Kevin Karnadi Kirmansjah 紀維鑫

109006241

Logic Design

Lab 1: Kevin Number Detector

I. Design Process

In this assignment, I am asked to construct a 4-bit Kevin number detector and describe it in different ways that we have learned before, which are the gate level, dataflow, and behavior descriptions. The ‘Kevin’ numbers are described as 1, 5, 6, 7, 9, 10, 12, 14 by the TA.

To be able to construct the Kevin number detector, I need to know about what the algebraic expression looks like first. To find out about that, I need to create a truth table, and then create the corresponding K-map out of it.

Truth Table:

A	B	C	D	Value	Output
0	0	0	0	0	0
0	0	0	1	1	0
0	0	1	0	2	0
0	0	1	1	3	0
0	1	0	0	4	0
0	1	0	1	5	1
0	1	1	0	6	1
0	1	1	1	7	1
1	0	0	0	8	0
1	0	0	1	9	1
1	0	1	0	10	1
1	0	1	1	11	0
1	1	0	0	12	1
1	1	0	1	13	0
1	1	1	0	14	1
1	1	1	1	15	0

After creating this truth table, I can make the corresponding K-map for it.

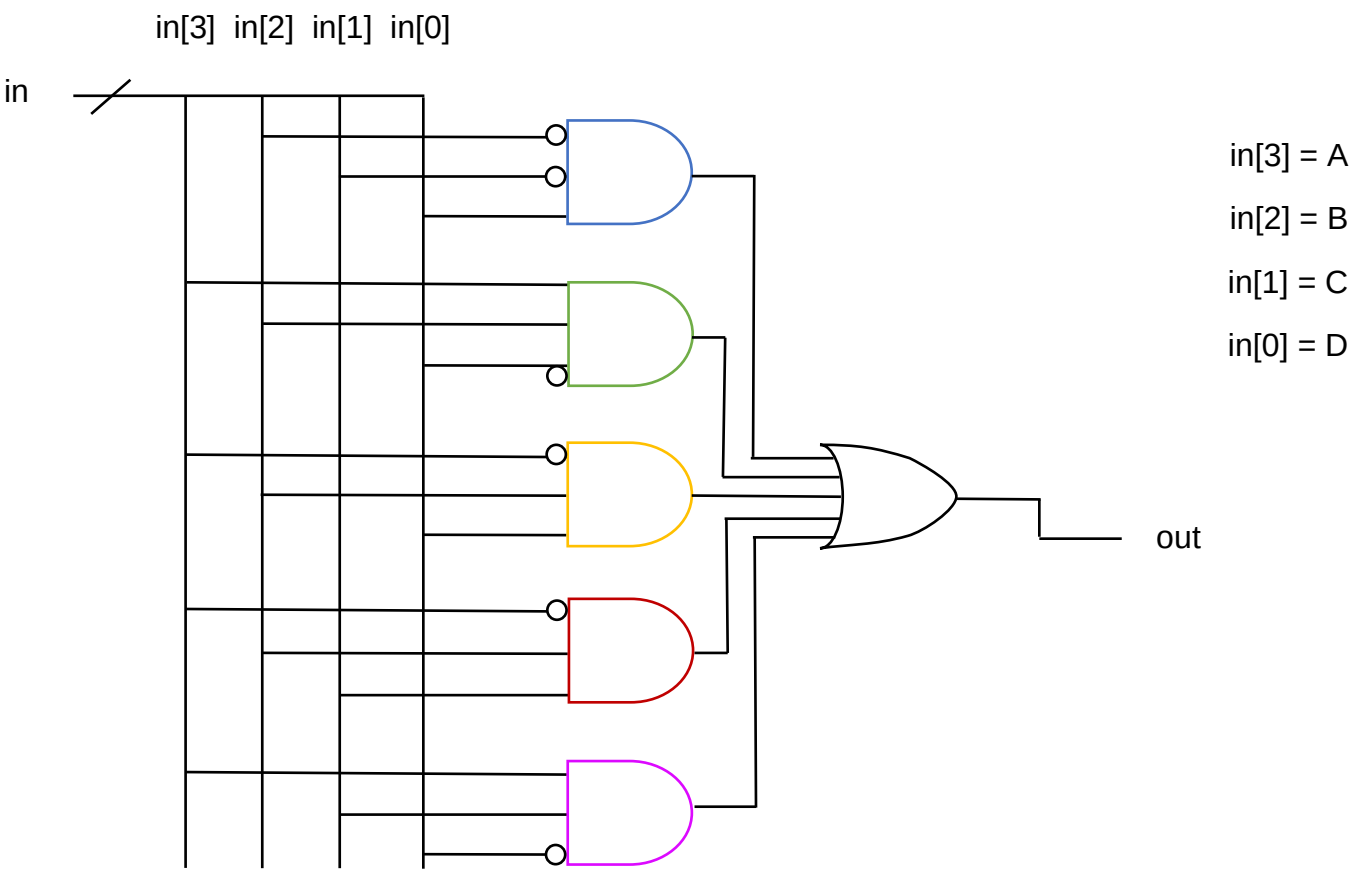
K-map:

AB \ CD	00	01	11	10
00	0	1	0	0
01	0	1	1	1
11	1	0	0	1
10	0	1	0	1

From the K-map above, we can obtain this algebraic expression:

B’C’D + ABD’ + A’BD + A’BC + ACD’

and from that expression, we can create this circuit diagram:



After all of the steps above, I wrote the Verilog program for the Kevin number detector using a text editor called Notepad++.

1. Gate Level Description

```
module kevin_G(in, out);

    input [3:0]in;
    output out;

    wire notA, notB, notC, notD;
    wire and1, and2, and3, and4, and5;

    not not_1(notA, in[3]);
    not not_2(notB, in[2]);
    not not_3(notC, in[1]);
    not not_4(notD, in[0]);

    and and_1(and1, notB, notC, in[0]);
    and and_2(and2, in[3], in[2], notD);
    and and_3(and3, notA, in[2], in[0]);
    and and_4(and4, notA, in[2], in[1]);
    and and_5(and5, in[3], in[1], notD);

    or or_1(out, and1, and2, and3, and4, and5);

endmodule
```

2. Dataflow Description

```
module kevin_D(in, out);

    input [3:0]in;
    output out;

    assign out = (!in[2]&!in[1]&in[0]) |
                 (in[3]&in[2]&!in[0])  |
                 (!in[3]&in[2]&in[0])  |
                 (!in[3]&in[2]&in[1])  |
                 (in[3]&in[1]&!in[0]);

endmodule
```

3. Behavior Description

```
module kevin_B(in, out);

    input [3:0]in;
    output out;

    reg out;

    always@(*)begin
        out=1'b0;
        case(in)
            1,5,6,7,9,10,12,14:begin
                out=1'b1;
            end
        endcase
    end

endmodule
```

After creating those 3 modules, I created a testbench which is used to test my Kevin number detector and see if it works or not.

```
module kevin_tb;

    parameter delay=5;

    wire out_G,out_D,out_B;
    reg [3:0]in;
    integer i;

    kevin_G hvg
    (
        .in(in),
        .out(out_G)
    );

    kevin_D hvd
    (
        .in(in),
        .out(out_D)
    );

    kevin_B hvb
    (
        .in(in),
        .out(out_B)
    );

    initial begin
        in=0;
        for(i=0;i<16;i=i+1)begin
            #delay
            $display("time=%4d,in=%b,out_G=%b,out_D=%b,out_B=%b",$time,in,out_G,out_D,out_B);
            if((out_G === 1'bx | out_D === 1'bx | out_B === 1'bx | out_G === 1'bz | out_D === 1'bz | out_B === 1'bz) ||
               ((in == 1 || in == 5 || in == 6 || in == 7 || in == 9 || in == 10 || in == 12 || in == 14)&&(out_G & out_D & out_B) == 0) ||
               ((in != 1 && in != 5 && in != 6 && in != 7 && in != 9 && in != 10 && in != 12 && in != 14)&&(out_G | out_D | out_B) == 1))
                begin
                    $display("You got wrong answer!!");
                    $finish;
                end
            in = in+1;
        end
        $display("Congratulations!!");
        $finish;
    end

endmodule
```

After I finish creating my modules and testbench, I checked if my code works or not through a program called MobaXterm. We went through some steps to before we can start simulating our code, such as logging in, connecting to a server, uploading our files, etc. We then use the command “ncverilog kevin_tb.v kevin.v” to simulate our code, and see how it executes.

If the program stops midway, and the message “You got wrong answer!!” is printed, it means that your program is not successful at detecting the Kevin numbers, so there is something wrong within your code. On the other hand, if a message saying “Congratulations!!” is printed, it means that you have successfully created a Kevin number detector.

This is what the program looks like after I tried to simulate my modules with the testbench:

```
time=   5,in=0000,out_G=0,out_D=0,out_B=0
time=  10,in=0001,out_G=1,out_D=1,out_B=1
time=  15,in=0010,out_G=0,out_D=0,out_B=0
time=  20,in=0011,out_G=0,out_D=0,out_B=0
time=  25,in=0100,out_G=0,out_D=0,out_B=0
time=  30,in=0101,out_G=1,out_D=1,out_B=1
time=  35,in=0110,out_G=1,out_D=1,out_B=1
time=  40,in=0111,out_G=1,out_D=1,out_B=1
time=  45,in=1000,out_G=0,out_D=0,out_B=0
time=  50,in=1001,out_G=1,out_D=1,out_B=1
time=  55,in=1010,out_G=1,out_D=1,out_B=1
time=  60,in=1011,out_G=0,out_D=0,out_B=0
time=  65,in=1100,out_G=1,out_D=1,out_B=1
time=  70,in=1101,out_G=0,out_D=0,out_B=0
time=  75,in=1110,out_G=1,out_D=1,out_B=1
time=  80,in=1111,out_G=0,out_D=0,out_B=0
Congratulations!!
```

II. Problems Faced

There are several problems that I faced during the process of doing this project.

The biggest problem that I've faced is about understanding how to write in Verilog. Because Verilog is a new programming language for me, and it is not similar to the programming languages that I've learned before, I need to do some research about it first.

For example, at first, I didn't know about the concept of vectors in Verilog. But after doing some research on the internet and watching some tutorial videos on YouTube, I understand it now.

I'm also not accustomed to the data types in Verilog, but again, after doing some research on the internet and watching some tutorial videos on YouTube, I understand it better now.

III. Questions for TAs

Between the Gate level description, Dataflow description, and the Behavior description, which one is the most efficient and fastest module to use?