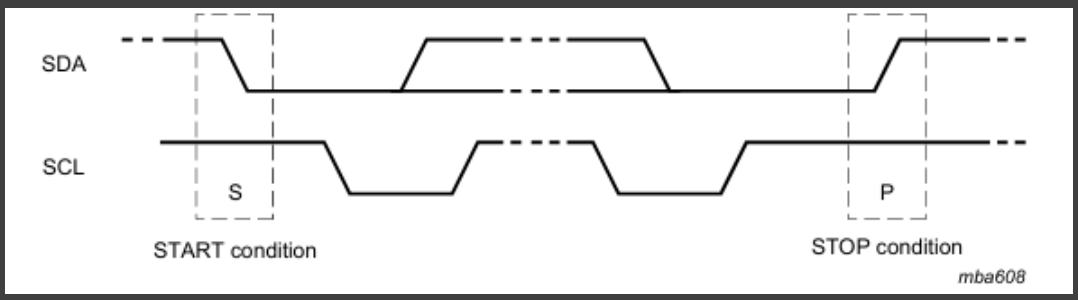


Figure 1

- The controller indicate the start and stop of each transaction through special sequences.

- The start sequence is when the SDA changes from 1 to 0 while the SCL line is 1.

- The stop sequence is when the SDA changes from 0 to 1 while the SCL line is 1.

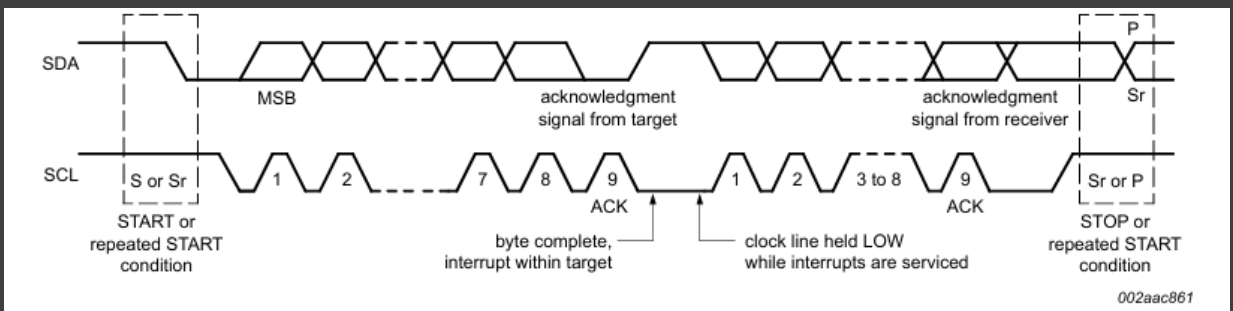


- When the target detect the start sequence it should start sampling the address from the SDA lane and if the address is the one assigned to it, the target should respond with ACK to the controller, else it responds with NACK.

- ACK and NACK are the responds of the targets for the controller transactions.

- They respond after each byte and ACK is represented by 0 on the SDA line while the NACK is a 1 on the SDA line.

- The controller should detect the ACK/NACK respond from the target and move-on based on this respond.



conditions that lead to the generation of a NACK:

1. No receiver is present on the bus with the transmitted address so there is no device to respond with an acknowledge.
2. The receiver is unable to receive or transmit because it is performing some real-time function and is not ready to start communication with the controller.
3. During the transfer, the receiver gets data or commands that it does not understand.
4. During the transfer, the receiver cannot receive any more data bytes.
5. A controller-receiver must signal the end of the transfer to the target transmitter.

2. I2C RTL Code.

2.1 I2C_interface module

```
////////////////////////////////////////////////////////////////////////////////
//Name: Abdelrahman Mohamed Ragab
// Module-Name: I2C_interface
////////////////////////////////////////////////////////////////////////////////

interface I2C_interface(clk);

parameter ADDRESS = 7'h10;
parameter ADDRESS_WIDTH = 7;
parameter DATA_WIDTH = 8;
parameter FORWARDED_CLOCK_SPEED = 100000;
parameter LOCAL_CLOCK_SPEED = 1000000;

input clk;
logic rst_n;

//input signals
logic SCL_in;
logic SDA_in;
logic [DATA_WIDTH-1:0] controller_data_req;
logic [ADDRESS_WIDTH-1:0] controller_addr_req;
logic controller_valid_req;
logic controller_operation_req;
logic controller_restart_req;
logic error_signal;

//output signals
logic [DATA_WIDTH-1:0] controller_data_rsp;
logic write_enable;
logic read_enable;
logic SDA_out;
logic SCL_out;
logic self_address_generated;
logic start_state;
logic stop_state;

endinterface //I2C_interface
```

2.2 I2C_Controller module

2.2.1 Verilog code

```
////////////////////////////////////////////////////////////////////////////////
//Name: Abdelrahman Mohamed Ragab
// Module-Name: I2C_Controller
////////////////////////////////////////////////////////////////////////////////

module I2C_Controller #(
    parameter DATA_WIDTH = 8,
    parameter ADDR_WIDTH = 7,
    parameter FORWARDED_CLOCK_SPEED = 100000,
    parameter LOCAL_CLOCK_SPEED = 1000000
) (
    input logic clk,
    input logic rst_n,
    input logic [DATA_WIDTH-1:0] controller_data_req,
    input logic [ADDR_WIDTH-1:0] controller_addr_req,
    input logic controller_valid_req,
    input logic controller_operation_req,
    input logic controller_restart_req,
    input logic controller_error_req,
    input logic SCL_in,
    input logic SDA_in,
    output logic SCL_out,
    output logic SDA_out,
    output logic write_enable,
    output logic read_enable,
    output logic self_address_generated
);

localparam IDLE_STATE = 3'b000;
localparam START_STATE = 3'b001;
localparam ACTIVE_ADDRESS_STATE = 3'b010;
localparam ACTIVE_DATA_STATE = 3'b011;
localparam STOP_STATE = 3'b100;

localparam BIT_NO = $clog2(DATA_WIDTH);

logic SDA_reg;
logic SDA_sync;
logic SDA_comb;
logic divided_clk;
logic divided_clk_rst_n;
logic operation;
logic [ADDR_WIDTH-1:0] address;
logic [2:0] NS, CS;

logic [BIT_NO-1:0] bit_sel;
logic [BIT_NO-1:0] bit_sel_next;
logic [4:0] counter;
logic [4:0] counter_next;
logic stop_counter;
logic stop_counter_next;
logic data_sent;
logic ACK_check;
logic stop_ACK;
logic stop_req;
logic restart_req;
logic error;
```

```
clock_divider #(.INPUT_CLOCK_SPEED(LOCAL_CLOCK_SPEED),.OUTPUT_CLOCK_SPEED(FORWARDED_CLOCK_SPEED)) clock_divider_inst (.clk(clk),.rst_n(divided_clk_rst_n),.divided_clk(divided_clk));

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        CS <= IDLE_STATE;
        stop_counter <= 0;
    end else begin
        CS <= NS;
        stop_counter <= stop_counter_next;
    end
end

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        SDA_sync <= 0;
    end else begin
        if (!SCL_out)  SDA_sync <= SDA_reg;
    end
end

always @(negedge SCL_in) begin
    if (CS == ACTIVE_DATA_STATE && counter == 9) stop_ACK = 1;
    else stop_ACK = 0;
end

always @(posedge SCL_in or negedge rst_n) begin
     if (!rst_n) begin
         counter <= 0;
         bit_sel <= 0;
     end else begin
        if ((ACK_check && counter == 7 && CS == ACTIVE_DATA_STATE) || (ACK_check && counter == 8 && CS == ACTIVE_ADDRESS_STATE)) begin
            if (SDA_in || !controller_valid_req) begin
                stop_req <= 1;
            end else if (controller_restart_req) begin
                restart_req <= 1;
            end
        end else begin
            stop_req <= 0;
            restart_req <= 0;
        end
        counter <= counter_next;
        bit_sel <= bit_sel_next;
    end
 end

 always @(*) begin
    if (CS == START_STATE || CS == STOP_STATE || CS == IDLE_STATE || (error &&  CS == ACTIVE_DATA_STATE)) begin
        SDA_out = SDA_comb;
    end else begin
        SDA_out = SDA_sync;
    end
end
```

```
always @(*) begin
    case (CS)
        IDLE_STATE: begin
            SCL_out = 1;
            SDA_comb = 1;
            data_sent = 0;
            divided_clk_rst_n = 1;
            stop_counter_next = 0;
            write_enable = 0;
            read_enable = 0;
            ACK_check = 0;
            bit_sel_next = 0;
            counter_next = 0;
            error = 0;
            self_address_generated = 0;
            if (controller_valid_req) begin
                NS = START_STATE;
                address = controller_addr_req;
                operation = controller_operation_req;
            end else NS = IDLE_STATE;
        end

        START_STATE: begin
            SDA_comb = 0;
            SCL_out = 1;
            divided_clk_rst_n = 0;
            self_address_generated = 1;
            bit_sel_next = 0;
            counter_next = 0;
            NS = ACTIVE_ADDRESS_STATE;
        end
```

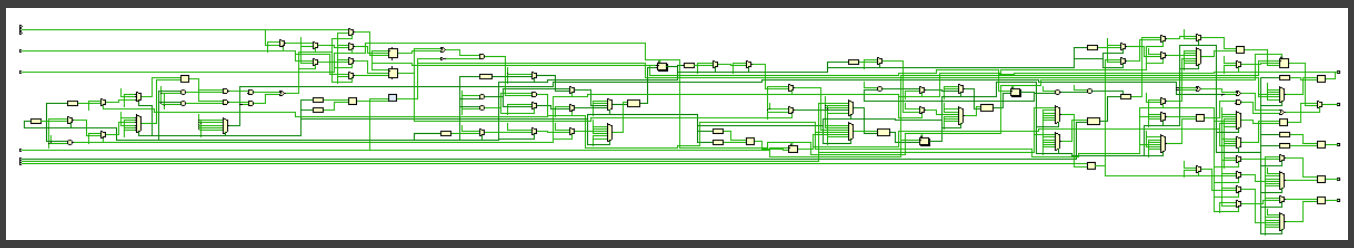
```
ACTIVE_ADDRESS_STATE: begin
    divided_clk_rst_n = 1;
    SCL_out = divided_clk;
    case (ADDR_WIDTH)
        7: begin
            if (counter < 7) begin
                SDA_reg = address [ADDR_WIDTH-bit_sel-1];
                bit_sel_next = bit_sel + 1;
                counter_next = counter + 1;
            end else if (counter == 7) begin
                SDA_reg = operation;
                counter_next = counter + 1;
            end else if (counter == 8) begin
                SDA_reg = 1;
                bit_sel_next = 0;
                counter_next = counter + 1;
                ACK_check = 1;
            end else if (counter == 9) begin
                NS = ACTIVE_DATA_STATE;
                counter_next = 0;
            end
        end
    endcase
end

ACTIVE_DATA_STATE: begin
    SCL_out = divided_clk;
    if (counter >= 8 && ((stop_req || restart_req) && !SCL_out)) begin
        if (stop_req) begin
                NS = STOP_STATE;
            end else begin
                NS = IDLE_STATE;
            end
            error = 1;
            SDA_comb = 0;
            counter_next = 0;
            data_sent = 0;
            ACK_check = 0;
            read_enable = 0;
            write_enable = 0;
    end else begin
        if (!operation) write_enable = 1;
        else read_enable = 1;
        if (counter >= 8) counter_next = 0;
        else counter_next = counter + 1;
        data_sent = 1;
        if (counter == 7) ACK_check = 1;
        else ACK_check = 0;
        SDA_reg = 1;
    end
end
```

```
            STOP_STATE: begin
                error = 0;
                SCL_out = 1;
                stop_counter_next = 1;
                if (stop_counter) begin
                    SDA_comb = 1;
                    NS = IDLE_STATE;
                end

            end
        endcase
    end
    endmodule
```

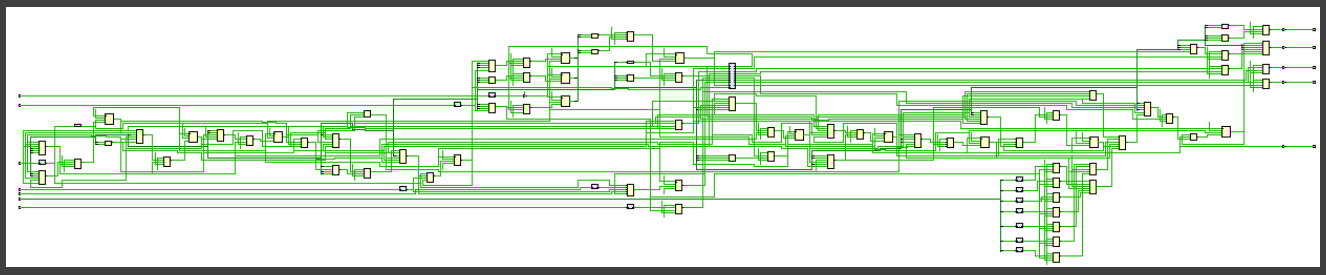
```
module clock_divider #(
    parameter INPUT_CLOCK_SPEED = 2000000,
    parameter OUTPUT_CLOCK_SPEED = 100000
) (
    input logic clk,
    input logic rst_n,
    output logic divided_clk
);

localparam DIVIDE_BY = INPUT_CLOCK_SPEED / (2*OUTPUT_CLOCK_SPEED) ;
localparam DIVIDE_BY_WIDTH = $clog2(DIVIDE_BY);

logic [DIVIDE_BY_WIDTH-1 : 0] counter;

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        counter <= 0;
        divided_clk <= 0;
    end else if (counter == DIVIDE_BY-1) begin
        counter <= 0;
        divided_clk <= ~divided_clk;
    end else
        counter <= counter + 1;
end

endmodule
```

2.2.2 RTL elaboration and synthesis schematic.

RTL elaboration



synthesis schematic



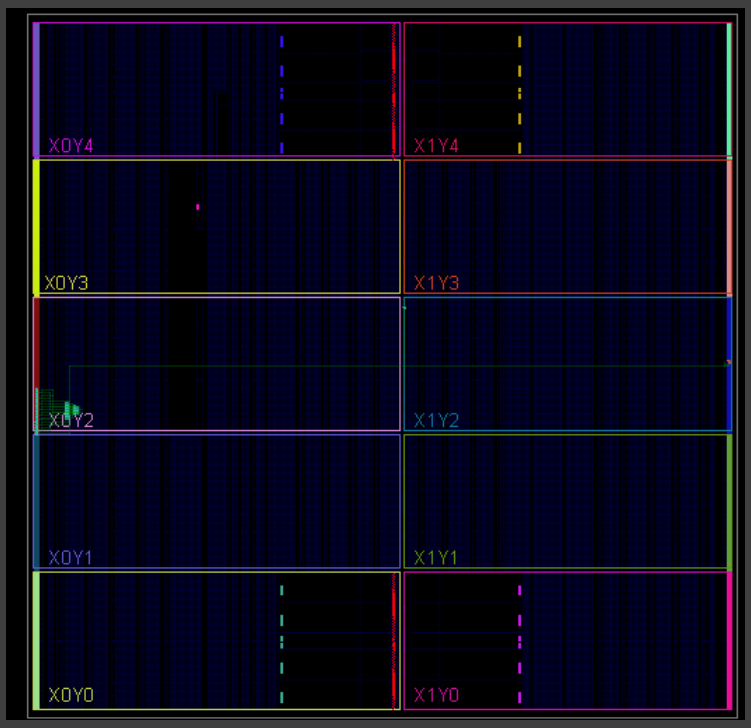
2.2.3 static time analysis reports and utilization.

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	4.596 ns	Worst Hold Slack (WHS):	0.131 ns	Worst Pulse Width Slack (WPWS):	2.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	5	Total Number of Endpoints:	5	Total Number of Endpoints:	10

All user specified timing constraints are met.

Name	Slice LUTs (134600)	Slice Registers (269200)	Bonded IOB (500)	BUFGCTRL (32)
N I2C_Controller	47	46	18	1
clock_divider_inst (clo...	11	4	0	0

2.2.4 implementation schematic



2.2.5 static time analysis reports and utilization.

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	4.828 ns	Worst Hold Slack (WHS):	0.249 ns	Worst Pulse Width Slack (WPWS):	2.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	5	Total Number of Endpoints:	5	Total Number of Endpoints:	10

All user specified timing constraints are met.

Name	Slice LUTs (133800)	Slice Registers (267600)	Slice (33450)	LUT as Logic (133800)	LUT Flip Flop Pairs (133800)	Bonded IOB (500)	BUFGCTRL (32)
I2C_Controller	47	46	20	47	16	18	1
clock_divider_inst (clo...	11	4	7	11	2	0	0

2.3 I2C_pattern_detector module

2.3.1 Verilog code

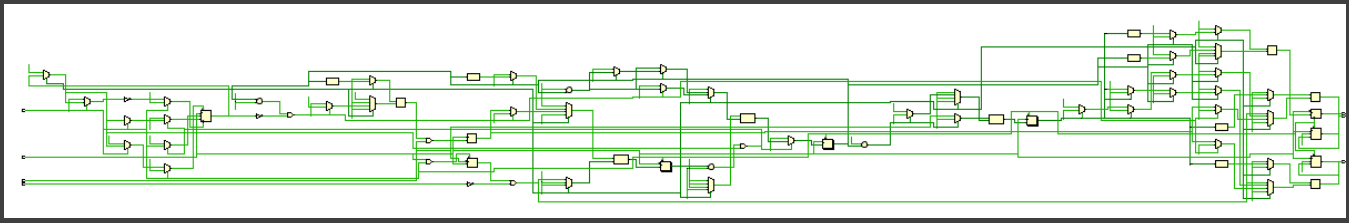
```
module I2C_Pattern_Detector #(
    parameter ADDRESS = 7'h10,
    parameter ADDRESS_WIDTH = 7
) (
    input logic SCL_in,
    input logic SDA_in,
    input logic rst_n,
    input logic self_address_generated,
    output logic SDA_out,
    output logic wr_enable,
    output logic rd_enable
);

parameter START_STOP_DECISION = 2'b00;
parameter OPERATING_DECISION = 2'b01;
parameter FINISH_DETECTION = 2'b10;

logic start_state, stop_state;

logic [ADDRESS_WIDTH:0] control_signal;
logic [ADDRESS_WIDTH:0] control_signal_reg;

logic [1:0] CS, NS;
logic [4:0] counter;
logic [4:0] counter_next;

logic SDA_out_reg;
logic ACK;
logic NACK_detected;
logic detect_ACK;
logic ack_done;
logic first_time;
logic reset_start_stop;

assign reset_start_stop = SCL_in & rst_n;

always @(negedge SCL_in or negedge rst_n) begin
    if (!rst_n) begin
        CS <= START_STOP_DECISION;
        SDA_out <= 1;
    end else begin
        CS <= NS;
        SDA_out <= SDA_out_reg;
    end
end

always @(posedge SCL_in or negedge rst_n) begin
    if (!rst_n) begin
        counter <= 0;
        control_signal <= 0;
        wr_enable <= 0;
        rd_enable <= 0;
    end else begin
        if (CS == OPERATING_DECISION) begin
            first_time = 1;
        end
        if (CS == FINISH_DETECTION && first_time)begin
            counter <= 0;
            first_time <= 0;
        end else counter <= counter_next;

        control_signal <= control_signal_reg;
    end
end
```

```
always @(posedge SCL_in or negedge rst_n) begin
    if (detect_ACK && SDA_in) begin
        NACK_detected <= 1;
    end else NACK_detected <= 0;
end

always @(negedge SDA_in or negedge reset_start_stop) begin
    if (!reset_start_stop) start_state <= 0;
    else if (SCL_in) start_state <= 1;
    else start_state <= 0;
end

always @(posedge SDA_in or negedge reset_start_stop) begin
    if (!reset_start_stop) stop_state <= 0;
    else if (SCL_in) stop_state <= 1;
    else stop_state <= 0;
end

always @(*) begin
    case (CS)
        START_STOP_DECISION: begin
                SDA_out_reg = 1;
                ACK = 0;
                detect_ACK = 0;
                counter_next = 0;
            if (start_state && !self_address_generated) begin
                NS = OPERATING_DECISION;
                counter_next = 0;
                wr_enable = 0;
                rd_enable = 0;
            end else begin
                NS = START_STOP_DECISION;
            end
        end

        OPERATING_DECISION: begin
            control_signal_reg[ADDRESS_WIDTH - counter] = SDA_in;
            counter_next = counter + 1;
            if (counter == ADDRESS_WIDTH + 1) begin
                if (control_signal[ADDRESS_WIDTH:1] == ADDRESS) begin
                    SDA_out_reg = 0;
                end else begin
                    SDA_out_reg = 1;
                end
            end else if (counter == ADDRESS_WIDTH + 2) begin
                if (control_signal[ADDRESS_WIDTH:1] == ADDRESS) begin
                    if (!control_signal[0]) rd_enable = 1;
                    else wr_enable = 1;
                end else begin
                    wr_enable = 0;
                    rd_enable = 0;
                end
                SDA_out_reg = 1;
                counter_next = 0;
                NS = FINISH_DETECTION;
            end
        end
```

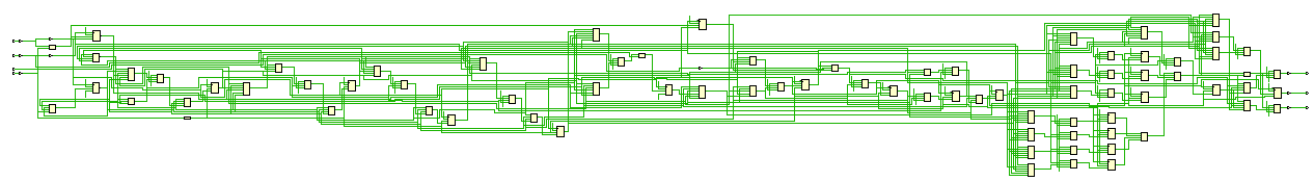
```
        FINISH_DETECTION: begin
            if ((counter == ADDRESS_WIDTH + 1 && !first_time) || stop_state) begin
                ACK = 1;
                if (NACK_detected || stop_state) begin
                    NS = START_STOP_DECISION;
                    wr_enable = 0;
                    rd_enable = 0;
                end
                counter_next = 0;
            end else begin
                if (counter == ADDRESS_WIDTH) detect_ACK = 1;
                else detect_ACK = 0;
                ACK = 0;
                counter_next = counter + 1;

            end
        end
    endcase
end
endmodule
```

2.3.2 RTL elaboration and synthesis schematic.

RTL elaboration



synthesis schematic



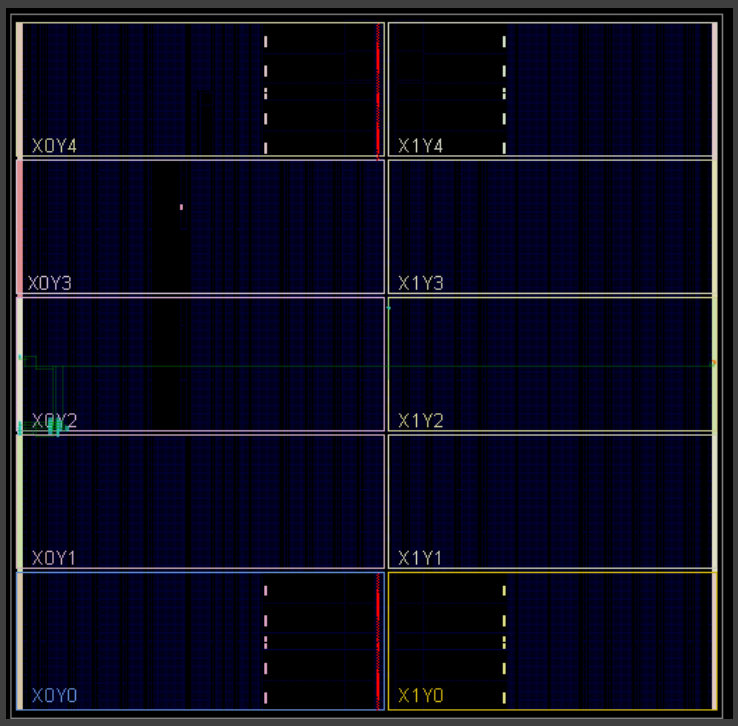
2.3.3 static time analysis reports and utilization.

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	1.644 ns	Worst Hold Slack (WHS):	0.321 ns	Worst Pulse Width Slack (WPWS):	2.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	6	Total Number of Endpoints:	6	Total Number of Endpoints:	22

All user specified timing constraints are met.

Name	Slice LUTs (134600)	Slice Registers (269200)	Bonded IOB (500)	BUFGCTRL (32)
N I2C_Pattern_Detector	37	41	7	2

2.3.4 implementation schematic



2.3.5 static time analysis reports and utilization.

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	1.630 ns	Worst Hold Slack (WHS):	0.186 ns	Worst Pulse Width Slack (WPWS):	2.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	6	Total Number of Endpoints:	6	Total Number of Endpoints:	22

All user specified timing constraints are met.

Name	Slice LUTs (133800)	Slice Registers (267600)	Slice (33450)	LUT as Logic (133800)	LUT Flip Flop Pairs (133800)	Bonded IOB (500)	BUFGCTRL (32)
N I2C_Pattern_Detector	37	41	24	37	13	7	2

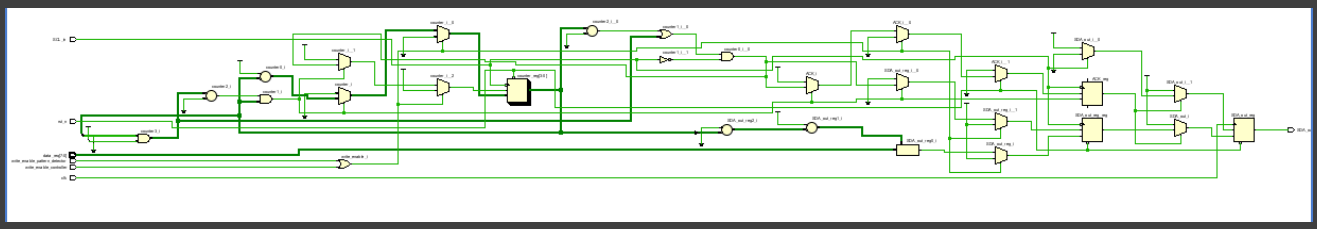
2.4 I2C_transmitter module

2.4.1 Verilog code

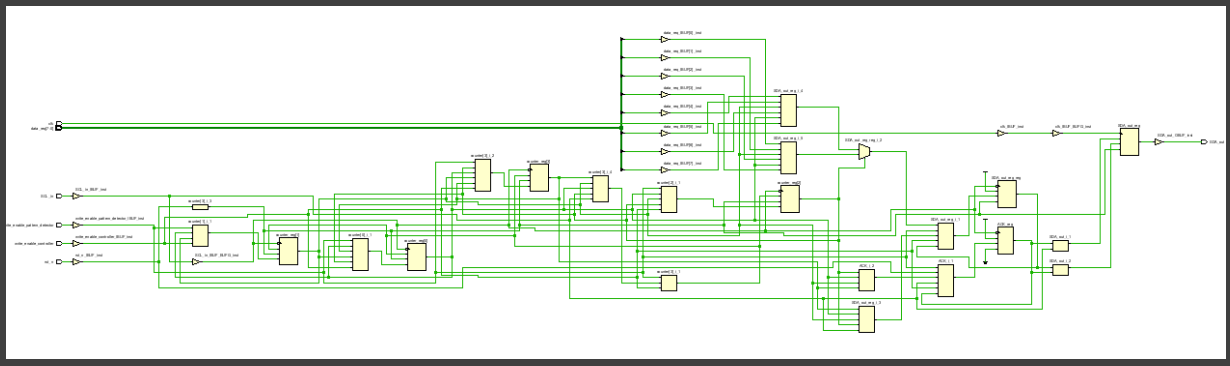
```
module I2C_Transmitter #(
    parameter DATA_WIDTH = 8
) (
    input logic clk,
    input logic rst_n,
    input logic [DATA_WIDTH-1:0] data_req,
    input logic write_enable_pattern_detector,
    input logic write_enable_controller,
    input logic SCL_in,
    input logic error,
    output logic SDA_out
);

localparam BIT_NO = $clog2(DATA_WIDTH);


logic write_enable;
logic SDA_out_reg;
logic error_check;
logic ACK;
logic [BIT_NO:0] counter;

assign write_enable = write_enable_pattern_detector || write_enable_controller;

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        SDA_out <= 1;
    end else begin
        if (ACK) SDA_out <= 1;
        else if (!SCL_in) SDA_out <= SDA_out_reg;
    end
end

always @(negedge SCL_in or negedge rst_n) begin
    if (!rst_n) begin
        counter <= 0;
        SDA_out_reg <= 1;
    end else begin
        if (write_enable) begin
            if ((!((counter) % 8)) && counter) begin
                ACK = 1;
                counter <= 0;
            end else if ((((counter) % 8) || !counter) && !SCL_in) begin
                SDA_out_reg <= data_req[DATA_WIDTH-counter-1];
                error_check <= 0;
                counter <= counter + 1;
                ACK = 0;
            end
        end else begin
            SDA_out_reg <= 1;
            counter <= 0;
        end
    end
end

endmodule
```

2.4.2 RTL elaboration and synthesis schematic.

RTL elaboration



synthesis schematic



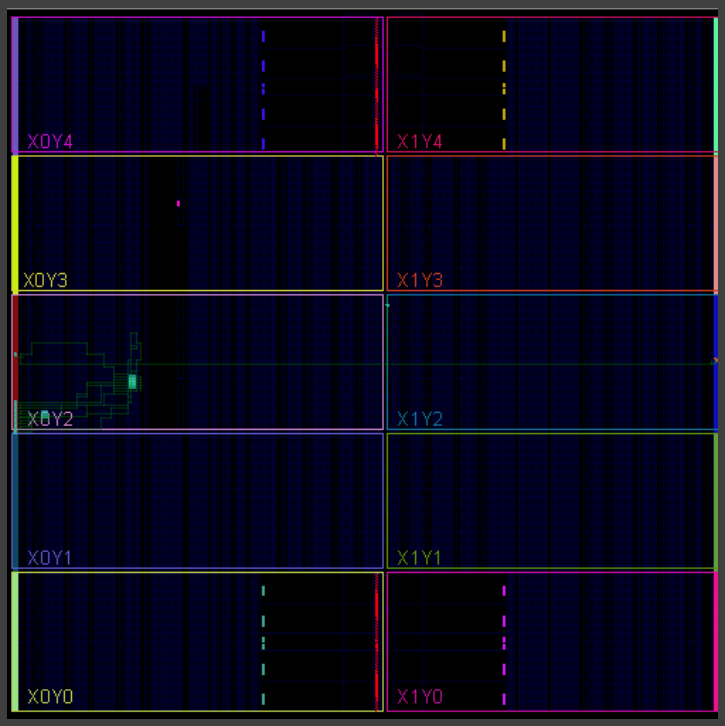
2.4.3 static time analysis reports and utilization.

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	1.249 ns	Worst Hold Slack (WHS):	0.245 ns	Worst Pulse Width Slack (WPWS):	2.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	14	Total Number of Endpoints:	14	Total Number of Endpoints:	15

All user specified timing constraints are met.

Name 1	Slice LUTs (133800)	Slice Registers (267600)	Slice (33450)	LUT as Logic (133800)	LUT Flip Flop Pairs (133800)	Bonded IOB (500)	BUFGCTRL (32)
N I2C_Receiver	18	22	8	18	13	16	2

2.4.4 implementation schematic



2.4.5 static time analysis reports and utilization.

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.437 ns	Worst Hold Slack (WHS):	0.058 ns	Worst Pulse Width Slack (WPWS):	2.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	12	Total Number of Endpoints:	12	Total Number of Endpoints:	9

All user specified timing constraints are met.

Name	Slice LUTs (133800)	Slice Registers (267600)	F7 Muxes (66900)	Slice (33450)	LUT as Logic (133800)	LUT Flip Flop Pairs (133800)	Bonded IOB (500)	BUFGCTRL (32)
N I2C_Transmitter	13	7	1	6	13	6	14	2

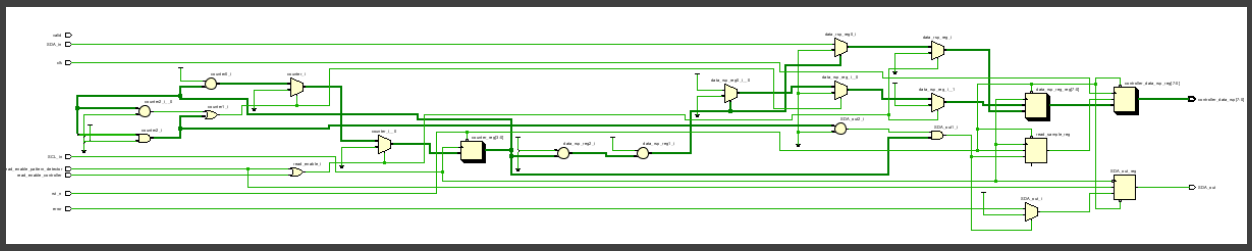
2.5 I2C_reciever module

2.5.1 Verilog code

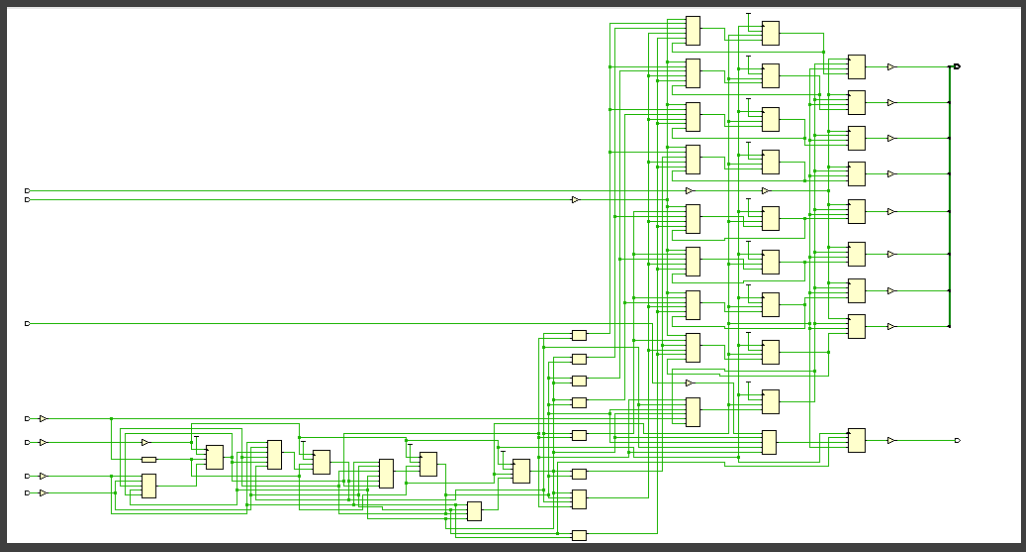
```
module I2C_Receiver #(
    parameter DATA_WIDTH = 8
) (
    input logic clk,
    input logic rst_n,
    input logic read_enable_pattern_detector,
    input logic read_enable_controller,
    input logic SCL_in,
    input logic valid,
    input logic SDA_in,
    input logic error,
    output logic [DATA_WIDTH-1:0] controller_data_rsp,
    output logic SDA_out
);

localparam BIT_NO = $clog2(DATA_WIDTH);

logic read_enable;
logic read_sample;
logic [DATA_WIDTH-1:0] data_rsp_reg;
logic [BIT_NO:0] counter;


assign read_enable = read_enable_pattern_detector || read_enable_controller;

always @(posedge SCL_in or negedge rst_n) begin
    if (!rst_n) begin
        counter <= 0;
        data_rsp_reg <= 0;
    end else begin
        if (read_enable) begin
                if (((counter) % 8) || !counter) begin
                    data_rsp_reg[DATA_WIDTH-counter-1] <= SDA_in;
                    counter <= counter + 1;
                end else begin
                    counter <= 0;
                end
            end else begin
            data_rsp_reg <= 0;
            counter <= 0;
        end
    end
end

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        controller_data_rsp <= 0;
    end else if (read_sample) begin
        controller_data_rsp <= data_rsp_reg;
    end
end

always @(negedge SCL_in or negedge rst_n) begin
    if (!rst_n) SDA_out <= 1;
    else begin
        if (read_enable_pattern_detector) begin
            if ((!((counter) % 8)) && (counter)) begin
                if (error) SDA_out <= 1;
                else SDA_out <= 0;
            end else SDA_out <= 1;
        end
    end
end

always @(posedge SCL_in or negedge rst_n) begin
    if (!rst_n) read_sample <= 0;
    else begin
        if ((!((counter) % 8)) && (counter)) read_sample = 1;
        else read_sample = 0;
    end
end
endmodule
```

2.5.2 RTL elaboration and synthesis schematic.

RTL elaboration



synthesis schematic



2.5.3 static time analysis reports and utilization.

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	1.597 ns	Worst Hold Slack (WHS):	0.139 ns	Worst Pulse Width Slack (WPWS):	2.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	14	Total Number of Endpoints:	14	Total Number of Endpoints:	15

All user specified timing constraints are met.

Name 1	Slice LUTs (134600)	Slice Registers (269200)	Bonded IOB (500)	BUFGCTRL (32)
N I2C_Receiver	18	22	16	2

2.6 I2C module

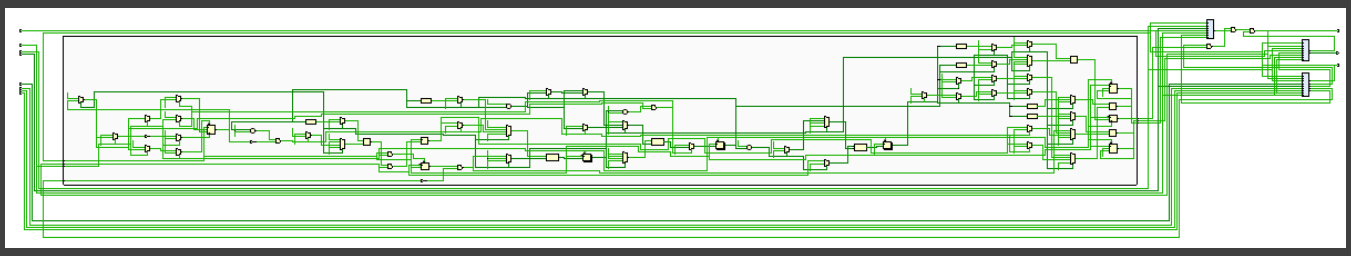
2.6.1 Verilog code

```
module I2C #(
    parameter DATA_WIDTH = 8,
    parameter ADDRESS = 7'h10,
    parameter ADDR_WIDTH = 7,
    parameter FORWARDED_CLOCK_SPEED = 100000,
    parameter LOCAL_CLOCK_SPEED = 1000000
) (
    input logic clk,
    input logic rst_n,
    input logic [DATA_WIDTH-1:0] controller_data_req,
    input logic [ADDR_WIDTH-1:0] controller_addr_req,
    input logic controller_valid_req,
    input logic controller_operation_req,
    input logic controller_restart_req,
    input logic error_signal,
    output logic [DATA_WIDTH-1:0] controller_data_rsp,
    inout wire SDA,
    inout wire SCL
);

logic write_enable_pattern_detector;
logic write_enable_controller;
logic read_enable_pattern_detector;
logic read_enable_controller;

logic SDA_out_controller;
logic SDA_out_detector;
logic SDA_out_transmitter;
logic SDA_out_receiver;
logic [DATA_WIDTH-1:0]data_rsp;

logic SCL_out_controller;

logic self_address_generated;

assign SDA = SDA_out_controller
& SDA_out_detector
& SDA_out_transmitter
& SDA_out_receiver;

assign SCL = SCL_out_controller;
```

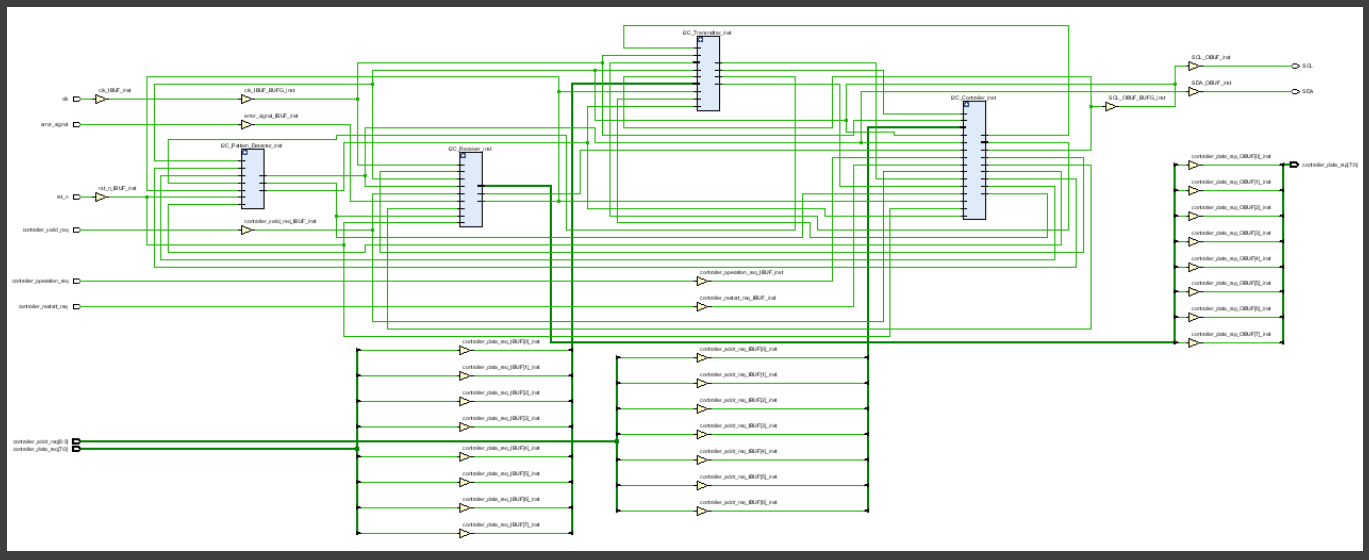
```
I2C_Pattern_Detector #(.ADDRESS(ADDRESS), .ADDRESS_WIDTH(ADDR_WIDTH)) I2C_Pattern_Detector_inst (
    .SCL_in(SCL),
    .SDA_in(SDA),
    .rst_n(rst_n),
    .self_address_generated(self_address_generated),
    .SDA_out(SDA_out_detector),
    .wr_enable(write_enable_pattern_detector),
    .rd_enable(read_enable_pattern_detector)
);

I2C_Controller #(.DATA_WIDTH(DATA_WIDTH), .ADDR_WIDTH(ADDR_WIDTH), .FORWARDED_CLOCK_SPEED(FORWARDED_CLOCK_SPEED), .LOCAL_CLOCK_SPEED(LOCAL_CLOCK_SPEED)) I2C_Controller_inst (
    .clk(clk),
    .rst_n(rst_n),
    .controller_data_req(controller_data_req),
    .controller_addr_req(controller_addr_req),
    .controller_valid_req(controller_valid_req),
    .controller_operation_req(controller_operation_req),
    .controller_restart_req(controller_restart_req),
    .controller_error_req(error_signal),
    .SCL_in(SCL),
    .SDA_in(SDA),
    .SCL_out(SCL_out_controller),
    .SDA_out(SDA_out_controller),
    .write_enable(write_enable_controller),
    .read_enable(read_enable_controller),
    .self_address_generated(self_address_generated)
);

I2C_Receiver #(.DATA_WIDTH(DATA_WIDTH)) I2C_Receiver_inst (
    .clk(clk),
    .rst_n(rst_n),
    .read_enable_pattern_detector(read_enable_pattern_detector),
    .read_enable_controller(read_enable_controller),
    .SCL_in(SCL),
    .valid(controller_valid_req),
    .SDA_in(SDA),
    .error(error_signal),
    .controller_data_rsp(controller_data_rsp),
    .SDA_out(SDA_out_receiver)
);

I2C_Transmitter #(.DATA_WIDTH(DATA_WIDTH)) I2C_Transmitter_inst (
    .clk(clk),
    .rst_n(rst_n),
    .data_req(controller_data_req),
    .write_enable_pattern_detector(write_enable_pattern_detector),
    .write_enable_controller(write_enable_controller),
    .SCL_in(SCL),
    .error(error_signal),
    .SDA_out(SDA_out_transmitter)
);


endmodule
```

2.6.2 RTL elaboration and synthesis schematic.

RTL elaboration



synthesis schematic



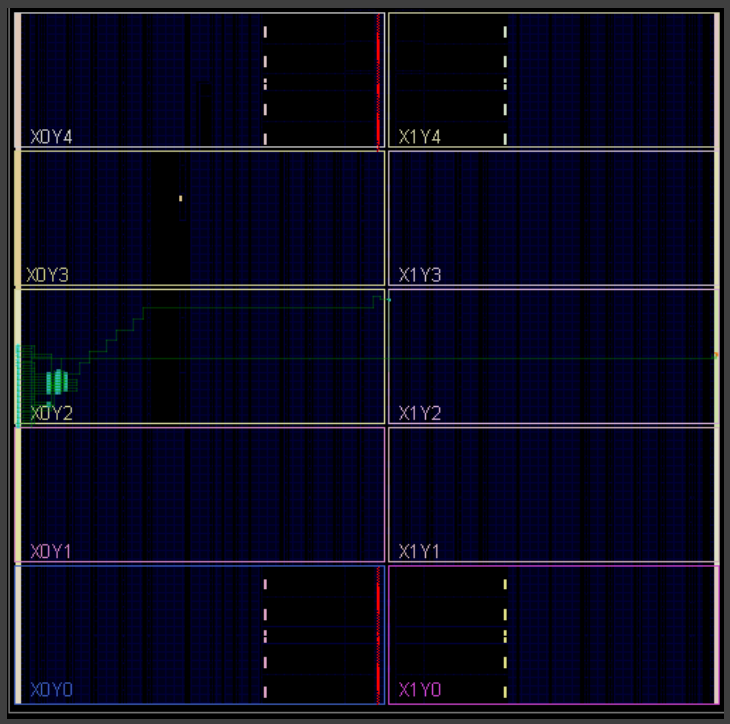
2.6.3 static time analysis reports and utilization.

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	4.596 ns	Worst Hold Slack (WHS):	0.127 ns	Worst Pulse Width Slack (WPWS):	2.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	6	Total Number of Endpoints:	6	Total Number of Endpoints:	19

All user specified timing constraints are met.

Name 1	Slice LUTs (134600)	Slice Registers (269200)	F7 Muxes (67300)	Bonded IOB (500)	BUFGCTRL (32)
N I2C	114	116	1	31	2
I2C_Controller_inst (I2...	50	46	0	0	0
I2C_Pattern_Detector_...	39	41	0	0	0
I2C_Receiver_inst (I2...	16	22	0	0	0
I2C_Transmitter_inst (I...	9	7	1	0	0

2.6.4 implementation schematic



2.6.5 static time analysis reports and utilization.

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	4.207 ns	Worst Hold Slack (WHS):	0.247 ns	Worst Pulse Width Slack (WPWS):	2.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	6	Total Number of Endpoints:	6	Total Number of Endpoints:	19

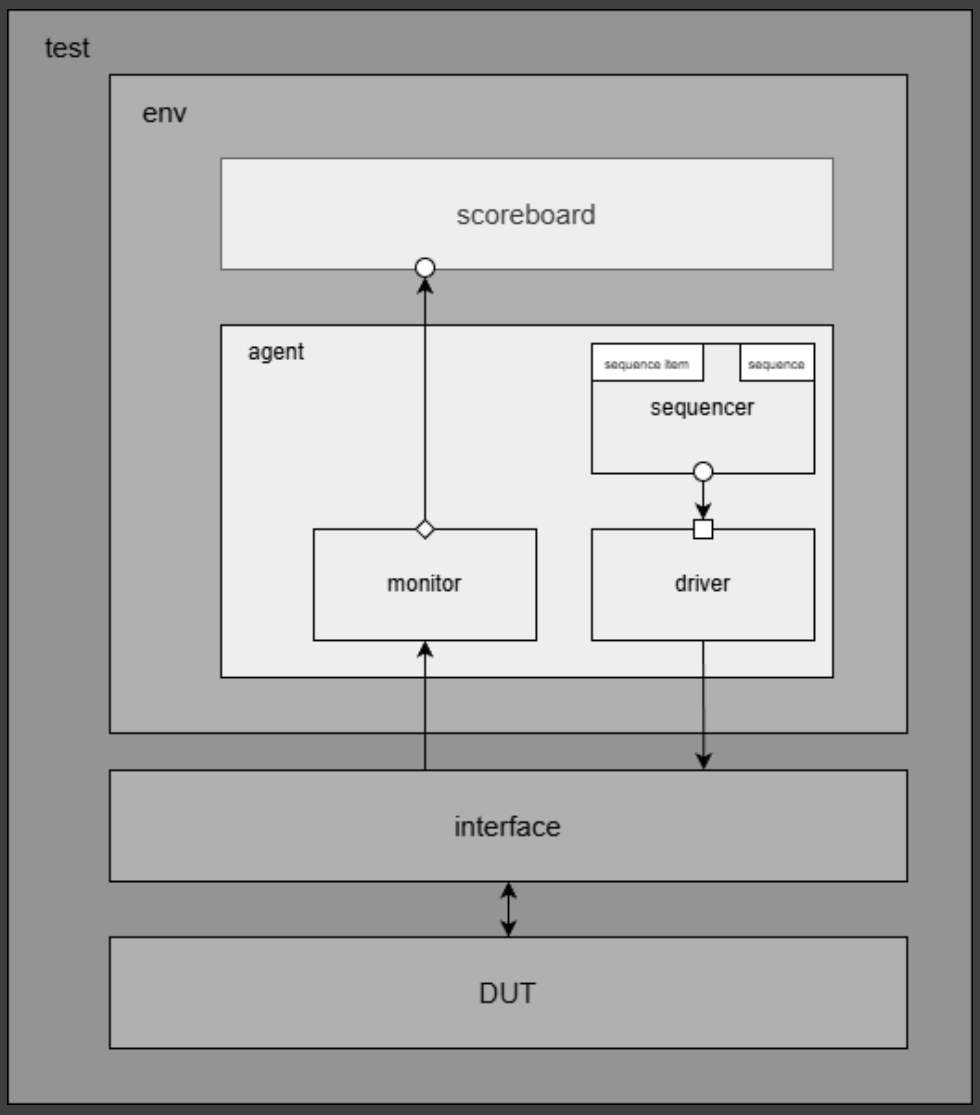
All user specified timing constraints are met.

Name	Slice LUTs (133800)	Slice Registers (267600)	F7 Muxes (66900)	Slice (33450)	LUT as Logic (133800)	LUT Flip Flop Pairs (133800)	Bonded IOB (500)	BUFGCTRL (32)
N I2C	114	116	1	55	114	48	31	2
I2C_Controller_inst (I2...	50	46	0	23	50	14	0	0
I2C_Pattern_Detector_...	39	41	0	24	39	15	0	0
I2C_Receiver_inst (I2...	16	22	0	6	16	13	0	0
I2C_Transmitter_inst (I...	9	7	1	6	9	5	0	0

3 I2C Verification

3.1 I2C_Controller module Block-Level UVM Environment

3.1.1 I2C_Controller Verification Plan and Environment



label	description	stimulus generated	checkers
randomization sequence	In this sequence we will randomize the control module to make sire that the overall functionality of the design	randomization for all the inputs with constraining the reset to be on 5% of the time and no restert request for 95% of the time	we check the functionality of the outputs using checkers comparing the output with the reference output from golden model
start read sequence	this is a directed seequence to check the starting of a read transaction	directed	we check the functionality of the outputs using checkers comparing the output with the reference output from golden model
start write sequence	this is a directed seequence to check the starting of a write transaction	directed	we check the functionality of the outputs using checkers comparing the output with the reference output from golden model
NACK feature	this is a directed sequence where the SDA_in will be held 1 to check the detection of NACK by the controller	directed	we check the functionality of the outputs using checkers comparing the output with the reference output from golden model
valid feaure	this is a directed sequence where we held the valid to low to check the ternination of the transaction based on the valid signal	directed	we check the functionality of the outputs using checkers comparing the output with the reference output from golden model
restart feature	this is a directed sequence where we held the restart request to 1 to check the restarting feature of the control module	directed	we check the functionality of the outputs using checkers comparing the output with the reference output from golden model

3.1.2 I2C_configuration_object.

```
package i2c_config_package;

    import uvm_pkg::*;
    `include "uvm_macros.svh";

    class i2c_config_class extends uvm_object;
        `uvm_object_utils(i2c_config_class)

        virtual I2C_interface vif;
        bit is_active = UVM_ACTIVE;
        bit is_top = 1;
        bit [6:0] i2c_address;

        function new(string name = "i2c_config_class");
            super.new(name);
        endfunction
    endclass
endpackage
```

3.1.3 I2C_Controller_sequence_item

```
////////////////////////////////////////////////////////////////////////////////
//Name: Abdelrahman Mohamed Ragab
// Module-Name: I2C_Controller_sequence_item
////////////////////////////////////////////////////////////////////////////////

package I2C_Controller_sequence_item_pkg;

    // Import UVM and related packages
    import uvm_pkg::*;
    `include "uvm_macros.svh";

    parameter ADDRESS = 7'h10;
    parameter ADDRESS_WIDTH = 7;
    parameter DATA_WIDTH = 8;
    parameter FORWARDED_CLOCK_SPEED = 100000;
    parameter LOCAL_CLOCK_SPEED = 1000000;

    // I2C_Controller TX sequence item class extending uvm_sequence_item
    class I2C_Controller_sequence_item_class extends uvm_sequence_item;

        // Factory registration
        `uvm_object_utils(I2C_Controller_sequence_item_class);

        //input signals
        logic SCL_in;
        rand logic SDA_in;
        rand logic rst_n;
        rand logic [DATA_WIDTH-1:0] controller_data_req;
        rand logic [ADDRESS_WIDTH-1:0] controller_addr_req;
        rand logic controller_valid_req;
        rand logic controller_operation_req;
        rand logic controller_restart_req;
        rand logic error_signal;

        //output signals
        logic write_enable;
        logic read_enable;
        logic SDA_out;
        logic SCL_out;
        logic self_address_generated;

        bit SCL_in_past;
        bit SDA_in_past;

        // Constructor
        function new(string name = "I2C_Controller_sequence_item_class");
            super.new(name);
        endfunction //new()

        // Main constraint: reset is mostly 0, write/read enables mostly 1
        constraint c {
            rst_n dist {0:/5 , 1:/95};
            controller_restart_req dist {1:/5 , 0:/95};
        }

    endclass //I2C_Controller_sequence_item_class extends uvm_sequence_item

endpackage //I2C_Controller_sequence_item_pkg
```

3.1.4 I2C_Controller_randomized_sequence

```
////////////////////////////////////////////////////////////////////////////////
//Name: Abdelrahman Mohamed Ragab
// Module-Name: I2C_Controller_randomized_sequence
////////////////////////////////////////////////////////////////////////////////

package I2C_Controller_randomized_sequence_pkg;

    // Import UVM and related packages
    import uvm_pkg::*;
    import I2C_Controller_sequence_item_pkg::*;
    `include "uvm_macros.svh";

    // I2C_Controller TX main sequence class extending uvm_sequence
    class I2C_Controller_randomized_sequence_class extends uvm_sequence #(I2C_Controller_sequence_item_class);

        // Factory registration
        `uvm_object_utils(I2C_Controller_randomized_sequence_class);

        // Sequence item declaration
        I2C_Controller_sequence_item_class MAIN_sequence_sequence_item;

        logic rst_n_value;
        logic random;
        logic not_valid;

        // Constructor
        function new(string name = "I2C_Controller_randomized_sequence_class");
            super.new(name);
        endfunction //new()

        // Body task
        task body();
        MAIN_sequence_sequence_item = I2C_Controller_sequence_item_class::type_id::create("MAIN_sequence_sequence_item");
        // Randomize the sequence item
        assert(MAIN_sequence_sequence_item.randomize());
        // Apply external constraints if needed
        if (random)
            MAIN_sequence_sequence_item.rst_n = rst_n_value;
        if (not_valid)
            MAIN_sequence_sequence_item.controller_valid_req = 0;
        start_item(MAIN_sequence_sequence_item);
        finish_item(MAIN_sequence_sequence_item);
    endtask

    endclass //I2C_Controller_randomized_sequence_class extends uvm_sequence

endpackage //I2C_Controller_randomized_sequence_pkg
```

3.1.5 I2C_Controller_directed_restart_sequence

```
////////////////////////////////////////////////////////////////////////////////
//Name: Abdelrahman Mohamed Ragab
// Module-Name: I2C_Controller_directed_restart_sequence
////////////////////////////////////////////////////////////////////////////////

package I2C_Controller_directed_restart_sequence_pkg;

    // Import UVM and related packages
    import uvm_pkg::*;
    import I2C_Controller_sequence_item_pkg::*;
    `include "uvm_macros.svh";

    // I2C_Controller TX main sequence class extending uvm_sequence
    class I2C_Controller_directed_restart_sequence_class extends uvm_sequence #(I2C_Controller_sequence_item_class);

        // Factory registration
        `uvm_object_utils(I2C_Controller_directed_restart_sequence_class);

        // Sequence item declaration
        I2C_Controller_sequence_item_class MAIN_sequence_item;

        bit top = 1;

        // Constructor
        function new(string name = "I2C_Controller_directed_restart_sequence_class");
            super.new(name);
        endfunction //new()

        // Body task
        task body;
            MAIN_sequence_item = I2C_Controller_sequence_item_class::type_id::create("MAIN_sequence_item");
               start_item(MAIN_sequence_item);
                    MAIN_sequence_item.rst_n=1;
                    MAIN_sequence_item.controller_data_req='h34;
                    MAIN_sequence_item.controller_addr_req='h10;
                    MAIN_sequence_item.controller_valid_req=1;
                    MAIN_sequence_item.controller_operation_req=0;
                    MAIN_sequence_item.controller_restart_req=1;
                    MAIN_sequence_item.error_signal=0;
                    if (!top) begin
                        MAIN_sequence_item.SDA_in=0;
                    end
                finish_item(MAIN_sequence_item);
        endtask //body()

    endclass //I2C_Controller_directed_restart_sequence_class extends uvm_sequence

endpackage //I2C_Controller_directed_restart_sequence_pkg
```

3.1.6 I2C_Controller_directed_start_transaction_read_sequence

```
////////////////////////////////////////////////////////////////////////////////
//Name: Abdelrahman Mohamed Ragab
// Module-Name: I2C_Controller_directed_start_transaction_read_sequence
////////////////////////////////////////////////////////////////////////////////

package I2C_Controller_directed_start_transaction_read_sequence_pkg;

    // Import UVM and related packages
    import uvm_pkg::*;
    import I2C_Controller_sequence_item_pkg::*;
    `include "uvm_macros.svh";

    // I2C_Controller TX main sequence class extending uvm_sequence
    class I2C_Controller_directed_start_transaction_read_sequence_class extends uvm_sequence #(I2C_Controller_sequence_item_class);

        // Factory registration
        `uvm_object_utils(I2C_Controller_directed_start_transaction_read_sequence_class);

        // Sequence item declaration
        I2C_Controller_sequence_item_class MAIN_sequence_item;

        // Constructor
        function new(string name = "I2C_Controller_directed_start_transaction_read_sequence_class");
            super.new(name);
        endfunction //new()

        bit [6:0] address = 'h10;
        bit top = 1;

        // Body task
        task body;
                MAIN_sequence_item = I2C_Controller_sequence_item_class::type_id::create("MAIN_sequence_item");

                if (top) begin
                start_item(MAIN_sequence_item);
                    MAIN_sequence_item.rst_n=1;
                    MAIN_sequence_item.controller_data_req='h34;
                    MAIN_sequence_item.controller_addr_req= address;
                    MAIN_sequence_item.controller_valid_req=1;
                    MAIN_sequence_item.controller_operation_req=1;
                    MAIN_sequence_item.controller_restart_req=0;
                    MAIN_sequence_item.error_signal=0;
                finish_item(MAIN_sequence_item);
                end else begin
                start_item(MAIN_sequence_item);
                    MAIN_sequence_item.rst_n=0;
                    MAIN_sequence_item.controller_data_req=0;
                    MAIN_sequence_item.controller_addr_req=0;
                    MAIN_sequence_item.controller_valid_req=1;
                    MAIN_sequence_item.controller_operation_req=0;
                    MAIN_sequence_item.controller_restart_req=0;
                    MAIN_sequence_item.SDA_in=0;
                finish_item(MAIN_sequence_item);

                start_item(MAIN_sequence_item);
                    MAIN_sequence_item.rst_n=1;
                    MAIN_sequence_item.controller_data_req='h34;
                    MAIN_sequence_item.controller_addr_req= address;
                    MAIN_sequence_item.controller_valid_req=1;
                    MAIN_sequence_item.controller_operation_req=1;
                    MAIN_sequence_item.controller_restart_req=0;
                    MAIN_sequence_item.SDA_in=0;
                finish_item(MAIN_sequence_item);
                end

        endtask //body()

    endclass //I2C_Controller_directed_start_transaction_read_sequence_class extends uvm_sequence

endpackage //I2C_Controller_directed_start_transaction_read_sequence_pkg
```

3.1.7 I2C_Controller_directed_start_transaction_write_sequence

```
////////////////////////////////////////////////////////////////////////////////
//Name: Abdelrahman Mohamed Ragab
// Module-Name: I2C_Controller_directed_start_transaction_write_sequence
////////////////////////////////////////////////////////////////////////////////

package I2C_Controller_directed_start_transaction_write_sequence_pkg;

    // Import UVM and related packages
    import uvm_pkg::*;
    import I2C_Controller_sequence_item_pkg::*;
    `include "uvm_macros.svh";

    // I2C_Controller TX main sequence class extending uvm_sequence
    class I2C_Controller_directed_start_transaction_write_sequence_class extends uvm_sequence #(I2C_Controller_sequence_item_class);

        // Factory registration
        `uvm_object_utils(I2C_Controller_directed_start_transaction_write_sequence_class);

        // Sequence item declaration
        I2C_Controller_sequence_item_class MAIN_sequence_item;

        // Constructor
        function new(string name = "I2C_Controller_directed_start_transaction_write_sequence_class");
            super.new(name);
        endfunction //new()

        bit [6:0] address = 'h10;
        bit top = 1;

        // Body task
        task body;
                MAIN_sequence_item = I2C_Controller_sequence_item_class::type_id::create("MAIN_sequence_item");

                if (top) begin
                    start_item(MAIN_sequence_item);
                    MAIN_sequence_item.rst_n=1;
                    MAIN_sequence_item.controller_data_req='h34;
                    MAIN_sequence_item.controller_addr_req= address;
                    MAIN_sequence_item.controller_valid_req=1;
                    MAIN_sequence_item.controller_operation_req=0;
                    MAIN_sequence_item.controller_restart_req=0;
                    MAIN_sequence_item.error_signal=0;
                finish_item(MAIN_sequence_item);
                end else begin
                start_item(MAIN_sequence_item);
                    MAIN_sequence_item.rst_n=0;
                    MAIN_sequence_item.controller_data_req=0;
                    MAIN_sequence_item.controller_addr_req=0;
                    MAIN_sequence_item.controller_valid_req=1;
                    MAIN_sequence_item.controller_operation_req=0;
                    MAIN_sequence_item.controller_restart_req=0;
                    MAIN_sequence_item.SDA_in=0;
                finish_item(MAIN_sequence_item);

                start_item(MAIN_sequence_item);
                    MAIN_sequence_item.rst_n=1;
                    MAIN_sequence_item.controller_data_req='h34;
                    MAIN_sequence_item.controller_addr_req= address;
                    MAIN_sequence_item.controller_valid_req=1;
                    MAIN_sequence_item.controller_operation_req=0;
                    MAIN_sequence_item.controller_restart_req=0;
                    MAIN_sequence_item.SDA_in=0;
                finish_item(MAIN_sequence_item);
                end

        endtask //body()

    endclass //I2C_Controller_directed_start_transaction_write_sequence_class extends uvm_sequence

endpackage //I2C_Controller_directed_start_transaction_write_sequence_pkg
```

3.1.8 I2C_Controller_directed_stop_SDA_sequence

```
////////////////////////////////////////////////////////////////////////////////
//Name: Abdelrahman Mohamed Ragab
// Module-Name: I2C_Controller_directed_stop_SDA_sequence
////////////////////////////////////////////////////////////////////////////////

package I2C_Controller_directed_stop_SDA_sequence_pkg;

    // Import UVM and related packages
    import uvm_pkg::*;
    import I2C_Controller_sequence_item_pkg::*;
    `include "uvm_macros.svh";

    // I2C_Controller TX main sequence class extending uvm_sequence
    class I2C_Controller_directed_stop_SDA_sequence_class extends uvm_sequence #(I2C_Controller_sequence_item_class);

        // Factory registration
        `uvm_object_utils(I2C_Controller_directed_stop_SDA_sequence_class);

        // Sequence item declaration
        I2C_Controller_sequence_item_class MAIN_sequence_item;

        bit top = 1;

        // Constructor
        function new(string name = "I2C_Controller_directed_stop_SDA_sequence_class");
            super.new(name);
        endfunction //new()

        // Body task
        task body;
            MAIN_sequence_item = I2C_Controller_sequence_item_class::type_id::create("MAIN_sequence_item");
               start_item(MAIN_sequence_item);
                    MAIN_sequence_item.rst_n=1;
                    MAIN_sequence_item.controller_data_req='h34;
                    MAIN_sequence_item.controller_addr_req='h10;
                    MAIN_sequence_item.controller_valid_req=1;
                    MAIN_sequence_item.controller_operation_req=0;
                    MAIN_sequence_item.controller_restart_req=0;
                    MAIN_sequence_item.error_signal=1;
                    if (!top) begin
                        MAIN_sequence_item.SDA_in=1;
                    end
                finish_item(MAIN_sequence_item);
        endtask //body()

    endclass //I2C_Controller_directed_stop_SDA_sequence_class extends uvm_sequence

endpackage //I2C_Controller_directed_stop_SDA_sequence_pkg
```

3.1.9 I2C_Controller_directed_stop_valid_sequence

```
////////////////////////////////////////////////////////////////////////////////
//Name: Abdelrahman Mohamed Ragab
// Module-Name: I2C_Controller_directed_stop_valid_sequence
////////////////////////////////////////////////////////////////////////////////

package I2C_Controller_directed_stop_valid_sequence_pkg;

    // Import UVM and related packages
    import uvm_pkg::*;
    import I2C_Controller_sequence_item_pkg::*;
    `include "uvm_macros.svh";

    // I2C_Controller TX main sequence class extending uvm_sequence
    class I2C_Controller_directed_stop_valid_sequence_class extends uvm_sequence #(I2C_Controller_sequence_item_class);

        // Factory registration
        `uvm_object_utils(I2C_Controller_directed_stop_valid_sequence_class);

        // Sequence item declaration
        I2C_Controller_sequence_item_class MAIN_sequence_item;

        bit top = 1;

        // Constructor
        function new(string name = "I2C_Controller_directed_stop_valid_sequence_class");
            super.new(name);
        endfunction //new()

        // Body task
        task body;
            MAIN_sequence_item = I2C_Controller_sequence_item_class::type_id::create("MAIN_sequence_item");
               start_item(MAIN_sequence_item);
                    MAIN_sequence_item.rst_n=1;
                    MAIN_sequence_item.controller_data_req='h34;
                    MAIN_sequence_item.controller_addr_req='h10;
                    MAIN_sequence_item.controller_valid_req=0;
                    MAIN_sequence_item.controller_operation_req=0;
                    MAIN_sequence_item.controller_restart_req=0;
                    MAIN_sequence_item.error_signal=0;
                    if (!top) begin
                        MAIN_sequence_item.SDA_in=0;
                    end
                finish_item(MAIN_sequence_item);
        endtask //body()

    endclass //I2C_Controller_directed_stop_valid_sequence_class extends uvm_sequence

endpackage //I2C_Controller_directed_stop_valid_sequence_pkg
```

3.1.10 I2C_Controller_sequencer

```
////////////////////////////////////////////////////////////////////////////////
//Name: Abdelrahman Mohamed Ragab
// Module-Name: I2C_Controller_sequencer
////////////////////////////////////////////////////////////////////////////////

package I2C_Controller_sequencer_pkg;

    // Import UVM and related packages
    import uvm_pkg::*;
    import I2C_Controller_sequence_item_pkg::*;
    `include "uvm_macros.svh";

    // I2C_Controller TX sequencer class extending uvm_sequencer
    class I2C_Controller_sequencer_class extends uvm_sequencer #(I2C_Controller_sequence_item_class);

        // Factory registration
        `uvm_component_utils(I2C_Controller_sequencer_class);

        // Constructor
        function new(string name = "I2C_Controller_sequencer_class" , uvm_component parent = null);
            super.new(name,parent);
        endfunction //new()

    endclass //I2C_Controller_sequencer_class

endpackage //I2C_Controller_sequencer_pkg
```

3.1.11 I2C_Controller_virtual_sequence

```
package I2C_Controller_virtual_sequence_pkg;

// Import UVM and related packages
    import uvm_pkg::*;
    import I2C_Controller_sequence_item_pkg::*;
    import I2C_Controller_directed_restart_sequence_pkg::*;
    import I2C_Controller_directed_start_transaction_read_sequence_pkg::*;
    import I2C_Controller_directed_start_transaction_write_sequence_pkg::*;
    import I2C_Controller_directed_stop_SDA_sequence_pkg::*;
    import I2C_Controller_directed_stop_valid_sequence_pkg::*;
    import I2C_Controller_sequencer_pkg::*;
    `include "uvm_macros.svh";

    // I2C Pattern Detector virtual sequence class extending uvm_sequence
    class I2C_Controller_virtual_sequence_class extends uvm_sequence #(I2C_Controller_sequence_item_class);

        // Factory registration
        `uvm_object_utils(I2C_Controller_virtual_sequence_class);

        virtual I2C_interface virtual_interface;

        // Sequence item declaration
        I2C_Controller_sequence_item_class virtual_sequence_item;
        I2C_Controller_directed_restart_sequence_class directed_restart_sequence;
        I2C_Controller_directed_start_transaction_read_sequence_class directed_start_read_sequence;
        I2C_Controller_directed_start_transaction_write_sequence_class directed_start_write_sequence;
        I2C_Controller_directed_stop_SDA_sequence_class directed_stop_SDA_sequence;
        I2C_Controller_directed_stop_valid_sequence_class directed_stop_valid_sequence;
        I2C_Controller_sequencer_class vitual_sequence_sequencer;

        logic break_loop;

        // Constructor
        function new(string name = "I2C_Controller_virtual_sequence_class");
            super.new(name);
        endfunction //new()

        task body();
            `uvm_info(get_type_name(), "virtual_seq: Inside Body", UVM_LOW);
            directed_restart_sequence = I2C_Controller_directed_restart_sequence_class::type_id::create("directed_restart_sequence");
            directed_start_read_sequence = I2C_Controller_directed_start_transaction_read_sequence_class::type_id::create("directed_start_read_sequence");
            directed_start_write_sequence = I2C_Controller_directed_start_transaction_write_sequence_class::type_id::create("directed_start_write_sequence");
            directed_stop_SDA_sequence = I2C_Controller_directed_stop_SDA_sequence_class::type_id::create("directed_stop_SDA_sequence");
            directed_stop_valid_sequence = I2C_Controller_directed_stop_valid_sequence_class::type_id::create("directed_stop_valid_sequence");

            uvm_config_db#(virtual I2C_interface)::get(m_sequencer, "", "int", virtual_interface);


        directed_restart_sequence.top = 0;
        directed_start_read_sequence.top = 0;
        directed_start_write_sequence.top = 0;
        directed_stop_SDA_sequence.top = 0;
        directed_stop_valid_sequence.top = 0;

            directed_start_write_sequence.start(vitual_sequence_sequencer);
            `uvm_info("run_phase","finish first directed test",UVM_MEDIUM);
            repeat (30) @(posedge virtual_interface.SCL_in);
            repeat (8) begin
                directed_stop_valid_sequence.start(vitual_sequence_sequencer);
                fork
                    begin
                        @(posedge virtual_interface.SCL_in);
                    end

                    begin
                        repeat (100) @(posedge virtual_interface.clk);
                        break_loop = 1;
                    end
                join_any
                if (break_loop) break;
            end
```

```
disable fork;
break_loop = 0;
`uvm_info("run_phase","finish second directed test",UVM_MEDIUM);
directed_start_write_sequence.start(vitual_sequence_sequencer);
`uvm_info("run_phase","finish forth directed test",UVM_MEDIUM);
repeat (15) @(posedge virtual_interface.SCL_in);
repeat (8) begin
    directed_stop_SDA_sequence.start(vitual_sequence_sequencer);
    fork
        begin
            @(posedge virtual_interface.SCL_in);
        end

        begin
            repeat (100) @(posedge virtual_interface.clk);
            break_loop = 1;
        end
    join_any
    if (break_loop) break;
end
disable fork;
break_loop = 0;
`uvm_info("run_phase","finish fifth directed test",UVM_MEDIUM);
directed_start_read_sequence.start(vitual_sequence_sequencer);
`uvm_info("run_phase","finish sixth directed test",UVM_MEDIUM);
repeat (15) @(posedge virtual_interface.SCL_in);
repeat (8) begin
    directed_restart_sequence.start(vitual_sequence_sequencer);
    fork
        begin
            @(posedge virtual_interface.SCL_in);
        end

        begin
            repeat (100) @(posedge virtual_interface.clk);
            break_loop = 1;
        end
    join_any
    if (break_loop) break;
end
disable fork;
break_loop = 0;
`uvm_info("run_phase","finish seventh directed test",UVM_MEDIUM);
directed_start_read_sequence.start(vitual_sequence_sequencer);
`uvm_info("run_phase","finish eighth directed test",UVM_MEDIUM);
repeat (15) begin
    directed_stop_valid_sequence.start(vitual_sequence_sequencer);
    fork
        begin
            @(posedge virtual_interface.SCL_in);
        end

        begin
            repeat (100) @(posedge virtual_interface.clk);
            break_loop = 1;
        end
    join_any
    if (break_loop) break;
end
```

```
            disable fork;
            break_loop = 0;
            `uvm_info("run_phase","finish ninth directed test",UVM_MEDIUM);
            directed_start_read_sequence.start(vitual_sequence_sequencer);
            `uvm_info("run_phase","finish tenth directed test",UVM_MEDIUM);
            repeat (8) @(posedge virtual_interface.SCL_in);
            repeat (8) begin
                directed_stop_SDA_sequence.start(vitual_sequence_sequencer);
                fork
                    begin
                        @(posedge virtual_interface.SCL_in);
                    end

                    begin
                        repeat (100) @(posedge virtual_interface.clk);
                        break_loop = 1;
                    end
                join_any
                if (break_loop) break;
            end
            disable fork;
            break_loop = 0;
            `uvm_info("run_phase","finish eleventh directed test",UVM_MEDIUM);
        endtask
    endclass //I2C_controller_virtual_sequence_class extends uvm_seq

endpackage
```

3.1.12 I2C_Controller_scoreboard

```
////////////////////////////////////////////////////////////////////////////////
//Name: Abdelrahman Mohamed Ragab
// Module-Name: I2C_Controller_scoreboard_pkg
////////////////////////////////////////////////////////////////////////////////

package I2C_Controller_scoreboard_pkg;

    // Import UVM and related packages
    import uvm_pkg::*;
    import I2C_Controller_sequence_item_pkg::*;
    import i2c_config_package::*;
    `include "uvm_macros.svh";

    // I2C_Controller full scoreboard class extending uvm_scoreboard
    class I2C_Controller_scoreboard_class extends uvm_scoreboard;

        // Factory registration
        `uvm_component_utils(I2C_Controller_scoreboard_class);

        // Sequence item and analysis port declarations
        I2C_Controller_sequence_item_class scoreboard_sequence_item;
        uvm_analysis_export #(I2C_Controller_sequence_item_class) scoreboard_analysis_export;
        uvm_tlm_analysis_fifo #(I2C_Controller_sequence_item_class) scoreboard_analysis_fifo;

        i2c_config_class scoreboard_confg_db;

        parameter ADDRESS_WIDTH = 7;
        parameter ADDRESS = 7'h10;
        parameter DATA_WIDTH = 8;
        parameter FORWARDED_CLOCK_SPEED = 100000;
        parameter LOCAL_CLOCK_SPEED = 1000000;

        localparam DIVIDE_BY = LOCAL_CLOCK_SPEED / (2*FORWARDED_CLOCK_SPEED);
        localparam DIVIDE_BY_WIDTH = $clog2(DIVIDE_BY);

        int correct_case = 0;
        int wrong_case = 0;

        //ref signals
        logic write_enable_ref;
        logic read_enable_ref;
        logic self_address_generated_ref;
        logic SCL_out_ref;
        logic SDA_out_ref;

        logic [2:0] CS;
        logic operation;
        logic ACK_done;
        logic after_start;
        logic pos_edge;
        logic neg_edge;
        logic stop_req;
        logic restart_req;
        logic old_stop_req;
        logic old_restart_req;
        logic SCL_check;
        logic SCL_old;
        logic delay;
        logic delay_done;
        logic [4:0] counter;
        logic [DIVIDE_BY_WIDTH-1 : 0] clock_counter;
        logic [ADDRESS_WIDTH-1:0] address;
```

```
// Constructor
function new(string name = "I2C_Controller_scoreboard_class" , uvm_component parent = null);
    super.new(name,parent);
endfunction //new()

// Build phase
function void build_phase (uvm_phase phase);
    super.build_phase(phase);
    uvm_config_db#(i2c_config_class)::get(this, "", "cfg", scoreboard_confg_db);
    scoreboard_analysis_export = new("scoreboard_analysis_export" , this);
    scoreboard_analysis_fifo = new("scoreboard_analysis_fifo" , this);
endfunction //build_phase()

// Connect phase
function void connect_phase (uvm_phase phase);
    super.connect_phase(phase);
    scoreboard_analysis_export.connect(scoreboard_analysis_fifo.analysis_export);
endfunction //connect_phase()

// Run phase
task run_phase (uvm_phase phase);
    super.run_phase(phase);
    forever begin
        `uvm_info("SB","==> Entered scoreboard run_phase", UVM_LOW);
        scoreboard_analysis_fifo.get(scoreboard_sequence_item);

        golden_reference(scoreboard_sequence_item);
        SCL_old = SCL_out_ref;
        if ((SCL_check && pos_edge) || !SCL_check) begin
            if ((scoreboard_sequence_item.write_enable === write_enable_ref)
            && (scoreboard_sequence_item.read_enable === read_enable_ref)
            && ((scoreboard_sequence_item.SDA_out === SDA_out_ref) || (scoreboard_confg_db.is_top))
            && (scoreboard_sequence_item.self_address_generated === self_address_generated_ref)
            && (scoreboard_sequence_item.SCL_out === SCL_out_ref) || (scoreboard_confg_db.is_top)) begin
                correct_case++;
                `uvm_info("run_phase",$sformatf("correct case = %0d , wrong case = %0d , time = %0d",correct_case,wrong_case,$time),UVM_MEDIUM);
            end else begin
                wrong_case++;
                `uvm_info("run_phase",$sformatf("correct case = %0d , wrong case = %0d , time = %0d",correct_case,wrong_case,$time),UVM_MEDIUM);
                display_signal_mismatches(scoreboard_sequence_item);
            end
        end
    end
endtask //run_phase()
```

```
task golden_reference(I2C_Controller_sequence_item_class scoreboard_sequence_item);

    if (!scoreboard_sequence_item.rst_n) begin
        CS = 3'b000;
        counter = 0;
        clock_counter = 0;
        pos_edge = 1;
        SCL_check = 0;
        write_enable_ref = 0;
        read_enable_ref = 0;
        SDA_out_ref = 1;
        SCL_out_ref = 1;
        pos_edge = 0;
        neg_edge = 0;
        ACK_done = 0;
        restart_req = 0;
        stop_req = 0;
        delay = 0;
        delay_done = 0;
        old_stop_req = 0;
        old_restart_req = 0;
        self_address_generated_ref = 0;
    end else begin
        if ((CS == 3'b010 || CS == 3'b011)) begin
            if (clock_counter == DIVIDE_BY-1) begin
                SCL_out_ref = ~SCL_out_ref;
                clock_counter = 0;
            end else clock_counter++;

            if (!SCL_old && SCL_out_ref) pos_edge = 1;
            else pos_edge = 0;

            if (SCL_old && !SCL_out_ref) neg_edge = 1;
            else neg_edge = 0;

        end

        if (pos_edge && ACK_done && !delay_done) begin
            if ((scoreboard_sequence_item.SDA_in || scoreboard_sequence_item.controller_restart_req  || !scoreboard_sequence_item.controller_valid_req) && !delay) begin
                delay = 1;
            end else begin
                delay = 0;
            end
        end

        if (pos_edge && ACK_done) begin
            if (scoreboard_sequence_item.SDA_in || !scoreboard_sequence_item.controller_valid_req) begin
                stop_req = 1;
                restart_req = 0;
            end else if (scoreboard_sequence_item.controller_restart_req) begin
                restart_req = 1;
                stop_req = 0;
            end else begin
                stop_req = 0;
                restart_req = 0;
            end
        end

        if (delay_done) begin
            SCL_check = 0;
        end

            case (CS)
                3'b000: begin
                if (scoreboard_sequence_item.controller_valid_req) begin
                    CS = 3'b001;
                end
                end
```

```
    3'b010: begin
        after_start = 0;
        if (counter == 9) begin
            CS = 3'b011;
            counter = 0;
        end
    end

    3'b011: begin
        if (counter < 8 && !SCL_out_ref && !delay) begin
            if (stop_req || old_stop_req) begin
                CS = 3'b100;
            end else if (restart_req || old_restart_req) begin
                CS = 3'b000;
            end
        end
    end

    3'b100: begin
        CS = 3'b101;
    end

    3'b101: begin
        CS = 3'b000;
    end
    endcase

    if (after_start) begin
        SCL_out_ref = 0;
        neg_edge = 1;
    end

if (((SCL_check && neg_edge) || !SCL_check)) begin
    case (CS)

        3'b000: begin
            counter = 0;
            clock_counter = 0;
            pos_edge = 1;
            SCL_check = 0;
            write_enable_ref = 0;
            read_enable_ref = 0;
            SDA_out_ref = 1;
            SCL_out_ref = 1;
            pos_edge = 0;
            neg_edge = 0;
            delay_done = 0;
            old_stop_req = 0;
            old_restart_req = 0;
            restart_req = 0;
            stop_req = 0;
            self_address_generated_ref = 0;
        end

        3'b001: begin
            SCL_out_ref = 1;
            SDA_out_ref = 0;
            write_enable_ref = 0;
            read_enable_ref = 0;
            self_address_generated_ref = 1;
            SCL_check = 1;
            address = scoreboard_sequence_item.controller_addr_req;
            operation = scoreboard_sequence_item.controller_operation_req;
        end
```

```
                3'b010: begin
                    if (counter < 7) begin
                        SDA_out_ref = address[6-counter];
                        counter++;
                    end else if (counter == 7) begin
                        SDA_out_ref = operation;
                        counter++;
                    end else if (counter == 8) begin
                        SDA_out_ref = 1;
                        counter++;
                        ACK_done = 1;
                    end
                end

                3'b011: begin
                    if (counter < 8 && !SCL_out_ref) begin
                        if (delay) begin
                            delay = 0;
                            delay_done = 1;
                            old_stop_req = stop_req;
                            old_restart_req = restart_req;
                        end else begin
                            if (!operation) write_enable_ref = 1;
                            else read_enable_ref = 1;
                            ACK_done = 0;
                            counter ++;
                        end
                    end else if (counter == 8) begin
                        SDA_out_ref = 1;
                        ACK_done = 1;
                        counter = 0;
                    end
                end

                3'b100: begin
                    delay_done = 0;
                    SDA_out_ref = 0;
                    SCL_out_ref = 1;
                    read_enable_ref = 0;
                    write_enable_ref = 0;
                    ACK_done = 0;
                    counter = 0;
                    SCL_check = 0;
                    old_stop_req = 0;
                    old_restart_req = 0;
                end

                3'b101: begin
                    SDA_out_ref = 1;
                    SCL_out_ref = 1;
                end
            endcase
        end
    end
endtask
```

```
    // Report phase
    function void report_phase (uvm_phase phase);
        super.report_phase(phase);
        `uvm_info("report_phase",$sformatf("correct case = %0d , wrong case = %0d",correct_case,wrong_case),UVM_MEDIUM);
    endfunction

endclass //I2C_Controller_scoreboard_class

endpackage //I2C_Controller_scoreboard_pkg
```

3.1.13 I2C_Controller_monitor

```
////////////////////////////////////////////////////////////////////////////////
//Name: Abdelrahman Mohamed Ragab
// Module-Name: I2C_Controller_monitor
////////////////////////////////////////////////////////////////////////////////

package I2C_Controller_monitor_pkg;

    // Import UVM and related packages
    import uvm_pkg::*;
    import I2C_Controller_sequence_item_pkg::*;
    `include "uvm_macros.svh";

    // I2C_Controller TX monitor class extending uvm_monitor
    class I2C_Controller_monitor_class extends uvm_monitor;

        // Factory registration
        `uvm_component_utils(I2C_Controller_monitor_class);

        // Virtual interface for monitoring DUT signals
        virtual I2C_interface monitor_interface;
        // Sequence item to capture monitored values
        I2C_Controller_sequence_item_class monitor_sequence_item;

        // Analysis port to send sequence items to subscribers
        uvm_analysis_port #(I2C_Controller_sequence_item_class) monitor_analysis_port;

        // Constructor
        function new(string name = "I2C_Controller_monitor_class" , uvm_component parent = null);
            super.new(name,parent);
        endfunction //new()

        // Build phase: create analysis port
        function void build_phase (uvm_phase phase);
            super.build_phase(phase);
            monitor_analysis_port = new("monitor_analysis_port" , this);
        endfunction

        logic start_state;
        logic stop_state;
        logic start_checking;
        logic [ADDRESS_WIDTH:0] control_signal [$];

        // Run phase: sample interface signals and send sequence items
        task run_phase (uvm_phase phase);
            super.run_phase(phase);

                forever begin
                    monitor_sequence_item = I2C_Controller_sequence_item_class::type_id::create("monitor_sequence_item");
                    @(negedge monitor_interface.clk);
                    monitor_sequence_item.write_enable = monitor_interface.write_enable;
                    monitor_sequence_item.read_enable = monitor_interface.read_enable;
                    monitor_sequence_item.SDA_out = monitor_interface.SDA_out;
                    monitor_sequence_item.SCL_out = monitor_interface.SCL_out;
                    monitor_sequence_item.SCL_in = monitor_interface.SCL_in;
                    monitor_sequence_item.SDA_in = monitor_interface.SDA_in;
                    monitor_sequence_item.rst_n = monitor_interface.rst_n;
                    monitor_sequence_item.controller_data_req = monitor_interface.controller_data_req;
                    monitor_sequence_item.controller_addr_req = monitor_interface.controller_addr_req;
                    monitor_sequence_item.controller_valid_req = monitor_interface.controller_valid_req;
                    monitor_sequence_item.controller_restart_req = monitor_interface.controller_restart_req;
                    monitor_sequence_item.controller_operation_req = monitor_interface.controller_operation_req;
                    monitor_sequence_item.self_address_generated = monitor_interface.self_address_generated;
                    monitor_analysis_port.write(monitor_sequence_item);
                end

        endtask //run_phase

    endclass //I2C_Controller_monitor_class extends uvm_monitor

endpackage //I2C_Controller_monitor_pkg
```

3.1.14 I2C_Controller_driver

```
////////////////////////////////////////////////////////////////////////////////
//Name: Abdelrahman Mohamed Ragab
// Module-Name: I2C_Controller_driver
////////////////////////////////////////////////////////////////////////////////

package I2C_Controller_driver_pkg;

    // Import UVM and related packages
    import uvm_pkg::*;
    import I2C_Controller_sequence_item_pkg::*;
    import i2c_config_package::*;
    `include "uvm_macros.svh";

    // I2C_Controller TX driver class extending uvm_driver
    class I2C_Controller_driver_class extends uvm_driver #(I2C_Controller_sequence_item_class);

        // Factory registration
        `uvm_component_utils(I2C_Controller_driver_class);

        // Sequence item and interface declarations
        I2C_Controller_sequence_item_class driver_sequence_item;
        i2c_config_class driver_config_db;
        virtual I2C_interface driver_interface;
        bit scl_past;

        // Constructor
        function new (string name = "uvm_driver_class" , uvm_component parent = null);
            super.new(name,parent);
        endfunction

        // Build phase
        function void build_phase (uvm_phase phase);
            super.build_phase(phase);
            uvm_config_db#(i2c_config_class)::get(this, "", "cfg", driver_config_db);
        endfunction //build_phase

        // Run phase: drive interface signals
        task run_phase (uvm_phase phase);
            super.run_phase(phase);
            driver_interface.rst_n = 0;
            @(negedge driver_interface.clk);
            forever begin
                driver_sequence_item = I2C_Controller_sequence_item_class::type_id::create("driver_sequence_item");
                seq_item_port.get_next_item(driver_sequence_item);
                if (!driver_config_db.is_top) begin
                    driver_interface.SDA_in = driver_sequence_item.SDA_in;
                end
                driver_interface.rst_n = driver_sequence_item.rst_n;
                driver_interface.controller_data_req <= driver_sequence_item.controller_data_req;
                driver_interface.controller_addr_req <= driver_sequence_item.controller_addr_req;
                driver_interface.controller_valid_req <= driver_sequence_item.controller_valid_req;
                driver_interface.controller_operation_req <= driver_sequence_item.controller_operation_req;
                driver_interface.controller_restart_req <= driver_sequence_item.controller_restart_req;
                driver_interface.error_signal <= driver_sequence_item.error_signal;
                scl_past <= driver_interface.SCL_in;
                @(negedge driver_interface.clk);
                seq_item_port.item_done();
                #0;
            end
        endtask //run_phase

    endclass //I2C_Controller_driver_class

endpackage //I2C_Controller_driver_pkg
```

3.1.15 I2C_Controller_agent

```
////////////////////////////////////////////////////////////////////////////////
//Name: Abdelrahman Mohamed Ragab
// Module-Name: I2C_Controller_agent
////////////////////////////////////////////////////////////////////////////////

package I2C_Controller_agent_pkg;

    // Import UVM and related packages
    import uvm_pkg::*;
    import I2C_Controller_sequencer_pkg::*;
    import I2C_Controller_monitor_pkg::*;
    import I2C_Controller_driver_pkg::*;
    import I2C_Controller_sequence_item_pkg::*;
    import i2c_config_package::*;
    `include "uvm_macros.svh";

    // I2C_Controller top TX agent class extending uvm_agent
    class I2C_Controller_agent_class extends uvm_agent;

        // Factory registration
        `uvm_component_utils(I2C_Controller_agent_class);

        // Component declarations
        I2C_Controller_driver_class          agent_driver;
        I2C_Controller_monitor_class         agent_monitor;
        I2C_Controller_sequence_item_class   agent_sequence_item;
        I2C_Controller_sequencer_class       agent_sequencer;
        i2c_config_class                     agent_config_db;

        // Analysis port declaration
        uvm_analysis_port#(I2C_Controller_sequence_item_class) agent_analysis_port;

        // Constructor
        function new(string name = "I2C_Controller_agent_class", uvm_component parent = null);
            super.new(name, parent);
        endfunction // new()

        // Build phase: create components
        function void build_phase(uvm_phase phase);
            super.build_phase(phase);

            uvm_config_db#(i2c_config_class)::get(this, "", "cfg", agent_config_db);

            if (agent_config_db.is_active == UVM_ACTIVE) begin
                agent_driver        = I2C_Controller_driver_class::type_id::create("agent_driver", this);
            end

            agent_monitor       = I2C_Controller_monitor_class::type_id::create("agent_monitor", this);
            agent_sequencer     = I2C_Controller_sequencer_class::type_id::create("agent_sequencer", this);

            agent_analysis_port = new("agent_analysis_port", this);
        endfunction // build_phase()

        // Connect phase: connect TLM ports and interfaces
        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);

            if (agent_config_db.is_active == UVM_ACTIVE) begin
                agent_driver.seq_item_port.connect(agent_sequencer.seq_item_export);
                agent_driver.driver_interface = agent_config_db.vif;
            end

            agent_monitor.monitor_interface = agent_config_db.vif;
            agent_monitor.monitor_analysis_port.connect(agent_analysis_port);
        endfunction // connect_phase()

    endclass // I2C_Controller_agent_class

endpackage // I2C_Controller_agent_pkg
```

3.1.16 I2C_Controller_environment

```
////////////////////////////////////////////////////////////////////////////////
//Name: Abdelrahman Mohamed Ragab
// Module-Name: I2C_Controller_enviroment
////////////////////////////////////////////////////////////////////////////////

package I2C_Controller_enviroment_pkg;

    // Import UVM and related packages
    import uvm_pkg::*;
    import I2C_Controller_agent_pkg::*;
    import I2C_Controller_scoreboard_pkg::*;
    import i2c_config_package::*;
    `include "uvm_macros.svh";

    // I2C_Controller TX environment class extending uvm_env
    class I2C_Controller_enviroment_class extends uvm_env;

        // Factory registration
        `uvm_component_utils(I2C_Controller_enviroment_class);

        // Agent, scoreboard, and coverage declarations
        I2C_Controller_agent_class enviroment_agent;
        I2C_Controller_scoreboard_class enviroment_scoreboard;
        i2c_config_class enviroment_config_db;

        // Constructor
        function new(string name = "I2C_Controller_enviroment_class" , uvm_component parent = null);
            super.new(name,parent);
        endfunction //new()

        // Build phase
        function void build_phase (uvm_phase phase);
            super.build_phase(phase);
            enviroment_agent = I2C_Controller_agent_class::type_id::create("enviroment_agent" , this);
            enviroment_config_db = i2c_config_class::type_id::create("enviroment_config_db");
            enviroment_scoreboard = I2C_Controller_scoreboard_class::type_id::create("enviroment_scoreboard" , this);

            uvm_config_db#(virtual I2C_interface)::get(this, "", "int", enviroment_config_db.vif);
            enviroment_config_db.is_active = UVM_ACTIVE;
            enviroment_config_db.i2c_address = 'h10;
            enviroment_config_db.is_top = 0;
            uvm_config_db#(i2c_config_class)::set(this, "*", "cfg", enviroment_config_db);
        endfunction //build_phase()

        // Connect phase
        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            enviroment_agent.agent_analysis_port.connect(enviroment_scoreboard.scoreboard_analysis_export);
        endfunction //connect_phase()

    endclass //I2C_Controller_enviroment_class

endpackage //I2C_Controller_enviroment_pkg
```

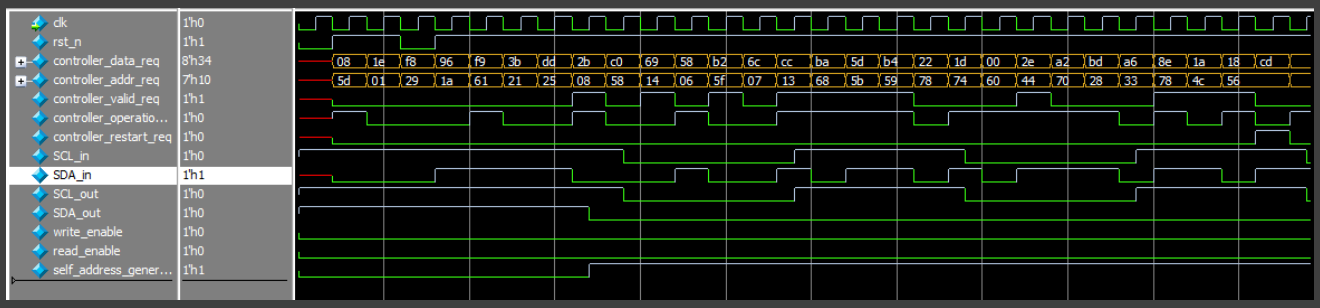
3.1.17 I2C_Controller_test

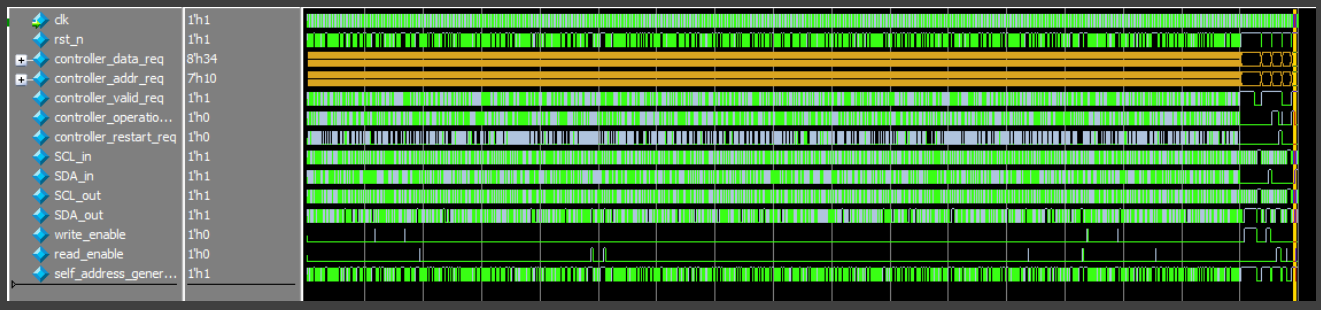
```
////////////////////////////////////////////////////////////////////////////////
//Name: Abdelrahman Mohamed Ragab
// Module-Name: I2C_Controller_test
////////////////////////////////////////////////////////////////////////////////

package I2C_Controller_test_pkg;

    // Import UVM and related packages
    import uvm_pkg::*;
    import I2C_Controller_enviroment_pkg::*;
    import I2C_Controller_randomized_sequence_pkg::*;
    import I2C_Controller_virtual_sequence_pkg::*;
    `include "uvm_macros.svh";

    // I2C_Controller TX test class extending uvm_test
    class I2C_Controller_test_class extends uvm_test;

        // Factory registration
        `uvm_component_utils(I2C_Controller_test_class);

        // Environment and sequence declarations
        I2C_Controller_enviroment_class test_enviroment;
        I2C_Controller_randomized_sequence_class test_randomized_sequence;
        I2C_Controller_virtual_sequence_class virtual_sequence;

        // Constructor
        function new(string name = "I2C_Controller_test_class" , uvm_component parent = null);
            super.new(name,parent);
        endfunction //new()

        // Build phase
        function void build_phase (uvm_phase phase);
            super.build_phase(phase);
            test_enviroment = I2C_Controller_enviroment_class::type_id::create("test_enviroment" ,this);
            test_randomized_sequence = I2C_Controller_randomized_sequence_class::type_id::create("test_randomized_sequence");
            virtual_sequence = I2C_Controller_virtual_sequence_class::type_id::create("virtual_sequence");
        endfunction //build_phase()

        // Run phase
        task run_phase (uvm_phase phase);
            super.run_phase(phase);
            virtual_sequence.vitual_sequence_sequencer = test_enviroment.enviroment_agent.agent_sequencer;
            phase.raise_objection(this);
            repeat(20000) begin
                test_randomized_sequence.start(test_enviroment.enviroment_agent.agent_sequencer);
            end
            `uvm_info("run_phase","finish first test",UVM_MEDIUM);
            virtual_sequence.start(null);
            `uvm_info("run_phase","finish second test",UVM_MEDIUM);
            phase.drop_objection(this);
        endtask //run_phase()

    endclass //I2C_Controller_test_class

endpackage
```

3.1.18 I2C_Controller_top

```
////////////////////////////////////////////////////////////////////////////////
//Name: Abdelrahman Mohamed Ragab
// Module-Name: I2C_Controller_top
////////////////////////////////////////////////////////////////////////////////

import uvm_pkg::*;
import I2C_Controller_test_pkg::*;
`include "uvm_macros.svh";

module I2C_Controller_top ();

bit clk;

initial begin
    clk = 0;
    forever begin
        #20;
        clk =~clk;
    end
end

I2C_interface I2C_Controller_interface (clk);

I2C_Controller I2C_Controller (
    .clk(I2C_Controller_interface.clk),
    .rst_n(I2C_Controller_interface.rst_n),
    .controller_data_req(I2C_Controller_interface.controller_data_req),
    .controller_addr_req(I2C_Controller_interface.controller_addr_req),
    .controller_valid_req(I2C_Controller_interface.controller_valid_req),
    .controller_operation_req(I2C_Controller_interface.controller_operation_req),
    .controller_restart_req(I2C_Controller_interface.controller_restart_req),
    .SCL_in(I2C_Controller_interface.SCL_in),
    .SDA_in(I2C_Controller_interface.SDA_in),
    .SCL_out(I2C_Controller_interface.SCL_out),
    .SDA_out(I2C_Controller_interface.SDA_out),
    .write_enable(I2C_Controller_interface.write_enable),
    .read_enable(I2C_Controller_interface.read_enable),
    .self_address_generated(I2C_Controller_interface.self_address_generated)
);

assign I2C_Controller_interface.SCL_in = I2C_Controller_interface.SCL_out;

initial begin
    uvm_config_db #(virtual I2C_interface)::set(null,"*","int",I2C_Controller_interface);
    run_test("I2C_Controller_test_class");
end


endmodule
```

3.1.19 Simulation Results



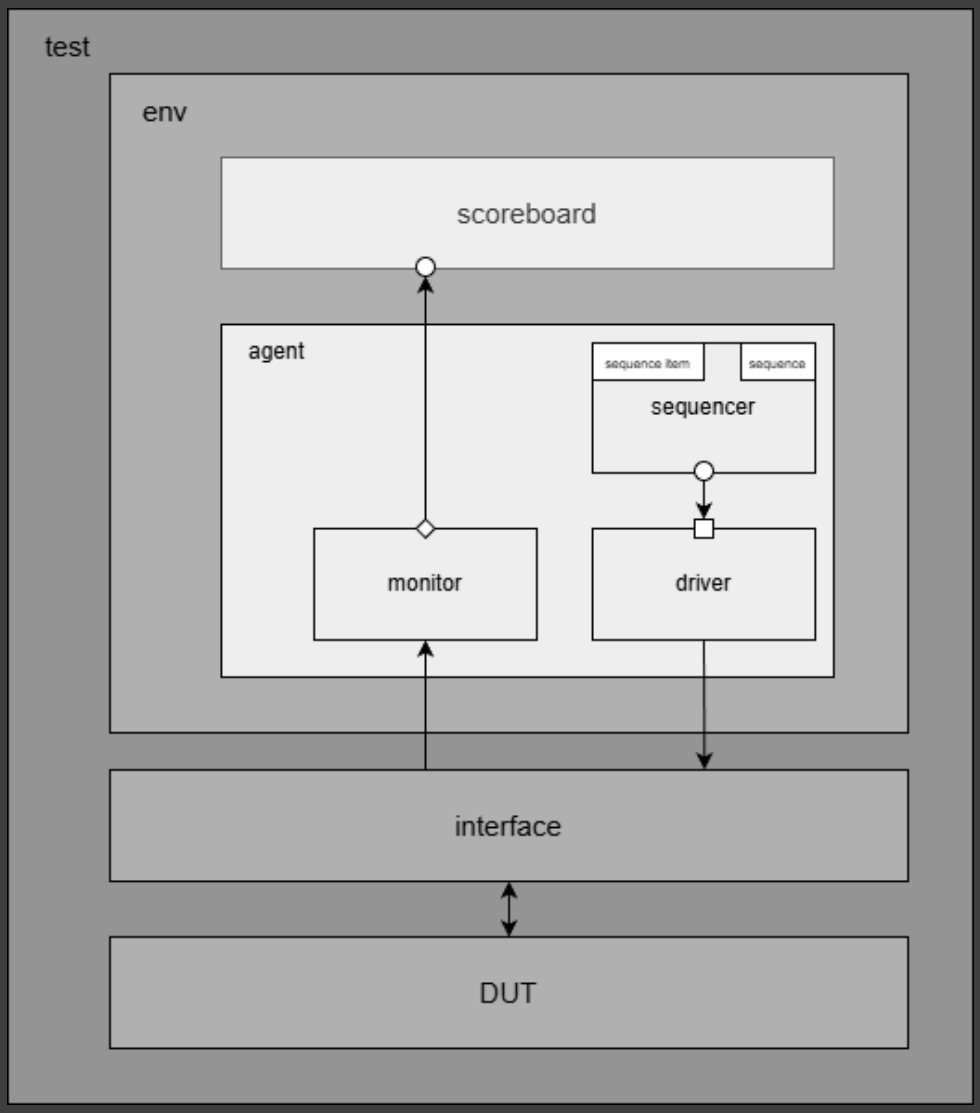



```
# UVM_INFO I2C_Controller_scoreboard.sv(357) @ 850060: uvm_test_top.test_enviroment.enviroment_scoreboard [report_phase] correct case = 3918 , wrong case = 0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :25188
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
```

As we can see there is no errors

3.2 I2C_pattern_detector module Block-Level UVM Environment

3.2.1 I2C_ pattern_detector Verification Plan and Environment



label	description	stimulus generated	checkers
randomization sequence	In this sequence we will randomize the control module to make sire that the overall functionality of the design	randomization for all the inputs with constraining the reset to be on 5% of the time and that the SDA cant change at the positive edge of the SCL to not violate the protocol setup time	we check the functionality of the outputs using checkers comparing the output with the reference output from golden model we check the ACK generation feature and the write/read enable property using assertion.
reset feature	this is a directed seequence to check the behaviour of the reset	directed	we check the functionality of the outputs using checkers comparing the output with the reference output from golden model we also use assertion to check the reset operation.
start detection feature	this is a directed seequence to check the detection of the start sequence	directed	we check the functionality of the outputs using checkers comparing the output with the reference output from golden model
stop detection feature	this is a directed seequence to check the detection of the stop sequence	directed	we check the functionality of the outputs using checkers comparing the output with the reference output from golden model
address detection	this is a directed sequences where the address is/isnt the address of the traget to check the behaviour of address detection	directed	we check the functionality of the outputs using checkers comparing the output with the reference output from golden model we check the ACK generation feature and the write/read enable property using assertion.
read/write detection	this is a directed sequences to check the detection of the write operation and generation of the read/write enables based on the message.	directed	we check the functionality of the outputs using checkers comparing the output with the reference output from golden model we check the ACK generation feature and the write/read enable property using assertion.

3.2.2 I2C_configuration_object.

```
package i2c_config_package;

    import uvm_pkg::*;
    `include "uvm_macros.svh";

    class i2c_config_class extends uvm_object;
        `uvm_object_utils(i2c_config_class)

        virtual I2C_interface vif;
        bit is_active = UVM_ACTIVE;
        bit is_top = 1;
        bit [6:0] i2c_address;

        function new(string name = "i2c_config_class");
            super.new(name);
        endfunction
    endclass
endpackage
```

3.2.3 I2C_ pattern_detector _sequence_item

```
////////////////////////////////////////////////////////////////////////////////
//Name: Abdelrahman Mohamed Ragab
// Module-Name: I2C_Pattern_Detector_sequence_item
////////////////////////////////////////////////////////////////////////////////

package I2C_Pattern_Detector_sequence_item_pkg;

    // Import UVM and related packages
    import uvm_pkg::*;
    `include "uvm_macros.svh";

    parameter ADDRESS = 7'h10;
    parameter ADDRESS_WIDTH = 7;

    // I2C_Pattern_Detector TX sequence item class extending uvm_sequence_item
    class I2C_Pattern_Detector_sequence_item_class extends uvm_sequence_item;

        // Factory registration
        `uvm_object_utils(I2C_Pattern_Detector_sequence_item_class);

        //input signals
        rand logic SCL_in;
        rand logic SDA_in;
        rand logic rst_n;
        rand logic self_address_generated;

        //output signals
        logic wr_enable;
        logic rd_enable;
        logic SDA_out;
        logic start_state;
        logic stop_state;
        logic random;

        //reference signals
        logic start_state_ref;
        logic stop_state_ref;

        logic [ADDRESS_WIDTH:0] control_signal [$];
        logic start_checking;
        bit SCL_in_past;
        bit SDA_in_past;

        // Constructor
        function new(string name = "I2C_Pattern_Detector_sequence_item_class");
            super.new(name);
        endfunction //new()

        function void post_randomize();
          SCL_in_past = SCL_in;
          SDA_in_past = SDA_in;
        endfunction

        // Main constraint: reset is mostly 0, write/read enables mostly 1
        constraint c {
            rst_n dist {0:/5 , 1:/95};
            self_address_generated dist {0:/95 , 1:/5};
            if (SCL_in == 1 && SCL_in_past == 0) SDA_in == SDA_in_past;
        }

    endclass //I2C_Pattern_Detector_sequence_item_class extends uvm_sequence_item

endpackage //I2C_Pattern_Detector_sequence_item_pkg
```

3.2.4 I2C_pattern_detector_randomized_sequence

```
////////////////////////////////////////////////////////////////////////////////
//Name: Abdelrahman Mohamed Ragab
// Module-Name: I2C_Pattern_Detector_randomized_sequence
////////////////////////////////////////////////////////////////////////////////

package I2C_Pattern_Detector_randomized_sequence_pkg;

    // Import UVM and related packages
    import uvm_pkg::*;
    import I2C_Pattern_Detector_sequence_item_pkg::*;
    `include "uvm_macros.svh";

    // I2C_Pattern_Detector TX main sequence class extending uvm_sequence
    class I2C_Pattern_Detector_randomized_sequence_class extends uvm_sequence #(I2C_Pattern_Detector_sequence_item_class);

        // Factory registration
        `uvm_object_utils(I2C_Pattern_Detector_randomized_sequence_class);

        // Sequence item declaration
        I2C_Pattern_Detector_sequence_item_class MAIN_sequence_sequence_item;

        // Constructor
        function new(string name = "I2C_Pattern_Detector_randomized_sequence_class");
            super.new(name);
        endfunction //new()

        // Body task
        task body;
            repeat(20000) begin
                MAIN_sequence_sequence_item = I2C_Pattern_Detector_sequence_item_class::type_id::create("MAIN_sequence_sequence_item");
                start_item(MAIN_sequence_sequence_item);
                assert (MAIN_sequence_sequence_item.randomize());
                finish_item(MAIN_sequence_sequence_item);
            end

        endtask //body()

    endclass //I2C_Pattern_Detector_randomized_sequence_class extends uvm_sequence

endpackage //I2C_Pattern_Detector_randomized_sequence_pkg
```

3.2.5 I2C_Pattern_Detector_directed_address_sequence

```
////////////////////////////////////////////////////////////////////////////////
//Name: Abdelrahman Mohamed Ragab
// Module-Name: I2C_Pattern_Detector_directed_address_sequence
////////////////////////////////////////////////////////////////////////////////

package I2C_Pattern_Detector_directed_address_sequence_pkg;

    // Import UVM and related packages
    import uvm_pkg::*;
    import I2C_Pattern_Detector_sequence_item_pkg::*;
    `include "uvm_macros.svh";

    // I2C_Pattern_Detector TX main sequence class extending uvm_sequence
    class I2C_Pattern_Detector_directed_address_sequence_class extends uvm_sequence #(I2C_Pattern_Detector_sequence_item_class);

        // Factory registration
        `uvm_object_utils(I2C_Pattern_Detector_directed_address_sequence_class);

        // Sequence item declaration
        I2C_Pattern_Detector_sequence_item_class MAIN_sequence_item;

        // Constructor
        function new(string name = "I2C_Pattern_Detector_directed_address_sequence_class");
            super.new(name);
        endfunction //new()

        logic [6:0] address = 7'h10;
        int i;

        // Body task
        task body;
                MAIN_sequence_item = I2C_Pattern_Detector_sequence_item_class::type_id::create("MAIN_sequence_item");
                i = 0;
                repeat (7) begin
                    start_item(MAIN_sequence_item);
                    MAIN_sequence_item.self_address_generated = 0;
                    MAIN_sequence_item.SCL_in= 0;
                    MAIN_sequence_item.rst_n = 1;
                    MAIN_sequence_item.SDA_in=address[6-i];
                    i++;
                    finish_item(MAIN_sequence_item);

                    start_item(MAIN_sequence_item);
                    MAIN_sequence_item.SCL_in= 1;
                    finish_item(MAIN_sequence_item);
                end

        endtask //body()

    endclass //I2C_Pattern_Detector_directed_address_sequence_class extends uvm_sequence

endpackage //I2C_Pattern_Detector_directed_address_sequence_pkg
```

3.2.6 I2C_Pattern_Detector_directed_read_sequence

```
////////////////////////////////////////////////////////////////////////////////
//Name: Abdelrahman Mohamed Ragab
// Module-Name: I2C_Pattern_Detector_driven_read_sequence
////////////////////////////////////////////////////////////////////////////////

package I2C_Pattern_Detector_driven_read_sequence_pkg;

    // Import UVM and related packages
    import uvm_pkg::*;
    import I2C_Pattern_Detector_sequence_item_pkg::*;
    `include "uvm_macros.svh";

    // I2C_Pattern_Detector TX main sequence class extending uvm_sequence
    class I2C_Pattern_Detector_driven_read_sequence_class extends uvm_sequence #(I2C_Pattern_Detector_sequence_item_class);

        // Factory registration
        `uvm_object_utils(I2C_Pattern_Detector_driven_read_sequence_class);

        // Sequence item declaration
        I2C_Pattern_Detector_sequence_item_class MAIN_sequence_item;

        // Constructor
        function new(string name = "I2C_Pattern_Detector_driven_read_sequence_class");
            super.new(name);
        endfunction //new()

        logic [6:0] address = 7'h10;
        int i;

        // Body task
        task body;
                MAIN_sequence_item = I2C_Pattern_Detector_sequence_item_class::type_id::create("MAIN_sequence_item");

                start_item(MAIN_sequence_item);
                    MAIN_sequence_item.SCL_in= 0;
                    MAIN_sequence_item.SDA_in=1;
                    MAIN_sequence_item.rst_n = 1;
                    MAIN_sequence_item.self_address_generated = 0;
                finish_item(MAIN_sequence_item);

                start_item(MAIN_sequence_item);
                    MAIN_sequence_item.SCL_in= ~MAIN_sequence_item.SCL_in;
                finish_item(MAIN_sequence_item);

                repeat (2) begin
                    start_item(MAIN_sequence_item);
                        MAIN_sequence_item.SCL_in= ~MAIN_sequence_item.SCL_in;
                    finish_item(MAIN_sequence_item);
                end

                repeat (10) begin
                    start_item(MAIN_sequence_item);
                    MAIN_sequence_item.SCL_in= ~MAIN_sequence_item.SCL_in;
                    MAIN_sequence_item.SDA_in=0;
                    finish_item(MAIN_sequence_item);

                    start_item(MAIN_sequence_item);
                    MAIN_sequence_item.SCL_in= ~MAIN_sequence_item.SCL_in;
                    finish_item(MAIN_sequence_item);
                end

        endtask //body()

    endclass //I2C_Pattern_Detector_driven_read_sequence_class extends uvm_sequence

endpackage //I2C_Pattern_Detector_driven_read_sequence_pkg
```

3.2.7 I2C_Pattern_Detector_directed_reset_sequence

```
////////////////////////////////////////////////////////////////////////////////
//Name: Abdelrahman Mohamed Ragab
// Module-Name: I2C_Pattern_Detector_directed_reset_sequence
////////////////////////////////////////////////////////////////////////////////

package I2C_Pattern_Detector_directed_reset_sequence_pkg;

    // Import UVM and related packages
    import uvm_pkg::*;
    import I2C_Pattern_Detector_sequence_item_pkg::*;
    `include "uvm_macros.svh";

    // I2C_Pattern_Detector TX main sequence class extending uvm_sequence
    class I2C_Pattern_Detector_directed_reset_sequence_class extends uvm_sequence #(I2C_Pattern_Detector_sequence_item_class);

        // Factory registration
        `uvm_object_utils(I2C_Pattern_Detector_directed_reset_sequence_class);

        // Sequence item declaration
        I2C_Pattern_Detector_sequence_item_class MAIN_sequence_item;

        // Constructor
        function new(string name = "I2C_Pattern_Detector_directed_reset_sequence_class");
            super.new(name);
        endfunction //new()

        logic [6:0] address = 7'h10;
        int i;

        // Body task
        task body;
                MAIN_sequence_item = I2C_Pattern_Detector_sequence_item_class::type_id::create("MAIN_sequence_item");

                start_item(MAIN_sequence_item);
                    MAIN_sequence_item.rst_n=0;
                    MAIN_sequence_item.SCL_in=1;
                    MAIN_sequence_item.SDA_in=1;
                    MAIN_sequence_item.self_address_generated = 0;
                finish_item(MAIN_sequence_item);

        endtask //body()

    endclass //I2C_Pattern_Detector_directed_reset_sequence_class extends uvm_sequence

endpackage //I2C_Pattern_Detector_directed_reset_sequence_pkg
```

3.2.8 I2C_Pattern_Detector_directed_start_sequence

```
////////////////////////////////////////////////////////////////////////////////
//Name: Abdelrahman Mohamed Ragab
// Module-Name: I2C_Pattern_Detector_directed_start_sequence
////////////////////////////////////////////////////////////////////////////////

package I2C_Pattern_Detector_directed_start_sequence_pkg;

    // Import UVM and related packages
    import uvm_pkg::*;
    import I2C_Pattern_Detector_sequence_item_pkg::*;
    `include "uvm_macros.svh";

    // I2C_Pattern_Detector TX main sequence class extending uvm_sequence
    class I2C_Pattern_Detector_directed_start_sequence_class extends uvm_sequence #(I2C_Pattern_Detector_sequence_item_class);

        // Factory registration
        `uvm_object_utils(I2C_Pattern_Detector_directed_start_sequence_class);

        // Sequence item declaration
        I2C_Pattern_Detector_sequence_item_class MAIN_sequence_item;

        // Constructor
        function new(string name = "I2C_Pattern_Detector_directed_start_sequence_class");
            super.new(name);
        endfunction //new()

        logic [6:0] address = 7'h10;

        // Body task
        task body;
                MAIN_sequence_item = I2C_Pattern_Detector_sequence_item_class::type_id::create("MAIN_sequence_item");

                start_item(MAIN_sequence_item);
                    MAIN_sequence_item.SCL_in=1;
                    MAIN_sequence_item.SDA_in=0;
                    MAIN_sequence_item.rst_n=1;
                    MAIN_sequence_item.self_address_generated = 0;
                finish_item(MAIN_sequence_item);

        endtask //body()

    endclass //I2C_Pattern_Detector_directed_start_sequence_class extends uvm_sequence

endpackage //I2C_Pattern_Detector_directed_start_sequence_pkg
```

3.2.9 I2C_Pattern_Detector_directed_stop_sequence

```
////////////////////////////////////////////////////////////////////////////////
//Name: Abdelrahman Mohamed Ragab
// Module-Name: I2C_Pattern_Detector_directed_stop_sequence
////////////////////////////////////////////////////////////////////////////////

package I2C_Pattern_Detector_directed_stop_sequence_pkg;

    // Import UVM and related packages
    import uvm_pkg::*;
    import I2C_Pattern_Detector_sequence_item_pkg::*;
    `include "uvm_macros.svh";

    // I2C_Pattern_Detector TX main sequence class extending uvm_sequence
    class I2C_Pattern_Detector_directed_stop_sequence_class extends uvm_sequence #(I2C_Pattern_Detector_sequence_item_class);

        // Factory registration
        `uvm_object_utils(I2C_Pattern_Detector_directed_stop_sequence_class);

        // Sequence item declaration
        I2C_Pattern_Detector_sequence_item_class MAIN_sequence_item;

        // Constructor
        function new(string name = "I2C_Pattern_Detector_directed_stop_sequence_class");
            super.new(name);
        endfunction //new()

        logic [6:0] address = 7'h10;
        int i;

        // Body task
        task body;
                MAIN_sequence_item = I2C_Pattern_Detector_sequence_item_class::type_id::create("MAIN_sequence_item");

                start_item(MAIN_sequence_item);
                    MAIN_sequence_item.SCL_in= 0;
                    MAIN_sequence_item.SDA_in=0;
                    MAIN_sequence_item.rst_n = 1;
                    MAIN_sequence_item.self_address_generated = 0;
                finish_item(MAIN_sequence_item);

                start_item(MAIN_sequence_item);
                    MAIN_sequence_item.SCL_in= 1;
                    MAIN_sequence_item.SDA_in=0;
                finish_item(MAIN_sequence_item);

                repeat (2) begin
                    start_item(MAIN_sequence_item);
                    MAIN_sequence_item.SCL_in= 1;
                    MAIN_sequence_item.SDA_in=1;
                    finish_item(MAIN_sequence_item);
                end

        endtask //body()

    endclass //I2C_Pattern_Detector_directed_stop_sequence_class extends uvm_sequence

endpackage //I2C_Pattern_Detector_directed_stop_sequence_pkg
```

3.2.10 I2C_Pattern_Detector_directed_write_sequence

```
////////////////////////////////////////////////////////////////////////////////
//Name: Abdelrahman Mohamed Ragab
// Module-Name: I2C_Pattern_Detector_driven_write_sequence
////////////////////////////////////////////////////////////////////////////////

package I2C_Pattern_Detector_driven_write_sequence_pkg;

    // Import UVM and related packages
    import uvm_pkg::*;
    import I2C_Pattern_Detector_sequence_item_pkg::*;
    `include "uvm_macros.svh";

    // I2C_Pattern_Detector TX main sequence class extending uvm_sequence
    class I2C_Pattern_Detector_driven_write_sequence_class extends uvm_sequence #(I2C_Pattern_Detector_sequence_item_class);

        // Factory registration
        `uvm_object_utils(I2C_Pattern_Detector_driven_write_sequence_class);

        // Sequence item declaration
        I2C_Pattern_Detector_sequence_item_class MAIN_sequence_item;

        // Constructor
        function new(string name = "I2C_Pattern_Detector_driven_write_sequence_class");
            super.new(name);
        endfunction //new()

        logic [6:0] address = 7'h10;
        int i;

        // Body task
        task body;
                MAIN_sequence_item = I2C_Pattern_Detector_sequence_item_class::type_id::create("MAIN_sequence_item");

                start_item(MAIN_sequence_item);
                    MAIN_sequence_item.SCL_in= 0;
                    MAIN_sequence_item.SDA_in=0;
                    MAIN_sequence_item.rst_n = 1;
                    MAIN_sequence_item.self_address_generated = 0;
                finish_item(MAIN_sequence_item);

                repeat (2) begin
                    start_item(MAIN_sequence_item);
                        MAIN_sequence_item.SCL_in= ~MAIN_sequence_item.SCL_in;
                    finish_item(MAIN_sequence_item);
                end

                repeat (10) begin
                    start_item(MAIN_sequence_item);
                    MAIN_sequence_item.SCL_in= ~MAIN_sequence_item.SCL_in;
                    MAIN_sequence_item.SDA_in=0;
                    finish_item(MAIN_sequence_item);

                    start_item(MAIN_sequence_item);
                    MAIN_sequence_item.SCL_in= ~MAIN_sequence_item.SCL_in;
                    finish_item(MAIN_sequence_item);
                end

        endtask //body()

    endclass //I2C_Pattern_Detector_driven_write_sequence_class extends uvm_sequence

endpackage //I2C_Pattern_Detector_driven_write_sequence_pkg
```

3.2.11 I2C_Pattern_Detector_directed_wrong_address_sequence

```
////////////////////////////////////////////////////////////////////////////////
//Name: Abdelrahman Mohamed Ragab
// Module-Name: I2C_Pattern_Detector_directed_wrong_address_sequence
////////////////////////////////////////////////////////////////////////////////

package I2C_Pattern_Detector_directed_wrong_address_sequence_pkg;

    // Import UVM and related packages
    import uvm_pkg::*;
    import I2C_Pattern_Detector_sequence_item_pkg::*;
    `include "uvm_macros.svh";

    // I2C_Pattern_Detector TX main sequence class extending uvm_sequence
    class I2C_Pattern_Detector_directed_wrong_address_sequence_class extends uvm_sequence #(I2C_Pattern_Detector_sequence_item_class);

        // Factory registration
        `uvm_object_utils(I2C_Pattern_Detector_directed_wrong_address_sequence_class);

        // Sequence item declaration
        I2C_Pattern_Detector_sequence_item_class MAIN_sequence_item;

        // Constructor
        function new(string name = "I2C_Pattern_Detector_directed_wrong_address_sequence_class");
            super.new(name);
        endfunction //new()

        logic [6:0] address = 7'h10;
        int i;

        // Body task
        task body;
                MAIN_sequence_item = I2C_Pattern_Detector_sequence_item_class::type_id::create("MAIN_sequence_item");
                repeat (7) begin
                    start_item(MAIN_sequence_item);
                    MAIN_sequence_item.SCL_in= 0;
                    MAIN_sequence_item.rst_n = 1;
                    MAIN_sequence_item.SDA_in=0;
                    MAIN_sequence_item.self_address_generated = 0;
                    finish_item(MAIN_sequence_item);

                    start_item(MAIN_sequence_item);
                    MAIN_sequence_item.SCL_in= 1;
                    finish_item(MAIN_sequence_item);
                end

        endtask //body()

    endclass //I2C_Pattern_Detector_directed_wrong_address_sequence_class extends uvm_sequence

endpackage //I2C_Pattern_Detector_directed_wrong_address_sequence_pkg
```

3.2.12 I2C_ Pattern_Detector _sequencer

```
////////////////////////////////////////////////////////////////////////////////
//Name: Abdelrahman Mohamed Ragab
// Module-Name: I2C_Pattern_Detector_sequencer
////////////////////////////////////////////////////////////////////////////////

package I2C_Pattern_Detector_sequencer_pkg;

    // Import UVM and related packages
    import uvm_pkg::*;
    import I2C_Pattern_Detector_sequence_item_pkg::*;
    `include "uvm_macros.svh";

    // I2C_Pattern_Detector TX sequencer class extending uvm_sequencer
    class I2C_Pattern_Detector_sequencer_class extends uvm_sequencer #(I2C_Pattern_Detector_sequence_item_class);

        // Factory registration
        `uvm_component_utils(I2C_Pattern_Detector_sequencer_class);

        // Constructor
        function new(string name = "I2C_Pattern_Detector_sequencer_class" , uvm_component parent = null);
            super.new(name,parent);
        endfunction //new()

    endclass //I2C_Pattern_Detector_sequencer_class

endpackage //I2C_Pattern_Detector_sequencer_pkg
```

3.2.13 I2C_Pattern_Detector _virtual_sequence

```
package I2C_Pattern_Detector_virtual_sequence_pkg;

// Import UVM and related packages
    import uvm_pkg::*;
    import I2C_Pattern_Detector_sequence_item_pkg::*;
    import I2C_Pattern_Detector_directed_reset_sequence_pkg::*;
    import I2C_Pattern_Detector_directed_start_sequence_pkg::*;
    import I2C_Pattern_Detector_directed_address_sequence_pkg::*;
    import I2C_Pattern_Detector_driven_read_sequence_pkg::*;
    import I2C_Pattern_Detector_driven_write_sequence_pkg::*;
    import I2C_Pattern_Detector_directed_stop_sequence_pkg::*;
    import I2C_Pattern_Detector_directed_stop_sequence_pkg::*;
    import I2C_Pattern_Detector_directed_wrong_address_sequence_pkg::*;
    import I2C_Pattern_Detector_sequencer_pkg::*;
    `include "uvm_macros.svh";

    // I2C Pattern Detector virtual sequence class extending uvm_sequence
    class I2C_Pattern_Detector_virtual_sequence_class extends uvm_sequence #(I2C_Pattern_Detector_sequence_item_class);

        // Factory registration
        `uvm_object_utils(I2C_Pattern_Detector_virtual_sequence_class);

        // Sequence item declaration
        I2C_Pattern_Detector_sequence_item_class virtual_sequence_item;
        I2C_Pattern_Detector_directed_reset_sequence_class directed_reset_sequence;
        I2C_Pattern_Detector_directed_start_sequence_class directed_start_sequence;
        I2C_Pattern_Detector_directed_address_sequence_class directed_address_sequence;
        I2C_Pattern_Detector_directed_wrong_address_sequence_class directed_wrong_address_sequence;
        I2C_Pattern_Detector_driven_read_sequence_class directed_read_sequence;
        I2C_Pattern_Detector_driven_write_sequence_class directed_write_sequence;
        I2C_Pattern_Detector_directed_stop_sequence_class directed_stop_sequence;
        I2C_Pattern_Detector_sequencer_class vitual_sequence_sequencer;

        // Constructor
        function new(string name = "I2C_Pattern_Detector_virtual_sequence_class");
            super.new(name);
        endfunction //new()

        task body();
            `uvm_info(get_type_name(), "virtual_seq: Inside Body", UVM_LOW);
            directed_reset_sequence = I2C_Pattern_Detector_directed_reset_sequence_class::type_id::create("directed_reset_sequence");
            directed_start_sequence = I2C_Pattern_Detector_directed_start_sequence_class::type_id::create("directed_start_sequence");
            directed_address_sequence = I2C_Pattern_Detector_directed_address_sequence_class::type_id::create("directed_address_sequence");
            directed_wrong_address_sequence = I2C_Pattern_Detector_directed_wrong_address_sequence_class::type_id::create("directed_wrong_address_sequence");
            directed_read_sequence = I2C_Pattern_Detector_driven_read_sequence_class::type_id::create("directed_read_sequence");
            directed_write_sequence = I2C_Pattern_Detector_driven_write_sequence_class::type_id::create("directed_write_sequence");
            directed_stop_sequence = I2C_Pattern_Detector_directed_stop_sequence_class::type_id::create("directed_stop_sequence");

            directed_reset_sequence.start(vitual_sequence_sequencer);
            directed_start_sequence.start(vitual_sequence_sequencer);
            `uvm_info("run_phase","finish first directed test",UVM_MEDIUM);
            directed_address_sequence.start(vitual_sequence_sequencer);
            `uvm_info("run_phase","finish second directed test",UVM_MEDIUM);
            directed_read_sequence.start(vitual_sequence_sequencer);
            `uvm_info("run_phase","finish forth directed test",UVM_MEDIUM);
            directed_stop_sequence.start(vitual_sequence_sequencer);
            `uvm_info("run_phase","finish fifth directed test",UVM_MEDIUM);
            directed_reset_sequence.start(vitual_sequence_sequencer);
            directed_start_sequence.start(vitual_sequence_sequencer);
            `uvm_info("run_phase","finish sixth directed test",UVM_MEDIUM);
            directed_address_sequence.start(vitual_sequence_sequencer);
            `uvm_info("run_phase","finish seventh directed test",UVM_MEDIUM);
            directed_write_sequence.start(vitual_sequence_sequencer);
            `uvm_info("run_phase","finish eighth directed test",UVM_MEDIUM);
            directed_stop_sequence.start(vitual_sequence_sequencer);
            `uvm_info("run_phase","finish ninth directed test",UVM_MEDIUM);
            directed_reset_sequence.start(vitual_sequence_sequencer);
            directed_start_sequence.start(vitual_sequence_sequencer);
            `uvm_info("run_phase","finish tenth directed test",UVM_MEDIUM);
            directed_wrong_address_sequence.start(vitual_sequence_sequencer);
            `uvm_info("run_phase","finish eleventh directed test",UVM_MEDIUM);
            directed_write_sequence.start(vitual_sequence_sequencer);
            `uvm_info("run_phase","finish twelfth directed test",UVM_MEDIUM);
            directed_stop_sequence.start(vitual_sequence_sequencer);
            `uvm_info("run_phase","finish thirteenth directed test",UVM_MEDIUM);
            #10;
        endtask
    endclass //I2C_Pattern_Detector_virtual_sequence_class extends ucm_seq

endpackage
```

3.2.14 I2C_Pattern_Detector_scoreboard

```
////////////////////////////////////////////////////////////////////////////////
//Name: Abdelrahman Mohamed Ragab
// Module-Name: I2C_Pattern_Detector_scoreboard_pkg
////////////////////////////////////////////////////////////////////////////////

package I2C_Pattern_Detector_scoreboard_pkg;

    // Import UVM and related packages
    import uvm_pkg::*;
    import I2C_Pattern_Detector_sequence_item_pkg::*;
    import i2c_config_package::*;
    `include "uvm_macros.svh";

    // I2C_Pattern_Detector full scoreboard class extending uvm_scoreboard
    class I2C_Pattern_Detector_scoreboard_class extends uvm_scoreboard;

        // Factory registration
        `uvm_component_utils(I2C_Pattern_Detector_scoreboard_class);

        // Sequence item and analysis port declarations
        I2C_Pattern_Detector_sequence_item_class scoreboard_sequence_item;
        uvm_analysis_export #(I2C_Pattern_Detector_sequence_item_class) scoreboard_analysis_export;
        uvm_tlm_analysis_fifo #(I2C_Pattern_Detector_sequence_item_class) scoreboard_analysis_fifo;

        i2c_config_class scoreboard_confg_db;

        parameter ADDRESS_WIDTH = 7;

        int correct_case = 0;
        int wrong_case = 0;

        //ref signals
        logic random;
        logic wr_enable_ref;
        logic rd_enable_ref;
        logic start_state_ref;
        logic stop_state_ref;
        logic [ADDRESS_WIDTH:0] control_signal [$];

        logic [6:0] address;

        // Constructor
        function new(string name = "I2C_Pattern_Detector_scoreboard_class" , uvm_component parent = null);
            super.new(name,parent);
        endfunction //new()

        // Build phase
        function void build_phase (uvm_phase phase);
            super.build_phase(phase);
            uvm_config_db#(i2c_config_class)::get(this, "", "cfg", scoreboard_confg_db);
            scoreboard_analysis_export = new("scoreboard_analysis_export" , this);
            scoreboard_analysis_fifo = new("scoreboard_analysis_fifo" , this);
        endfunction //build_phase()

        // Connect phase
        function void connect_phase (uvm_phase phase);
            super.connect_phase(phase);
            scoreboard_analysis_export.connect(scoreboard_analysis_fifo.analysis_export);
        endfunction //connect_phase()

        // Run phase
        task run_phase (uvm_phase phase);
            super.run_phase(phase);
            forever begin
                `uvm_info("SB","--> Entered scoreboard run_phase", UVM_LOW);
                scoreboard_analysis_fifo.get(scoreboard_sequence_item);


                if (scoreboard_sequence_item.start_checking) begin
                    golden_reference(scoreboard_sequence_item, scoreboard_sequence_item.control_signal, wr_enable_ref, rd_enable_ref, address);
                        start_state_ref = scoreboard_sequence_item.start_state_ref;
                        stop_state_ref = scoreboard_sequence_item.stop_state_ref;
                    if (((scoreboard_sequence_item.wr_enable === wr_enable_ref) || scoreboard_confg_db.is_top)
                    && ((scoreboard_sequence_item.start_state === scoreboard_sequence_item.start_state_ref))
                    && ((scoreboard_sequence_item.stop_state === scoreboard_sequence_item.stop_state_ref))
                    && ((scoreboard_sequence_item.rd_enable === rd_enable_ref) || scoreboard_confg_db.is_top)) begin
                        correct_case++;
                        `uvm_info("run_phase",$sformatf("correct case = %0d , wrong case = %0d , time = %0d",correct_case,wrong_case,$time),UVM_MEDIUM);
                    end else begin
                        wrong_case++;
                        `uvm_info("run_phase",$sformatf("correct case = %0d , wrong case = %0d , time = %0d",correct_case,wrong_case,$time),UVM_MEDIUM);
                        display_signal_mismatches(scoreboard_sequence_item);
                    end
                end
            end
        endtask //run_phase()
```

```
    function void display_signal_mismatches(I2C_Pattern_Detector_sequence_item_class scoreboard_sequence_item);
        string mismatch_msg = "";
        int mismatch_count = 0;

        // Check each signal individually (in order from the original if condition)
        if (scoreboard_sequence_item.wr_enable !== wr_enable_ref) begin
            `uvm_error("run_phase", $sformatf("Found %0d signal mismatch(es) at time %0t:%s", 1, $time, $sformatf("[MISMATCH] wr_enable: DUT=%0h, REF=%0h", scoreboard_sequence_item.wr_enable, wr_enable_ref)));

            mismatch_msg = {mismatch_msg, $sformatf("[MISMATCH] wr_enable: DUT=%0h, REF=%0h", scoreboard_sequence_item.wr_enable, wr_enable_ref)};

            mismatch_count++;
        end

        if (scoreboard_sequence_item.start_state !== scoreboard_sequence_item.start_state_ref) begin
            `uvm_error("run_phase", $sformatf("Found %0d signal mismatch(es) at time %0t:%s", 1, $time, $sformatf("[MISMATCH] start_state: DUT=%0h, REF=%0h", scoreboard_sequence_item.start_state, scoreboard_sequence_item.start_state_ref)));

            mismatch_msg = {mismatch_msg, $sformatf("[MISMATCH] start_state: DUT=%0h, REF=%0h", scoreboard_sequence_item.start_state, scoreboard_sequence_item.start_state_ref)};

            mismatch_count++;
        end

        if (scoreboard_sequence_item.stop_state !== scoreboard_sequence_item.stop_state_ref) begin
            `uvm_error("run_phase", $sformatf("Found %0d signal mismatch(es) at time %0t:%s", 1, $time, $sformatf("[MISMATCH] stop_state: DUT=%0h, REF=%0h", scoreboard_sequence_item.stop_state, scoreboard_sequence_item.stop_state_ref)));

            mismatch_msg = {mismatch_msg, $sformatf("[MISMATCH] stop_state: DUT=%0h, REF=%0h", scoreboard_sequence_item.stop_state, scoreboard_sequence_item.stop_state_ref)};

            mismatch_count++;
        end

        if (scoreboard_sequence_item.rd_enable !== rd_enable_ref) begin
            `uvm_error("run_phase", $sformatf("Found %0d signal mismatch(es) at time %0t:%s", 1, $time, $sformatf("[MISMATCH] rd_enable: DUT=%0h, REF=%0h", scoreboard_sequence_item.rd_enable, rd_enable_ref)));

            mismatch_msg = {mismatch_msg, $sformatf("[MISMATCH] rd_enable: DUT=%0h, REF=%0h", scoreboard_sequence_item.rd_enable, rd_enable_ref)};

            mismatch_count++;
        end

        if (mismatch_count > 0) begin
            `uvm_info("run_phase", $sformatf("Found %0d signal mismatch(es) at time %0t:%s", mismatch_count, $time, mismatch_msg), UVM_HIGH);
        end
    endfunction

    task golden_reference(I2C_Pattern_Detector_sequence_item_class scoreboard_sequence_item, input logic [ADDRESS_WIDTH:0] control_signal [5],
                          output logic wr_enable_ref, output logic rd_enable_ref, output logic [6:0] address);
        if (!scoreboard_sequence_item.rst_n || scoreboard_sequence_item.stop_state_ref) begin
            rd_enable_ref = 0;
            wr_enable_ref = 0;
        end else begin
            for (int i = 6; i >=0; i--) begin
                address[i] = control_signal.pop_front();
            end
            if (address == scoreboard_config_db.i2c_address) begin
                if (control_signal[0]) begin
                    rd_enable_ref = 0;
                    wr_enable_ref = 1;
                end else begin
                    rd_enable_ref = 1;
                    wr_enable_ref = 0;
                end
            end else begin
                rd_enable_ref = 0;
                wr_enable_ref = 0;
            end
            end
    endtask

    // Report phase
    function void report_phase (uvm_phase phase);
        super.report_phase(phase);
        `uvm_info("report_phase",$sformatf("correct case = %0d , wrong case = %0d",correct_case,wrong_case),UVM_MEDIUM);
    endfunction

endclass //I2C_Pattern_Detector_scoreboard_class

endpackage //I2C_Pattern_Detector_scoreboard_pkg
```

3.2.15 I2C_ Pattern_Detector _monitor

```
////////////////////////////////////////////////////////////////////////////////
//Name: Abdelrahman Mohamed Ragab
// Module-Name: I2C_Pattern_Detector_monitor
////////////////////////////////////////////////////////////////////////////////

package I2C_Pattern_Detector_monitor_pkg;

    // Import UVM and related packages
    import uvm_pkg::*;
    import I2C_Pattern_Detector_sequence_item_pkg::*;
    import i2c_config_package::*;
    `include "uvm_macros.svh";

    // I2C_Pattern_Detector TX monitor class extending uvm_monitor
    class I2C_Pattern_Detector_monitor_class extends uvm_monitor;

        // Factory registration
        `uvm_component_utils(I2C_Pattern_Detector_monitor_class);

        // Virtual interface for monitoring OUT signals
        virtual I2C_interface monitor_interface;
        i2c_config_class monitor_config_db;
        // Sequence item to capture monitored values
        I2C_Pattern_Detector_sequence_item_class monitor_sequence_item;

        logic past_scl;

        // Analysis port to send sequence items to subscribers
        uvm_analysis_port #(I2C_Pattern_Detector_sequence_item_class) monitor_analysis_port;

        // Constructor
        function new(string name = "I2C_Pattern_Detector_monitor_class" , uvm_component parent = null);
            super.new(name,parent);
        endfunction //new()

        // Build phase: create analysis port
        function void build_phase (uvm_phase phase);
            super.build_phase(phase);
            uvm_config_db#(i2c_config_class)::get(this, "", "cfg", monitor_config_db);
            monitor_analysis_port = new("monitor_analysis_port" , this);
        endfunction

        logic start_state;
        logic stop_state;
        logic broke_early;
        logic start_checking;
        logic [ADDRESS_WIDTH:0] control_signal [$];
```

```
// Run phase: sample interface signals and send sequence items
task run_phase (uvm_phase phase);
    super.run_phase(phase);
    fork
        // Thread 1: Main monitoring loop
        begin
            forever begin
                monitor_sequence_item = I2C_Pattern_Detector_sequence_item_class::type_id::create("monitor_sequence_item");
                if (monitor_config_db.is_top) @(negedge monitor_interface.clk);
                else #10;
                // Capture interface signals into sequence item
                if (!monitor_interface.rst_n) begin
                    start_checking = 1;
                    control_signal.delete();
                end

                if (!monitor_interface.SCL_in || !monitor_interface.rst_n) begin
                    monitor_sequence_item.start_state_ref = 0;
                    monitor_sequence_item.stop_state_ref = 0;
                    start_state = 0;
                    stop_state = 0;
                end else if (start_state || stop_state) begin
                    if (start_state) begin
                        monitor_sequence_item.start_state_ref = 1;
                        monitor_sequence_item.stop_state_ref = stop_state;
                    end
                    if (stop_state) begin
                        monitor_sequence_item.stop_state_ref = 1;
                        monitor_sequence_item.start_state_ref = start_state;
                    end
                end else begin
                    monitor_sequence_item.start_state_ref = start_state;
                    monitor_sequence_item.stop_state_ref = stop_state;
                end

                monitor_sequence_item.wr_enable = monitor_interface.write_enable;
                monitor_sequence_item.rd_enable = monitor_interface.read_enable;
                monitor_sequence_item.start_state = monitor_interface.start_state;
                monitor_sequence_item.stop_state = monitor_interface.stop_state;
                monitor_sequence_item.SCL_in = monitor_interface.SCL_in;
                monitor_sequence_item.SDA_in = monitor_interface.SDA_in;
                monitor_sequence_item.start_checking = start_checking;
                monitor_sequence_item.rst_n = monitor_interface.rst_n;
                monitor_sequence_item.control_signal = control_signal;
                monitor_sequence_item.self_address_generated = monitor_interface.self_address_generated;
                $display("%0b , %0b",  monitor_sequence_item.start_state, monitor_sequence_item.stop_state);
                monitor_analysis_port.write(monitor_sequence_item);
            end
        end

        begin
            // Thread 2: Sampling the SDA_in signal
            forever begin
                @(posedge start_state);
                if (!monitor_interface.self_address_generated) begin
                    control_signal.delete();
                    repeat (8) begin
                        start_checking = 0;
                        @(posedge monitor_interface.SCL_in or negedge monitor_interface.rst_n);
                        control_signal.push_back(monitor_interface.SDA_in);
                        if (!monitor_interface.rst_n) begin
                            broke_early = 1;
                            control_signal.delete();
                            break;
                        end
                    end
                    if (!broke_early) begin
                        @(posedge monitor_interface.SCL_in);
                    end else begin
                        broke_early = 0;
                    end
                    start_checking = 1;
                end
            end
        end
```

```
                // Thread 3: Negedge detection on SDA_in
                forever begin
                    @(negedge monitor_interface.SDA_in);
                    if (monitor_interface.SCL_in) begin
                        start_state = 1;
                    end
                end
            end

            begin
                // Thread 4: Positive detection on SDA_in
                forever begin
                    @(posedge monitor_interface.SDA_in);
                    if (monitor_interface.SCL_in) begin
                        stop_state = 1;
                    end
                end
            end
        join_none
    endtask //run_phase

endclass //I2C_Pattern_Detector_monitor_class extends uvm_monitor

endpackage //I2C_Pattern_Detector_monitor_pkg
```

3.2.16 I2C_Pattern_Detector _driver

```
////////////////////////////////////////////////////////////////////////////////
//Name: Abdelrahman Mohamed Ragab
// Module-Name: I2C_Pattern_Detector_driver
////////////////////////////////////////////////////////////////////////////////

package I2C_Pattern_Detector_driver_pkg;

    // Import UVM and related packages
    import uvm_pkg::*;
    import I2C_Pattern_Detector_sequence_item_pkg::*;
    `include "uvm_macros.svh";

    // I2C_Pattern_Detector TX driver class extending uvm_driver
    class I2C_Pattern_Detector_driver_class extends uvm_driver #(I2C_Pattern_Detector_sequence_item_class);

        // Factory registration
        `uvm_component_utils(I2C_Pattern_Detector_driver_class);

        // Sequence item and interface declarations
        I2C_Pattern_Detector_sequence_item_class driver_sequence_item;
        virtual I2C_interface driver_interface;

        // Constructor
        function new (string name = "uvm_driver_class" , uvm_component parent = null);
            super.new(name,parent);
        endfunction

        // Build phase
        function void build_phase (uvm_phase phase);
            super.build_phase(phase);
        endfunction //build_phase

        // Run phase: drive interface signals
        task run_phase (uvm_phase phase);
            super.run_phase(phase);
            driver_interface.rst_n <= 0;
            #10;
            forever begin
                driver_sequence_item = I2C_Pattern_Detector_sequence_item_class::type_id::create("driver_sequence_item");
                seq_item_port.get_next_item(driver_sequence_item);

                driver_interface.SCL_in <= driver_sequence_item.SCL_in;
                driver_interface.SDA_in <= driver_sequence_item.SDA_in;
                driver_interface.rst_n <= driver_sequence_item.rst_n;
                driver_interface.self_address_generated <= driver_sequence_item.self_address_generated;
                #10;
                seq_item_port.item_done();
            end
        endtask //run_phase

    endclass //I2C_Pattern_Detector_driver_class

endpackage //I2C_Pattern_Detector_driver_pkg
```

3.2.17 I2C_ Pattern_Detector _agent

```
////////////////////////////////////////////////////////////////////////////////
//Name: Abdelrahman Mohamed Ragab
// Module-Name: I2C_Pattern_Detector_agent
////////////////////////////////////////////////////////////////////////////////

package I2C_Pattern_Detector_agent_pkg;

    // Import UVM and related packages
    import uvm_pkg::*;
    import I2C_Pattern_Detector_sequencer_pkg::*;
    import I2C_Pattern_Detector_monitor_pkg::*;
    import I2C_Pattern_Detector_driver_pkg::*;
    import I2C_Pattern_Detector_sequence_item_pkg::*;
    import i2c_config_package::*;
    `include "uvm_macros.svh";

    // I2C_Pattern_Detector top TX agent class extending uvm_agent
    class I2C_Pattern_Detector_agent_class extends uvm_agent;

        // Factory registration
        `uvm_component_utils(I2C_Pattern_Detector_agent_class);

        // Component declarations
        I2C_Pattern_Detector_driver_class          agent_driver;
        I2C_Pattern_Detector_monitor_class         agent_monitor;
        I2C_Pattern_Detector_sequence_item_class   agent_sequence_item;
        I2C_Pattern_Detector_sequencer_class       agent_sequencer;
        i2c_config_class                           agent_config_db;

        // Analysis port declaration
        uvm_analysis_port#(I2C_Pattern_Detector_sequence_item_class) agent_analysis_port;

        // Constructor
        function new(string name = "I2C_Pattern_Detector_agent_class", uvm_component parent = null);
            super.new(name, parent);
        endfunction // new()

        // Build phase: create components
        function void build_phase(uvm_phase phase);
            super.build_phase(phase);

            uvm_config_db#(i2c_config_class)::get(this, "", "cfg", agent_config_db);

            if (agent_config_db.is_active == UVM_ACTIVE) begin
                agent_driver        = I2C_Pattern_Detector_driver_class::type_id::create("agent_driver", this);
            end
            agent_monitor       = I2C_Pattern_Detector_monitor_class::type_id::create("agent_monitor", this);
            agent_sequencer     = I2C_Pattern_Detector_sequencer_class::type_id::create("agent_sequencer", this);

            agent_analysis_port = new("agent_analysis_port", this);
        endfunction // build_phase()

        // Connect phase: connect TLM ports and interfaces
        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            if (agent_config_db.is_active == UVM_ACTIVE) begin
                agent_driver.seq_item_port.connect(agent_sequencer.seq_item_export);
                agent_driver.driver_interface = agent_config_db.vif;
            end

            agent_monitor.monitor_interface = agent_config_db.vif;
            agent_monitor.monitor_analysis_port.connect(agent_analysis_port);
        endfunction // connect_phase()

    endclass // I2C_Pattern_Detector_agent_class

endpackage // I2C_Pattern_Detector_agent_pkg
```

3.2.18 I2C_ Pattern_Detector _environment

```
////////////////////////////////////////////////////////////////////////////////
//Name: Abdelrahman Mohamed Ragab
// Module-Name: I2C_Pattern_Detector_enviroment
////////////////////////////////////////////////////////////////////////////////

package I2C_Pattern_Detector_enviroment_pkg;

    // Import UVM and related packages
    import uvm_pkg::*;
    import I2C_Pattern_Detector_agent_pkg::*;
    import I2C_Pattern_Detector_scoreboard_pkg::*;
    import i2c_config_package::*;
    `include "uvm_macros.svh";

    // I2C_Pattern_Detector TX environment class extending uvm_env
    class I2C_Pattern_Detector_enviroment_class extends uvm_env;

        // Factory registration
        `uvm_component_utils(I2C_Pattern_Detector_enviroment_class);

        // Agent, scoreboard, and coverage declarations
        I2C_Pattern_Detector_agent_class enviroment_agent;
        I2C_Pattern_Detector_scoreboard_class enviroment_scoreboard;
        i2c_config_class enviroment_config_db;

        // Constructor
        function new(string name = "I2C_Pattern_Detector_enviroment_class" , uvm_component parent = null);
            super.new(name,parent);
        endfunction //new()

        // Build phase
        function void build_phase (uvm_phase phase);
            super.build_phase(phase);
            enviroment_agent = I2C_Pattern_Detector_agent_class::type_id::create("enviroment_agent" , this);
            enviroment_scoreboard = I2C_Pattern_Detector_scoreboard_class::type_id::create("enviroment_scoreboard" , this);
            enviroment_config_db = i2c_config_class::type_id::create("enviroment_config_db");

            uvm_config_db#(virtual I2C_interface)::get(this, "", "int", enviroment_config_db.vif);
            enviroment_config_db.is_active = UVM_ACTIVE;
            enviroment_config_db.i2c_address = 'h10;
            enviroment_config_db.is_top = 0;
            uvm_config_db#(i2c_config_class)::set(this, "*", "cfg", enviroment_config_db);
        endfunction //build_phase()

        // Connect phase
        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            enviroment_agent.agent_analysis_port.connect(enviroment_scoreboard.scoreboard_analysis_export);
        endfunction //connect_phase()

    endclass //I2C_Pattern_Detector_enviroment_class

endpackage //I2C_Pattern_Detector_enviroment_pkg
```

3.2.19 I2C_ Pattern_Detector _test

```
////////////////////////////////////////////////////////////////////////////////
//Name: Abdelrahman Mohamed Ragab
// Module-Name: I2C_Pattern_Detector_test
////////////////////////////////////////////////////////////////////////////////

package I2C_Pattern_Detector_test_pkg;

    // Import UVM and related packages
    import uvm_pkg::*;
    import I2C_Pattern_Detector_enviroment_pkg::*;
    import I2C_Pattern_Detector_randomized_sequence_pkg::*;
    import I2C_Pattern_Detector_virtual_sequence_pkg::*;
    `include "uvm_macros.svh";

    // I2C_Pattern_Detector TX test class extending uvm_test
    class I2C_Pattern_Detector_test_class extends uvm_test;

        // Factory registration
        `uvm_component_utils(I2C_Pattern_Detector_test_class);

        // Environment and sequence declarations
        I2C_Pattern_Detector_enviroment_class test_enviroment;
        I2C_Pattern_Detector_randomized_sequence_class test_randomized_sequence;
        I2C_Pattern_Detector_virtual_sequence_class virtual_sequence;

        // Constructor
        function new(string name = "I2C_Pattern_Detector_test_class" , uvm_component parent = null);
            super.new(name,parent);
        endfunction //new()

        // Build phase
        function void build_phase (uvm_phase phase);
            super.build_phase(phase);
            test_enviroment = I2C_Pattern_Detector_enviroment_class::type_id::create("test_enviroment" ,this);
            test_randomized_sequence = I2C_Pattern_Detector_randomized_sequence_class::type_id::create("test_randomized_sequence");
            virtual_sequence = I2C_Pattern_Detector_virtual_sequence_class::type_id::create("virtual_sequence");
        endfunction //build_phase()

        // Run phase
        task run_phase (uvm_phase phase);
            super.run_phase(phase);
            virtual_sequence.vitual_sequence_sequencer = test_enviroment.enviroment_agent.agent_sequencer;
            phase.raise_objection(this);
            test_randomized_sequence.start(test_enviroment.enviroment_agent.agent_sequencer);
            `uvm_info("run_phase","finish first test",UVM_MEDIUM);
            virtual_sequence.start(null);
            `uvm_info("run_phase","finish second test",UVM_MEDIUM);
            phase.drop_objection(this);
        endtask //run_phase()

    endclass //I2C_Pattern_Detector_test_class

endpackage
```

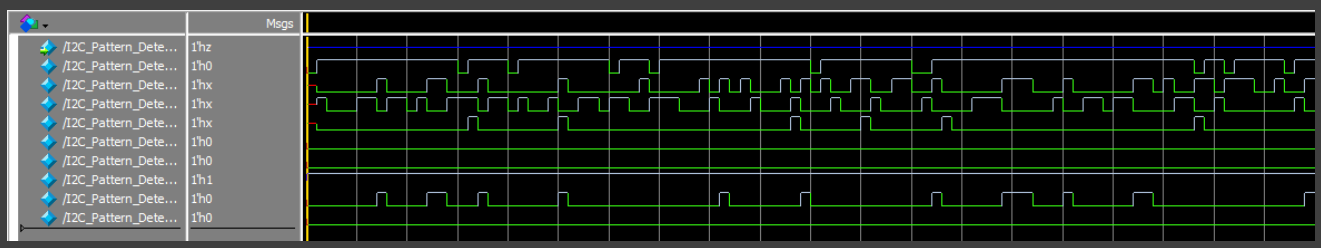
3.2.20 I2C_ Pattern_Detector _assertions

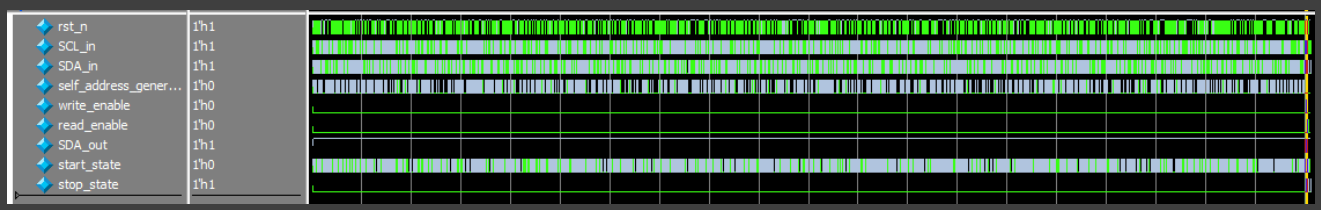
```
module I2C_Pattern_Detector_assertion #(
    parameter ADDRESS = 7'h10,
    parameter ADDRESS_WIDTH = 7
) (
    input logic SCL_in,
    input logic SDA_in,
    input logic rst_n,
    input logic self_address_generated,
    input logic SDA_out,
    input logic wr_enable,
    input logic rd_enable
);

logic neg_edge;
logic pos_edge;

    always_comb begin :reset_assertion
        if (!rst_n) begin
            assert final (!wr_enable
                    && !rd_enable
                    && SDA_out)
            else $display("error: reset state");
        end
    end

    sequence read_write_enable_sequence;
        I2C_Pattern_Detector.CS == 2'b01
        && I2C_Pattern_Detector.counter == 8;
    endsequence

    sequence detect_NACK_sequence;
        I2C_Pattern_Detector.CS == 2'b10 &&
        I2C_Pattern_Detector.counter == 8 &&
        !I2C_Pattern_Detector.first_time;
    endsequence

    sequence ACK_sequence;
        I2C_Pattern_Detector.CS == 2'b01
        && I2C_Pattern_Detector.counter == 8;
    endsequence

    property write_enable_property;
        @(posedge SCL_in) disable iff (!rst_n)
        read_write_enable_sequence |=> if (I2C_Pattern_Detector.control_signal[7:1] == 7'h10)
            if (!I2C_Pattern_Detector.control_signal[0]) wr_enable == 0
            else wr_enable == 1
        else wr_enable == 0;
    endproperty

    property read_enable_property;
        @(posedge SCL_in) disable iff (!rst_n)
        read_write_enable_sequence |=> if (I2C_Pattern_Detector.control_signal[7:1] == 7'h10)
            if (!I2C_Pattern_Detector.control_signal[0]) rd_enable == 1
            else rd_enable == 0
        else rd_enable == 0;
    endproperty

    property ACK_property;
        @(negedge SCL_in) disable iff (!rst_n)
        ACK_sequence |=> if (I2C_Pattern_Detector.control_signal[7:1] == 7'h10) SDA_out == 0
        else SDA_out == 1;
    endproperty

    assert property (write_enable_property)
    else $display("error: write operation");
    cover property (write_enable_property);

    assert property (read_enable_property)
    else $display("error: read operation");
    cover property (read_enable_property);

    assert property (ACK_property)
    else $display("error: ACK operation");
    cover property (ACK_property);

endmodule
```

3.2.21 I2C_ Pattern_Detector _top

```
////////////////////////////////////////////////////////////////////////////////
//Name: Abdelrahman Mohamed Ragab
// Module-Name: I2C_Pattern_Detector_top
////////////////////////////////////////////////////////////////////////////////

import uvm_pkg::*;
import I2C_Pattern_Detector_test_pkg::*;
`include "uvm_macros.svh";

module I2C_Pattern_Detector_top ();

I2C_interface I2C_Pattern_Detector_interface ();

I2C_Pattern_Detector I2C_Pattern_Detector (
    .SCL_in(I2C_Pattern_Detector_interface.SCL_in),
    .SDA_in(I2C_Pattern_Detector_interface.SDA_in),
    .rst_n(I2C_Pattern_Detector_interface.rst_n),
    .self_address_generated(I2C_Pattern_Detector_interface.self_address_generated),
    .wr_enable(I2C_Pattern_Detector_interface.write_enable),
    .SDA_out(I2C_Pattern_Detector_interface.SDA_out),
    .rd_enable(I2C_Pattern_Detector_interface.read_enable)
);

bind I2C_Pattern_Detector I2C_Pattern_Detector_assertion I2C_Pattern_Detector_assertion_inst (
    .SCL_in(I2C_Pattern_Detector_interface.SCL_in),
    .SDA_in(I2C_Pattern_Detector_interface.SDA_in),
    .rst_n(I2C_Pattern_Detector_interface.rst_n),
    .self_address_generated(I2C_Pattern_Detector_interface.self_address_generated),
    .wr_enable(I2C_Pattern_Detector_interface.write_enable),
    .SDA_out(I2C_Pattern_Detector_interface.SDA_out),
    .rd_enable(I2C_Pattern_Detector_interface.read_enable)
);

assign I2C_Pattern_Detector_interface.start_state = I2C_Pattern_Detector.start_state;
assign I2C_Pattern_Detector_interface.stop_state = I2C_Pattern_Detector.stop_state;

initial begin
    uvm_config_db #(virtual I2C_interface)::set(null,"*","int",I2C_Pattern_Detector_interface);
    run_test("I2C_Pattern_Detector_test_class");
end


endmodule
```

3.2.22 Simulation Results






```
# UVM_INFO I2C_Pattern_Detector_scoreboard.sv(157) @ 201320: uvm_test_top.test_enviroment.enviroment_scoreboard [report_phase] correct case = 6276 , wrong case = 0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :26429
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
```

As we can see there is no checker errors

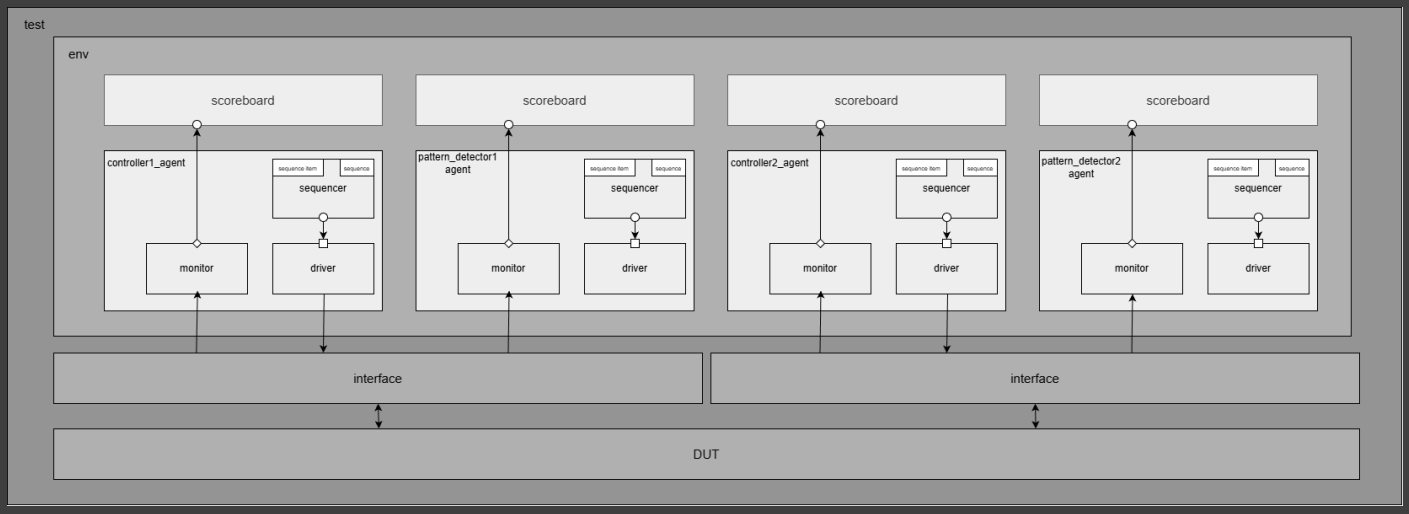
/I2C_Pattern_Detector_randomized_sequence_pkg::I2C_Pattern_Detector_randomized_sequence_class::body/#ublk#21...	Immediate	SVA	on	0	1
/I2C_Pattern_Detector_top/I2C_Pattern_Detector/I2C_Pattern_Detector_assertion_inst/assert__write_enable_property	Concurrent	SVA	on	0	1
/I2C_Pattern_Detector_top/I2C_Pattern_Detector/I2C_Pattern_Detector_assertion_inst/assert__read_enable_property	Concurrent	SVA	on	0	1
/I2C_Pattern_Detector_top/I2C_Pattern_Detector/I2C_Pattern_Detector_assertion_inst/assert__ACK_property	Concurrent	SVA	on	0	1
/I2C_Pattern_Detector_top/I2C_Pattern_Detector/I2C_Pattern_Detector_assertion_inst/reset_assertion/#ublk#2590643...	Immediate	SVA	on	0	1

/I2C_Pattern_Detector_top/I2C_Pattern_Detector/I2C_Pattern_Detector_assertion_inst/cover__ACK_property	SVA	✓	Off	99	1	Unli...	1	100%
/I2C_Pattern_Detector_top/I2C_Pattern_Detector/I2C_Pattern_Detector_assertion_inst/cover__read_enable_property	SVA	✓	Off	90	1	Unli...	1	100%
/I2C_Pattern_Detector_top/I2C_Pattern_Detector/I2C_Pattern_Detector_assertion_inst/cover__write_enable_property	SVA	✓	Off	90	1	Unli...	1	100%

As we can see there is no assertion errors

3.3 I2C_top module System-Level UVM Environment

3.3.1 I2C_top_Verification Plan and Environment



label	description	stimulus generated	checkers
randomization sequence 1	In this sequence we will randomize the first control module to make sure that the overall functionality of the top design and the connection between the two I2C modules is correct	randomization for all the inputs with constraining the reset to be on 5% of the time and no restert request for 95% of the time	we check the functionality of the outputs using checkers comparing the output with the reference output from golden model
randomization sequence 2	In this sequence we will randomize the second control module to make sure that the overall functionality of the top design and the connection between the two I2C modules is correct	randomization for all the inputs with constraining the reset to be on 5% of the time and no restert request for 95% of the time	we check the functionality of the outputs using checkers comparing the output with the reference output from golden model
start read sequence 1	this is a directed seequence to check the starting of a read transaction and detection of the target in the same transaction	directed	we check the functionality of the outputs using checkers comparing the output with the reference output from golden model we use an assertion to check the detection of the address and operation in the target
start write sequence 1	this is a directed seequence to check the starting of a write transaction and detection of the target in the same transaction	directed	we check the functionality of the outputs using checkers comparing the output with the reference output from golden model we use an assertion to check the detection of the address and operation in the target
NACK feature 1	this is a directed sequence where we check the target bahavior with the error respond and NACK response	directed	we check the functionality of the outputs using checkers comparing the output with the reference output from golden model we use an assertion to check the generation of the NACK.
valid feaure 1	this is a directed sequence where we held the valid to low to check the termination of the transaction based on the valid signal	directed	we check the functionality of the outputs using checkers comparing the output with the reference output from golden model
restart feature 1	this is a directed sequence where we held the restart request to 1 to check the restarting feature of the control module	directed	we check the functionality of the outputs using checkers comparing the output with the reference output from golden model
start read sequence 2	this is a directed seequence to check the starting of a read transaction and detection of the target in the same transaction	directed	we check the functionality of the outputs using checkers comparing the output with the reference output from golden model we use an assertion to check the detection of the address and operation in the target
start write sequence 2	this is a directed seequence to check the starting of a write transaction and detection of the target in the same transaction	directed	we check the functionality of the outputs using checkers comparing the output with the reference output from golden model we use an assertion to check the detection of the address and operation in the target
NACK feature 2	this is a directed sequence where we check the target behavior with the error respond and NACK response	directed	we check the functionality of the outputs using checkers comparing the output with the reference output from golden model we use an assertion to check the generation of the NACK.
valid feaure 2	this is a directed sequence where we held the valid to low to check the termination of the transaction based on the valid signal	directed	we check the functionality of the outputs using checkers comparing the output with the reference output from golden model
restart feature 2	this is a directed sequence where we held the restart request to 1 to check the restarting feature of the control module	directed	we check the functionality of the outputs using checkers comparing the output with the reference output from golden model
data transmition feature	we check the data transmition in all previous condition between the first I2C module and the I2C target module	directed / randomized	we check the functionality of the outputs using an assertion to check the received data in the I2C target module

3.3.2 I2C_virtual_sequence

```
package I2C_virtual_sequence_pkg;

// Import UVM and related packages
    import uvm_pkg::*;
    import I2C_Pattern_Detector_sequence_item_pkg::*;
    import I2C_Controller_sequence_item_pkg::*;
    import I2C_Controller_directed_restart_sequence_pkg::*;
    import I2C_Controller_directed_start_transaction_read_sequence_pkg::*;
    import I2C_Controller_directed_start_transaction_write_sequence_pkg::*;
    import I2C_Controller_directed_stop_SDA_sequence_pkg::*;
    import I2C_Controller_directed_stop_valid_sequence_pkg::*;
    import I2C_Controller_randomized_sequence_pkg::*;
    import I2C_Controller_sequencer_pkg::*;
    `include "uvm_macros.svh";

    // I2C Pattern Detector virtual sequence class extending uvm_sequence
    class I2C_virtual_sequence_class extends uvm_sequence #(I2C_Controller_sequence_item_class);

        // Factory registration
        `uvm_object_utils(I2C_virtual_sequence_class);

        virtual I2C_interface virtual_interface1;
        virtual I2C_interface virtual_interface2;

        // Sequence item declaration
        I2C_Pattern_Detector_sequence_item_class I2C_Pattern_Detector_virtual_sequence_item;
        I2C_Controller_directed_restart_sequence_class directed_restart_sequence1;
        I2C_Controller_directed_restart_sequence_class directed_restart_sequence2;
        I2C_Controller_directed_start_transaction_read_sequence_class directed_start_read_sequence1;
        I2C_Controller_directed_start_transaction_read_sequence_class directed_start_read_sequence2;
        I2C_Controller_directed_start_transaction_write_sequence_class directed_start_write_sequence1;
        I2C_Controller_directed_start_transaction_write_sequence_class directed_start_write_sequence2;
        I2C_Controller_directed_stop_SDA_sequence_class directed_stop_SDA_sequence1;
        I2C_Controller_directed_stop_SDA_sequence_class directed_stop_SDA_sequence2;
        I2C_Controller_directed_stop_valid_sequence_class directed_stop_valid_sequence1;
        I2C_Controller_directed_stop_valid_sequence_class directed_stop_valid_sequence2;
        I2C_Controller_randomized_sequence_class random_sequence1;
        I2C_Controller_randomized_sequence_class random_sequence2;
        I2C_Controller_sequencer_class vitual_sequence_sequencer1;
        I2C_Controller_sequencer_class vitual_sequence_sequencer2;

        logic break_loop;
        logic random;

        // Constructor
        function new(string name = "I2C_Controller_virtual_sequence_class");
            super.new(name);
        endfunction //new()

        task body();
            `uvm_info(get_type_name(), "virtual_seq: Inside Body", UVM_LOW);
            I2C_Pattern_Detector_virtual_sequence_item = I2C_Pattern_Detector_sequence_item_class::type_id::create("I2C_Pattern_Detector_virtual_sequence_item");
            directed_restart_sequence1 = I2C_Controller_directed_restart_sequence_class::type_id::create("directed_restart_sequence1");
            directed_restart_sequence2 = I2C_Controller_directed_restart_sequence_class::type_id::create("directed_restart_sequence2");
            directed_start_read_sequence1 = I2C_Controller_directed_start_transaction_read_sequence_class::type_id::create("directed_start_read_sequence1");
            directed_start_read_sequence2 = I2C_Controller_directed_start_transaction_read_sequence_class::type_id::create("directed_start_read_sequence2");
            directed_start_write_sequence1 = I2C_Controller_directed_start_transaction_write_sequence_class::type_id::create("directed_start_write_sequence1");
            directed_start_write_sequence2 = I2C_Controller_directed_start_transaction_write_sequence_class::type_id::create("directed_start_write_sequence2");
            directed_stop_SDA_sequence1 = I2C_Controller_directed_stop_SDA_sequence_class::type_id::create("directed_stop_SDA_sequence1");
            directed_stop_SDA_sequence2 = I2C_Controller_directed_stop_SDA_sequence_class::type_id::create("directed_stop_SDA_sequence2");
            directed_stop_valid_sequence1 = I2C_Controller_directed_stop_valid_sequence_class::type_id::create("directed_stop_valid_sequence1");
            directed_stop_valid_sequence2 = I2C_Controller_directed_stop_valid_sequence_class::type_id::create("directed_stop_valid_sequence2");
            random_sequence1 = I2C_Controller_randomized_sequence_class::type_id::create("random_sequence1");
            random_sequence2 = I2C_Controller_randomized_sequence_class::type_id::create("random_sequence2");

            uvm_config_db#(virtual I2C_interface)::get(m_sequencer, "", "int1", virtual_interface1);
            uvm_config_db#(virtual I2C_interface)::get(m_sequencer, "", "int2", virtual_interface2);

                virtual_interface1.rst_n = 0;
                virtual_interface2.rst_n = 0;
                @(posedge virtual_interface1.clk);
                virtual_interface1.rst_n = 1;
                virtual_interface2.rst_n = 1;

                repeat (1000) begin
                    random_sequence1.start(vitual_sequence_sequencer1);
                    random_sequence2.random = 1;
                    random_sequence2.rst_n_value = random_sequence1.MAIN_sequence_sequence_item.rst_n;
                    random_sequence2.not_valid = 1;
                    random_sequence2.start(vitual_sequence_sequencer2);
                end
                random_sequence2.random = 0;
                random_sequence2.not_valid = 0;
```

```
repeat (1000) begin
    random_sequence2.start(vitual_sequence_sequencer2);
    random_sequence1.random = 1;
    random_sequence1.rst_n_value = random_sequence2.MAIN_sequence_sequence_item.rst_n;
    random_sequence1.not_valid = 1;
    random_sequence1.start(vitual_sequence_sequencer1);
end
random_sequence1.random = 0;
random_sequence1.not_valid = 0;
random_sequence2.random = 0;
random_sequence2.not_valid = 0;

virtual_interface1.rst_n = 0;
virtual_interface2.rst_n = 0;
@(posedge virtual_interface1.clk);
virtual_interface1.rst_n = 1;
virtual_interface2.rst_n = 1;

virtual_interface2.controller_valid_req = 0;

directed_start_read_sequence1.address = 'h10;
directed_start_write_sequence1.address = 'h10;
directed_start_read_sequence2.address = 'h01;
directed_start_write_sequence2.address = 'h01;

virtual_interface1.rst_n = 0;
virtual_interface2.rst_n = 0;
@(posedge virtual_interface1.clk);
virtual_interface1.rst_n = 1;
virtual_interface2.rst_n = 1;
directed_start_write_sequence1.start(vitual_sequence_sequencer1);
`uvm_info("run_phase","finish first directed test",UVM_MEDIUM);
repeat (30) @(posedge virtual_interface1.SCL_in);
repeat (10) begin
    directed_stop_valid_sequence1.start(vitual_sequence_sequencer1);
    fork
        begin
            @(posedge virtual_interface1.SCL_in);
        end
        begin
            repeat (100) @(posedge virtual_interface1.clk);
            break_loop = 1;
        end
    join_any
    if (break_loop) break;
end
disable fork;
break_loop = 0;
`uvm_info("run_phase","finish second directed test",UVM_MEDIUM);
virtual_interface1.rst_n = 0;
virtual_interface2.rst_n = 0;
@(posedge virtual_interface1.clk);
virtual_interface1.rst_n = 1;
virtual_interface2.rst_n = 1;
directed_start_write_sequence1.start(vitual_sequence_sequencer1);
`uvm_info("run_phase","finish forth directed test",UVM_MEDIUM);
repeat (15) @(posedge virtual_interface1.SCL_in);
repeat (8) begin
    directed_stop_SDA_sequence2.start(vitual_sequence_sequencer2);
    fork
        begin
            @(posedge virtual_interface1.SCL_in);
        end
        begin
            repeat (100) @(posedge virtual_interface1.clk);
            break_loop = 1;
        end
    join_any
    if (break_loop) break;
end
```

```
end
disable fork;
break_loop = 0;
`uvm_info("run_phase","finish fifth directed test",UVM_MEDIUM);
virtual_interface1.rst_n = 0;
virtual_interface2.rst_n = 0;
@(posedge virtual_interface1.clk);
virtual_interface1.rst_n = 1;
virtual_interface2.rst_n = 1;
directed_start_read_sequence1.start(vitual_sequence_sequencer1);
`uvm_info("run_phase","finish sixth directed test",UVM_MEDIUM);
repeat (15) @(posedge virtual_interface1.SCL_in);
repeat (8) begin
    directed_restart_sequence1.start(vitual_sequence_sequencer1);
    fork
        begin
            @(posedge virtual_interface1.SCL_in);
        end
        begin
            repeat (100) @(posedge virtual_interface1.clk);
            break_loop = 1;
        end
    join_any
    if (break_loop) break;
end
disable fork;
break_loop = 0;
`uvm_info("run_phase","finish seventh directed test",UVM_MEDIUM);
virtual_interface1.rst_n = 0;
virtual_interface2.rst_n = 0;
@(posedge virtual_interface1.clk);
virtual_interface1.rst_n = 1;
virtual_interface2.rst_n = 1;
directed_start_read_sequence1.start(vitual_sequence_sequencer1);
`uvm_info("run_phase","finish eighth directed test",UVM_MEDIUM);
repeat (15) begin
    directed_stop_valid_sequence1.start(vitual_sequence_sequencer1);
    fork
        begin
            @(posedge virtual_interface1.SCL_in);
        end
        begin
            repeat (100) @(posedge virtual_interface1.clk);
            break_loop = 1;
        end
    join_any
    if (break_loop) break;
end
disable fork;
break_loop = 0;
`uvm_info("run_phase","finish ninth directed test",UVM_MEDIUM);
directed_start_read_sequence1.start(vitual_sequence_sequencer1);
`uvm_info("run_phase","finish tenth directed test",UVM_MEDIUM);
repeat (20) @(posedge virtual_interface1.SCL_in);
repeat (8) begin
    directed_stop_SDA_sequence2.start(vitual_sequence_sequencer2);
    fork
        begin
            @(posedge virtual_interface1.SCL_in);
        end
        begin
            repeat (100) @(posedge virtual_interface1.clk);
            break_loop = 1;
        end
    join_any
    if (break_loop) break;
end
disable fork;
break_loop = 0;
`uvm_info("run_phase","finish eleventh directed test",UVM_MEDIUM);
```

```
virtual_interface1.controller_valid_req = 0;
repeat (8) begin
    fork
        begin
            @(posedge virtual_interface2.SCL_in);
        end
        begin
            repeat (100) @(posedge virtual_interface2.clk);
            break_loop = 1;
        end
    join_any
end
virtual_interface1.error_signal = 0;
virtual_interface1.rst_n = 0;
virtual_interface2.rst_n = 0;
@(posedge virtual_interface1.clk);
virtual_interface1.rst_n = 1;
virtual_interface2.rst_n = 1;
directed_start_write_sequence2.start(vitual_sequence_sequencer2);
`uvm_info("run_phase","finish first directed test",UVM_MEDIUM);
repeat (30) @(posedge virtual_interface2.SCL_in);
repeat (8) begin
    directed_stop_valid_sequence2.start(vitual_sequence_sequencer2);
    fork
        begin
            @(posedge virtual_interface2.SCL_in);
        end
        begin
            repeat (100) @(posedge virtual_interface2.clk);
            break_loop = 1;
        end
    join_any
    if (break_loop) break;
end
disable fork;
break_loop = 0;
`uvm_info("run_phase","finish second directed test",UVM_MEDIUM);
virtual_interface1.rst_n = 0;
virtual_interface2.rst_n = 0;
@(posedge virtual_interface1.clk);
virtual_interface1.rst_n = 1;
virtual_interface2.rst_n = 1;
directed_start_write_sequence2.start(vitual_sequence_sequencer2);
`uvm_info("run_phase","finish forth directed test",UVM_MEDIUM);
repeat (15) @(posedge virtual_interface2.SCL_in);
repeat (8) begin
    directed_stop_SDA_sequence1.start(vitual_sequence_sequencer1);
    fork
        begin
            @(posedge virtual_interface2.SCL_in);
        end
        begin
            repeat (100) @(posedge virtual_interface2.clk);
            break_loop = 1;
        end
    join_any
    if (break_loop) break;
end
disable fork;
break_loop = 0;
`uvm_info("run_phase","finish fifth directed test",UVM_MEDIUM);
virtual_interface1.rst_n = 0;
virtual_interface2.rst_n = 0;
@(posedge virtual_interface1.clk);
virtual_interface1.rst_n = 1;
virtual_interface2.rst_n = 1;
directed_start_read_sequence2.start(vitual_sequence_sequencer2);
`uvm_info("run_phase","finish sixth directed test",UVM_MEDIUM);
repeat (15) @(posedge virtual_interface2.SCL_in);
repeat (8) begin
    directed_restart_sequence2.start(vitual_sequence_sequencer2);
    fork
        begin
            @(posedge virtual_interface2.SCL_in);
        end
        begin
            repeat (100) @(posedge virtual_interface2.clk);
            break_loop = 1;
        end
```

```
        join_any
        if (break_loop) break;
    end
    disable fork;
    break_loop = 0;
    `uvm_info("run_phase","finish seventh directed test",UVM_MEDIUM);
    virtual_interface1.rst_n = 0;
    virtual_interface2.rst_n = 0;
    @(posedge virtual_interface1.clk);
    virtual_interface1.rst_n = 1;
    virtual_interface2.rst_n = 1;
    directed_start_read_sequence2.start(vitual_sequence_sequencer2);
    `uvm_info("run_phase","finish eighth directed test",UVM_MEDIUM);
    repeat (15) begin
        directed_stop_valid_sequence2.start(vitual_sequence_sequencer2);
        fork
            begin
                @(posedge virtual_interface2.SCL_in);
            end
            begin
                repeat (100) @(posedge virtual_interface2.clk);
                break_loop = 1;
            end
        join_any
        if (break_loop) break;
    end
    disable fork;
    break_loop = 0;
    `uvm_info("run_phase","finish ninth directed test",UVM_MEDIUM);
    virtual_interface1.rst_n = 0;
    virtual_interface2.rst_n = 0;
    @(posedge virtual_interface1.clk);
    virtual_interface1.rst_n = 1;
    virtual_interface2.rst_n = 1;
    directed_start_read_sequence2.start(vitual_sequence_sequencer2);
    `uvm_info("run_phase","finish tenth directed test",UVM_MEDIUM);
    repeat (8) @(posedge virtual_interface2.SCL_in);
    repeat (8) begin
        directed_stop_SDA_sequence1.start(vitual_sequence_sequencer1);
        fork
            begin
                @(posedge virtual_interface2.SCL_in);
            end
            begin
                repeat (100) @(posedge virtual_interface2.clk);
                break_loop = 1;
            end
        join_any
        if (break_loop) break;
    end
    disable fork;
    break_loop = 0;
    `uvm_info("run_phase","finish eleventh directed test",UVM_MEDIUM);
endtask
endclass //I2C_Controller_virtual_sequence_class extends uvm_seq

endpackage
```

3.3.3 I2C_environment

```
////////////////////////////////////////////////////////////////////////////////
//Name: Abdelrahman Mohamed Ragab
// Module-Name: I2C_enviroment
////////////////////////////////////////////////////////////////////////////////

package I2C_enviroment_pkg;

    // Import UVM and related packages
    import uvm_pkg::*;
    import I2C_Pattern_Detector_agent_pkg::*;
    import I2C_Controller_agent_pkg::*;
    import I2C_Controller_scoreboard_pkg::*;
    import I2C_Pattern_Detector_scoreboard_pkg::*;
    import i2c_config_package::*;
    `include "uvm_macros.svh";

    // I2C TX environment class extending uvm_env
    class I2C_enviroment_class extends uvm_env;

        // Factory registration
        `uvm_component_utils(I2C_enviroment_class);

        // Agent, scoreboard, and coverage declarations
        I2C_Pattern_Detector_agent_class enviroment_I2C_Pattern_Detector_agent1;
        I2C_Pattern_Detector_agent_class enviroment_I2C_Pattern_Detector_agent2;
        I2C_Controller_agent_class enviroment_I2C_Controller_agent1;
        I2C_Controller_agent_class enviroment_I2C_Controller_agent2;
        I2C_Controller_scoreboard_class enviroment_controller_scoreboard1;
        I2C_Controller_scoreboard_class enviroment_controller_scoreboard2;
        I2C_Pattern_Detector_scoreboard_class enviroment_Pattern_Detector_scoreboard1;
        I2C_Pattern_Detector_scoreboard_class enviroment_Pattern_Detector_scoreboard2;
        i2c_config_class enviroment_config_db1;
        i2c_config_class enviroment_config_db2;
        i2c_config_class enviroment_config_db3;
        i2c_config_class enviroment_config_db4;

        // Constructor
        function new(string name = "I2C_enviroment_class" , uvm_component parent = null);
            super.new(name,parent);
        endfunction //new()

        // Build phase
        function void build_phase (uvm_phase phase);
            super.build_phase(phase);
            enviroment_I2C_Pattern_Detector_agent1 = I2C_Pattern_Detector_agent_class::type_id::create("enviroment_I2C_Pattern_Detector_agent1" , this);
            enviroment_I2C_Pattern_Detector_agent2 = I2C_Pattern_Detector_agent_class::type_id::create("enviroment_I2C_Pattern_Detector_agent2" , this);
            enviroment_I2C_Controller_agent1 = I2C_Controller_agent_class::type_id::create("enviroment_I2C_Controller_agent1" , this);
            enviroment_I2C_Controller_agent2 = I2C_Controller_agent_class::type_id::create("enviroment_I2C_Controller_agent2" , this);
            enviroment_controller_scoreboard1 = I2C_Controller_scoreboard_class::type_id::create("enviroment_controller_scoreboard1" , this);
            enviroment_controller_scoreboard2 = I2C_Controller_scoreboard_class::type_id::create("enviroment_controller_scoreboard2" , this);
            enviroment_Pattern_Detector_scoreboard1 = I2C_Pattern_Detector_scoreboard_class::type_id::create("enviroment_Pattern_Detector_scoreboard1" , this);
            enviroment_Pattern_Detector_scoreboard2 = I2C_Pattern_Detector_scoreboard_class::type_id::create("enviroment_Pattern_Detector_scoreboard2" , this);
            enviroment_config_db1 = i2c_config_class::type_id::create("enviroment_config_db1");
            enviroment_config_db2 = i2c_config_class::type_id::create("enviroment_config_db2");
            enviroment_config_db3 = i2c_config_class::type_id::create("enviroment_config_db3");
            enviroment_config_db4 = i2c_config_class::type_id::create("enviroment_config_db4");

            uvm_config_db#(virtual I2C_interface)::get(this, "", "int1", enviroment_config_db1.vif);
            uvm_config_db#(virtual I2C_interface)::get(this, "", "int1", enviroment_config_db2.vif);
            enviroment_config_db1.is_active = UVM_ACTIVE;
            enviroment_config_db1.i2c_address = 'h01;
            enviroment_config_db1.is_top = 1;
            enviroment_config_db2.is_active = UVM_PASSIVE;
            enviroment_config_db2.i2c_address = 'h01;
            enviroment_config_db2.is_top = 1;
            uvm_config_db#(i2c_config_class)::set(this, "enviroment_I2C_Controller_agent1", "cfg", enviroment_config_db1);
            uvm_config_db#(i2c_config_class)::set(this, "enviroment_I2C_Controller_agent1.agent_driver", "cfg", enviroment_config_db1);
            uvm_config_db#(i2c_config_class)::set(this, "enviroment_controller_scoreboard1", "cfg", enviroment_config_db1);
            uvm_config_db#(i2c_config_class)::set(this, "enviroment_I2C_Pattern_Detector_agent1", "cfg", enviroment_config_db2);
            uvm_config_db#(i2c_config_class)::set(this, "enviroment_Pattern_Detector_scoreboard1", "cfg", enviroment_config_db2);
            uvm_config_db#(i2c_config_class)::set(this, "enviroment_I2C_Pattern_Detector_agent1.agent_monitor", "cfg", enviroment_config_db2);

            uvm_config_db#(virtual I2C_interface)::get(this, "", "int2", enviroment_config_db3.vif);
            uvm_config_db#(virtual I2C_interface)::get(this, "", "int2", enviroment_config_db4.vif);
            enviroment_config_db3.is_active = UVM_ACTIVE;
            enviroment_config_db3.i2c_address = 'h10;
            enviroment_config_db3.is_top = 1;
            enviroment_config_db4.is_active = UVM_PASSIVE;
            enviroment_config_db4.i2c_address = 'h10;
            enviroment_config_db4.is_top = 1;
            uvm_config_db#(i2c_config_class)::set(this, "enviroment_I2C_Controller_agent2", "cfg", enviroment_config_db3);
            uvm_config_db#(i2c_config_class)::set(this, "enviroment_I2C_Controller_agent2.agent_driver", "cfg", enviroment_config_db3);
            uvm_config_db#(i2c_config_class)::set(this, "enviroment_controller_scoreboard2", "cfg", enviroment_config_db3);
            uvm_config_db#(i2c_config_class)::set(this, "enviroment_I2C_Pattern_Detector_agent2", "cfg", enviroment_config_db4);
            uvm_config_db#(i2c_config_class)::set(this, "enviroment_Pattern_Detector_scoreboard2", "cfg", enviroment_config_db4);
            uvm_config_db#(i2c_config_class)::set(this, "enviroment_I2C_Pattern_Detector_agent2.agent_monitor", "cfg", enviroment_config_db4);
        endfunction //build_phase()
```

3.3.4 I2C_test

```
////////////////////////////////////////////////////////////////////////////////
//Name: Abdelrahman Mohamed Ragab
// Module-Name: I2C_test
////////////////////////////////////////////////////////////////////////////////

package I2C_test_pkg;

    // Import UVM and related packages
    import uvm_pkg::*;
    import I2C_enviroment_pkg::*;
    import I2C_virtual_sequence_pkg::*;
    `include "uvm_macros.svh";

    // I2C TX test class extending uvm_test
    class I2C_test_class extends uvm_test;

        // Factory registration
        `uvm_component_utils(I2C_test_class);

        // Environment and sequence declarations
        I2C_enviroment_class test_enviroment;
        I2C_virtual_sequence_class virtual_sequence;

        // Constructor
        function new(string name = "I2C_test_class" , uvm_component parent = null);
            super.new(name,parent);
        endfunction //new()

        // Build phase
        function void build_phase (uvm_phase phase);
            super.build_phase(phase);
            test_enviroment = I2C_enviroment_class::type_id::create("test_enviroment" ,this);
            virtual_sequence = I2C_virtual_sequence_class::type_id::create("virtual_sequence");
        endfunction //build_phase()

        // Run phase
        task run_phase (uvm_phase phase);
            super.run_phase(phase);
            phase.raise_objection(this);
            virtual_sequence.vitual_sequence_sequencer1 = test_enviroment.enviroment_I2C_Controller_agent1.agent_sequencer;
            virtual_sequence.vitual_sequence_sequencer2 = test_enviroment.enviroment_I2C_Controller_agent2.agent_sequencer;
            virtual_sequence.start(null);
            `uvm_info("run_phase","finish first test",UVM_MEDIUM);
            phase.drop_objection(this);
        endtask //run_phase()

    endclass //I2C_test_class

endpackage
```

3.3.5 I2C_assertion1

```
module I2C_assertion1 #(
    parameter DATA_WIDTH = 8,
    parameter ADDRESS = 7'h10,
    parameter ADDRESS_WIDTH = 7
) (
    input logic clk,
    input logic SCL_in,
    input logic SDA_in,
    input logic rst_n,
    input logic [DATA_WIDTH-1:0] controller_data_req,
    input logic self_address_generated,
    input logic SDA_out,
    input logic wr_enable,
    input logic rd_enable,
    input logic [DATA_WIDTH-1:0] controller_data_rsp
);

logic neg_edge;
logic pos_edge;

    always_comb begin :reset_assertion
        if (!rst_n) begin
            assert final (!wr_enable
                    && !rd_enable
                    && SDA_out)
            else $display("error: reset state");
        end
    end

    sequence read_write_enable_sequence;
        I2C_top_design_inst.I2C1.I2C_Pattern_Detector_inst.CS == 2'b01
        && I2C_top_design_inst.I2C1.I2C_Pattern_Detector_inst.counter == 8;
    endsequence

    sequence detect_NACK_sequence;
        I2C_top_design_inst.I2C1.I2C_Pattern_Detector_inst.CS == 2'b10 &&
        I2C_top_design_inst.I2C1.I2C_Pattern_Detector_inst.counter == 8 &&
        !I2C_top_design_inst.I2C1.I2C_Pattern_Detector_inst.first_time;
    endsequence

    sequence ACK_sequence;
        I2C_top_design_inst.I2C1.I2C_Pattern_Detector_inst.CS == 2'b01
        && I2C_top_design_inst.I2C1.I2C_Pattern_Detector_inst.counter == 8;
    endsequence

    sequence signal_recieved_check_sequence;
        I2C_top_design_inst.I2C2.I2C_Controller_inst.CS == 2'b11
        && rd_enable && I2C_top_design_inst.I2C2.I2C_Controller_inst.counter == 8
        && !SDA_in;
    endsequence

    property write_enable_property;
        @(posedge SCL_in) disable iff (!rst_n)
        read_write_enable_sequence |=> if (I2C_top_design_inst.I2C1.I2C_Pattern_Detector_inst.control_signal[7:1] == 7'h01)
            if (!I2C_top_design_inst.I2C1.I2C_Pattern_Detector_inst.control_signal[0]) wr_enable == 0
            else wr_enable == 1
        else wr_enable == 0;
    endproperty

    property read_enable_property;
        @(posedge SCL_in) disable iff (!rst_n)
        read_write_enable_sequence |=> if (I2C_top_design_inst.I2C1.I2C_Pattern_Detector_inst.control_signal[7:1] == 7'h01)
            if (!I2C_top_design_inst.I2C1.I2C_Pattern_Detector_inst.control_signal[0]) rd_enable == 1
            else rd_enable == 0
        else rd_enable == 0;
    endproperty

    property ACK_property;
        @(negedge SCL_in) disable iff (!rst_n)
        ACK_sequence |=> if (I2C_top_design_inst.I2C1.I2C_Pattern_Detector_inst.control_signal[7:1] == 7'h01) SDA_out == 0
        else SDA_out == 1;
    endproperty

    property signal_recieved_check_property;
        @(posedge clk) disable iff (!rst_n)
        signal_recieved_check_sequence |=> controller_data_rsp == $past(controller_data_req);
    endproperty
```

```
    assert property (write_enable_property)
    else $display("error: write operation");
    cover property (write_enable_property);

    assert property (read_enable_property)
    else $display("error: read operation");
    cover property (read_enable_property);

    assert property (ACK_property)
    else $display("error: ACK operation");
    cover property (ACK_property);

    assert property (signal_recieved_check_property)
    else $display("error: recieve operation");
    cover property (signal_recieved_check_property);

endmodule
```

3.3.6 I2C_assertion2

```
module I2C_assertion2 #(
    parameter DATA_WIDTH = 8,
    parameter ADDRESS = 7'h10,
    parameter ADDRESS_WIDTH = 7
) (
    input logic clk,
    input logic SCL_in,
    input logic SDA_in,
    input logic rst_n,
    input logic [DATA_WIDTH-1:0] controller_data_req,
    input logic self_address_generated,
    input logic SDA_out,
    input logic wr_enable,
    input logic rd_enable,
    input logic [DATA_WIDTH-1:0] controller_data_rsp
);

logic neg_edge;
logic pos_edge;

    always_comb begin :reset_assertion
        if (!rst_n) begin
            assert final (!wr_enable
                    && !rd_enable
                    && SDA_out)
            else $display("error: reset state");
        end
    end

    sequence read_write_enable_sequence;
        I2C_top_design_inst.I2C2.I2C_Pattern_Detector_inst.CS == 2'b01
        && I2C_top_design_inst.I2C2.I2C_Pattern_Detector_inst.counter == 8;
    endsequence

    sequence detect_NACK_sequence;
        I2C_top_design_inst.I2C2.I2C_Pattern_Detector_inst.CS == 2'b10 &&
        I2C_top_design_inst.I2C2.I2C_Pattern_Detector_inst.counter == 8 &&
        !I2C_top_design_inst.I2C2.I2C_Pattern_Detector_inst.first_time;
    endsequence

    sequence ACK_sequence;
        I2C_top_design_inst.I2C2.I2C_Pattern_Detector_inst.CS == 2'b01
        && I2C_top_design_inst.I2C2.I2C_Pattern_Detector_inst.counter == 8;
    endsequence

    sequence signal_recieved_check_sequence;
        I2C_top_design_inst.I2C1.I2C_Controller_inst.CS == 2'b11
        && rd_enable && I2C_top_design_inst.I2C1.I2C_Controller_inst.counter == 8
        && !SDA_in;
    endsequence

    property write_enable_property;
        @(posedge SCL_in) disable iff (!rst_n)
        read_write_enable_sequence |=> if (I2C_top_design_inst.I2C2.I2C_Pattern_Detector_inst.control_signal[7:1] == 7'h10)
            if (!I2C_top_design_inst.I2C2.I2C_Pattern_Detector_inst.control_signal[0]) wr_enable == 0
            else wr_enable == 1
        else wr_enable == 0;
    endproperty

    property read_enable_property;
        @(posedge SCL_in) disable iff (!rst_n)
        read_write_enable_sequence |=> if (I2C_top_design_inst.I2C2.I2C_Pattern_Detector_inst.control_signal[7:1] == 7'h10)
            if (!I2C_top_design_inst.I2C2.I2C_Pattern_Detector_inst.control_signal[0]) rd_enable == 1
            else rd_enable == 0
        else rd_enable == 0;
    endproperty

    property ACK_property;
        @(negedge SCL_in) disable iff (!rst_n)
        ACK_sequence |=> if (I2C_top_design_inst.I2C2.I2C_Pattern_Detector_inst.control_signal[7:1] == 7'h10) SDA_out == 0
        else SDA_out == 1;
    endproperty

    property signal_recieved_check_property;
        @(posedge clk) disable iff (!rst_n)
        signal_recieved_check_sequence |=> controller_data_rsp == $past(controller_data_req);
    endproperty
```

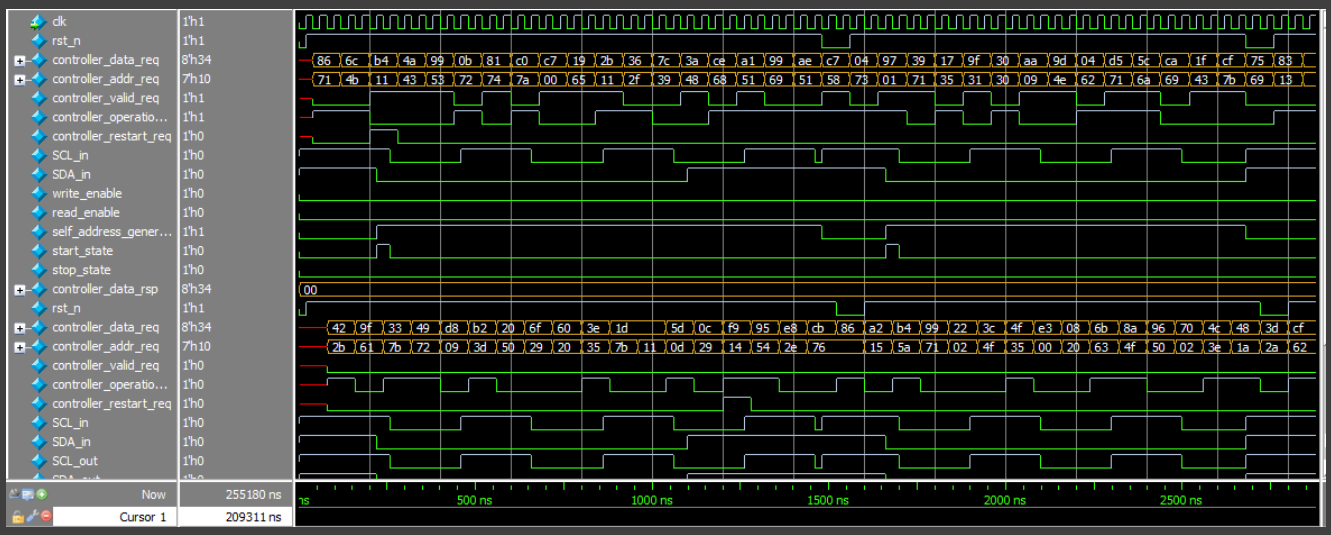
```
    assert property (write_enable_property)
    else $display("error: write operation");
    cover property (write_enable_property);

    assert property (read_enable_property)
    else $display("error: read operation");
    cover property (read_enable_property);

    assert property (ACK_property)
    else $display("error: ACK operation");
    cover property (ACK_property);

    assert property (signal_recieved_check_property)
    else $display("error: recieve operation");
    cover property (signal_recieved_check_property);

endmodule
```

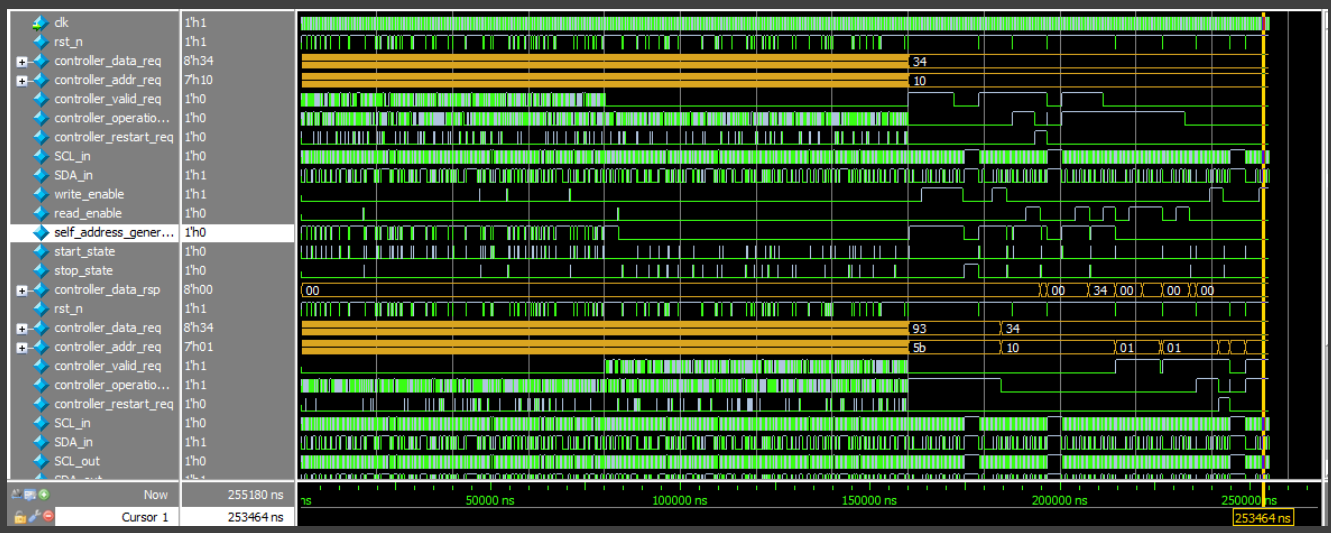
3.3.7 I2C_top

```
////////////////////////////////////////////////////////////////////////////////
//Name: Abdelrahman Mohamed Ragab
// Module-Name: I2C_top
////////////////////////////////////////////////////////////////////////////////

import uvm_pkg::*;
import I2C_test_pkg::*;
`include "uvm_macros.svh";

module I2C_top ();

bit clk;

initial begin
    clk = 0;
    forever begin
        #20;
        clk =~clk;
    end
end

I2C_interface I2C_interface_1 (clk);
I2C_interface I2C_interface_2 (clk);

logic rst_n;

I2C_top_design I2C_top_design_inst (
    .clk(clk),
    .rst_n1(I2C_interface_1.rst_n),
    .rst_n2(I2C_interface_2.rst_n),
    .controller_data_req1(I2C_interface_1.controller_data_req),
    .controller_addr_req1(I2C_interface_1.controller_addr_req),
    .controller_valid_req1(I2C_interface_1.controller_valid_req),
    .controller_operation_req1(I2C_interface_1.controller_operation_req),
    .controller_restart_req1(I2C_interface_1.controller_restart_req),
    .error_signal1(I2C_interface_1.error_signal),
    .controller_data_req2(I2C_interface_2.controller_data_req),
    .controller_addr_req2(I2C_interface_2.controller_addr_req),
    .controller_valid_req2(I2C_interface_2.controller_valid_req),
    .controller_operation_req2(I2C_interface_2.controller_operation_req),
    .controller_restart_req2(I2C_interface_2.controller_restart_req),
    .error_signal2(I2C_interface_2.error_signal),
    .controller_data_rsp1(I2C_interface_1.controller_data_rsp),
    .controller_data_rsp2(I2C_interface_2.controller_data_rsp)
);

bind I2C_top_design_inst I2C_assertion1 I2C_assertion1_inst (
    .clk(clk),
    .SCL_in(I2C_top_design_inst.I2C1.I2C_Pattern_Detector_inst.SCL_in),
    .SDA_in(I2C_top_design_inst.I2C1.I2C_Pattern_Detector_inst.SDA_in),
    .rst_n(I2C_top_design_inst.I2C1.I2C_Pattern_Detector_inst.rst_n),
    .controller_data_req(I2C_interface_2.controller_data_req),
    .self_address_generated(I2C_top_design_inst.I2C1.I2C_Pattern_Detector_inst.self_address_generated),
    .wr_enable(I2C_top_design_inst.I2C1.I2C_Pattern_Detector_inst.wr_enable),
    .SDA_out(I2C_top_design_inst.I2C1.I2C_Pattern_Detector_inst.SDA_out),
    .rd_enable(I2C_top_design_inst.I2C1.I2C_Pattern_Detector_inst.rd_enable),
    .controller_data_rsp(I2C_interface_1.controller_data_rsp)
);

bind I2C_top_design_inst I2C_assertion2 I2C_assertion2_inst (
    .clk(clk),
    .SCL_in(I2C_top_design_inst.I2C2.I2C_Pattern_Detector_inst.SCL_in),
    .SDA_in(I2C_top_design_inst.I2C2.I2C_Pattern_Detector_inst.SDA_in),
    .rst_n(I2C_top_design_inst.I2C2.I2C_Pattern_Detector_inst.rst_n),
    .controller_data_req(I2C_interface_1.controller_data_req),
    .self_address_generated(I2C_top_design_inst.I2C2.I2C_Pattern_Detector_inst.self_address_generated),
    .wr_enable(I2C_top_design_inst.I2C2.I2C_Pattern_Detector_inst.wr_enable),
    .SDA_out(I2C_top_design_inst.I2C2.I2C_Pattern_Detector_inst.SDA_out),
    .rd_enable(I2C_top_design_inst.I2C2.I2C_Pattern_Detector_inst.rd_enable),
    .controller_data_rsp(I2C_interface_2.controller_data_rsp)
);
```

```
assign I2C_interface_1.start_state = I2C_top_design_inst.I2C1.I2C_Pattern_Detector_inst.start_state;
assign I2C_interface_2.start_state = I2C_top_design_inst.I2C2.I2C_Pattern_Detector_inst.start_state;
assign I2C_interface_1.stop_state = I2C_top_design_inst.I2C1.I2C_Pattern_Detector_inst.stop_state;
assign I2C_interface_2.stop_state = I2C_top_design_inst.I2C2.I2C_Pattern_Detector_inst.stop_state;

assign I2C_interface_1.SDA_in = I2C_top_design_inst.SDA;
assign I2C_interface_1.SCL_in = I2C_top_design_inst.SCL;
assign I2C_interface_2.SDA_in = I2C_top_design_inst.SDA;
assign I2C_interface_2.SCL_in = I2C_top_design_inst.SCL;

assign I2C_interface_1.SDA_out = I2C_top_design_inst.SDA;
assign I2C_interface_1.SCL_out = I2C_top_design_inst.SCL;
assign I2C_interface_2.SDA_out = I2C_top_design_inst.SDA;
assign I2C_interface_2.SCL_out = I2C_top_design_inst.SCL;

assign I2C_interface_1.self_address_generated = I2C_top_design_inst.I2C1.self_address_generated;
assign I2C_interface_2.self_address_generated = I2C_top_design_inst.I2C2.self_address_generated;

assign I2C_interface_1.write_enable = I2C_top_design_inst.I2C1.write_enable_pattern_detector | I2C_top_design_inst.I2C1.write_enable_controller;
assign I2C_interface_2.write_enable = I2C_top_design_inst.I2C2.write_enable_pattern_detector | I2C_top_design_inst.I2C2.write_enable_controller;
assign I2C_interface_1.read_enable = I2C_top_design_inst.I2C1.read_enable_pattern_detector | I2C_top_design_inst.I2C1.read_enable_controller;
assign I2C_interface_2.read_enable = I2C_top_design_inst.I2C2.read_enable_pattern_detector | I2C_top_design_inst.I2C2.read_enable_controller;

initial begin
    uvm_config_db #(virtual I2C_interface)::set(null,"*","int1",I2C_interface_1);
    uvm_config_db #(virtual I2C_interface)::set(null,"*","int2",I2C_interface_2);
    run_test("I2C_test_class");
end


endmodule
```

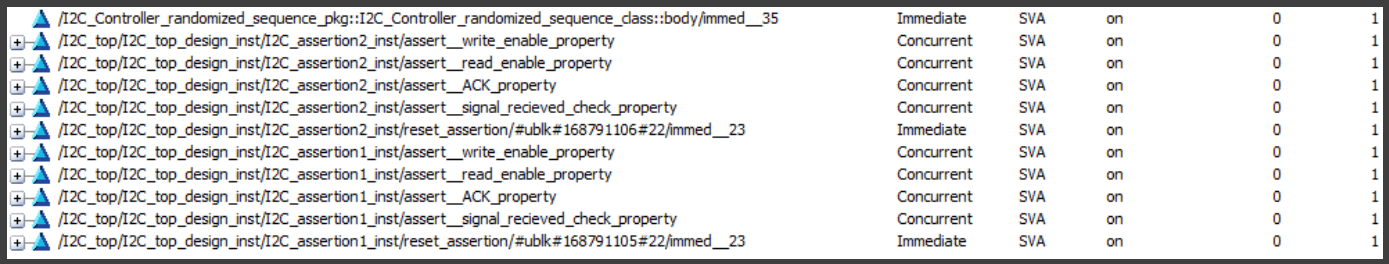
3.3.8 Simulation Results



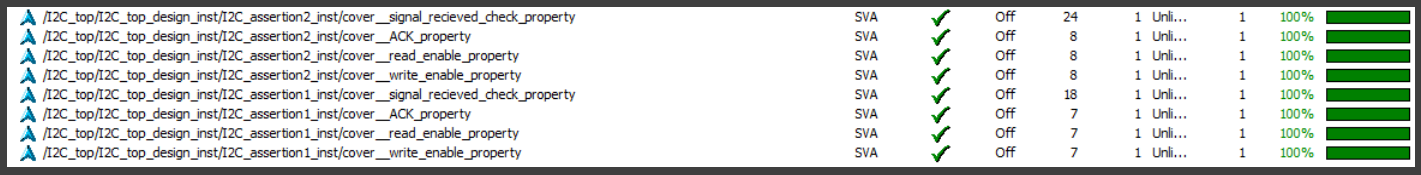



```
# UVM_INFO I2C_Pattern_Detector_scoreboard.sv(157) @ 255180: uvm_test_top.test_enviroment.enviroment_Pattern_Detector_scoreboard1 [report_phase] correct case = 4367 , wrong case = 0
# UVM_INFO I2C_Pattern_Detector_scoreboard.sv(157) @ 255180: uvm_test_top.test_enviroment.enviroment_Pattern_Detector_scoreboard2 [report_phase] correct case = 4020 , wrong case = 0
# UVM_INFO I2C_Controller_scoreboard.sv(357) @ 255180: uvm_test_top.test_enviroment.enviroment_controller_scoreboard1 [report_phase] correct case = 3590 , wrong case = 0
# UVM_INFO I2C_Controller_scoreboard.sv(357) @ 255180: uvm_test_top.test_enviroment.enviroment_controller_scoreboard2 [report_phase] correct case = 3813 , wrong case = 0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :41340
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
```

As we can see there is no checker errors



/I2C_Controller_randomized_sequence_pkg::I2C_Controller_randomized_sequence_class::body/immed__35	Immediate	SVA	on	0	1
/I2C_top/I2C_top_design_inst/I2C_assertion2_inst/assert__write_enable_property	Concurrent	SVA	on	0	1
/I2C_top/I2C_top_design_inst/I2C_assertion2_inst/assert__read_enable_property	Concurrent	SVA	on	0	1
/I2C_top/I2C_top_design_inst/I2C_assertion2_inst/assert__ACK_property	Concurrent	SVA	on	0	1
/I2C_top/I2C_top_design_inst/I2C_assertion2_inst/assert__signal_recieved_check_property	Concurrent	SVA	on	0	1
/I2C_top/I2C_top_design_inst/I2C_assertion2_inst/reset_assertion/#ublk#168791106#22/immed__23	Immediate	SVA	on	0	1
/I2C_top/I2C_top_design_inst/I2C_assertion1_inst/assert__write_enable_property	Concurrent	SVA	on	0	1
/I2C_top/I2C_top_design_inst/I2C_assertion1_inst/assert__read_enable_property	Concurrent	SVA	on	0	1
/I2C_top/I2C_top_design_inst/I2C_assertion1_inst/assert__ACK_property	Concurrent	SVA	on	0	1
/I2C_top/I2C_top_design_inst/I2C_assertion1_inst/assert__signal_recieved_check_property	Concurrent	SVA	on	0	1
/I2C_top/I2C_top_design_inst/I2C_assertion1_inst/reset_assertion/#ublk#168791105#22/immed__23	Immediate	SVA	on	0	1



/I2C_top/I2C_top_design_inst/I2C_assertion2_inst/cover__signal_recieved_check_property	SVA	✓	Off	24	1	Unli...	1	100%
/I2C_top/I2C_top_design_inst/I2C_assertion2_inst/cover__ACK_property	SVA	✓	Off	8	1	Unli...	1	100%
/I2C_top/I2C_top_design_inst/I2C_assertion2_inst/cover__read_enable_property	SVA	✓	Off	8	1	Unli...	1	100%
/I2C_top/I2C_top_design_inst/I2C_assertion2_inst/cover__write_enable_property	SVA	✓	Off	8	1	Unli...	1	100%
/I2C_top/I2C_top_design_inst/I2C_assertion1_inst/cover__signal_recieved_check_property	SVA	✓	Off	18	1	Unli...	1	100%
/I2C_top/I2C_top_design_inst/I2C_assertion1_inst/cover__ACK_property	SVA	✓	Off	7	1	Unli...	1	100%
/I2C_top/I2C_top_design_inst/I2C_assertion1_inst/cover__read_enable_property	SVA	✓	Off	7	1	Unli...	1	100%
/I2C_top/I2C_top_design_inst/I2C_assertion1_inst/cover__write_enable_property	SVA	✓	Off	7	1	Unli...	1	100%

As we can see there is no assertion errors

4. Reference

The I2C specfile: I2C-bus specification and user manual