Digital Design Diploma

Assignment 3 (Extra)

Sequential Logic Design

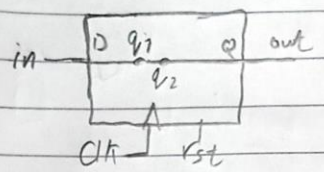
Name	كريم حسن عاطف علي
Group	G2

Submitted to: Eng. Kareem Waseem

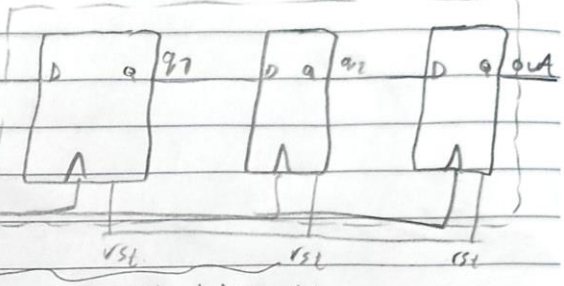
Subject _____ Date _____

Assignment 3 [Extra]

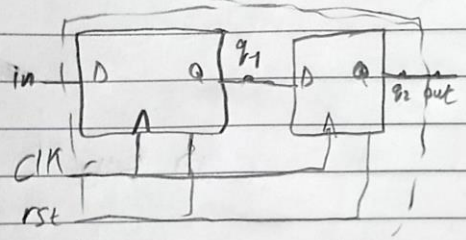
1- reg_blocking1



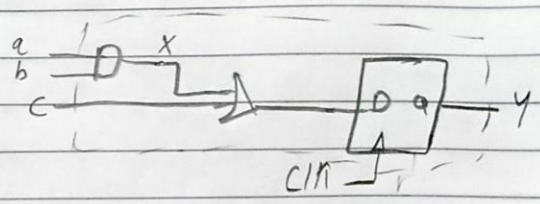
reg_non_blocking1
reg_non_blocking2
reg_blocking3
reg_non_blocking3



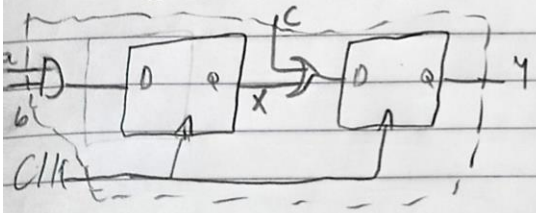
reg_blocking2



2- Comb_blocking1



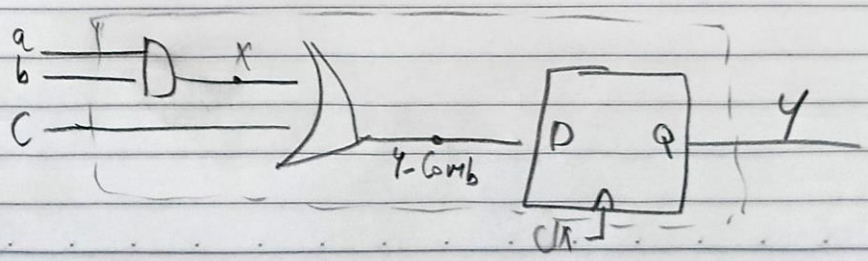
Comb_non_blocking1



Comb_non_blocking2

Comb_blocking2

Comb_non_blocking3



2) 4-bit synchronous counter with asynchronous active low set:

```
4bit_Counter.v > ...
module Counter_4_bits (input clk, set, output reg [3:0] out);

always @(posedge clk or negedge set) begin
    if (!set)           out <= 4'b1111;
    else                out <= out + 1;
end
endmodule
```

Figure1: Q2 Code

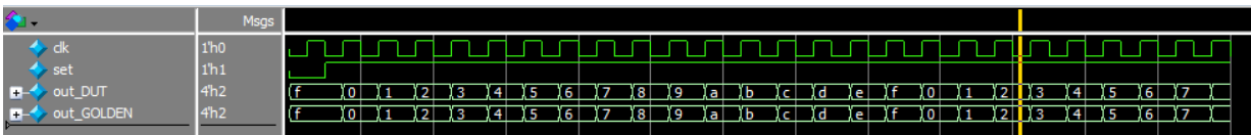


Figure2: Q2 Wave

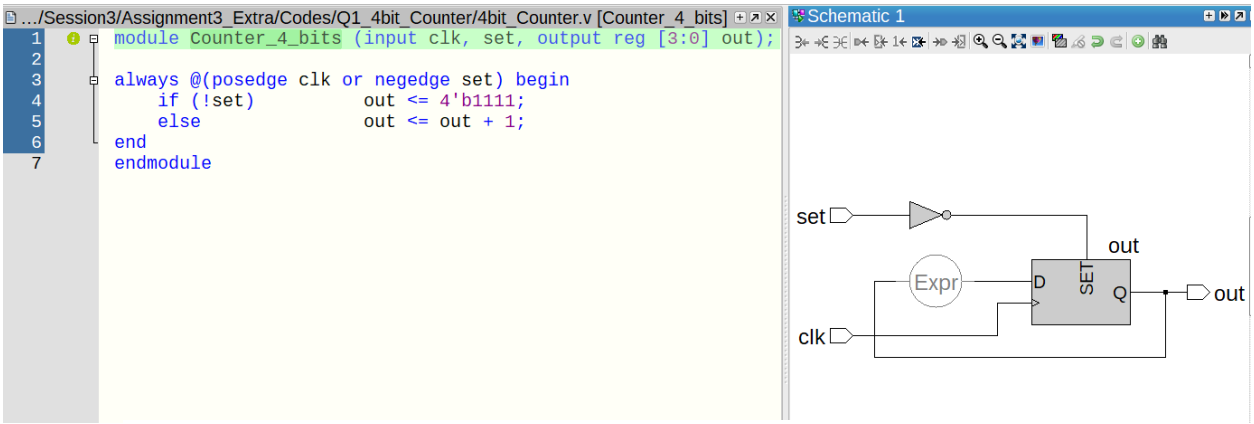


Figure3: Q2 Lint

4bit_Counter_tb.v > Counter_4_bits_tb > set

```
module Counter_4_bits_tb ();
reg clk, set;
wire [3:0] out_DUT,out_GOLDEN;

//module Counter_4_bits (input clk, set, output reg [3:0] out);
Counter_4_bits DUT(clk,set,out_DUT);
//module Ripple_Counter_4bits (input rstn,clk, output [3:0] out);
Ripple_Counter_4bits GOLDEN(set,clk,out_GOLDEN);

initial begin
    clk=0;
    forever #5 clk = ~clk;
end

initial begin
    set=0;
    @(negedge clk);
    if (out_DUT != 4'b1111 || out_DUT != out_GOLDEN) begin
        $display("Initial value is not 1111");
        $stop;
    end
    set=1;
    repeat (25) begin
        @(negedge clk);
        if (out_DUT != out_GOLDEN) begin
        $display("Error");
        $stop;
        end
    end
    $stop;
end
endmodule
```

Figure4: Q2 Testbench

3) Extended Counter:

```
module Counter_4_bits_with_out_clk (input clk, set, output reg [3:0] out, output div_2,div_4);

always @(posedge clk or negedge set) begin
    if (!set)          out <= 4'b1111;
    else               out <= out + 1;
end

assign div_2 = out[0];
assign div_4 = out[1];
endmodule
```

Figure5: Q3 Code

```
module Counter_4_bits_with_out_clk_tb ();
reg clk, set;
wire [3:0] out;
wire div_2,div_4;

//module Counter_4_bits_with_out_clk (input clk, set, output reg [3:0] out, output div_2,div_4);
Counter_4_bits_with_out_clk DUT(clk,set,out,div_2,div_4);


initial begin
    clk=0;
    forever #5 clk = ~clk;
end

integer clk_count=0;
integer div_2_count=0;
integer div_4_count=0;
always @(posedge clk)    clk_count = clk_count + 1;
always @(posedge div_2)  div_2_count = div_2_count + 1;
always @(posedge div_4)  div_4_count = div_4_count + 1;

initial begin
    set = 0;
    @(negedge clk);
    set = 1;
    repeat(100) @(negedge clk);
    if (!((div_2_count >= (clk_count/2 - 1)) && (div_2_count <= (clk_count/2 + 1)))) begin  //Check if div_2 is about half of clk
    $display("Test failed: Clock division by 2 is incorrect.");
    $stop;
    end
    if (!((div_4_count >= (clk_count/4 - 1)) && (div_4_count <= (clk_count/4 + 1)))) begin  //Check if div_4 is about quarter of clk
    $display("Test failed: Clock division by 4 is incorrect.");
    $stop;
    end
    $stop;
end
endmodule
```

Figure6: Q3 Testbench

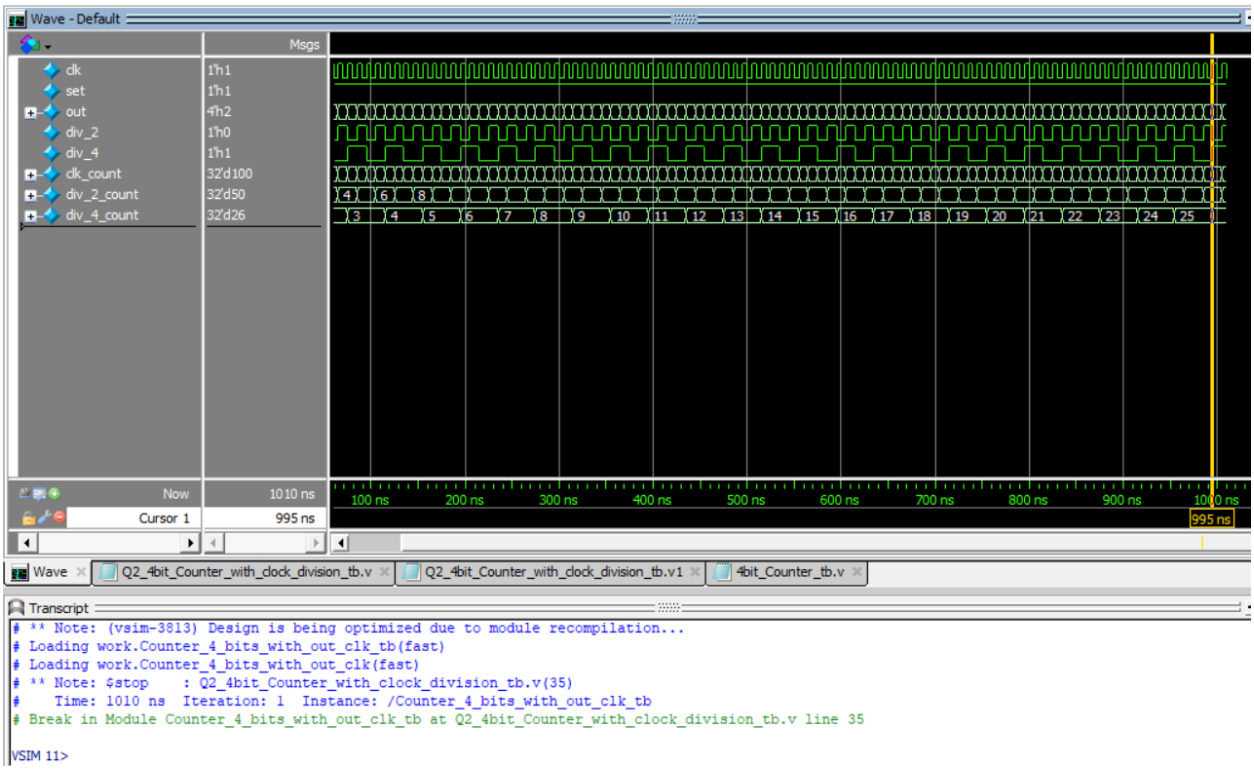


Figure7: Q3 Wave

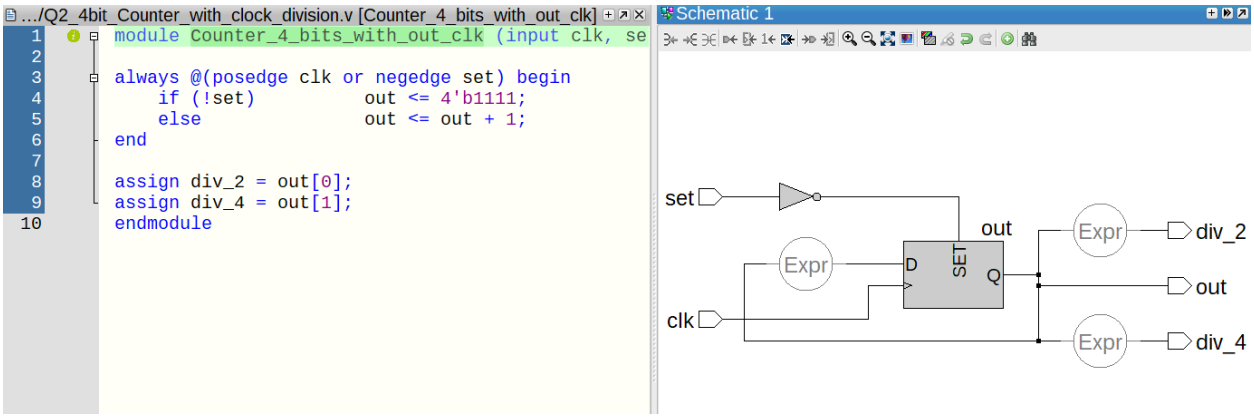


Figure8: Q3 Lint

4) Gray Counter:

```
module Gray_Counter (input clk,rst, output reg [1:0] gray_out);
reg [1:0] binary_out;

always @(posedge clk or posedge rst) begin
    if (rst)       {binary_out,gray_out} <= {2{2'b00}};
    else begin
        binary_out <= binary_out + 1;
        gray_out   <= {binary_out[1] , ^binary_out};
    end
end
endmodule
```

Figure9: Q4 Code

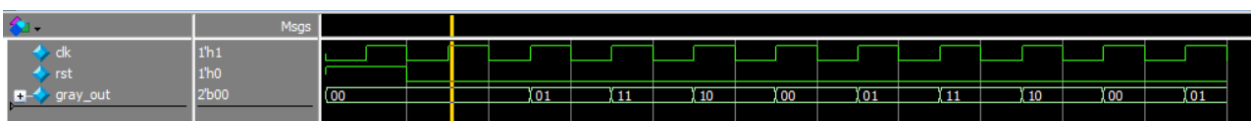


Figure10: Q4 Wave

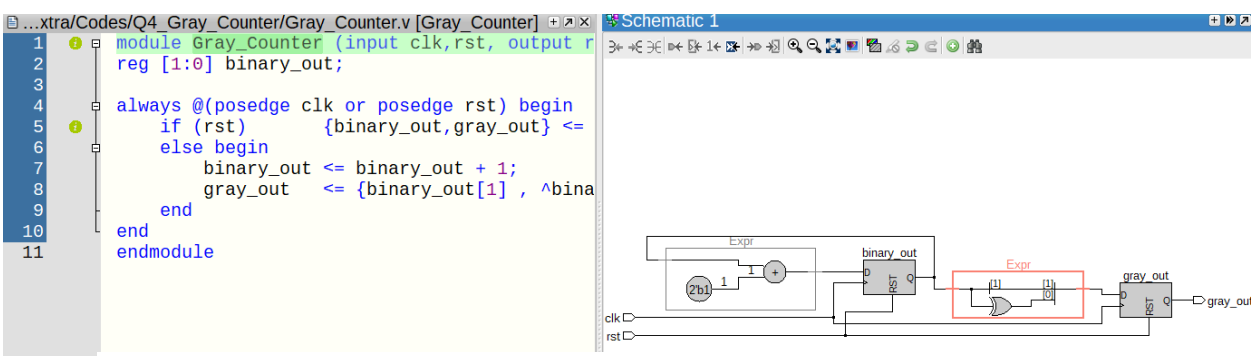


Figure11: Q4 Lint

```
module Gray_Counter_tb ();
reg clk,rst;
wire [1:0] gray_out;

//module Gray_Counter (input clk,rst, output reg [1:0] gray_out);
Gray_Counter DUT(clk,rst,gray_out);

initial begin
    clk=0;
    forever #10 clk = ~clk;
end

initial begin
    rst = 1;
    @(negedge clk);
    rst = 0;
    repeat (10) @(negedge clk);
    $stop;
end
endmodule
```

Figure12: Q4 Testbench