Timing Circuit Designs and Their Applications

Homework 2
Phased-Locked Loop

姓名：宋易柔（Yi-Rou, Song)

學號：114061529

系所：電機系 (EE)

1. 題目說明

設計運作在 100MHz 的 PLL，並輸出 1GHz 時脈訊號。

2. DCO

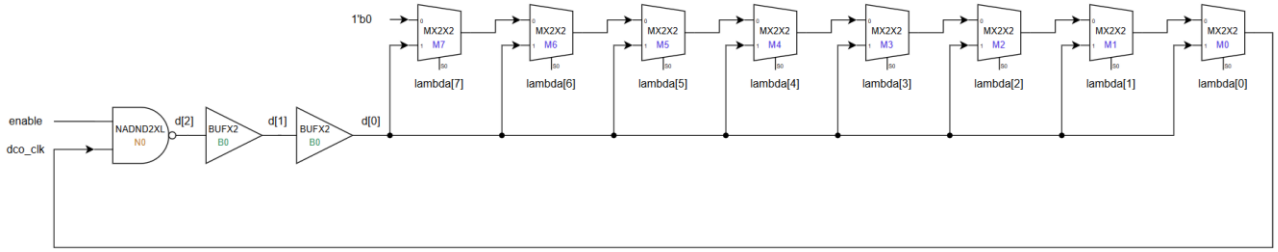
DCO:

Signal Name	I/O	Width	Simple Description
enable	I	1	enable 拉起時表示 DCO 開始振盪。
lambda	I	8	控制 DCO 的λ-code。
clk_dco	O	1	DCO 輸出振盪訊號。

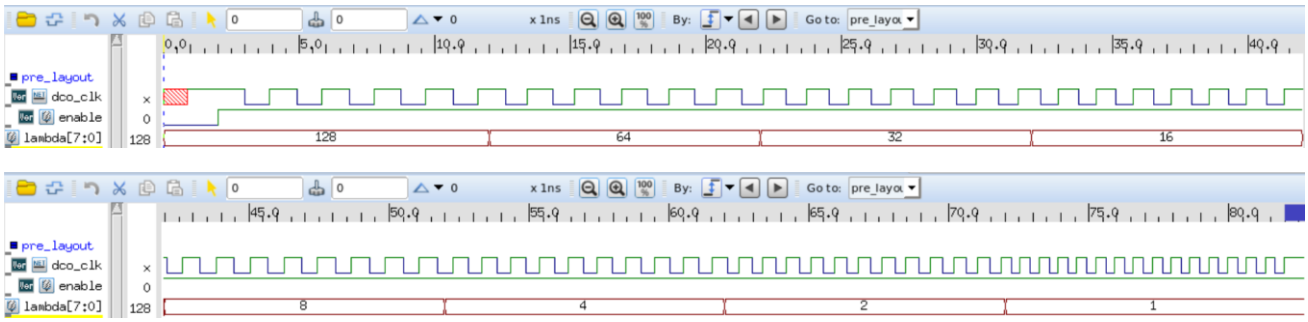
由於作業提供的 DCO 架構使用的 tri-buffer 會因為λ-code 的變動不穩定容易輸出 z 或 unknown，並且每個 resolution 稍大，不適用於 1GHz 的 DCO，所以我微調架構，同樣有一個 NAND2XL 做訊號反向和兩個 BUFX2，不同的是使用八個 MX2X2 取代 tri-buffer。值得注意的是λ-code 是 one-hot，所以每次只有一條 path 會通，每個 resolution 的差是 MX2X2 的 delay。

最短的 path 會經過一個 NAND、兩個 BUF 和一個 MUX；

最長的 path 會經過一個 NAND、兩個 BUF 和八個 MUX。



Pre-layout:

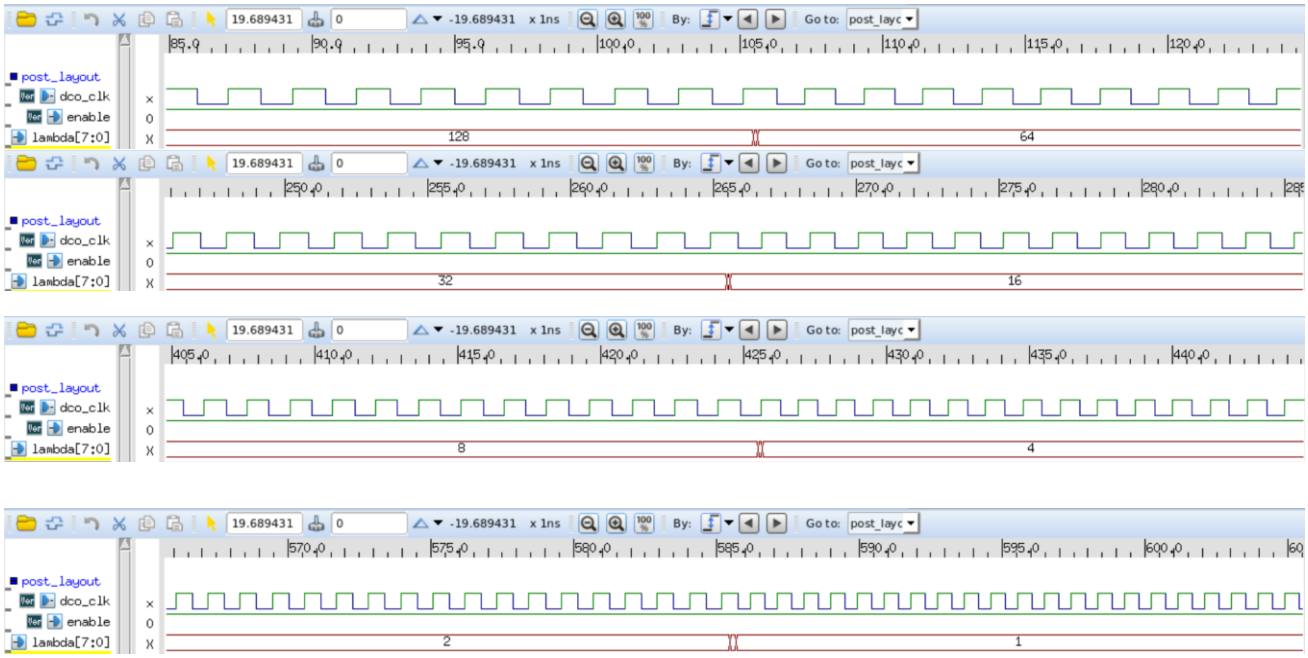


λ-code (decimal)	128	64	32	16	8	4	2	1
Period (ps)	2021	1843	1663	1483	1305	1127	947	765
Frequency (MHz)	494.48	542.59	601.32	674.31	766.28	887.31	1055.96	1307.19

DCO 的振盪周期的可調變的範圍為 2021 ps ~ 765 ps，對應的頻率範圍為 494.48 MHz ~ 1307.19 MHz。

$$Timing\ Resolution\ = \frac{2021 - 765}{7} = 179.43\ ps$$

Post-layout:



λ-code (decimal)	128	64	32	16	8	4	2	1
Period (ns)	2258	2067	1882	1694	1498	1309	1123	935
Frequency (MHz)	442.86	483.79	531.35	590.32	667.56	763.94	890.47	1069.52

DCO 的振盪周期的可調變的範圍為 2258 ps ~ 935 ps，對應的頻率範圍為 494.48 MHz ~ 1307.19 MHz。

$$Timing\ Resolution\ = \frac{2258 - 935}{7} = 189\ ps$$

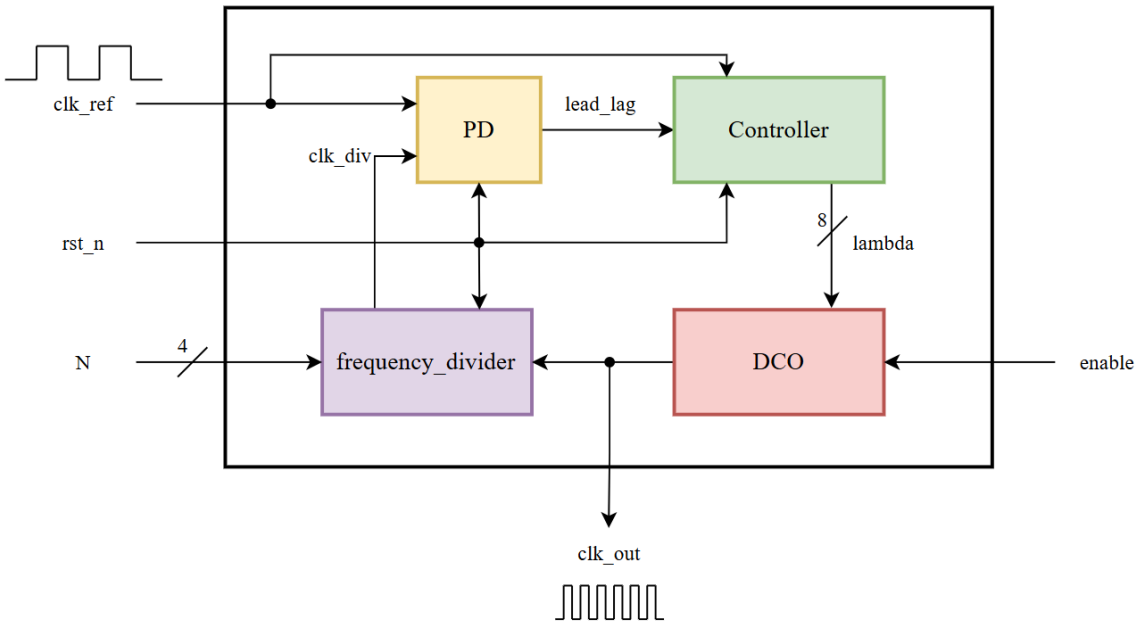
```
module DCO(
    input       enable,
    input [7:0] lambda,
    output wire dco_clk
);

    wire  [6:0] c;

    parameter buffer_chain = 2;
    wire  [buffer_chain:0] d;

    NAND2XL N0(.A(enable), .B(dco_clk), .Y(d[buffer_chain]));

    MX2X2 M0(.A(c[0]), .B(d[0]), .Y(dco_clk), .S0(lambda[0]));
    MX2X2 M1(.A(c[1]), .B(d[0]), .Y(c[0]),    .S0(lambda[1]));
    MX2X2 M2(.A(c[2]), .B(d[0]), .Y(c[1]),    .S0(lambda[2]));
    MX2X2 M3(.A(c[3]), .B(d[0]), .Y(c[2]),    .S0(lambda[3]));
    MX2X2 M4(.A(c[4]), .B(d[0]), .Y(c[3]),    .S0(lambda[4]));
    MX2X2 M5(.A(c[5]), .B(d[0]), .Y(c[4]),    .S0(lambda[5]));
    MX2X2 M6(.A(c[6]), .B(d[0]), .Y(c[5]),    .S0(lambda[6]));
    MX2X2 M7(.A(1'b0), .B(d[0]), .Y(c[6]),    .S0(lambda[7]));

    genvar i;

    generate
        for (i = 0; i < buffer_chain; i = i + 1) begin:loop1
            BUFX2 B0( .A (d[i+1]), .Y (d[i]) );
        end
    endgenerate

endmodule
```

3. Design

PLL:

Signal Name	I/O	Width	Simple Description
clk_ref	I	1	100 MHz 時脈訊號。
rst_n	I	1	低準位非同步(active low asynchronous)之系統重置信號。
enable	I	1	enable 拉起時表示 DCO 開始振盪。
N	I	4	除數。
clk_dco	O	1	PLL 輸出時脈訊號。
lambda	O	8	控制 DCO 的λ-code。

PLL 裡有四個 sub-module，分別是 PD、Controller、DCO 與 frequency_divider。

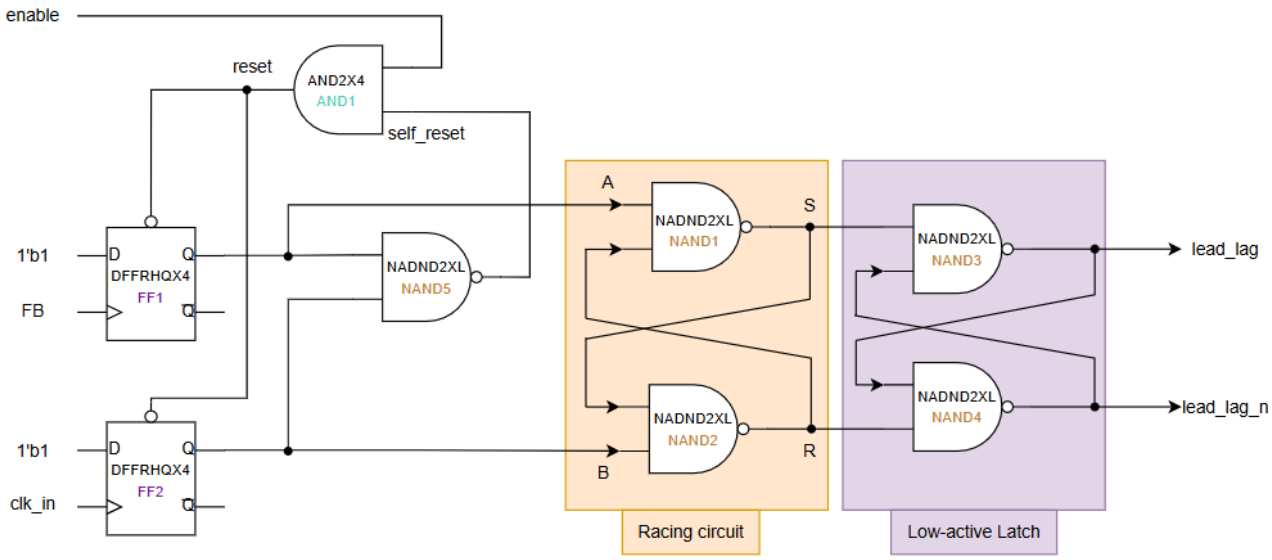


```v
`include "../2.Gate_level_simulation/PD_syn.v"
`include "../2.Gate_level_simulation/DCO_syn.v"
`include "../2.Gate_level_simulation/frequency_divider_syn.v"
`include "Controller.v"

module PLL (
    input          clk_ref,
    input          rst_n,
    input          enable,
    input  [3:0]   N,
    output         clk_out,
    output         lead_lag,
    output [7:0]   lambda
);

    wire           clk_div;

    Controller Controller (
        .lead_lag (lead_lag),
        .clk      (clk_ref),
        .rst_n    (rst_n),
        .lambda   (lambda)
    );

    PD PD (
        .clk_in   (clk_ref),
        .FB       (clk_div),
        .enable   (rst_n),
        .lead_lag (lead_lag)
    );

    DCO DCO (
        .enable   (enable),
        .lambda   (lambda),
        .dco_clk  (clk_out)
    );

    frequency_divider frequency_divider (
        .fin      (clk_out),
        .N        (N),
        .rst_n    (rst_n),
        .fout     (clk_div)
    );

endmodule
```

Phase Detector:

Signal Name	I/O	Width	Simple Description
clk_in	I	1	100 MHz 時脈訊號。
FB	1	1	frequency_divider 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 FB 領先 clk_in，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
```

Frequency Divider:

Signal Name	I/O	Width	Simple Description
fin	I	1	DCO 輸出振盪訊號。
rst_n	I	1	低準位非同步(active low asynchronous)之系統重置信號。
N	I	4	除數。
fout	O	1	除頻訊號。

此模組的功能為將 fin 頻率轉換為較低的 $\frac{fin}{N}$ 頻率。

N 由除數器產生，而 N_h 是 N 的一半，利用二進制的概念，往後移一位即為除以 2。運作原理是每當輸入 fin 的正緣觸發時，計數器 count 若尚未達到 N 則加 1，否則計為 1；輸出 fout 則利用 N_h 的大小控制 1 或 0，當計數器 count 小於等於 N_h 則輸出 1，當計數器 count 大於 N_h 則輸出 0。

```
module frequency_divider (
    input        fin,
    input        rst_n,
    input  [3:0] N,
    output reg   fout
);

    reg    [3:0] count;

    // Half N
    wire   [3:0] N_h = {1'b0, N[3:1]};

    // count
    always @(posedge fin or negedge rst_n) begin
        if (!rst_n)
            count <= 4'd1;
        else
            count <= (count >= N)? 4'd1 : count + 4'd1;
    end

    // fout
    always @(posedge fin or negedge rst_n) begin
        if (!rst_n)
            fout <= 1'b0;
        else
            fout <= (count <= N_h)? 1'b1 : 1'b0;
    end

endmodule
```

Controller:

Signal Name	I/O	Width	Simple Description
clk	I	1	100 MHz 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 FB 領先 clk_in，為 0 則落後。
lambda	O	8	控制 DCO 的λ-code。

利用 count 每 8 個 clock cycle 檢查一次 lead_lag，如果 lead_lag 為 1，代表 DCO 振盪太快，所以將α-code 加 1；如果 lead_lag 為 0，代表 DCO 振盪太慢，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            lead_lag,
    input            clk,
    input            rst_n,
    output reg [7:0] lambda
);

    reg        [2:0] alpha_next;
    reg        [2:0] alpha;
    reg        [2:0] count;

    // count
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n)
            count <= 3'd1;
        else
            count <= count + 3'd1;
    end

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

    always @(*) begin
        alpha_next = alpha;
        if (lead_lag && count == 7) begin
            if (alpha == 3'd7)
                alpha_next = 3'd7;
            else
                alpha_next = alpha + 3'd1;
        end
        else if (!lead_lag && count == 7) begin
            if (alpha == 3'd0)
                alpha_next = 3'd0;
            else
                alpha_next = alpha - 3'd1;
        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

endmodule
```

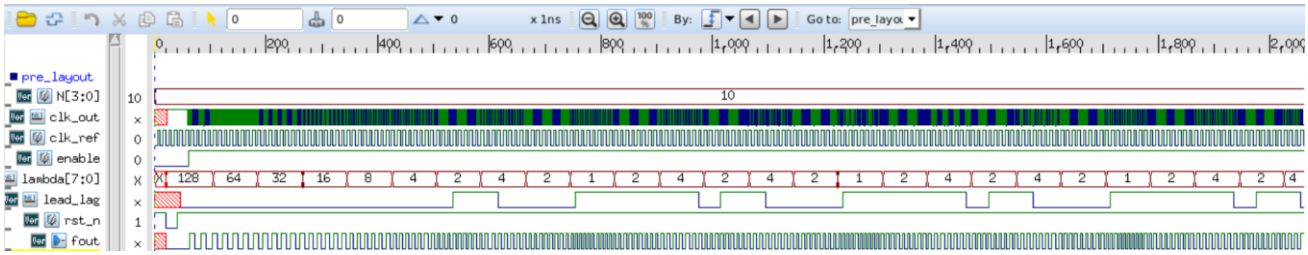
Testbench:

N 設為 10，代表除頻為 1/10 frequency。

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

    parameter period = 100;

    reg         clk_ref;
    reg         rst_n;
    reg         enable;
    reg  [3:0] N;

    wire        clk_out;
    wire        lead_lag;
    wire [7:0] lambda;

    PLL UUT (
            .clk_ref(clk_ref),
            .rst_n(rst_n),
            .N(N),
            .enable(enable),
            .clk_out(clk_out),
            .lead_lag(lead_lag),
            .lambda(lambda)
            );

    always #(period/2) clk_ref = ~clk_ref;

    initial begin
      clk_ref  = 0; rst_n  = 1; N = 10; enable = 0;
      #(period * 2) rst_n  = 0;
      #(period * 2) rst_n  = 1;
      #(period * 2) enable = 1;
      #100000
      $finish;
    end

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

4. Pre-layout Waveform

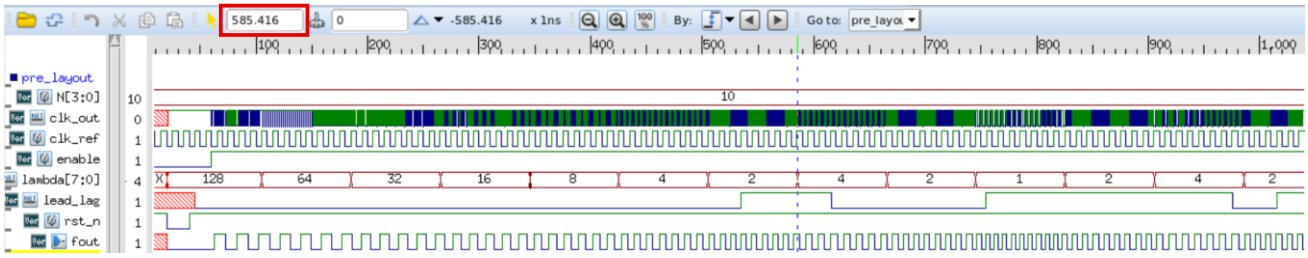
Pre-layout waveform:



λ-code (decimal)	128	64	32	16	8	4	2	1
dco_clk Period (ps)	2021	1843	1663	1483	1305	1127	947	765

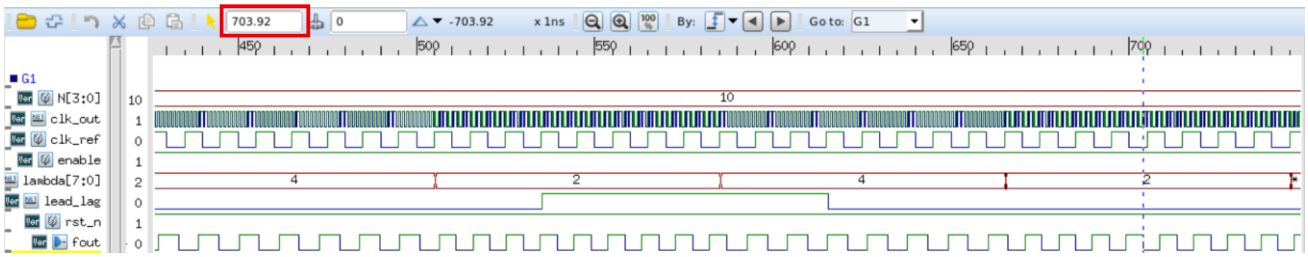
當λ-code=4 時，dco_clk 的 period 剛好大於 1000 ps；
當λ-code=2 時，dco_clk 的 period 剛好小於 1000 ps。
因此可預期λ-code 會在這附近跳動。

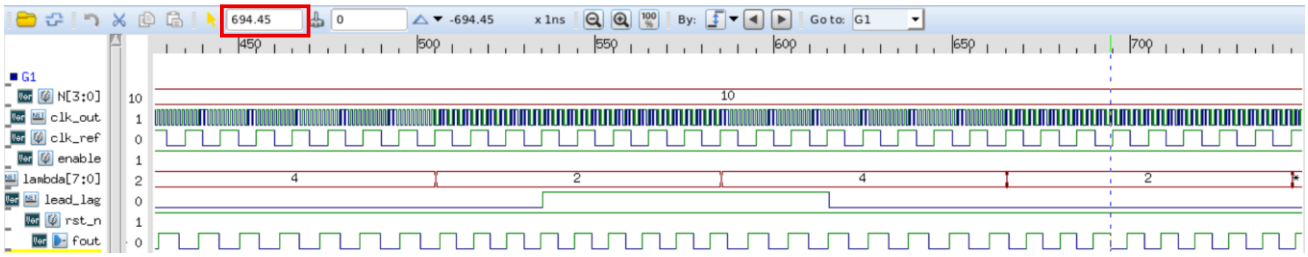
frequency acquisition:



在 585.416 ns 確認 frequency acquisition，λ-code 徘徊在 2、4 之間，分別測量λ-code=2 與λ-code=4，每十個 clk_out 的波形可以將 frequency_divider 的輸出(fout)拉出來觀察。

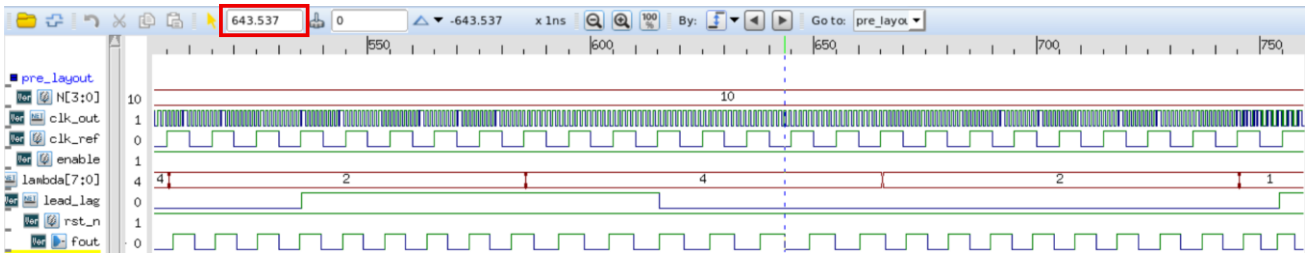
λ-code=2:

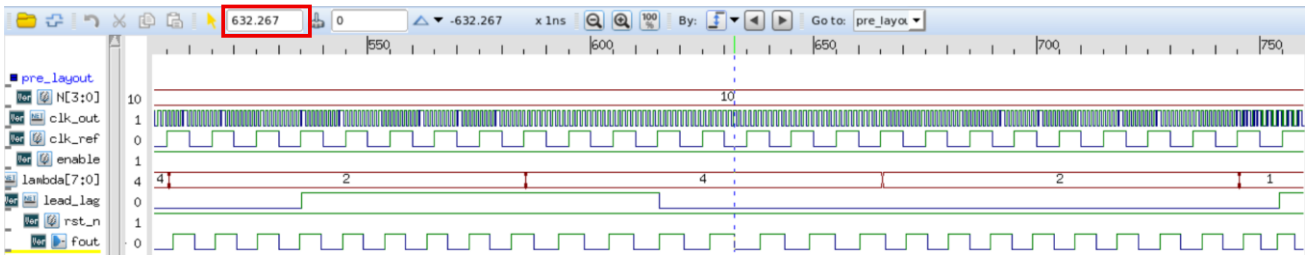




fout 的 period = 703.92-694.45=9.47 ns

λ-code=4:



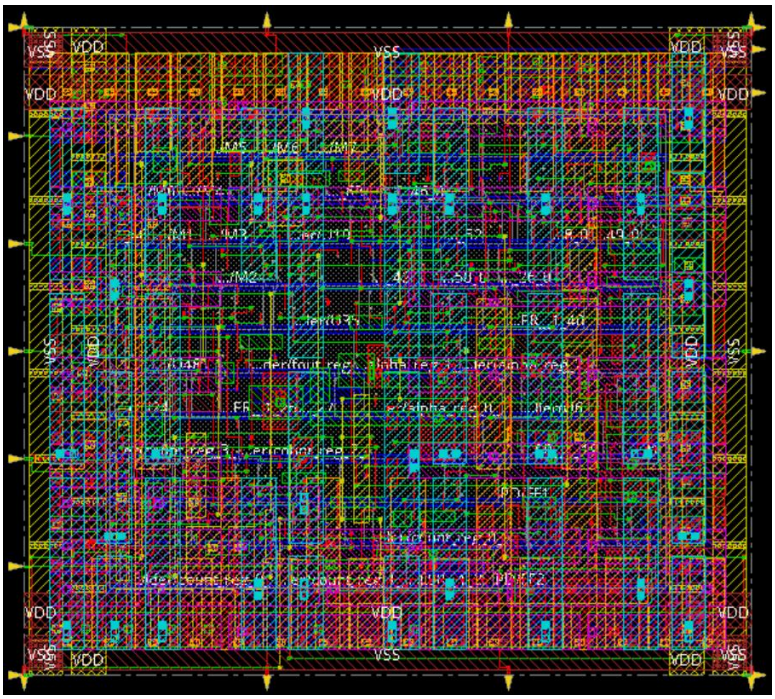


fout 的 period = 643.537-632.267 = 11.27 ns

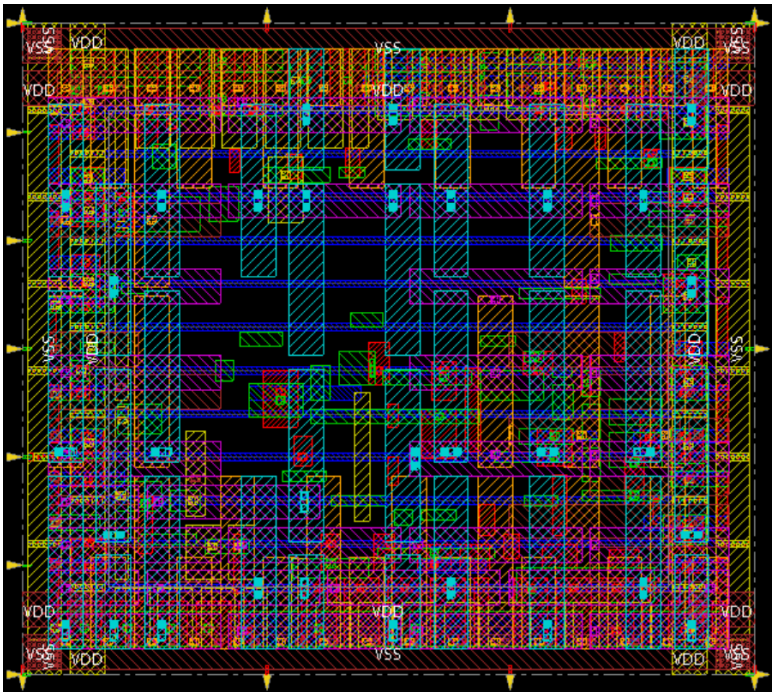
$$error\ compared\ to\ the\ ideal\ clock\ cycle\ time\ (1\ ns)$$
$$= \frac{Period(\lambda = 4) - Period(\lambda = 2)}{1} \times 100\%$$
$$= \frac{|11.27 - 9.47|}{10} \times 100\% = 18\%$$

5. P&R

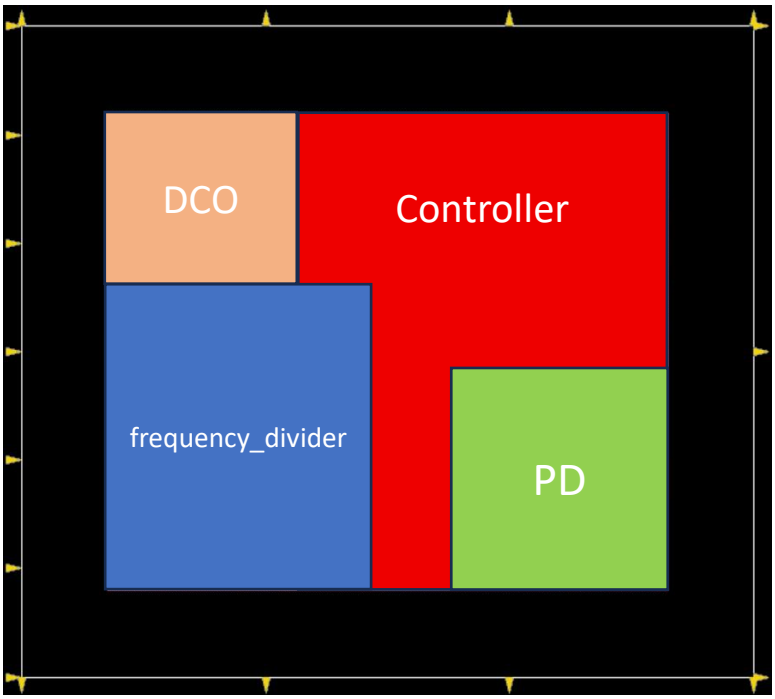
Physical view:



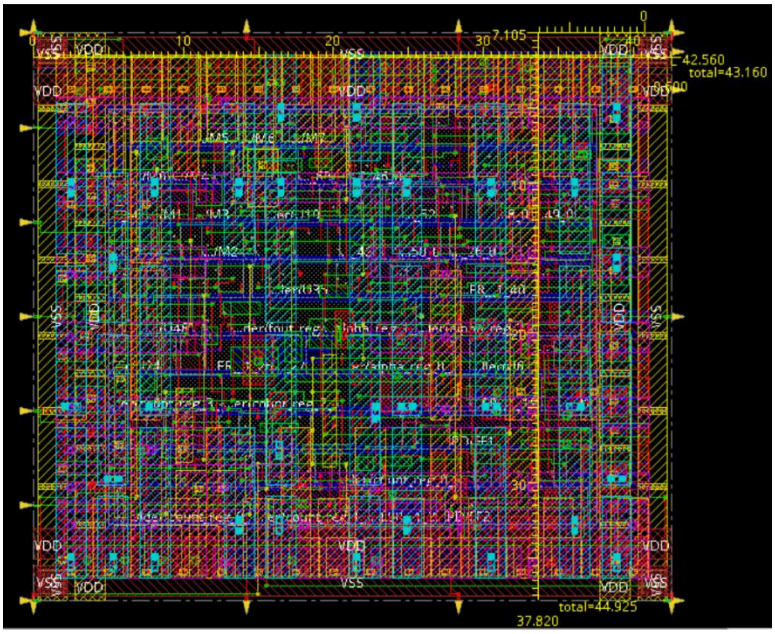
Floorplan view:



Amoeba view:



Measured Area:



Total area is $43.160(\mu m) \times 43.820(\mu m) = 1632.3112(\mu m^2)$.

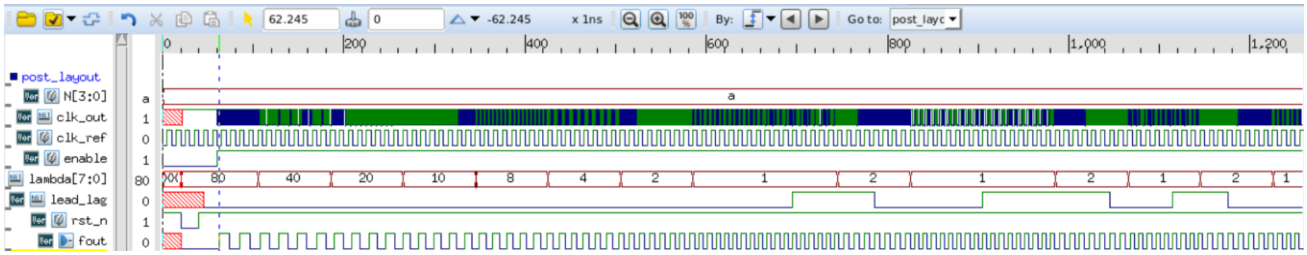
```
innovus 2> dbGet top.fplan.coreBox_area
900.3456
innovus 3> dbshape -output area [dbGet top.fplan.boxes]
1608.768
```

Core area is $\boldsymbol{900.3456\ (\mu m^2)}$.

Chip area is $\boldsymbol{1608.768\ (\mu m^2)}$.

6. Post-layout Waveform

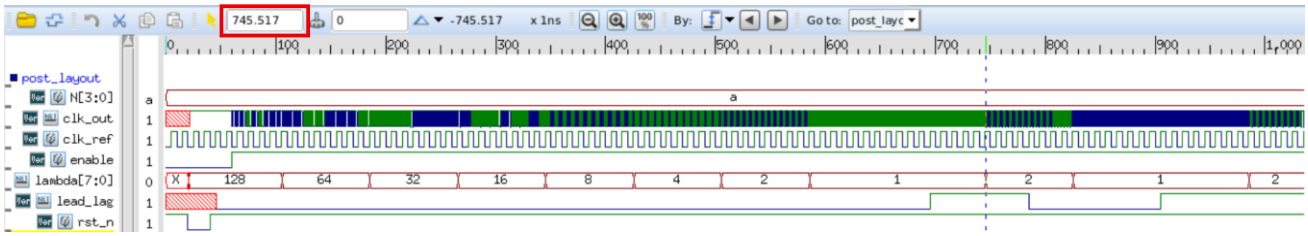
Post-layout simulation waveform:



λ-code (decimal)	128	64	32	16	8	4	2	1
dco_clk Period (ps)	2258	2067	1882	1694	1498	1309	1123	935

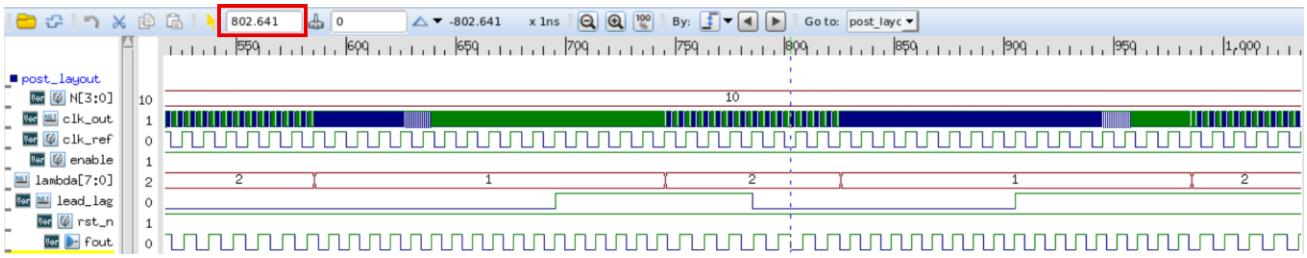
當λ-code=2 時，dco_clk 的 period 剛好大於 1000 ps；
當λ-code=1 時，dco_clk 的 period 剛好小於 1000 ps。
因此可預期λ-code 會在這附近跳動。

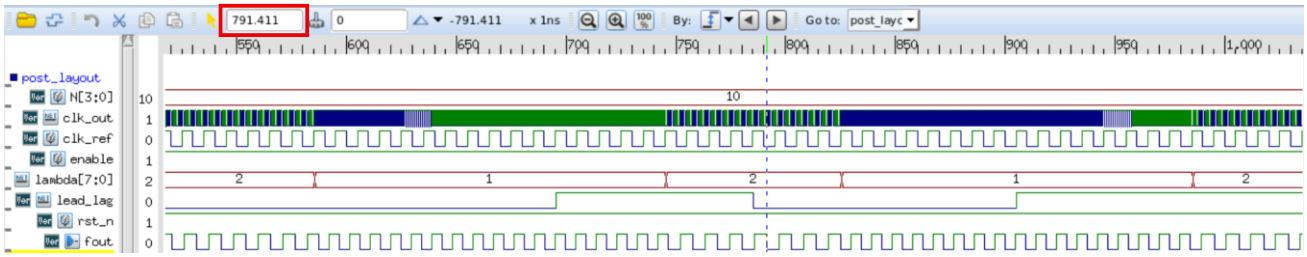
frequency acquisition:



在 745.517 ns 確認 frequency acquisition，λ-code 徘徊在 1、2 之間，分別測量λ-code=1 與λ-code=2，每十個 clk_out 的波形可以將 frequency_divider 的輸出(fout)拉出來觀察。

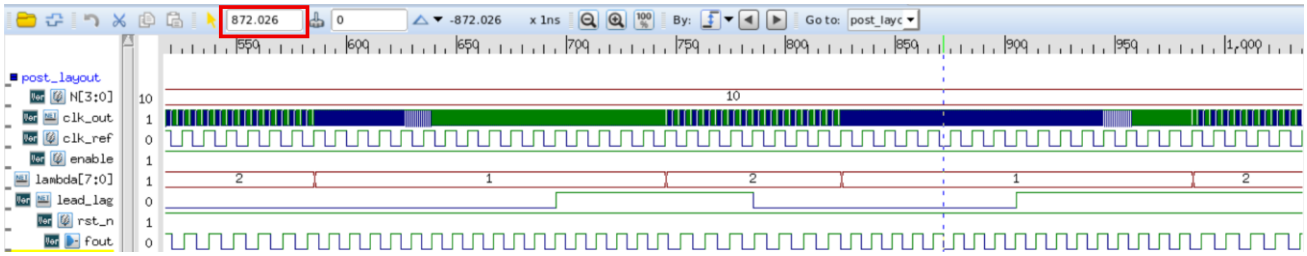
λ-code=2:

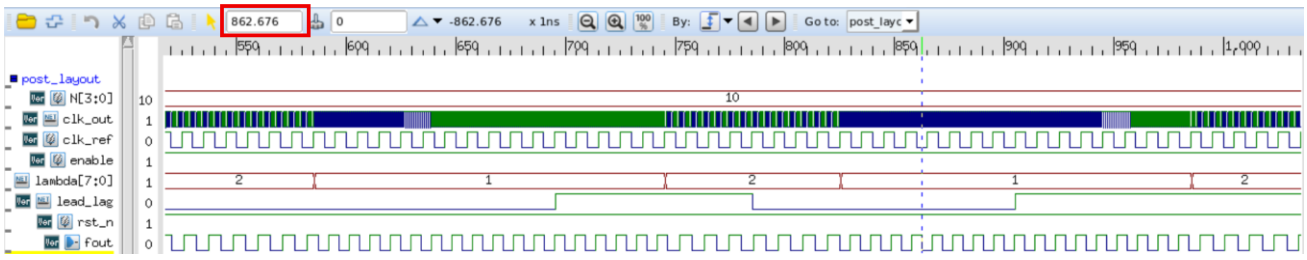




fout 的 period = 802.641-791.411=11.23 ns

λ-code=1:

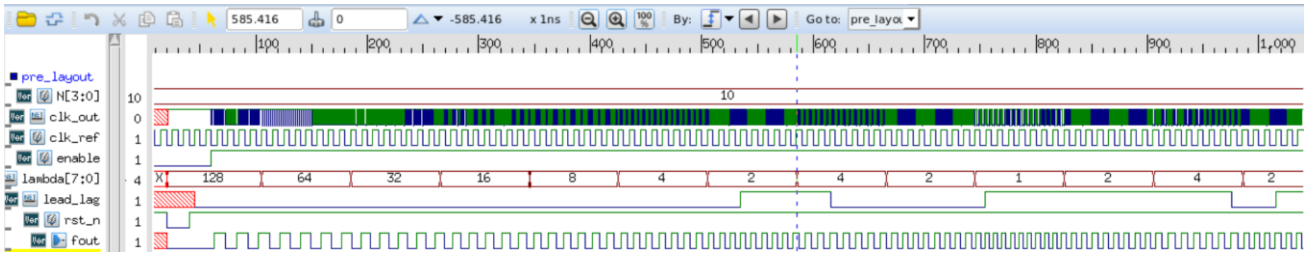




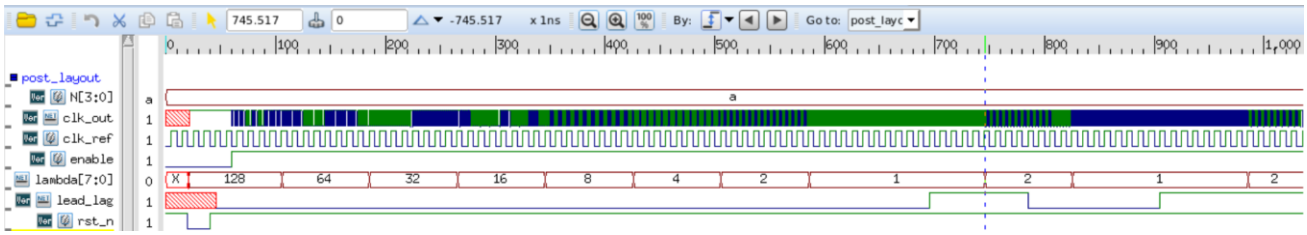
fout 的 period = 872.026-862.676=9.35 ns

$$error\ compared\ to\ the\ ideal\ clock\ cycle\ time\ (1\ ns)$$
$$= \frac{Period(\lambda = 2) - Period(\lambda = 1)}{10} \times 100\%$$
$$= \frac{|11.23 - 9.35|}{10} \times 100\% = 18.8\%$$

Compare:



Pre-layout 在 585.416 ns 確認 frequency acquisition，λ-code 徘徊在 2、4 之間，脫鎖後λ-code 會變成 1。



Post-layout 在 745.517 ns 確認 frequency acquisition，λ-code 徘徊在 1、2 之間，並沒有脫鎖發生。

λ-code (decimal)	128	64	32	16	8	4	2	1
pre-layout clk_out Period (ps)	2021	1843	1663	1483	1305	1127	947	765
post-layout clk_out Period (ps)	2258	2067	1882	1694	1498	1309	1123	935

整體而言，layout 後 DCO 的振盪週期有抬升的趨勢。