DIC Design Guide

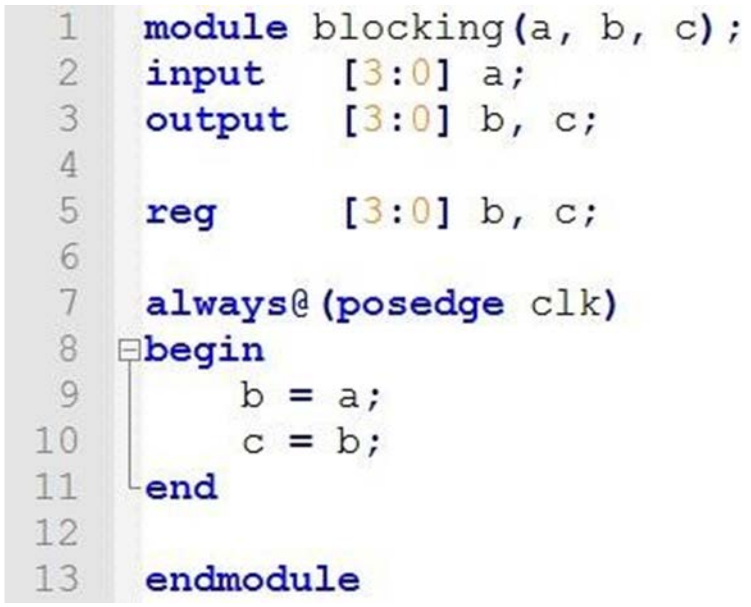
指導教授：卿文龍

助教：張正宗、陳品崴

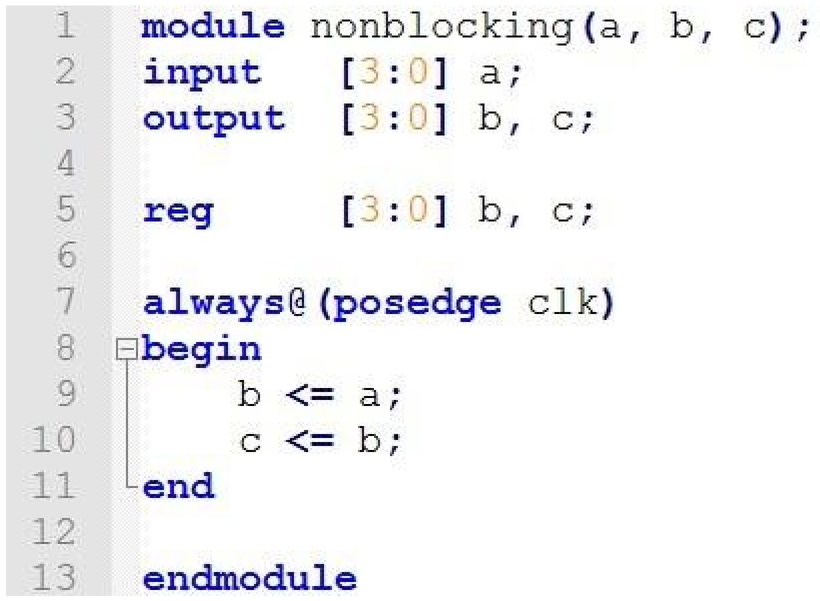
outline

- RTL coding style
  - Blocking and Nonblocking
  - Combinational circuit and Sequential circuit
  - Asynchronous Reset and Synchronous Reset
  - Finite State Machine
  - notes

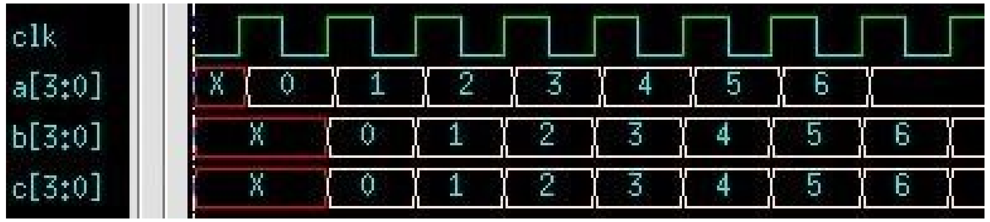
Blocking and Nonblocking

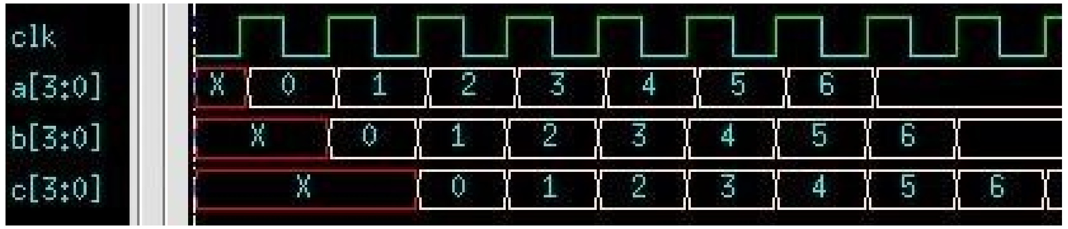


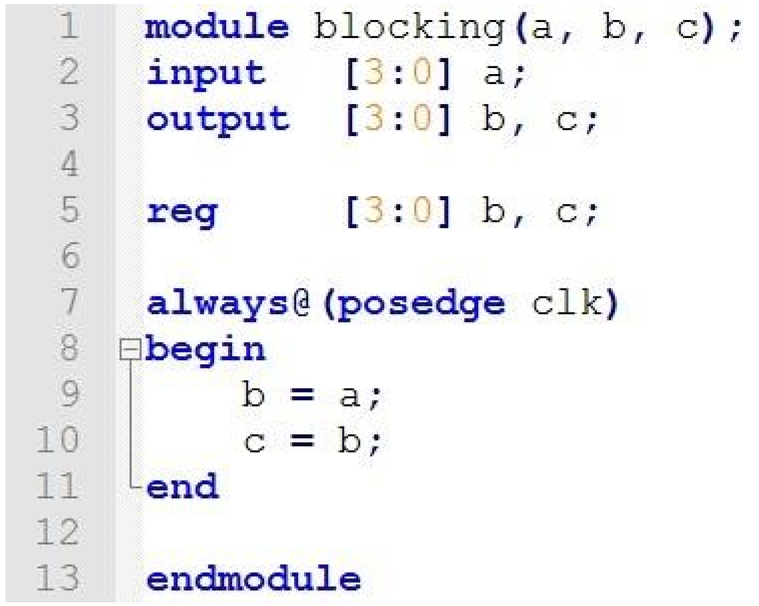
```
module blocking(a, b, c);
input   [3:0] a;
output  [3:0] b, c;

reg     [3:0] b, c;

always@(posedge clk)
begin
    b = a;
    c = b;
end

endmodule
```



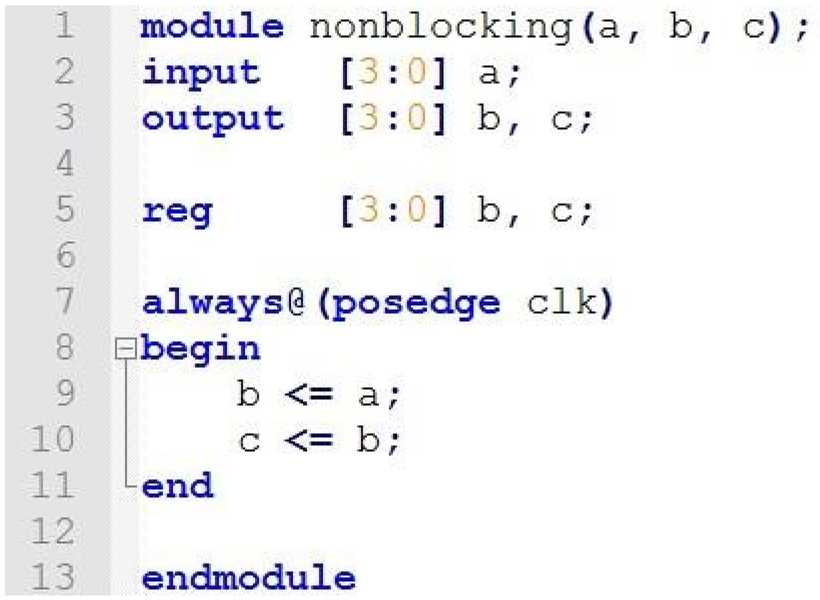
```
module nonblocking(a, b, c);
input   [3:0] a;
output  [3:0] b, c;

reg     [3:0] b, c;

always@(posedge clk)
begin
    b <= a;
    c <= b;
end

endmodule
```



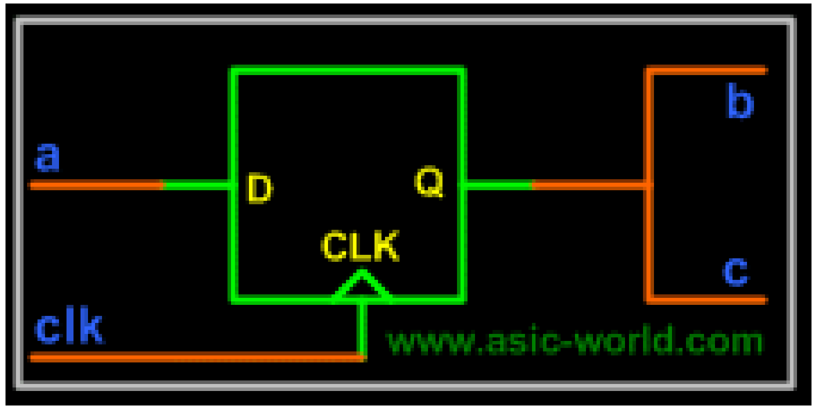


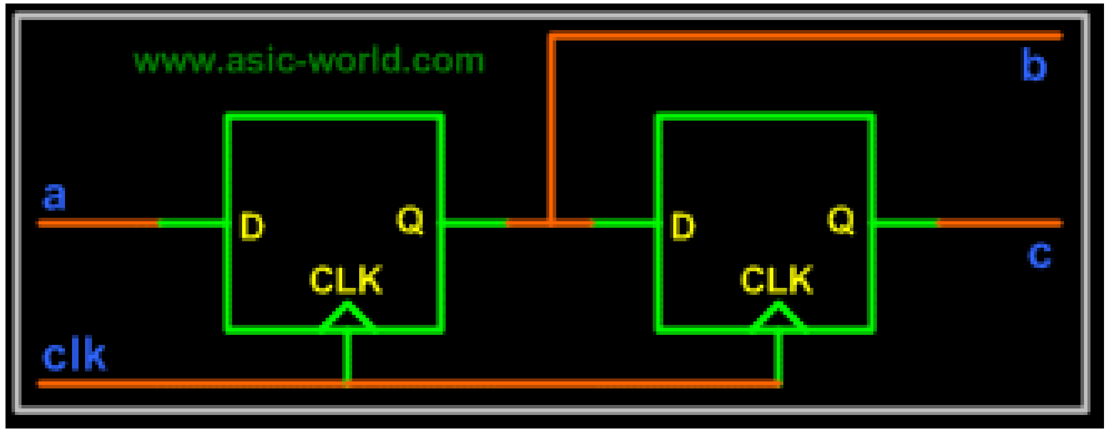


```
module blocking(a, b, c);
input   [3:0] a;
output  [3:0] b, c;

reg     [3:0] b, c;

always@(posedge clk)
begin
    b = a;
    c = b;
end

endmodule
```



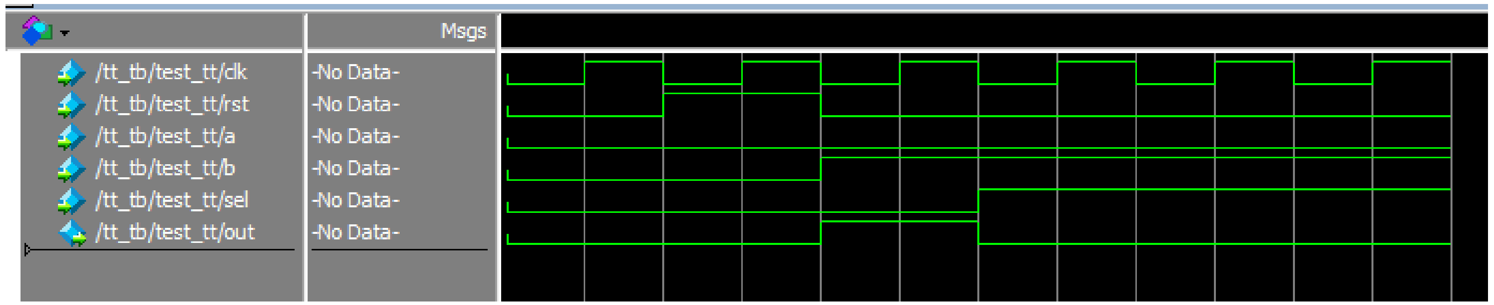
```
module nonblocking(a, b, c);
input   [3:0] a;
output  [3:0] b, c;

reg     [3:0] b, c;

always@(posedge clk)
begin
    b <= a;
    c <= b;
end

endmodule
```





Combinational circuit and Sequential circuit

- Combinational circuit (組合電路)
  - 輸出訊號會在輸入訊號進來的時候，立即反應(假設gate delay為0)
  - 賦值方式請使用 “=“ (blocking)
  - 組合電路可以用assign或是always block去描述，如果是用always block去寫，須宣告成reg，但不為register

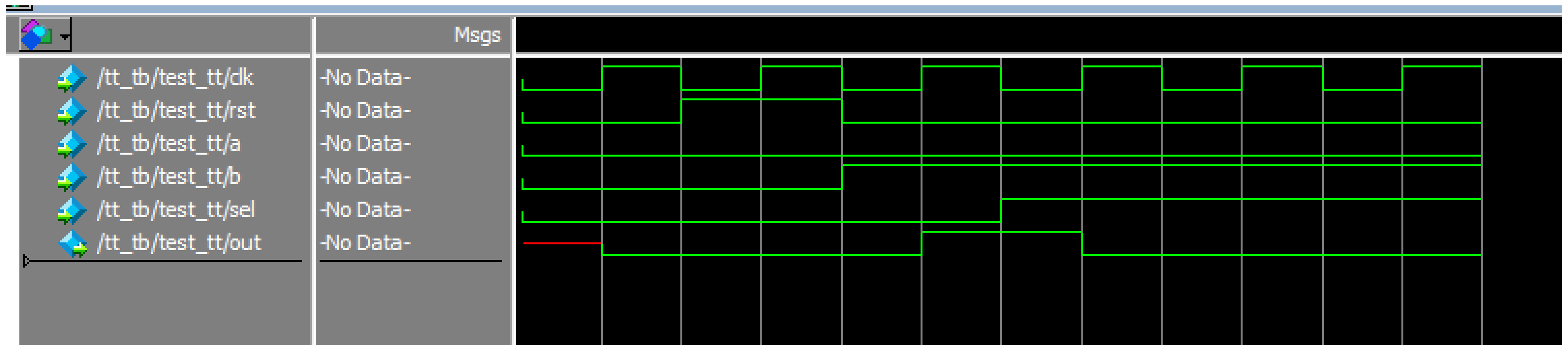


```
reg out;

always@(*)
if(sel)
        out = a;
else
        out = b;
```

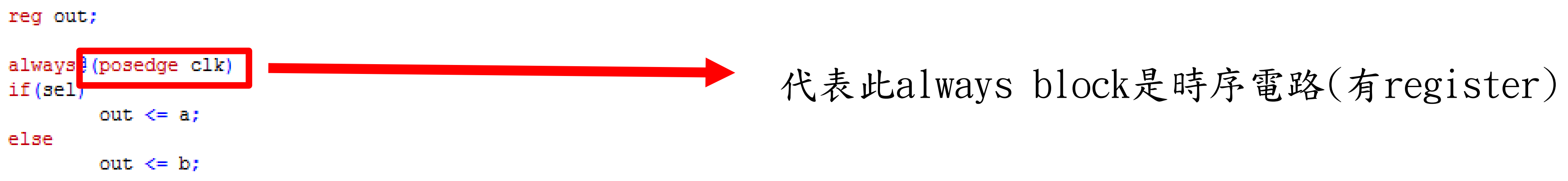
$\xrightarrow{\hspace{3em}}$ 代表此always block是組合電路(無register)

Combinational circuit and Sequential circuit

- Sequential circuit (時序電路)
  - 輸出訊號會依照輸入訊號先前的狀態在clk正緣輸出
  - 賦值方式請使用”<=”(non-blocking)



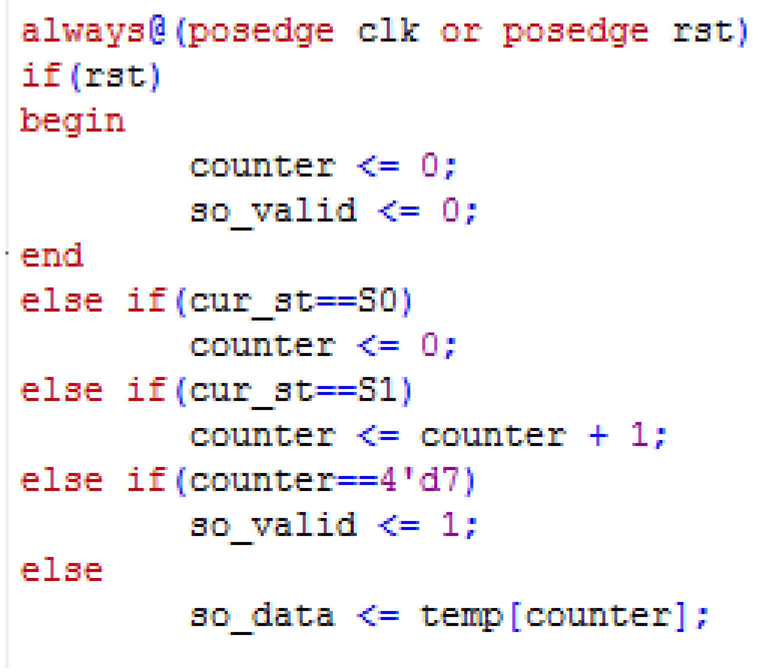
	Msgs
/tt_tb/test_tt/clk	-No Data-
/tt_tb/test_tt/rst	-No Data-
/tt_tb/test_tt/a	-No Data-
/tt_tb/test_tt/b	-No Data-
/tt_tb/test_tt/sel	-No Data-
/tt_tb/test_tt/out	-No Data-



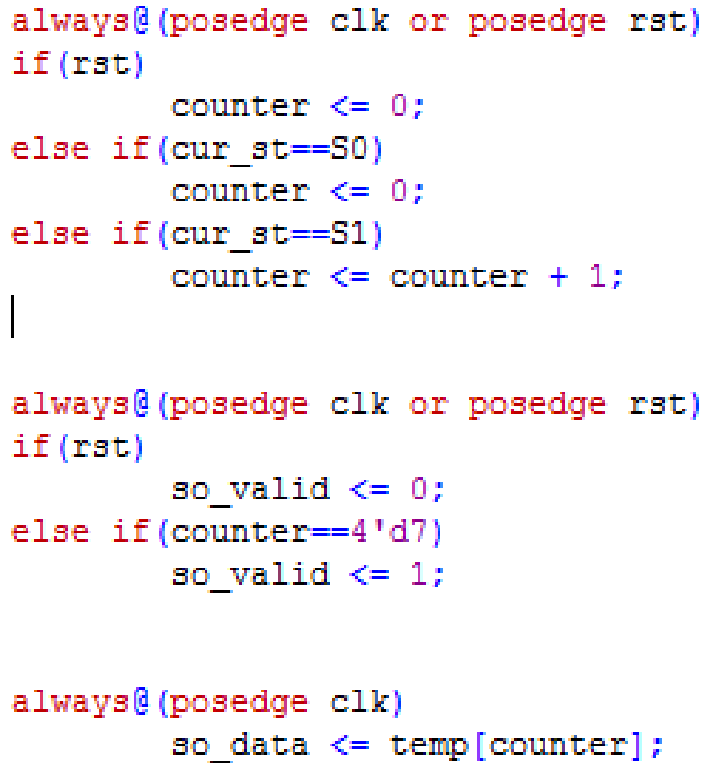
```
reg out;

always@(posedge clk)
if(sel)
        out <= a;
else
        out <= b;
```

一個always block 描述一個訊號

- 把訊號分開寫有助於debug，也較容易維護



```
always@(posedge clk or posedge rst)
if(rst)
begin
        counter <= 0;
        so_valid <= 0;
end
else if(cur_st==S0)
        counter <= 0;
else if(cur_st==S1)
        counter <= counter + 1;
else if(counter==4'd7)
        so_valid <= 1;
else
        so_data <= temp[counter];
```

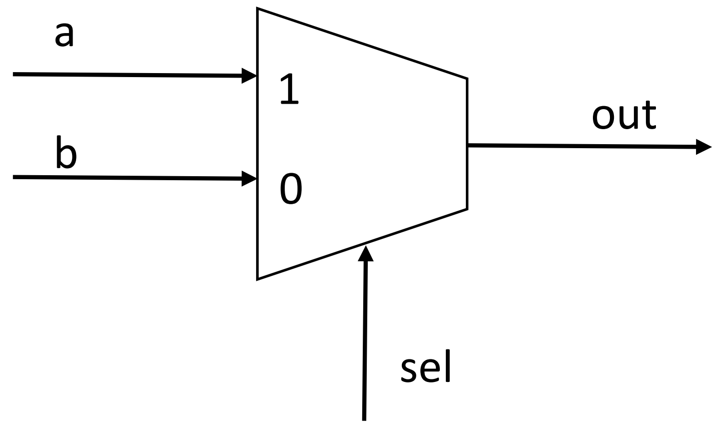


```
always@(posedge clk or posedge rst)
if(rst)
        counter <= 0;
else if(cur_st==S0)
        counter <= 0;
else if(cur_st==S1)
        counter <= counter + 1;
|

always@(posedge clk or posedge rst)
if(rst)
        so_valid <= 0;
else if(counter==4'd7)
        so_valid <= 1;


always@(posedge clk)
        so_data <= temp[counter];
```

了解自己寫的code會生成什麼電路(架構)

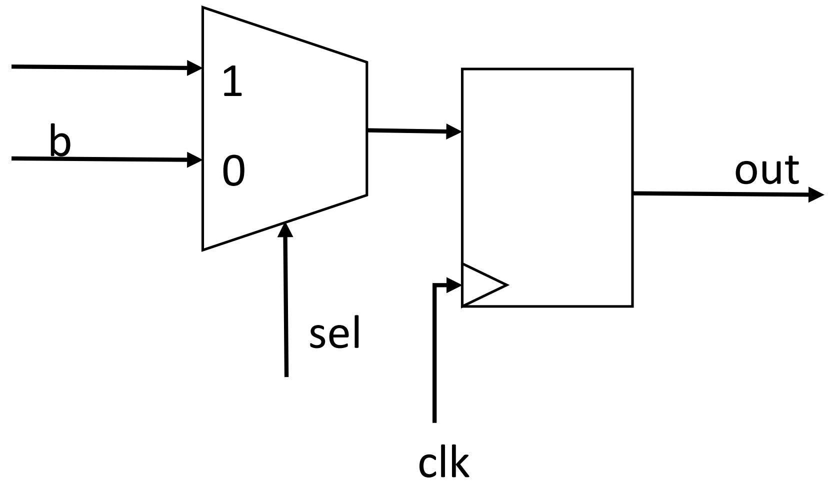
- 組合電路輸出的MUX

```
reg out;

always@(*)
if(sel)
        out = a;
else
        out = b;
```



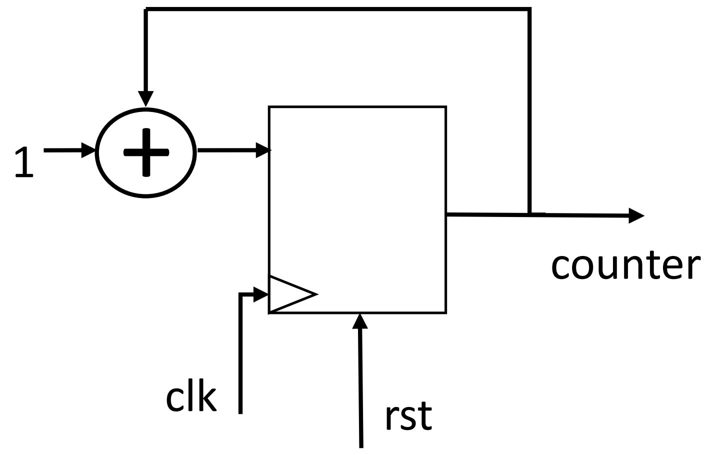
- 時序電路輸出的MUX

```
reg out;

always@(posedge clk)
if(sel)
        out <= a;
else
        out <= b;
```

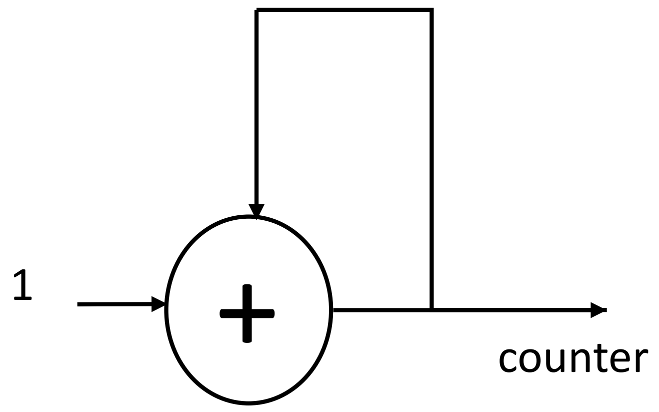


了解自己寫的code會生成什麼電路(架構)

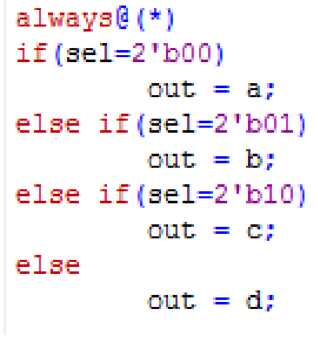
```
always@(posedge clk or posedge rst)
if(rst)
        counter <= 0;
else
        counter <= counter + 1;
```



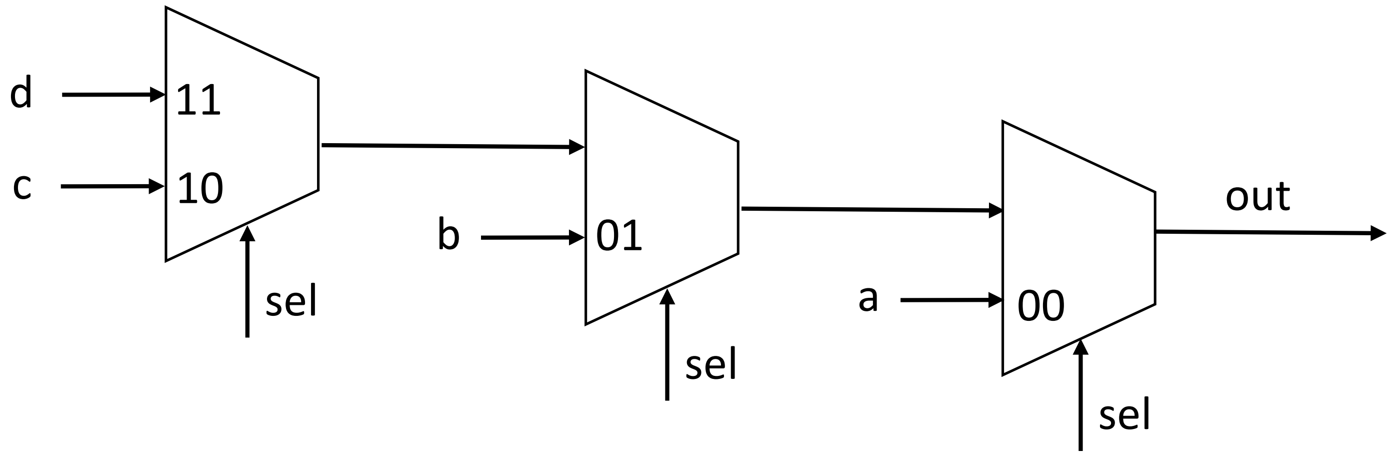
```
always@(*)
if(rst)
        counter = 0;
else
        counter = counter + 1;
```

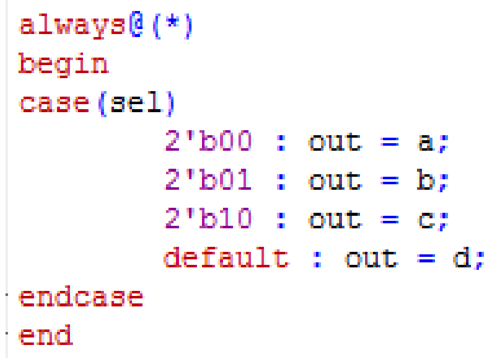


了解自己寫的code會生成什麼電路(架構)

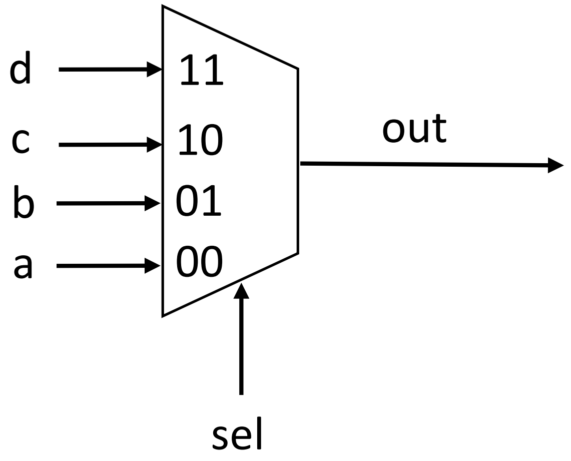


```
always@(*)
if(sel=2'b00)
        out = a;
else if(sel=2'b01)
        out = b;
else if(sel=2'b10)
        out = c;
else
        out = d;
```





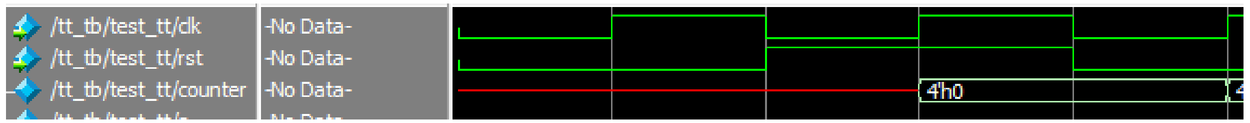
```
always@(*)
begin
case(sel)
        2'b00 : out = a;
        2'b01 : out = b;
        2'b10 : out = c;
        default : out = d;
endcase
end
```



Synchronous Reset vs. Asynchronous Reset

●Synchronous Reset (同步reset)

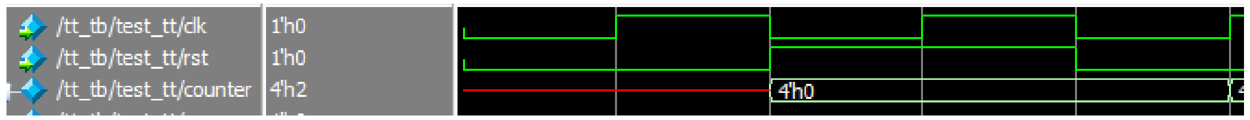
```
always@(posedge clk)
if(rst)
        counter <= 0;
else
        counter <= counter + 1;
```



```
/tt_tb/test_tt/clk      -No Data-
/tt_tb/test_tt/rst      -No Data-
/tt_tb/test_tt/counter  -No Data-    4'h0
```

●Asynchronous Reset (非同步reset)

```
always@(posedge clk or posedge rst)
if(rst)
        counter <= 0;
else
        counter <= counter + 1;
```

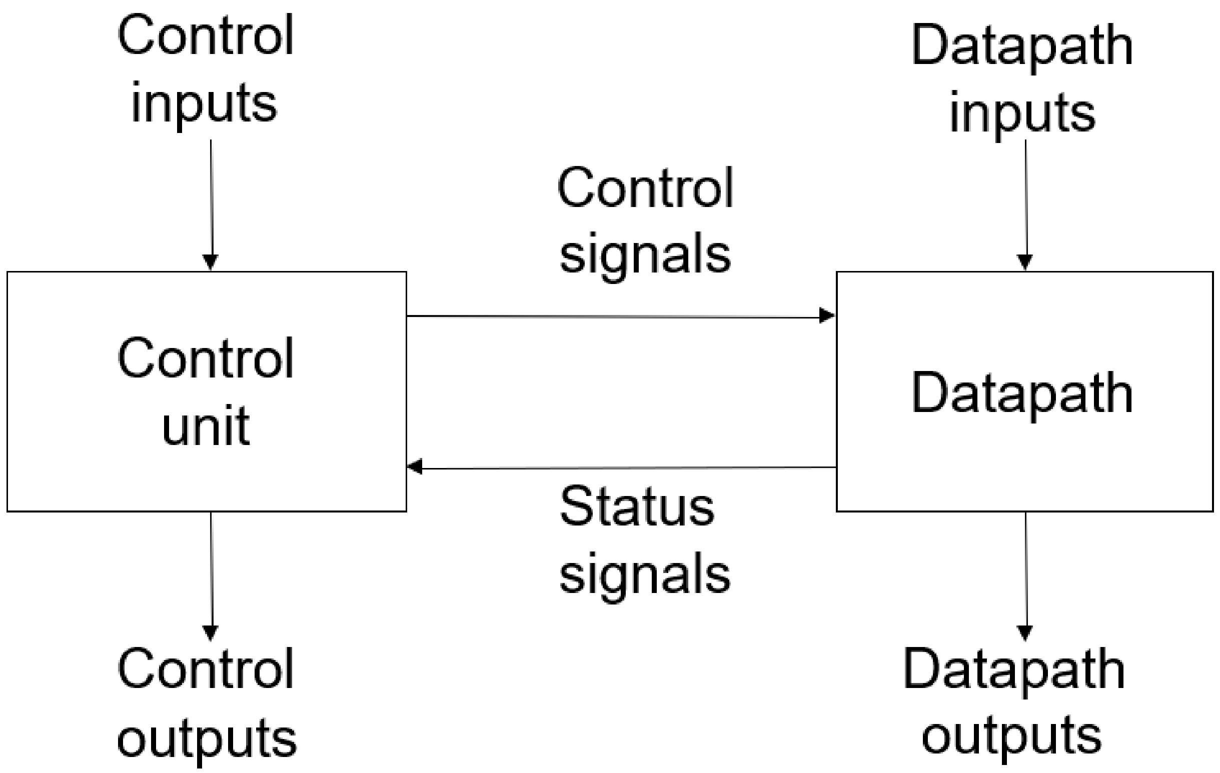


```
/tt_tb/test_tt/clk      1'h0
/tt_tb/test_tt/rst      1'h0
/tt_tb/test_tt/counter  4'h2    4'h0
```

有reset的話盡量用”非同步reset”

控制訊號一定要reset

- 何謂控制訊號？

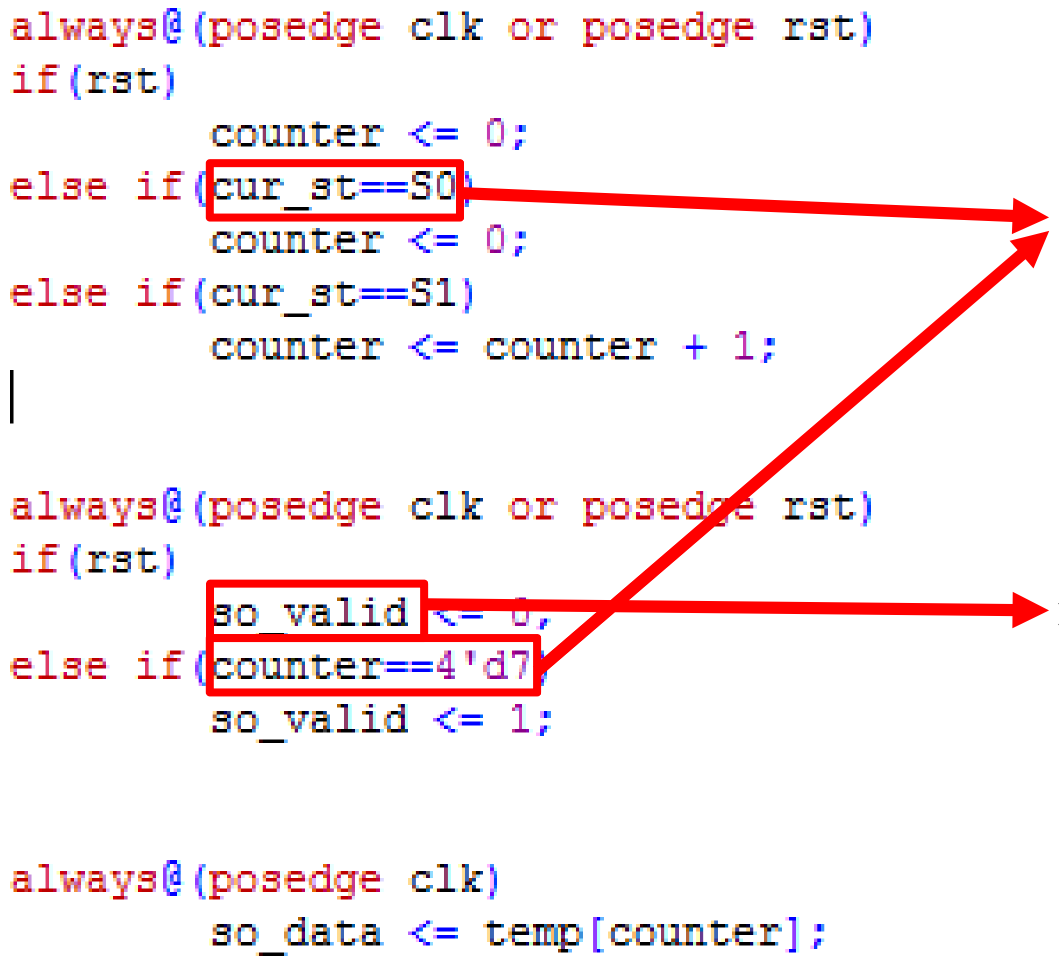


Modern design is composed of

(1) Datapath and

(2) Controller (control unit or control path)

控制訊號一定要reset



```
always@(posedge clk or posedge rst)
if(rst)
        counter <= 0;
else if(cur_st==S0)
        counter <= 0;
else if(cur_st==S1)
        counter <= counter + 1;


always@(posedge clk or posedge rst)
if(rst)
        so_valid <= 0;
else if(counter==4'd7)
        so_valid <= 1;


always@(posedge clk)
        so_data <= temp[counter];
```

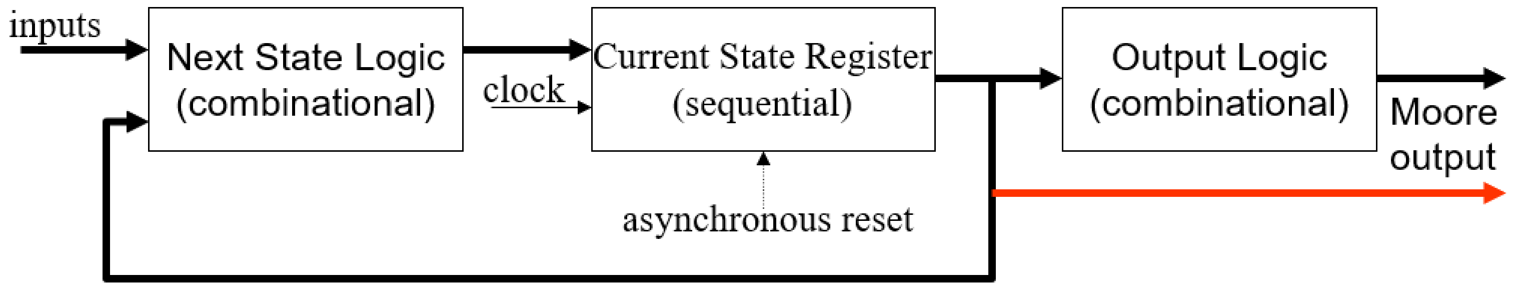
自己design內新增的訊號，如果該訊號有被拿來做的條件判斷，則視為一種控制訊號。e.g. 狀態機、counter…

輸出給testbench的訊號如果有被拿來作條件判斷也要reset，因為testbench會偵測你design裡的so_valid是不是1。如果是1，才會去看那時候你的so_data的值是不是對的。
所以如果一開始so_valid沒有reset，呈現一個unknown的狀態的話，會誤判你當時的so_data。

如果還不清楚那些訊號一定要被reset，那就全部訊號都先reset

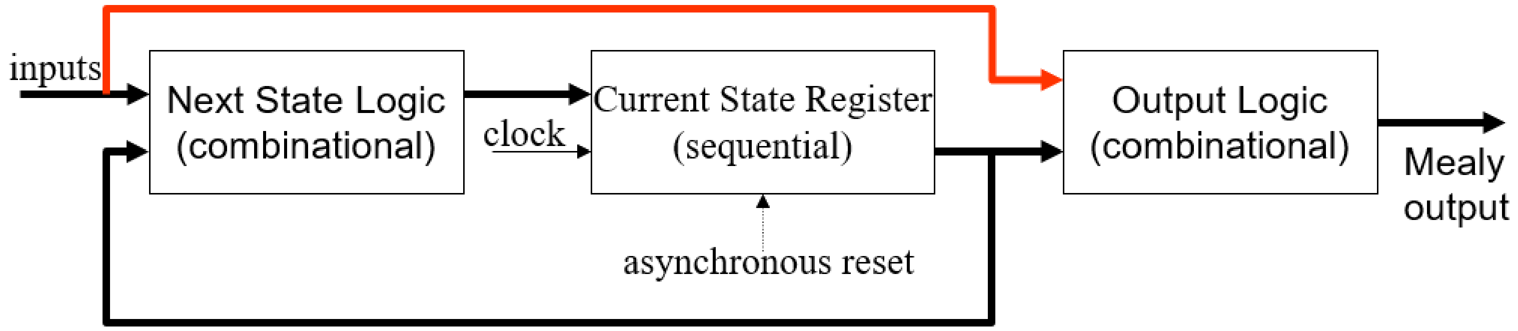
Finite State Machine

$S \rightarrow O$


inputs
Next State Logic (combinational)
clock
Current State Register (sequential)
asynchronous reset
Output Logic (combinational)
Moore output

Moore Machine (state-based machine)

$S \times I \rightarrow O$


inputs
Next State Logic (combinational)
clock
Current State Register (sequential)
asynchronous reset
Output Logic (combinational)
Mealy output

Mealy Machine (input-based machine)

Finite State Machine

```
module FSM(Clock, Reset, Control, Y)
input Clock, Reset, Control;
output [2:0] Y;

reg [1:0] CurrentState, Nextstate;
reg [2:0] Y;

parameter  ST0 = 2'b00,
           ST1 = 2'b01,
           ST2 = 2'b10,
           ST3 = 2'b11;

always @(posedge Clock or posedge Reset)
   if (Reset)
      CurrentState <= ST0;
   else
      CurrentState <= NextState;
```

State name (parameter)

Current State Register (Seq.C.)

Next state logic (Comb.C.)

```
always @(*)
   begin
      NextState = ST0;
      case (CurrentState)
         ST0: NextState = ST1;
         ST1:  if (Control)
                  NextState = ST2;
                  else
                  NextState = ST3;
         ST2: NextState = ST3;
         ST3: NextState = ST0;
      endcase    end
```

Output logic (Comb.C.)

```
always @(*)
   begin
      case(CurrentState)
         ST0: Y = 1;   ST1: Y = 2;
         ST2: Y = 3;   ST3: Y = 4;
      endcase
   end      endmodule
```

notes

- 不使用循環次數不確定的迴圈語句，如while等
- 除非是關鍵電路的設計，一般不使用gate level語句來設計，建議採用behavior level語句來設計
- 盡量一個always block 描述一個訊號，會比較不容易出錯
- 對時序電路的描述，應使用non-blocking (<=)賦值；對組合電路的描述，應使用 blocking (=) 賦值
- 不能在一個以上的always block中對同一個reg賦值，也不能對同一個reg重複使用blocking 與 non-blocking 賦值
- 在組合電路中if-else語句或case語句要把條件寫滿，以免合成出latch
- 避免混合使用 posedge 和 negedge clock觸發