Timing Circuit Designs and Their Applications

Homework 3
Duty-Cycle Correction Circuit Design

姓名：宋易柔（Yi-Rou, Song)

學號：114061529

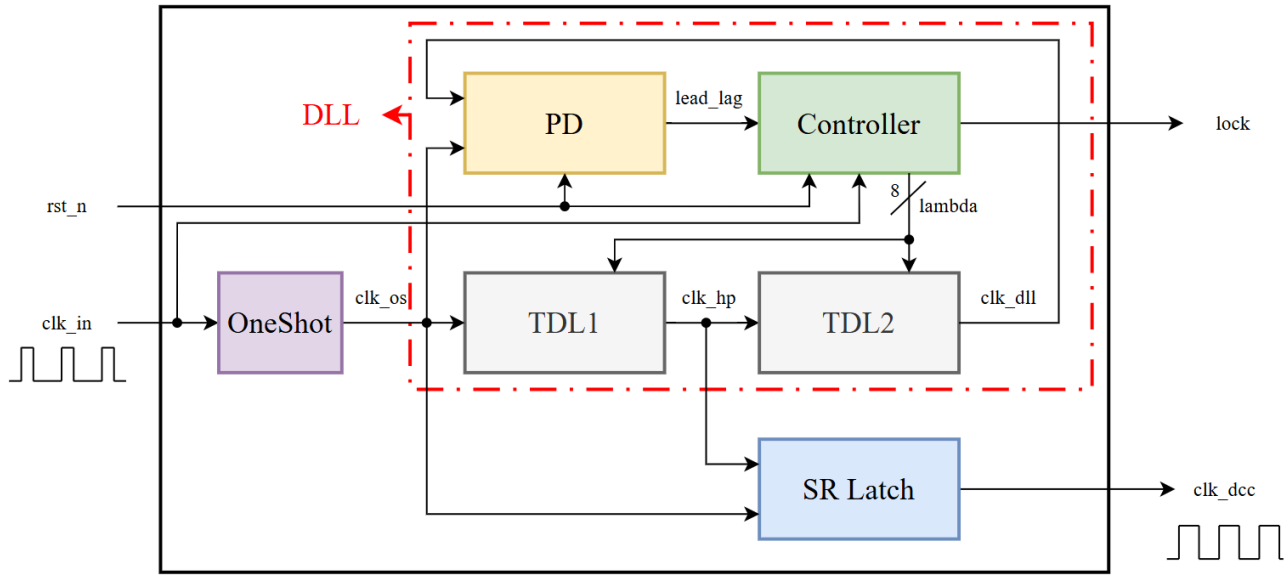
系所：電機系 (EE)

1. 題目說明

設計 Duty-Cycle Correction Circuit (以下簡稱 DCC)，使輸入 clk_in 1 GHz 時脈訊號調整為 duty cycle 接近 50% 的輸出時脈訊號 clk_dcc。

2. Block Diagram

DCC Block diagram:



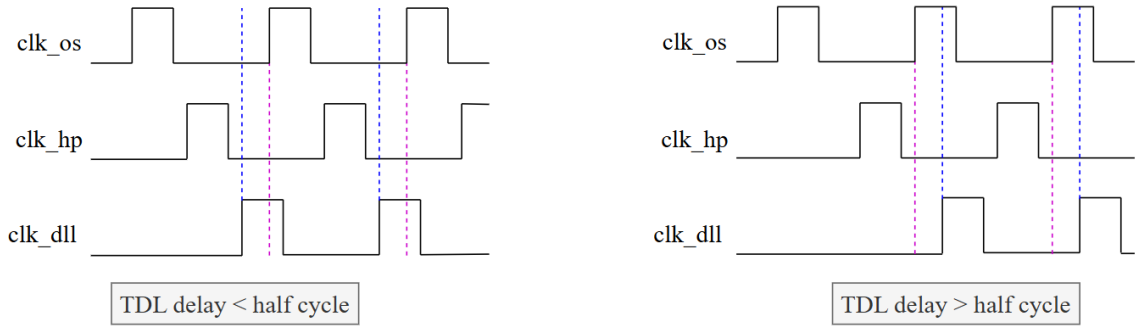
DCC 的架構由一個 One Shot、一個 DLL 和一個 High-Active SR-Latch 組成。而 DLL 則由 Phase Detector、Controller 以及 TDL 組成。

我將 TDL 複製兩份 TDL1、TDL2，每份約可延遲半個週期，因此 TDL1 的輸出 clk_hp 延遲 clk_os 半個週期，TDL2 的輸出 clk_dll 延遲 clk_os 一個週期，將 clk_dll 傳入 PD 與 clk_os 比較領先/落後，Controller 決定 lambda code 並傳入 TDL1 和 TDL2，並輸出 lock 訊號，SR Latch 藉由 clk_os 與 clk_hp 的時間差，輸出 clk_dcc。

PD:

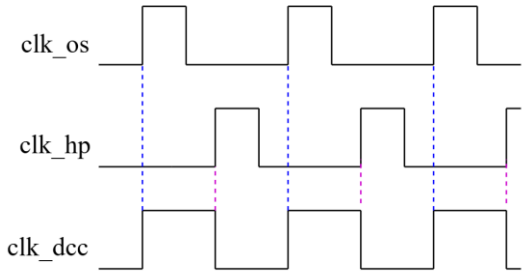
如果 TDL 的 Delay 小於半個週期，則 clk_dll 就會領先 clk_os，PD 會判斷 lead_lag 為 1；

如果 TDL 的 Delay 大於半個週期，則 clk_dll 就會落後 clk_in，PD 會判斷 lead_lag 為 0。



SR Latch:

clk_os 升起時，clk_dcc 就會升起；clk_hp 升起時，clk_dcc 就會落下。



3. Design

DCC:

Signal Name	I/O	Width	Simple Description
clk_in	I	1	1 GHz 時脈訊號。
rst_n	I	1	低準位非同步(active low asynchronous)之系統重置信號。
clk_dcc	O	1	DCC 輸出時脈訊號。
lambda	O	8	控制 TDL 的λ-code。
lock	O	1	lock 為 high 時表示 duty cycle 接近 50%。

DCC 裡有六個 sub-module，分別是 OneShot、PD、Controller、TDL1、TDL2 與 SRLatch。

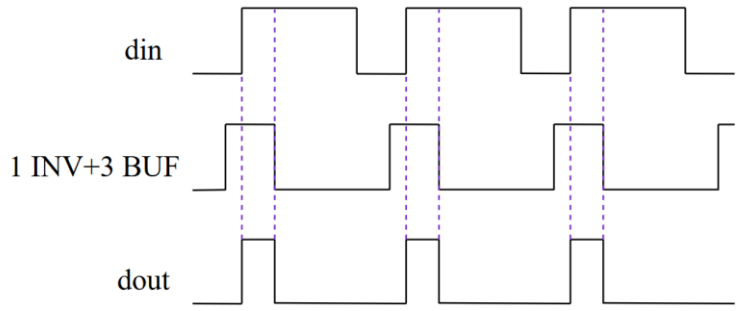
```
`include "../2.Gate_level_simulation/OneShot_syn.v"
`include "../2.Gate_level_simulation/PD_syn.v"
`include "Controller.v"
`include "../2.Gate_level_simulation/TDL_syn.v"
`include "../2.Gate_level_simulation/SRLatch_syn.v"

module DCC (
    input          clk_in,
    input          rst_n,
    output         clk_dcc,
    output [7:0]   lambda,
    output         lock
);
    wire clk_os;
    wire lead_lag;
    wire clk_hp;
    wire clk_dll;

    OneShot OneShot (
        .din        (clk_in),
        .dout       (clk_os)
    );

    PD PD (
        .clk_in     (clk_os),
        .FB         (clk_dll),
        .enable     (rst_n),
        .lead_lag   (lead_lag)
    );
```

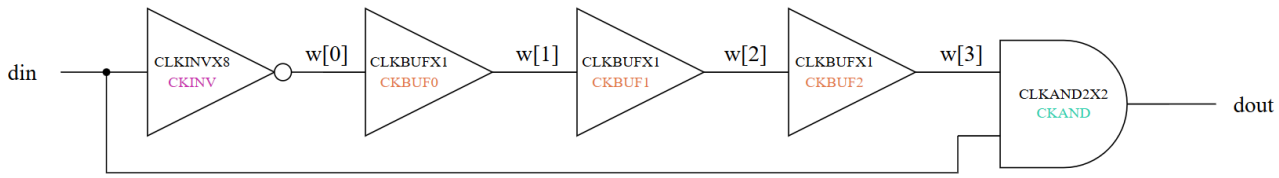
```
    Controller Controller (
        .lead_lag   (lead_lag),
        .clk        (clk_in),
        .rst_n      (rst_n),
        .lambda     (lambda),
        .lock       (lock)
    );

    TDL TDL1 (
        .clk_in     (clk_os),
        .lambda     (lambda),
        .clk_out    (clk_hp)
    );

    TDL TDL2 (
        .clk_in     (clk_hp),
        .lambda     (lambda),
        .clk_out    (clk_dll)
    );

    SRLatch SRLatch(
        .S          (clk_os),
        .R          (clk_hp),
        .Q          (clk_dcc)
    );

endmodule
```

OneShot:

Signal Name	I/O	Width	Simple Description
din	I	1	輸入訊號。
dout	O	1	輸入的 One shot 訊號。

此模組的功能為防止送入 SR-Latch 的兩個輸入 high phase 重疊。利用一個 CLKINV 和三個 CLKBUFX1，造成反向及延遲，再與 din 做 AND operation，使不論 din 的 duty cycle 為何，輸出 One shot 訊號。



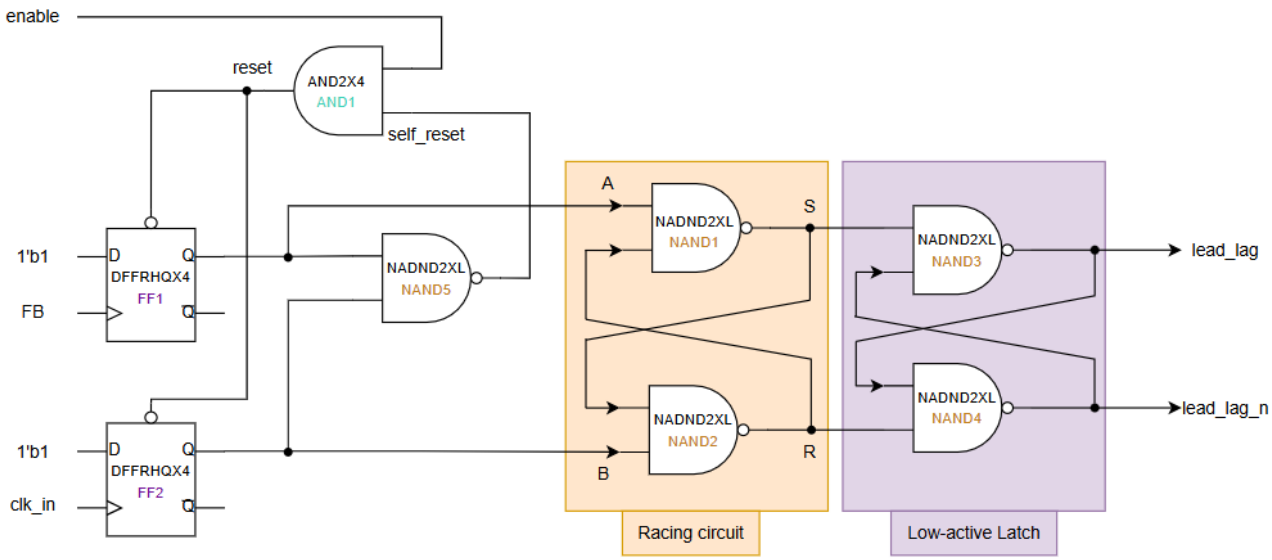



```
module OneShot(
    input   din,
    output  dout
);
    parameter buffer_chain = 3;
    wire  [buffer_chain:0] w;

    CLKINVX8  CKINV  (.A(din),       .Y(w[0]));
    CLKBUFX1  CKBUF0 (.A(w[0]),      .Y(w[1]));
    CLKBUFX1  CKBUF1 (.A(w[1]),      .Y(w[2]));
    CLKBUFX1  CKBUF2 (.A(w[2]),      .Y(w[3]));
    CLKAND2X2 CKAND  (.A(din),       .B(w[3]),      .Y(dout));
endmodule
```

Phase Detector:

Signal Name	I/O	Width	Simple Description
clk_in	I	1	1 GHz 時脈訊號。
FB	1	1	TDL2 輸出迴授時脈訊號。
enable	I	1	enable 拉起時表示 PD 開始運作。
lead_lag	O	1	lead_lag 為 1 時表示 clk_dll 領先 clk_os，為 0 則落後。

PD 的架構有一組 Racing circuit、一組 Low-active Latch、兩個 DFFRHQX4，和控制 reset 的 NAND2XL 和 AND2X4。




```
module PD (
    input  clk_in,
    input  FB,
    input  enable,
    output lead_lag
);
    wire   dummy, lead_lag_n;
    wire   A, B, S, R;
    wire   reset, self_reset;

    // DFF
    DFFRHQX4 FF1  (.D(1'b1),       .CK(FB),          .Q(A),       .RN(reset));
    DFFRHQX4 FF2  (.D(1'b1),       .CK(clk_in),      .Q(B),       .RN(reset));

    // Racing Circuit
    NAND2XL NAND1 (.A(A),          .B(R),            .Y(S)                  );
    NAND2XL NAND2 (.A(S),          .B(B),            .Y(R)                  );

    // Low-Active Circuit
    NAND2XL NAND3 (.A(S),          .B(lead_lag_n),   .Y(lead_lag)           );
    NAND2XL NAND4 (.A(lead_lag),   .B(R),            .Y(lead_lag_n)         );

    // Reset Signal
    NAND2XL NAND5 (.A(A),          .B(B),            .Y(self_reset)         );
    NAND2XL NAND6 (.A(B),          .B(A),            .Y(dummy)              );
    AND2X4  AND1  (.A(self_reset), .B(enable),       .Y(reset)              );

endmodule
```

Controller:

Signal Name	I/O	Width	Simple Description
clk	I	1	1 GHz 時脈訊號。
rst_n	I	1	低準位非同步(active low asynchronous)之系統重置信號。
lead_lag	I	1	lead_lag 為 1 時表示 clk_dll 領先 clk_os，為 0 則落後。
lambda	O	8	控制 TDL 的λ-code。
lock	O	1	lock 為 high 時表示 duty cycle 接近 50%。

利用 count 每 16 個 clock cycle 檢查一次。如果 lead_lag 與上次 lead_lag_last 不同則鎖定 lock 為 1，如果 lead_lag 為 1，代表 TDL 延遲太少，所以將α-code 加 1；如果 lead_lag 為 0，代表 TDL 延遲太多，所以將α-code 減 1，並且有α-code 邊界檢查條件。
每個有α-code 都有對應一組 one-hot 的λ-code。

```
module Controller (
    input             clk,
    input             rst_n,
    input             lead_lag,
    output reg [7:0]  lambda,
    output reg        lock
);
    reg        [2:0] alpha_next;
    reg        [2:0] alpha;
    reg        [3:0] count;
    reg              lock_next;
    reg              lead_lag_last;

    // count
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n)
            count <= 4'd1;
        else
            count <= count + 4'd1;
    end

    // lead_lag_last
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n)
            lead_lag_last <= 1'd0;
        else if (count == 4'd0)
            lead_lag_last <= lead_lag;
    end
```

```
// lock
always @(posedge clk or negedge rst_n) begin
    if (!rst_n)
        lock <= 1'd0;
    else
        lock <= lock_next;
end

// alpha
always @(posedge clk or negedge rst_n) begin
    if (!rst_n)
        alpha <= 3'd7;
    else
        alpha <= alpha_next;
end

// next logic
always @(*) begin
    lock_next = lock;
    alpha_next = alpha;
    if (count == 4'd15) begin
        if (lead_lag != lead_lag_last)
            lock_next = 1'b1;
        if (lead_lag) begin
            if (alpha == 3'd7)
                alpha_next = 3'd7;
            else
                alpha_next = alpha + 3'd1;
        end
        else if (!lead_lag) begin
            if (alpha == 3'd0)
                alpha_next = 3'd0;
            else
                alpha_next = alpha - 3'd1;
        end
    end
end

// decoder
always @(*) begin
    lambda = 8'd128;
    case (alpha)
        3'd0 : lambda = 8'd1;
        3'd1 : lambda = 8'd2;
        3'd2 : lambda = 8'd4;
        3'd3 : lambda = 8'd8;
        3'd4 : lambda = 8'd16;
        3'd5 : lambda = 8'd32;
        3'd6 : lambda = 8'd64;
        3'd7 : lambda = 8'd128;
        default : lambda = 8'd1;
    endcase
end
```

TDL:

Signal Name	I/O	Width	Simple Description
clk_in	I	1	輸入訊號。
lambda	I	8	控制 TDL 的λ-code。
clk_out	O	1	延遲訊號。

Path-selection TDL 我選擇以 MUX 和 BUF 來搭建而成。

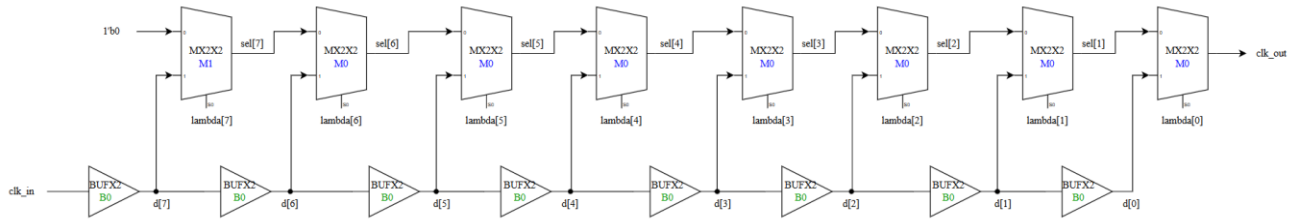
值得注意的是λ-code 是 one-hot，所以每次只有一條 path 會通，**每個 resolution 的差是 MX2X2 的 delay－BUFX2 的 delay**。

最短的 path 會經過八個 BUFX2 和一個 MX2X2；

次短的 path 會經過七個 BUFX2 和兩個 MX2X2；

次長的 path 會經過兩個 BUFX2 和七個 MX2X2；

最長的 path 會經過一個 BUFX2 和八個 MX2X2。



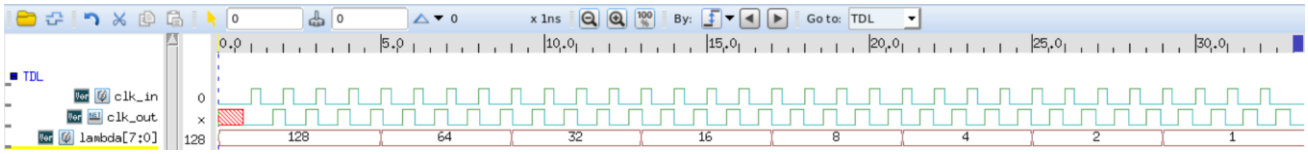

```
module TDL(
    input       clk_in,
    input [7:0] lambda,
    output wire clk_out
);

    parameter TAPS = 8;

    wire [TAPS-1:0] d;
    wire [TAPS-1:0] sel;

    assign d[TAPS-1] = clk_in;

    // buffer chain
    genvar i;
    generate
        for (i = TAPS-1; i > 0; i = i - 1) begin : buffer_chain
            BUFX2 B0(.A(d[i]), .Y(d[i-1]));
        end
    endgenerate

    // MUX chain for selection
    generate
        for (i = 0; i < TAPS-1; i = i + 1) begin : mux_chain
            MX2X2 M0(.A(sel[i+1]), .B(d[i]), .Y(sel[i]), .S0(lambda[i]));
        end
    endgenerate

    // last level
    MX2X2 M1(.A(1'b0), .B(d[TAPS-1]), .Y(sel[TAPS-1]), .S0(lambda[TAPS-1]));

    assign clk_out = sel[0];

endmodule
```

TDL_tb.v

```
`timescale 100ps/1ps
module TDL_tb();

    parameter period = 10;
    parameter real DUTY = 0.3; // 0.3 = 30%

    reg clk_in;
    wire clk_out;
    reg [7:0] lambda;

    TDL U0(
      .clk_in(clk_in),
      .lambda(lambda),
      .clk_out(clk_out)
    );

    initial begin
      clk_in = 0;
      forever begin
        clk_in = 1; #(period * DUTY);
        clk_in = 0; #(period * (1.0 - DUTY));
      end
      end

    initial begin
      clk_in = 0;   lambda = 8'd128;
      #(period * 5) lambda = 8'd64;
      #(period * 4) lambda = 8'd32;
      #(period * 4) lambda = 8'd16;
      #(period * 4) lambda = 8'd8;
      #(period * 4) lambda = 8'd4;
      #(period * 4) lambda = 8'd2;
      #(period * 4) lambda = 8'd1;
      #(period * 4)
      $finish;
    end
```

```
    // Delay measurement block
    real t_in, t_out, delay_ps;
    reg [7:0] lambda_last;
    reg measure_enable;

    initial begin
        lambda_last = lambda;
        measure_enable = 1;
        forever begin
            if (lambda != lambda_last) begin
                lambda_last = lambda;
                measure_enable = 1;
            end
            if (measure_enable) begin
                @(posedge clk_in);
                t_in = $realtime;
                @(posedge clk_out);
                t_out = $realtime;
                delay_ps = (t_out - t_in) * 100;
                $display("lambda = %d (decimal)    TDL delay = %0d ps", lambda, delay_ps);
                measure_enable = 0;
            end
            else begin
                @(posedge clk_in);
            end
        end
    end

    initial begin
        $sdf_annotate("./TDL_syn.sdf", U0);
        $fsdbDumpfile("../4.Simulation_Result/TDL_syn.fsdb");
        $fsdbDumpvars;
    end

endmodule
```

Waveform:




```
*Verdi* : Create FSDB file '../4.Simulation_Result/TDL_syn.fsdb'
*Verdi* : Begin traversing the scopes, layer (0).
*Verdi* : End of traversing.
lambda = 128 (decimal)    TDL delay = 673 ps
lambda =  64 (decimal)    TDL delay = 646 ps
lambda =  32 (decimal)    TDL delay = 620 ps
lambda =  16 (decimal)    TDL delay = 593 ps
lambda =   8 (decimal)    TDL delay = 565 ps
lambda =   4 (decimal)    TDL delay = 538 ps
lambda =   2 (decimal)    TDL delay = 512 ps
lambda =   1 (decimal)    TDL delay = 476 ps
$finish called from file "TDL_tb.v", line 44.
$finish at simulation time                33000
          V C S   S i m u l a t i o n   R e p o r t
```

測量結果:

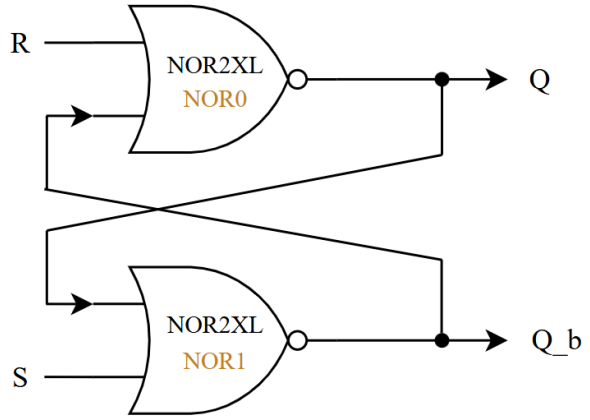
λ-code (decimal)	128	64	32	16	8	4	2	1
TDL Delay (ps)	673	646	620	593	565	538	512	476

$$Average\ resolution = \frac{673\ ps - 476\ ps}{7} \cong 28.14\ ps$$

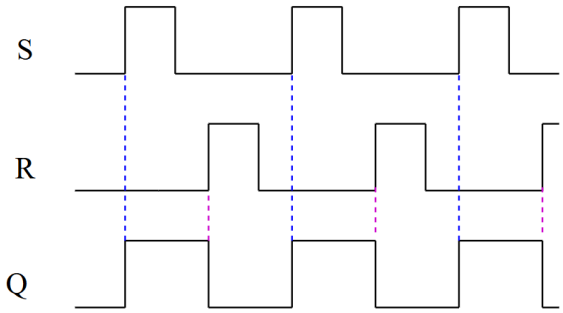
由上表可預測 DCC 會鎖定在λ-code 為 1 與 2 之間跳動時，因為 500 ps 剛好介於兩者延遲之間。

High-Active SR-Latch:

Signal Name	I/O	Width	Simple Description
S	I	1	Set.
R	I	1	Reset.
Q	O	1	Inversion of output Q_b.
Q_b	O	1	Inversion of output Q.



S	R	Q	Q_b	
1	0	1	0	
0	0	1	0	after S=1, R=0
0	1	0	1	
0	0	0	1	after S=0, R=1
1	1	0	0	




```
module SRLatch (
    input  S,
    input  R,
    output Q,
    output Q_b
);

    // High-Active Circuit
    NOR2XL NOR0 (.A(R),      .B(Q_b),     .Y(Q)   );
    NOR2XL NOR1 (.A(Q),      .B(S),       .Y(Q_b));
endmodule
```

Testbench:

```
`timescale 100ps/1ps
module DCC_tb();

    parameter period = 10;
    parameter delay = 1;
    parameter real DUTY = 0.7; // 0.3 = 30%


    reg         clk_in;
    reg         rst_n;

    wire        clk_dcc;
    wire [7:0]  lambda;
    wire        lock;


    initial begin
        clk_in = 0;
        forever begin
            clk_in = 1; #(period * DUTY);
            clk_in = 0; #(period * (1.0 - DUTY));
        end
    end


    DCC UUT (
            .clk_in(clk_in),
            .rst_n(rst_n),
            .clk_dcc(clk_dcc),
            .lambda(lambda),
            .lock(lock)
            );

    initial begin
                                        rst_n  = 1;
        #(period * 1)                   rst_n  = 0;
        #(period * 2 + delay * 2)       rst_n  = 1;
        #(period * 200)
        $finish;
    end
```

```
    // Duty cycle measurement block
    real t_rise, t_fall, t_next_rise;
    real high_time, period_time, duty_cycle, duty_cycle_n;

    integer cycle_cnt = 0;

    initial begin
        @(posedge clk_dcc);
        t_rise = $realtime;
        $display(" ------------------------- Simulation Start ----------------------------");
        $display("                            clk_in  [Duty Cycle] = %.1f %%", DUTY*100);
        $display(" -----------------------------------------------------------------------");
        forever begin
            @(negedge clk_dcc);
            t_fall = $realtime;

            @(posedge clk_dcc);
            t_next_rise = $realtime;
            high_time   = t_fall - t_rise;
            period_time = (t_next_rise - t_rise);
            duty_cycle  = (high_time / period_time) * 100.0;
            cycle_cnt = cycle_cnt + 1;

            if (cycle_cnt == 15) begin
                $display(" lambda = %d (decimal)    clk_dcc [Duty Cycle] = %.1f %%     period = %0d ns",
                        UUT.lambda, duty_cycle, period_time * 0.1);
                cycle_cnt = 0;
            end
            t_rise = t_next_rise;
        end
    end

    initial begin
        $sdf_annotate("./DCC_syn.sdf", UUT);
        $fsdbDumpfile("../4.Simulation_Result/DCC_syn.fsdb");
        $fsdbDumpvars;
    end
endmodule
```

4. Synthesis and Related Report

Synthesized Gate-level netlist:

```
////////////////////////////////////////////////////////////
// Created by: Synopsys DC Expert(TM) in wire load mode
// Version   : R-2020.09-SP5
// Date      : Mon Dec 22 13:43:31 2025
////////////////////////////////////////////////////////////


module OneShot ( din, dout );
  input din;
  output dout;

  wire   [3:0] w;

  CLKINVX8 CKINV ( .A(din), .Y(w[0]) );
  CLKBUFX1 CKBUF0 ( .A(w[0]), .Y(w[1]) );
  CLKBUFX1 CKBUF1 ( .A(w[1]), .Y(w[2]) );
  CLKBUFX1 CKBUF2 ( .A(w[2]), .Y(w[3]) );
  CLKAND2X2 CKAND ( .A(din), .B(w[3]), .Y(dout) );
endmodule


module PD ( clk_in, FB, enable, lead_lag );
  input clk_in, FB, enable;
  output lead_lag;
  wire   reset, A, B, R, S, lead_lag_n, self_reset;

  DFFRHQX4 FF1 ( .D(1'b1), .CK(FB), .RN(reset), .Q(A) );
  DFFRHQX4 FF2 ( .D(1'b1), .CK(clk_in), .RN(reset), .Q(B) );
  NAND2XL NAND1 ( .A(A), .B(R), .Y(S) );
  NAND2XL NAND2 ( .A(S), .B(B), .Y(R) );
  NAND2XL NAND3 ( .A(S), .B(lead_lag_n), .Y(lead_lag) );
  NAND2XL NAND4 ( .A(lead_lag), .B(R), .Y(lead_lag_n) );
  NAND2XL NAND5 ( .A(A), .B(B), .Y(self_reset) );
  NAND2XL NAND6 ( .A(B), .B(A), .Y() );
  AND2X4 AND1 ( .A(self_reset), .B(enable), .Y(reset) );
endmodule


module Controller ( clk, rst_n, lead_lag, lambda, lock );
  output [7:0] lambda;
  input clk, rst_n, lead_lag;
  output lock;
  wire   N15, N16, N17, lead_lag_last, n19, n20, n21, n22, n23, n1, n2, n3, n4,
         n5, n6, n7, n8, n9, n10, n11, n12, n13, n14, n150, n160, n170, n18,
         n24, n25, n26, n27, n28, n29, n30, n31, n32, n33, n34, n35, n36, n37,
         n38;
  wire   [3:0] count;
  wire   [2:0] alpha;

  DFFRQX2 count_reg_3_ ( .D(N17), .CK(clk), .RN(rst_n), .Q(count[3]) );
  DFFRQX2 count_reg_2_ ( .D(N16), .CK(clk), .RN(rst_n), .Q(count[2]) );
  DFFRHQX2 lock_reg ( .D(n22), .CK(clk), .RN(rst_n), .Q(lock) );
  DFFRHQX2 lead_lag_last_reg ( .D(n23), .CK(clk), .RN(rst_n), .Q(lead_lag_last) );
  DFFSX2 alpha_reg_1_ ( .D(n20), .CK(clk), .SN(rst_n), .Q(alpha[1]), .QN(n35)
       );
```

Area report:

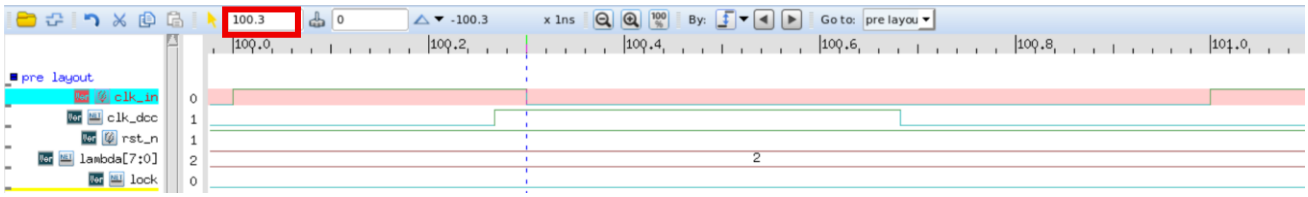
```
****************************************
Report : area
Design : DCC
Version: R-2020.09-SP5
Date   : Mon Dec 22 13:43:30 2025
****************************************

Information: Updating design information... (UID-85)
Information: Timing loop detected. (OPT-150)
    SRLatch/NOR1/A SRLatch/NOR1/Y SRLatch/NOR0/B SRLatch/NOR0/Y
Information: Timing loop detected. (OPT-150)
    PD/NAND2/A PD/NAND2/Y PD/NAND1/B PD/NAND1/Y
Warning: Disabling timing arc between pins 'A' and 'Y' on cell 'SRLatch/NOR1'
         to break a timing loop. (OPT-314)
Warning: Disabling timing arc between pins 'A' and 'Y' on cell 'PD/NAND2'
         to break a timing loop. (OPT-314)
Warning: Disabling timing arc between pins 'A' and 'Y' on cell 'PD/NAND4'
         to break a timing loop. (OPT-314)
Library(s) Used:

    slow (File: /usr/cadtool/ee5216/CBDK_TSMC90GUTM_Arm_f1.0/CIC/SynopsysDC/db/slow.db)

Number of ports:                          54
Number of nets:                          149
Number of cells:                         110
Number of combinational cells:            90
Number of sequential cells:               11
Number of macros/black boxes:              0
Number of buf/inv:                        29
Number of references:                      5

Combinational area:               370.440011
Buf/Inv area:                      76.910402
Noncombinational area:            193.334403
Macro/Black Box area:               0.000000
Net Interconnect area:      undefined  (No wire load specified)

Total cell area:                  563.774414
Total area:                 undefined
1
```

根據 90nm 的 NAND2X1 area 面積($2.52 \times 1.12\ \mu m^2$)換算

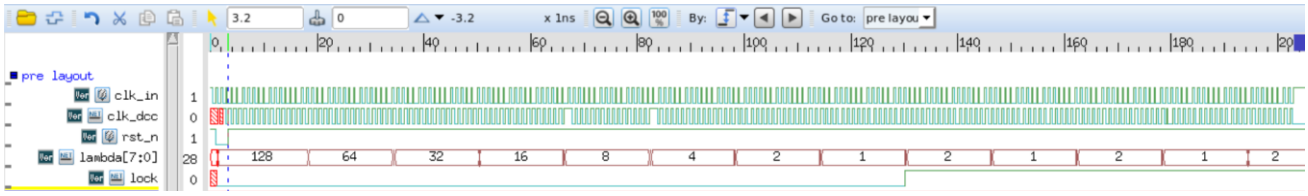
$$Gate\ count \cong \frac{563.774414\ \mu m^2}{2.52 \times 1.12\ \mu m^2} = \frac{563.774414\ \mu m^2}{2.8224\ \mu m^2} \cong 199.75\ of\ NAND2X1\ gate$$

5. Verification

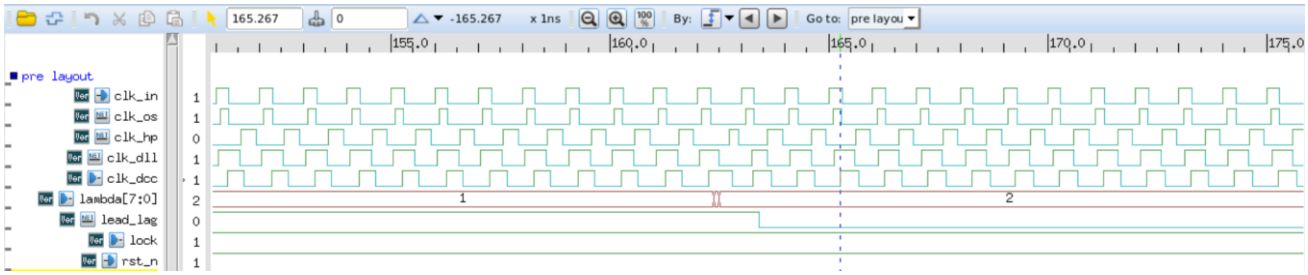
When clk_in duty cycle is 30%, also period is 1 ns.



DCC pre-layout waveform:



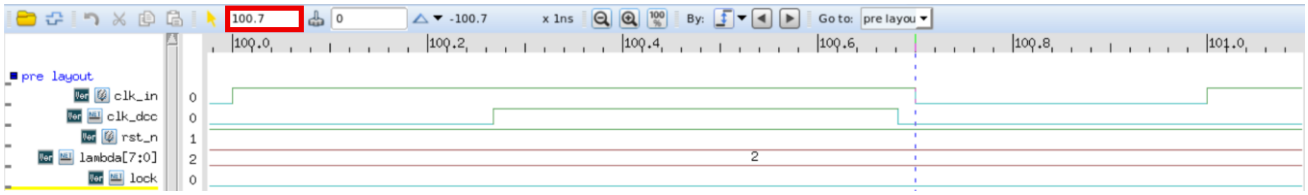
Zoom in the phase lock region:



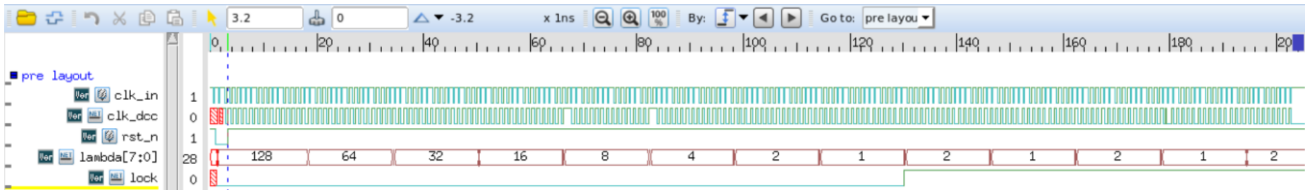
測量結果：

```
*Verdi* : Create FSDB file '../4.Simulation_Result/DCC_syn.fsdb'
*Verdi* : Begin traversing the scopes, layer (0).
*Verdi* : End of traversing.
 -------------------------- Simulation Start ----------------------------
                        clk_in  [Duty Cycle] = 30.0 %
 ------------------------------------------------------------------------
 lambda = 128 (decimal)    clk_dcc [Duty Cycle] = 58.3 %     period = 1 ns
 lambda =  64 (decimal)    clk_dcc [Duty Cycle] = 55.6 %     period = 1 ns
 lambda =  32 (decimal)    clk_dcc [Duty Cycle] = 52.7 %     period = 1 ns
 lambda =  16 (decimal)    clk_dcc [Duty Cycle] = 50.0 %     period = 1 ns
 lambda =   8 (decimal)    clk_dcc [Duty Cycle] = 47.3 %     period = 1 ns
 lambda =   4 (decimal)    clk_dcc [Duty Cycle] = 44.6 %     period = 1 ns
 lambda =   2 (decimal)    clk_dcc [Duty Cycle] = 41.5 %     period = 1 ns
 lambda =   1 (decimal)    clk_dcc [Duty Cycle] = 37.9 %     period = 1 ns
 lambda =   2 (decimal)    clk_dcc [Duty Cycle] = 41.5 %     period = 1 ns
 lambda =   1 (decimal)    clk_dcc [Duty Cycle] = 37.9 %     period = 1 ns
 lambda =   2 (decimal)    clk_dcc [Duty Cycle] = 41.5 %     period = 1 ns
 lambda =   1 (decimal)    clk_dcc [Duty Cycle] = 37.9 %     period = 1 ns
 lambda =   2 (decimal)    clk_dcc [Duty Cycle] = 41.5 %     period = 1 ns
$finish called from file "DCC_tb.v", line 39.
$finish at simulation time                203200
           V C S   S i m u l a t i o n   R e p o r t
Time: 203200 ps
CPU Time:      0.310 seconds;       Data structure size:   0.0Mb
Mon Dec 22 15:15:35 2025
```

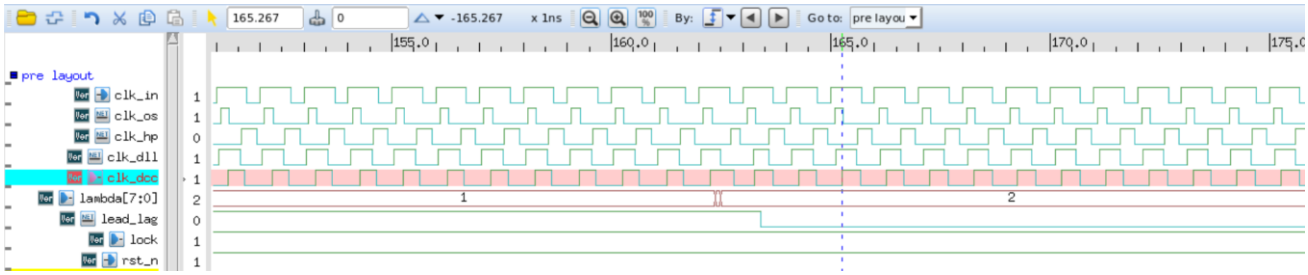
When clk_in duty cycle is 70%, also period is 1 ns.



DCC pre-layout waveform:



Zoom in the phase lock region:



測量結果:

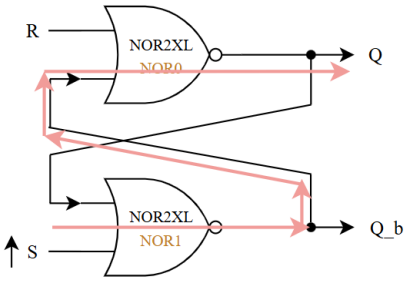
```
*Verdi* : Create FSDB file '../4.Simulation_Result/DCC_syn.fsdb'
*Verdi* : Begin traversing the scopes, layer (0).
*Verdi* : End of traversing.
 -------------------------- Simulation Start ----------------------------
                         clk_in  [Duty Cycle] = 70.0 %
 ------------------------------------------------------------------------
 lambda = 128 (decimal)    clk_dcc [Duty Cycle] = 58.3 %     period = 1 ns
 lambda =  64 (decimal)    clk_dcc [Duty Cycle] = 55.6 %     period = 1 ns
 lambda =  32 (decimal)    clk_dcc [Duty Cycle] = 52.7 %     period = 1 ns
 lambda =  16 (decimal)    clk_dcc [Duty Cycle] = 50.0 %     period = 1 ns
 lambda =   8 (decimal)    clk_dcc [Duty Cycle] = 47.3 %     period = 1 ns
 lambda =   4 (decimal)    clk_dcc [Duty Cycle] = 44.6 %     period = 1 ns
 lambda =   2 (decimal)    clk_dcc [Duty Cycle] = 41.5 %     period = 1 ns
 lambda =   1 (decimal)    clk_dcc [Duty Cycle] = 37.9 %     period = 1 ns
 lambda =   2 (decimal)    clk_dcc [Duty Cycle] = 41.5 %     period = 1 ns
 lambda =   1 (decimal)    clk_dcc [Duty Cycle] = 37.9 %     period = 1 ns
 lambda =   2 (decimal)    clk_dcc [Duty Cycle] = 41.5 %     period = 1 ns
 lambda =   1 (decimal)    clk_dcc [Duty Cycle] = 37.9 %     period = 1 ns
 lambda =   2 (decimal)    clk_dcc [Duty Cycle] = 41.5 %     period = 1 ns
$finish called from file "DCC_tb.v", line 39.
$finish at simulation time              203200
           V C S   S i m u l a t i o n   R e p o r t
Time: 203200 ps
CPU Time:      0.320 seconds;       Data structure size:   0.0Mb
Mon Dec 22 15:26:44 2025
```

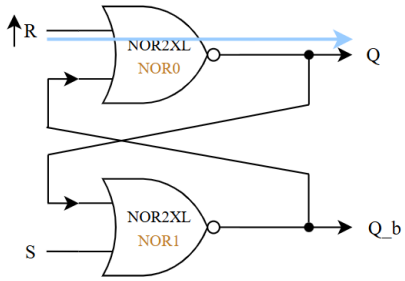
λ-code (decimal)	128	64	32	16	8	4	2	1
TDL Delay (ps)	673	646	620	593	565	538	512	476
Duty Cycle	58.3%	55.6%	52.7%	50.0%	47.3%	44.6%	41.5%	37.9%
Error	8.3%	5.6%	2.7%	0.0%	2.7%	5.4%	8.5%	12.1%

可由 Waveform 發現 Duty cycle 為 30%與 70%都是在λ-code 於 1、2 之間變動時鎖定(lock)，這與預期的結果相符，但是由上表可看出當λ-code 為 16 時 duty-cycle 才是完美的 50%，λ-code 為 1 或 2 時的 duty cycle 與 50%相差甚大，會有如此誤差，我認為是以下原因導致。

SR Latch 當 S go high 須經過兩個 NOR2 才會抵達 Q，而 R go high 只須經過一個 NOR2 就會抵達 Q，因此造成結果與預期不符。

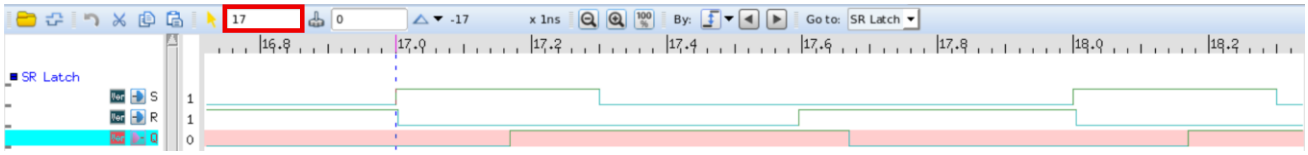
S	R	Q	Q_b	
1	0	1	0	
0	0	1	0	after S=1, R=0
0	1	0	1	
0	0	0	1	after S=0, R=1
1	1	0	0	



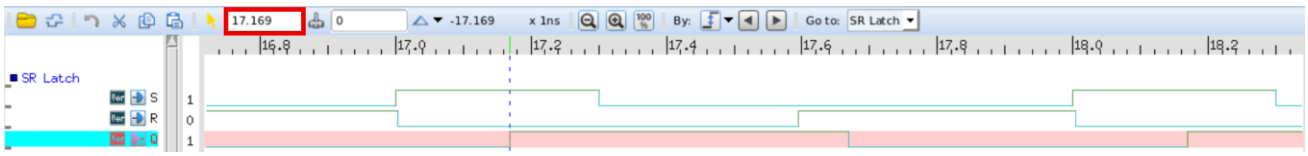


Proof for SR Latch Delay:

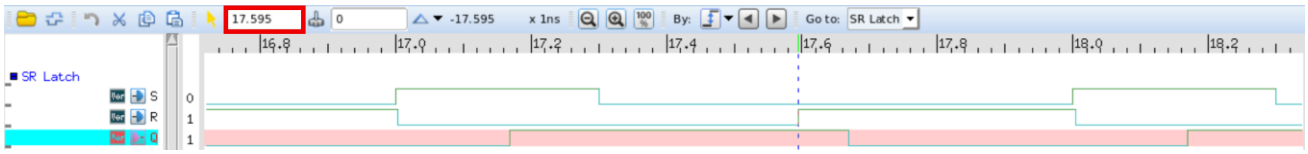
$S\uparrow$



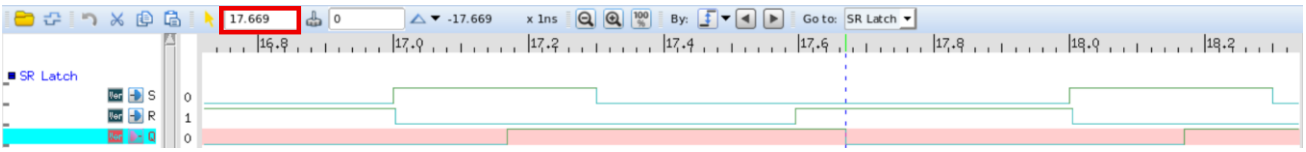
$Q\uparrow$



$R\uparrow$



$Q\uparrow$



$$Delay(S\uparrow \rightarrow Q\uparrow) = 17.169\,ns - 17.000\,ns = 169\,ps$$

$$Delay(R\uparrow \rightarrow Q\uparrow) = 17.669\,ns - 17.595\,ns = 74\,ps$$

$$Delay(S\uparrow \rightarrow Q\uparrow) - Delay(R\uparrow \rightarrow Q\uparrow) = 169\,ps - 74\,ps = 95\,ps$$

這說明實際上 Pulse width 會比理想中少了約$95\,ps$。