
**lab report 1**

Andrew Nady
900184042

# Problem (2)
# Technical summary:

Steps:

We had to implement 3 experiments, which were to refresh our knowledge about Verilog.

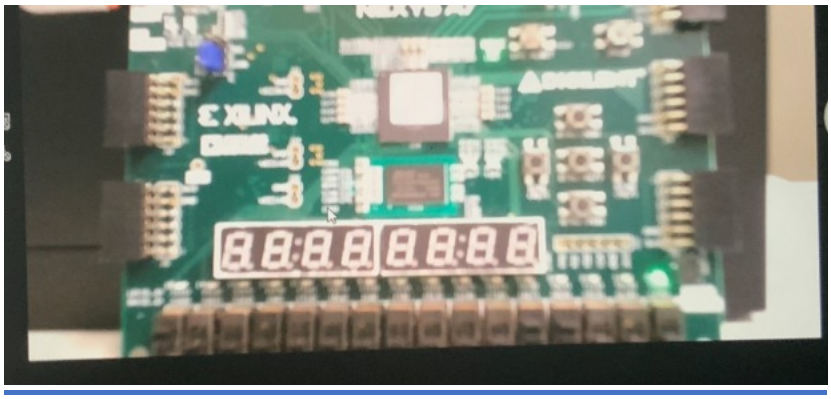
- In the first experiment, we implemented a simple program that invert the input from 1 to 0 or vice versa. The contribution file was supported for us.
- In second experiment, we needed to make a 4-digit 7-segment display driver**.** we took the code from the lab manual and changed small things in it

    1. We commented the input [12:0] num because we won't be able to test it and added a wire of 13 bits instead inside the module and gave the number a value 1234
    2. Secondly, we added output [3:0] sAnode which is not reg and assigned a value for all the bits with zeros and then used this sAnode in the constrain File to disactivate the left 4 digits

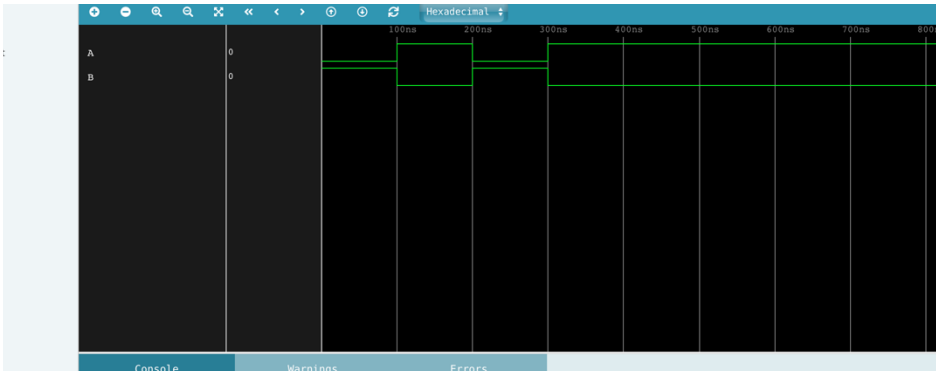
- In experiment 3, we implemented the same functionality as the second one but using more efficient algorithm. We did the same changes as problem 2 except assigning 4962 to the number.

Results:

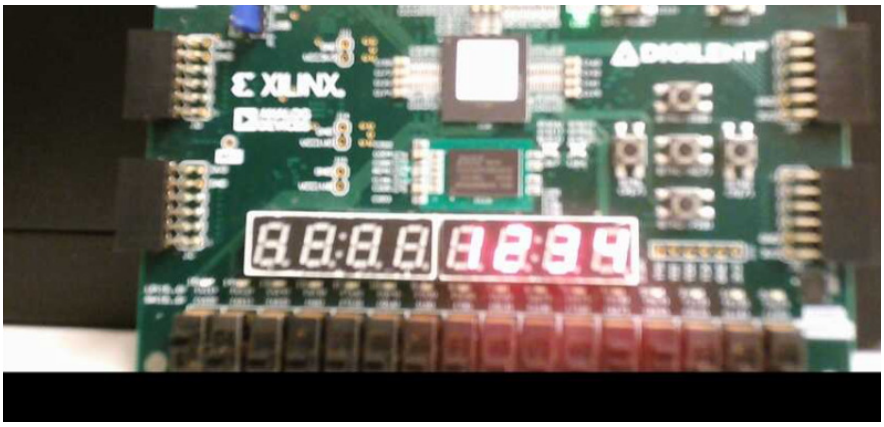
- In the first experiment, we assigned the input as a wire giving it 0, so that we can test our program on the fpga. The green led in the right turned on.



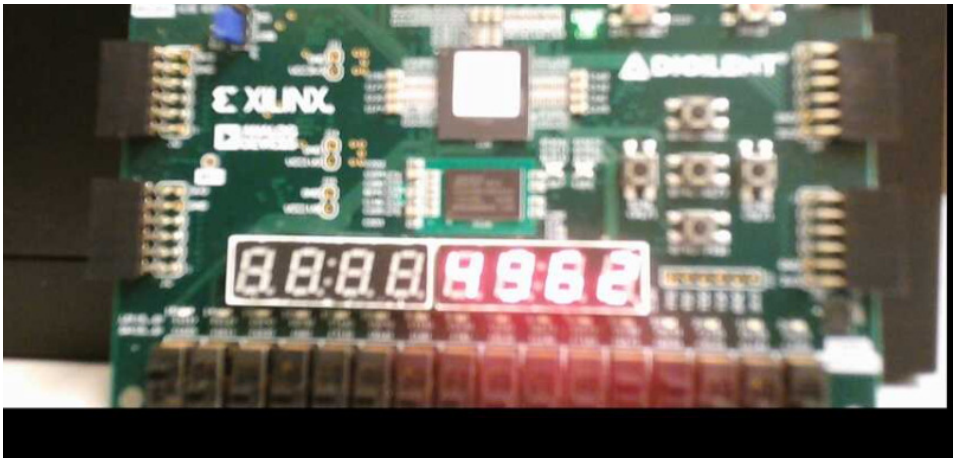
This is the simulation of the testbench of exp 2.




- In experiment 2, we assigned the number to 1234, and disactivated the left 4 digits by giving the sAnodes 1's, because they are active low




- In experiment 3, we assigned the number to 4963, and disactivated the left 4 digits by giving the sAnodes 1's, because they are active low

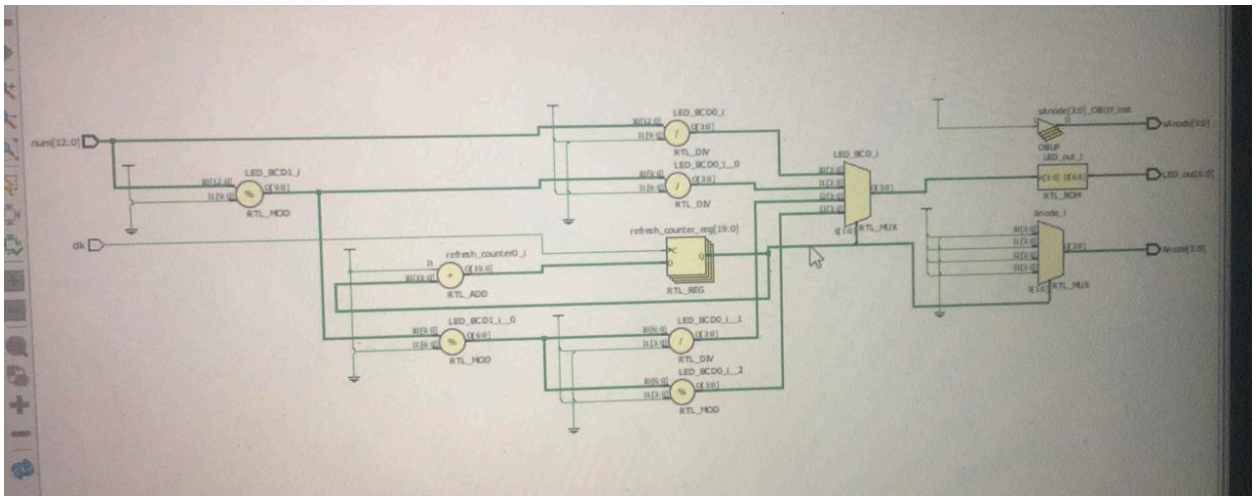


## Components:

- Equipment

1. Vivado
2. VNC
3. FPGA (Nexys A7 100T)

- Screenshot Schematic of experiment 2,



## Code Functional:

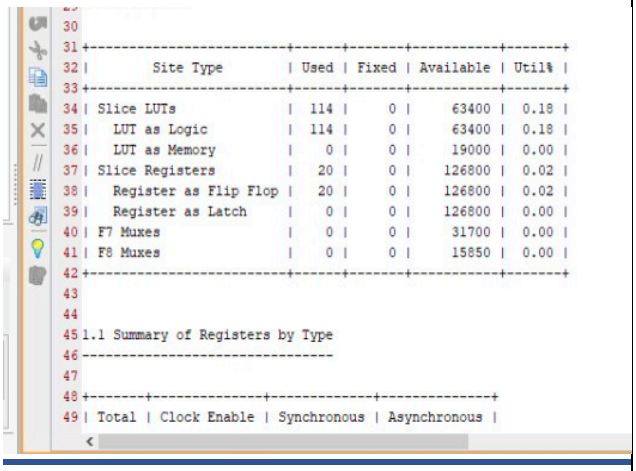
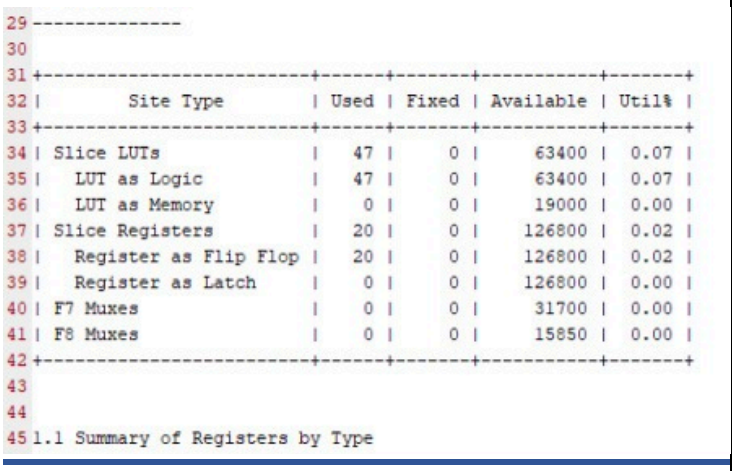
- First experiment: inverts the signal of a bit

  Input -> output
  1 -> 0
  0 -> 1

- Second and third experiment, displays seven segments led on only four digits of the 8 digits.

# Problem (3)
# Comparison:

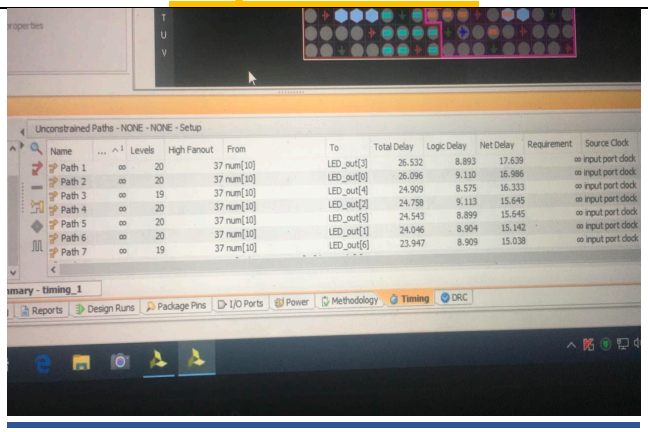
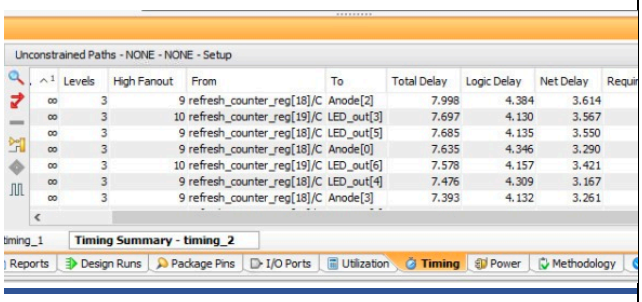
Utilization:

| | Experiment 2 | Experiment 3 |
|---|---|---|
| SCREENSHOTS |  |  |

```
30
31 +-------------------------+------+-------+-----------+-------+
32 |        Site Type        | Used | Fixed | Available | Util% |
33 +-------------------------+------+-------+-----------+-------+
34 | Slice LUTs              |  114 |     0 |     63400 |  0.18 |
35 |   LUT as Logic          |  114 |     0 |     63400 |  0.18 |
36 |   LUT as Memory         |    0 |     0 |     19000 |  0.00 |
37 | Slice Registers         |   20 |     0 |    126800 |  0.02 |
38 |   Register as Flip Flop |   20 |     0 |    126800 |  0.02 |
39 |   Register as Latch     |    0 |     0 |    126800 |  0.00 |
40 | F7 Muxes                |    0 |     0 |     31700 |  0.00 |
41 | F8 Muxes                |    0 |     0 |     15850 |  0.00 |
42 +-------------------------+------+-------+-----------+-------+
43
44
45 1.1 Summary of Registers by Type
46 ---------------------------------
47
48 +-------+--------------+-------------+--------------+
49 | Total | Clock Enable | Synchronous | Asynchronous |
```

```
29 --------------
30
31 +-------------------------+------+-------+-----------+-------+
32 |        Site Type        | Used | Fixed | Available | Util% |
33 +-------------------------+------+-------+-----------+-------+
34 | Slice LUTs              |   47 |     0 |     63400 |  0.07 |
35 |   LUT as Logic          |   47 |     0 |     63400 |  0.07 |
36 |   LUT as Memory         |    0 |     0 |     19000 |  0.00 |
37 | Slice Registers         |   20 |     0 |    126800 |  0.02 |
38 |   Register as Flip Flop |   20 |     0 |    126800 |  0.02 |
39 |   Register as Latch     |    0 |     0 |    126800 |  0.00 |
40 | F7 Muxes                |    0 |     0 |     31700 |  0.00 |
41 | F8 Muxes                |    0 |     0 |     15850 |  0.00 |
42 +-------------------------+------+-------+-----------+-------+
43
44
45 1.1 Summary of Registers by Type
```

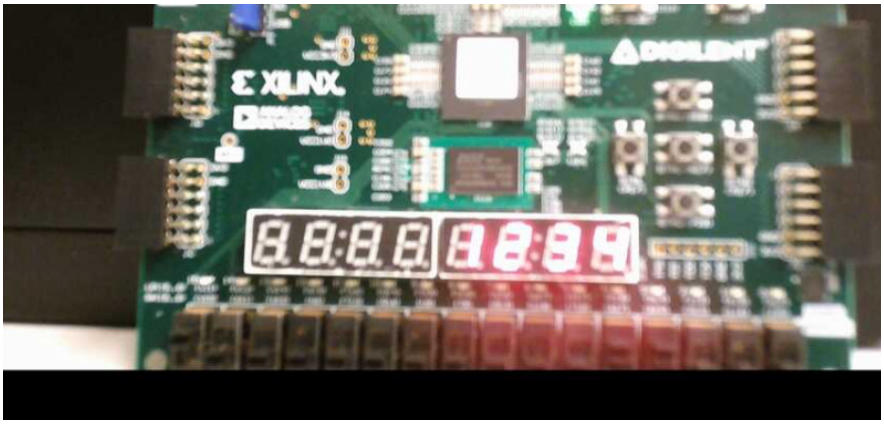
Delay:

| | Experiment 2 | Experiment 3 |
|---|---|---|
| SCREENSHOTS |  |  |

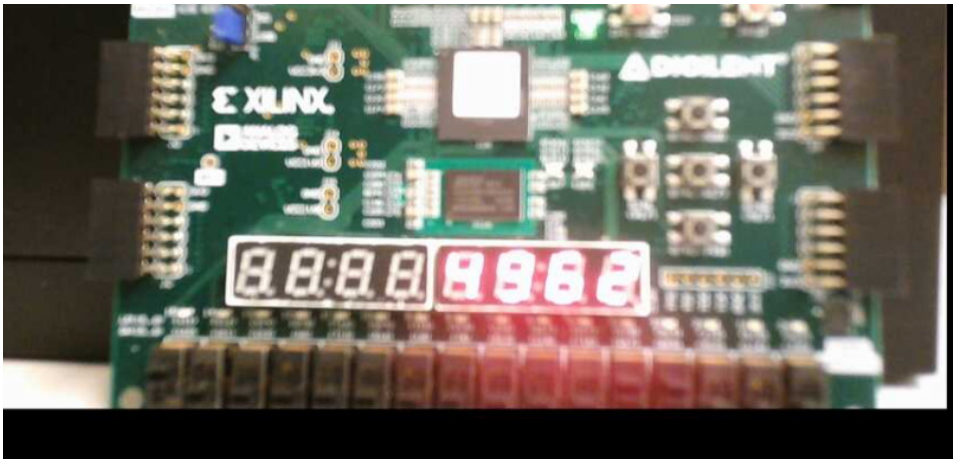
Description:
when looking to the results, we will find that the utilization and the delay have smaller values in experiment 3, which means that the hardware resources and the time taken in experiment 3 are fewer than that of experiment 2.  That indicates that the algorithm we used in experiment 3 is more efficient

## Problem (4)
## Results of 2&3:

Experiment 2:



Experiment 3:



## Problem (5):

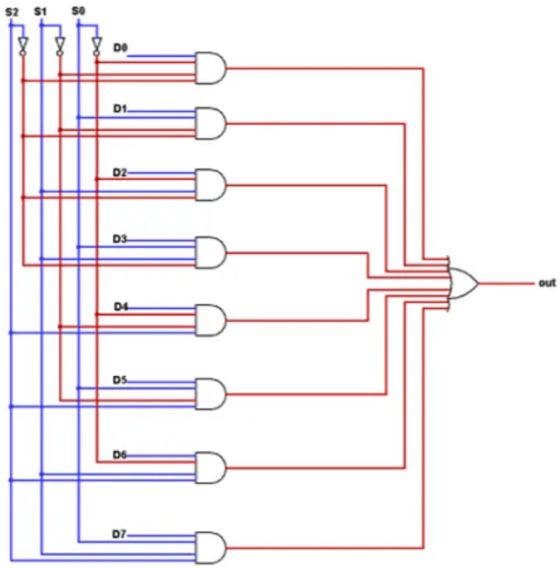
- First of all, we have to about this boundary
  99999999 has **27** bits because or $2^{26} \leq 99999999 \leq 2^{27} - 1$

  the FPGA contains 9 digits so the max number can be 9999 9999; however if we make the max number of bits = 27, that will lead to overflow as it can represent till $2^{27} - 1$ which is more than 8 digits. In this case, to avoid overflow, the max number of bits =26 bits. So the max number the FPGA can represent in decimal = 67108864.
- This is also happened in the code which is in the lab manual, the max number is 13 bits, however 9999 can be represented in 14 bits. This because 14 bits can cause overflow because it can represent more than 4 digits
- When we will implement 8 digits instead of 4 digits, then we should modify in our algorithm to compute the 10 thousand and so on.
  - We will change the for loop inside the BCD to illiterate over the 26 bit

The answer: 26 bits

## Problem (6):

This is the design that we should implement, where there are 8 inputs, 3 selectors, and 1 output.




The result should be as the following :

| S0 | S1 | S0 | OUTPUT |
|---|---|---|---|
| 0 | 0 | 0 | D0 |
| 0 | 0 | 1 | D1 |
| 0 | 1 | 0 | D2 |
| 0 | 1 | 1 | D3 |
| 1 | 0 | 0 | D4 |
| 1 | 0 | 1 | D5 |
| 1 | 1 | 0 | D6 |
| 1 | 1 | 1 | D7 |

# Gate level:

Code:

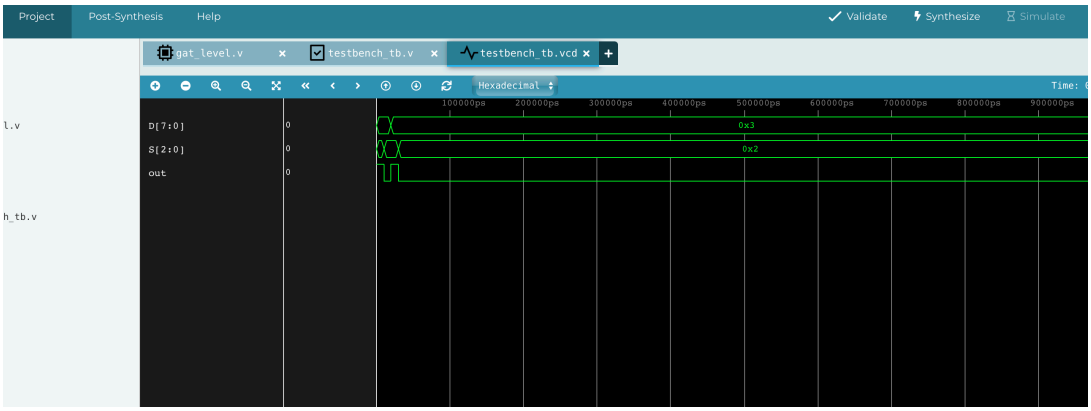
```
// file: gat_level.v
// author: @andrewhany

`timescale 1ns/1ps
module gat_level(
input  [7:0] D,
input  [2:0] S,
output out);
```

```
wire [10:0]w;
not(w[0], S[0]);
not(w[1], S[1]);
not(w[2], S[2]);
and(w[3], D[0], w[0], w[1], w[2]);
and (w[4], D[1], S[0] ,w[1], w[2]);
and(w[5], D[2], w[0], S[1], w[2]);
and (w[6], D[3], S[0], S[1], w[2]);
and(w[7], D[4], w[0], w[1], S[2]);
and (w[8], D[5], S[0], w[1], S[2]);
and(w[9], D[6], w[0], S[1], S[2]);
and (w[10], D[7], S[0], S[1], S[2]);
or(out, w[3], w[4], w[5], w[6], w[7], w[8], w[9], w[10]);
endmodule
```

Testbench to test if my design right or not:

```
// file: testbench_tb.v
// author: @andrewhany
// Testbench testbench_tb

`timescale 1ns/1ns

module testbench_tb;

reg [7:0] D=8'b00000000;
reg  [2:0] S=3'b000;
output out;
 gat_level l(
 D[7:0] ,
   S[2:0],
 out);

initial
  begin

  D[7:0]=8'b00000001;
  #10
  S[2:0]=3'b001;
  #10
  D[7:0]=8'b00000011;
  #10
   S[2:0]=3'b010;



  end
endmodule
```

Simulation:
According to the above truth table above, my implementation is correct.




# Dataflow:

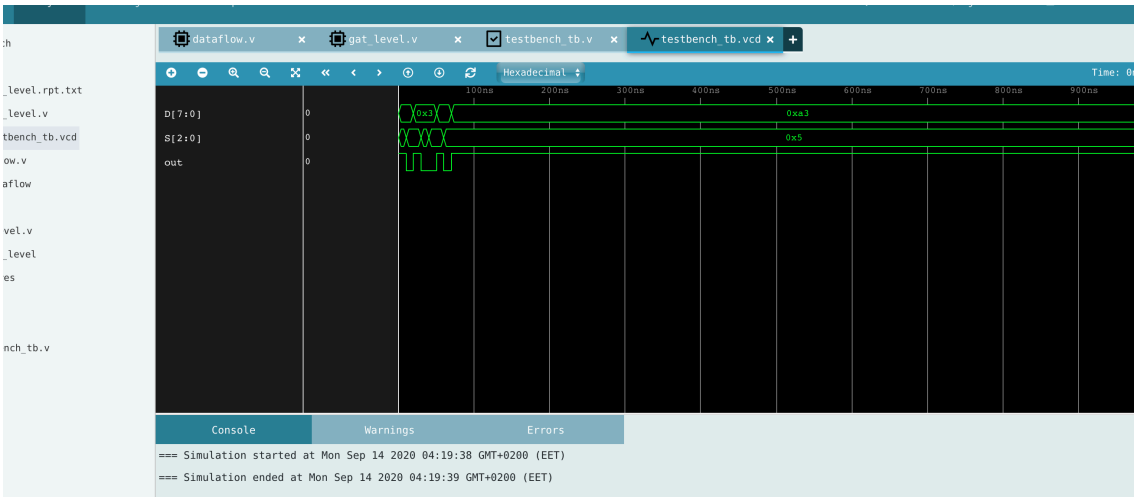
# Code:

```
// file: dataflow.v
// author: @andrewhany

`timescale 1ns/1ns

module dataflow(
input [7:0] D,
input  [2:0] s,
output out);

assign out=(D[0]&~s[0]&~s[1]&~s[2])|
       (D[1]&s[0]&~s[1]&~s[2])|
         (D[2]&~s[0]&s[1]&~s[2])|
         (D[3]&s[0]&s[1]&~s[2])|
         (D[4]&~s[0]&~s[1]&s[2])|
           (D[5]&s[0]&~s[1]&s[2])|
           (D[6]&~s[0]&s[1]&s[2])|
           (D[7]&s[0]&s[1]&s[2]);
endmodule
```

In the Testbench, I tested more condition so I can make sure that I am right

```
// file: testbench_tb.v
// author: @andrewhany
// Testbench testbench_tb


// file: testbench_tb.v
// author: @andrewhany
// Testbench testbench_tb

`timescale 1ns/1ns

module testbench_tb;

reg [7:0] D=8'b00000000;
reg  [2:0] S=3'b000;
output out;
 dataflow l(
 D[7:0] ,
   S[2:0],
 out);

initial
  begin

  D[7:0]=8'b00000001;
  #10
  S[2:0]=3'b001;
  #10
  D[2:0]=8'b00000011;
  #10
   S[2:0]=3'b010;
   #10
  S[2:0]=3'b111;
  #10
  D[7:0]=8'b10000011;
  #10
  S[2:0]=3'b101;
  #10
  D[7:0]=8'b10100011;


  end
endmodule
```

# Simulation:




# behavioral modeling:
# code:

```
// file: behavioral.v
// author: @andrewhany

`timescale 1ns/1ns

module behavioral(input  [7:0] D,input [2:0] S, output reg out);

always @(*)
begin
  case(S)
          3'b000: out=D[0];
          3'b001: out=D[1];
          3'b010: out=D[2];
          3'b011: out=D[3];
          3'b100: out=D[4];
          3'b101: out=D[5];
          3'b110: out=D[6];
          3'b111: out=D[7];

          default: out=1'b0;
  endcase
```

```
end
endmodule
```

Testbench is the same as the last one, so the simulation is the same

note: I could not simulate at first because when triying to simulate, it gives me errors. Then I discovered that the problem is that I forget the reg