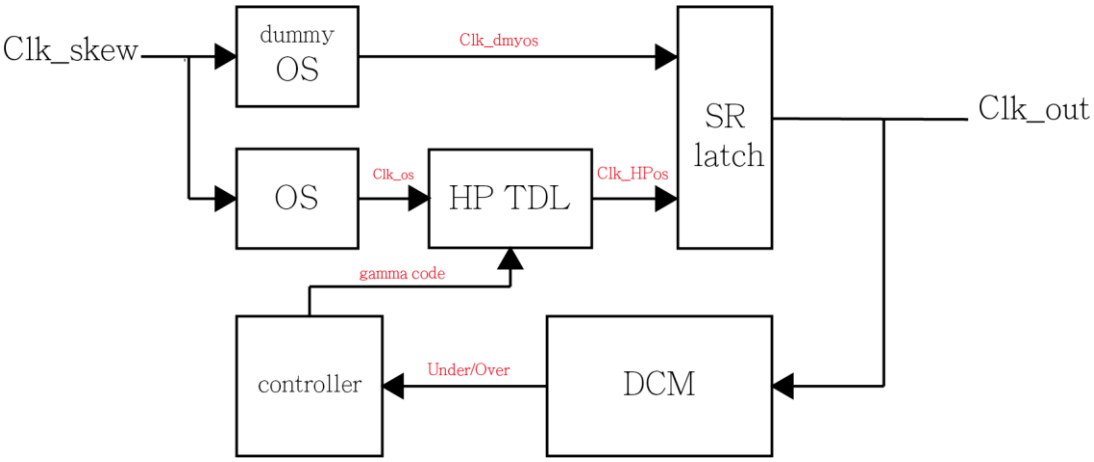
# 時序電路設計及應用 HW2

## Group 11

106061216 電機系 魏胤皓 Ying-Hao, Wei
106010006 電機系 黃詩瑜 Shih-Yu, Huang
106061128 電機系 陳禾育 He-Yu, Chen

(a)




這次我們採用的是微調講義裡的 Closed-Loop Duty-Cycle Corrector 後的架構。首先將輸入的 Clk_skew 分別經過 dummy OS 及 OS 產生 Clk_dmyos 和 Clk_os。接著把 Clk_os 透過 Half-Period Tunable Delay Line 做延遲後得到的 Clk_HPos 作為 reset 輸入 SR latch。並以剛剛的 Clk_dmyos 作為 SR latch 的 set 訊號。Clk_out 輸入 DCM 後，會由 DCM 透過 pulse shrinking 後將正負波長相減的方式，判定現在 Clk_out 的狀態，並將資訊輸出給 controller，由 controller 透過 gamma code 控制 HP TDL 需要產生的 delay 時間。

(Note: Since there are pulse expansion in the TDL and would cause oscillation fail in certain duty cycles, we have to move the OS “before” the TDL)

(b)

dummy OS & OS：

```
module dff_os ( rstn, ck, Y );
  input rstn, ck;
  output Y;
  wire   n2;

  DFFRX1 dff_1 ( .D(1'b1), .CK(ck), .RN(n2), .Q(Y), .QN() );
  NOR2BX1 U2 ( .AN(rstn), .B(Y), .Y(n2) );
endmodule
```

```
dff_os dummy_os(.rstn(rst_n),.ck(clk_in),.Y(clk_os));

dff_os TDL_os(.rstn(rst_n),.ck(clk_in),.Y(clk_in_os));
```

我們以 D-type Flip Flop 搭配 NOR2，以 trigger DFF 並在短暫 delay 後 reset 的方式實現 one shot 的功能。

HP-TDL：

```
module delay_chain#(
    parameter counts = 3
)(
    input  wire A,
    output wire Y
);
wire [counts:0] nodes;
wire [counts:0] nodes_b;

assign nodes[0] = A;
assign Y = nodes[counts];

genvar i;
generate
    for(i=0;i<counts;i=i+1)begin:d_block
        CLKINVX1 a1(.A(nodes[i]) ,.Y(nodes_b[i]));
        CLKINVX1 a2(.A(nodes_b[i]) ,.Y(nodes[i+1]));
    end
endgenerate


endmodule
```

```
genvar i;
generate
    for(i=1;i<delay_nodes;i=i+1)begin: delay
        delay_chain#(.counts(1)) chain(.A(clk_temp[i-1]),.Y(clk_temp[i]));
    end
endgenerate


integer j;
always@(*) begin
    clk_out <=  0;
    for(j = 0; j < delay_nodes; j=j+1) begin
        if (gamma == j)
            clk_out <= clk_temp[j];
   end
end
```

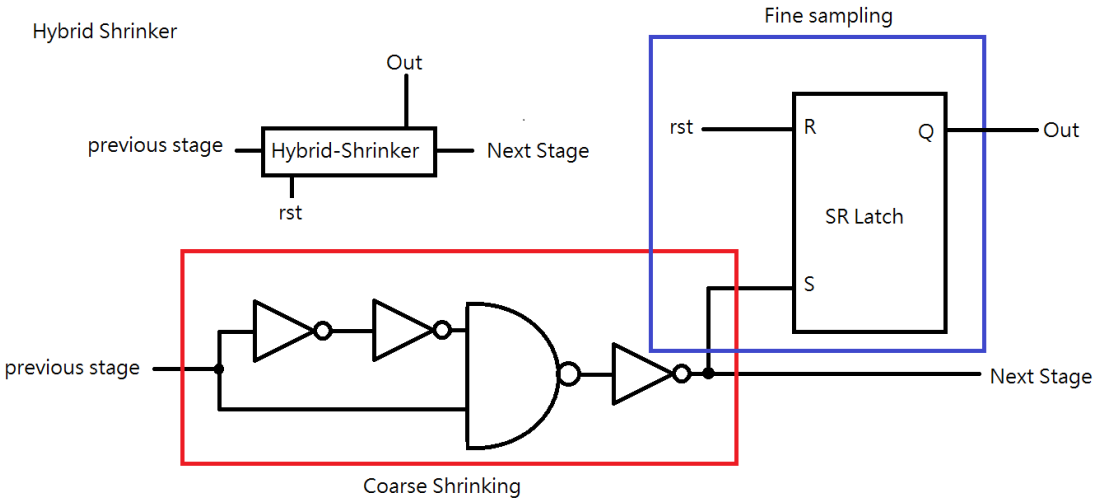
TDL 的部分我們使用串聯兩個 inverter 的方式構成一個 delay cell，如此一來便可達到使 pull up 及 pull down 時間一樣的效果。再以 gamma code 決定 clock out 所接的 delay cell 數量。

DCM：

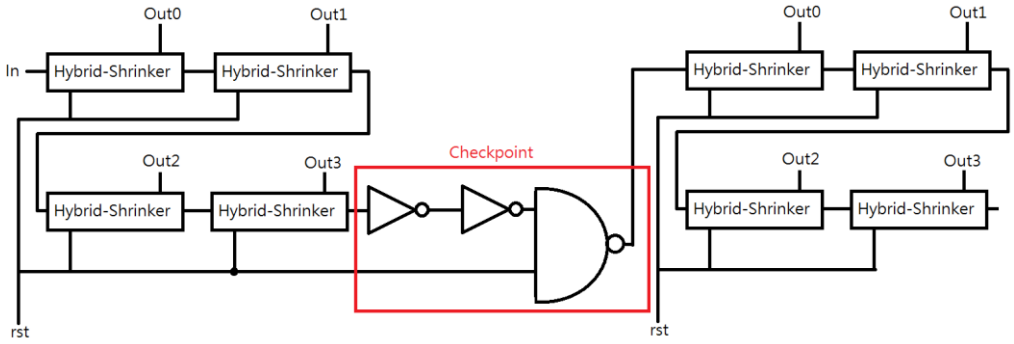
```
genvar i;
generate
    for(i=0;i<level;i=i+1)begin:shrink_cell
        if(i % 4 == 0)begin : rst_post
            NOR2BX2 mux0(.AN(shrinker_node[i]) ,.B(rst),.Y(shrinker_node_masked[i]));
            CLKINVX1 buf1(.A(shrinker_node_masked[i]),.Y(shrinker_node_b[i]));
            CLKINVX1 buf2(.A(shrinker_node_b[i]),.Y(shrinker_node[i+1]));
        end
        else begin : normal
            CLKINVX1 buf1(.A(shrinker_node[i]),.Y(shrinker_node_b[i]));
            CLKINVX1 buf2(.A(shrinker_node_b[i]),.Y(subnode[i]));
            NAND2X2 nand1(.A(shrinker_node[i]),.B(subnode[i]),.Y(subnode_2[i]));
            CLKINVX2 inv1(.A(subnode_2[i]),.Y(shrinker_node[i+1]));

        end


        SR_latch sr1(.S(shrinker_node[i+1]),.R(rst),.Q(thermo_out[i]),.QN());
    end
endgenerate
```

DCM 裡我們是採用串聯上圖 shrink cell 去做 coarse shrinking 的方式得出 theta 值。當 controller 輸出 request=1 時，DCM 就會開始抓取 theta 值，得到值之後，DCM 會輸出 ready 訊號，使 controller 可以讀取 theta 值，並將 finish 設為 1 表示 DCM 可進行下一輪的運算。

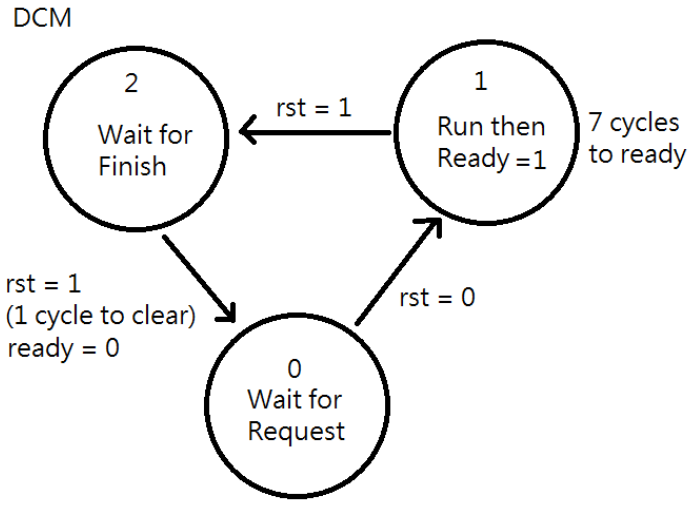
## The Hybrid shrinking cell




Checkpoint: In order delayed signal cause reset failure, we insert checkpoints every 4 shrinker. Hence, the reset pulse would kill all shrinker signal in at least 4 delays.




Interface: To communicate with Controller, a FSM is deployed to hand-shake with.

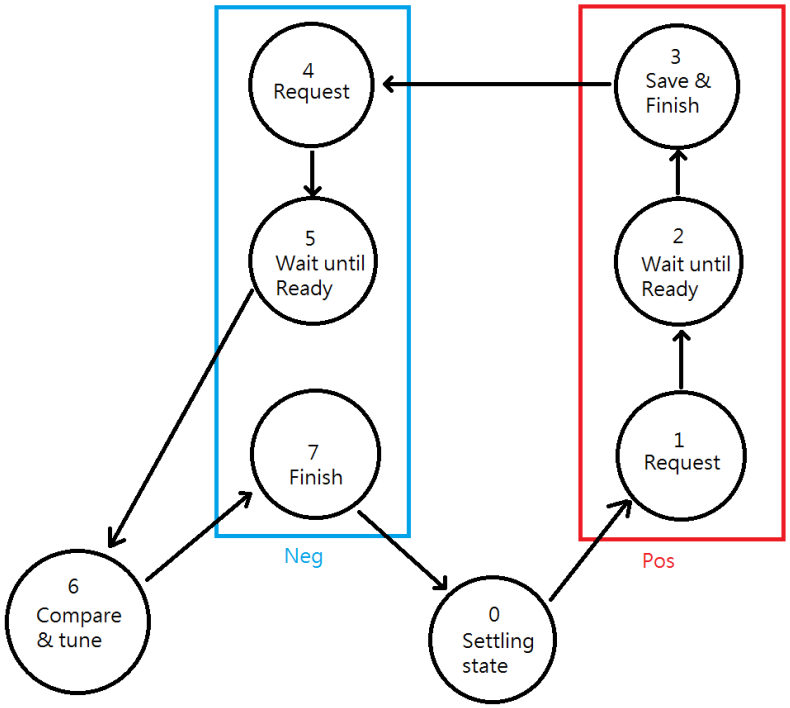


Controller：

```
always@(posedge clk_in or negedge rstn)begin
    if(rstn == 0)begin
        state <= 0;
        counter <= 0;
        gamma <= 0;
        request <= 0;
        finish <= 0;
        pos_neg <= 0;
        locked_flag <= 0;
    end
    else begin
        if(state == 0)begin
            if(counter == 10)begin
                state <= 1;
                counter <= 0;
                request <= 1;
                pos_neg <= 0;
            end
            else begin
                counter <= counter + 1;
            end
        end
        else if(state == 1)begin
            if(counter == 3)begin
                state <= 2;
                counter <= 0;
                request <= 0;
            end
            else begin
                counter <= counter + 1;
            end
        end
        else if(state == 2)begin
            if(ready == 1)begin
                state <= 3;
            end
        end
        else if(state == 3)begin
            if(counter == 1)begin
                pos_theta <= theta;
                counter <= counter + 1;
            end
            else if(counter == 3)begin
                counter <= counter + 1;
                finish <= 1;
            end
            else if(counter == 7)begin
                state <= 4;
                counter <= 0;
                finish <= 0;
            end
            else begin
                counter <= counter + 1;
            end
        end
        else if(state == 4)begin
            if(counter == 5)begin
                state <= 5;
                counter <= 0;
                request <= 0;
            end
            else begin
                counter <= counter + 1;
                request <= 1;
                pos_neg <= 1;
            end

        end
        else if(state == 5)begin
            if(ready == 1)begin
                state <= 6;
            end
        end
        else if(state == 6)begin
            if(theta == pos_theta)begin
                gamma_decision <= 2;
            end
            else begin
                gamma_decision <= |(theta && (~pos_theta));
            end
            state <= 7;
        end
        else begin
            if(counter == 3)begin
                counter <= counter + 1;
                finish <= 1;
            end
            else if(counter == 7)begin
                state <= 0;
                counter <= 0;
                finish <= 0;
                if(gamma_decision == 1'b1)begin
                    if(gamma < 31)begin
                        gamma <= gamma + 1;
                        locked_flag <= 0;
                    end
                end
                else if(gamma_decision == 1'b0)begin
                    if(gamma > 0)begin
                        gamma <= gamma - 1;
                        locked_flag <= 0;
                    end
                end
                else begin
                    locked_flag <= 1;
                end
            end
            else begin
                counter <= counter + 1;
            end

        end
    end
end
```

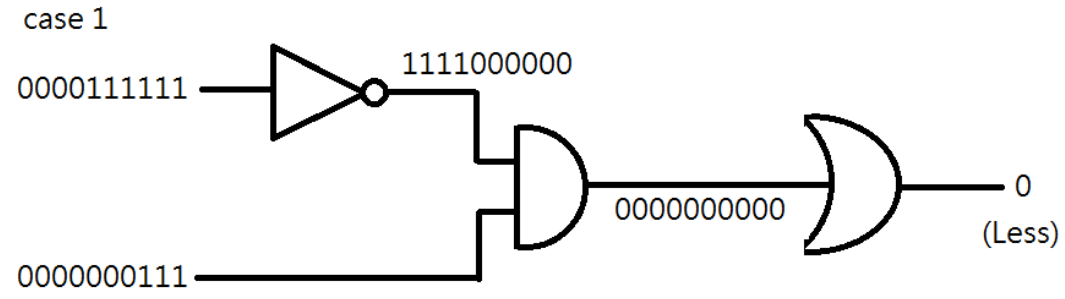
在 Controller 裡透過 finite state machine 控制 DCM 開始運算以及接收 DCM 運算結束的訊號並讀取 theta 值。再透過 theta 值以及正負號(pos_theta)去控制 gamma code。直到 clk_out duty cycle 接近 50%時，使 locked_flag=1。

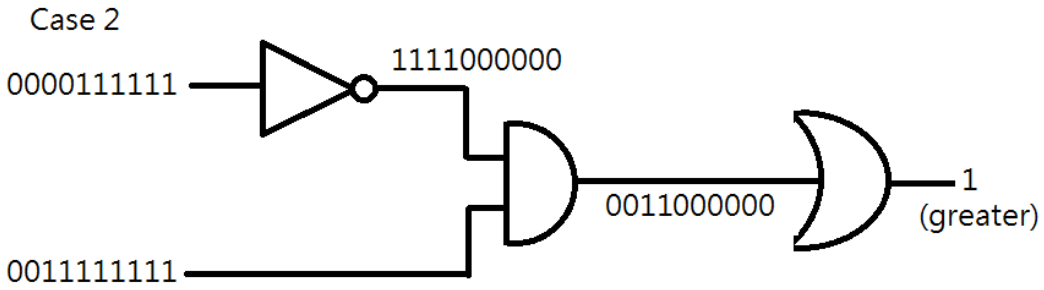
The Controller is designed in a hand-shaking fashion, with the states illustrated below.




The rather bizarre order would save the half of the flip-flops needed.

Thermal code compare scheme:



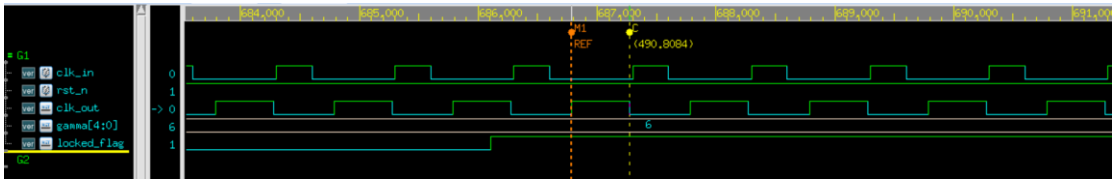

(c) All design synthesized successfully without error.

```
#/*-------------------------------------------------------------------*/
#/*----------------------- 5.Write out files ---------------------*/
#/*-------------------------------------------------------------------*/
report_area > ./top_syn.report
report_timing -path full -delay max >> ./top_syn.report
report_power >> ./top_syn.report
write -format verilog -hierarchy -output ./top_syn.v
Writing verilog file '/home/u106/u106061128/109_2/tcd/old/HW3/syn_top/top_syn.v
'.
Warning: Changed wire name subnode_2 to subnode_2_snps_int_bus in module hybrid
_shrinker_level20_1.  Please use the change_names command to make the correct c
hanges before invoking the verilog writer. (VO-2)
1
write_sdf -version 1.0 -context verilog -load_delay cell ./top_syn.sdf
Information: Writing timing information to file '/home/u106/u106061128/109_2/tc
d/old/HW3/syn_top/top_syn.sdf'. (WT-3)
1
write_sdc ./top_syn.sdc
1
exit

Thank you...
```

(d)

gate-level:

input clk_in duty cycle: 30%

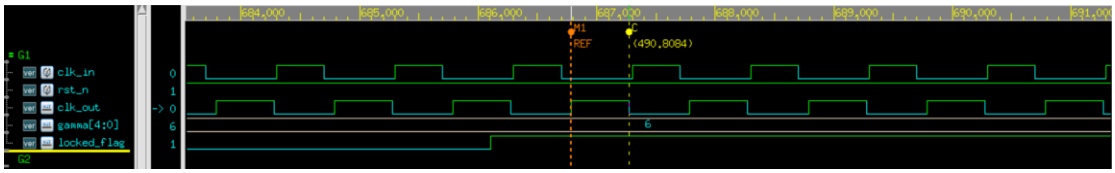



```
*Verdi* : Create FSDB file './top.fsdb'
*Verdi* : Begin traversing the scopes, layer (0).
*Verdi* : End of traversing.
pos_time:          488, neg_time:          512
duty_cycle = 0.488000
Simulation complete via $finish(1) at time 1 US + 0
./testbench.v:60        $finish;
```

High phase interval: 512ps  Low phase interval: 488ps

output clk_dcc duty cycle: 48.8%

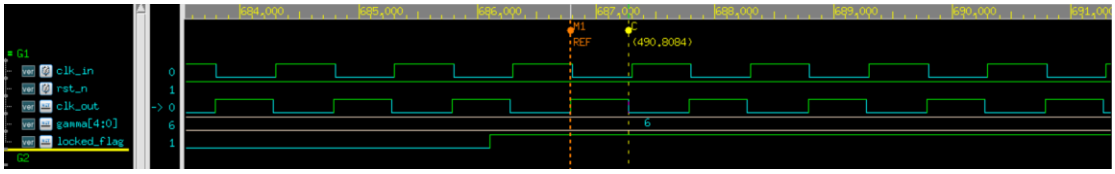
clk_in: 40%




High phase interval: 512ps  Low phase interval: 488ps

output clk_dcc duty cycle: 48.8%

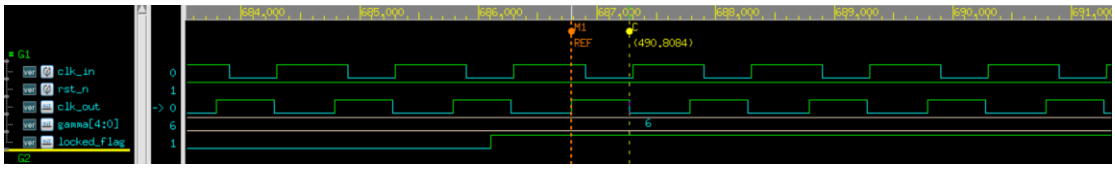
clk_in: 50%




High phase interval: 512ps  Low phase interval: 488ps

output clk_dcc duty cycle: 48.8%

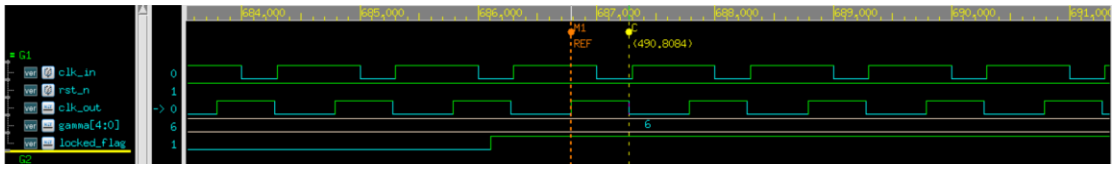
clk_in: 60%




High phase interval: 512ps  Low phase interval: 488ps

output clk_dcc duty cycle: 48.8%

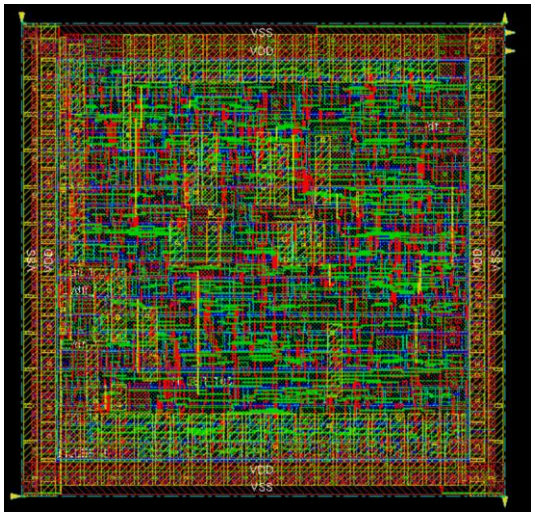
clk_in: 70%

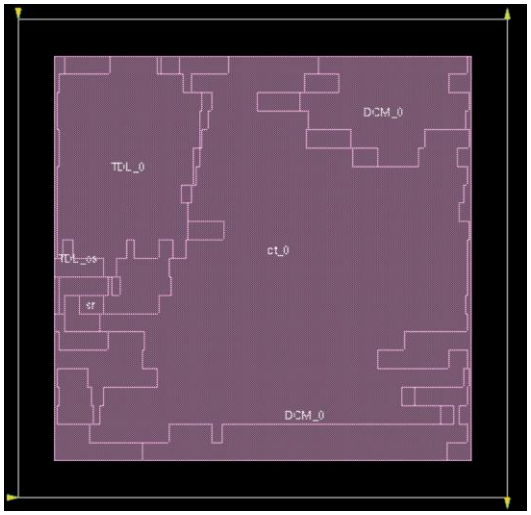


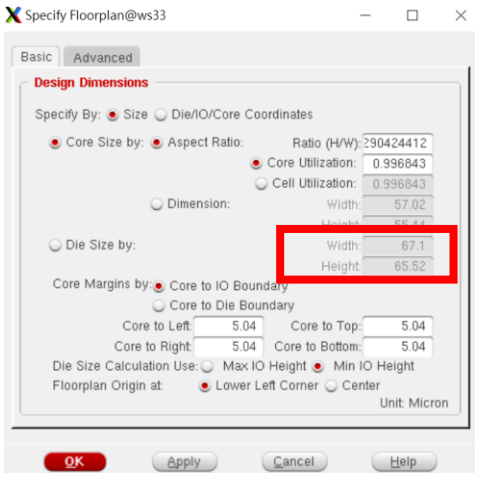

High phase interval: 512ps  Low phase interval: 488ps

output clk_dcc duty cycle: 48.8%

(e)



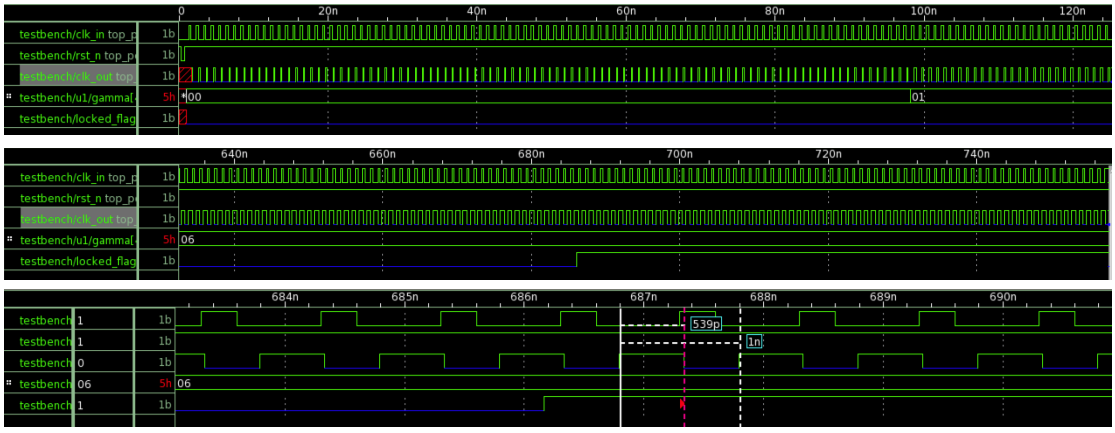




Layout Area(Width*Height): 67.1*65.52um$^2$

(f)

post-layout:

input clk_in duty cycle: 30%

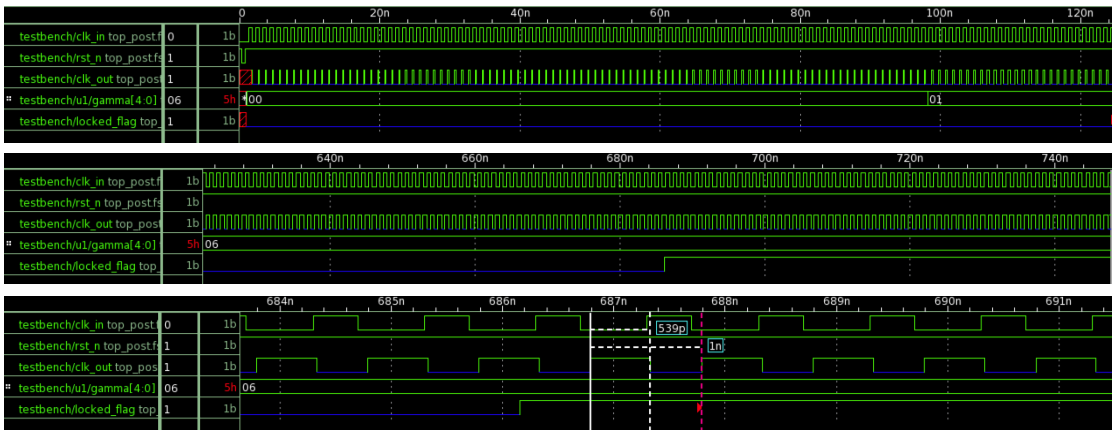



```
*Novas* : Create FSDB file './top_post.fsdb'
*Novas* : Begin traversing the scopes, layer (0).
*Novas* : End of traversing.
         539          461
duty_cycle = 0.539000
Simulation complete via $finish(1) at time 1 US + 0
./testbench.v:60        $finish;
ncsim> exit
```

High phase interval: 539ps  Low phase interval: 461ps

output clk_dcc duty cycle: 53.9%

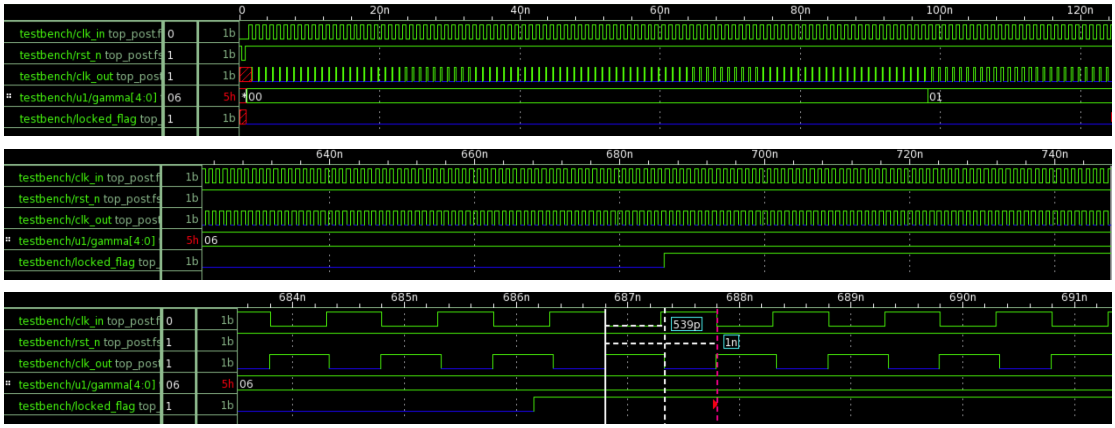
clk_in: 40%




High phase interval: 539ps  Low phase interval: 461ps

output clk_dcc duty cycle: 53.9%

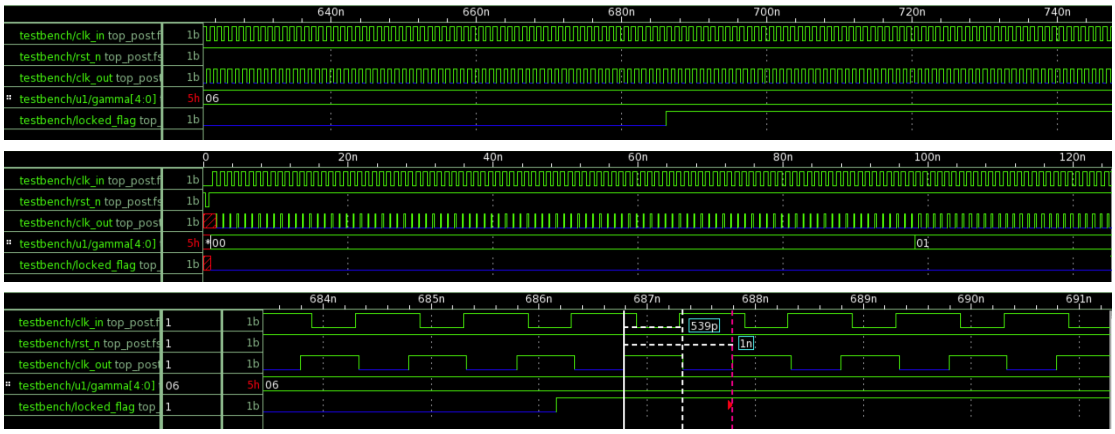
clk_in: 50%




High phase interval: 539ps  Low phase interval: 461ps

output clk_dcc duty cycle: 53.9%

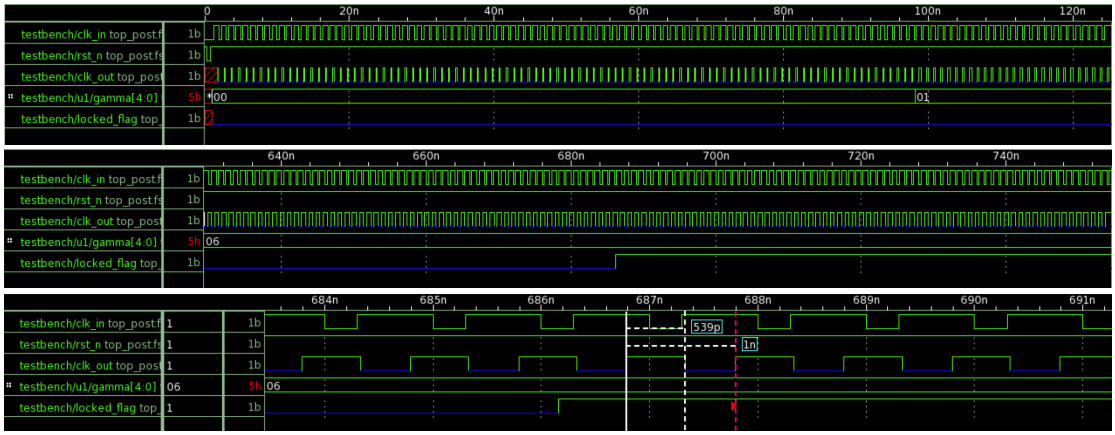
clk_in: 60%




High phase interval: 539ps  Low phase interval: 461ps

output clk_dcc duty cycle: 53.9%

clk_in: 70%




High phase interval: 539ps  Low phase interval: 461ps

output clk_dcc duty cycle: 53.9%

locked 後，pre-layout 的 output clk_dcc duty cycle 為 48.8%，而 post-layout 為 53.9%，造成 post-layout high phase interval 較長的原因，推測是因為 post-layout 加入了 RC delay 後造成 delay 較長，讓控制 clk_dcc 的電路比較晚將之從 1 拉到 0。

組員分工:

魏胤皓 : Verilog coding/simulation

黃詩瑜 : 報告撰寫

陳禾育 : APR