Triangle Rendering Engine

EE 465 Final Project

10 December 2015

Cory Snooks
Aaron Pedersen

Contents

Introduction

This project consisted of writing the Verilog code, performing RTL synthesized, and completing layout for a triangle rendering engine (TRE). The TRE took three sets of 3-bit x, y coordinate inputs and returned all of the points contained within the triangle. The TRE had to be functional for the base cases of x1 = x3 and y1 < y2 < y3. The input triangles could be left or right facing and were limited to a minimum coordinate value of zero and a maximum coordinate value of seven. A test bench was provided for the project that checked all of the outputs for the both facing triangles. The Verilog code was synthesized after the functionality was deemed correct to ensure that the code was physically possible. After solving some multiple driver errors, a full layout of the circuit was completed.

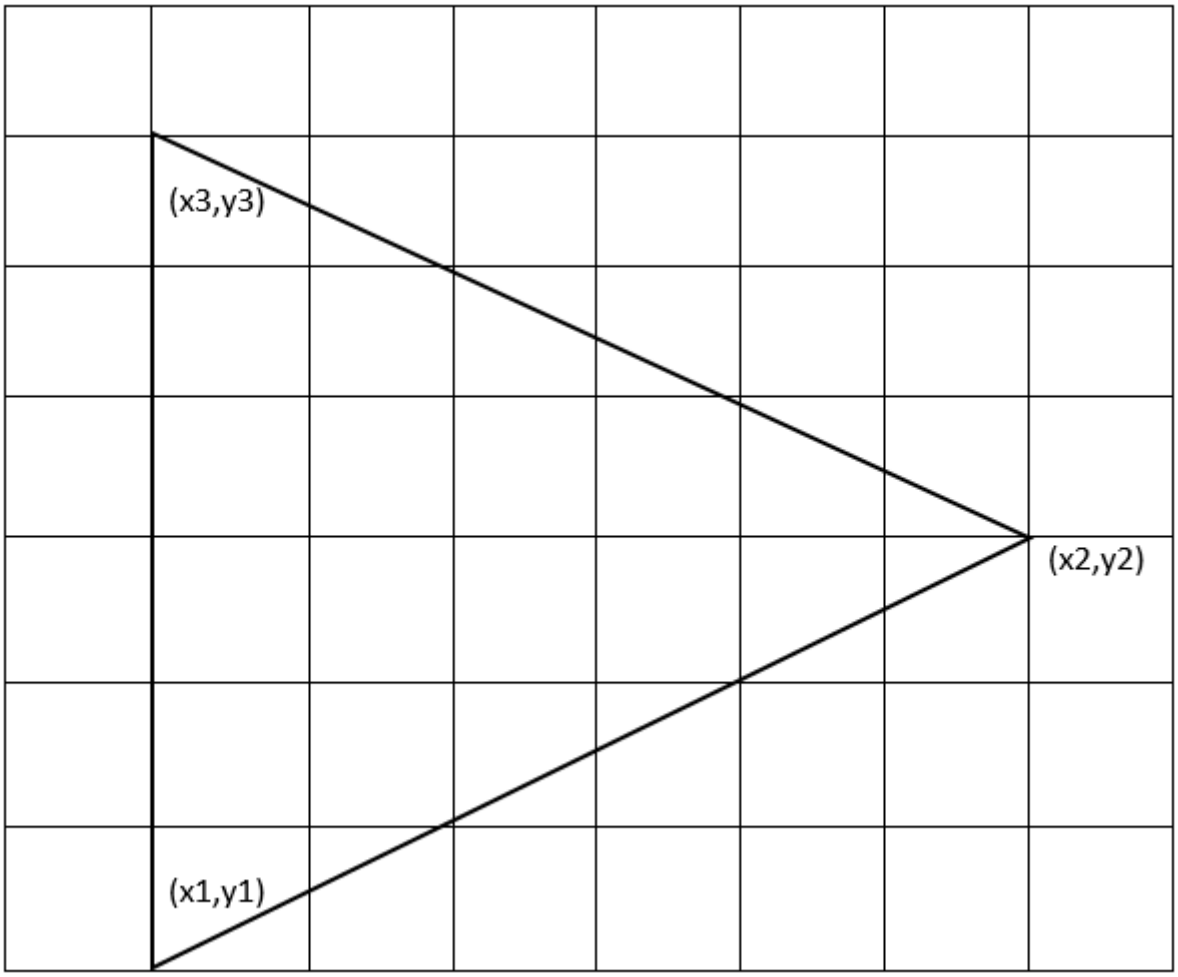


Figure 1: A sample triangle is shown above.

Design Methodology

From the hint given in the project specifications, the equation to determine if a point on a line is as follows:

$$\frac{x - x_1}{y - y_1} - \frac{x_2 - x_1}{y_2 - y_1} = 0$$

If the point is to the right hand side of the line, this equation is true:

$$\frac{x - x_1}{y - y_1} - \frac{x_2 - x_1}{y_2 - y_1} > 0$$

If the point is to the left hand side of the line, this equation is true:

$$\frac{x - x_1}{y - y_1} - \frac{x_2 - x_1}{y_2 - y_1} < 0$$

For our design we will use a modification of these equations in order to accurately determine if a coordinate was located in the triangle. The direction of the triangle had to be found by comparing the value of x1 to x2 in order to determine which metric to compare the equation to. If x1 < x2 then the triangle is right facing and if x1 > x2 then the triangle is left facing.

To compute whether a point of interest is inside the triangle, we will be using the following assumptions:

- $y_1 = y_3$
- $y_1 1 < y_2 < y_3$

Manipulating the equation $\frac{x-x_1}{y-y_1} - \frac{x_2-x_1}{y_2-y_1} = 0$, we can derive the equation:

$\frac{x-x_1}{y-y_1} = \frac{x_2-x_1}{y_2-y_1} \quad \rightarrow \quad (x - x_1)(y_2 - y_1) - (x_2 - x_1)(y - y_1)$

To do all these computations in one clock cycle, we would need 2 multipliers and 5 adders. To implement sharing of these blocks, we will do these computations in multiple clock cycles. For our design, we used the following variables to represent parts of the equation:

$(x - x_1)\ or\ (x - x_3) = A$
$(y - y_1)\ or\ (y - y_3) = B$
$(x_2 - x_1) = C0$
$(y_2 - y_1) = D0$
$(x_2 - x_3) = C1$
$(y_2 - y_3) = D1$
$(x - x_1)(y_2 - y_1) = AD$
$(x_2 - x_1)(y - y_1) = \mathrm{BC}$
$(x - x_1)(y_2 - y_1) - (x_2 - x_1)(y - y_1) = RLO$

To generate the desired output, we must start by analyzing the (x1, y1) coordinate and outputting that. Then we must increment an x variable to analyze new (x,y) coordinates. When the row is finished outputting all valid points, we must then increment y and repeat the process until all points are checked.

Since y does not change when scanning one row, we will only need to compute B once for each row. We have implemented this in the code.

Point is to the right or left of a slanted line

First, we need to know if the line we want to compare a coordinate to is the upper line of the triangle or the lower line of the triangle. To do this, we simply compare the y value of the coordinate of interest to the y2 value that was input. If the y value of interest is greater than the y2 value, then the line we want to compare to is the upper line. If it is equal to y2, we still compare it to the bottom line.

When comparing the current coordinates against the bottom line, x-x1 was stored into register A, y-y1 was stored into register B, x2-x1 was stored into register C0 and y2-y1 was stored into register D0. When comparing the current coordinates against the top line, x-x3 was stored into register A, y-y3 was stored into register B, x2-x3 was stored into register C1 and y2-y3 was stored into register D1.
With these three things known (if it is an upper line, if the triangle is right facing or left facing, and the RLO result), we can tell if the point is valid or not.

For bottom lines, if RLO is positive, the point is right of the line.
For top lines, if RLO is negative, the point is to the right of the line.
If RLO is zero, the point is on the line.

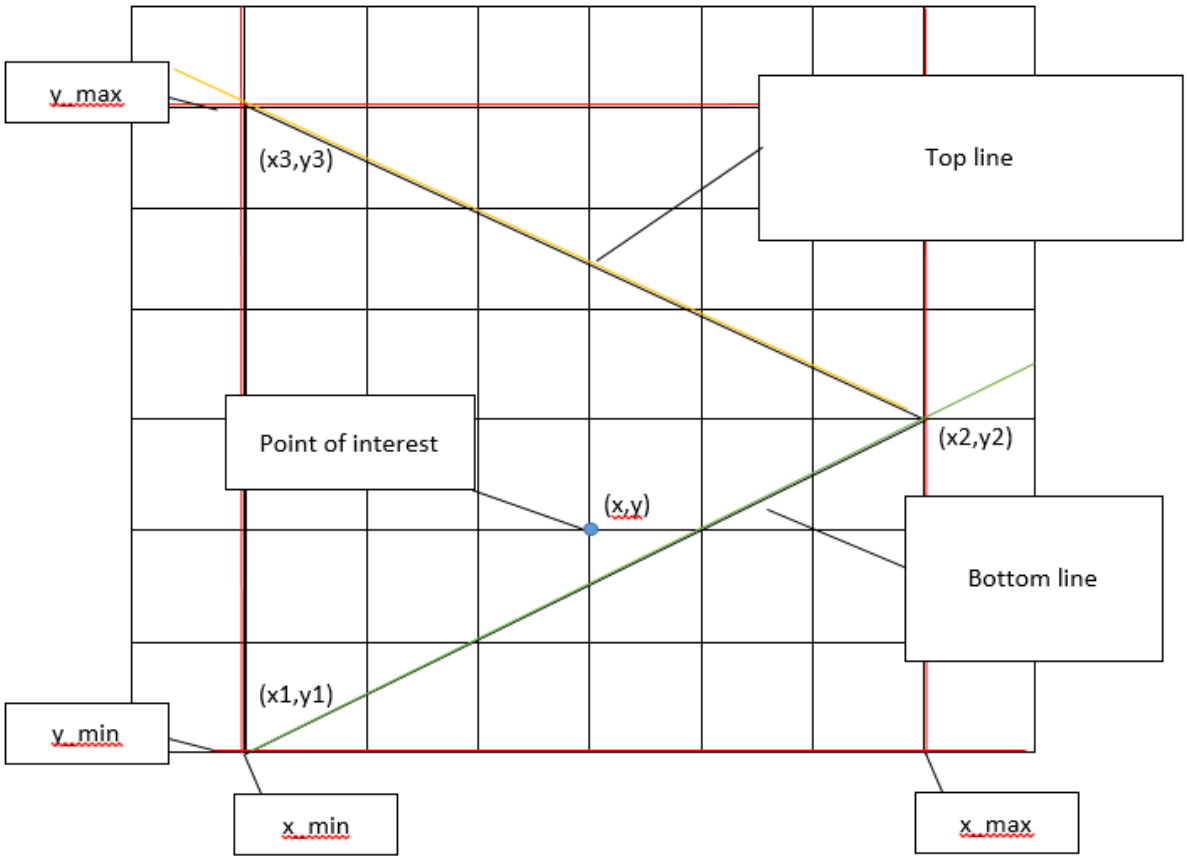


Figure 2: The diagram above shows how the Verilog broke down an input triangle and found coordinates inside of the triangle.

Throughput

The throughput for this design methodology is quite complicated to report. For every new row, B and BC values are calculated only once and there are different sizes/layouts of right and left-facing triangles. For right facing triangles there is one extra point analyzed which is not in the triangle boundary, whereas for left facing triangles all points inside a rectangle must be analyzed. The rectangle coordinates are $(x2, y1), (x2, y3), (x1, y1), (x3, y3)$.

Case 1 - Left-facing triangle:

In this case, we waste clock cycles because we must start at x_min. We could generate an algorithm that calculates the first valid x coordinate, but that would be time consuming and make the design more complicated and area-consuming.

$$(x_1, y_1) = (6,1)$$
$$(x_2, y_2) = (1,4)$$
$$(x_3, y_3) = (6,5)$$

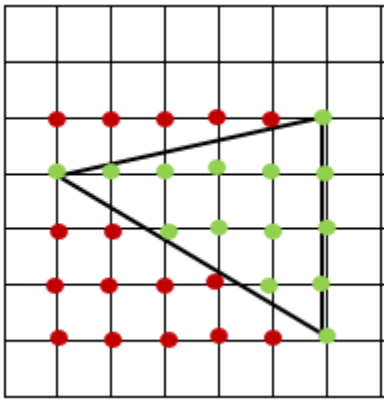


Figure 3: Example Left-facing Triangle

$$(x_1 - x_2 + 1) \times (y_3 - y_1 + 1) \times 5 + (\, y_3 - y_1 + 1) \times 2 + 4$$

Case 2 - Right-facing triangle:

In this case, we can detect a "negative edge" of valid outputs and decide to move on to the next row in order to save some clock cycles.

$$(x_1, y_1) = (1,1)$$
$$(x_2, y_2) = (6,4)$$
$$(x_3, y_3) = (1,5)$$

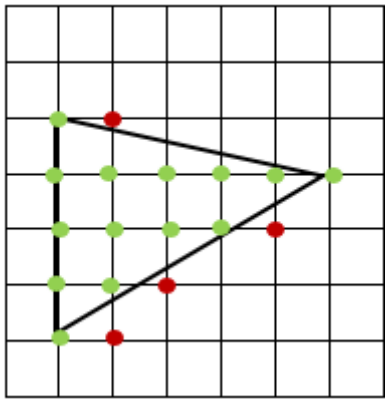


Figure 4: Example Right-facing Triangle

$$(\#\ of\ points\ in\ the\ triangle) \times 5 + (y_3 - y_1 + 1) \times 2 + (y_3 - y_1) \times 5 + 4$$

Device Sharing

Here is a top-level concept diagram of the sharing functionality of the design:

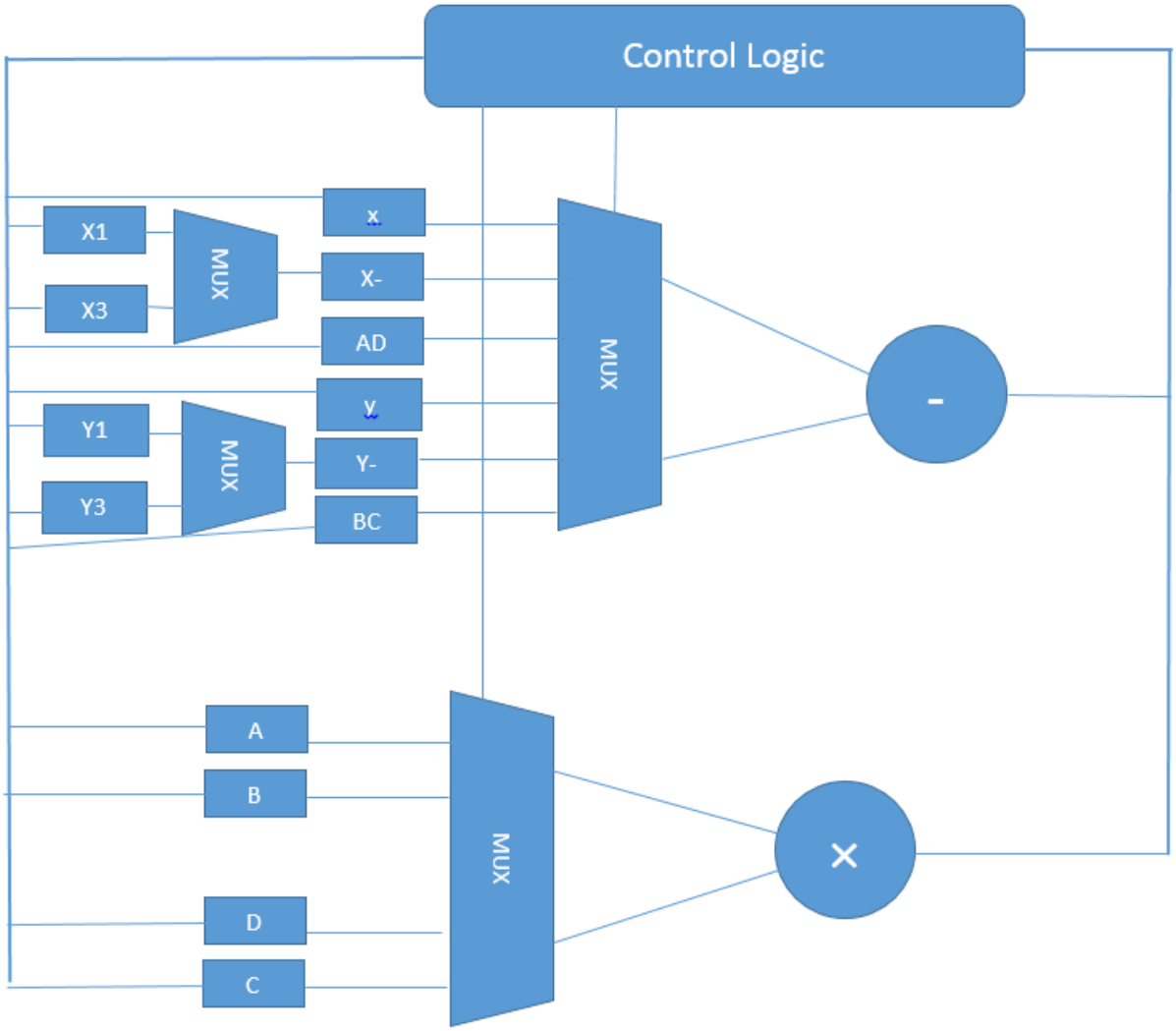


Figure 5: Device sharing block diagram

The thicker line is a bus which selects signals from the outputs of the multiplier or adder depending on which state the system is in. This is a rough diagram to show the basic idea of sharing the multiplier and adder.

Verilog Code

```
`timescale 100ps/10ps
module triangle(clk, reset, nt, xi, yi, busy, po, xo, yo);
input clk, reset, nt;
input [2:0] xi, yi;
output busy, po;
output [2:0] xo, yo;

wire clk, reset, nt;
wire [2:0] xi, yi;
reg  [2:0] xo, yo, x_min, x_max, y_min, y_max, x, y;
reg  [2:0] xi_ff[2:0], yi_ff[2:0];
reg busy, good, mul_en, sub_en, check, rst_int, inc_y, inc_x; //internal
reset
reg po, po_, on_line, right_line , right_triangle, top_line; //right_triangle
= 1 means right, 0 means left
reg signed [6:0] sub_ans , sub_op0, sub_op1, mul_op0, mul_op1;
reg signed [6:0] mul_ans, A, B, C0, D0, C1, D1, AD, BC, RLO;
reg  [3:0] control, control_1;


always @ (posedge clk)begin
  if(reset || rst_int) begin
    xi_ff[0] <= 0;
    xi_ff[1] <= 0;
    xi_ff[2] <= 0;
    yi_ff[0] <= 0;
    yi_ff[1] <= 0;
    yi_ff[2] <= 0;
    x_min <= 7;
    x_max <= 0;
    y_min <= 7;
    y_max <= 0;
    top_line <= 0;
    A <= 0;
    B <= 0;
    C0 <= 0;
    C1 <= 0;
    D0 <= 0;
    D1 <= 0;
    AD <= 0;
    BC <= 0;
    RLO <= 0;
    control <= 0;
    check <= 0;
    xo <= 0;
    yo <= 0;
    busy <= 0;
    rst_int <= 0;
    inc_y <= 0;
    inc_x <= 0;

  end

    else begin

        if(control)begin
            //control <= control_1;
            if(control < 13)control <= control + 1;
            else if(x == x_min) control <= 7;
```

```
            else control <= 9;
        end
        else if(nt)begin
            xi_ff[0] <= xi;
            yi_ff[0] <= yi;
            control <= 1;
        end


        if(control < 6)begin
            if(xi > x_max) x_max <= xi;
            if(xi < x_min) x_min <= xi;
            if(yi > y_max) y_max <= yi;
            if(yi < y_min) y_min <= yi;
        end
        else begin
            if(y > yi_ff[1])top_line <= 1;
            else top_line <= 0;
        end

        case(control)

            1: begin
                xi_ff[1] <= xi;
                yi_ff[1] <= yi;
                busy <= 1;
            end

            2: begin
                xi_ff[2] <= xi;
                yi_ff[2] <= yi;
            end
            3: begin
                C0  <= sub_ans;
            end

            4: begin
                D0 <= sub_ans;
            end

            5: begin
                C1 <= sub_ans;
                inc_y <= 1;
            end

            6: begin
                D1 <= sub_ans;
                inc_y <= 0;
            end

            7: begin    //B = Y-Y1   B should be calculated first because it
remains the same for all values on this row
                        //it will return here if there is a new row with a
new y value
                if(check)check <= 0;
                B <=  sub_ans;
                if(inc_y)inc_y <= 0;    //resets inc_y back to zero to
prevent extra incrementing
                if(inc_x)inc_x <= 0;    //resets inc_x back to zero to
prevent extra incrementing
            end
```

```
            8:begin     //BC = B*C   BC should be calculated first because it
remains the same for all values on this row
                BC <= mul_ans;
            end

            9: begin    //A = X-X1   It will return here if it is just a new
value of x
                if(check)check <= 0;
                A <= sub_ans;
                if(inc_y)inc_y <= 0;    //resets inc_y back to zero to
prevent extra incrementing
                if(inc_x)inc_x <= 0;    //resets inc_x back to zero to
prevent extra incrementing
            end

            10: begin
                AD <=  mul_ans;
            end

            11: begin   //B = ans
                RLO <= sub_ans;
            end

            12: begin
                check <= 1; //check should be high for 2 clock cycles to
pulse po and check for valid
                xo <= x;
                yo <= y;
            end

            13: begin
                if(yo >= y_max)begin
                    if(right_triangle ^ (xo == x_max)) rst_int <= 1;
                    else if(xo == x_max)    rst_int <= 1;
                end
                if((x == x_max) || ~good && right_triangle && ~inc_y) inc_y
<= 1;
                if((po_ || ~right_triangle) && ~inc_y) inc_x <= 1;
            end
        endcase
    end//else begin
end//always @ (posedge clk)begin

always @(negedge clk) begin
    if(inc_y)begin
        if((control == 6)) y <= y_min;
        else y <= y + 1;
        x <= x_min;
    end
    else if(inc_x) x <= x + 1;

    if(check)begin

        if(good)begin
            if(po)po<=0;
            else begin
                po <= 1;
                po_ <= 1; // on the positive edge of po, po_ is set to 1
            end
        end//if(good)begin
```

```
        else begin
            po_ <= 0; // on the negative edge of po, po_ should be set to 0
        end//else(~good)
    end //if(check)begin

end //always @(negedge clk) begin


always@(posedge check)begin
    if(((x == xi_ff[0]) && (y==yi_ff[0]))  ||  ((x == xi_ff[1]) && (y ==
yi_ff[1]))  ||  ((x == xi_ff[2]) && (y == yi_ff[2]))) good <= 1;
    else if((right_triangle ~^ right_line) || on_line) good <= 1;
    else good <= 0;
end


always @(*) begin //multiplier and adder modules
    if(sub_en) sub_ans = sub_op0 - sub_op1;//multiplier block
    if(mul_en) mul_ans = mul_op0 * mul_op1;//adder block
end

always @(*) begin
    if(xi_ff[1]>xi_ff[0]) right_triangle = 1;
    else right_triangle = 0;
end

always @(*) begin
    if((RLO == 0) || (x == xi_ff[0]))begin
        on_line = 1;
        right_line = 0;
    end
    else begin
        on_line = 0;
        if(~top_line)right_line = RLO[6];
        else right_line = ~RLO[6];
    end
end

always @(*) begin   //control logic
    //if(control < 13) //determines next control
    //  control_1 = control + 1;
    //else control_1 = 7;

    case(control)
        0: begin

        end

        1: begin

        end

        3: begin    //C0 = X2-X1
            sub_en = 1;
            sub_op0 = xi_ff[1];
            sub_op1 = xi_ff[0];
        end

        4: begin    //D0 = Y2-Y1
            sub_en = 1;
            sub_op0 = yi_ff[1];
```

```
        sub_op1 = yi_ff[0];
    end

    5:begin     //C1 = x2-x3

        sub_en = 1;
        sub_op0 = xi_ff[1];
        sub_op1 = xi_ff[2];
    end

    6: begin    //D1 = Y2-Y3

        sub_en = 1;
        sub_op0 = yi_ff[1];
        sub_op1 = yi_ff[2];
    end

    7: begin    //B = Y-Y1
        sub_en = 1;
        mul_en = 0;
        sub_op0 = y;
        if(~top_line)sub_op1 = yi_ff[0];
        else sub_op1 = yi_ff[2];
    end

    8: begin    //BC = B*C
        sub_en = 0;
        mul_en = 1;
        mul_op0 = B;
        if(~top_line)mul_op1 = C0;
        else mul_op1 = C1;
    end

    9: begin    //A = X-X1
        sub_en = 1;
        mul_en = 0;
        sub_op0 = x;
        if(~top_line)sub_op1 = xi_ff[0];
        else sub_op1 = xi_ff[2];
    end

    10: begin   //AD = A*D
        sub_en = 0;
        mul_en = 1;
        mul_op0 = A;
        if(~top_line)mul_op1 = D0;
        else mul_op1 = D1;
    end

    11: begin   //RLO = AB-CD
        mul_en = 0;
        sub_en = 1;
        sub_op0 = AD;
        sub_op1 = BC;
    end

    12: begin
        sub_en = 0;
        mul_en = 0;
    end
```

```
    endcase
end
endmodule
```

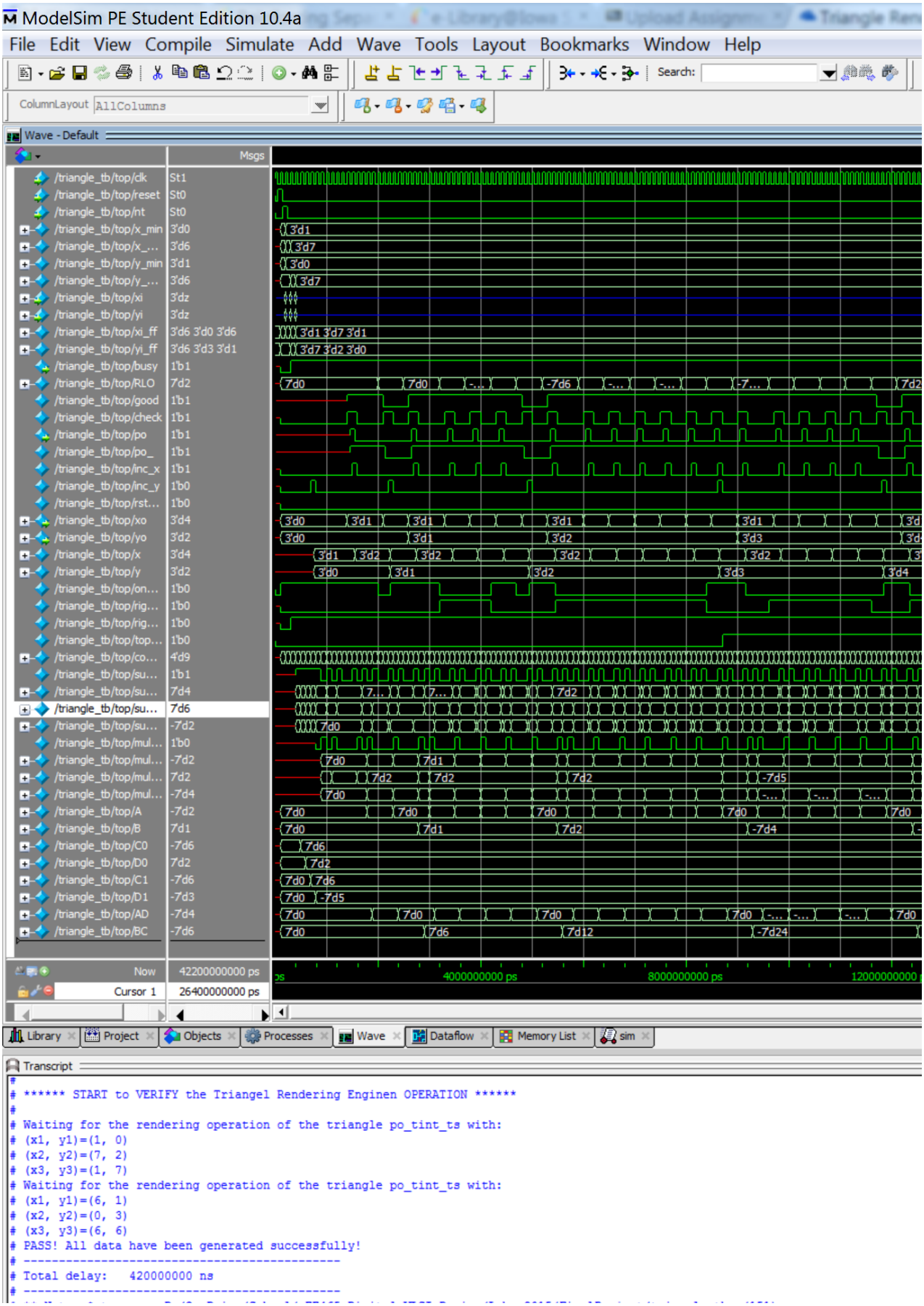


Figure 6: The ModelSim outputs for a right facing triangle are shown above.

RTL Synthesis

Cadence RTL Synthesizer was used to perform RTL (Register Transfer Level) synthesis. The first attempts at RTL synthesis were unsuccessful and gave multiple driver warnings. Those warnings came from setting register values under different always blocks in the Verilog code. When the synthesizer sees multiple drivers, it just assigns the register a value. Sometimes it assigns the register to be both high and low at the same time, which means that the register output is connected to both power and ground. Those problems were fixed by re-writing the Verilog code so that values were only being set under one always block.

The timing report had to be verified for a positive slack time after running synthesis. If the slack time was negative in synthesis, then it would definitely be negative in layout. A negative slack time means that a signal arrives at an input later than it needs to in order for the functionality to remain the same. A positive slack time means that the signal arrived at an input early. The goal is to get the slack time as close to zero as possible to reduce wasted power, area, and clock cycles.

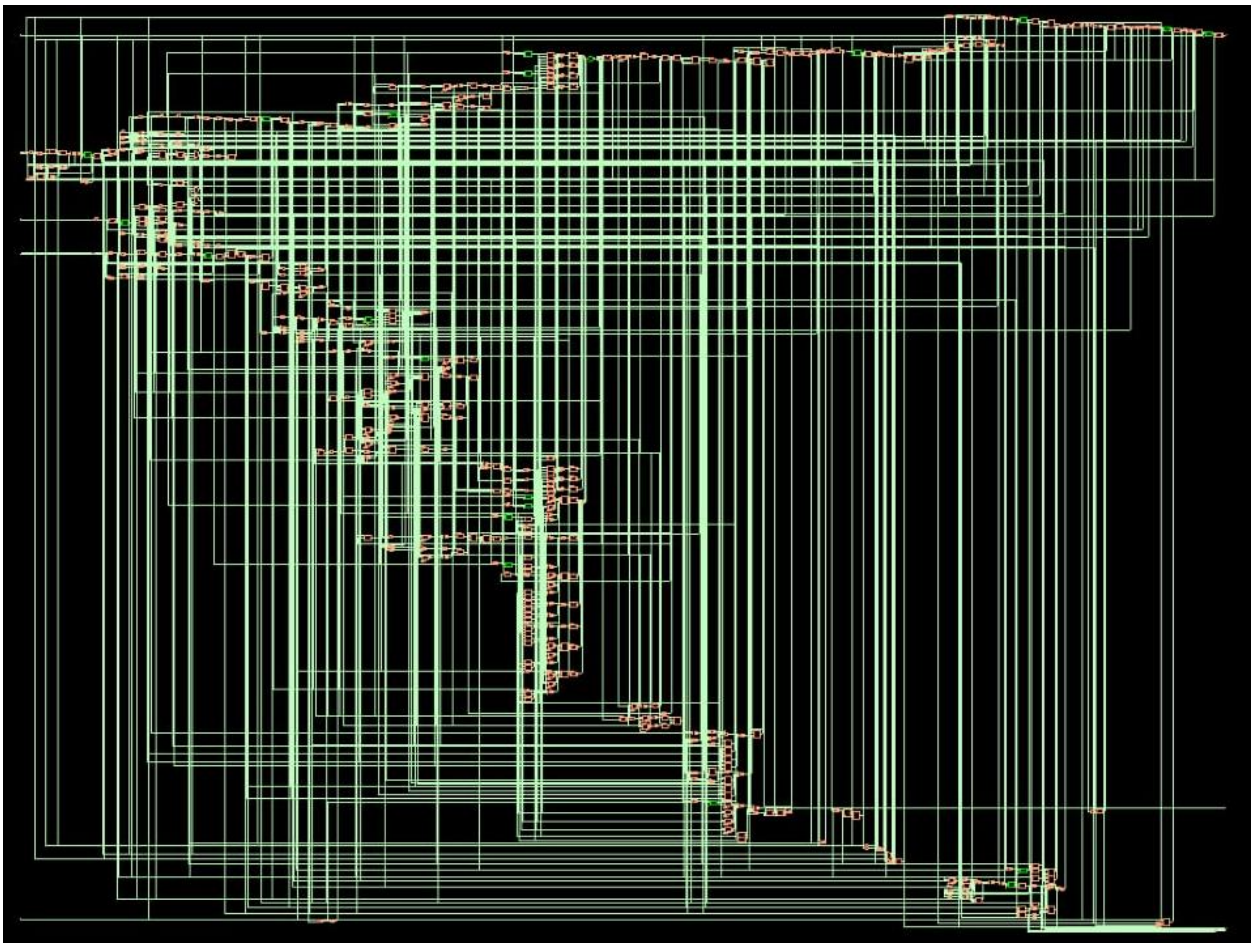


Figure 7: The image above shows the completed RTL synthesized Triangle Rendering Engine.

Layout

Cadence Encounter was used to perform the layout of the Triangle Rendering Engine following the instructions given in the appendix.

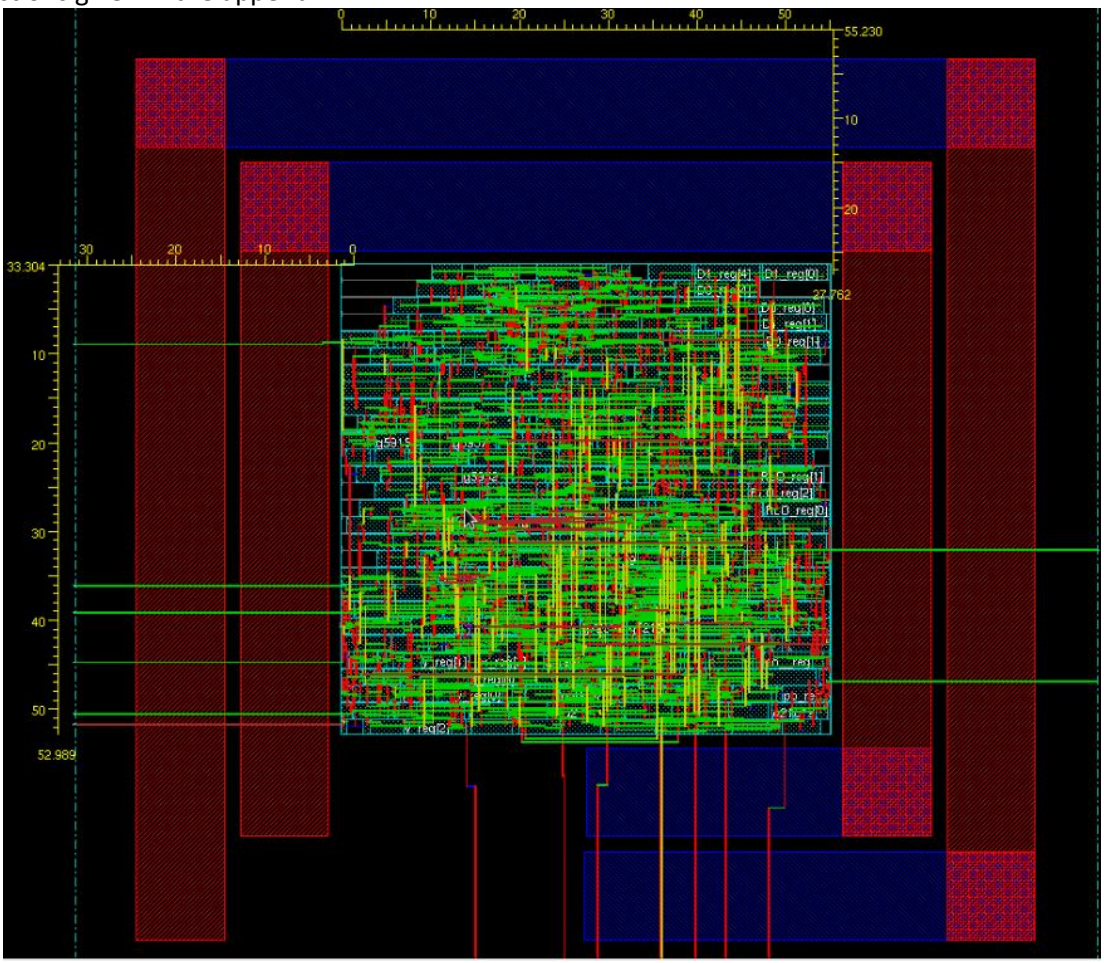


Figure 8: The completed layout for the Triangle Rendering Engine is shown above.

Results

The slack time shown in Table 1 below is a negative value that can be adjusted by decreasing the clock speed or by changing the layout dimensions and core utilization. The clock period of 400 ps can be adjusted to 500 ps to ensure a positive slack time. A 500 ps clock period equates to a clock frequency of 2 GHz.

Area	0.002927 μm^2
Clock Period	0.400 ns
Slack	-.071 ns
Total Power Consumption	3.639 mW

Table 1: The values listed in the table above are taken after layout.

```
#####################################################################
#  Generated by:      Cadence Encounter 10.12-s181_1
#  OS:                Linux x86_64(Host ID co2046-06.ece.iastate.edu)
#  Generated on:      Thu Dec 10 21:11:50 2015
#  Design:            triangle
#  Command:           report_timing > timing.rpt
#####################################################################
Path 1: VIOLATED Clock Gating Setup Check with Pin RC_CG_HIER_INST17/RC_CGIC_
INST/CP
Endpoint:   RC_CG_HIER_INST17/RC_CGIC_INST/E (v) checked with  leading edge of
'clk'
Beginpoint: y_min_reg[1]/Q                   (v) triggered by  leading edge of
'clk'
Other End Arrival Time          0.137
- Clock Gating Setup            0.048
+ Phase Shift                   0.400
= Required Time                 0.490
- Arrival Time                  0.561
= Slack Time                   -0.071
     Clock Rise Edge                 0.040
     + Clock Network Latency (Prop)  0.178
     = Beginpoint Arrival Time       0.218
     +-------------------------------------------------------------------------------------------+
     |            Instance            |     Arc      |     Cell      | Delay | Arrival | Required |
     |                                |              |               |       |  Time   |   Time   |
     |--------------------------------+--------------+---------------+-------+---------+----------|
     | y_min_reg[1]                   | CP ^         |               |       |   0.218 |    0.147 |
     | y_min_reg[1]                   | CP ^ -> Q v  | DFQD4         | 0.143 |   0.361 |    0.290 |
     | g7363                          | A2 v -> ZN ^ | OAI211D2      | 0.037 |   0.398 |    0.327 |
     | g7362                          | A1 ^ -> ZN v | ND2D1         | 0.040 |   0.438 |    0.367 |
     | g7361                          | A v -> CON ^ | FCICOND1      | 0.073 |   0.511 |    0.440 |
     | g5975                          | A2 ^ -> ZN v | OAI21D2       | 0.049 |   0.561 |    0.490 |
     | RC_CG_HIER_INST17              | enable v     | RC_CG_MOD_17  |       |   0.561 |    0.490 |
     | RC_CG_HIER_INST17/RC_CGIC_INST | E v          | CKLNQD1       | 0.000 |   0.561 |    0.490 |
     +-------------------------------------------------------------------------------------------+
```

Figure 9: The after layout timing report is shown above.

```
*------------------------------------------------------------------------------------------
*   Encounter 10.12-s181_1 (64bit) 07/28/2011 22:52 (Linux 2.6)
*
*
*   Date & Time:    2015-Dec-12 16:30:22 (2015-Dec-12 22:30:22 GMT)
*
*------------------------------------------------------------------------------------------
*
*   Design: triangle
*
*   Liberty Libraries used:
*           ./../libdir/tcbn65gpluswc.lib
*
*   Power Domain used:
*
*       User-Defined Activity : N.A.
*
*       Activity File: N.A.
*
*       Hierarchical Global Activity: N.A.
*
*       Global Activity: N.A.
*
*       Sequential Element Activity: N.A.
*
*       Clock Gates Output Activity: N.A.
*
*       Clock Gates Enable Activity: N.A.
*
*       Primary Input Activity: 0.200000
*
*   Power Units = 1mW
*
*   Time Units = 1e-09 secs
*
*       report_power -outfile Layout_power -sort total
*
------------------------------------------------------------------------------------------
```

```
Total Power
-----------------------------------------------------------------------------------------
Total Internal Power:       2.692      73.97%
Total Switching Power:     0.9241      25.39%
Total Leakage Power:       0.0233     0.6403%
Total Power:                3.639
-----------------------------------------------------------------------------------------


Group                          Internal   Switching     Leakage       Total  Percentage
                               Power      Power         Power         Power  (%)
-----------------------------------------------------------------------------------------
Sequential                        1.405     0.08867     0.01033       1.504       41.34
Macro                                 0           0           0           0           0
IO                                    0           0           0           0           0
Combinational                    0.1708      0.2008    0.008846      0.3804       10.46
Clock (Combinational)            0.7685      0.4265     0.00315       1.198       32.93
Clock (Sequential)               0.3469      0.2081   0.0009784      0.5559       15.28
-----------------------------------------------------------------------------------------
Total                             2.692      0.9241      0.0233       3.639         100
-----------------------------------------------------------------------------------------


Rail                  Voltage   Internal   Switching     Leakage       Total  Percentage
                                Power      Power         Power         Power  (%)
-----------------------------------------------------------------------------------------
Default                   0.9      2.692      0.9241      0.0233       3.639         100


Clock                          Internal   Switching     Leakage       Total  Percentage
                               Power      Power         Power         Power  (%)
-----------------------------------------------------------------------------------------
clk                               1.115      0.6346    0.004129       1.754        48.2
-----------------------------------------------------------------------------------------
Total (excluding duplicates)      1.115      0.6346    0.004129       1.754        48.2
-----------------------------------------------------------------------------------------



-----------------------------------------------------------------------------------------
*   Power Distribution Summary:
*       Highest Average Power:               clk__L1_I0 (INVD24):      0.1365
*       Highest Leakage Power:               clk__L4_I6 (INVD24):    0.000306
*       Total Cap:  3.32019e-12 F
*       Total instances in design:   660
*       Total instances in design with no power:     0
*          Total instances in design with no activity:     0
*       Total Fillers and Decap:     0
-----------------------------------------------------------------------------------------
```

Figure 10: The after layout power report is shown above.

Conclusion

The hardest part of this project was understanding how to determine if the points were in the triangle and getting the timing correct to report those values. Since for loops are generally not synthesizable, counters were used to loop through the rows between y1 and y3 and the columns between x1 and x2. We also learned early on that there is not a direct synthesizable division function. Dr. Chu pointed out that the equations could be re-written in such a way that they only used subtraction and multiplication which are synthesizable. After the Verilog code was tested for functionality, problems were identified with multiple drivers when synthesizing the circuit. The Verilog code was re-written to solve the problems as stated earlier in the report.

Appendix

ModelSim

Use the following commands in the terminal to launch ModelSim.

```
source /remote/Xilinx/12.2/settings64.sh
export PATH=$PATH:/remote/Modelsim/6.5c/modeltech/linux_x86_64/
export LM_LICENSE_FILE=1717@io.ece.iastate.edu:27006@io.ece.iastate.edu
```

RTL Synthesis Instructions

Tutorial for Cadence RTL Compiler

1. Setting up: Please download the file rc.rar and decompress it to a newly created project directory. Go to the folder named "rc". It should have 3 folders, named libdir, rtl and syn.

- libdir: contains the library files the tool will use.
- rtl: contains the Verilog codes needed to be synthesized. Please copy your Verilog file which already made in Lab 1 to this folder and renamed it as "ALT_MULTADD.v". Please do not include your test bench file because that is for simulation only.
- syn: contains run_dir (which holds the results of running synthesis) and scripts (which holds the scripts for running synthesis). More importantly, in the scripts folder, there are 3 files. They are:

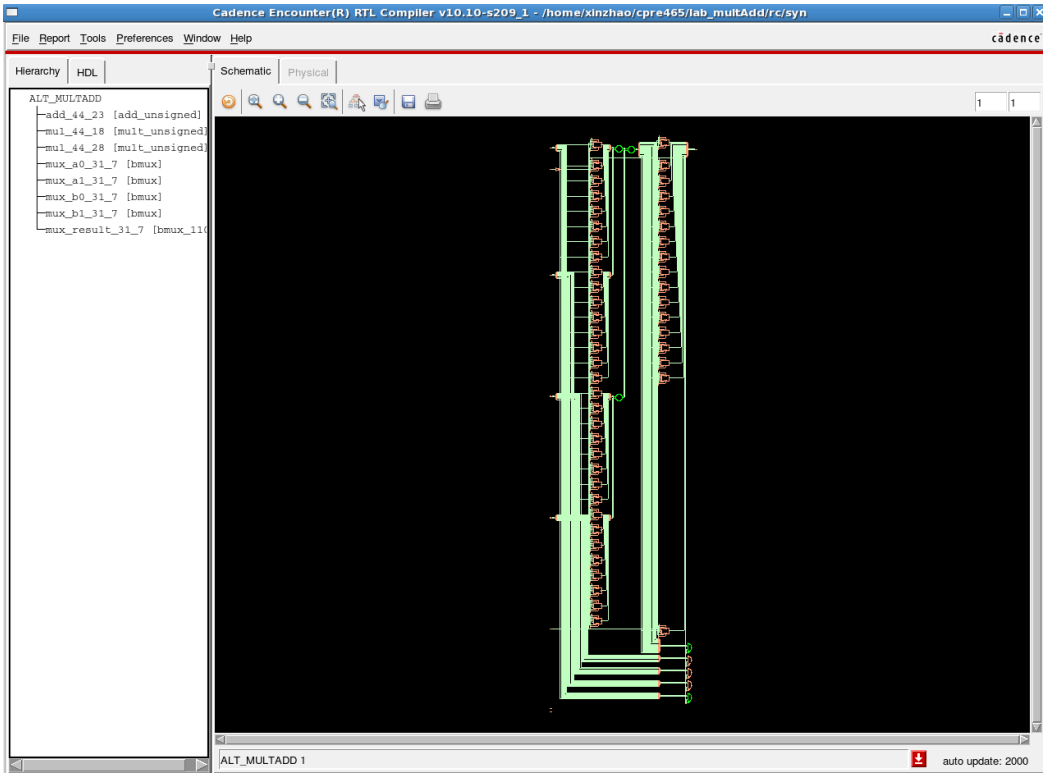
1) design.sdc: contains the constraints you want to add to the design. They are already set. Please note in the Verilog file you made in Lab 1, if you changed the port names that are defined in the Lab 1 instruction, you need to modify this file to adapt to your port names.

2) read_rtl.tcl: is a script used to read in your Verilog file. Please note if you have more than one Verilog files to be read in, you need to add lines in this file to read all your Verilog files.

3) run_synth.tcl: is the top level script to drive the synthesis tool. This file will use the other 2 files.

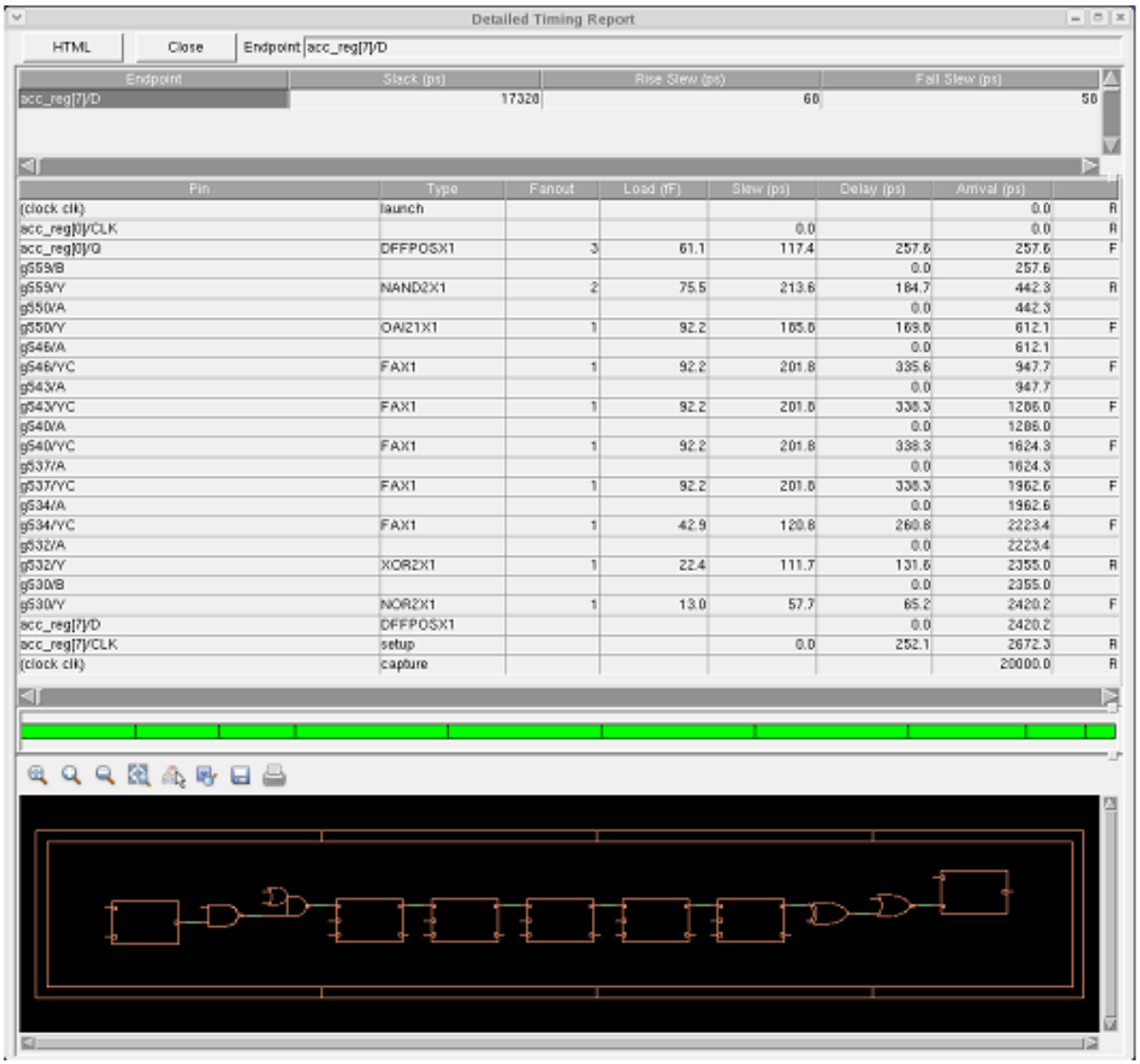
2. Starting RTL Compiler: Open a new terminal. In the newly created project directory, type "source /etc/software/edi" and hit Enter. Go to the syn directory. Then type “rc –gui” to invoke RTL Compiler, our synthesis tool from Cadence.

3. Performing synthesis: You can perform synthesis by running the script that is already made for you. Go to File ---> Source Script from the File menu of the Menu Bar. Select the run_synth.tcl in the "scripts" folder. Click OK. The tool will do the synthesis job for you. Just wait for the result. A gate level schematic will be shown in the gui window as below:



Please find the log file in the "syn" folder, and search for the keyword "error" to make sure there is no error happened during the synthesis.

4. Timing report: You can check the timing report by going to Report ---> Timing ---> Worst Path. (You may also generate a plain text version of the timing report by type "report timing" in the command window.)



Note that for the report in the diagram above, the "Slack time" is 17328ps. Since it is positive, the tool is telling you that the signal arrives at the FF on the right much earlier than necessary. This means that the circuit can work with a much higher clock frequency. If you want your design to run at a higher frequency, you need to change the design.sdc file. There is a constraint to set up the clock period. Change it to what you want. And re---run the whole flow again.

5. Area report: For the area report, go to Report ---> Netlist ---> Area. (You may also generate a plain text version of area report by type "report area" in the command window.) Check the total area. If we ask the synthesis tool to produce a faster circuit, this value is likely to increase.

6. Power report: To report the power, go to Report ---> Power ---> detailed report. (You may also generate a plain text version of power report by type "report power" in the command window.)

The power, timing and area reports are also generated by scripts and are stored in the run_dir folder. Please check it.

Encounter Instructions

Instructions on Placement and Routing by Encounter

1. Go to "run_dir" folder, source /etc/software/edi Type "usr/local/cadence/EDI101/bin/encounter" to start.

2. In the command window, type "set rda_Input(ui_pwrnet) {VDD}" and "set rda_Input(ui_gndnet) {VSS}". By using these 2 commands, we set two nets VDD and VSS.

3. Select File→→Import RTL In the "Logical" tab:

- Set "Verilog Files" to your synthesis result of Lab 4. It should be in "rc/syn/run_dir" of your Lab 4 directory. *Please double-click the file to select.*
- For "Top Level", select "Auto Assign".
- Set "Max Libs" to the "tcbn65gpluswc.lib", which is located in "encounter/libdir/lib" folder.
- Set "Constraint Files" to the .sdc file which is generated in your Lab 4 and should be located in "rc/syn/run_dir" directory of your Lab 4. It describes the constraint settings of your Lab 4. The RTL compiler outputted them as a file for Encounter to use.

In the "Physical" tab:

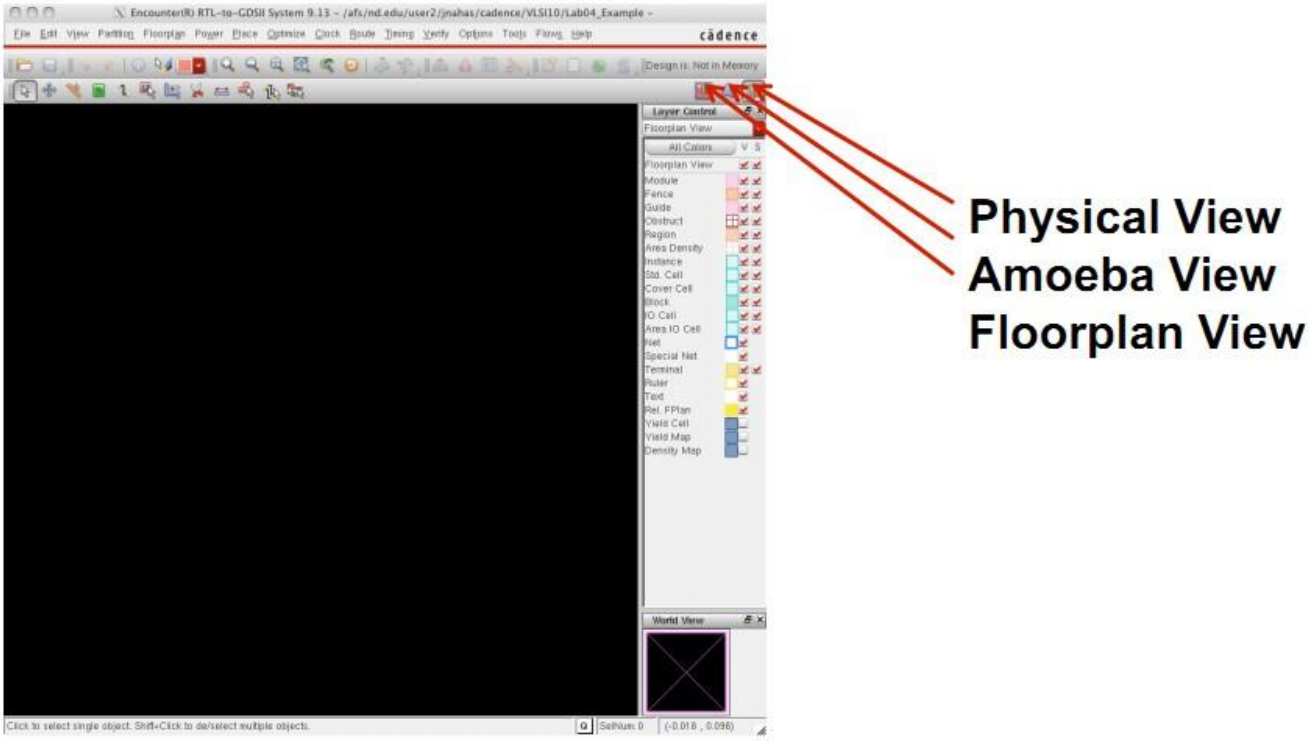
1. Set "LEF Files" to "tcbn65gplus_8lmT2.lef", which is located in "encounter/libdir/lef". It contains the geometry information of the standard cells, which is needed during placement and

routing.

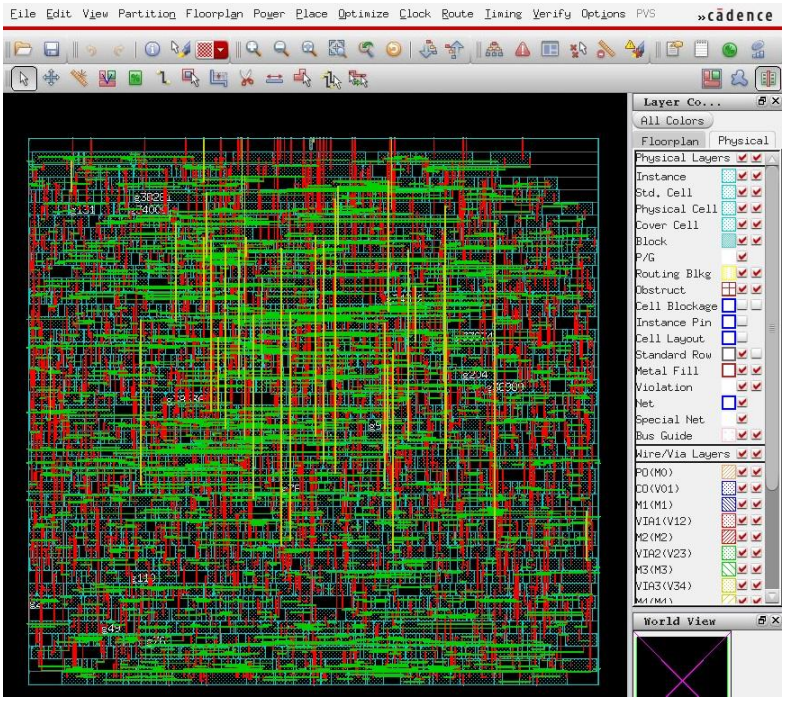
Click "OK" to submit. Now, we finished specifying the inputs for placement and routing.

4. Select File→→RTL Synthesis

Select "Proceed with Placement", and then click "OK". Ignore the warning about not specifying floorplan or def file.

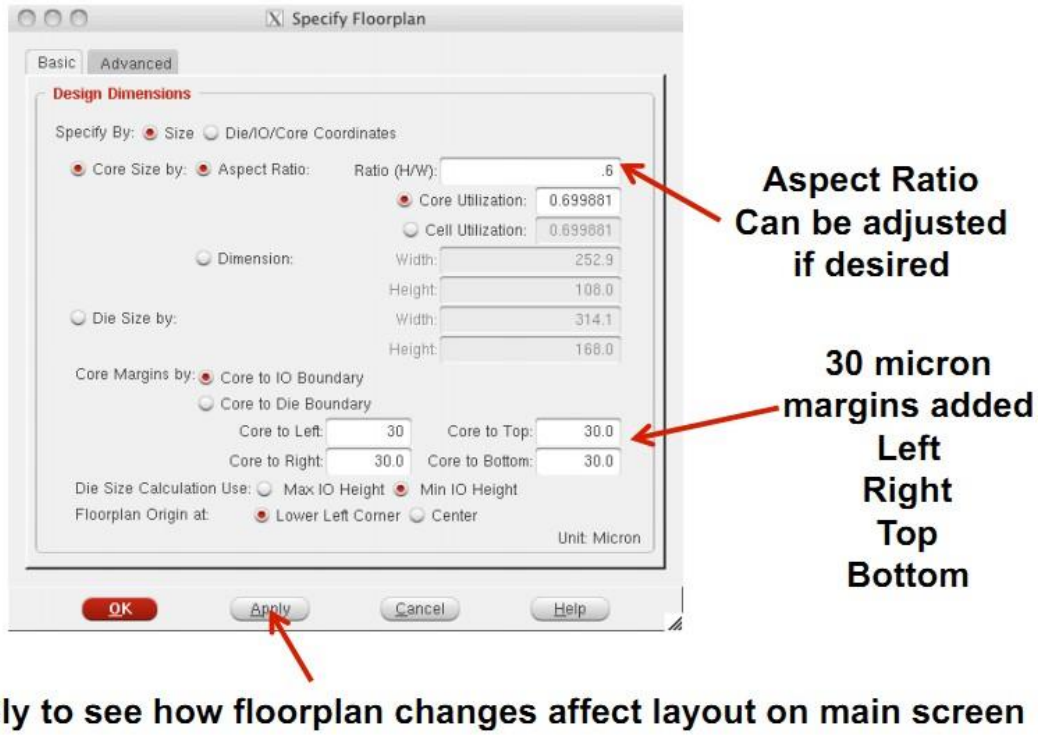


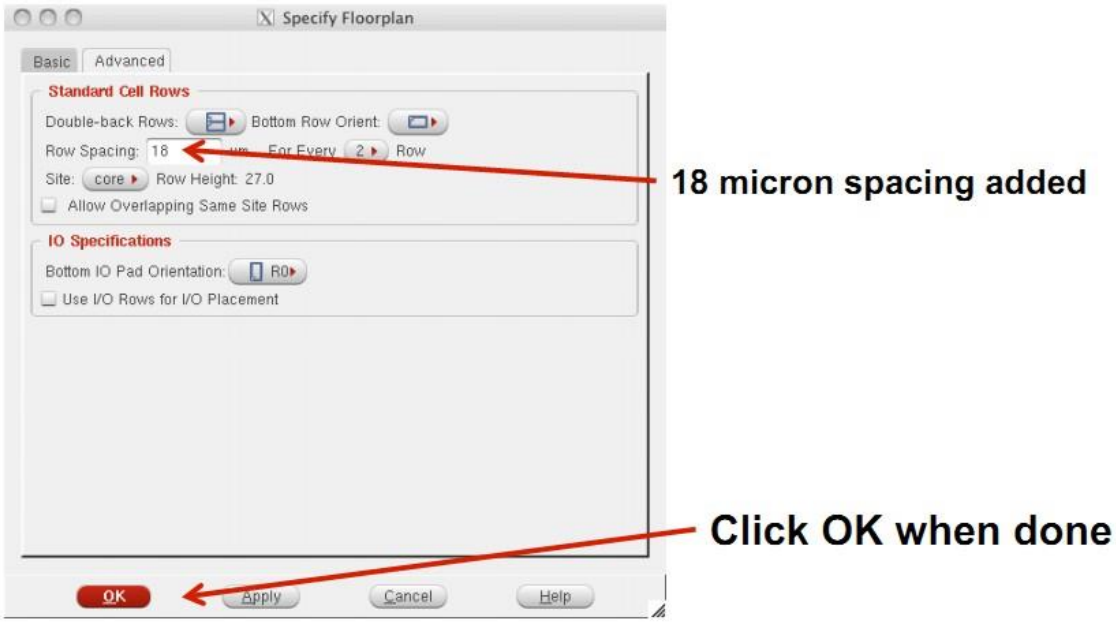
Now, please select the physical view to display your layout. You may need to press "F" to see the whole circuit.



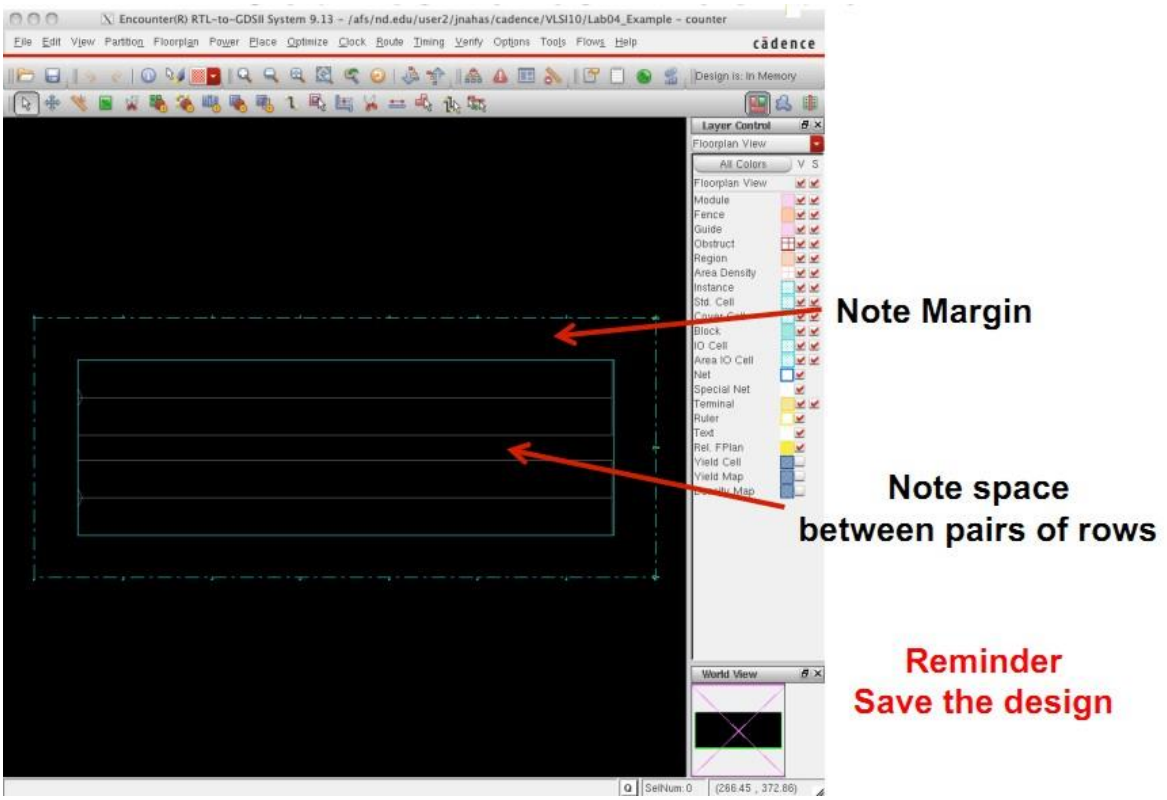
5. Select Floorplan→→Specify Floorplan

Set the parameters and options for both "Basic" and "Advanced" tabs using the values as shown in the figures below. Please note that these parameters will affect your layout result.



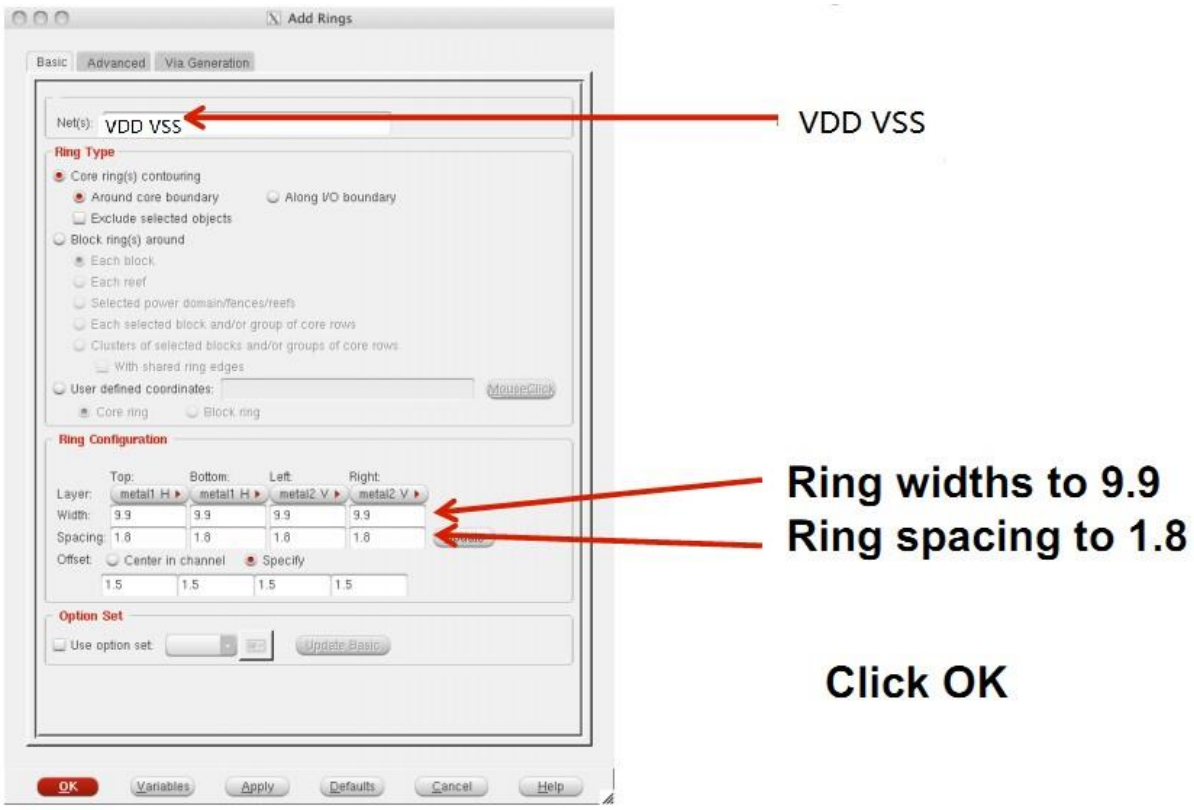


Then the placement region will be displayed:

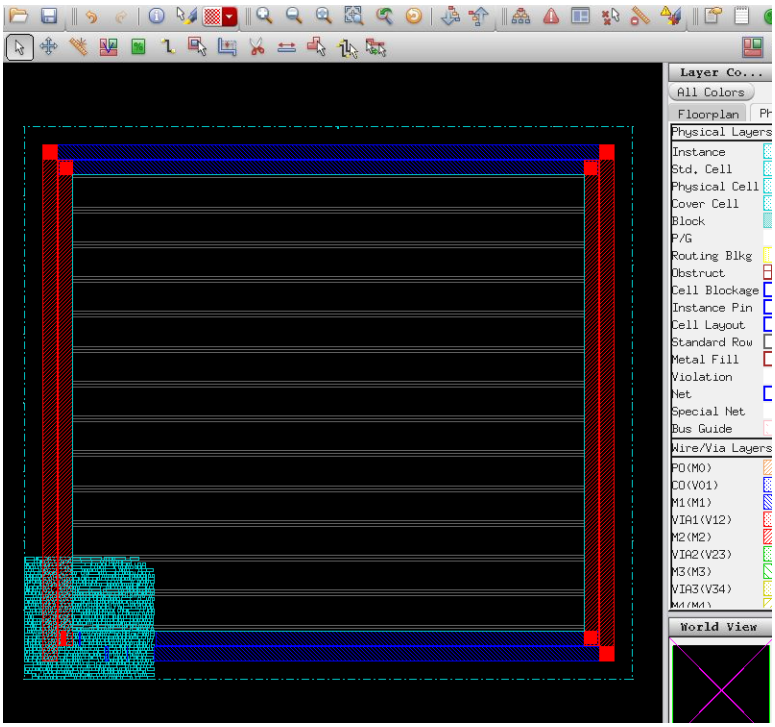


6. Select Power→→Power Planning→→Add Ring

Set the parameters and options for "Basic" tab using the values as shown in the figure below.

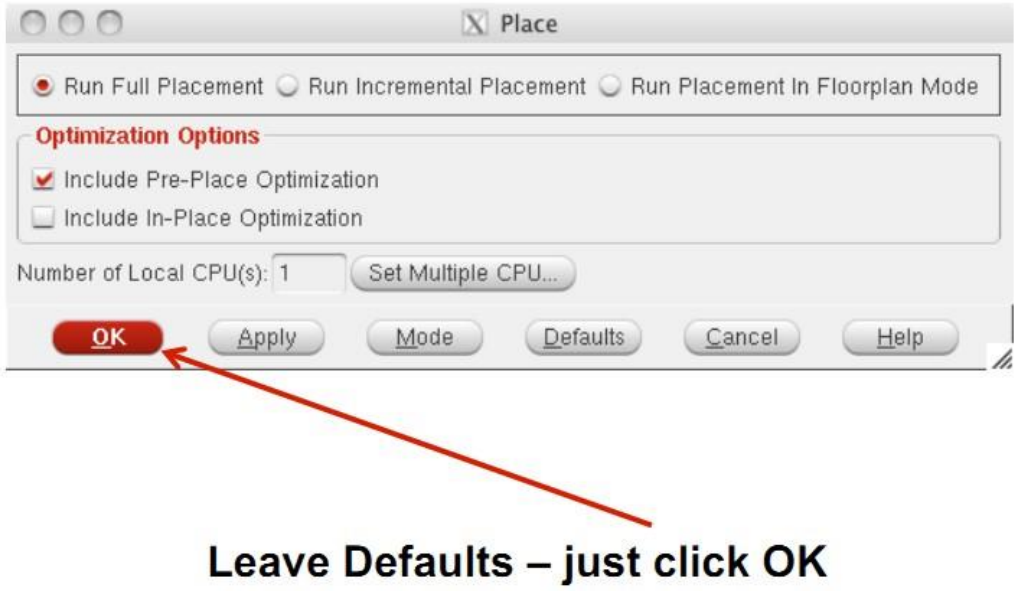


Then the power rings will be displayed:

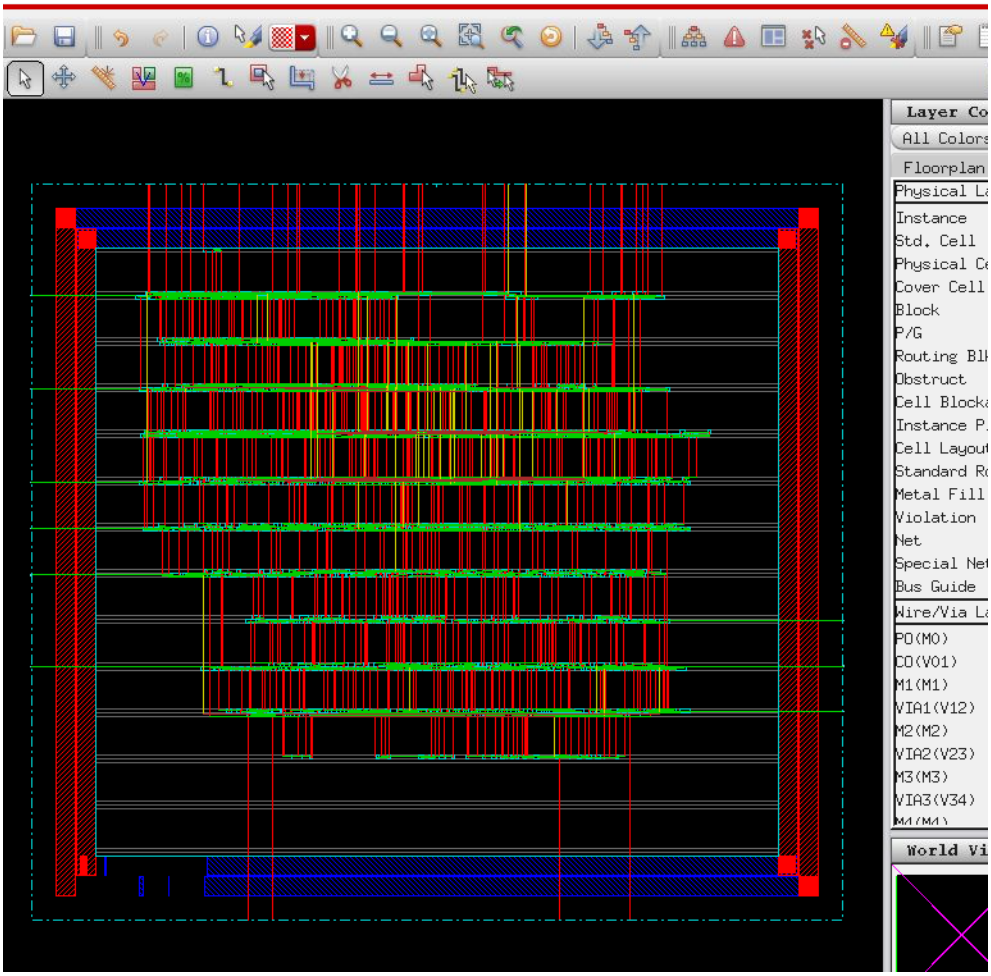


7. Select Place→→Place Standard Cell

Set the options as shown in the figure below.

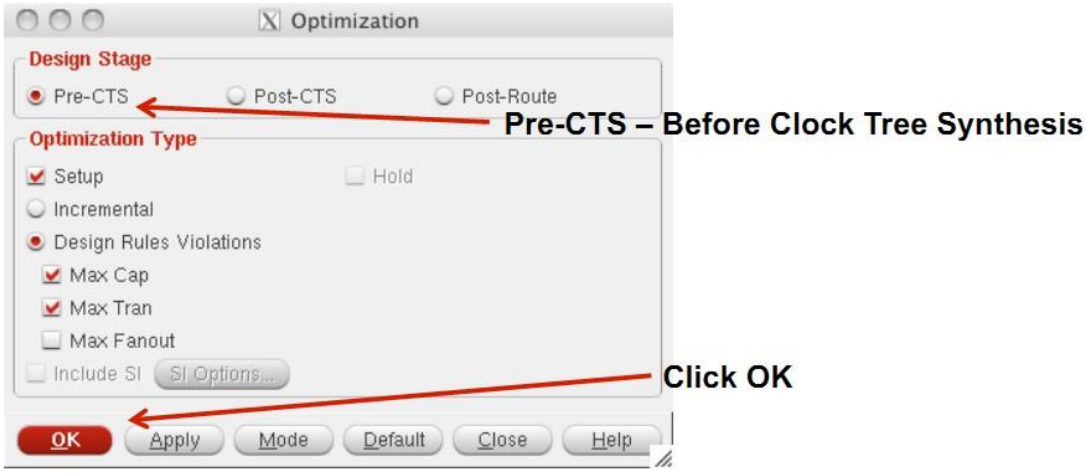


You may need to press "F" again to view the whole circuit.



8. Select Optimize→→Optimize Design

Set the options as shown in the figure below.

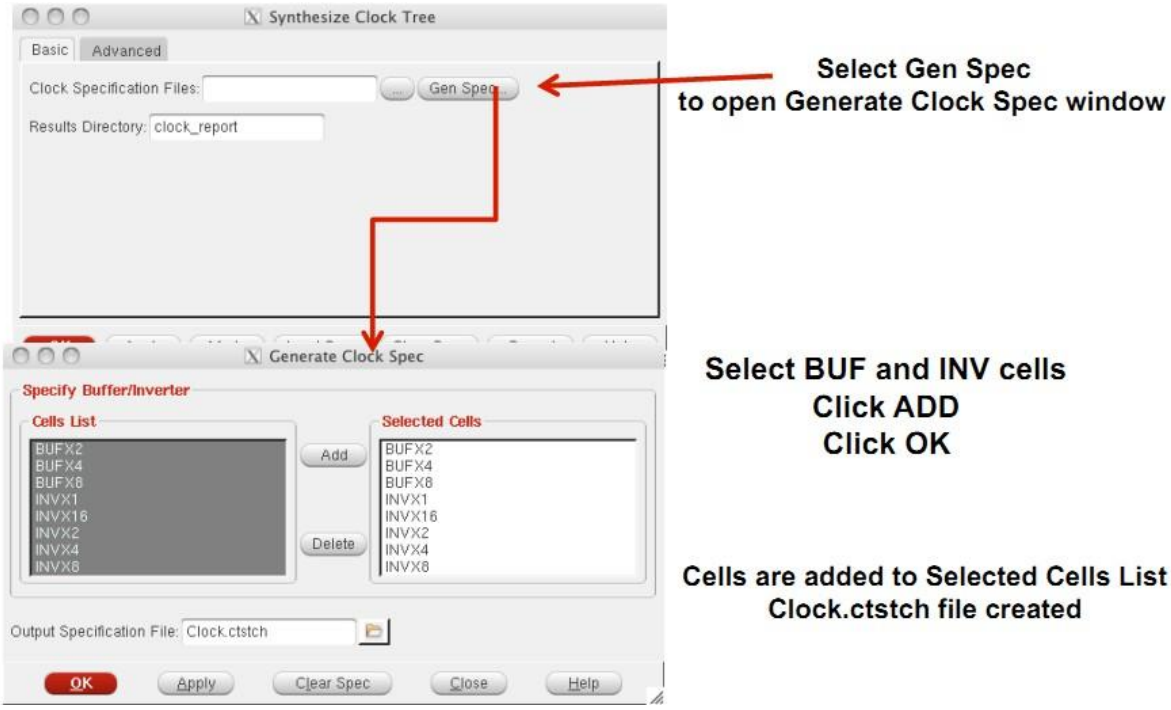


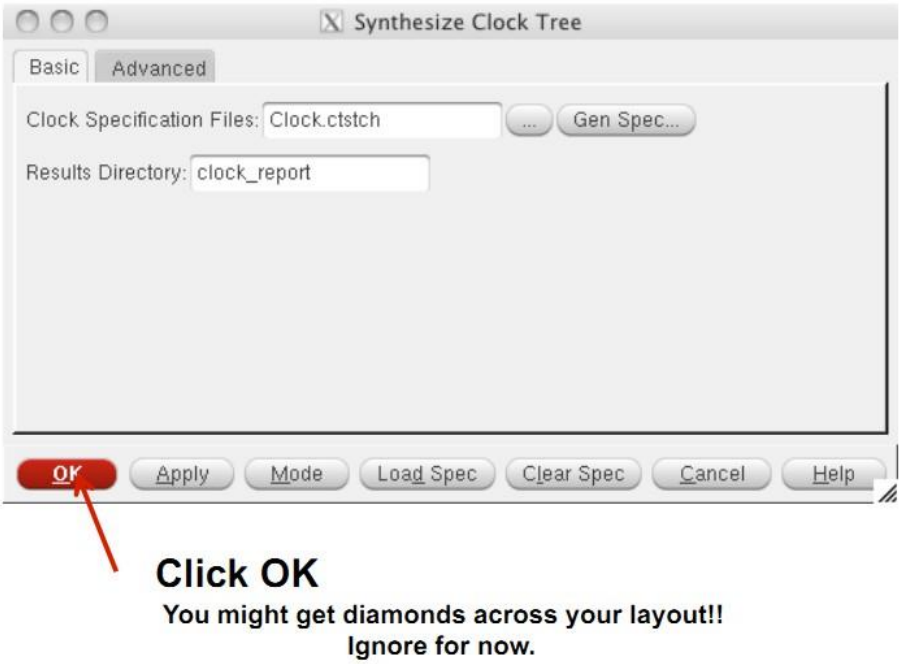
Note: You might get an error message
Ignore for now.

"CTS" means clock tree synthesis. Pre-CTS means before clock tree synthesis. In here, please note that we have selected the option to correct setup time violations.

9. Select Clock→→Synthesize Clock Tree

Set the options as shown in the figures below.





Click OK

You might get diamonds across your layout!!
Ignore for now.

You can view the clock tree by selecting Clock→→Display→→Display Clock Tree.

10. Select Optimize→→Optimize Design again Set the options as shown in the figure below.

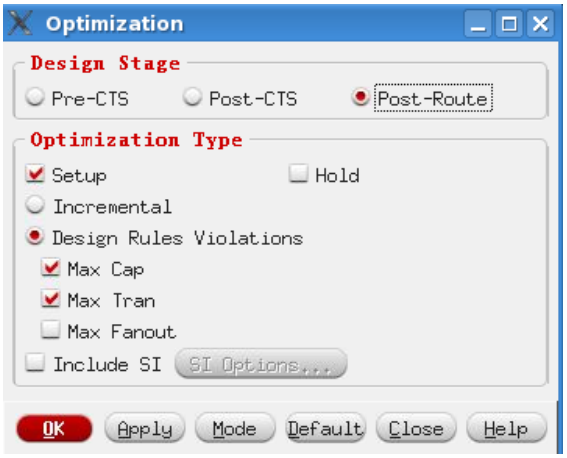
We selected Post-CTS this time. It means the optimization is performed after clock tree synthesis.

11. Select Route→→NanoRoute→→Route

Set the parameters and options as shown in the figure below.

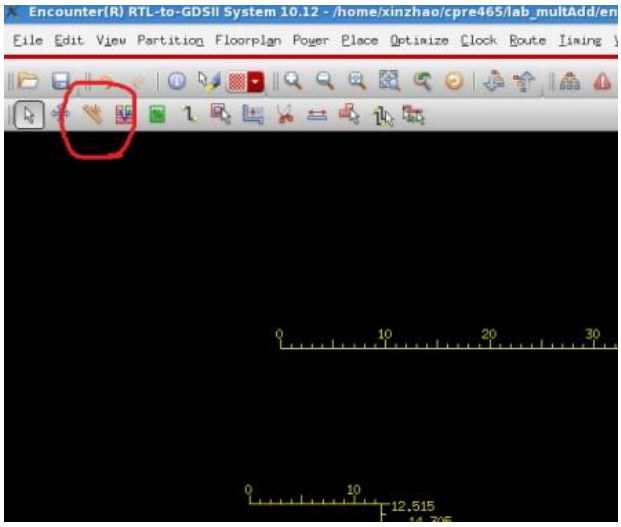
12. Select Optimize→→Optimize Design

This time, we select Post-Route because we have already performed routing.



13. Now you have finished placement and routing.

- To report power, type report_power.
- To get area information, use the ruler which is circled by red below:



- To report worst timing path, type report_timing in the command window.
- To debug timing violations, select Timing→→Debug Timing.
- To save your design, type "saveNetlist -excludeLeafCell design_pr.v" in the command window.
- To output RC parameters of your design, type "rcOut -spef design.spef" in the command window.
- To output .sdf (Standard Delay Format) file, select Timing→→Write SDF. The .sdf file contains the information required for signal delay calculation. This file would be needed if we perform post layout simulation in ModelSim.

run_synth.tcl

```
## This sets the name of the directory in which area/timing/power reports
## and synthesized (mapped) netlists are stored.
set OUTPUT_DIR ./run_dir
if { ![file exists ${OUTPUT_DIR}] }  { sh mkdir ${OUTPUT_DIR} }


#### Step 1 ####
## This tells the compiler where to look for the libraries
```

```
set_attribute lib_search_path ../libdir

## This defines the libraries to use
set_attribute library {tcbn65gpluswc.lib}
##set_attribute library {tcbn65gplustc.lib}
##set_attribute library {tcbn90ghpbc_ccs.lib}
##set_attribute lp_insert_clock_gating true

#set_attribute lp_insert_operand_isolation true
load -v2001 ../../triangle.v
elaborate
rm /designs/*


#### Step 2 ####
##This must point to your VHDL/verilog file
load -v2001 ../../triangle.v

set_attribute lp_insert_clock_gating true


#### Step 3 ####
## This builds the general block
elaborate

read_sdc ./scripts/design.sdc

dc::set_time_unit -picoseconds
dc::set_load_unit -picofarads
define_clock -period 400 -name clk [dc::get_ports {clk}] -rise 10 -fall 10
set_attribute lp_power_unit {nW}
set_attribute max_leakage_power 10000 /designs/triangle

set_attribute power_optimization_effort high
synthesize -to_mapped -effort high


report area > ${OUTPUT_DIR}/area.rpt
report gates > ${OUTPUT_DIR}/gates.rpt
report timing > ${OUTPUT_DIR}/timing.rpt
report timing -lint > ${OUTPUT_DIR}/lint.rpt
report summary > ${OUTPUT_DIR}/summary.rpt
report power > ${OUTPUT_DIR}/power.rpt
report clock_gating -summary > ${OUTPUT_DIR}/clk_gating.rpt

write -mapped > ${OUTPUT_DIR}/design_mapped.v
write_script > ${OUTPUT_DIR}/design_mapped.g
write_sdc > ${OUTPUT_DIR}/design_mapped.sdc
```

design.sdc

```
set sdc version 1.4
create_clock -period 1.0 -waveform {0 0.5} [get_ports {clk}]
set_input_delay 0.001 -max -clock "clk" [get_ports {nt}]
set_input_delay 0.001 -max -clock "clk" [get_ports {xi}]
set_input_delay 0.001 -max -clock "clk" [get_ports {yi}]
set_input_delay 0.001 -max -clock "clk" [get_ports {reset}]
```

triangle_tb.v

```
`timescale 100ps/10ps
`define CYCLE      100000   // Modify yo_tur clock period here (unit: 0.1ns)
```

```
`define INFILE1    "input.dat"
`define IN_LENGTH  6
`define INFILE2    "expect.dat"
`define OUT_LENGTH 48
`define SDF_FILE   "triangle.sdf"

module triangle_tb;
parameter INPUT_DATA = `INFILE1;
parameter EXPECT_DATA = `INFILE2;
parameter period = `CYCLE * 10;
reg     clk_t;
reg     reset_t;
reg     nt_t;
reg     [2:0] xi_t, yi_t;

wire    [2:0] xo_t, yo_t;
wire    po_t;
wire    busy_t;


integer i, j, k, l, out_f, err, pattern_num, total_num, total_cycle_num;
integer a, b, c, d;
reg [5:0]  data_base [0:`IN_LENGTH - 1];
reg [5:0]  data_base_expect [0:`OUT_LENGTH - 1];
reg [5:0]  data_tmp_expect;
reg [5:0]  data_tmp_i1, data_tmp_i2, data_tmp_i3;


triangle top(clk_t, reset_t, nt_t, xi_t, yi_t, busy_t, po_t, xo_t, yo_t);

//initial $sdf_annotate(`SDF_FILE,top);

initial $readmemb(INPUT_DATA,  data_base);
initial $readmemb(EXPECT_DATA,  data_base_expect);

initial begin
   $dumpvars();
   $dumpfile("triangle.vcd");

   clk_t   = 1'b1;
   reset_t = 1'b0;
   nt_t    = 1'b0;
   xi_t    = 3'bz;
   yi_t    = 3'bz;
   l = 0;
   i = 0;
   j = 0;
   k = 0;
   err = 0;
   pattern_num = 1 ;
   total_num = 0 ;
end

initial begin
   out_f = $fopen("OUT.DAT");
   if (out f == 0) begin
      $display("Output file open error !");
      $finish;
   end
end
```

```
always
   #(period/2)  clk_t = ~clk_t;


always
   #(period*700) $stop;


initial begin
   @(negedge clk_t)
      reset_t = 1'b1;
      $display ("\n****** START to VERIFY the Triangel Rendering Enginen
OPERATION ******\n");
      #(period - 0.1)
      reset_t = 1'b0;

   for(i = 0; i < `IN_LENGTH; i = i + k) begin
      if(busy_t == 1'b1) begin
         @(negedge clk_t)
            nt_t =1'b0;
            k  =0;
      end else begin
         k  = 3;
         // cycle 1
         @(negedge clk_t)
            nt_t = 1'b1;
            #(`CYCLE*3)  // read x1 & y1
               data_tmp_i1 = data_base[i];
               xi_t = data_tmp_i1[5:3];
               yi_t = data_tmp_i1[2:0];
         @(posedge clk_t)
            #(`CYCLE*2)  // close x1 & y1
               xi_t = 3'bz;
               yi_t = 3'bz;
         // cycle 2
         @(negedge clk_t)
            nt_t =1'b0;
            #(`CYCLE*3)  // read x2 & y2
               data_tmp_i2 = data_base[i+1];
               xi_t = data_tmp_i2[5:3];
               yi_t = data_tmp_i2[2:0];
         @(posedge clk_t)
            #(`CYCLE*2)  // close x2 & y2
               xi_t = 3'bz;
               yi_t = 3'bz;
         // cycle 3
         @(negedge clk_t)
            #(`CYCLE*3)  // read x3 & y3
               data_tmp_i3 = data_base[i+2];
               xi_t = data_tmp_i3[5:3];
               yi_t = data_tmp_i3[2:0];
         @(posedge clk_t)
            #(`CYCLE*2)  // close x3 & y3
               xi_t = 3'bz;
               yi_t = 3'bz;

         $display("Waiting for the rendering operation of the triangle
po_tint_ts with:");
         $display("(x1, y1)=(%h, %h)",data_tmp_i1[5:3], data_tmp_i1[2:0]);
         $display("(x2, y2)=(%h, %h)",data_tmp_i2[5:3], data_tmp_i2[2:0]);
         $display("(x3, y3)=(%h, %h)",data_tmp_i3[5:3], data_tmp_i3[2:0]);
```

```
      end
   end
end

always @(posedge clk_t) begin
   if (po_t ==1'b1) begin
      data_tmp_expect = data_base_expect[l];
      if ((xo_t !== data_tmp_expect[5:3])|| (yo_t!== data_tmp_expect[2:0]))
begin
         $display("ERROR at %d:xo_t=(%h) yo_t=(%h)!=expect xo_t=(%h),
yo_t=(%h)",l
         ,xo_t, yo_t, data_tmp_expect[5:3], data_tmp_expect[2:0]);
         err = err + 1 ;
      end
      $fdisplay(out_f,"%h%h",xo_t,yo_t);
      l = l + 1;
   end

   if( l == `OUT_LENGTH ) begin
      if (err == 0)
         $display("PASS! All data have been generated successfully!");
      else begin
         $display("------------------------------------------------");
         $display("There are %d errors!", err);
         $display("------------------------------------------------");
      end
      $display("------------------------------------------------");
      total_num = total_cycle_num * period;
      $display("Total delay: %d ns", total_num );
      $display("------------------------------------------------");
      $stop;
   end
end

always @(posedge clk_t) begin
   if (reset_t == 1'b1)
      total_cycle_num = 0 ;
   else
      total_cycle_num = total_cycle_num + 1 ;
end

endmodule
```