# ELEC4320 Midterm Examination Sample

## Part I: Short Questions

1. Write down the result of B for each of the following Verilog codes after execution:
   i) A <= 7'b1001010; B <= & A;

      B = 1'b0 (unary bit wise and)

   ii) A <= 7'b1001010; B <= A << 3;

      B = 7'b1010000 (left shift A by 3 bits)

   iii) A <= 4'b1001; B <= {2{A}};

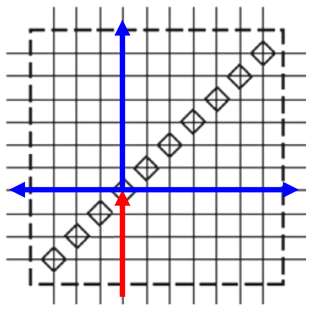
      B = 8'b10011001 (replicate A twice)

2. You are going to implement a system running at 200MHz using a FPGA device. What should be the clock period constraint?

   PERIOD = 5ns

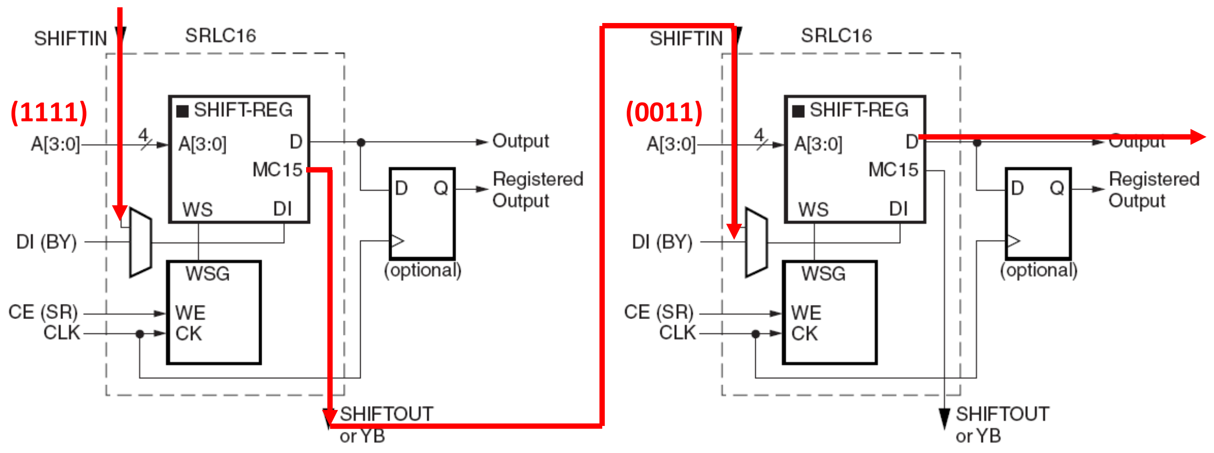
3. What are the advantages of FPGA compared to processor? What are the main steps needed for implementing a design in FPGA?
   FPGA can provide higher performance and lower power consumption than processor. The main steps include programming, pre-simulation, synthesis, implementation and post-simulation.

4. For a switch block in FPGA, what is the meaning of flexibility? Given an example with figure.
   Flexibility defines the number of other wire segments that each wire segment entering the switch block can be connected to.



As shown above, for any particular input, the wire segments can be connected to all 3 other directions (one each), and the flexibility is 3.

5. Below shows two SRLC16 primitive available in the Spartan-II device. Please design (with clear datapath) a 20 bit shift register. State with explanation about how many CLBs are required. You only need to draw arrows and show the connections for the datapath as shown in red below.

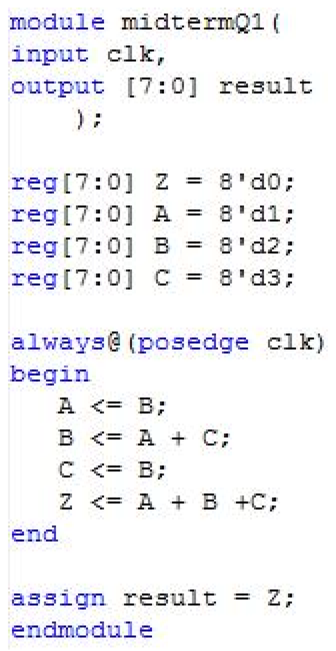


The above shows one possible implementation for a 20-bit shift register. Only 1 CLB is required, as a CLB can implement two SRLC16 primitives.

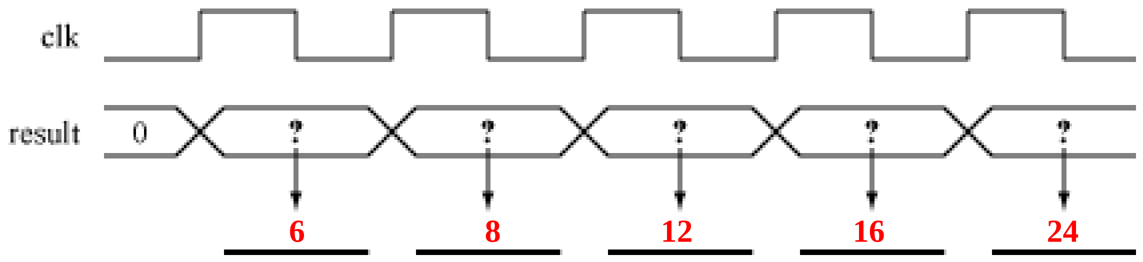
## Part II: Long Questions

### 1. Verilog programming

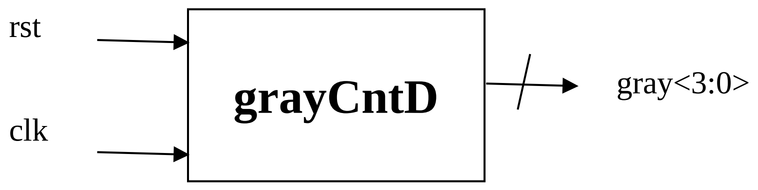
a) Write down the value of Z for the first 5 clock cycles for the given code:



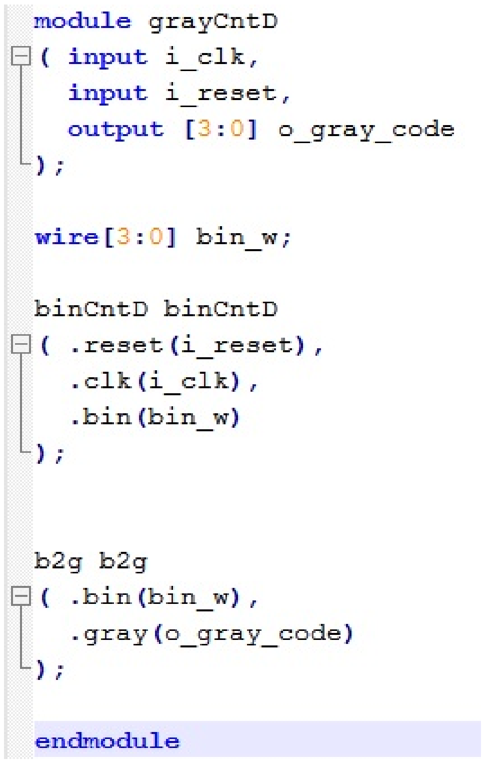
```
module midtermQ1(
input clk,
output [7:0] result
    );

reg[7:0] Z = 8'd0;
reg[7:0] A = 8'd1;
reg[7:0] B = 8'd2;
reg[7:0] C = 8'd3;

always@(posedge clk)
begin
   A <= B;
   B <= A + C;
   C <= B;
   Z <= A + B +C;
end

assign result = Z;
endmodule
```



b) Write up the Verilog code for implementing a 4-bit gray down counter by using two different modules: a 4-bit binary down counter module (**binCntD**) and a binary-to-gray module (**bin2gray**). Below shows the top module of the 4-bit gray down counter (**grayCntD**). Implement the three files: **grayCntD.v, binCntD.v and b2g.v**

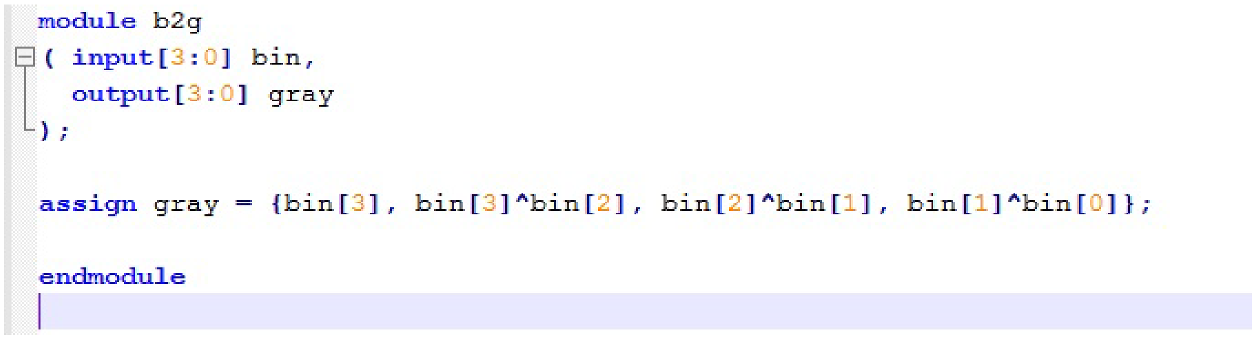


grayCntD.v



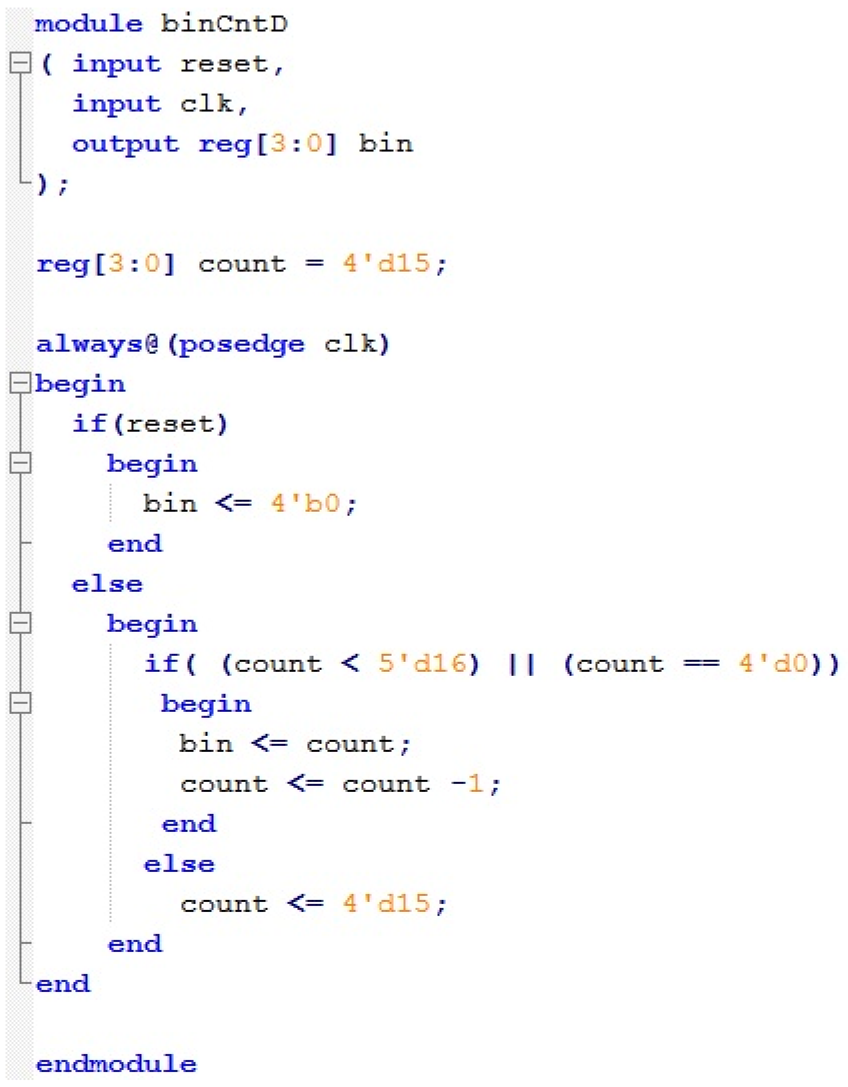
```
module grayCntD
( input i_clk,
  input i_reset,
  output [3:0] o_gray_code
);

wire[3:0] bin_w;

binCntD binCntD
( .reset(i_reset),
  .clk(i_clk),
  .bin(bin_w)
);


b2g b2g
( .bin(bin_w),
  .gray(o_gray_code)
);

endmodule
```

b2g.v



```
module b2g
( input[3:0] bin,
  output[3:0] gray
);

assign gray = {bin[3], bin[3]^bin[2], bin[2]^bin[1], bin[1]^bin[0]};

endmodule
```

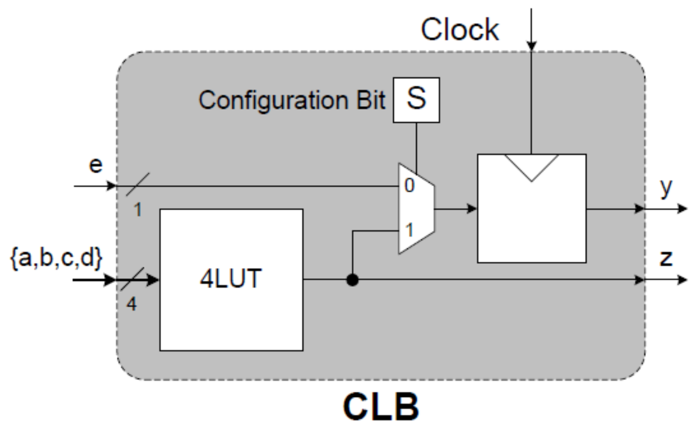
binCntD.v

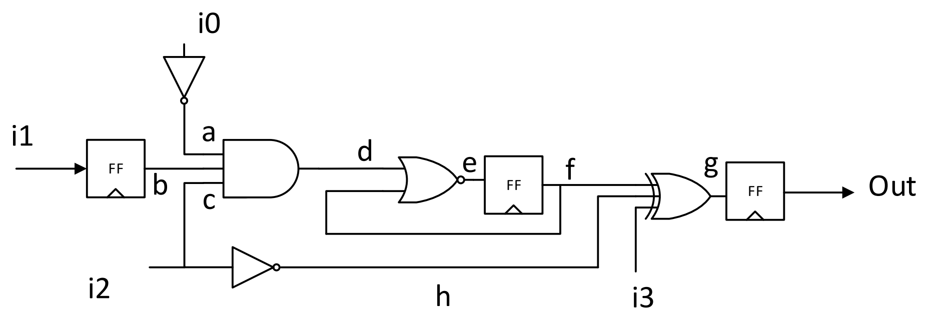


```
module binCntD
( input reset,
  input clk,
  output reg[3:0] bin
);

reg[3:0] count = 4'd15;

always@(posedge clk)
begin
  if(reset)
    begin
      bin <= 4'b0;
    end
  else
    begin
      if( (count < 5'd16) || (count == 4'd0))
       begin
        bin <= count;
        count <= count -1;
       end
      else
        count <= 4'd15;
    end
end

endmodule
```

## 2. Technology Mapping

Assuming a basic CLB as shown below, each CLB features a 4LUT, flip-flop and a mux to decide what input is connected to the flip-flop. Map the circuit shown directly below the CLB into as few of these CLBs as you can. You may connect each CLB input to a signal, logic 0, logic 1, or "nc" (for "no connection"). To answer the question, fill out the table at the bottom.

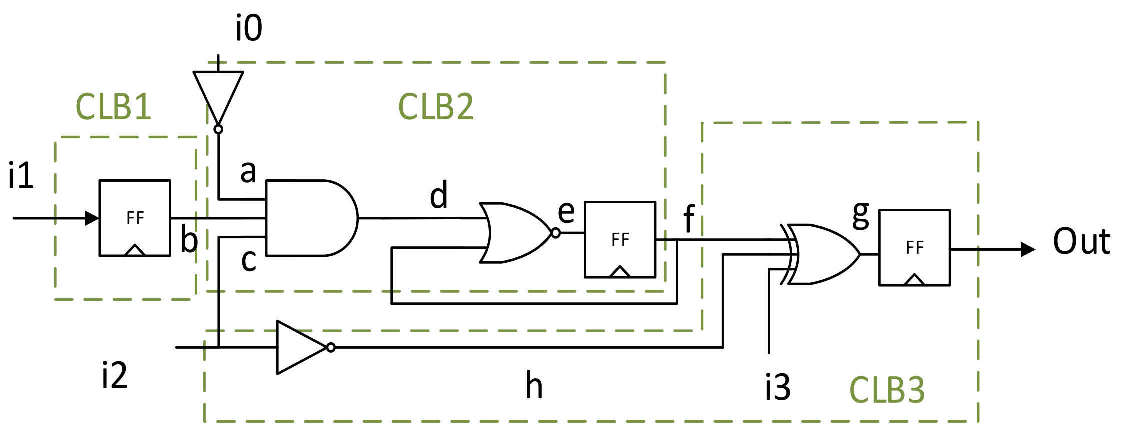




**CLB inputs/outputs/configuration bits**

| | a | b | c | d | e | S | y | z |
|---|---|---|---|---|---|---|---|---|
| CLB1 | | | | | | | | |
| CLB2 | | | | | | | | |
| …. | | | | | | | | |

Solution: The partition of the circuit is shown as follows.



The input/output of each CLB is shown in the table.

| | a | b | c | d | e | S | y | z |
|---|---|---|---|---|---|---|---|---|
| CLB1 | | | | | i1 | 0 | b | |
| CLB2 | i0 | b | i2 | f | | 1 | f | |
| CLB3 | f | i2 | i3 | | | 1 | Out | |

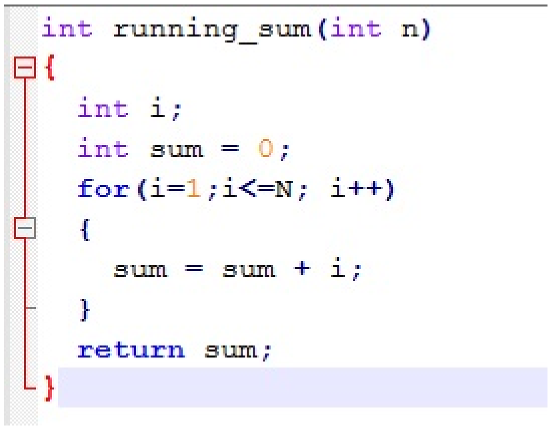
## 3. High Level Synthesis

A function as shown in the code below is provided. It calculates the sum of first ‘n’ positive integers. Since ‘n’ is not known at compile time, none of the loop related optimizations can be applied. However, if we assuming that ‘n’ is always greater than 6, it is possible to apply at least one loop optimization.

a) Can you suggest which optimization this might be?

b) Can you rewrite the for-loop code in the optimized version?

c) Can you suggest why the restructured for loop is better? Can you suggest any other optimization?

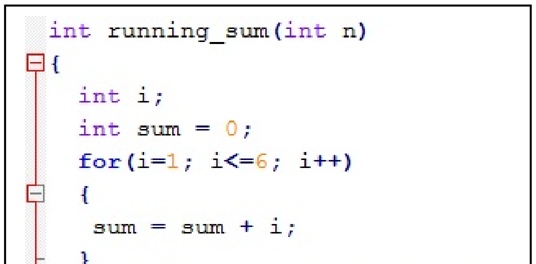


```
int running_sum(int n)
{
   int i;
   int sum = 0;
   for(i=1;i<=N; i++)
   {
     sum = sum + i;
   }
   return sum;
}
```

Fig. Given Code

a) The loop optimization is “Loop Fission”

b)



```
int running_sum(int n)
{
   int i;
   int sum = 0;
   for(i=1; i<=6; i++)
   {
    sum = sum + i;
   }
```

c) The restructured for-loop as shown in b) is better because one can further unroll the first loop ( i = 1 to 6)as a result of which its body can be executed in 1 clock cycle instead of 6 clock cycles. It is also possible to keep only the second loop (i = 7 to N) and initialize sum with the sum of first 6 integers (i.e 1+2+3+4+5+6 = 21) instead of zero.

**END OF SAMPLE PAPER**