106061151 劉安得

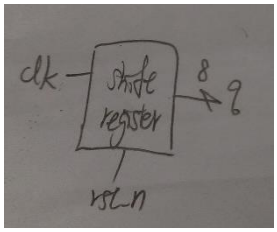
Prelab4 & Lab4-1.

1. **Design Specification**

   8-bits shift register

   Input : clk, rst_n

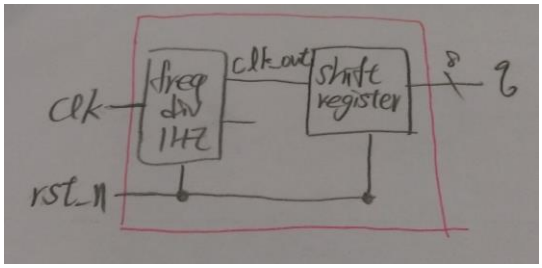
   Output : q[7:0]



2. **Design implementation**

   使用 prelab 所寫的 shift register 和 lab3-2 的 1HZ frequency divider 來 implement

   Declare a wire “clk_out”, to connect shift register and 1HZ frequency divider

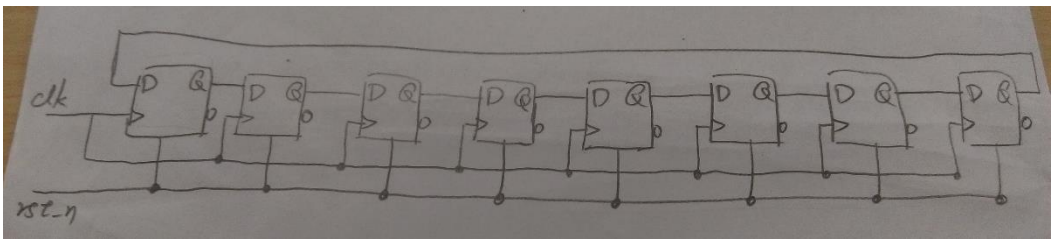


   shift shifter

   Connect the out Q to input D

   If(rst_n = 0), q = 01010101

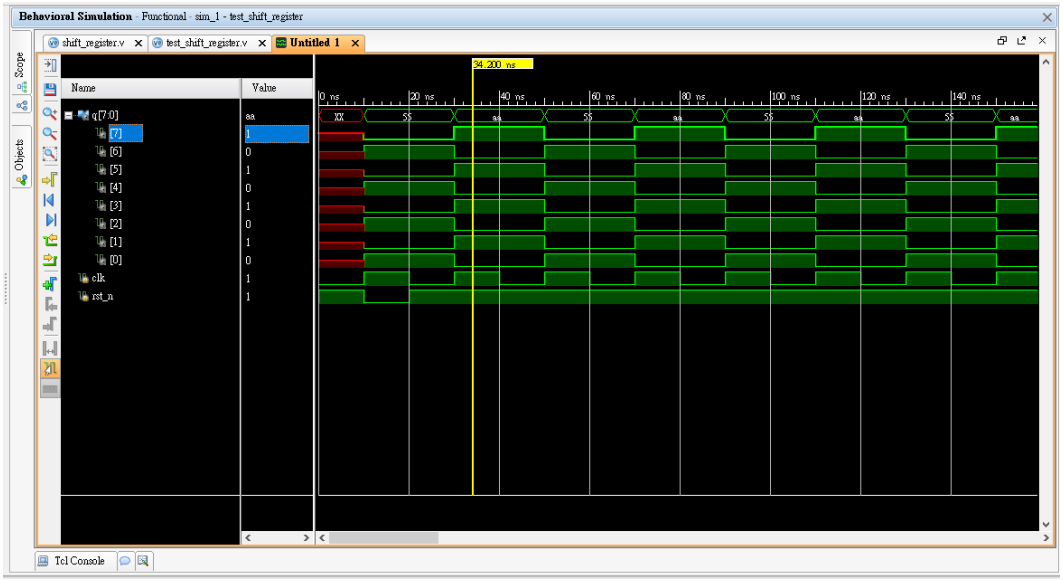
   Else, q shift right 1-bit



I/O	Q[7]	Q[6]	Q[5]	Q[4]	Q[3]	Q[2]	Q[1]	Q[0]	Clk	Rst_n
Pin	V14	U14	U15	W18	V19	U19	E19	U16	W5	V17

3. Simutation

Shift register



01010101 -> 10101010 -> 01010101 repeat

4. discussion

看似短短的 verilog code，其實裡面有 implement 到 8 個 DFF。之後應該還要寫個 load 功能，讓他可以從外面賦值，可以暫存外面所存入的值。

5. conclusion

做完這次實驗，讓我學會如何 implement 簡單的 shift register

6. reference

04_verilog(3) p.12

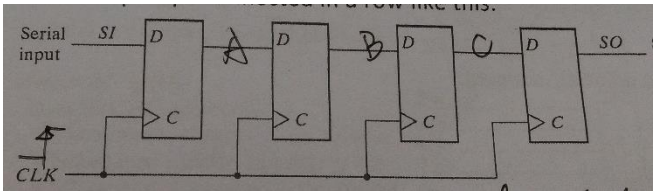
the Verilog of shift register

```
`define BIT_WIDTH 4
module shifter(
  q,  // shifter output
  clk,  // global clock
  rst_n  // active low reset
);

output [`BIT_WIDTH-1:0] q;  // output
input clk;  // global clock
input rst_n;  // active low reset

reg [`BIT_WIDTH-1:0] q;  // output

// Sequential logics: Flip flops
always @(posedge clk or negedge rst_n)
  if (~rst_n)
  begin
    q<=`BIT_WIDTH'b0101;
  end                    initial value 0101
  else
  begin
    q[0]<=q[3];
    q[1]<=q[0];
    q[2]<=q[1];
    q[3]<=q[2];
  end
endmodule
```

Logic Design Lecture 8 Registers and Counters

The schematic of shift register

這個 shift register 只有 4-bit，並且有 serial input



the Verilog of shift register

同樣的，它也只有 4-bits，並且有 sin 這個 serial input。還有一點不同處是，它 reset 完，全部設為 0

```
module Shifter_Register1(clk, rst, sin, state);
parameter n=4;
input clk;
input rst;
input sin;
output [n-1:0] state;

wire [n-1:0] next = rst ? {n{1'b0}} :
{state[n-2:0], sin};

DFF #(n) cnt(.clk(clk), .in(next), .out(out));

endmodule
```

sin

```
module Shifter_Register2(clk, rst, sin, state);
parameter n=4;
input clk;
input rst;
input sin;
output [n-1:0] state;

wire [n-1:0] next = {state[n-2:0], sin};

always @(posedge clk)
if (rst)
 state <= n'b0;
else
 state <= next;

endmodule
```

serial in

EE228001 Fall 2017

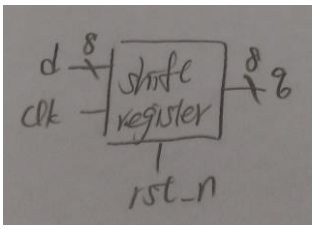
Lab4-2.

1. **Design Specification**

   8-bits shift register with parallel load

   Input : d[7:0], clk, rst_n

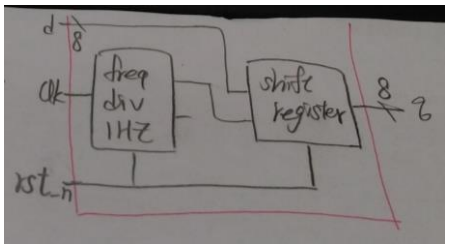
   Output : q[7:0]



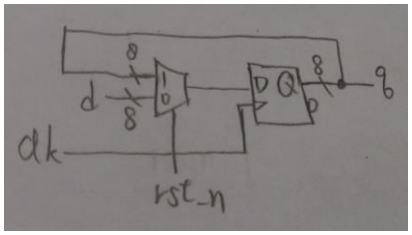
2. **Design implementation**

   修改 lab4-1 的 shift register

   當 rst_n = 0 時，不再 reset 為 01010101，而是 d



   If(rst_n = 0), q = d

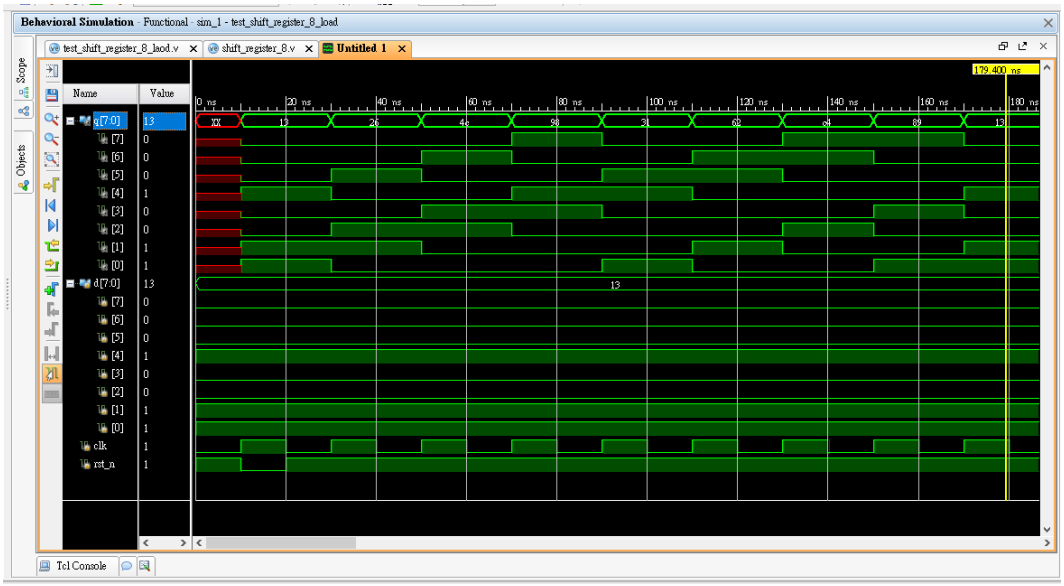


I/O	Q[7]	Q[6]	Q[5]	Q[4]	Q[3]	Q[2]	Q[1]	Q[0]	Clk	Rst_n
Pin	V14	U14	U15	W18	V19	U19	E19	U16	W5	V17

I/O	D[7]	D[6]	D[5]	D[4]	D[3]	D[2]	D[1]	D[0]
Pin	W13	W14	V15	W15	W17	W16	V16	V17

3. **Simutation**

   Shift register



The initial value is 00010011

4. discussion

這個子實驗，新增了 parallel load，讓 shift register 的可用性增加。

我在 top module 叫出其他 module 的時候，可能會因為 bits 位數不對而產生 bug，這方面在打好 verilog 後，要一再檢查

5. conclusion

做完這次實驗，讓我學會如何 implement shift register with parallel load

6. reference

04_verilog(3) p.12

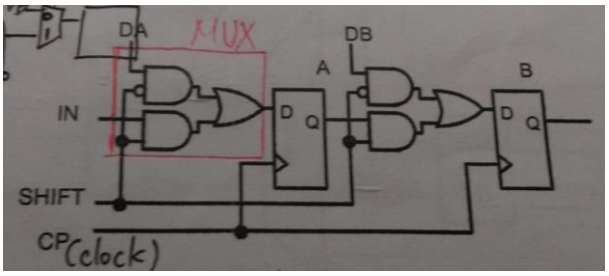
the Verilog of shift register，不過它沒 parallel load

```
`define BIT_WIDTH 4
module shifter(
  q, // shifter output
  clk, // global clock
  rst_n // active low reset
);

output [`BIT_WIDTH-1:0] q; // output
input clk; // global clock
input rst_n; // active low reset

reg [`BIT_WIDTH-1:0] q; // output

// Sequential logics: Flip flops
always @(posedge clk or negedge rst_n)
  if (~rst_n)
  begin
    q<=`BIT_WIDTH'b0101;
  end                initial value 0101
  else
  begin
    q[0]<=q[3];
    q[1]<=q[0];
    q[2]<=q[1];
    q[3]<=q[2];
  end
endmodule
```

Logic Design Lecture 8 Registers and Counters

Shifter with parallel load

不過它只有 1-bits，而這次 lab，我做的是 8-bits



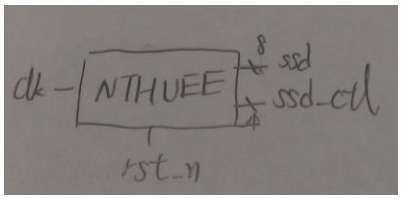
Lab4-3.

1. **Design Specification**

   NTHUEE scroll display

   Input : clk, rst_n

   Output : ssd[7:0], ssd_ctl[3:0]

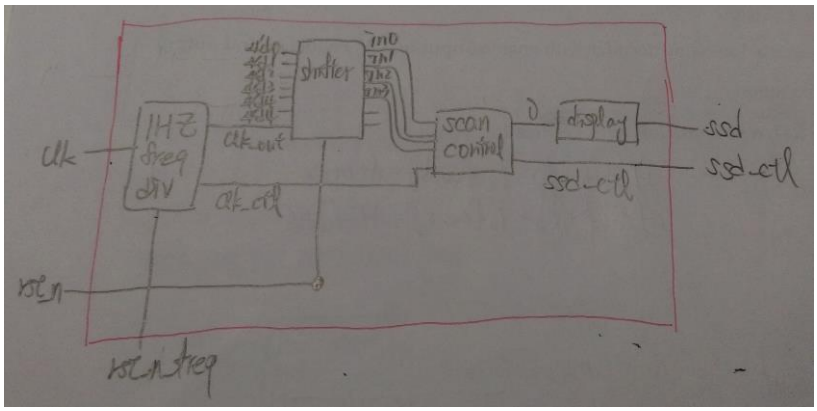


2. **Design implementation**

   如同之前的 lab 需要七段顯示器時一樣，需要 1HZ frequency divider & scan control。除此之外，再自己額外寫一個可以 display'N' 'T' 'H' 'U' 'E'的 ssd display，以及 6-bits BCD shift register

   我分別將 NTHUE 用 BCD 的 0,1,2,3,4 來當代號，以方便 scan control & ssd display implement

   以下是各 module 間 wire 的連接，與 lab3 bonus 相同，必須在一開始就重設 frequency divider，否則 reset 的時候不會同時顯示 NTHU



I/O	Ssd_ctl[3]	Ssd_ctl[2]	Ssd_ctl[1]	Ssd_ctl[0]	Clk	Rst_n
Pin	W4	V4	U4	U2	W5	V17

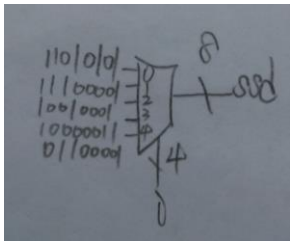
I/O	ssd[7]	ssd[6]	ssd[5]	ssd[4]	ssd[3]	ssd[2]	ssd[1]	ssd[0]
Pin	W7	W6	U8	V8	U5	V5	U7	V7

Ssd display

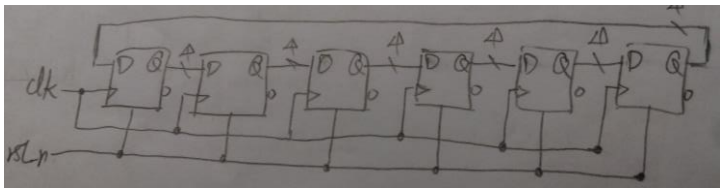
8'b11010101 represent 'N'

8'b11100001 represent ‘T’
8'b10010001 represent ‘H’
8'b10000011 represent ‘U’
8'b01100001 represent ‘E’

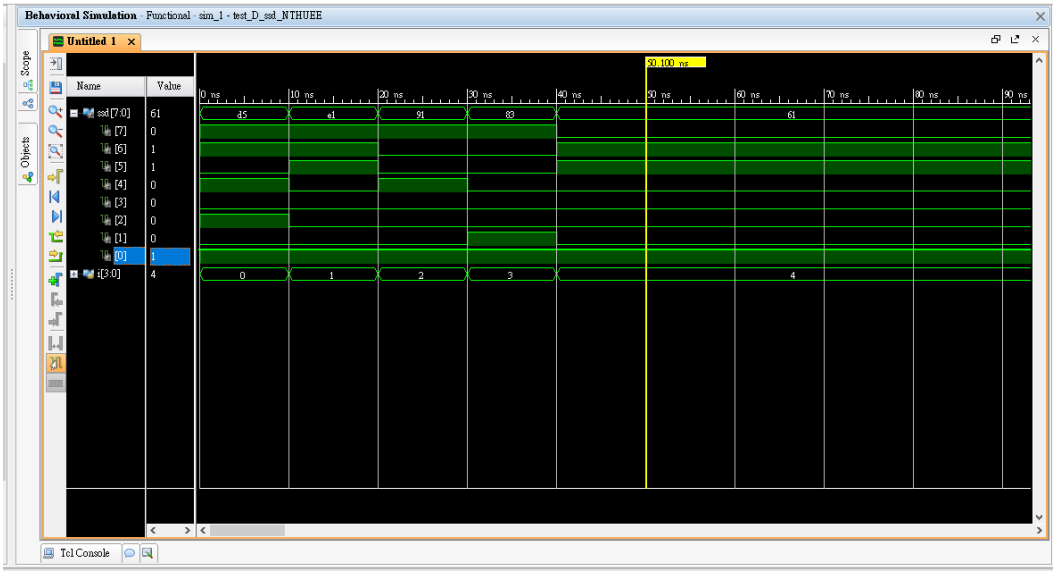
Case(i)
0(N) : 8'b11010101
1(T) : 8'b11100001
2(H) : 8'b10010001
3(U) : 8'b10000011
4(E) : 8'b01100001



BCD shift register
If(rst_n = 0), BCD = d
Else, BCD shift left 1-bit

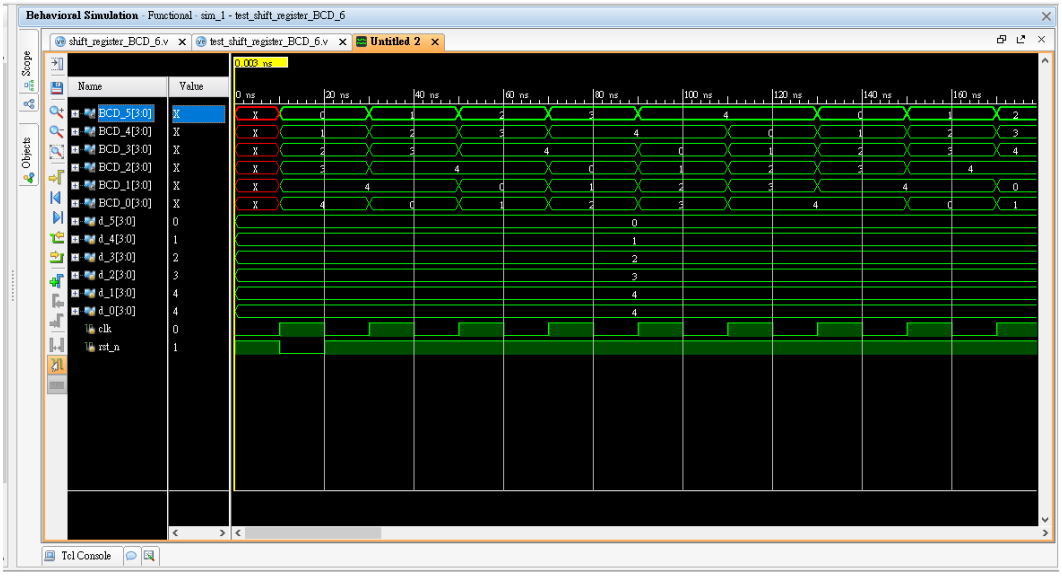


3. **Simutation**
   Ssd display



I[3:0]	Ssd[7:0]
0000(0)	11010101(N)
0001(1)	11100001(T)
0010(2)	10010001(H)
0011(3)	10000011(U)
0100(4)	01100001(E)

BCD shift register



012344 shift left repeatly

4. Discussion

我發現以後每次使用 frequency divider implement scan control 時，都需要在一開始就 reset frequency divider

在打 module 的時候，可以先預想以後可能還會用到哪些功能，這樣未來才

有更佳的可用性，例如我這次 lab，就寫了 BCD 6-bits shifter 並有 6 個 output，而非純粹應題目需求只有 4 個 output

5. **conclusion**

做完這個實驗，讓我學會如何做字元的 shifter，並將 shifter 與七段顯示器結合，做到這裏我覺得越做越有成就感，而不像以前只亮個 LED，或者只有數字稍微改變一下

6. **reference**

同 lab4-1, lab4-2

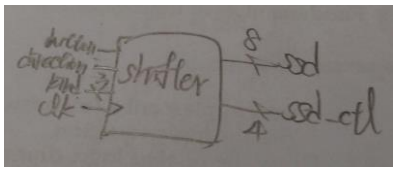
Lab4-4.

1. **Design Specification**

   Shifter which can be controlled shift left/right, shift functional/arithmetic/barrel

   Input : button, direction, kind [2:0], clk

   Output : ssd[7:0], ssd_ctl[3:0]



2. **Design implementation**

   先寫三種不同 type 的 shifter，再加上之前寫的 1HZ frequency divider, scan control & ssd display

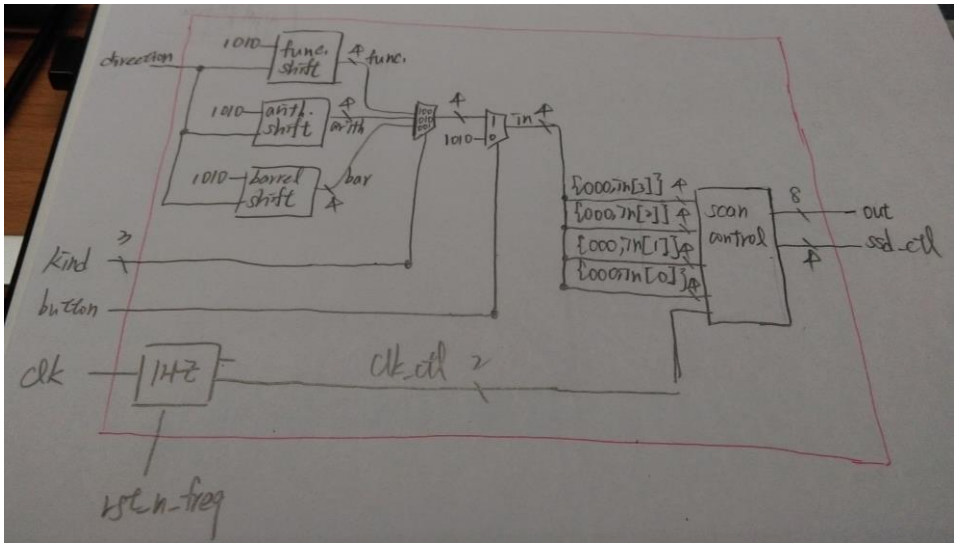
   Direction = 1, shift left; direction = 0, shift right

   Kind = 3'b100, functional shift

   Kind = 3'b010, arithmetic shift

   Kind = 3'b001, barrel shift

   以下是各 module 間 wire 的連接，同樣一開始要重設 frequency divider



I/O	button	direction	Kind[2]	Kind[1]	Kind[0]	Clk
Pin	U18	R2	W16	V16	V17	W5

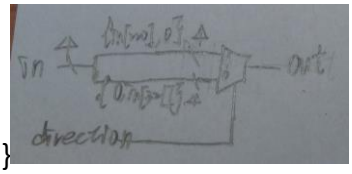
I/O	Ssd_ctl[3]	Ssd_ctl[2]	Ssd_ctl[1]	Ssd_ctl[0]	Clk
Pin	W4	V4	U4	U2	W5

I/O	ssd[7]	ssd[6]	ssd[5]	ssd[4]	ssd[3]	ssd[2]	ssd[1]	ssd[0]
Pin	W7	W6	U8	V8	U5	V5	U7	V7

Functional shift

If(direction = 1), out = {in[2:0], 1'b0}

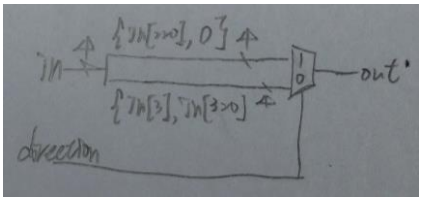
Else, out = {1'b0, in[3:1]}



arithmetic shift

If(direction = 1), out = {in[2:0], 1'b0}

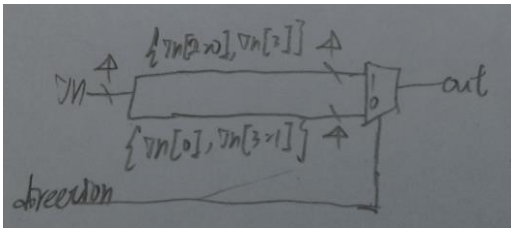
Else, out = {in[3], in[3:1]}



barrel shift

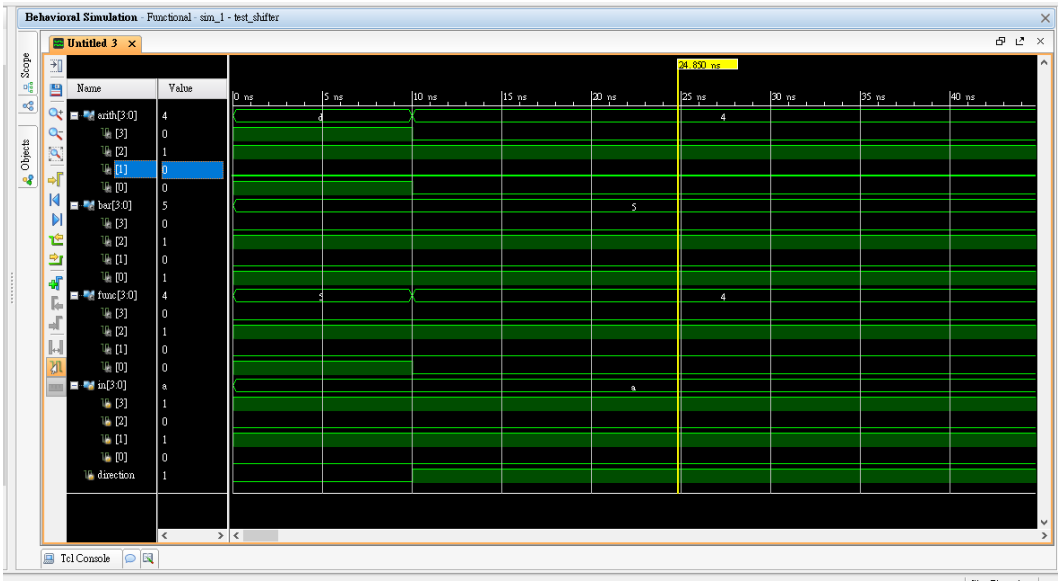
If(direction = 1), out = {in[2:0], in[3]}

Else, out = {in[0], in[3:1]}



3. Simutation

因為三個 shifter 功能很像，所以我一次做三個 module 的 testbench



Direction	in	arithmetic	barrel	functional
0(right)	1010	1101	0101	0101
1(left)	1010	0100	0101	0100

4. Discussion

我一開始以為自己設左邊的 DIP 為 kind，右邊的是 direction，後來發現季相反，讓我一度以為有 bug

這次因為只要跑一次，所以 shifter 不需要接 clock，在之後對 button 有更多了解後，可能可以做出按著就一直 shift，或是按一下便開始動，再按一下便停止，或者是按一下動一次，的 shifter

這個 lab 最難的地方在於一開始的構想，想要怎麼 implement，在把 block diagram 畫出來後，就會順利許多

5. conclusion

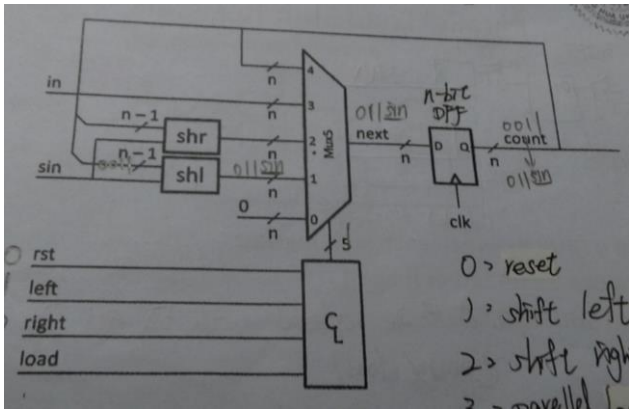
這次實驗是第一個用到 button 的 lab，也讓我了解到三種不同的 shift 方法。

6. reference

Logic Design Lecture 8 Registers and Counters

The schematic of right/left/load shifter

雖然和我們這次 lab 看似毫無相關，但其實有許多相同之處，首先它同樣可以改變 shift 的方向。再者我 implement 不同種類的 shifter 的時候，也與這張圖一樣，使用 MUX 來選擇



The verilog of right/left/load shifter

```
module LRL_Shifter_Register1(clk, rst, left, rig
parameter n=4;
input clk, rst, left, right, load, sin;
input [n-1:0] in;
output [n-1:0] out;
reg [n-1:0] next;

DFF #(n) cnt(.clk(clk), .in(next), .out(out));

always @* begin
casex({rst,left,right,load})
 4'b1xxx: next = n'b0;                  //reset
 4'b01xx: next = {out[n-2:0],sin};  // left
 4'b001x: next = {sin,out[n-1:1]};  // right
 4'b0001: next = in;                    // load
 default: next = out;                   // hold
endcase
end
endmodule
```