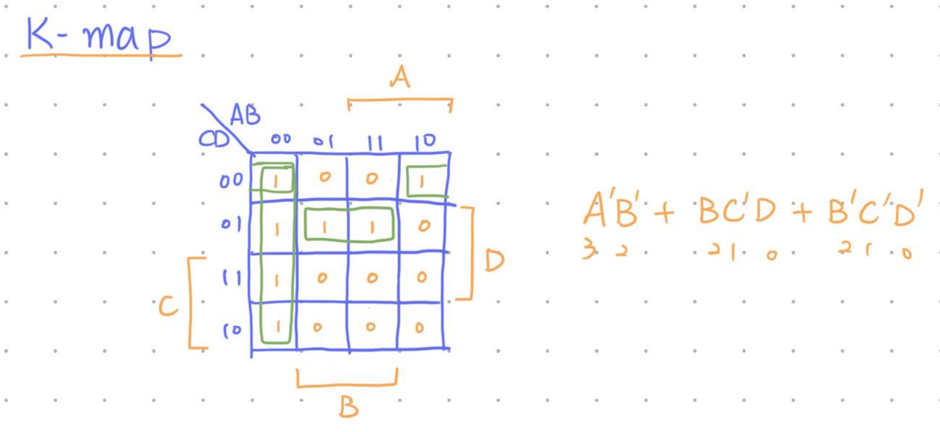
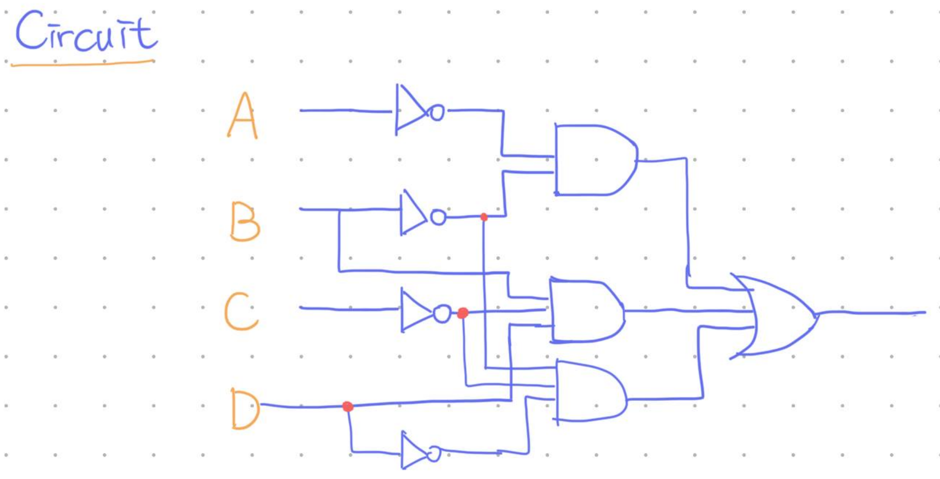
Lab1 Fibonacci number detector

Design Process

The main idea is use K-map to construct SOP, and use verilog to implement it.

Since Fibonacci numbers in range $[0, 15]$ are: $0, 1, 2, 3, 5, 8, 13$, so I write down some `1` in the corresponding block in K-map.





So now we know the result is equal to $A'B' + BC'D + B'C'D'$.

For ***Gate Level Description***, we need to use some logic gates to implement the circuit.
Since the not operation can be replace with `~`, so here I didn't use `not not0()` to get Not.
From K-map we can figure out we have three Products, so I create `a0`, `a1`, `a2`, and in the end sum them up will get the result.

```
module Fib_G(in, out);
        input [3:0] in;
        output out;

        wire a0, a1, a2;

        and and0(a0, ~in[3], ~in[2]);
        and and1(a1, in[2], ~in[1], in[0]);
        and and2(a2, ~in[2], ~in[1], ~in[0]);

        or or0(out, a0, a1, a2);
endmodule
```

For ***Dataflow Description***, we could just write the whole logical operation in single line, just like what tutorial says.

```
module Fib_D(in, out);
        input [3:0] in;
        output out;

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

For ***Behavior Description***, we could directly write the numbers that should output `1` in the code, when receive these values, let output be `1`, otherwise `0`.

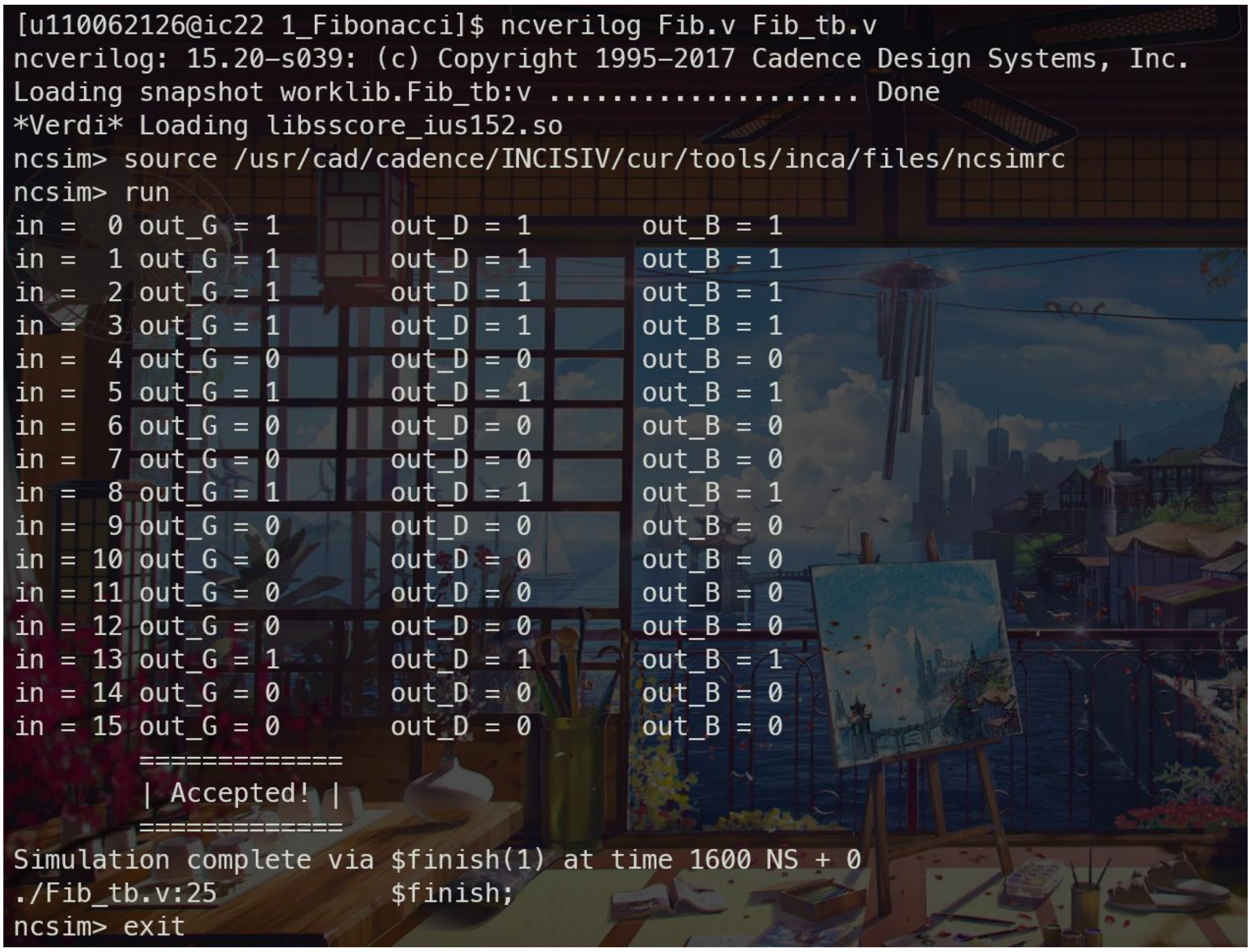
```
module Fib_B(in, out);
        input [3:0] in;
        output out;
        reg out;

        always@(*) begin
                case(in)
                        0, 1, 2, 3, 5, 8, 13: out = 1'b1;
                        default: out = 1'b0;
                endcase
        end
endmodule
```

Execution Result

I wrote a testbranch like this:

```
module Fib_tb;
        parameter delay = 100;

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

        wire isFib;
        assign isFib = in == 0 || in == 1 || in == 2 || in == 3 || in == 5 || in == 8 || in == 13;

        initial begin
                in = 0;
                repeat (16) begin
                        #delay
                        $display("in = %2d\tout_G = %1b\tout_D = %1b\tout_B = %1b", in, out_G, out_D, out_B);
                        if((isFib && (out_G & out_D & out_B) == 0) || (!isFib && (out_G | out_D | out_B) == 1))
                                begin
                                        $display("Wrong Answer!");
                                        $finish;
                                end
                        in = in + 1;
                end
                $display("\t=============");
                $display("\t| Accepted! |");
                $display("\t=============");
                $finish;
        end

        Fib_G FG(in, out_G);
        Fib_D FD(in, out_D);
        Fib_B FB(in, out_B);

endmodule
```

I directly write the correct fibonacci numbers in testbranch, and for every output check whether they are the same as the answer.
If there's wrong answer, it'll output `Wrong Answer!`, otherwise `Accepted!`



```
[u110062126@ic22 1_Fibonacci]$ ncverilog Fib.v Fib_tb.v
ncverilog: 15.20-s039: (c) Copyright 1995-2017 Cadence Design Systems, Inc.
Loading snapshot worklib.Fib_tb:v .................... Done
*Verdi* Loading libsscore_ius152.so
ncsim> source /usr/cad/cadence/INCISIV/cur/tools/inca/files/ncsimrc
ncsim> run
in =  0 out_G = 1       out_D = 1       out_B = 1
in =  1 out_G = 1       out_D = 1       out_B = 1
in =  2 out_G = 1       out_D = 1       out_B = 1
in =  3 out_G = 1       out_D = 1       out_B = 1
in =  4 out_G = 0       out_D = 0       out_B = 0
in =  5 out_G = 1       out_D = 1       out_B = 1
in =  6 out_G = 0       out_D = 0       out_B = 0
in =  7 out_G = 0       out_D = 0       out_B = 0
in =  8 out_G = 1       out_D = 1       out_B = 1
in =  9 out_G = 0       out_D = 0       out_B = 0
in = 10 out_G = 0       out_D = 0       out_B = 0
in = 11 out_G = 0       out_D = 0       out_B = 0
in = 12 out_G = 0       out_D = 0       out_B = 0
in = 13 out_G = 1       out_D = 1       out_B = 1
in = 14 out_G = 0       out_D = 0       out_B = 0
in = 15 out_G = 0       out_D = 0       out_B = 0
        =============
        | Accepted! |
        =============
Simulation complete via $finish(1) at time 1600 NS + 0
./Fib_tb.v:25           $finish;
ncsim> exit
```

The problem I faced

The main problem I faced is I accidentally type some wrong syntax, just use the error message to debug and find the solution on websites.

What I've learned in this Lab

From this lab, I first implement a logic circuit in verilog by myself, I know

- How to use input and output
- How to use `and`, `or`, `not` logic gates
- Difference between `~` and `!` in verilog
- Difference between `wire` and `reg`
- How to write a testbranch