Logic Design

EECS101001

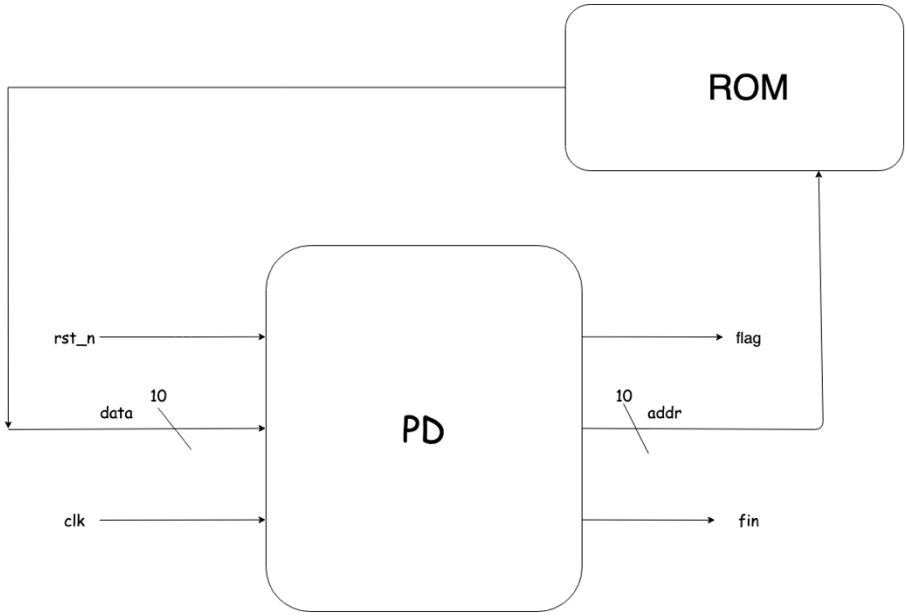
Lab 3: Pattern Detector – Report

104021215 熊 磊

1. Foreword

This Lab is trying to design a Pattern Detector. When we detect the following pattern, the machine should set the flag to 1.

$$110(010)^{+}10$$



Input

name	width	function
clk	1-bit	clock signal
rst_n	1-bit	當 rst_n = 0 的時候，reset detector 並從下個 cycle 開始進行運算
data	10-bit	從 ROM 讀出來的訊號，我們的 bit pattern 會放在 data[0]

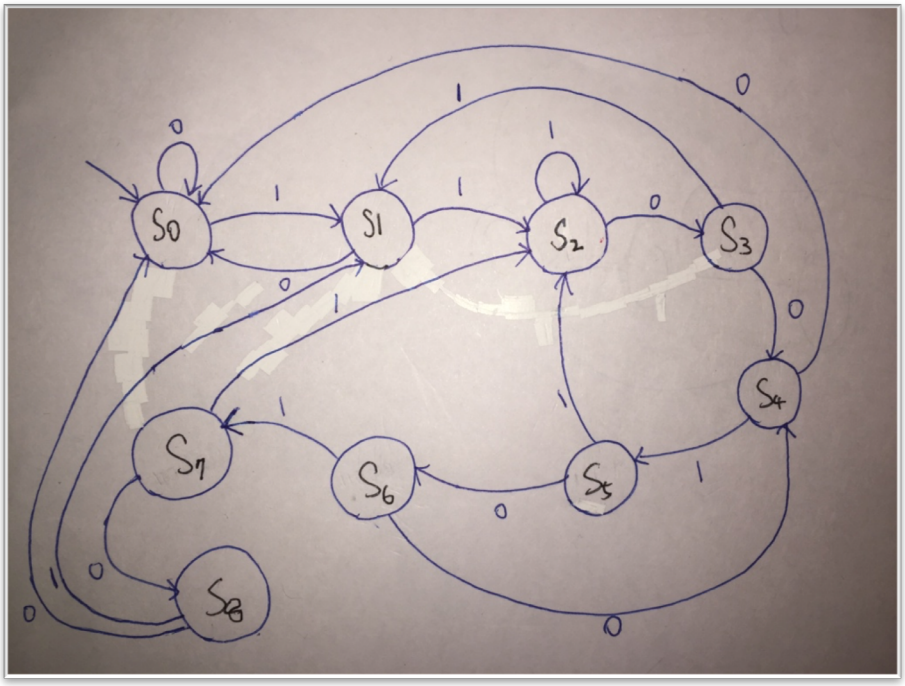
Output

name	width	function
flag	1-bit	當我們讀取到一個 pattern 的尾端的 cycle 拉成 1
addr	10-bit	控制我們要從 ROM 裡面的哪個位置讀取值，有效資料的長度會放在 addr = 1023
fin	1-bit	當運算結束（偵測完有效資料之後）設成 1

2. Implementation Details and Designs

Pattern Detector

First, we draw a State Transition Graph as Follow:



We need at least 9 States to detect the pattern, hence we must define states with at least 4 bits. And we use another two states, INIT and FINI, for initial and finish state.

```
`define S0 4'b0000
`define S1 4'b0001
`define S2 4'b0010
`define S3 4'b0011
`define S4 4'b0100
`define S5 4'b0101
`define S6 4'b0110
`define S7 4'b0111
`define S8 4'b1000
`define INIT 4'b1110
`define FINI 4'b1111
```

Since, we have to design an independent DFF outside the PD module, we design a more powerful DFF, with `rst_n` and `init_out`.

The code is design like this:

```
module DFF (clk, init_out, next_out, out, rst_n);
    parameter n = 1;
    input clk,rst_n;
    input [n-1:0] next_out,init_out;
    output reg [n-1:0] out;
    always@(posedge clk, negedge rst_n) begin
        if(rst_n==0)begin
            out <= init_out;
        end
        else begin
            out <= next_out;
        end
    end
endmodule
```

`out` will be assign to `init_out` when `rst_n` is 0, otherwise is `next_out`. We have to use DFF to remember the state. Another reason that why using DFF to set up value is DFF is more stable for hardware description language.

> ***Observation:*** *If I assign value without using DFF, e.g.* `counter`*, the value will be wrong value.*

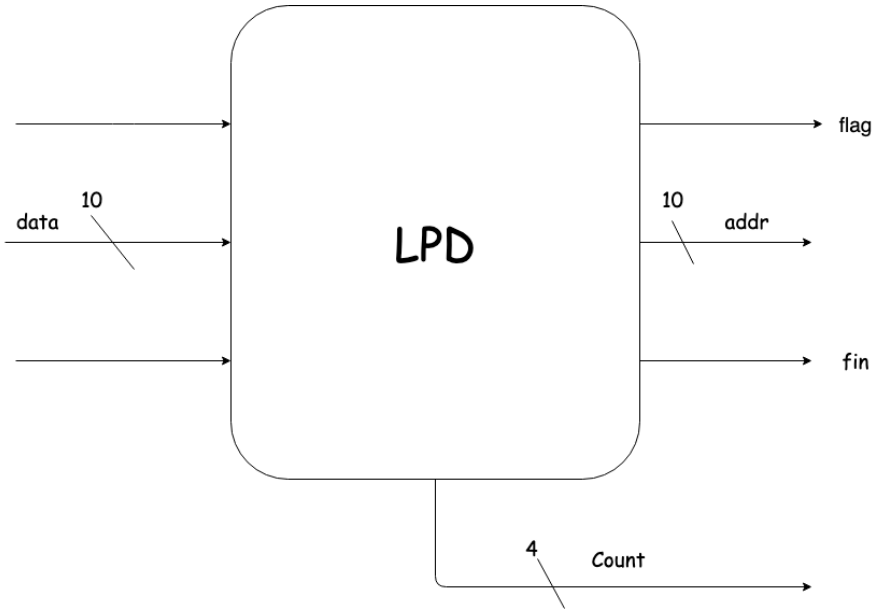
The remain part is the PD module; we introduce the design choice first. Since we can't assign value in `always` block, they need to be declared as wire.

```
reg [9:0]length;
wire [9:0]next_addr;
wire [3:0]state,next_state;
reg [3:0]next_state1, init_state;

DFF #(4) DFF1(clk, `INIT, next_state, state, rst_n);
DFF #(10) DFF2(clk, 10'd1023 ,next_addr, addr, rst_n);

assign next_addr = (state == `INIT)? 1'b0 : addr+1;
assign fin = (state ==  `FINI) ? 1'b1 : 1'b0;
assign next_state = (addr == length)? `FINI : next_state1;
```

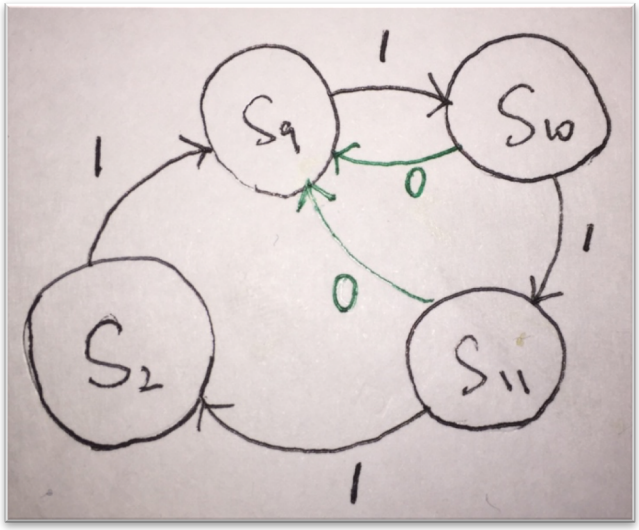
```
48      always @(*)
49      begin
50          case(state)
51              `INIT:begin
52                  next_state1 = `S0;
53                  length = data;
54                  flag = 1'b0;
55              end
56
57              `FINI:begin
58                  next_state1 = `FINI;
59                  flag = 1'b0;
60              end
61
62              `S0:begin
63                  next_state1 = (data[0]==1)? `S1 : `S0;
64                  flag = 1'b0;
65              end
66
67              `S1:begin
68                  next_state1 = (data[0]==1)? `S2 : `S0;
69                  flag = 1'b0;
70              end
```

We take a small part of `always` block for explanation; remaining part is similar. As the State Transition Graph, we drew above; `next_state` of Initial state is `S0`. `flag` will only be assigned to 1 in a moment when state `S7` transiting to `S8`.

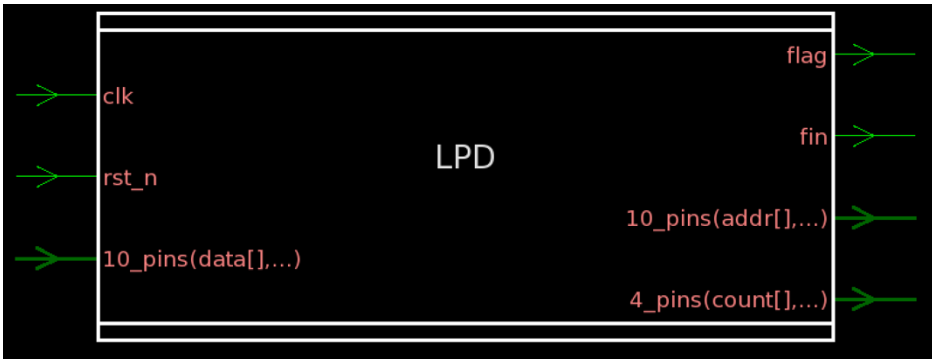
Luxury Pattern Detector



Since for Luxury Pattern Detector, we need to detect one more pattern `(111)`, it takes three more states. It can, however, be extend from `S2` we designed before in PD.




```
`define S0 4'b0000
`define S1 4'b0001
`define S2 4'b0010
`define S3 4'b0011
`define S4 4'b0100
`define S5 4'b0101
`define S6 4'b0110
`define S7 4'b0111
`define S8 4'b1000
`define S9 4'b1001
`define S10 4'b1010
`define S11 4'b1011

`define INIT 4'b1110
`define FINI 4'b1111
```



On the other hand, as we need to count how many `(010)` has been detected and stop counting when we detect the pattern `(111)`.

We set the increment of `counter` at state `S6`.

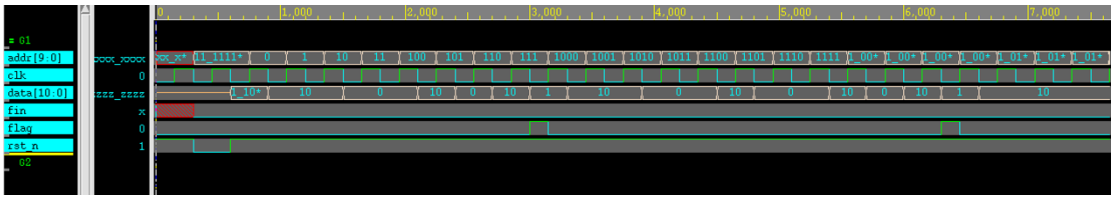
```
`S6:begin
    next_state1 = (data[0]==1)? `S7 : `S4;
    flag = 1'b0;
    next_cnt = cnt + 4'b1;
end
```

It is very clear after we drew the State Transition Graph above. The point that worth to be emphasize is that we should set count to total number of patterns at the end of the valid digits.

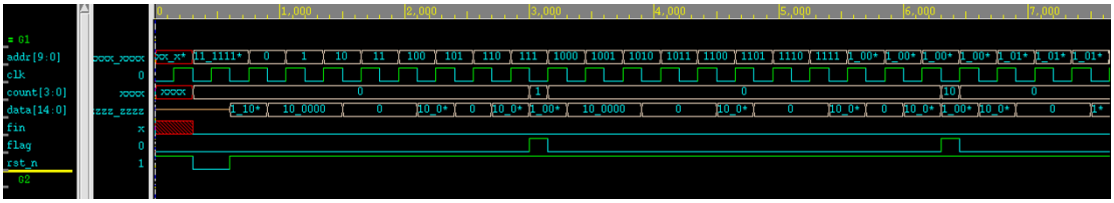
```
DFF #(4) DFF1(clk, `INIT, next_state, state, rst_n);
DFF #(10) DFF2(clk, 10'd1023 ,next_addr, addr, rst_n);
DFF #(4) DFF3(clk, 4'b0, next_cnt, cnt, rst_n);
DFF #(4) DFF4(clk, 4'b0, next_total, total, rst_n);

assign next_addr = (state == `INIT)? 1'b0 : addr+1;
assign fin = (state ==  `FINI) ? 1'b1 : 1'b0;
assign next_state = (addr == length)? `FINI : next_state1;
assign count = (state == `FINI)? total : (flag==1'b1) ? cnt : 4'b0;
assign next_total = (state ==`S8) ? total+1 : total;
```

3. nWave Result

Pattern Detector



Luxury Pattern Detector



4. Report Area

Pattern Detector

```
****************************************
Report : area
Design : PD
Version: K-2015.06-SP1
Date   : Tue Apr 30 03:27:45 2019
****************************************

Library(s) Used:

    slow (File: /theda21_2/CBDK_IC_Contest/cur/SynopsysDC/db/slow.db)

Number of ports:                           90
Number of nets:                           219
Number of cells:                          136
Number of combinational cells:            108
Number of sequential cells:                25
Number of macros/black boxes:               0
Number of buf/inv:                         24
Number of references:                      22

Combinational area:               840.212993
Buf/Inv area:                      91.659599
Noncombinational area:            692.539211
Macro/Black Box area:               0.000000
Net Interconnect area:      undefined  (No wire load specified)

Total cell area:                 1532.752204
Total area:                 undefined
```

Luxury Pattern Detector

```
****************************************
Report : area
Design : LPD
Version: K-2015.06-SP1
Date   : Tue Apr 30 03:23:20 2019
****************************************

Library(s) Used:

    slow (File: /theda21_2/CBDK_IC_Contest/cur/SynopsysDC/db/slow.db)

Number of ports:                          122
Number of nets:                           326
Number of cells:                          217
Number of combinational cells:            178
Number of sequential cells:                34
Number of macros/black boxes:               0
Number of buf/inv:                         45
Number of references:                      30

Combinational area:              1290.023987
Buf/Inv area:                     169.739997
Noncombinational area:            991.281612
Macro/Black Box area:               0.000000
Net Interconnect area:      undefined  (No wire load specified)

Total cell area:                 2281.305599
Total area:                 undefined
```

5. Encountered Problems and Solving

- We can assign address with DFF.
- If I assign value without using DFF, e.g. `counter`, the value will be wrong value.

6. Questions

- Why I have to use `negedge` here, since I will get error if I set `posedge`?

```
module DFF (clk, init_out, next_out, out, rst_n);
    parameter n = 1;
    input clk,rst_n;
    input [n-1:0] next_out,init_out;
    output reg [n-1:0] out;
    always@(posedge clk, negedge rst_n) begin
        if(rst_n==0)begin
            out <= init_out;
        end
        else begin
            out <= next_out;
        end
    end
endmodule
```

7. Impression and Experience

   This time is trying to use DFF to design a so-called Finite State Machine. DFF is really special, and also amazing module for me. Since it can help machine to remember and also make it more stable!