भारतीय प्रौद्योगिकी संस्थान गांधीनगर
पालज, गांधीनगर, गुजरात 382 055
INDIAN INSTITUTE OF TECHNOLOGY GANDHINAGAR
PALAJ, GANDHINAGAR, GUJARAT 382 055

Name : Bhavik Patel
Roll no. : 22110047

IITGN

Digital Systems
LAB Assignment-1

Question 1) Smart Math Tutor Hardware

You are to design a hardware that can be used by Kindergarten kids to understand multiples of different numbers.

Write a Verilog code that can implement multiples of 2,3, 4, 5,6, 7, 8, 9. Allow the user to decide the number whose multiple s/he wants to find out. You can use "*select line"* to select the number whose multiple has to be found.

You need to do this for 5-bit input (which decides the range within which all the multiples have to be found.)

Module has 5-bit input, select inputs and ONE output signal. (Don't write separate Verilog codes for each number whose multiples are to be found.)

Write the code using **always** block.

Next, assume that you have a large 5x32 decoder. Implement the same question using only decoder and OR gates. You can write the code using any of structural/continuous assignment/procedural coding.

Compare and comment on the simplicity of the two approaches.

```
`timescale 1ns / 1ps

module lab1_multiple(select, num, out );
input [2:0]select;
input [4:0]num;
output reg out;

reg a,b,c,d,e;
always @(*)
begin
a=num[4]; b=num[3]; c=num[2]; d=num[1]; e=num[0];
    case (select)
        3'b000: out = ~e;

        3'b001: out = ((~a)&(~b)&(~c)&(~d)&(~e)) | ((~a)&(~b)&(~c)&d&e) |((~a)&(~b)&c&d&(~e))
                    |((~a)&b&(~c)&(~d)&e) |((~a)&b&c&d&e) |((~a)&b&c&(~d)&(~e))
                    |(a&b&(~c)&(~d)&(~e)) |(a&b&(~c)&d&e) |(a&b&c&d&(~e)) |(a&(~b)&(~c)&d&(~e))
                    |(a&(~b)&c&(~d)&e) ;

        3'b010: out = (~d) & (~e);

        3'b011: out = ((~a)&(~b)&(~c)&(~d)&(~e)) |((~a)&(~b)&c&(~d)&e) |((~a)&b&(~c)&d&(~e))
                    |((~a)&b&c&d&e) |(a&b&(~c)&(~d)&e) |(a&b&c&d&(~e)) |(a&(~b)&c&(~d)&(~e)) ;

        3'b100: out = ((~a)&(~b)&(~c)&(~d)&(~e)) |((~a)&(~b)&c&d&(~e)) |((~a)&b&c&(~d)&(~e))
                    |(a&b&(~c)&(~d)&(~e)) |(a&b&c&d&(~e)) |(a&(~b)&(~c)&d&(~e)) ;

        3'b101: out = ((~a)&(~b)&(~c)&(~d)&(~e)) |((~a)&(~b)&c&d&e) |((~a)&b&c&d&(~e))
                    |(a&b&c&(~d)&(~e)) |(a&(~b)&c&(~d)&e);

        3'b110: out = (~c)&(~d)&(~e);

        3'b111: out = ((~a)&(~b)&(~c)&(~d)&(~e)) |((~a)&b&(~c)&(~d)&e) |(a&b&(~c)&d&e) |(a&(~b)&(~c)&d&(~e)) ;
        default: out = 5'b0;
    endcase

end
endmodule
```

Code for question 1

```
`timescale 1ns / 1ps

module lab1_multiple(select, num, out );
input [2:0]select;
input [4:0]num;
output reg out;
reg a,b,c,d,e;
always @(*)
begin
a=num[4]; b=num[3]; c=num[2]; d=num[1]; e=num[0];
case (select)
3'b000: out = ~num[0];

3'b001: out = ((~a)&(~b)&(~c)&(~d)&(~e)) | ((~a)&(~b)&(~c)&d&e) |((~a)&(~b)&c&d&(~e))
|((~a)&b&(~c)&(~d)&e) |((~a)&b&c&d&e) |((~a)&b&c&(~d)&(~e)) |(a&b&(~c)&(~d)&(~e))
|(a&b&(~c)&d&e) |(a&b&c&d&(~e)) |(a&(~b)&(~c)&d&(~e)) |(a&(~b)&c&(~d)&e) ;
```

```
3'b010: out = (~d) & (~e);

3'b011: out = ((~a)&(~b)&(~c)&(~d)&(~e)) |((~a)&(~b)&c&(~d)&e) |((~a)&b&(~c)&d&(~e))
|((~a)&b&c&d&e) |(a&b&(~c)&(~d)&e) |(a&b&c&d&(~e)) |(a&(~b)&c&(~d)&(~e)) ;

3'b100: out = ((~a)&(~b)&(~c)&(~d)&(~e)) |((~a)&(~b)&c&d&(~e)) |((~a)&b&c&(~d)&(~e))
|(a&b&(~c)&(~d)&(~e)) |(a&b&c&d&(~e)) |(a&(~b)&(~c)&d&(~e)) ;

3'b101: out = ((~a)&(~b)&(~c)&(~d)&(~e)) |((~a)&(~b)&c&d&e) |((~a)&b&c&d&(~e)) |(a&b&c&(~d)&(~e))
|(a&(~b)&c&(~d)&e);

3'b110: out = (~c)&(~d)&(~e);

3'b111: out = ((~a)&(~b)&(~c)&(~d)&(~e)) |((~a)&b&(~c)&(~d)&e) |(a&b&(~c)&d&e)
|(a&(~b)&(~c)&d&(~e)) ;

default: out = 5'b0;
endcase

end
endmodule
```

NOTE: For this question, we assume that the select line corresponding to 3’b000 represents the number 2 and 3’b001 represents the number 3, and so on. Therefore, 3’b000=2, 3’b001=3, 3’b010=4, 3’b011=5, 3’b100=6, 3’b101=7, 3’b110=8 and 3’b111=9

```
`timescale 1ns / 1ps

module lab1_tb();
reg [4:0]num;
reg [2:0] select;
wire out;

lab1_multiple uut(select,num,out);


initial
begin
select=3'b010;
num=5'b00000;
#10
num=5'b00001;
#10
num=5'b00010;
#10
num=5'b00011;
#10
num=5'b00100;
#10
num=5'b00101;
#10
num=5'b00110;
#10
num=5'b00111;
#10
num=5'b01000;
#10
#10
num=5'b10011;
#10
num=5'b10100;
#10
num=5'b10101;
#10
num=5'b10110;
#10
num=5'b10111;
#10
num=5'b11000;
#10
num=5'b11001;
#10
num=5'b11010;
#10
num=5'b11011;
#10
num=5'b11100;
#10
num=5'b11101;
#10
num=5'b11110;
#10
num=5'b11111;
#10
$finish();
end
endmodule
```

Test Bench

`timescale 1ns / 1ps

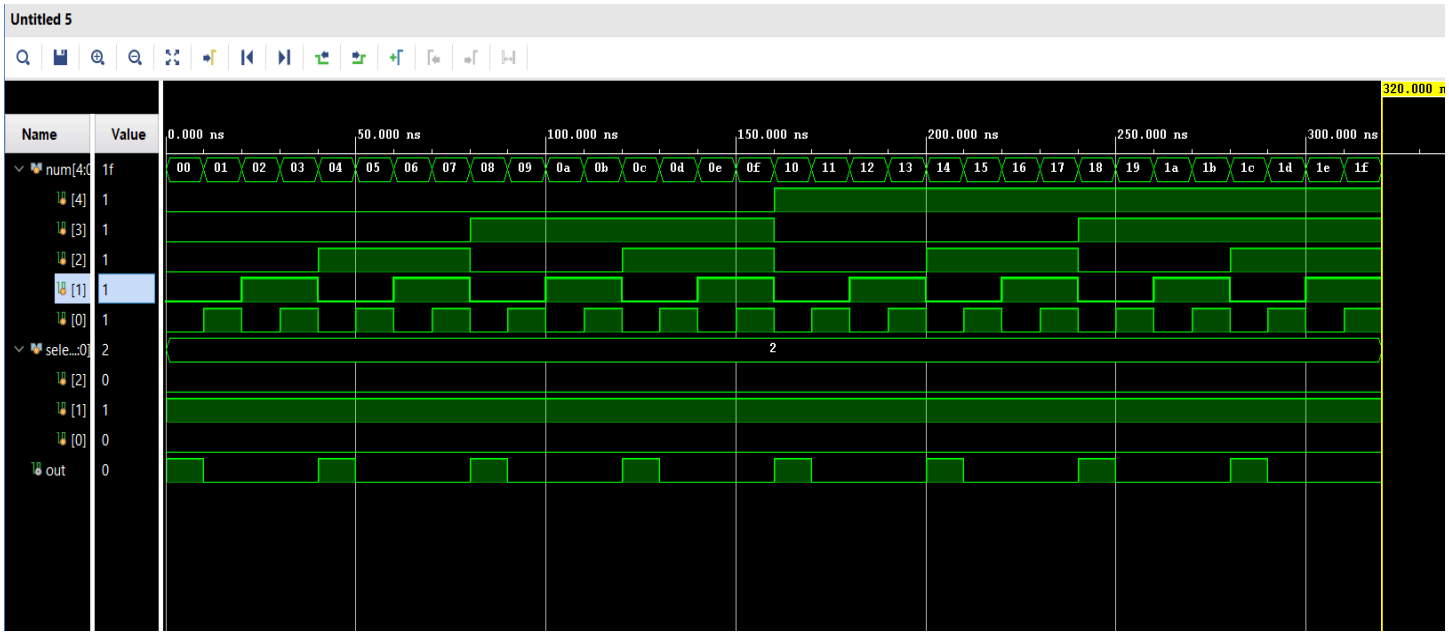
module lab1_tb();
reg [4:0]num;
reg [2:0] select;
wire out;

lab1_multiple uut(select,num,out);

initial
begin
select=3'b010;
num=5'b00000;
#10
num=5'b00001;
#10
num=5'b00010;
#10
num=5'b00011;
#10
num=5'b00100;
#10
num=5'b00101;
#10
num=5'b00110;
#10
num=5'b00111;
#10
num=5'b01000;
#10

```
num=5'b01001;
#10
num=5'b01010;
#10
num=5'b01011;
#10
num=5'b01100;
#10
num=5'b01101;
#10
num=5'b01110;
#10
num=5'b01111;
#10
num=5'b10000;
#10
num=5'b10001;
#10
num=5'b10010;
#10
num=5'b10011;
#10
num=5'b10100;
#10
num=5'b10101;
#10
num=5'b10110;
#10
num=5'b10111;
#10
num=5'b11000;
#10
num=5'b11001;
#10
num=5'b11010;
#10
num=5'b11011;
#10
num=5'b11100;
#10
num=5'b11101;
#10
num=5'b11110;
#10
num=5'b11111;
#10
$finish();
end
endmodule
```

Simulation Result(This simulation identifies multiple of 4 given select as 3’b010)



Question 2)

Assume that you have a large 5x32 decoder. Implement the same question using only decoder and OR gates. You can write the code using any of structural/continuous assignment/procedural coding.

```
`timescale 1ns / 1ps

module lab1_decoder(select, num, out );
input [2:0]select;
input [4:0]num;
output reg out;
reg [9:2]decoder3x8;
reg [31:0]decoder5x32;
reg mult2, mult3, mult4, mult5, mult6, mult7, mult8, mult9,
    result2, result3, result4, result5, result6, result7, result8, result9;


always@(*) begin
    out=1'b0;
    decoder3x8=8'b0;
    decoder5x32=32'b0;
    case(select)
        3'b000: decoder3x8[2] =1;
        3'b001: decoder3x8[3] =1;
        3'b010: decoder3x8[4] =1;
        3'b011: decoder3x8[5] =1;
        3'b100: decoder3x8[6] =1;
        3'b101: decoder3x8[7] =1;
        3'b110: decoder3x8[8] =1;
        3'b111: decoder3x8[9] =1;
        default: decoder3x8 = 8'b0;
    endcase

    case(num)
        32'b00000: decoder5x32[0] = 1;
        32'b00001: decoder5x32[1] = 1;
        32'b00010: decoder5x32[2] = 1;
        32'b00011: decoder5x32[3] = 1;
        32'b00100: decoder5x32[4] = 1;
        32'b00101: decoder5x32[5] = 1;
        32'b00110: decoder5x32[6] = 1;
        32'b00111: decoder5x32[7] = 1;
        32'b01000: decoder5x32[8] = 1;
        32'b01001: decoder5x32[9] = 1;
        32'b01010: decoder5x32[10] = 1;
        32'b01011: decoder5x32[11] = 1;
        32'b01100: decoder5x32[12] = 1;
        32'b01101: decoder5x32[13] = 1;
        32'b01110: decoder5x32[14] = 1;
        32'b01111: decoder5x32[15] = 1;
        32'b10000: decoder5x32[16] = 1;
        32'b10001: decoder5x32[17] = 1;
        32'b10010: decoder5x32[18] = 1;
        32'b10011: decoder5x32[19] = 1;
        32'b10100: decoder5x32[20] = 1;
        32'b10101: decoder5x32[21] = 1;
        32'b10110: decoder5x32[22] = 1;
        32'b10111: decoder5x32[23] = 1;
        32'b11000: decoder5x32[24] = 1;
        32'b11001: decoder5x32[25] = 1;
        32'b11010: decoder5x32[26] = 1;
        32'b11011: decoder5x32[27] = 1;
        32'b11100: decoder5x32[28] = 1;
        32'b11101: decoder5x32[29] = 1;
        32'b11110: decoder5x32[30] = 1;
```

```
mult2=decoder5x32[2] | decoder5x32[4] | decoder5x32[6] | decoder5x32[8] | decoder5x32[10] | decoder5x32[12] | decoder5x32[14] | decoder5x32[16] | d
mult3=decoder5x32[3] |decoder5x32[6] | decoder5x32[9] | decoder5x32[12] | decoder5x32[15] | decoder5x32[18] | decoder5x32[21] | decoder5x32[24] | d
mult4=decoder5x32[4] | decoder5x32[8] | decoder5x32[12] | decoder5x32[16] | decoder5x32[20] | decoder5x32[24] | decoder5x32[28];
mult5=decoder5x32[5] | decoder5x32[10] | decoder5x32[15] | decoder5x32[20] | decoder5x32[25] | decoder5x32[30] ;
mult6=decoder5x32[6] | decoder5x32[12] | decoder5x32[18] | decoder5x32[24] | decoder5x32[30] ;
mult7=decoder5x32[7] | decoder5x32[14] | decoder5x32[21] | decoder5x32[28] ;
mult8=decoder5x32[8] | decoder5x32[16] | decoder5x32[24] ;
mult9=decoder5x32[9] | decoder5x32[18] | decoder5x32[27] ;

result2 = mult2 & decoder3x8[2];
result3 = mult3 & decoder3x8[3];
result4 = mult4 & decoder3x8[4];
result5 = mult5 & decoder3x8[5];
result6 = mult6 & decoder3x8[6];
result7 = mult7 & decoder3x8[7];
result8 = mult8 & decoder3x8[8];
result9 = mult9 & decoder3x8[9];

out= result2 | result3 | result4 | result5 | result6 | result7 | result8 | result9 ;
end
```

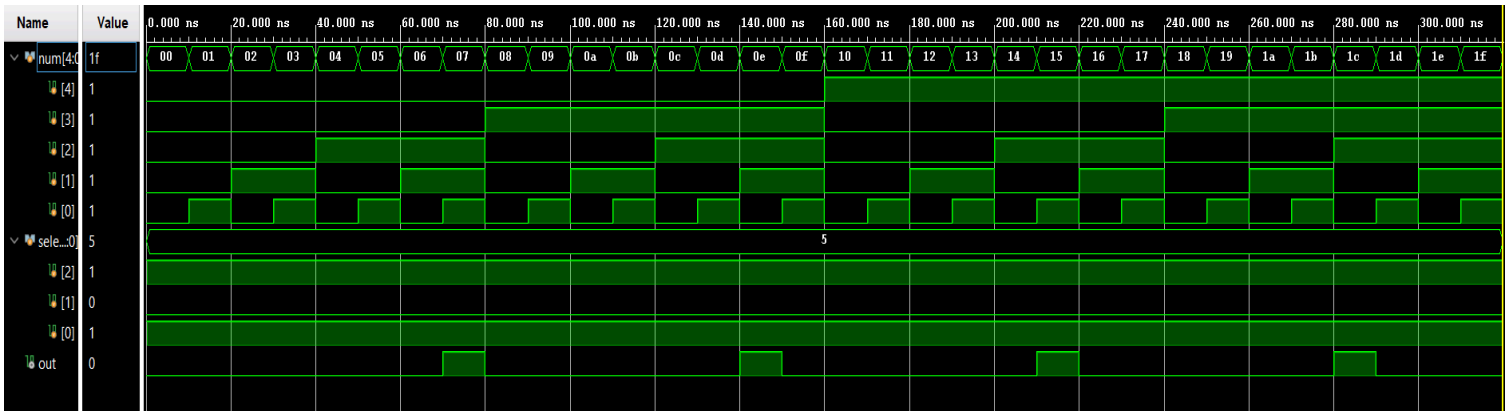
```
`timescale 1ns / 1ps

module lab1_decoder(select, num, out );
input [2:0]select;
input [4:0]num;
output reg out;
reg [9:2]decoder3x8;
reg [31:0]decoder5x32;
reg mult2, mult3, mult4, mult5, mult6, mult7, mult8, mult9,
    result2, result3, result4, result5, result6, result7, result8, result9;


always@(*) begin
    out=1'b0;
    decoder3x8=8'b0;
    decoder5x32=32'b0;
    case(select)
        3'b000: decoder3x8[2] =1;
        3'b001: decoder3x8[3] =1;
        3'b010: decoder3x8[4] =1;
        3'b011: decoder3x8[5] =1;
        3'b100: decoder3x8[6] =1;
        3'b101: decoder3x8[7] =1;
        3'b110: decoder3x8[8] =1;
        3'b111: decoder3x8[9] =1;
        default: decoder3x8 = 8'b0;
    endcase
    case(num)
        32'b00000: decoder5x32[0] = 1;
        32'b00001: decoder5x32[1] = 1;
        32'b00010: decoder5x32[2] = 1;
        32'b00011: decoder5x32[3] = 1;
        32'b00100: decoder5x32[4] = 1;
        32'b00101: decoder5x32[5] = 1;
        32'b00110: decoder5x32[6] = 1;
        32'b00111: decoder5x32[7] = 1;
        32'b01000: decoder5x32[8] = 1;
        32'b01001: decoder5x32[9] = 1;
        32'b01010: decoder5x32[10] = 1;
        32'b01011: decoder5x32[11] = 1;
        32'b01100: decoder5x32[12] = 1;
        32'b01101: decoder5x32[13] = 1;
        32'b01110: decoder5x32[14] = 1;
        32'b01111: decoder5x32[15] = 1;
        32'b10000: decoder5x32[16] = 1;
        32'b10001: decoder5x32[17] = 1;
```

```
        32'b10010: decoder5x32[18] = 1;
        32'b10011: decoder5x32[19] = 1;
        32'b10100: decoder5x32[20] = 1;
        32'b10101: decoder5x32[21] = 1;
        32'b10110: decoder5x32[22] = 1;
        32'b10111: decoder5x32[23] = 1;
        32'b11000: decoder5x32[24] = 1;
        32'b11001: decoder5x32[25] = 1;
        32'b11010: decoder5x32[26] = 1;
        32'b11011: decoder5x32[27] = 1;
        32'b11100: decoder5x32[28] = 1;
        32'b11101: decoder5x32[29] = 1;
        32'b11110: decoder5x32[30] = 1;
        32'b11111: decoder5x32[31] = 1;
        default: decoder5x32=31'b0;
    endcase

    mult2=decoder5x32[2] | decoder5x32[4] | decoder5x32[6] | decoder5x32[8] | decoder5x32[10] | decoder5x32[12] | decoder5x32[14] | decoder5x32[16] | decoder5x32[18]
|decoder5x32[20] | decoder5x32[22] | decoder5x32[24] | decoder5x32[26] | decoder5x32[28] | decoder5x32[30] ;
    mult3=decoder5x32[3] |decoder5x32[6] | decoder5x32[9] | decoder5x32[12] | decoder5x32[15] | decoder5x32[18] | decoder5x32[21] | decoder5x32[24] | decoder5x32[27] |
decoder5x32[30] ;
    mult4=decoder5x32[4] | decoder5x32[8] | decoder5x32[12] | decoder5x32[16] | decoder5x32[20] | decoder5x32[24] | decoder5x32[28];
    mult5=decoder5x32[5] | decoder5x32[10] | decoder5x32[15] | decoder5x32[20] | decoder5x32[25] | decoder5x32[30] ;
    mult6=decoder5x32[6] | decoder5x32[12] | decoder5x32[18] | decoder5x32[24] | decoder5x32[30] ;
    mult7=decoder5x32[7] | decoder5x32[14] | decoder5x32[21] | decoder5x32[28] ;
    mult8=decoder5x32[8] | decoder5x32[16] | decoder5x32[24] ;
    mult9=decoder5x32[9] | decoder5x32[18] | decoder5x32[27] ;

    result2 = mult2 & decoder3x8[2];
    result3 = mult3 & decoder3x8[3];
    result4 = mult4 & decoder3x8[4];
    result5 = mult5 & decoder3x8[5];
    result6 = mult6 & decoder3x8[6];
    result7 = mult7 & decoder3x8[7];
    result8 = mult8 & decoder3x8[8];
    result9 = mult9 & decoder3x8[9];

    out= result2 | result3 | result4 | result5 | result6 | result7 | result8 | result9 ;
    end
endmodule
```

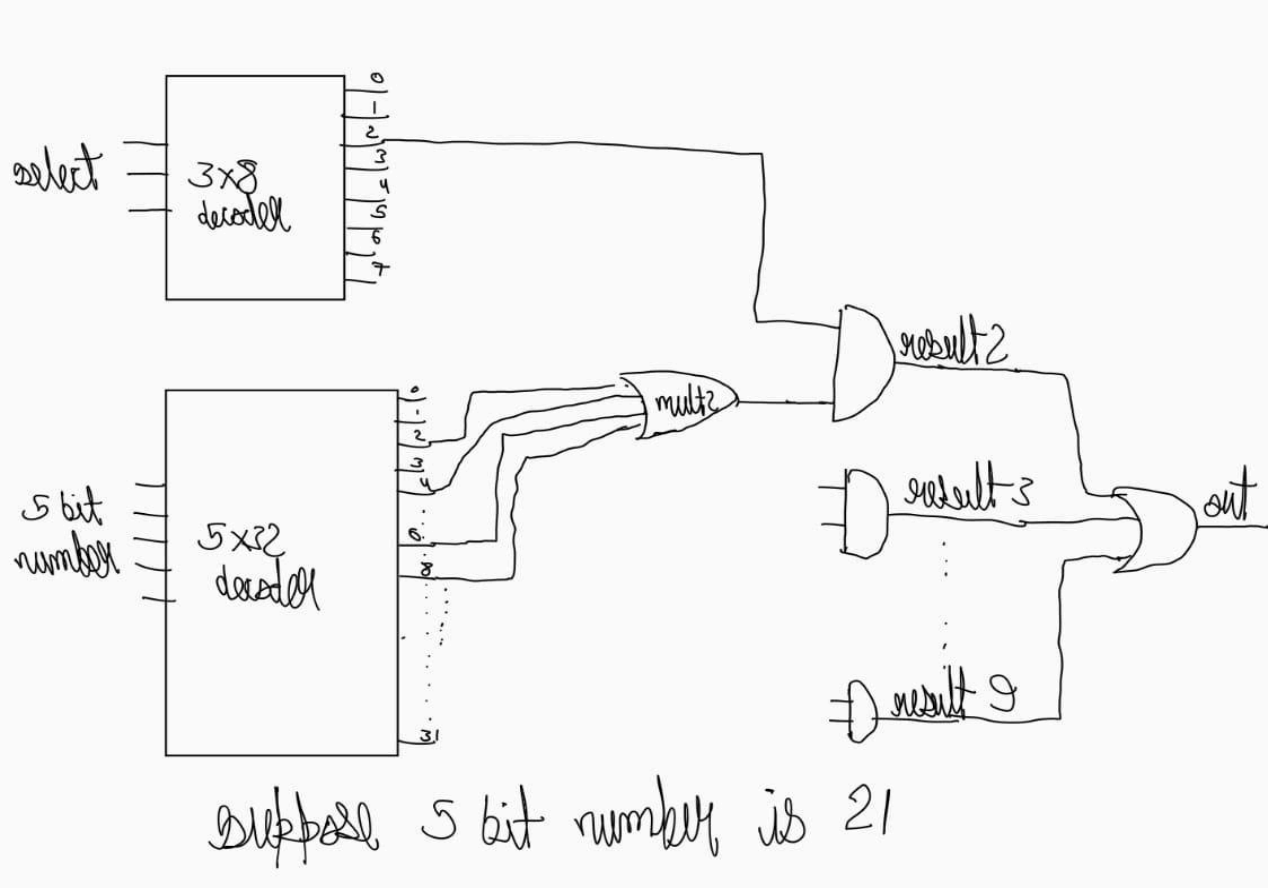
Test Bench:

The test bench will be similar to that shown above in question 1.

Simulation Result:



In the above simulation, select is 3’b101 which is 5 as shown in the image but we said earlier that for this question we assume 3’b101 = 7. Hence, we can see that our code correctly identifies the multiple of 7 and have made they high keeping other values low.



Approach 1 (Using always block): This approach uses a case statement inside an always block to compute the multiples directly based on the selected input. It is straightforward and easy to understand.

Approach 2 (Using Decoder and OR gates): This approach involves using a decoder to create individual signals for each multiple and then combining OR gates.