ASSIGNMENT – 3

1. Synthesize the up-down counter on an FPGA and simulate using test vectors generated from the ARM processor.

a. Initial set of test vectors should be generated using python

b. Those vectors should be given as input to the design

c. The output should be verified

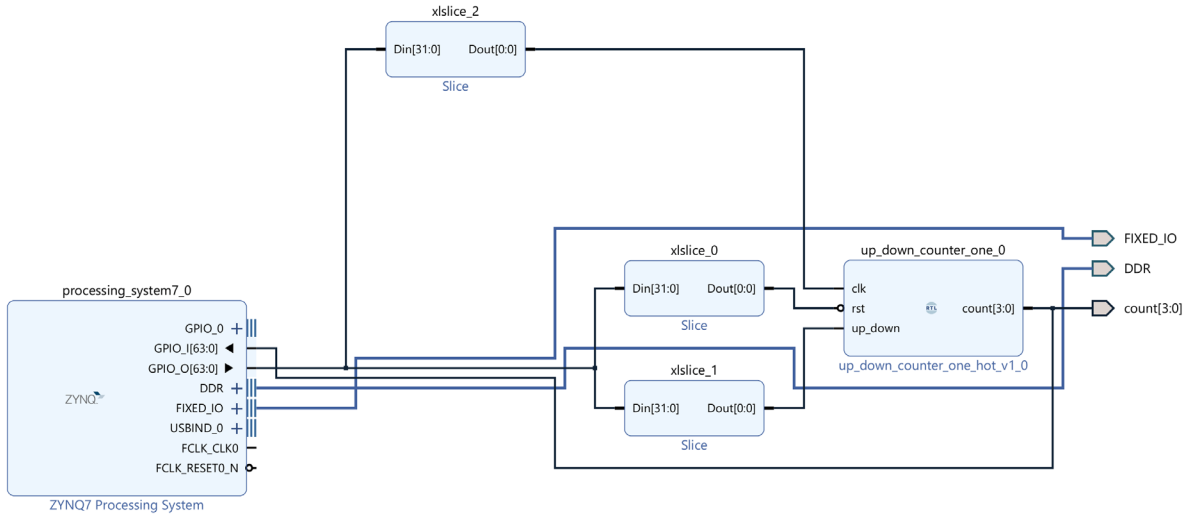
d. Repeat the process, but using input vectors from a file and writing the output to a file and checking the validity using a golden file(expected output file).

Module code:

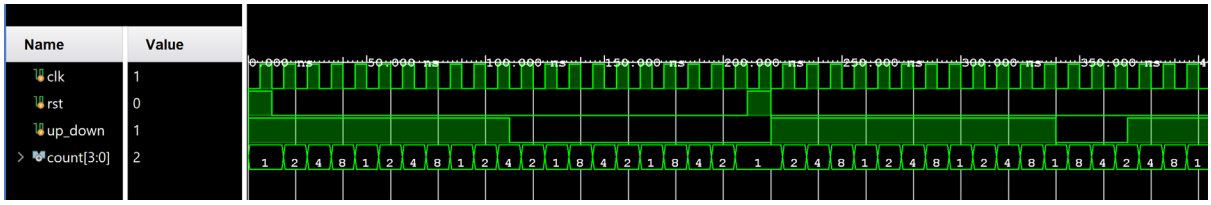
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
////
// Company:
// Engineer: RAAHUL L S
//
// Create Date: 16.09.2024 19:03:22
// Design Name:
// Module Name: up_down_counter_one_hot
// Project Name:
// Target Devices:
// Tool Versions:
// Description: One-hot encoded up-down counter
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
////

module up_down_counter_one_hot (
    input clk,
    input rst,
    input up_down,    // 1: up, 0: down
    output reg [3:0] count // One-hot encoded output
);
reg [25:0] count_value; // 26-bit counter register
```

```
reg clk_out;
always @(posedge clk or posedge rst) begin
        if (rst) begin
            count_value <= 0;
            clk_out <= 0;
        end else if (count_value == 50_000_000 - 1) begin
            count_value <= 0;
            clk_out <= ~clk_out;
        end else begin
            count_value <= count_value + 1;
        end
    end
    // Initial state
    always @(posedge clk or posedge rst) begin
        if (rst)
            count <= 4'b0001;  // Start in one-hot state 0001
        else if (up_down) begin
            // Count up: Shift left, wrap around to bit 0 if overflow
            if (count == 4'b1000)
                count <= 4'b0001;
            else
                count <= count << 1;
        end
        else begin
            // Count down: Shift right, wrap around to bit 3 if underflow
            if (count == 4'b0001)
                count <= 4'b1000;
            else
                count <= count >> 1;
        end
    end
endmodule
```

BLOCK DESIGN:



SIMULATION:



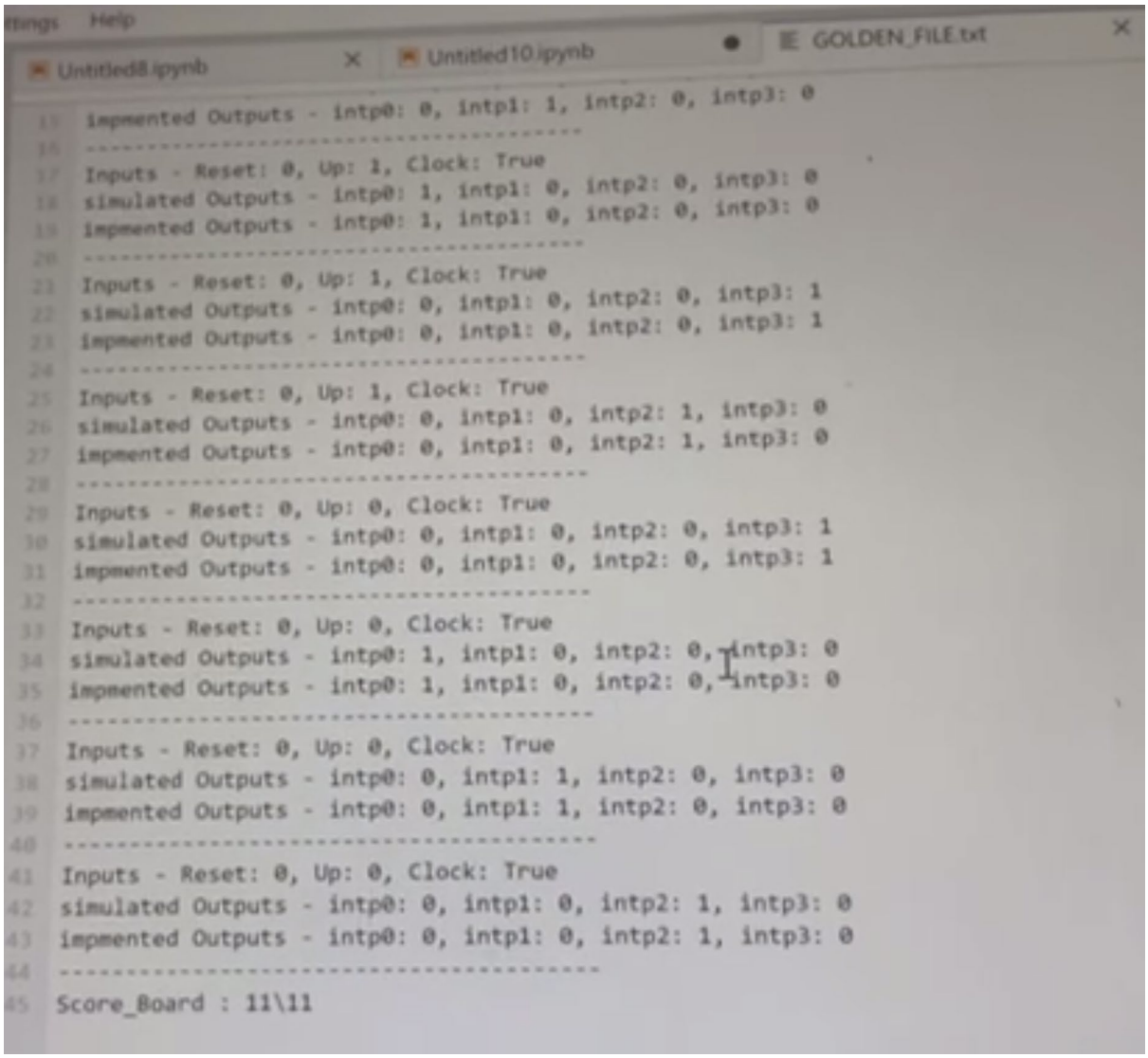
Test vector and Golden file generation code:

```
class OneHotCounter:
    def __init__(self):
        self.outputs = [1, 0, 0, 0]  # Initially, intp0 is high (One-Hot
encoding)

    def reset(self):
        """Resets the counter to its initial state."""
        self.outputs = [1, 0, 0, 0]  # Reset makes intp0 high

    def clock_rising_edge(self, up):
        """Simulates the counter behavior on the rising edge of the clock."""
        if up:  # Move left (shift left)
            self.outputs = self.outputs[1:] + [self.outputs[0]]  # Left cyclic
shift
        else:  # Move right (shift right)
            self.outputs = [self.outputs[-1]] + self.outputs[:-1]  # Right
cyclic shift
```

```
    def get_outputs(self):
        """Returns the current output states (intp0, intp1, intp2, intp3)."""
        return self.outputs

# Function to log inputs and outputs to a file
def log_to_file(file_name, reset, up, clock, outputs):
    with open(file_name, 'a') as f:
        f.write(f"Inputs - Reset: {reset}, Up: {up}, Clock: {clock}\n")
        f.write(f"Outputs - intp0: {outputs[0]}, intp1: {outputs[1]}, intp2:
{outputs[2]}, intp3: {outputs[3]}\n")
        f.write("-" * 40 + "\n")

# Simulation
import time

# Instantiate the counter
counter = OneHotCounter()

# Simulation setup
reset = 0
up = 1  # If True, counter shifts left; if False, shifts right
clock = 0  # Initial clock state

# File to store inputs and outputs
file_name = "counter_log.txt"

# Clear file at the start of the simulation
open(file_name, 'w').close()

# Simulating a continuous clock signal and counter behavior
for i in range(20):
    # Clock rising edge
    clock = not clock  # Toggling the clock
    if clock:  # Rising edge of the clock
        if reset:
            counter.reset()  # Reset on rising edge
        else:
            counter.clock_rising_edge(up)  # Shift left or right based on `up`
pin

        # Get the current state of the outputs
        outputs = counter.get_outputs()

        # Log inputs and outputs to file
        log_to_file(file_name, reset, up, clock, outputs)

    # Simulate a delay for the clock
    time.sleep(0.5)
```

```
    # For demonstration, toggle up/down every 10 clock cycles
    if i == 10:
        up = not up  # Toggle up direction after 10 cycles
```

GOLDEN FILE OUTPUT:



IMPLEMENTATION:

[PYNQ_IMPLENTATION.MOV]

Brief overview of hardware software co-design

Hardware-software co-design is a design methodology that involves developing systems by integrating both hardware and software components, allowing designers to partition tasks based on their computational complexity and real-time requirements. This approach leverages the strengths of software processors (for sequential, less-intensive tasks) and hardware accelerators such as FPGAs (for highly parallel, computationally intensive tasks) to optimize performance, power consumption, and flexibility.

In the typical hardware-software co-design workflow, the software component is used to control the system, and the hardware component accelerates specific tasks that benefit from parallelism and customization. The system benefits from the flexibility of software development while achieving significant performance boosts through custom hardware.

1. The above implementation shows abstract view of hardware software codesign. Here the software meaning the processor is used only for testing.
2. This can be extended to full fledged design where the Processor could take care of all the sequentially and less computationally intensive tasks while more computationally intensive tasks and tasks that can be optimized using parallel processing would be taken care by the FPGAs.

Steps followed in the implementation

1. Write a Verilog code for counter include the clock-divider module inside.
2. Using the module written create a block diagram to establish connection between software and FPGA
3. Generate wrapper and output for the block design
4. Now generate the .hwh , .bit and .tcl and save it with same name.
5. Now connect the PYNQ board to the computer through the ethernet and login to the Jupyter lab.
6. Upload all the generated hardware and design file to the board.
7. Use the overlay module to implement the design.
8. Run the python to generate test cvetors and monitor the output and save it in the golden file.

LABSHEET 1:

Variables Declarations

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
////
// Company:
// Engineer: RAAHUL L S
//
// Create Date: 07.10.2024 07:11:08
// Design Name: VARIABLE DECLARATION
// Module Name: qp1
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
////


byte s2b8 = 0;    // 8-bit signed integer
shortint  s = 16'sd0;  // 16-bit signed integer
int       s2b3 = 32'sd0;  // 32-bit signed integer
longint   s2b6 = 64'sd0;  // 64-bit signed integer

// 2-state 32-bit unsigned integer
int   s2b3u = 32'd0;  // 32-bit unsigned integer

// Declare a dynamic array of int data type
int data_da[];  // Dynamic array declaration

// Declare queues of int data type
int data_q[$];  // Queue for data
int addr_q[$];  // Queue for addresses

// Declare associative array indexed by bit[7:0]
int data_mem[bit[7:0]];  // Associative array declaration
```

```
module qp1(

    );
endmodule
```

USAGE OF VARIABLES:

DYNAMIC ARRAY:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
////
// Company:
// Engineer: RAAHUL L S
//
// Create Date: 07.10.2024 07:18:28
// Design Name:
// Module Name: dynamic_array_example
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
////


module dynamic_array;

    int data_da[];
    int result;

    initial begin
        data_da = new[10];
        for (int i = 0; i < 10; i++) begin
            data_da[i] = $urandom_range(0, 24);
        end
        $display("Array before sorting: ");
        foreach (data_da[i]) begin
            $display("data_da[%0d] = %0d", i, data_da[i]);
        end
```

```
        result = data_da.sum();
        $display("\nSum of all elements: %0d", result);
        data_da.sort();
        $display("\nArray after sorting: ");
        foreach (data_da[i]) begin
            $display("data_da[%0d] = %0d", i, data_da[i]);
        end
    end

endmodule
```

OUTPUT:

```
Vivado Simulator does not support tracing of System
# run 1000ns
Array before sorting:
data_da[0] = 24
data_da[1] = 20
data_da[2] = 16
data_da[3] = 17
data_da[4] = 10
data_da[5] = 13
data_da[6] = 0
data_da[7] = 9
data_da[8] = 11
data_da[9] = 20

Sum of all elements: 140

Array after sorting:
data_da[0] = 0
data_da[1] = 9
data_da[2] = 10
data_da[3] = 11
data_da[4] = 13
data_da[5] = 16
data_da[6] = 17
data_da[7] = 20
data_da[8] = 20
data_da[9] = 24
INFO: [USF-XSim-96] XSim completed. Design snapshot
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
```

QUEUES :

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
////
// Company:
// Engineer: RAAHUL L S
//
// Create Date: 07.10.2024 07:36:37
// Design Name:
// Module Name: queue_example
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
////

module queue_example;

    int data_q[$];
    int addr_q[$];

    initial begin
        data_q.push_back(5);
        data_q.push_back(10);
        data_q.push_back(15);
        $display("Queue after push_back operations: %p", data_q);
        $display("Queue size: %0d", data_q.size());

        data_q.insert(1, 20);
        $display("Queue after insert(1, 20): %p", data_q);

        data_q.delete(2);
        $display("Queue after delete(2): %p", data_q);

        data_q.push_front(3);
        $display("Queue after push_front(3): %p", data_q);

        data_q.pop_front();
        $display("Queue after pop_front: %p", data_q);
        $display("Queue size after pop_front: %0d", data_q.size());
```

```
        repeat (10) begin
            int random_address = $urandom_range(1000, 9999);
            addr_q.push_back(random_address);
        end

        $display("\nGenerated random addresses in addr_q: %p", addr_q);
    end

endmodule
```

OUTPUT:

```
WARNING: Simulation object /queue_example/addr_q was
Vivado Simulator does not support tracing of System
INFO: [Wavedata 42-43] There are no traceable object
# run 1000ns
Queue after push_back operations: '{5,10,15}
Queue size: 3
Queue after insert(1, 20): '{5,20,10,15}
Queue after delete(2): '{5,20,15}
Queue after push_front(3): '{3,5,20,15}
Queue after pop_front: '{5,20,15}
Queue size after pop_front: 3

Generated random addresses in addr_q: '{2299,3595,15
INFO: [USF-XSim-96] XSim completed. Design snapshot
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
```

ASSOSCIATIVE ARRAYS:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
////
// Company:
// Engineer: RAAHUL L S
//
// Create Date: 07.10.2024 08:32:35
// Design Name:
// Module Name: associative_array_example
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
```

```
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
////
bit [7:0] first_index;
bit [7:0] last_index;

module associative_array_example;

    int addr_q[$];           // Queue for addresses
    int data_mem[bit[7:0]];   // Associative array indexed by bit[7:0]

    initial begin
        // Step 1: Generate 10 random addresses and push them into addr_q
        repeat (10) begin
            int random_address = $urandom_range(0, 255);  // Limit addresses
to fit bit[7:0] range
            addr_q.push_back(random_address);
        end

        // Step 2: Update the associative array with random data based on the
address stored in addr_q
        while (addr_q.size() > 0) begin
            int address = addr_q.pop_front();             // Pop an address
from addr_q
            data_mem[address] = $urandom_range(0, 100);    // Assign random
data to that address
        end

        // Step 3: Display the contents of the associative array using foreach
        $display("Contents of associative array:");
        foreach (data_mem[i]) begin
            $display("data_mem[%0d] = %0d", i, data_mem[i]);
        end

        // Step 4: Display the first index of the array using 'first' method

        if (data_mem.first(first_index)) begin
            $display("\nFirst index: %0d", first_index);
        end

        // Step 5: Display the first element of the array
        $display("First element: %0d", data_mem[first_index]);
```

```
        // Step 6: Display the last index of the array using 'last' method

        if (data_mem.last(last_index)) begin
            $display("\nLast index: %0d", last_index);
        end

        // Step 7: Display the last element of the array
        $display("Last element: %0d", data_mem[last_index]);
    end

endmodule
```

OUTPUT:

```
INFO: [Wavedata 42-43] There are no
# run 1000ns
Contents of associative array:
data_mem[11] = 43
data_mem[23] = 15
data_mem[67] = 37
data_mem[168] = 32
data_mem[193] = 92
data_mem[197] = 96
data_mem[205] = 34
data_mem[212] = 12
data_mem[227] = 71
data_mem[246] = 11

First index: 11
First element: 43

Last index: 246
Last element: 11
INFO: [USF-XSim-96] XSim completed.
INFO: [USF-XSim-97] XSim simulation
```

USER DEFINED TYPE:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
////
// Company:
// Engineer: RAAHUL L S
//
// Create Date: 07.10.2024 08:38:58
// Design Name:
// Module Name: pixel_example
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
////


module pixel_example;

    // Step 1: Create a user-defined type (struct) to store pixel values
    typedef struct packed {
        bit [7:0] red;
        bit [7:0] green;
        bit [7:0] blue;
    } pixel_t;

    // Step 2: Create 5 pixels and initialize them with values for red,
violet, and pink
    pixel_t pixels[5];

    initial begin
        // Initialize the pixels with color values (use appropriate RGB
values)
        pixels[0] = '{8'hFF, 8'h00, 8'h00};   // Red: (255, 0, 0)
        pixels[1] = '{8'hEE, 8'h82, 8'hEE};   // Violet: (238, 130, 238)
        pixels[2] = '{8'hFF, 8'hC0, 8'hCB};   // Pink: (255, 192, 203)
        pixels[3] = '{8'hFF, 8'h00, 8'h00};   // Red: (255, 0, 0)
        pixels[4] = '{8'hEE, 8'h82, 8'hEE};   // Violet: (238, 130, 238)

        // Step 3: Display the pixel values using foreach loop
```

```
        $display("Pixel values:");
        foreach (pixels[i]) begin
            $display("Pixel[%0d] -> R: %0d, G: %0d, B: %0d", i, pixels[i].red,
pixels[i].green, pixels[i].blue);
        end
    end

endmodule
```

OUTPUT:

```
# run 1000ns
Pixel values:
Pixel[0] -> R: 255, G: 0, B: 0
Pixel[1] -> R: 238, G: 130, B: 238
Pixel[2] -> R: 255, G: 192, B: 203
Pixel[3] -> R: 255, G: 0, B: 0
Pixel[4] -> R: 238, G: 130, B: 238
INFO: [USF-XSim-96] XSim completed. De
```

FINITE STATE MACHINE:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
////
// Company:
// Engineer: RAAHUL L S
//
// Create Date: 07.10.2024 08:51:15
// Design Name:
// Module Name: simple_fsm
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
```

```
//
//////////////////////////////////////////////////////////////////////////////
////


module simple_fsm;

    // Define an enumerated type for the states ON and OFF
    typedef enum logic {
        OFF,  // 0
        ON    // 1
    } state_t;

    // Declare a variable for the current state
    state_t current_state;

    initial begin
        // Initialize the state to OFF
        current_state = OFF;

        forever begin
            // Display the current state
            $display("Current state: %s", state_name(current_state));

            // Wait for some time (2 time units)
            #2;

            // Toggle the state
            current_state = (current_state == OFF) ? ON : OFF;
        end
    end

    // Function to return the name of the state as a string
    function string state_name(state_t state);
        case (state)
            OFF: state_name = "OFF";
            ON:  state_name = "ON";
        endcase
    endfunction

endmodule
```

OUTPUT:

```
# run 1000ns
Current state: OFF
Current state: ON
Current state: OFF
Current state: ON
Current state: OFF
Current state: ON
Current state: OFF
Current state: ON
Current state: OFF
Current state: ON
```

STRINGS:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
////
// Company:
// Engineer: Raahul L S
//
// Create Date: 10.10.2024 10:00:10
// Design Name:
// Module Name: str
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
////
```

```
module str;
string my_string = "Hello, SystemVerilog!";
string sug = "Hello, SystemVerilog!";
string replaced_str = "Hello, SystemVerilog!";
        // Step 1: Define and initialize a string


        // Step 2: Use string methods
        // Method 1: Length of the string
        int length = my_string.len();
        $display("Length of string: %0d", length);

        sug = my_string.substr(7,19);// Start at index 7, length 6
        $display("Substring: %s", sug);

        // Method 3: Replace "System" with "World"
        //replaced_str = my_string.replace("System", "World");
        //$display("Replaced string: %s", replaced_str);
      end
endmodule
```

output:

```
Built simulation snapshot str_behav
INFO: [USF-XSim-69] 'elaborate' step finished in '2' seconds
Time resolution is 1 ps
Length of string: 21
Substring: SystemVerilog
```

LAB SHEET 2

SET 1:

TOP MODULE:

```
module arb_top;

    // Clock signal
    logic clk = 0;
    always #5 clk = ~clk;
    // Instantiate the interface
    arb_if arb_if_inst(clk);
    arb_monitor monitor(arb_if_inst);
    // Instantiate the DUT
    arbiter dut(arb_if_inst);

    // Instantiate the testbench
    arb_tb tb(arb_if_inst);

    // Instantiate the monitor


endmodule
```

DUT MODULE:

```
module arbiter(arb_if arb);

    always_ff @(posedge arb.clk or posedge arb.reset) begin
        if (arb.reset) begin
            arb.grant <= 2'b00;  // Reset the grant signal
        end else begin
            case (arb.request)
                2'b01: arb.grant <= 2'b01; // Grant block 1
                2'b10: arb.grant <= 2'b10; // Grant block 2
                default: arb.grant <= 2'b00; // No grant
            endcase
        end
    end
endmodule
```

TEST MODULE:

```
module arb_tb(arb_if arb_if_inst);


    initial begin
        // Initialize signals
        arb_if_inst.request = 2'b00;
        arb_if_inst.reset = 1'b1;
        $display("Time: %0t | Reset asserted", $time);
        #10 arb_if_inst.request = 2'b01;
        $display("Time: %0t | Request Generated for Block 1", $time);
        #10 arb_if_inst.request = 2'b00;
        @(posedge arb_if_inst.clk ) begin
        arb_if_inst.reset = 1'b0;
        $display("Time: %0t | Reset Deasserted", $time);

        // Generate request signals
        #10 arb_if_inst.request = 2'b01;
        $display("Time: %0t | Request Generated for Block 1", $time);

        #30 arb_if_inst.request = 2'b10;
        $display("Time: %0t | Request Generated for Block 2", $time);

        // Finish simulation
        #15
        $finish;
        end
    end

endmodule
```

INTERFACE MODULE:

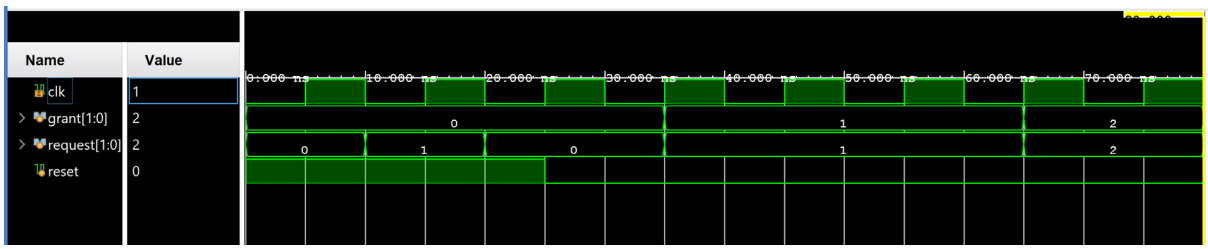
```
interface arb_if(input logic clk);
    // Interface signals
    logic [1:0] grant;    // 2-bit grant signal
    logic [1:0] request;  // 2-bit request signal
    logic reset;          // 1-bit reset signal
endinterface
```

MONITOR MODULE:

```
module arb_monitor(arb_if arb);

    initial begin
        forever begin
            // Wait for a request
            @(posedge arb.clk);
            if (arb.request != 2'b00) begin
                $display("Time: %0t | Request Detected: %b", $time,
arb.request);
            end

            // Wait for a grant
            @(posedge arb.clk);
            if (arb.grant != 2'b00) begin
                $display("Time: %0t | Grant Issued: %b", $time, arb.grant);
            end
        end
    end

endmodule
```

OUTPUT:

```
# }
# run 1000ns
Time: 0 | Reset asserted
Time: 10000 | Request Generated for Block 1
Time: 25000 | Reset Deasserted
Time: 35000 | Request Generated for Block 1
Time: 45000 | Request Detected: 01
Time: 55000 | Grant Issued: 01
Time: 65000 | Request Generated for Block 2
Time: 65000 | Request Detected: 10
Time: 75000 | Grant Issued: 10
```



SET 2

INTERFACE MODULE:

```
interface arb_if(input logic clk);
    // Interface signals
    logic [1:0] grant;    // 2-bit grant signal
    logic [1:0] request;  // 2-bit request signal
    logic reset;          // 1-bit reset signal

    modport test (output request,output reset, input grant,input clk);
    modport dut (input request,input reset, output grant,input clk);
    modport monitor (input request,input reset, input grant,input clk);


endinterface
```

TOP MODULE:

```
module arb_top;

    // Clock signal
    logic clk = 0;
    always #5 clk = ~clk;
    // Instantiate the interface
    arb_if arb_if_inst(clk);
    arb_monitor monitor(arb_if_inst.monitor);
    // Instantiate the DUT
    arbiter dut(arb_if_inst.dut);

    // Instantiate the testbench
    arb_tb tb(arb_if_inst.test);

    // Instantiate the monitor


endmodule
```

DUT MODULE:

```
module arbiter(arb_if.dut arb);

    always_ff @(posedge arb.clk or posedge arb.reset) begin
        if (arb.reset) begin
            arb.grant <= 2'b00;  // Reset the grant signal
        end else begin
```

```
            case (arb.request)
                2'b01: arb.grant <= 2'b01; // Grant block 1
                2'b10: arb.grant <= 2'b10; // Grant block 2
                default: arb.grant <= 2'b00; // No grant
            endcase
        end
    end
endmodule
```

TEST MODULE:

```
module arb_tb(arb_if.test arb_if_inst);


    initial begin
        // Initialize signals
        arb_if_inst.request = 2'b00;
        arb_if_inst.reset = 1'b1;
        $display("Time: %0t | Reset asserted", $time);
        #10 arb_if_inst.request = 2'b01;
        $display("Time: %0t | Request Generated for Block 1", $time);
        #10 arb_if_inst.request = 2'b00;
        @(posedge arb_if_inst.clk ) begin
        arb_if_inst.reset = 1'b0;
        $display("Time: %0t | Reset Deasserted", $time);

        // Generate request signals
        #10 arb_if_inst.request = 2'b01;
        $display("Time: %0t | Request Generated for Block 1", $time);

        #30 arb_if_inst.request = 2'b10;
        $display("Time: %0t | Request Generated for Block 2", $time);

        // Finish simulation
        #15
        $finish;
        end
    end

endmodule
```

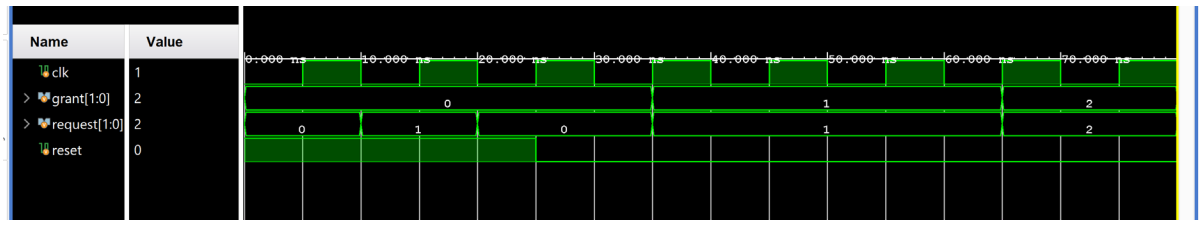
MONITOR MODULE:

```
module arb_monitor(arb_if.monitor arb);

    initial begin
```

```
        forever begin
            // Wait for a request
            @(posedge arb.clk);
            if (arb.request != 2'b00) begin
                $display("Time: %0t | Request Detected: %b", $time,
arb.request);
            end

            // Wait for a grant
            @(posedge arb.clk);
            if (arb.grant != 2'b00) begin
                $display("Time: %0t | Grant Issued: %b", $time, arb.grant);
            end
        end
    end

endmodule
```

OUTPUT:

```
#  }
# run 1000ns
Time: 0 | Reset asserted
Time: 10000 | Request Generated for Block 1
Time: 25000 | Reset Deasserted
Time: 35000 | Request Generated for Block 1
Time: 45000 | Request Detected: 01
Time: 55000 | Grant Issued: 01
Time: 65000 | Request Generated for Block 2
Time: 65000 | Request Detected: 10
Time: 75000 | Grant Issued: 10
$finish called at time : 80 ns : File "F:/VIVADO_PROJ
INFO: [USF-XSim-96] XSim completed. Design snapshot '
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
```



SET 3

INTERFACE MODULE:

```
interface arb_if(input logic clk);
    // Interface signals
    logic [1:0] grant;    // 2-bit grant signal
    logic [1:0] request;  // 2-bit request signal
    logic reset;          // 1-bit reset signal

    parameter ISKEW = 1;
    parameter OSKEW = 2;

    clocking cbtest @(posedge clk);
        default input #ISKEW output #OSKEW;
        input grant;
        output request,reset;
    endclocking

    clocking cbdut @(posedge clk);
        default input #ISKEW output #OSKEW;
        output grant;
        input request,reset;
    endclocking

    clocking cbmon @(posedge clk);
        default input #ISKEW output #OSKEW;
        input request,reset,grant;
    endclocking

    modport test (clocking cbtest, output request,output reset, input
grant,input clk);
    modport dut (input request,input reset, output grant,input clk);
    modport monitor (input request,input reset, input grant,input clk);




endinterface
```

TOP MODULE:

```
module arb_top;

    // Clock signal
    logic clk = 0;
    always #3
```

```
     clk = ~clk;
    // Instantiate the interface
    arb_if arb_if_inst(clk);
    arb_monitor monitor(arb_if_inst.monitor);
    // Instantiate the DUT
    arbiter dut(arb_if_inst.dut);

    // Instantiate the testbench
    arb_tb tb(arb_if_inst.test);

    // Instantiate the monitor


endmodule
```

DUT MODULE:

```
module arbiter(arb_if.dut arb);
    always_ff @(posedge arb.clk or posedge arb.reset) begin
        if (arb.reset) begin
            arb.grant <= 2'b00;  // Reset the grant signal
        end else begin
            case (arb.request)
                2'b01: arb.grant <= 2'b01; // Grant block 1
                2'b10: arb.grant <= 2'b10; // Grant block 2
                default: arb.grant <= 2'b00; // No grant
            endcase
        end
    end
endmodule
```

TEST MODULE:

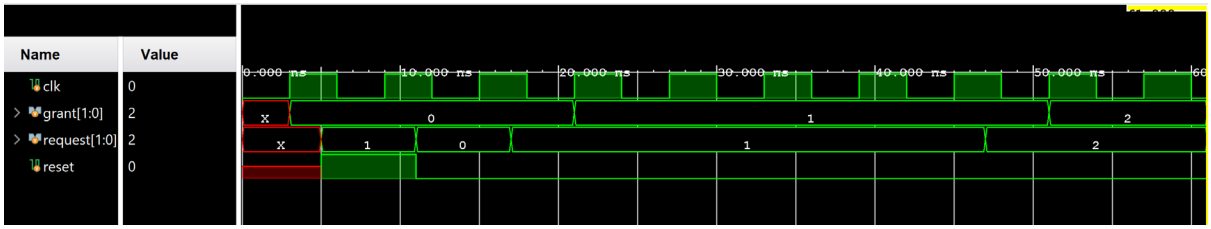
```
module arb_tb(arb_if.test arb_if_inst);


    initial begin
        // Initialize signals
        arb_if_inst.cbtest.request <= 2'b00;
        arb_if_inst.cbtest.reset <= 1'b1;
        $display("Time: %0t | Reset asserted", $time);
#2
arb_if_inst.cbtest.request <= 2'b01;
        $display("Time: %0t | Request Generated for Block 1", $time);
#2
arb_if_inst.cbtest.request <= 2'b00;
```

```
        $display("Time: %0t | Request deGenerated for Block 1", $time);
#2
        @(posedge arb_if_inst.clk ) begin
        arb_if_inst.cbtest.reset <= 1'b0;
        $display("Time: %0t | Reset Deasserted", $time);
#2
        // Generate request signals
arb_if_inst.cbtest.request <= 2'b01;
        $display("Time: %0t | Request Generated for Block 1", $time);
#30
arb_if_inst.cbtest.request <= 2'b10;
        $display("Time: %0t | Request Generated for Block 2", $time);

        // Finish simulation
        #20
        $finish;
        end
    end

endmodule
```

MONITOR MODULE:

```
module arb_monitor(arb_if.monitor arb);

    initial begin
        forever begin
            // Wait for a request
            @(posedge arb.clk) begin
            if (arb.request != 2'b00) begin
                $display("Time: %0t | Request Detected: %b", $time,
arb.request);
            end

            // Wait for a grant
            if (arb.grant != 2'b00) begin
                $display("Time: %0t | Grant Issued: %b", $time, arb.grant);
            end end
        end
    end

endmodule
```

OUTPUT:




```
Time resolution is 1 ps
Time: 0 | Reset asserted
Time: 2000 | Request Generated for Block 1
Time: 4000 | Request deGenerated for Block 1
Time: 9000 | Request Detected: 01
Time: 9000 | Reset Deasserted
Time: 11000 | Request Generated for Block 1
Time: 21000 | Request Detected: 01
Time: 27000 | Request Detected: 01
Time: 27000 | Grant Issued: 01
Time: 33000 | Request Detected: 01
Time: 33000 | Grant Issued: 01
Time: 39000 | Request Detected: 01
Time: 39000 | Grant Issued: 01
Time: 41000 | Request Generated for Block 2
Time: 45000 | Request Detected: 01
Time: 45000 | Grant Issued: 01
Time: 51000 | Request Detected: 10
Time: 51000 | Grant Issued: 01
Time: 57000 | Request Detected: 10
Time: 57000 | Grant Issued: 10
```

RAAHUL LS
CB.EN.U4ECE22143

14ECE347 - ESLD
Lab sheet - 1

1.
```
byte      s2b6 = 0; // s-2, b-8
shortint  s6 = 0;   // s-2, b-16
int       s2b3 = 1; // s-2, b-32
longint   s2b6 = 2; // s-2, b-64
bit [31:0] s2b32 = 0; // s-2, b-32u
int data_da
int dyna[];  // dynamic array
int data_q[$], addr_q[$];
int[:0]data_mem[*];
int data_mem[bit[7:0]];
```

2.
```
module dynamic_array;
    int data_da[];
    int result[];
    initial begin
        data_da = new[10];
        for(int i=0; i<10; i++) begin
            data_da[i] = $urandom_range(0,2)
        end
    foreach (data_da[i]) begin
        $display("data[%0d] = %0d", i, data_da[i]
    end.
    result = data_da.sum();
    $display(" result : %0d", result);
    data_da.sort();
    foreach (data_da[i]) begin
        $display("data[%0d] = %d", i, data_da[i])
    end
end
endmodule.
```

3.
```
module q;
    int data_q[$];
    int addr_q[$];
initial begin
    data_q.push_back(5);
    data_q.push_back(10);
    data_q.push_back(15);
  $display("Queue = %p", data_q);
  $display(" size = %d", data_q.size());
  // {5, 10, 15}.
  data_q.insert(1,20); $display("q: %p,
  // {5,20,10,15}                    data_q)
  data_q.delete(2); $display("q: %p", data_q);
  // {5,20,15}
  data_q.push_front(3); $display("q: %p", data_q);
  // {3,5,20,15}
  data_q.pop_front(); $display("q: %p", data_q);
  // {5,20,15}
  repeat(10) begin
      int addr = $urandom_range(100,999)
      addr_q.push_back(addr)
  end.
  $display("address queue: %p", addr_q);
end
endmodule
```

4. ~~repeat (10) begin~~

```
while (addr_q.size() > 0) begin
    int address = addr_q.pop_front();
    data_mem[address] = $random range(0,100)
end

foreach (data_mem[i]) begin
    ~~int address~~
    $display("data[%0d] = %0d", i, data[i])
end
  [7:0]
bit first_index;
data_mem.first(first_index)
$display("first index : %0d", first_index)
$display("first element : %0d", data_mem[first_index])
bit [7:0] last_index;
data_mem.last(last_index)
$display("last element : %0d", data[last_index])
end
endmodule.
```

5.
```
module Pixel_example;
typedef struct packed {
    bit [7:0] red;
    bit [7:0] green;
    bit [7:0] blue; } pixel_t;
pixel_t pixels[5];
initial begin
    pixels[0] = {8'hFF, 8'h00, 8'h00};
    pixel[1] = {8'hEE, 8'h82, 8'hEE};
    pixel[2] = {8'hFF, 8'hC0, 8'hCB};
    pixel[3] = {8'hFF, 8'h00, 8'h00};
    pixel[4] = {8'hEE, 8'h82, 8'hEE};
foreach pixel[i] begin
    $display("pixel[%0d], R:%0d, G=%0d
    B=%0d", i, pixel[i].red,
    pixel[i].green, pixel[i].blue);
```

6.
```
Module simple_fsm;
    typedef enum logic {
        OFF, ON } state_t;
state_t current_state;
initial begin
    current_state = OFF;
    forever begin
        $display("current state : %s", state_name
                                    (current_state));
    #2
    current_state = (current == OFF) ? ON : OFF;
    end
end
function string state_name(state_t state);
    case (state)
        OFF : state_name = "off";
        ON : state_name = "ON";
    endcase
endfunction
```

LAB SHEET-2

7.
```
Module string_example;
string = "Hello, system";
initial begin
    int length = my_string.len();
    $display("length : %0d", length);
    string sub = my_string.substr(7,6);
    $display("substring: %s", sub);
    string rep = my_string.replace
                (sub, "world");
    $display("Replaced: %s", rep);
end
endmodule
```

1. Interface Design

```
interface arb_if (input logic clk);
    logic [1:0] grant;
    logic [1:0] request;
    logic reset;
endinterface

module arbiter (arb_if arb);
  always_ff @(posedge arb.clk or
                posedge arb.reset) begin
  if (arb.reset) begin
      arb.grant <= 2'b00;
  end else begin
      case (arb.request)
          2'b01 : arb.grant <= 2'b01;
          2'b10 : arb.grant <= 2'b10;
          default : arb.grant <= 2'b00;
      endcase
  end end
endmodule

arb_tb.sv
module arb_tb;
    logic clk
    always #5 clk = ~clk;
    arb_if arb_if_inst(clk);
    arbiter dut(arb_if_inst);
initial begin
```

```
arbif_inst.reset = 1'b1;
arb_if_inst.reset = 2'b00;
#10;
arb_if_inst.reset = 1'b0;
$display("time:%0t | reset", $time);

#10
arb_if_inst.requs = 2'b01;
$display("time:%0t | Request", $time);
#20
arb_if_inst.requs = 2'b10;
$display("time:%0t | Request2", $time);
#50 $finish
end endmodule
```

Monitor Design

```
module arb_monitor(arb_if arb);
initial begin
forever begin
@(posedge arb.clk);
if(arb.request != 2'b00) begin
$display(time:%0t | Req detected" (time);
end
@(posedge arb.clk);
  if (arb.grant != 2'b00) begin
  $display("time:%0t | Grant issued",
          $time); end end endmodul
```

- Top module

```
module arb_top;
  logic clk;
  always #5 clk = ~clk;
  arb_if arb_if_inst(clk);
  arbitter dut(arb_if_inst);
  arb_tb tb();
  arb_monitor monitor(arb_if_inst);
endmodule
```

2. Interface with modports.

```
interface arb_iff(input logic clk);
  logic [1:0] grant;
  logic [1:0] request;
  logic reset;
  modport DUT(input request, reset, clk
              output grant);
  modport TB(output request, reset,
             input clk);
  modport monitor(input request, grant,
                  clk);
end interface

module arbitter(arb_if.DUT aab);
  always_ff @(posedge arb.clk or
              arb.reset) begin
if(arb.reset) begin
    arb.grant <= 2'b00;
end else begin
```

```
case (arb.request)
    2'b01 : arb.grant <= 2'b01;
    2'b10 : arb grant <= 2'b10;
 default : arb.grant <= 2'b00;
 endcase end end
endmodule
```

Test bench -

```
module arb_tb;
    logic clk;
    always #5 clk = ~clk;
arb_if arb_if_inst(clk);
 arbiter dut (arb_if_inst.DUT);
initial begin
    arb_if_inst.TB.rest = 1'b1;
    arb_if_inst.TB.request = 2'b00;

#10
arb_if_inst.TB.reset = 1'b0;
$display("time: %0t\Reset", $time);

#10
arb_if_inst.TB.request = 2'b01;
$display("Time: %0t : Request", $time);

#20
arb_if_inst.TB.request = 2'b10;
$display ("Time: %0t : Reqm2", $time);
#50 $finish
end endmodule
```

Monitor

```
module arb_monitor (arb_if.Monitor arb);
    initial begin
     forever begin
     @(posedg arb.clk);
     if (arb.rqus != 2'b00) begin
      $display("time: %0t \request",
                $time); end
    @posedg (arb.clk);
     if (arb.grant != 2'b00) begin
     $display ("time: %0t grant issued"
                , $time);
     end end end endmodule
```

TOP

```
module arb_top;
    logic clk;
    always #5 clk = ~clk;
    arb_if arb_if_inst(clk);
    arbiter dut (arb_if_inst.DUT);
    arb_tb tb();
    arb_monitor monitor (arb_if_inst.Monit);
endmodule.
```

3. interface arb_if(input logic clk)

```
    logic [1:0] grant;
    logic [1:0] request;
    logic reset;

clocking cb @(posedge clk);
    input #1 request, reset;
    output #2 grant;
endclocking
modport DUT (input cb.request
    cb.reset, output cb.grant);
modport TB (output cb.request, cb.reset,
 modport monitor (input cb.request, cb.grant)
endinterface.

module arbiter (arb_if.DUT arb);
  always @(posedge arb.clk) begin
  if(arb.cb.reset) begin
      arb.cb.grant <= 2'b00;
end else begin
   case (arb.cb.request)
     2'b01 : arb.cb.grant <= 2'b01
     2'b10 : arb.cb.grant <= 2'b10
     default : arb.cb.grant <= 2'b00
endcase end end endmodule

module arb_tb;
    logic clk;
    always #5 clk = ~clk;
arb_if arb_if_inst(clk);
arbiter dut(arb_if_inst.DUT);
initial begin
    arb_if_inst.cb.reset <= 1'b1;
    arb_if_inst.cb.request <= 2'b00;
```

```
#10
arb_if_inst.cb.reset <= 1'b0;
$display ("time: %0t | Reset, $time).

#10
arb_if_inst.cb.request <= 2'b01
 $display ("time: %0t | Request1, $time);

#20
arb_if_inst.cb.request <= 2'b10
$display ("time: %0t | Request2, $time);

#50
$finish
module arb_monitor (arbif.monitor arb);
initial begin
forever begin
@(posedge arb.clk);
if(arb.cb.request) begin
 $display ("time: %0t | request", $time);
@(posedge (arbr.clk);
if arb.cb.grant != 2'b00
 $display ("time: %0t | grant", $time);
TOP module
module arb_top;
 logic clk;
always #5 clk = ~clk;
arb_if arb_if_inst(clk);
arbiter dut (arb_if_inst.DUT);
arb_tb tb(),
arb_monitor monitor (arb_if_inst.monitor)
end module.
```