ASSIGNMENT: 1

ES 204: DIGITAL SYSTEMS

Prof. Joycee Mekie

Indian Institute of Technology, Gandhinagar

Aditya N, Mehta

22110017

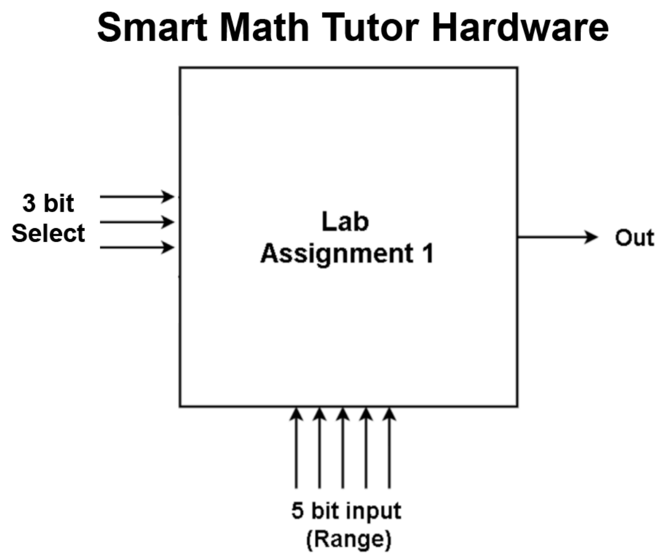
Hrriday V. Ruparel

22110099

Table of Contents

1. Introduction 2
2. Implementation without Decoder 3
    2.1 Code 3
    2.2 Test Bench 4
    2.3 Stimulation 6
3. Implementation with Decoder 7
    3.1 Code: 7
    3.2 Test Bench 8
    3.3 Stimulation: 10
4. Conclusion and Learnings 11

1. Introduction



This hardware design project targets Kindergarten children, offering an engaging module to comprehend multiples of numbers 2 through 9. Utilising Verilog code, an "always" block manages a 5-bit input, enabling users to select their desired number (using 3-bit select lines) and range for multiple explorations. Additionally, an alternative approach employs a 5x32 decoder and OR gates for the same functionality. Our report consists of the Verilog codes, Test benches and Simulation results as well as thoroughly assesses and compares the simplicity of these two methods, emphasising their effectiveness and ease of use.

2. Implementation without Decoder

2.1 Code

```
`timescale 1ns / 1ps
/*

Sel: (Encoding)
2 <- 000
3 <- 001
4 <- 010
5 <- 011
6 <- 100
7 <- 101
8 <- 110
9 <- 111

Number:
Convert decimal representation of any number from 0 to 31 (both inclusive) to 5-bit binary (a5,a4,a3,a2,a1,a0)
a5 is the MSB

ismultiple = 1, if Number (from 0 to 31 both inclusive) is multiple of the number represented by input Sel
ismultiple = 0, elsewhere

*/
module Lab_Assignment_1_Without_Decoder(sel,number,ismultiple);
input [2:0]sel;
input [4:0]number;
output reg ismultiple;
   always @ (*)
   begin
       if (sel == 3'b000 & number[0]==0)
       begin ismultiple = 1; end

       else if (sel == 3'b001 & ((number == 5'b00000) | (number == 5'b00011) | (number == 5'b00110) | (number
== 5'b01001) | (number == 5'b01100) | (number == 5'b01111) | (number == 5'b10010)
       | (number == 5'b10101) | (number == 5'b11000) | (number == 5'b11011) | (number == 5'b11110)))
       begin ismultiple = 1; end

       else if (sel == 3'b010 & (number[1:0] == 2'b00))
       begin ismultiple = 1; end

       else if (sel == 3'b011 & ((number == 5'b00000) | (number == 5'b00101) | (number == 5'b01010) | (number
== 5'b01111) | (number == 5'b10100) | (number == 5'b11001) | (number == 5'b11110)))
       begin ismultiple = 1; end

       else if (sel == 3'b100 & ((number == 5'b00000) | (number == 5'b00110) | (number == 5'b01100) | (number
== 5'b10010) | (number == 5'b11000) | (number == 5'b11110)))
       begin ismultiple = 1; end

       else if (sel == 3'b101 & ((number == 5'b00000) | (number == 5'b00111) | (number == 5'b01110) | (number
== 5'b10101) | (number == 5'b11100)))
       begin ismultiple = 1; end

       else if (sel == 3'b110 & (number[2:0] == 3'b000))
       begin ismultiple = 1; end

       else if (sel == 3'b111 & ((number == 5'b00000) | (number == 5'b01001) | (number == 5'b10010) | (number
== 5'b11011)))
       begin ismultiple = 1; end

       else
       begin ismultiple = 0; end
   end
Endmodule
```

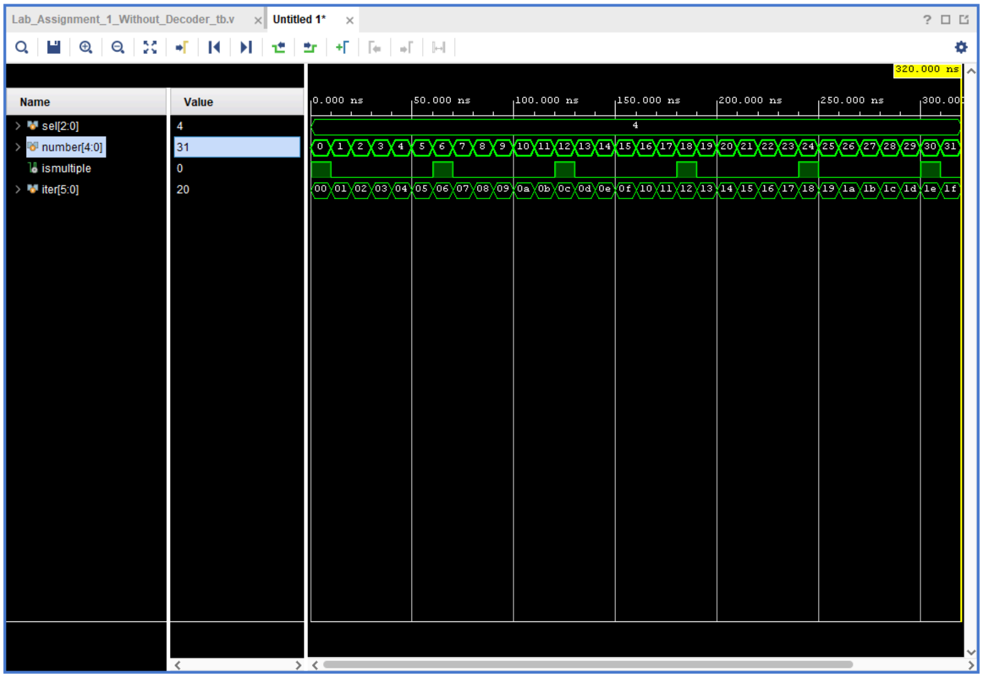
2.2 Test Bench

```
`timescale 1ns / 1ps

/*

sel: (Encoding)
2 <- 000
3 <- 001
4 <- 010
5 <- 011
6 <- 100
7 <- 101
8 <- 110
9 <- 111

number:
Convert decimal representation of any number from 0 to 31 (both inclusive) to 5-bit binary (a5,a4,a3,a2,a1,a0)
a5 is the MSB

ismultiple = 1, if Number (from 0 to 31 both inclusive) is multiple of the number represented by input Sel
ismultiple = 0, elsewhere


*/


module Lab_Assignment_1_Without_Decoder_tb();
reg [2:0]sel;
reg [4:0]number;
wire ismultiple;
//reg [5:0]iter; //Uncomment this when using type I


Lab_Assignment_1_Without_Decoder uut(sel,number,ismultiple);

initial
Begin


/*
Two types of test benches:
I) Find all the multiples of a number
II) Find whether a random number is a multiple of another number corresponding to the select.
*/


// I) Uncommment this for 1st type
//for(iter=6'b00000;iter<=6'b011111;iter=iter+6'b000001)
//begin
//sel = 3'b111; number = iter[4:0];
//#10;
//end
```

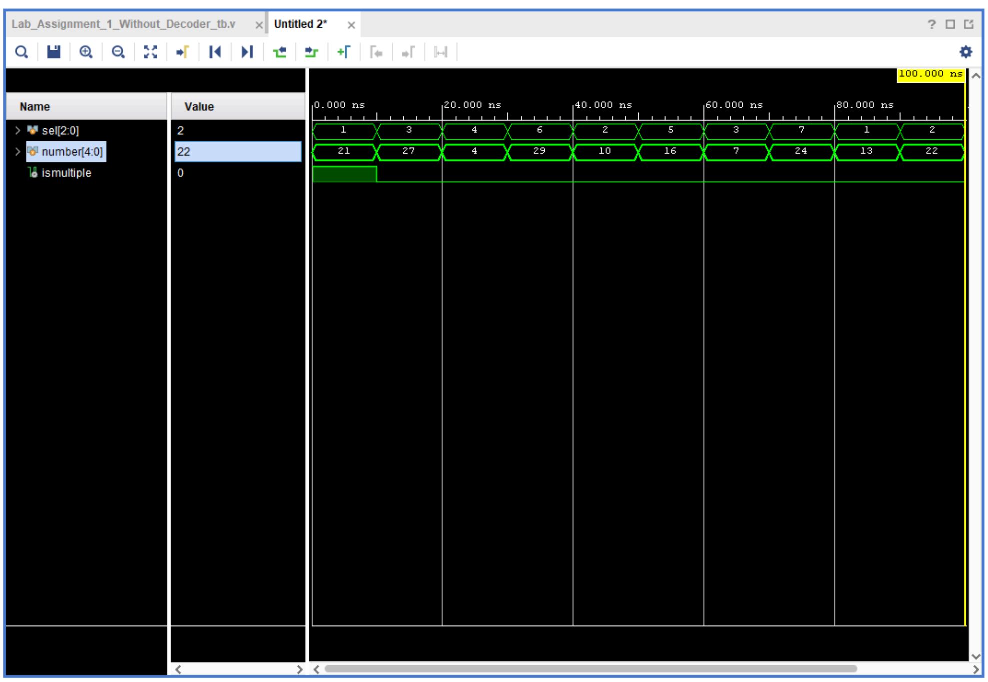
```
// II) Uncomment this for 2nd type
   // Example 1
   sel = 3'b001; // Encoding: 3
   number = 5'b10101; // Decimal equivalent: 21
   #10;
   // Example 2
   sel = 3'b011; // Encoding: 7
   number = 5'b11011; // Decimal equivalent: 27
   #10;
   // Example 3
   sel = 3'b100; // Enciding: 6
   number = 5'b00100; // Decimal equivalent: 4
   #10;
   // Example 4
   sel = 3'b110; // Enciding: 8
   number = 5'b11101; // Decimal equivalent: 29
   #10;
   // Example 5
   sel = 3'b010; // Enciding: 4
   number = 5'b01010; // Decimal equivalent: 10
   #10;
   // Example 6
   sel = 3'b101; // Enciding: 7
   number = 5'b10000; // Decimal equivalent: 16
   #10;
   // Example 7
   sel = 3'b011; // Enciding: 5
   number = 5'b00111; // Decimal equivalent: 7
   #10;
   // Example 8
   sel = 3'b111; // Enciding: 9
   number = 5'b11000; // Decimal equivalent: 24
   #10;
   // Example 9
   sel = 3'b001; // Enciding: 3
   number = 5'b01101; // Decimal equivalent: 13
   #10;
   // Example 10
   sel = 3'b010; // Enciding: 2
   number = 5'b10110; // Decimal equivalent: 22
   #10;
   $finish();
end

Endmodule
```

2.3 Stimulation

Chosen Number = 6, Multiples = Full range



Random Numbers, Multiple checking



NOTE: The values taken by sel are **NOT** the chosen numbers. They are decimal equivalents of the encoding employed to encode the numbers as 3-bit inputs. Hence, the values of **sel** are not to be confused with the chosen numbers.

3. Implementation with Decoder

3.1 Code

```
`timescale 1ns / 1ps
/*
5:32 Decoder with input as [4:0] number
*/
module Lab_Assignment_1_With_Decoder(sel, number, ismultiple);
input [2:0]sel;
input [4:0]number;
reg [31:0]out;
reg a,b,c,d,e,f,g,h;
output reg ismultiple;
always @ (*)
Begin

/* Enable function although not implemented here, but can be easily used by taking an AND of all the 32 outputs
with the 32 enable input. The AND gate will be enable for Enable = 1 and Disabled otherwise */

   out[31] = (number[4] & number[3] & number[2] & number[1] & number[0]);
   out[30] = (number[4] & number[3] & number[2] & number[1] & ~number[0]);
   out[29] = (number[4] & number[3] & number[2] & ~number[1] & number[0]);
   out[28] = (number[4] & number[3] & number[2] & ~number[1] & ~number[0]);
   out[27] = (number[4] & number[3] & ~number[2] & number[1] & number[0]);
   out[26] = (number[4] & number[3] & ~number[2] & number[1] & ~number[0]);
   out[25] = (number[4] & number[3] & ~number[2] & ~number[1] & number[0]);
   out[24] = (number[4] & number[3] & ~number[2] & ~number[1] & ~number[0]);
   out[23] = (number[4] & ~number[3] & number[2] & number[1] & number[0]);
   out[22] = (number[4] & ~number[3] & number[2] & number[1] & ~number[0]);
   out[21] = (number[4] & ~number[3] & number[2] & ~number[1] & number[0]);
   out[20] = (number[4] & ~number[3] & number[2] & ~number[1] & ~number[0]);
   out[19] = (number[4] & ~number[3] & ~number[2] & number[1] & number[0]);
   out[18] = (number[4] & ~number[3] & ~number[2] & number[1] & ~number[0]);
   out[17] = (number[4] & ~number[3] & ~number[2] & ~number[1] & number[0]);
   out[16] = (number[4] & ~number[3] & ~number[2] & ~number[1] & ~number[0]);
   out[15] = (~number[4] & number[3] & number[2] & number[1] & number[0]);
   out[14] = (~number[4] & number[3] & number[2] & number[1] & ~number[0]);
   out[13] = (~number[4] & number[3] & number[2] & ~number[1] & number[0]);
   out[12] = (~number[4] & number[3] & number[2] & ~number[1] & ~number[0]);
   out[11] = (~number[4] & number[3] & ~number[2] & number[1] & number[0]);
   out[10] = (~number[4] & number[3] & ~number[2] & number[1] & ~number[0]);
   out[9] = (~number[4] & number[3] & ~number[2] & ~number[1] & number[0]);
   out[8] = (~number[4] & number[3] & ~number[2] & ~number[1] & ~number[0]);
   out[7] = (~number[4] & ~number[3] & number[2] & number[1] & number[0]);
   out[6] = (~number[4] & ~number[3] & number[2] & number[1] & ~number[0]);
   out[5] = (~number[4] & ~number[3] & number[2] & ~number[1] & number[0]);
   out[4] = (~number[4] & ~number[3] & number[2] & ~number[1] & ~number[0]);
   out[3] = (~number[4] & ~number[3] & ~number[2] & number[1] & number[0]);
   out[2] = (~number[4] & ~number[3] & ~number[2] & number[1] & ~number[0]);
   out[1] = (~number[4] & ~number[3] & ~number[2] & ~number[1] & number[0]);
   out[0] = (~number[4] & ~number[3] & ~number[2] & ~number[1] & ~number[0]);

   ismultiple = 0;
   a =  (sel == 3'b000) & (out[0] | out[2] | out[4] | out[6] | out[8] | out[10] | out[12] | out[14] | out[16] |
out[18] | out[20] | out[22] | out[24] | out[26] | out[28] | out[30]);
   b = (sel == 3'b001) & (out[0] | out[3] | out[6] | out[9] | out[12] | out[15] | out[18] | out[21] | out[24] |
out[27] | out[30]);
   c = (sel == 3'b010) & ( out[0] | out[4] | out[8] | out[12] | out[16] | out[20] | out[24] | out[28]);
   d =  (sel == 3'b011) & ( out[0] | out[5] | out[10] | out[15] | out[20] | out[25] | out[30]);
   e = (sel == 3'b100) & ( out[0] | out[6] | out[12] | out[18] | out[24] | out[30]);
   f = (sel == 3'b101) & ( out[0] | out[7] | out[14] | out[21] | out[28]);
   g = (sel == 3'b110) & ( out[0] | out[8] | out[16] | out[24]);
   h = (sel == 3'b111) & ( out[0] | out[9] | out[18] | out[27]);
   ismultiple = (a | b | c | d | e | f | g | h);
end
endmodule
```

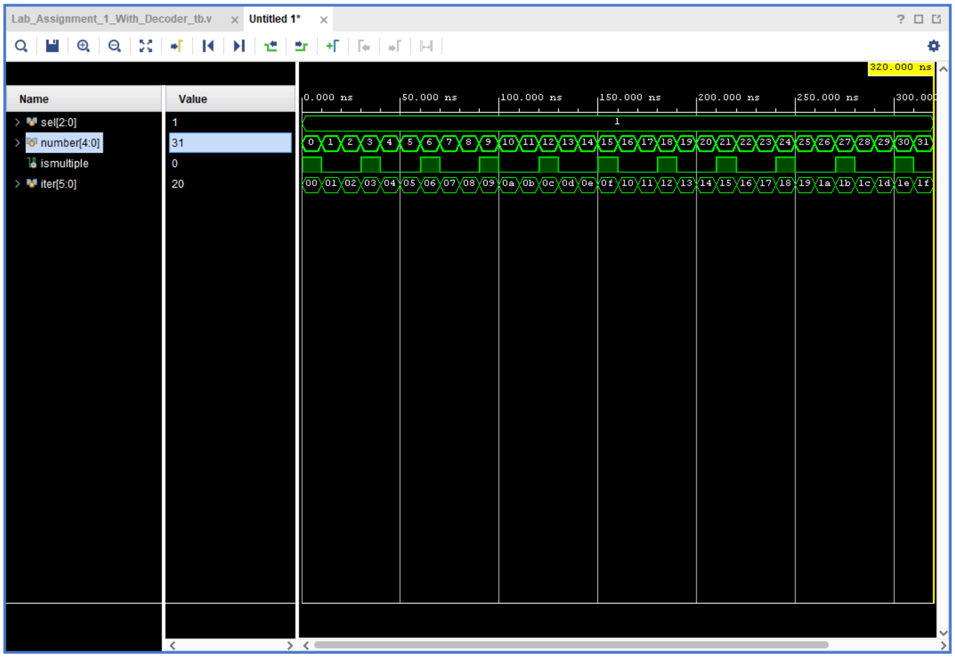
3.2 Test Bench

```
`timescale 1ns / 1ps

/*
sel: (Encoding)

2 <- 000
3 <- 001
4 <- 010
5 <- 011
6 <- 100
7 <- 101
8 <- 110
9 <- 111


number:
Convert decimal representation of any number from 0 to 31 (both inclusive) to 5-bit binary (a5,a4,a3,a2,a1,a0)
a5 is the MSB

ismultiple = 1, if Number (from 0 to 31 both inclusive) is multiple of the number represented by input Sel
ismultiple = 0, elsewhere

*/


module Lab_Assigment_1_With_Decoder_tb();
reg [2:0]sel;
reg [4:0]number;
wire ismultiple;
//reg [5:0]iter; //Uncomment this while using type I
Lab_Assignment_1_With_Decoder uut(sel,number,ismultiple);

initial
Begin


/*
Two types of test benches:
I) Find all the multiples of a number
II) Find whether a random number is a multiple of another number corresponding to the select.
*/



// I) Uncommment this for 1st type
//for(iter=6'b00000;iter<=6'b011111;iter=iter+6'b000001)
//begin
//sel = 3'b111; number = iter[4:0];
//#10;
//end
```

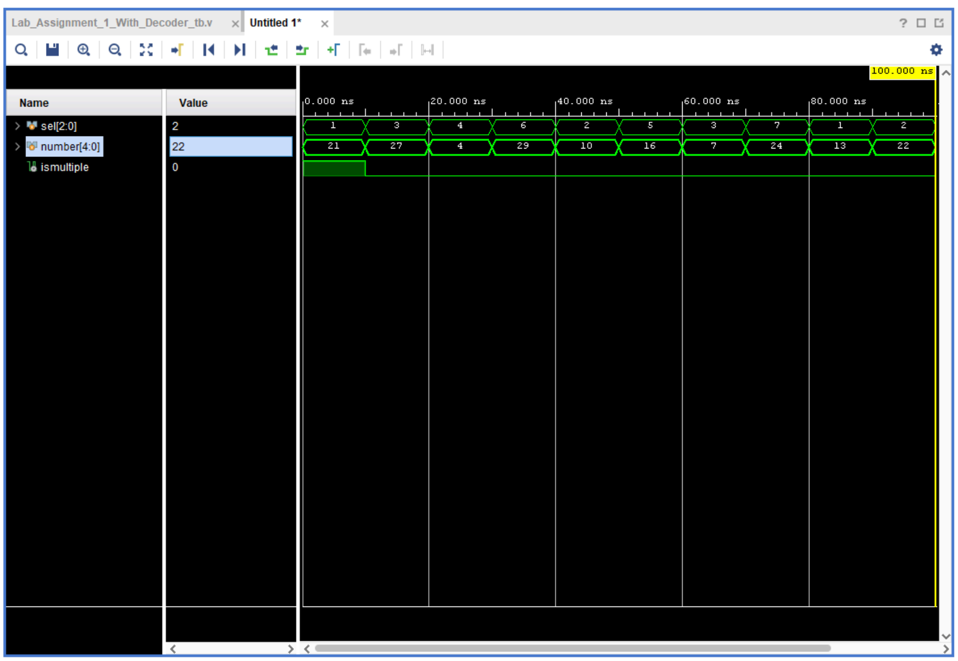
```
// II) Uncomment this for 2nd type
   // Example 1
   sel = 3'b001; // Encoding: 3
   number = 5'b10101; // Decimal equivalent: 21
   #10;
   // Example 2
   sel = 3'b011; // Encoding: 7
   number = 5'b11011; // Decimal equivalent: 27
   #10;
   // Example 3
   sel = 3'b100; // Enciding: 6
   number = 5'b00100; // Decimal equivalent: 4
   #10;
   // Example 4
   sel = 3'b110; // Enciding: 8
   number = 5'b11101; // Decimal equivalent: 29
   #10;
   // Example 5
   sel = 3'b010; // Enciding: 4
   number = 5'b01010; // Decimal equivalent: 10
   #10;
   // Example 6
   sel = 3'b101; // Enciding: 7
   number = 5'b10000; // Decimal equivalent: 16
   #10;
   // Example 7
   sel = 3'b011; // Enciding: 5
   number = 5'b00111; // Decimal equivalent: 7
   #10;
   // Example 8
   sel = 3'b111; // Enciding: 9
   number = 5'b11000; // Decimal equivalent: 24
   #10;
   // Example 9
   sel = 3'b001; // Enciding: 3
   number = 5'b01101; // Decimal equivalent: 13
   #10;
   // Example 10
   sel = 3'b010; // Enciding: 2
   number = 5'b10110; // Decimal equivalent: 22
   #10;
   $finish();
end

endmodule
```

3.3 Stimulation

Chosen Number = 3, Multiples = Full Range



Random Numbers, Multiple checking



NOTE: The values taken by sel are **NOT** the chosen numbers. They are decimal equivalents of the encoding employed to encode the numbers as 3-bit inputs. Hence, the values of **sel** are not to be confused with the chosen numbers.

4. Conclusion and Learnings

Case 1:

- This uses **always** block, and we check for each of the multiple needed.
- Implemented using a simple if-else if-else ladder.
- Used k-map implementation for each of the **select** options.
- For a different **select** (let's say 10), we have to make a different k-map for finding the divisibility function.
- Becomes more and more complex when there are more than 5-bit inputs.
- It's more pythonic in nature.

Case 2:

- This uses a 5 x 32 Decoder to implement the logic
- It exploits the fact that a function can be generated using the sum of minterms (which are the outputs of the decoder)
- We can make multiple functions (Multiples of 2, 3, … 9, in this case) and implement them using OR and AND gate logics.
- Even in the case of more than 5-bit inputs, we can use n x 2^n decoder and **OR** the required minterms for simpler implementation.
- More digital and logic-based in nature.