A blue and white logo

Description automatically generated with medium confidence **San Francisco Bay University**

**EE577 Functional Verification with System Verilog**

**Homework #4**

**Due day: 7/6/2023**

**Instruction:**

1. **Push the answer sheets/source code to Github.**
2. **Please follow the code style rule like programs on handout.**
3. **Overdue homework submission cannot be accepted.**
4. **Takes academic honesty and integrity seriously (Zero Tolerance of Cheating & Plagiarism)**
5. Design a counter with mode 9 (from 0, 1,2 ....8, 0,1,2 ... …) & up-down counting control signal with the function or task structures in Verilog. And then write module-testbench in SV to test “reset” signal, up-down control signal and counting range.

***Program***

*module Counter (*

*input wire clk,*

*input wire reset,*

*input wire up\_down,*

*output reg [3:0] count*

*);*

*always @(posedge clk) begin*

*if (reset) begin*

*count <= 4'b0000; // Reset the counter to 0*

*end else begin*

*if (up\_down) begin*

*if (count == 4'b1001) // When count reaches 9, wrap around to 0*

*count <= 4'b0000;*

*else*

*count <= count + 4'b0001; // Increment the counter*

*end else begin*

*if (count == 4'b0000) // When count reaches 0, wrap around to 9*

*count <= 4'b1001;*

*else*

*count <= count - 4'b0001; // Decrement the counter*

*end*

*end*

*end*

*endmodule*

***Testbench***

*module tb\_Counter;*

*reg clk, reset, up\_down;*

*wire [3:0] count;*

*// Instantiate the Counter module*

*Counter dut (*

*.clk(clk),*

*.reset(reset),*

*.up\_down(up\_down),*

*.count(count)*

*);*

*// Clock generator*

*always begin*

*clk = 0;*

*#5;*

*clk = 1;*

*#5;*

*end*

*initial begin*

*// Test 1: Reset*

*reset = 1;*

*up\_down = 1;*

*#10;*

*reset = 0;*

*#20;*

*// Test 2: Up count*

*reset = 0;*

*up\_down = 1;*

*#90;*

*// Test 3: Down count*

*up\_down = 0;*

*#90;*

*// Test 4: Reset and change counting range*

*reset = 1;*

*#10;*

*reset = 0;*

*up\_down = 1;*

*#90;*

*// Finish simulation*

*$finish;*

*end*

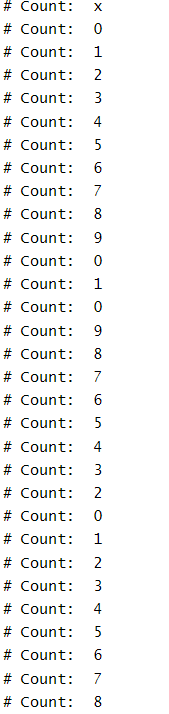
*// Display output*

*always @(posedge clk)*

*$display("Count: %d", count);*

*endmodule*

***Result***



1. Design Verilog module for input bit stream divisible by 7. And then verify it in module-testbench in SV to test “reset” signal and output results for your given sequential input signals, like 1010\_1010.

***Program***

module divisible\_by\_7 (

input wire clk,

input wire reset,

input wire [7:0] din,

output reg out

);

always @(posedge clk or posedge reset) begin

if (reset) begin

out <= 0;

end else begin

out <= (din[0] ^ din[1] ^ din[2] ^ din[3] ^ din[4] ^ din[5] ^ din[6] ^ din[7]);

end

end

endmodule

***Testbench***

module testbench;

reg clk, reset;

reg [7:0] din;

wire out;

divisible\_by\_7 dut (

.clk(clk),

.reset(reset),

.din(din),

.out(out)

);

always #1 clk = ~clk;

initial begin

reset = 1;

din = 8'b1010\_1010;

#2 reset = 0;

end

always @(posedge clk) begin

if (out) begin

$display("Input bit stream is divisible by 7");

end else begin

$display("Input bit stream is not divisible by 7");

end

end

initial begin

#10;

$display("Simulation completed. Exiting...");

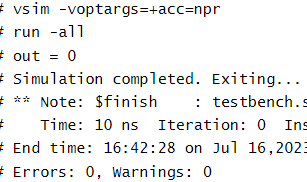
$finish;

end

initial $monitor("out = %b", out);

endmodule

**Result**



1. Generate CRC-4 serial encoder Verilog module with generator polynomial for 8-bits input. After that, write module-testbench in SV to verify “reset”/ “enable” and output signal for given inputs, like “1010\_0101”.

**Program**

module crc4\_encoder (

input wire clk,

input wire reset,

input wire enable,

input wire [7:0] data\_in,

output wire [11:0] crc\_out

);

reg [11:0] crc\_reg;

wire [3:0] feedback;

assign crc\_out = crc\_reg;

always @(posedge clk or posedge reset) begin

if (reset)

crc\_reg <= 0;

else if (enable)

crc\_reg <= {crc\_reg[6:0], data\_in} ^ (feedback << 4);

end

assign feedback = crc\_reg[11:8] ^ 4'b1101;

endmodule

**Testbench**

module crc4\_encoder\_tb;

reg clk;

reg reset;

reg enable;

reg [7:0] data\_in;

wire [11:0] crc\_out;

// Instantiate the crc4\_encoder module

crc4\_encoder dut (

.clk(clk),

.reset(reset),

.enable(enable),

.data\_in(data\_in),

.crc\_out(crc\_out)

);

initial begin

clk = 0;

reset = 1;

enable = 0;

data\_in = 8'b10100101;

#10 reset = 0; // Deassert reset

#5 enable = 1; // Enable CRC calculation

// Wait for some clock cycles

#50;

// Display the CRC output

$display("CRC Out: %b", crc\_out);

// End simulation

$finish;

end

always #5 clk = ~clk;

endmodule

**Running result**

