Succesive_Approximation_Controller_Report

112061533 潘金生

A. Objective:

Controller that locks to a specific value by successive approximation.

B. Experience to be Learnt from this Homework:

To get familiar with the front-end synthesis-based digital design flow(including RTL coding, Testbench development, Simulation, and Synthesis) commonly used in an all-digital timing circuit design

C. Functionality:

Generate a linear function y=2.4*x+300, where x is the value of an 8-bit digital code x[7:0]. As a result, y is a value in the range of [300, 912.2]. Try to find a proper value of x so that the value of y is closest to a given target value, target.

D. Step-by-Step Procedure

a. Goal :

Develop a software program in any software language (e.g., C/C++ or Python) that can solve the above problem, using the successive approximation scheme. Verify your program by trying target values of 550 and 800。

Execute :

我是使用 python 軟體去完成 successive approximation scheme(sa)，我實作了兩個版本，兩個皆由 binary search 去達到目的，第一版本(v1)使用較傳統的 binary seach 方法實作，第二版本(v2)使用課堂上所教的 binary seach 方法帶入硬體的概念去實作。

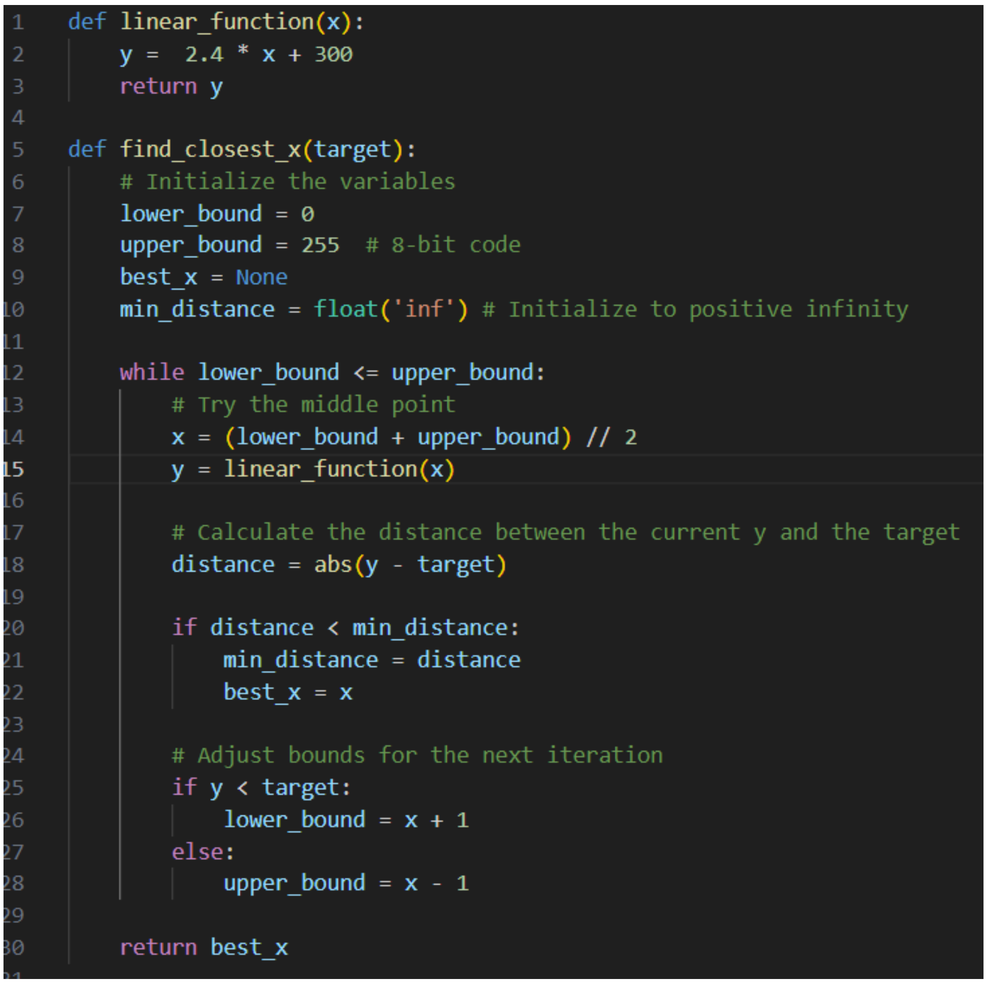
sa_v1 程式介紹:

主要分成兩個函數實作:

linear_function: 產生 y=2.4*x+300 目標函數。

Find_closet_x:

1. 首先進行初始化參數，將上界定為 255(8bit)，下界定為 0，並把 y 和 target_y 其距離先設成無限大(min_distance)。
2. 進行迴圈判斷如果下界小於上界，則假設 x 為平均數帶入。
3. 將假設 x 算出 y 並計算與 target_y 差距
4. 如果距離小於先前的 min_distance，判斷目前為最好的 x
5. 更新上下界。

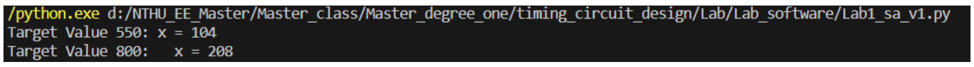


```python
def linear_function(x):
    y =  2.4 * x + 300
    return y

def find_closest_x(target):
    # Initialize the variables
    lower_bound = 0
    upper_bound = 255  # 8-bit code
    best_x = None
    min_distance = float('inf') # Initialize to positive infinity

    while lower_bound <= upper_bound:
        # Try the middle point
        x = (lower_bound + upper_bound) // 2
        y = linear_function(x)

        # Calculate the distance between the current y and the target
        distance = abs(y - target)

        if distance < min_distance:
            min_distance = distance
            best_x = x

        # Adjust bounds for the next iteration
        if y < target:
            lower_bound = x + 1
        else:
            upper_bound = x - 1

    return best_x
```

圖一 sa_v1 python code

sa_v1 實驗結果:

由下圖可知，在 target value 550 時 x 為 104，target value 800 時 x 為 208。



```
/python.exe d:/NTHU_EE_Master/Master_class/Master_degree_one/timing_circuit_design/Lab/Lab_software/Lab1_sa_v1.py
Target Value 550: x = 104
Target Value 800:    x = 208
```

圖二 sa_v1 實驗結果

sa_v2 程式介紹:

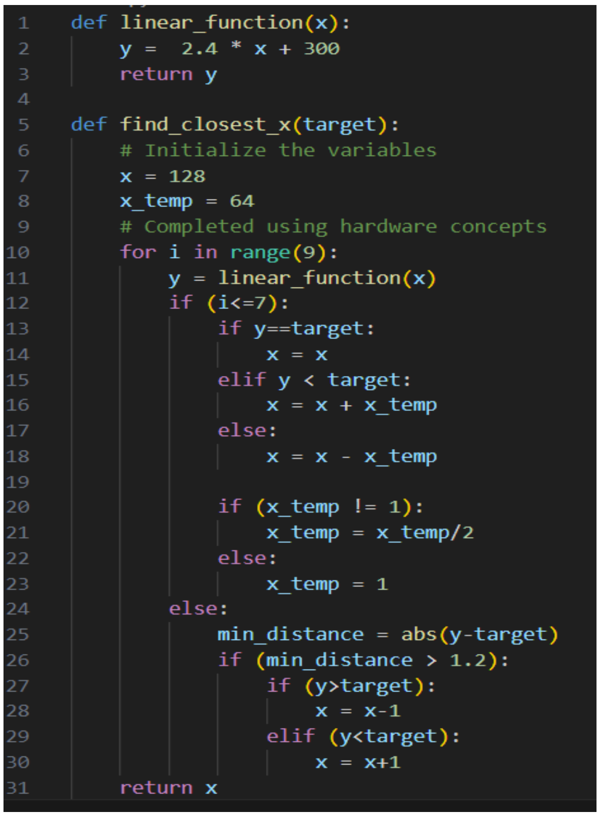
主要分成兩個函數實作:

linear_function: 產生 y=2.4*x+300 目標函數。

Find_closet_x:

1. 首先進行初始化參數，x = 128 相當於 8bits:10000000，在創造專門存 shift 的 x_temp 為 64 相當於 8bits:01000000
2. 進行迴圈判斷當前 x 算出的 y 與 target_y 大小，決定是否要加或減 x_temp

3. 算到最後 1 個 bit 時，需考慮目前是否為最接近 target_y 的 x，如果與 target 差了大於 1.2 則須考慮目前+1,-1 的問題。

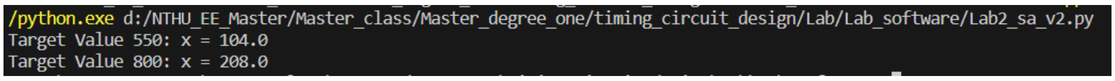


```python
def linear_function(x):
    y =  2.4 * x + 300
    return y

def find_closest_x(target):
    # Initialize the variables
    x = 128
    x_temp = 64
    # Completed using hardware concepts
    for i in range(9):
        y = linear_function(x)
        if (i<=7):
            if y==target:
                x = x
            elif y < target:
                x = x + x_temp
            else:
                x = x - x_temp

            if (x_temp != 1):
                x_temp = x_temp/2
            else:
                x_temp = 1
        else:
            min_distance = abs(y-target)
            if (min_distance > 1.2):
                if (y>target):
                    x = x-1
                elif (y<target):
                    x = x+1
    return x
```

圖三 sa_v2 python code

sa_v2 實驗結果:

由下圖可知，在 target value 550 時 x 為 104，target value 800 時 x 為 208。



```
/python.exe d:/NTHU_EE_Master/Master_class/Master_degree_one/timing_circuit_design/Lab/Lab_software/Lab2_sa_v2.py
Target Value 550: x = 104.0
Target Value 800: x = 208.0
```

圖四 sa_v2 實驗結果

b. Goal :

Convert your software subroutine into a synthesizable RTL code (in Verilog or VHDL). Verify your RTL code with a testbench. Make sure that the results are consistent with those produced by your software program.

Execute:

1. Synthesizable RTL code (Verilog):

   Verilog 主要實現 successive approximation 的方式是使用 FSM 完成，我採用 5 個 state 來進行，分別為 IDLE, CALC, UPDATE, COMP, FINISH。

※ 與軟體不同，2.4 以 2 進制表示為無限循環，因此在硬體運算時，我們會把他先乘上 10，達到準確運算。

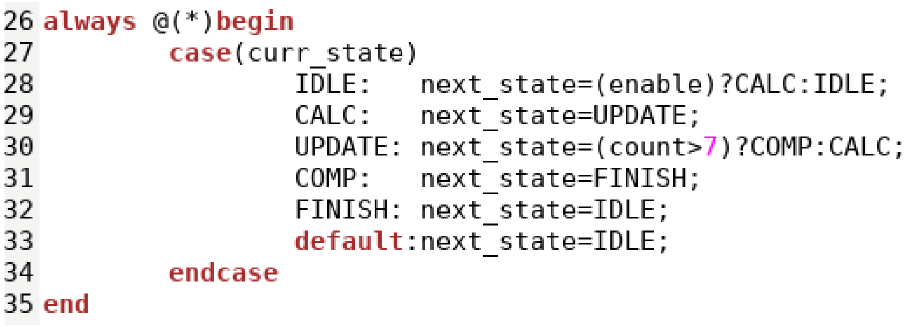
IDLE: 初始化參數和等待 enable 訊號，如果 enable=1，則開始進行 successive approximation。

CALC: 更新 count，判斷 x 產生的 y 和 target 大小，對 x 值更新。

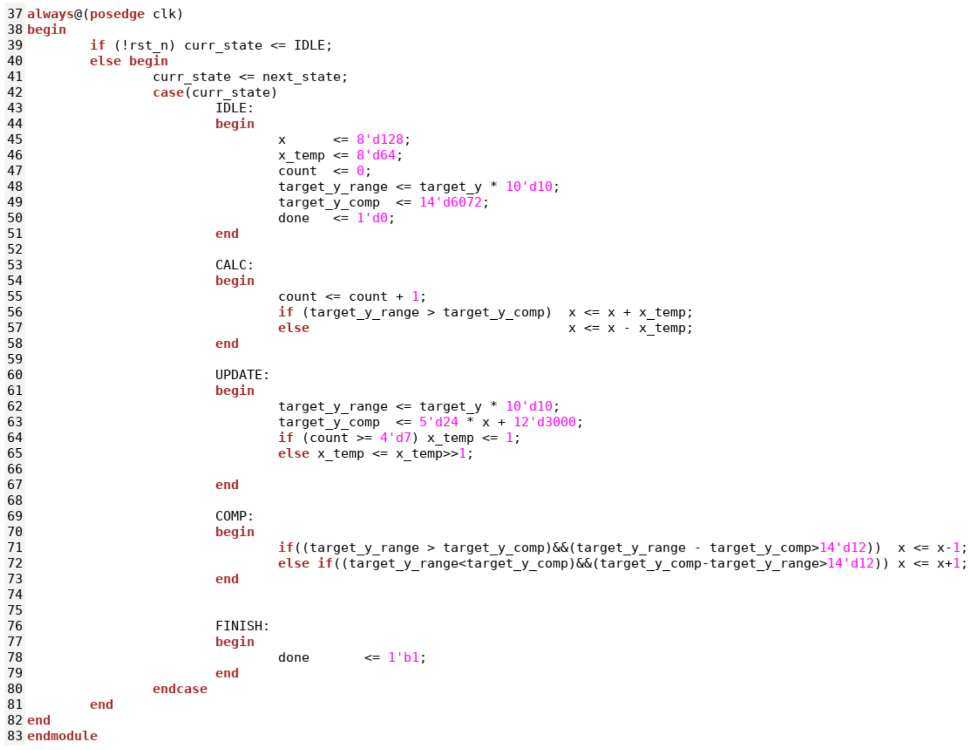
UPDATE: 更新 target_y 和 x 產生的 y，設置 x_temp 存 shift 數值，如 v2 軟體程式碼，在 count<7 時，跳回 CALC，結束後再跳到 COMP。

COMP: 比較最後 1 個 bits，判斷 x 產生的 y 最接近 target 的大小。

FINISH: 完成 successive approximation，設置 done=1。



```
always @(*)begin
        case(curr_state)
                IDLE:   next_state=(enable)?CALC:IDLE;
                CALC:   next_state=UPDATE;
                UPDATE: next_state=(count>7)?COMP:CALC;
                COMP:   next_state=FINISH;
                FINISH: next_state=IDLE;
                default:next_state=IDLE;
        endcase
end
```



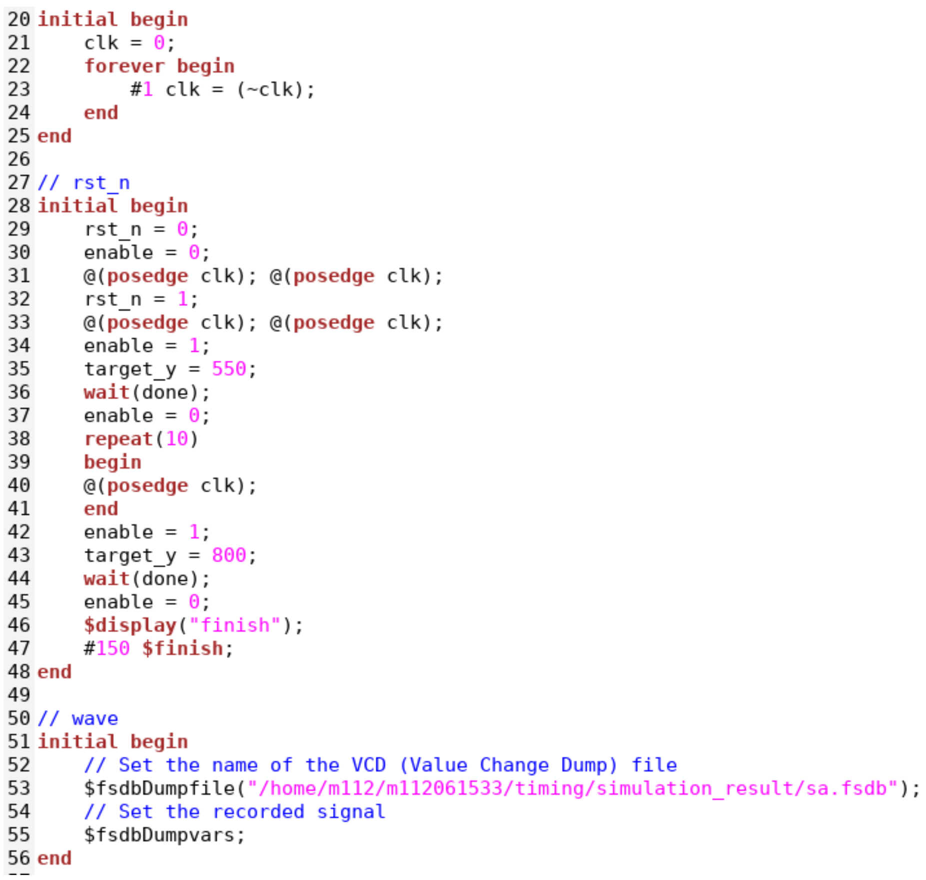
```
always@(posedge clk)
begin
        if (!rst_n) curr_state <= IDLE;
        else begin
                curr_state <= next_state;
                case(curr_state)
                        IDLE:
                        begin
                                x      <= 8'd128;
                                x_temp <= 8'd64;
                                count  <= 0;
                                target_y_range <= target_y * 10'd10;
                                target_y_comp  <= 14'd6072;
                                done   <= 1'd0;
                        end

                        CALC:
                        begin
                                count <= count + 1;
                                if (target_y_range > target_y_comp)  x <= x + x_temp;
                                else                                 x <= x - x_temp;
                        end

                        UPDATE:
                        begin
                                target_y_range <= target_y * 10'd10;
                                target_y_comp  <= 5'd24 * x + 12'd3000;
                                if (count >= 4'd7) x_temp <= 1;
                                else x_temp <= x_temp>>1;

                        end

                        COMP:
                        begin
                                if((target_y_range > target_y_comp)&&(target_y_range - target_y_comp>14'd12))  x <= x-1;
                                else if((target_y_range<target_y_comp)&&(target_y_comp-target_y_range>14'd12)) x <= x+1;
                        end


                        FINISH:
                        begin
                                done       <= 1'b1;
                        end
                endcase
        end
end
endmodule
```

圖五 sa verilog code

2. Testbench:

在 testbench 中，產生週期為 2ns 的 clock 訊號和初始化 rst_n 和

enable，在 2 個 clk 後，將 rst_n 設為 1，在兩個 clk 後，enable 設為 1 和給定 target_y，代表開始進行操作，等到接收到 done 時候，在 delay 10 個 clk 在開始下一筆操作，在 delay 一些時間後，完成這次的 testbench，另外和產生波型檔。

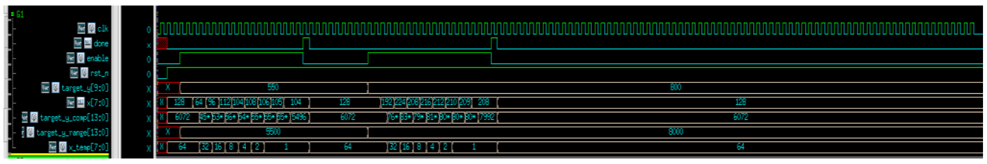


```verilog
initial begin
    clk = 0;
    forever begin
        #1 clk = (~clk);
    end
end

// rst_n
initial begin
    rst_n = 0;
    enable = 0;
    @(posedge clk); @(posedge clk);
    rst_n = 1;
    @(posedge clk); @(posedge clk);
    enable = 1;
    target_y = 550;
    wait(done);
    enable = 0;
    repeat(10)
    begin
    @(posedge clk);
    end
    enable = 1;
    target_y = 800;
    wait(done);
    enable = 0;
    $display("finish");
    #150 $finish;
end

// wave
initial begin
    // Set the name of the VCD (Value Change Dump) file
    $fsdbDumpfile("/home/m112/m112061533/timing/simulation_result/sa.fsdb");
    // Set the recorded signal
    $fsdbDumpvars;
end
```

圖六 sa_tb code

3. Waveform

波型顯示上述結果和在 target value 550 時 x 為 104，target value 800 時 x 為 208，符合軟體結果。



圖七 sa_waveform

c. Goal :

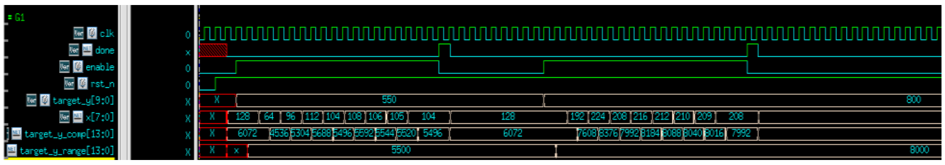
Use a synthesis script to convert your RTL code into a gate-level netlist. Perform gate-level simulation to verify the gate-level netlist again, to ensure that its behavior is identical to your previous RTL code.

Execute:

1. Description:

   在執行 Synthesis 時，發現 clock 不能調太快，否則會遇到 slack 為負的問題，因為有些運算時間太長，因此調整 clock，並驗證是否符合 slack 標準，和觀察 Power 和 Area。

2. Waveform



圖八 sa_syn waveform

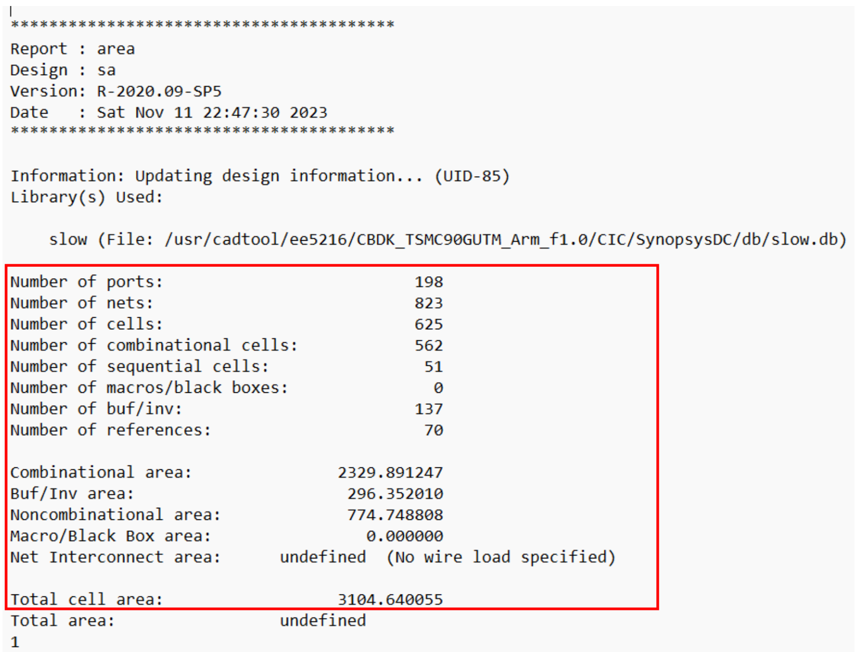
d. Goal:

Report the final gate count, the maximum operating speed (in MHz), and the estimated power consumption in (mW) of your design using Design Compiler

Execute:

1. Gate count :

   下圖顯示了本次所使用的 cell 和其面積，可以看到 total cell area 為 3104.640055um^2，根據講義給的 nand2 area:2.8224um^2，因此算出的 final gate count=3104.640055/2.8224=1100.000019。



```
****************************************
Report : area
Design : sa
Version: R-2020.09-SP5
Date   : Sat Nov 11 22:47:30 2023
****************************************

Information: Updating design information... (UID-85)
Library(s) Used:

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

Number of ports:                          198
Number of nets:                           823
Number of cells:                          625
Number of combinational cells:            562
Number of sequential cells:                51
Number of macros/black boxes:               0
Number of buf/inv:                        137
Number of references:                      70

Combinational area:               2329.891247
Buf/Inv area:                      296.352010
Noncombinational area:             774.748808
Macro/Black Box area:                0.000000
Net Interconnect area:      undefined  (No wire load specified)

Total cell area:                  3104.640055
Total area:                 undefined
1
```

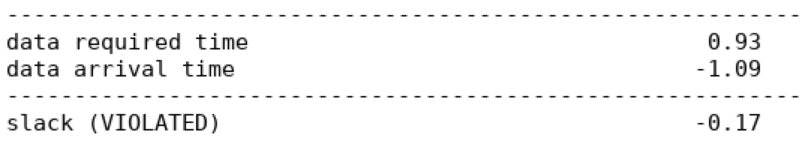
圖九 area report

2. Maximum operating speed:

   下圖可以看到本次使用的 clk 週期為 2ns，操作頻率約為

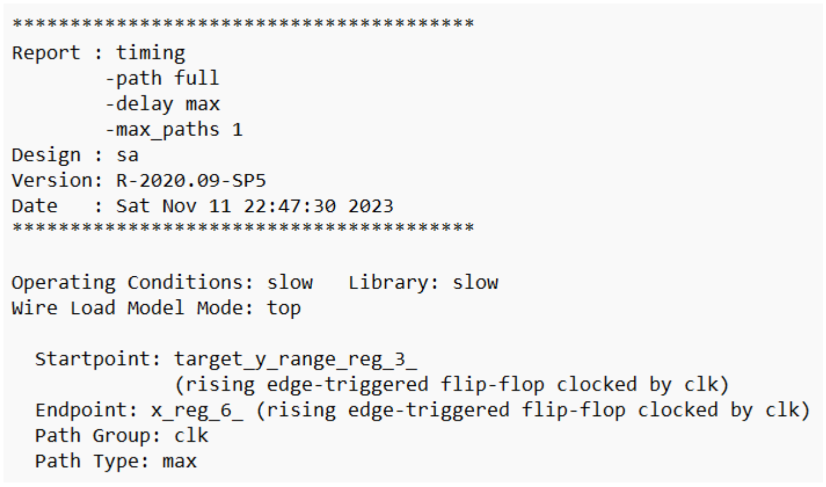
500MHz，在此操作頻率下 slack 還有調整範圍。

因此調整 clk 週期為 1ns，使他 slack 產生負數，因此來算他的最大頻率，最大操作頻率約為 855Mhz。

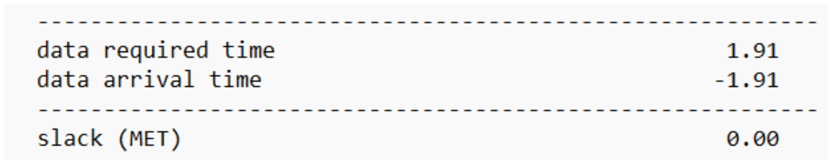


```
-----------------------------------------------------------
data required time                                     0.93
data arrival time                                     -1.09
-----------------------------------------------------------
slack (VIOLATED)                                      -0.17
```

圖十 clk 為 1ns timing report



```
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : sa
Version: R-2020.09-SP5
Date   : Sat Nov 11 22:47:30 2023
****************************************

Operating Conditions: slow   Library: slow
Wire Load Model Mode: top

  Startpoint: target_y_range_reg_3_
              (rising edge-triggered flip-flop clocked by clk)
  Endpoint: x_reg_6_ (rising edge-triggered flip-flop clocked by clk)
  Path Group: clk
  Path Type: max
```

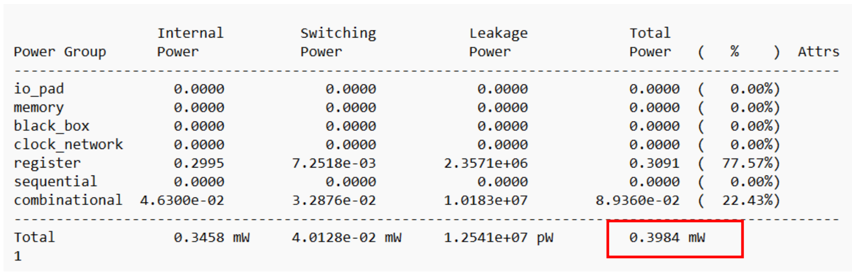


```
  -----------------------------------------------------------
  data required time                                     1.91
  data arrival time                                     -1.91
  -----------------------------------------------------------
  slack (MET)                                            0.00
```

圖十一 clk 為 2ns timing report

3. Power consumption:

   預估的 power consumption 約為 0.3984mW。



```
                 Internal         Switching          Leakage            Total
Power Group      Power            Power              Power              Power   (   %    )  Attrs
--------------------------------------------------------------------------------------------------
io_pad             0.0000           0.0000            0.0000             0.0000  (   0.00%)
memory             0.0000           0.0000            0.0000             0.0000  (   0.00%)
black_box          0.0000           0.0000            0.0000             0.0000  (   0.00%)
clock_network      0.0000           0.0000            0.0000             0.0000  (   0.00%)
register           0.2995       7.2518e-03        2.3571e+06             0.3091  (  77.57%)
sequential         0.0000           0.0000            0.0000             0.0000  (   0.00%)
combinational  4.6300e-02       3.2876e-02        1.0183e+07         8.9360e-02  (  22.43%)
--------------------------------------------------------------------------------------------------
Total              0.3458 mW    4.0128e-02 mW     1.2541e+07 pW          0.3984 mW
1
```

圖十二 power report

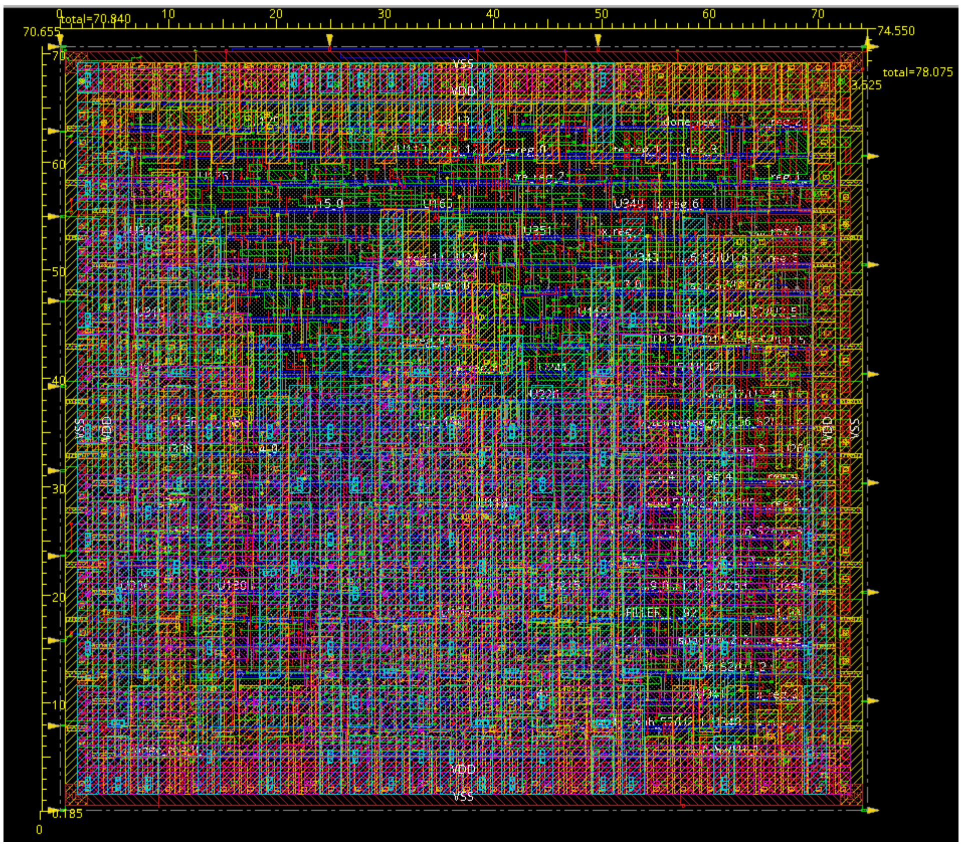
e. Goal:

   Generate the layout by running some Automatic Placement and Routing Tool (e.g., SoC Encounter or the like). Report the size of the layout. Perform post-layout simulation or analysis and report again the maximum operating speed and power consumption. Compare how these results are different from those in pre-layout analysis。

Execute:

1. Size of Layout:

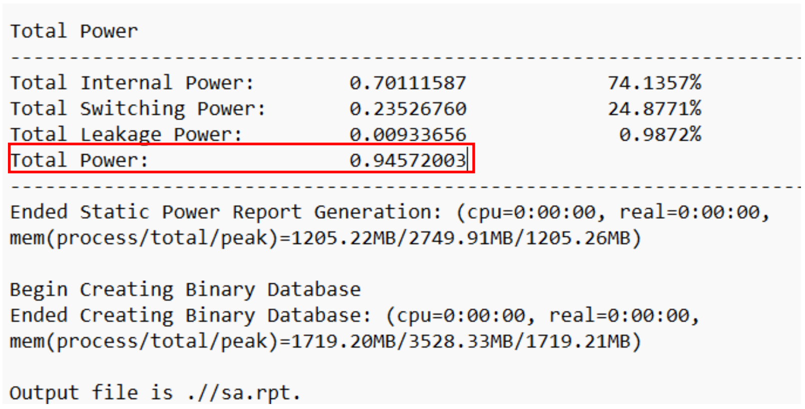
   從下圖可以看到，Layout 面積約為 74.550*70.655 um^2



圖十三 Layout

2. Power Consumption:

   Power consumption 約為 0.946mW。

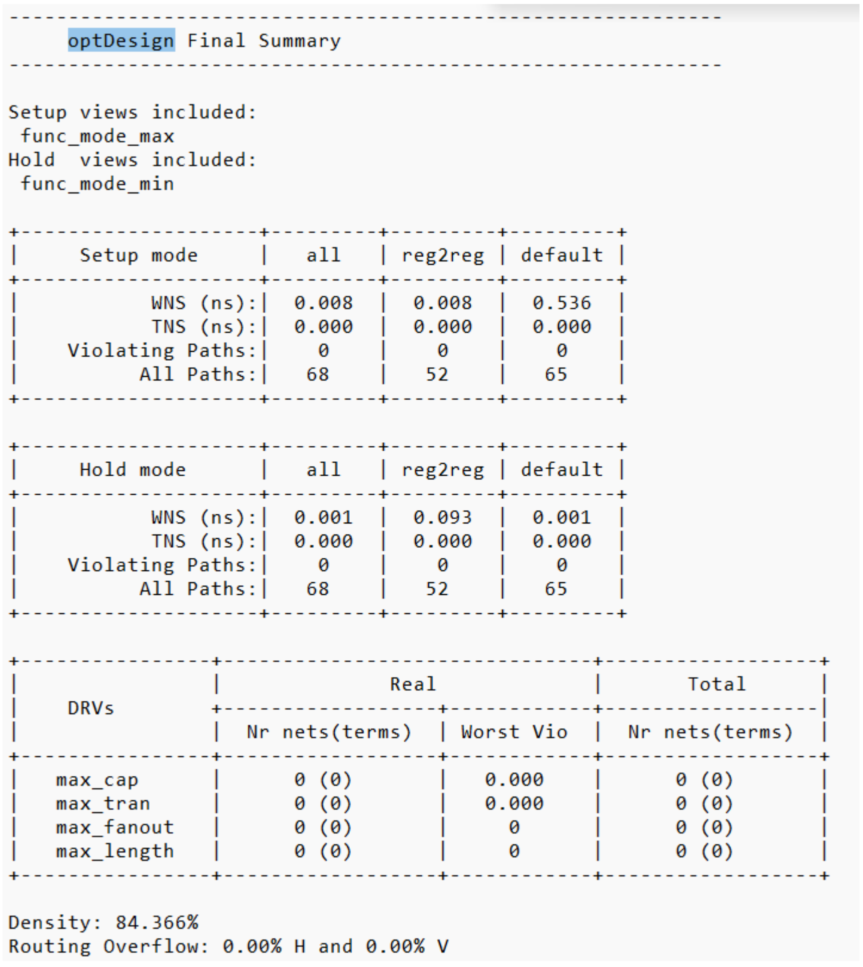


```
Total Power
-----------------------------------------------------------------------
Total Internal Power:          0.70111587          74.1357%
Total Switching Power:         0.23526760          24.8771%
Total Leakage Power:           0.00933656           0.9872%
Total Power:                   0.94572003
-----------------------------------------------------------------------
Ended Static Power Report Generation: (cpu=0:00:00, real=0:00:00,
mem(process/total/peak)=1205.22MB/2749.91MB/1205.26MB)

Begin Creating Binary Database
Ended Creating Binary Database: (cpu=0:00:00, real=0:00:00,
mem(process/total/peak)=1719.20MB/3528.33MB/1719.21MB)

Output file is .//sa.rpt.
```

圖十四 Power Consumption

3. maximum operating speed

本次 APR 操作頻率與 syn 相同，clk 週期為 2ns，操作頻率約為 500MHz，經過 testbench 測試，可以發現 clk 週期為 1.8ns，操作頻率約為 556Mhz。

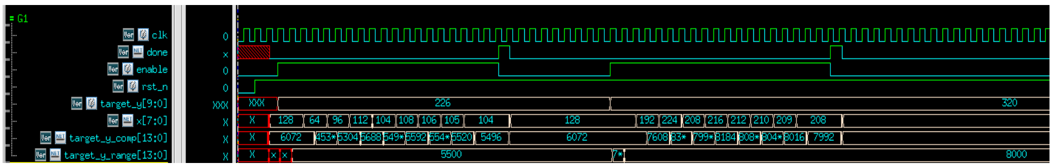


```
------------------------------------------------------------
     optDesign Final Summary
------------------------------------------------------------

Setup views included:
 func_mode_max
Hold  views included:
 func_mode_min

+--------------------+---------+---------+---------+
|     Setup mode     |   all   | reg2reg | default |
+--------------------+---------+---------+---------+
|           WNS (ns):|  0.008  |  0.008  |  0.536  |
|           TNS (ns):|  0.000  |  0.000  |  0.000  |
|    Violating Paths:|    0    |    0    |    0    |
|          All Paths:|   68    |   52    |   65    |
+--------------------+---------+---------+---------+

+--------------------+---------+---------+---------+
|     Hold mode      |   all   | reg2reg | default |
+--------------------+---------+---------+---------+
|           WNS (ns):|  0.001  |  0.093  |  0.001  |
|           TNS (ns):|  0.000  |  0.000  |  0.000  |
|    Violating Paths:|    0    |    0    |    0    |
|          All Paths:|   68    |   52    |   65    |
+--------------------+---------+---------+---------+

+----------------+-------------------------------+------------------+
|                |              Real             |       Total      |
|    DRVs        +------------------+------------+------------------|
|                |  Nr nets(terms)  | Worst Vio  |  Nr nets(terms)  |
+----------------+------------------+------------+------------------+
|   max_cap      |      0 (0)       |   0.000    |      0 (0)       |
|   max_tran     |      0 (0)       |   0.000    |      0 (0)       |
|   max_fanout   |      0 (0)       |     0      |      0 (0)       |
|   max_length   |      0 (0)       |     0      |      0 (0)       |
+----------------+------------------+------------+------------------+

Density: 84.366%
Routing Overflow: 0.00% H and 0.00% V
```

圖十五 timing report

4. Waveform:

波型顯示上述結果和在 target value 550 時 x 為 104，target value 800 時 x 為 208，符合軟體結果。



圖十六 sa_apr waveform

5. Comments:

可以看到在 APR 之後，產生的功耗明顯增加，且在速度上也有所改變，因此 synthesis 完如何 APR 是非常需要謹慎且思考的問題，因為需要經過很多步驟像是 FloorPlan, Placement, CTS, Routing 等等。