



# Lab1 report




作者

周延儒 Bill Chou 103011227

digital logic design

林永隆

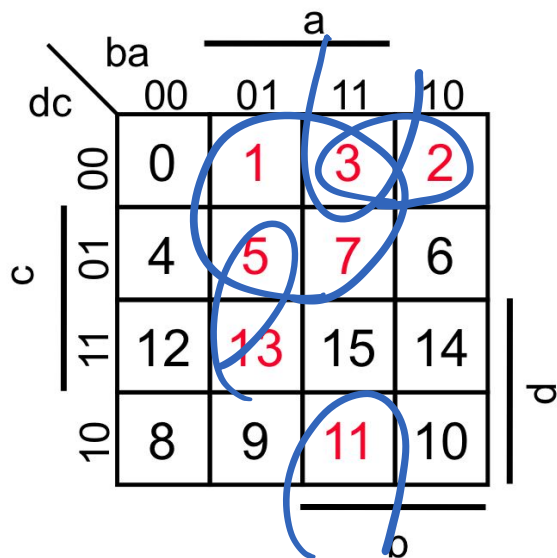


In this lab we are required to design the prime number detector(PND) in terms of 3 ways , gate\_level, data\_flow,and behavior\_modeling.

In the beginning, we have to analyze the PND. Since the outputs depends on the inputs only,so it the **combinational circuit** , compared with the sequential circuit ,where the outputs depend on the inputs and the previous inputs.

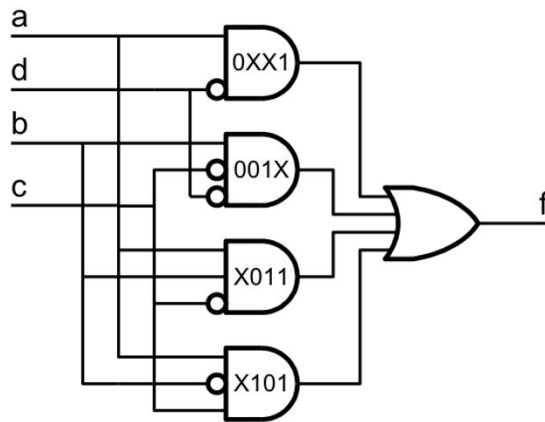
Then, we have to determine its truth table with K-map.

Because the prime number with 4 bits are 1,2,3,5,7,11,13, the K-map should be



$$F = (A \& \sim D) \mid (B \& \sim C \& \sim D) \mid (A \& \sim B \& C) \mid (A \& B \& \sim C)$$

Thus , the circuit diagram is



The prework is done , and then we can design the circuit with HDL , the Verilog

## 1. Gate\_level

```
module prime(in,p);
input [3:0]in;
wire notin0,notin1,notin2,notin3;
wire end0,end1,end2,end3;
output p;

not x1(notin0,in[0]);
not x2(notin1,in[1]);
not x3(notin2,in[2]);
not x4(notin3,in[3]);

and x5(end0,in[0],notin3);
and x6(end1,in[1],notin2,notin3);
and x7(end2,in[0],notin1,in[2]);
and x8(end3,in[0],in[1],notin2);
or result(p,end0,end1,end2,end3);

endmodule
```

```

[[dld043@ic21 ~]$ ncverilog prime_gate.v testbench.v
ncverilog: 14.10-s005: (c) Copyright 1995-2014 Cadence Design Systems, Inc.
file: prime_gate.v
    module worklib.prime:v
        errors: 0, warnings: 0
        Caching library 'worklib' ..... Done
    Elaborating the design hierarchy:
    Building instance overlay tables: ..... Done
    Generating native compiled code:
        worklib.test:v <0x7d0ddc01>
            streams: 2, words: 1389
    Building instance specific data structures.
    Loading native compiled code: ..... Done
    Design hierarchy summary:
            Instances  Unique
    Modules:           2      2
    Primitives:        9      3
    Registers:         1      1
    Expanded wires:    4      1
    Initial blocks:    1      1
    Pseudo assignments: 1      1
        Writing initial simulation snapshot: worklib.test:v
    Loading snapshot worklib.test:v ..... Done
*Verdi3* Loading libsscore_ius141.so
*Verdi3* : Enable Parallel Dumping.
ncsim> source /usr/cad/cadence/INCISIV/cur/tools/inca/files/ncsimrc
ncsim> run
input= 0,out= 0
input= 1,out= 1
input= 2,out= 1
input= 3,out= 1
input= 4,out= 0
input= 5,out= 1
input= 6,out= 0
input= 7,out= 1
input= 8,out= 0
input= 9,out= 0
input= 10,out= 0
input= 11,out= 1
input= 12,out= 0
input= 13,out= 1
input= 14,out= 0
input= 15,out= 0
Simulation complete via $finish(1) at time 160 NS + 0
./testbench.v:20      $finish;
ncsim> exit

```

explanation:

Since we have 4 bit input, and 1 bit p, so we declare them

as input [3:0]in; output p , and the rest are nets , denoted

```
as wire notin0,notin1,notin2,notin3;  
    wire end0,end1,end2,end3;
```

the only thing we have to do is implement the design  
with all types of gates .

## 2. date\_flow

```
module prime(in,p);  
    input [3:0]in;  
    output p;  
  
    assign p= (in[0] & ~in[3]) | (in[1] & ~in[2] & ~in[3]) |  
    ( in[0] & ~in[1] & in[2]) | (in[0] & in[1] & ~in[2] ) ;  
  
endmodule
```

```

[dld043@ic21 ~]$ ncverilog prime_df.v testbench.v
ncverilog: 14.10-s005: (c) Copyright 1995-2014 Cadence Design Systems, Inc
file: prime_df.v
    module worklib.prime:v
        errors: 0, warnings: 0
        Caching library 'worklib' ..... Done
    Elaborating the design hierarchy:
    Building instance overlay tables: ..... Done
    Generating native compiled code:
        worklib.prime:v <0x1b3bbc15>
            streams: 1, words: 343
    Building instance specific data structures.
    Loading native compiled code: ..... Done
    Design hierarchy summary:
        Instances Unique
    Modules:          2      2
    Registers:         1      1
    Scalar wires:      1      -
    Vectored wires:    1      -
    Initial blocks:    1      1
    Cont. assignments: 1      1
    Pseudo assignments: 1      1
    Writing initial simulation snapshot: worklib.test:v
    Loading snapshot worklib.test:v ..... Done
*Verdi3* Loading libsscore_ius141.so
*Verdi3* : Enable Parallel Dumping.
ncsim> source /usr/cad/cadence/INCISIV/cur/tools/inca/files/ncsimrc
ncsim> run
input= 0,out= 0
input= 1,out= 1
input= 2,out= 1
input= 3,out= 1
input= 4,out= 0
input= 5,out= 1
input= 6,out= 0
input= 7,out= 1
input= 8,out= 0
input= 9,out= 0
input= 10,out= 0
input= 11,out= 1
input= 12,out= 0
input= 13,out= 1
input= 14,out= 0
input= 15,out= 0
Simulation complete via $finish(1) at time 160 NS + 0
./testbench.v:20      $finish;
ncsim> exit
_

```

explanation:

In the second design , we can use more convenient way with continuous assignment , which are the most basic assignment in dataflow modeling.

With continuous assignment , we can model the circuit via Boolean logic rather than gate connections

### 3. Behavior modeling

```
module prime(in,p);  
  input [3:0]in;  
  output reg p;  
  
  always@(*)begin  
    case (in)  
      0   : p=0;  
      1   : p = 1;  
      2   : p= 1;  
      3   : p = 1;  
      4   : p = 0;  
      5   : p = 1;  
      6   : p= 0;  
      7   : p = 1;  
      8   : p=0;  
      9   : p = 0;  
      10  : p = 0;  
      11  : p = 1;  
      12  : p = 0;  
      13  : p = 1;  
      14  : p = 0;  
      default : p = 0;  
    endcase  
  
  end  
endmodule
```

```

[[dld043@ic21 ~]$ ncvverilog prime_behavior.v testbench.v
ncvverilog: 14.10-s005: (c) Copyright 1995-2014 Cadence Design Systems, Inc.
file: prime_behavior.v
    module worklib.prime:v
        errors: 0, warnings: 0
        Caching library 'worklib' ..... Done
    Elaborating the design hierarchy:
    Building instance overlay tables: ..... Done
    Generating native compiled code:
        worklib.prime:v <0x6c7133ea>
            streams: 1, words: 287
    Building instance specific data structures.
    Loading native compiled code: ..... Done
    Design hierarchy summary:
            Instances  Unique
    Modules:           2      2
    Registers:          2      2
    Scalar wires:       1      -
    Vectored wires:     1      -
    Always blocks:      1      1
    Initial blocks:     1      1
    Pseudo assignments: 1      1
        Writing initial simulation snapshot: worklib.test:v
    Loading snapshot worklib.test:v ..... Done
*Verdi3* Loading libsscore_ius141.so
*Verdi3* : Enable Parallel Dumping.
ncsim> source /usr/cad/cadence/INCISIV/cur/tools/inca/files/ncsimrc
ncsim> run
input= 0,out= 0
input= 1,out= 1
input= 2,out= 1
input= 3,out= 1
input= 4,out= 0
input= 5,out= 1
input= 6,out= 0
input= 7,out= 1
input= 8,out= 0
input= 9,out= 0
input= 10,out= 0
input= 11,out= 1
input= 12,out= 0
input= 13,out= 1
input= 14,out= 0
input= 15,out= 0
Simulation complete via $finish(1) at time 160 NS + 0
./testbench.v:20      $finish;
ncsim> exit

```

explanation:

behavior modeling in Verilog is Higher level of modeling where behavior of logic is modeled.

Since the assignment statements assign values to reg,the output must be declared as reg (register), which we will learn in the sequential circuit.



Moreover , we declare the always block always@(\*), since it is the combinational circuit rather than sequential circuit where it may be always@(posedge clk), or always@(posedge clk or negedge rst) etc.

Plus , we use =(blocking) Assignments which happen sequentially instead of <= (non-blocking) Assignments which happen in sequence and we use always@(\*) because we want to infer an element(s) that changes its value as soon as one or more of its inputs change

#### 4. Testbench

```
module test;

    // Declare your input regs and output wires here
    reg [3:0] in;
    wire p;
    // Initiate your modules here
    prime test(in,p);

    initial begin
        in=0;
        repeat(16)begin
            #10
            $display("input= %d,out= %b",in,p);
            in=in+1;
        end
        $finish;
    end

endmodule
```

explanation:

In RTL design, it is crucial to design the Testbench to verify our design, so we design our own Testbench.

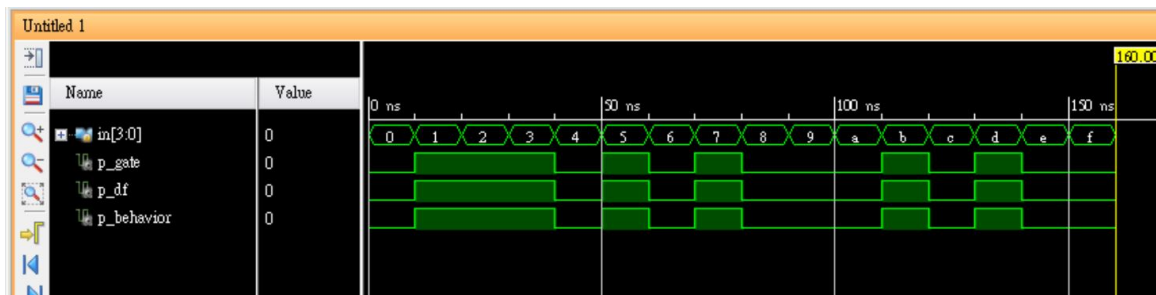
Because there are  $2^4=16$  cases, so we use `repeat(16)` to repeat for 16 times, and also we can rewrite as

```
Integer i  
for(i=0;i<16;i++)
```

or

```
while(i<16)
```

Waveform:



(I generated the waveform by the vivado, a Verilog tool)

## Reviews:

In this lab, we learn to manipulate the hardware description languages to design the circuit .

To design the circuits, it follows the steps:

Determine the design , analyze the design ,draw the truth table, simplify it, and use Verilog to implement the circuit.

With larger and complicated circuit, the behavior modeling can help reduce the time , so the behavior modeling will be very important in the following weeks

Last but not least , I' ve actually taken the Hardware Design and Lab last semester, so it is not hard to design the all the circuit with Verilog . Hope we can learn more in the future in this coarse.