Digital Systems : Lab 1

Nimitt (2211169)

Pratham Sharda (2211203)

Problem Statement

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.

The following sections describe our solutions. To begin lets first describe the port names used for the input and output.

Port Names

The following port names were used to represent corresponding values :

1. **Select lines - 4 bit Array - [3 : 0] S**
2. **Range input - 5 bit Array - [4 : 0] R**
3. **Output - Scalar - OUT**

Paradigm

We know that the output is logical function of the Range input (R) and Select Input (S). First lets, lets define Rd ([9:1]) such that Rd[i] = 1, if R is multiple of i. Similarly, we define Sd(9:1) such that Sd[i] = 1, if i == S. Now, OUT goes high whenever R is the multiple of S. We can clearly see OUT is 1 whenever (Rd[i] AND Sd[i] == 1) because Rd[i] represents R is some multiple of i, but Sd = 1 says S == i, implying R is multiple of S, hence OUT = 1. Therefore,

OUT = OR(Rd[i] AND Sd[i]) where i goes from 1 to 9.

Now, our task reduces to implementing Rd and Sd.

Implementation

Approach A. : Using Karnaugh Maps

First we intend to implement this using Karnaugh Maps. For Rd and Sd, we compute each element by minimizing the corresponding Truth Tables.

$$R_d[i] = f(Multuples(i))$$

$$S_d[i] = f(S == i)$$

$$OUT = OR(R_d[i] AND S_d[i])$$

The following is the Corresponding Verliog Code :

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 01/13/2024 11:38:49 AM
// Design Name:
// Module Name: Question1
// Project Name:
```

```
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////


module Question1(
    input [3:0] S,
    input [4:0] R,
    output reg OUT
    );


    reg [31:0] Rd;
    reg [9:1] Sd;
    always @(*) begin


// Rd Array
        Rd[2] = (~R[0]); // Multiples of 2
        Rd[3] = ( ~R[4]& ~R[3]& ~R[2]& ~R[1]& ~R[0]  | ~R[4]& ~R
        Rd[4] = (~R[1]& ~R[0] );
        Rd[5] = (~R[4]& ~R[3]& ~R[2]& ~R[1]& ~R[0] | ~R[4]& ~R[3
        Rd[6] = (~R[4]& ~R[3]& ~R[2]& ~R[1]& ~R[0]  | ~R[4]& ~R
        Rd[7] = (~R[4]& ~R[3]& ~R[2]& ~R[1]& ~R[0]  | ~R[4]& ~R
        Rd[8] = (~R[2]& ~R[1]& ~R[0] );
        Rd[9] = ( ~R[4]& ~R[3]& ~R[2]& ~R[1]& ~R[0]  | ~R[4]& R
```

```
// Sd Array
        Sd[1] = (S[0] & ~S[1] & ~S[2] & ~S[3]);
        Sd[2] = (~S[0] & S[1] & ~S[2] & ~S[3]);
        Sd[3] = (S[0] & S[1] & ~S[2] & ~S[3]);
        Sd[4] = (~S[0] & ~S[1] & S[2] & ~S[3]);
        Sd[5] = (S[0] & ~S[1] & S[2] & ~S[3]);
        Sd[6] = (~S[0] & S[1] & S[2] & ~S[3]);
        Sd[7] = (S[0] & S[1] & S[2] & ~S[3]);
        Sd[8] = (~S[0] & ~S[1] & ~S[2] & S[3]);
        Sd[9] = (S[0] & ~S[1] & ~S[2] & S[3]);

// Final Output
        OUT = ( Sd[1] | (Sd[2] & Rd[2]) | (Sd[3] & Rd[3]) | (Sd
    end
endmodule
```

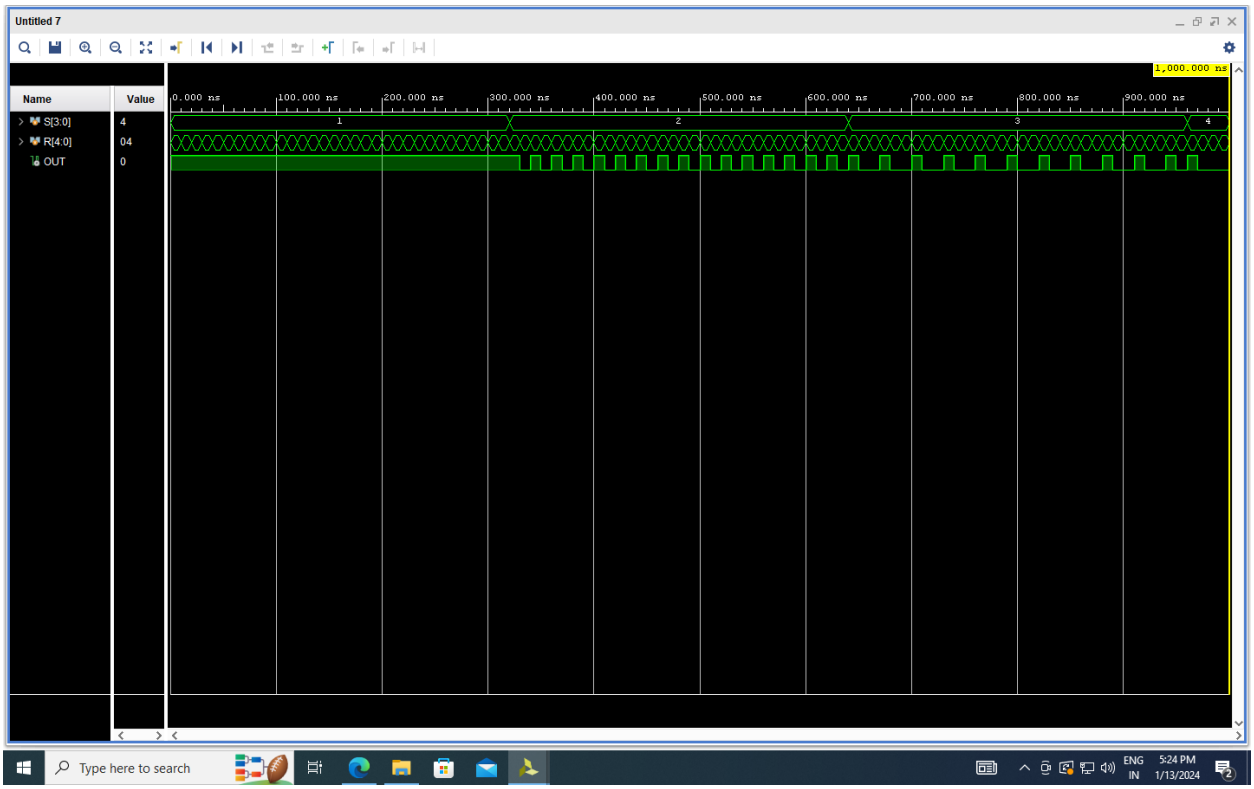
So, this implements the logic correctly. Now, we created the TestBench File for different inputs S and R and finally ran the testbenches to compare the results. The following section shows the testbench file and simulation results.

```
TestBench
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 01/13/2024 04:25:51 PM
// Design Name:
// Module Name: Question2_tb
// Project Name:
// Target Devices:
// Tool Versions:
```

```
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////


module Question2_tb(

    );

    reg [3:0] S;
    reg [4:0] R;
    wire OUT;

    Question1 uut(S,R,OUT);

    initial
    begin
    // S == 1
        S = 4'b0001 ; R = 5'b00000;
        #10;
        S = 4'b0001 ; R = 5'b00001;
        #10;
        S = 4'b0001 ; R = 5'b00010;
        #10;
        S = 4'b0001 ; R = 5'b00011;
        #10;
        S = 4'b0001 ; R = 5'b00100;
        #10;
        S = 4'b0001 ; R = 5'b00101;
        #10;
```

```
        S = 4'b0001 ; R = 5'b00110;
        #10;
        S = 4'b0001 ; R = 5'b00111;
        #10;

.....  // Ommiting the inbetween code HERE (original file has it
       // length as the same has been repeated for every S


           // S == 9
        S = 4'b1001 ; R = 5'b00000;
        #10;
        S = 4'b1001 ; R = 5'b00001;
        #10;
        S = 4'b1001 ; R = 5'b00010;
        #10;
        S = 4'b1001 ; R = 5'b00011;
        #10;
```

We get the following simulations results :


Untitled 7
Name
Value
S[3:0]
4
R[4:0]
04
OUT
0
1,000.000 ns
Type here to search
ENG
IN
5:24 PM
1/13/2024

S == 1; S == 2; S == 3


Untitled 2
Name
Value
S[3:0]
3
R[4:0]
09
OUT
1
95.057 ns

S == 3

S == 6

We get OUT == 1 whenever R is multiple of S, the desired output.

Approach B. : Using 5*32 Decoder

Now, We implement the same logic using Decoder. In the implementation of Rd, we had computed the function using Karnaugh maps. But, we can very easily implement the same logic using a Decoder. As for Rd, every element Rd[i] is the OR operation between Signals from the decoder output at multiples of i. Futher, we compute the OUT by operating on Rd and Sd.

The following is the Verilog code that implements this :

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////
// Company:
// Engineer:
```

```
//
// Create Date: 01/13/2024 11:38:49 AM
// Design Name:
// Module Name: Question2
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////


module Question2(
    input [3:0] S,
    input [4:0] R,
    output reg OUT
    );

    reg [31:0] Rd;
    reg [9:2] Rm;
    reg [9:1] Sd;
    always @(*)
        begin
        // Decoder
        Rd[0] = (~R[0] & ~R[1] & ~R[2] & ~R[3] & ~R[4]);
        Rd[1] = (R[0] & ~R[1] & ~R[2] & ~R[3] & ~R[4]);
        Rd[2] = (~R[0] & R[1] & ~R[2] & ~R[3] & ~R[4]);
        Rd[3] = (R[0] & R[1] & ~R[2] & ~R[3] & ~R[4]);
        Rd[4] = (~R[0] & ~R[1] & R[2] & ~R[3] & ~R[4]);
        Rd[5] = (R[0] & ~R[1] & R[2] & ~R[3] & ~R[4]);
```

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

// Creating Rm
        Rm[2] = (Rd[0] | Rd[2] | Rd[4] | Rd[6] | Rd[8] | Rd[10]
        Rm[3] = (Rd[0] | Rd[3] | Rd[6] | Rd[9] | Rd[12] | Rd[15]
        Rm[4] = (Rd[0] | Rd[4] | Rd[8] | Rd[12] | Rd[16] | Rd[2(
        Rm[5] = (Rd[0] | Rd[5] | Rd[10] | Rd[15] | Rd[20] | Rd[
        Rm[6] = (Rd[0] | Rd[6] | Rd[12] | Rd[18] | Rd[24] | Rd[
        Rm[7] = (Rd[0] | Rd[7] | Rd[14] | Rd[21] | Rd[28]);
        Rm[8] = (Rd[0] | Rd[8] | Rd[16] | Rd[24]);
        Rm[9] = (Rd[0] | Rd[9] | Rd[18] | Rd[27]);
```

```
        Sd[1] = (S[0] & ~S[1] & ~S[2] & ~S[3]);
        Sd[2] = (~S[0] & S[1] & ~S[2] & ~S[3]);
        Sd[3] = (S[0] & S[1] & ~S[2] & ~S[3]);
        Sd[4] = (~S[0] & ~S[1] & S[2] & ~S[3]);
        Sd[5] = (S[0] & ~S[1] & S[2] & ~S[3]);
        Sd[6] = (~S[0] & S[1] & S[2] & ~S[3]);
        Sd[7] = (S[0] & S[1] & S[2] & ~S[3]);
        Sd[8] = (~S[0] & ~S[1] & ~S[2] & S[3]);
        Sd[9] = (S[0] & ~S[1] & ~S[2] & S[3]);

        // computing the final out
        OUT = ( Sd[1] | (Sd[2] & Rm[2]) | (Sd[3] & Rm[3]) | (Sd
        end
endmodule
```

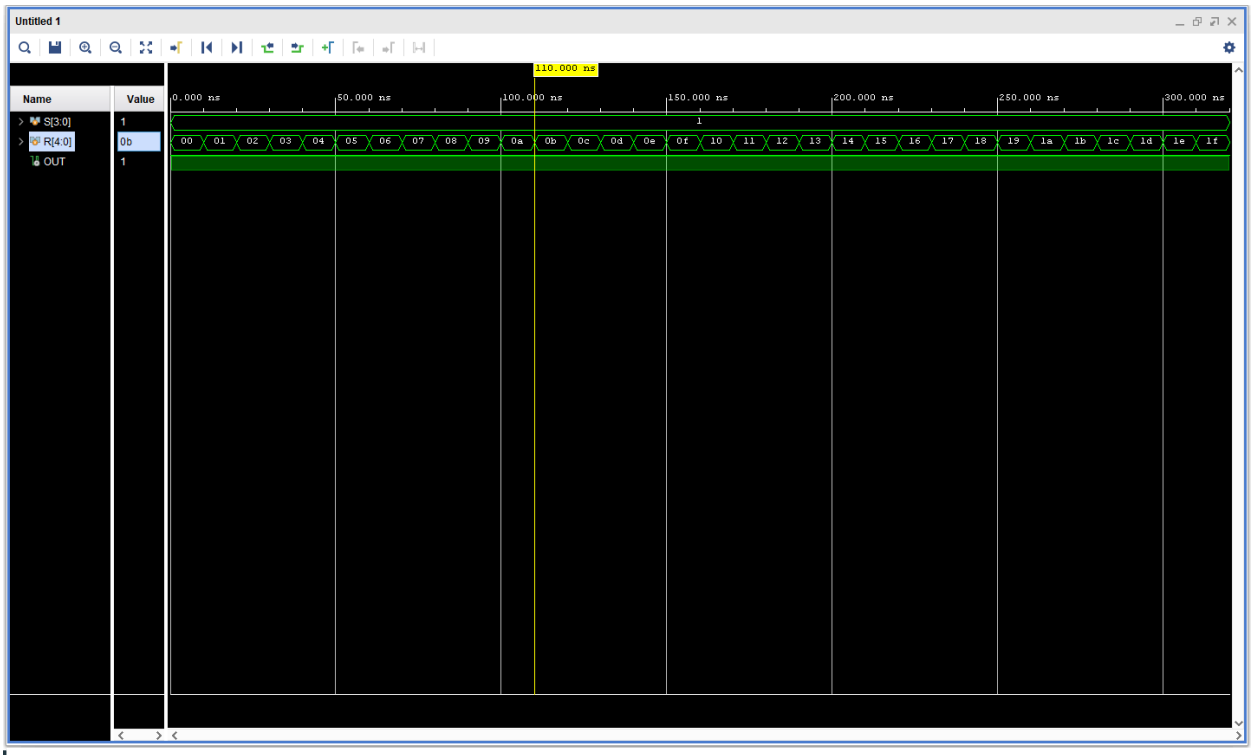
Now, we test it :

```
// TestBench
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 01/13/2024 04:25:51 PM
// Design Name:
// Module Name: Question2_tb
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
```

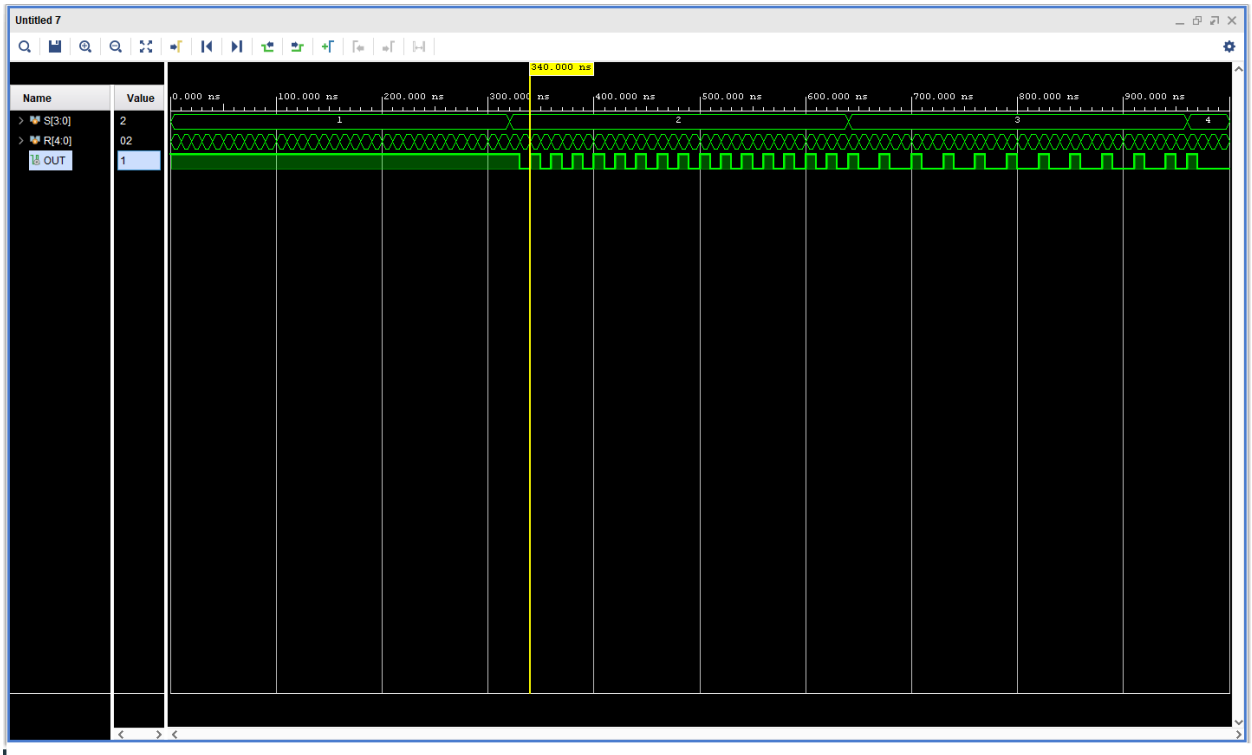
```
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////


module Question2_tb(

    );

    reg [3:0] S;
    reg [4:0] R;
    wire OUT;

    Question2 uut(S,R,OUT);

    initial
    begin
    // S == 1
        S = 4'b0001 ; R = 5'b00000;
        #10;
        S = 4'b0001 ; R = 5'b00001;
        #10;
        S = 4'b0001 ; R = 5'b00010;
        #10;
        S = 4'b0001 ; R = 5'b00011;
        #10;
        S = 4'b0001 ; R = 5'b00100;
        #10;
        S = 4'b0001 ; R = 5'b00101;
        #10;
        S = 4'b0001 ; R = 5'b00110;
        #10;
        S = 4'b0001 ; R = 5'b00111;
```

```
        #10;

.....  // Ommiting the inbetween code HERE (original file has it
       // length as the same has been repeated for every S


   // S == 9
        S = 4'b1001 ; R = 5'b00000;
        #10;
        S = 4'b1001 ; R = 5'b00001;
        #10;
        S = 4'b1001 ; R = 5'b00010;
        #10;
        S = 4'b1001 ; R = 5'b00011;
        #10;
```

We get the same desired simulation results :


Untitled 1
110.000 ns
Name
Value
S[3:0]
R[4:0]
OUT
1
0b
1
0.000 ns
50.000 ns
100.000 ns
150.000 ns
200.000 ns
250.000 ns
300.000 ns
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f

S == 1


Untitled 7
340.000 ns
Name
Value
S[3:0]
R[4:0]
OUT
2
02
1
0.000 ns
100.000 ns
200.000 ns
300.000 ns
400.000 ns
500.000 ns
600.000 ns
700.000 ns
800.000 ns
900.000 ns
1
2
3
4

S == 1; S == 2; S == 3

Approach A VS Approach B

In approach A, we had to compute the function using Karnaugh maps that amounts more tedious

work. But in approach B, we could implement the logic using Decoder easily as we could now use the Minterms directly. This way the decoder helped to simplify the hardware design and made us lesser prone to errors. Clearly, Decoder approach comes out to be farely simpler.