**EESM5020 Digital VLSI System Design and Design Automation (Spring, 2021)**

**Design Assignment 1: ASIC Design Flow**

**Due Date: 19th April 2021**

**Objective**

In this assignment, you need to go through the complete ASIC design flow for a 64-bit **Square-Root** Carry-Select (CSL) adder. Before starting the design, please refer to the sample project of an N-bit divider provided in the tutorial on Canvas. You can use this sample project as a template for designing the Carry-Select adder. Besides, please carefully read the instructions in “ReadMe-to get start.pdf” for using the design tools. After the assignment, you will gain essential understandings about ASIC design flow.

**Hand In**

There are -- questions related to the 64-bit Square-Root Carry-Select (CSL) adder. Please hand in a report to answer all the questions and include copies of your waveforms, schematic, and layout.

**Details**

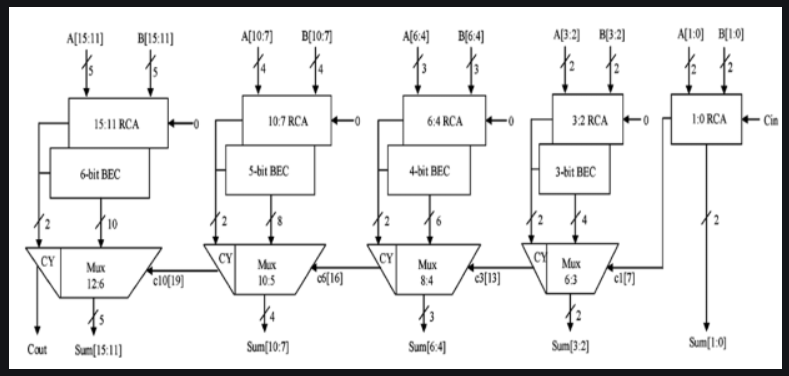
For the structure of the Square-Root Carry-Select (CSL) adder, please refer to the course notes. You can also find information on building Uniform-sized/Variable-sized CSL adders based on the 4-bit building blocks in the following website:

<https://en.wikipedia.org/wiki/Carry-select_adder>

Please pay attention that the 64-bit CSL adder should be implemented in a Square-Root fashion with minimum latency and 4-bit unit at the first stage. Please do not simply cascading 4-bit CSL units. Two code files are provided for this assignment, including “**csla\_64bit.v**” and “**csla\_64bit\_tb.v**”. Please read the code carefully. In “csla\_64bit.v”, a 16-bit Square-Root CSL adder has been implemented based on 2-bit adder units at the first stage for your reference. Please finish the part that has been marked as “TODO”.

**Questions on RTL**

**Q.1.** After reading the code in “csla\_64bit.v”, please plot the structure of the 16-bit Square-Root CSL adder and explain the advantages of this implementation compared to directly cascading four 4-bit CLS units.



The idea of the square-root is to equalize the delay of the carry chain and the select signal generated from the multiplexer from the previous stage. As the above point, making the adder one bit more can be equalize the delay rather using the same delay that the next 4 bit CLS unit need to be wait the first CLS unit

**Q.2.** Complete the **TODO** part in the provided skeleton of the Carry-Select adder (i.e. csla \_64bit.v). [Hint: please read the comments in the provided code file. For writing Verilog, you can refer to the tutorial slides on Canvas.]



**// =============================================================================**

**// ----------------------------------------------------**

**// Part I: 64-bit CSL adder**

**// ----------------------------------------------------**

**// TODO: please implement 64-bit CSL adder here, with 4bit 1st-stage.**

**module csla\_64bit(**

**input wire [63:0] a, // operator 1**

**input wire [63:0] b, // operator 2**

**input wire cin, // carry in**

**output wire [63:0] sum, // sum**

**output wire cout // carry out**

**);**

**wire [63:0] add1 ;**

**wire [63:0] add2 ;**

**assign add1 = a ;**

**assign add2 = b ;**

**wire [8\*2-1:0] mid\_c ; //guessed carries out**

**wire [127:0] mid\_sum ; // guessed sum**

**wire [8:0] selected\_c ; // selected carries out**

**wire [64-1:0] selected\_sum ; // selected sum**

**assign cout = selected\_c[8];**

**assign sum = selected\_sum;**

**RCadder #(.N(4)) bit64\_RCadder1\_i1(.add1(add1[3:0]),.add2(add2[3:0]),.cin(cin),.result(selected\_sum[3:0]),.cout(selected\_c[0]));**

**RCadder #(.N(4)) bit64\_RCadder1\_i2(.add1(add1[7:4]),.add2(add2[7:4]),.cin(1'b0),.result(mid\_sum[11:8]),.cout(mid\_c[0]));**

**RCadder #(.N(4)) bit64\_RCadder2\_i2(.add1(add1[7:4]),.add2(add2[7:4]),.cin(1'b1),.result(mid\_sum[15:12]),.cout(mid\_c[1]));**

**bit5mux #(.N(4)) bit64\_bit5mux\_i2(.in0({mid\_sum[11:8],mid\_c[0]}),.in1({mid\_sum[15:12],mid\_c[1]}),.sel(selected\_c[0]),.ou1({selected\_sum[7:4],selected\_c[1]}));**

**RCadder #(.N(5)) bit64\_RCadder1\_i3(.add1(add1[12:8]),.add2(add2[12:8]),.cin(1'b0),.result(mid\_sum[20:16]),.cout(mid\_c[2]));**

**RCadder #(.N(5)) bit64\_RCadder2\_i3(.add1(add1[12:8]),.add2(add2[12:8]),.cin(1'b1),.result(mid\_sum[25:21]),.cout(mid\_c[3]));**

**bit5mux #(.N(5)) bit64\_bit5mux\_i3(.in0({mid\_sum[20:16],mid\_c[2]}),.in1({mid\_sum[25:21],mid\_c[3]}),.sel(selected\_c[1]),.ou1({selected\_sum[12:8],selected\_c[2]}));**

**RCadder #(.N(6)) bit64\_RCadder1\_i4(.add1(add1[18:13]),.add2(add2[18:13]),.cin(1'b0),.result(mid\_sum[31:26]),.cout(mid\_c[4]));**

**RCadder #(.N(6)) bit64\_RCadder2\_i4(.add1(add1[18:13]),.add2(add2[18:13]),.cin(1'b1),.result(mid\_sum[37:32]),.cout(mid\_c[5]));**

**bit5mux #(.N(6)) bit64\_bit5mux\_i4(.in0({mid\_sum[31:26],mid\_c[4]}),.in1({mid\_sum[37:32],mid\_c[5]}),.sel(selected\_c[2]),.ou1({selected\_sum[18:13],selected\_c[3]}));**

**RCadder #(.N(7)) bit64\_RCadder1\_i5(.add1(add1[25:19]),.add2(add2[25:19]),.cin(1'b0),.result(mid\_sum[44:38]),.cout(mid\_c[6]));**

**RCadder #(.N(7)) bit64\_RCadder2\_i5(.add1(add1[25:19]),.add2(add2[25:19]),.cin(1'b1),.result(mid\_sum[51:45]),.cout(mid\_c[7]));**

**bit5mux #(.N(7)) bit64\_bit5mux\_i5(.in0({mid\_sum[44:38],mid\_c[6]}),.in1({mid\_sum[51:45],mid\_c[7]}),.sel(selected\_c[3]),.ou1({selected\_sum[25:19],selected\_c[4]}));**

**RCadder #(.N(8)) bit64\_RCadder1\_i6(.add1(add1[33:26]),.add2(add2[33:26]),.cin(1'b0),.result(mid\_sum[59:52]),.cout(mid\_c[8]));**

**RCadder #(.N(8)) bit64\_RCadder2\_i6(.add1(add1[33:26]),.add2(add2[33:26]),.cin(1'b1),.result(mid\_sum[67:60]),.cout(mid\_c[9]));**

**bit5mux #(.N(8)) bit64\_bit5mux\_i6(.in0({mid\_sum[59:52],mid\_c[8]}),.in1({mid\_sum[67:60],mid\_c[9]}),.sel(selected\_c[4]),.ou1({selected\_sum[33:26],selected\_c[5]}));**

**RCadder #(.N(9)) bit64\_RCadder1\_i7(.add1(add1[42:34]),.add2(add2[42:34]),.cin(1'b0),.result(mid\_sum[76:68]),.cout(mid\_c[10]));**

**RCadder #(.N(9)) bit64\_RCadder2\_i7(.add1(add1[42:34]),.add2(add2[42:34]),.cin(1'b1),.result(mid\_sum[85:77]),.cout(mid\_c[11]));**

**bit5mux #(.N(9)) bit64\_bit5mux\_i7(.in0({mid\_sum[76:68],mid\_c[10]}),.in1({mid\_sum[85:77],mid\_c[11]}),.sel(selected\_c[5]),.ou1({selected\_sum[42:34],selected\_c[6]}));**

**RCadder #(.N(10)) bit64\_RCadder1\_i8(.add1(add1[52:43]),.add2(add2[52:43]),.cin(1'b0),.result(mid\_sum[95:86]),.cout(mid\_c[12]));**

**RCadder #(.N(10)) bit64\_RCadder2\_i8(.add1(add1[52:43]),.add2(add2[52:43]),.cin(1'b1),.result(mid\_sum[105:96]),.cout(mid\_c[13]));**

**bit5mux #(.N(10)) bit64\_bit5mux\_i8(.in0({mid\_sum[95:86],mid\_c[12]}),.in1({mid\_sum[105:96],mid\_c[13]}),.sel(selected\_c[6]),.ou1({selected\_sum[52:43],selected\_c[7]}));**

**RCadder #(.N(11)) bit64\_RCadder1\_i9(.add1(add1[63:53]),.add2(add2[63:53]),.cin(1'b0),.result(mid\_sum[116:106]),.cout(mid\_c[14]));**

**RCadder #(.N(11)) bit64\_RCadder2\_i9(.add1(add1[63:53]),.add2(add2[63:53]),.cin(1'b1),.result(mid\_sum[127:117]),.cout(mid\_c[15]));**

**bit5mux #(.N(11)) bit64\_bit5mux\_i9(.in0({mid\_sum[116:106],mid\_c[14]}),.in1({mid\_sum[127:117],mid\_c[15]}),.sel(selected\_c[7]),.ou1({selected\_sum[63:53],selected\_c[8]}));**

**endmodule**

**Q.3.** (1) How many Carry-Select units are needed to implement a 64-bit Square-Root Carry-Select adder?

17

(2) The gate delay for the 64-bit Square-Root CSLA is how many times larger than 4-bit CSLA?

**(Information to be included for Q.1 to Q.3: finished csla\_64bit.v)**

**Questions on Behavior Simulation**

**Q.4.** Run the behavior-level simulation of your design and check the result printed in the terminal. A sample result is shown as below.

Then, modify the input stimulus in csla\_64bit\_tb.v. (You should hand in your own screenshot of behavior simulation. **Different** input stimulus must be applied to your design to generate a different result.) Report the result you generated. When you take the screenshot, please include your username in it.

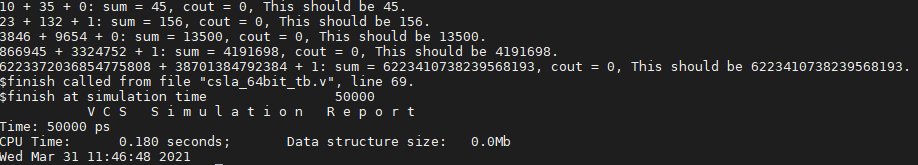
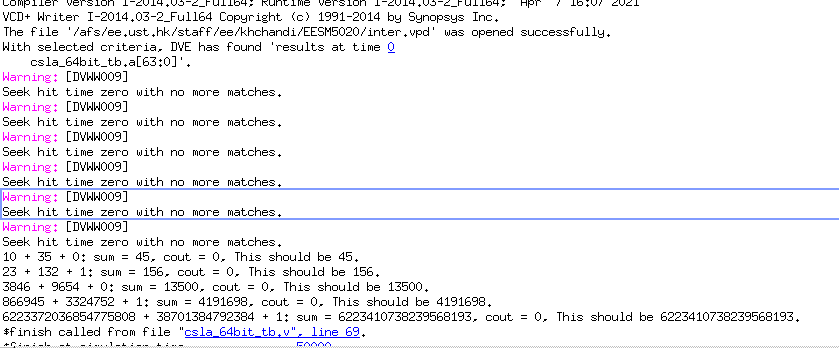
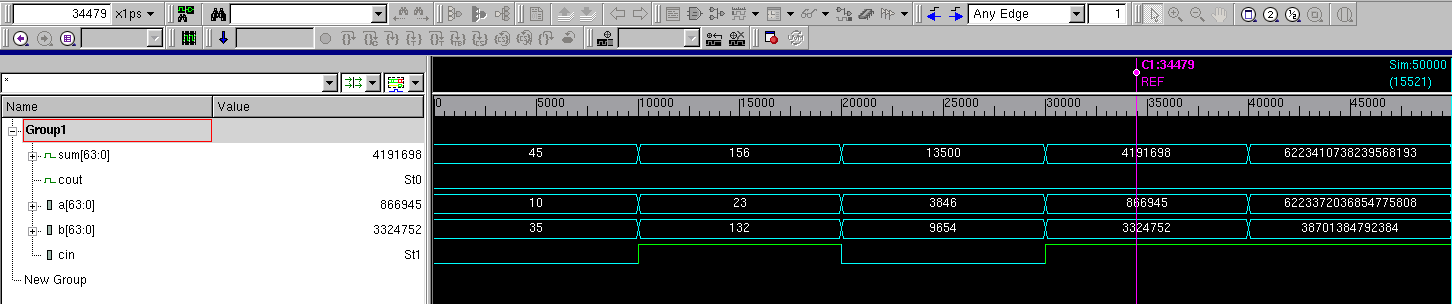


Fig. 1. Behavior-level simulation (information printed in the terminal)



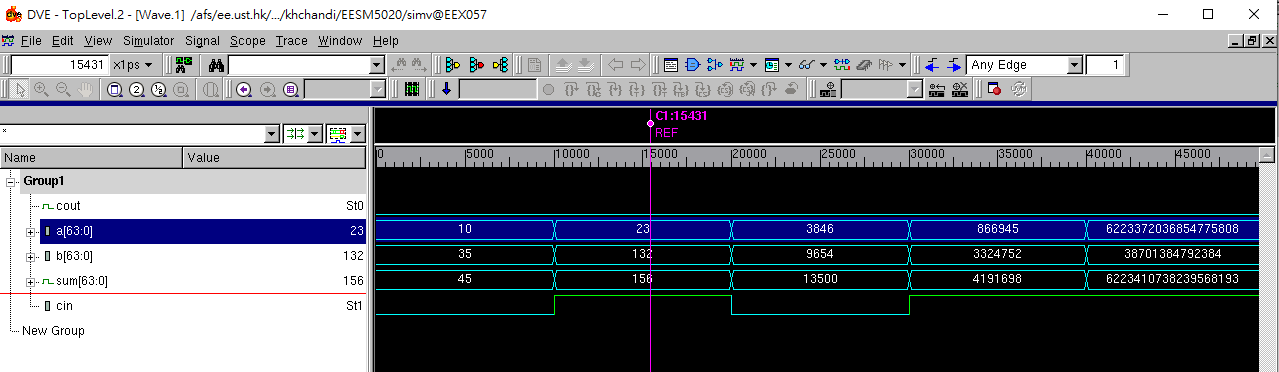
**Q.5.** Please report the waveform based on your input stimulus. To check the waveform, use the command “./simv -gui”. After opening DVE, please select the signals, right click, and select “Add Waveforms”. Then, press the button marked in the following figure to run the simulation, and you will get the waveform. Before taking the screenshot, please change the radix from hexadecimal to decimal by selecting the signals, right click, and select “Set Radix”. A sample waveform is shown as follows:



**run**

**Zoom in/out**

Fig. 2. Behavior-level simulation (waveform)



**(Information to be included for Q.4 and Q.5: results printed in the terminal and waveforms of the behavior-level simulation.)**

**Questions on Synthesis**

We will use the Nangate FreePDK45nm as the sampled PDK for our design. For your convenience, the library has already been included in the sample project of divider on Canvas. Please check the sub-directory named “lib”.

You don’t need to download the PDK or do the library conversion (from \*.lib to \*.db), because we have already done that for you. In this assignment, you can use the library “**NangateOpenCellLibrary\_slow.db**” for synthesis.

**Q.6.** We need to understand the library information before running our synthesis. Complete the following table about Nangate FreePDK 45nm. [Hint: you can find the information in “divider/lib/NangateOpenCellLibrary\_PDKv1\_3\_v2010\_12/Front\_End/Liberty/NLDM” in the sample project on Canvas. Use the command “less NangateOpenCellLibrary\_slow.lib” to read the details. Then, type “q” to exit.]

|  |  |
| --- | --- |
| Nangate FreePDK 45nm library information | |
| Area unit | um2 |
| Time unit | ns |
| Leakage power unit | nW |
| Voltage unit | V |
| Capacitance unit | fF |

The area of the 2-input NAND gate of unit drive strength is an important metric to derive the equivalent gate number of the circuit. Please find out the area of the 2-input NAND gate. [Hint: you can find the information in the following path in the sample project on Canvas: “divider/lib/NangateOpenCellLibrary\_PDKv1\_3\_v2010\_12/Front\_End/Liberty”. Use the command “less NangateOpenCellLibrary\_functional.lib” to read the details. Then, type “q” to exit.]

|  |  |
| --- | --- |
| Nangate FreePDK 45nm 2-input NAND gate (Cell name: NAND2\_X1) | |
| Area [um2] | 0.798 |

**(Information to be included for Q.6: two tables.)**

**Q.7.** Modify the provided scripts to run the whole synthesis flow. [Hint: run.tcl contains the library setup. The library location may be different from your system. Please modify the script accordingly. Note that since the designed module is a pure combinational logic, we need to create a virtual clock to constraint the time. Please refer to page 82 of the tutorial slides for how to create a virtual clock.] After successfully running the synthesis, you should be able to get different reports for your design.

Please complete the following summary table of your design:

|  |  |
| --- | --- |
| Adder synthesized under timing constraint = 5ns | |
| Area | 869.82um2 |
| Critical path [Hint 1] | 3.59ns |
| Leakage power | 14.0404uW |
| Dynamic power [Hint 2] | 90.4856uW |
| Total power | 104.5260uW |

**(Information to be included for Q.7: one table.)**

**Q.8.** Modify the timing constraint to different values and re-run the synthesis under different constraints. [Hint: the timing constraint is defined in divider.constraints.tcl in the sample project on Canvas. You need to modify the file.]

|  |  |  |  |
| --- | --- | --- | --- |
| Adder synthesized in Nangate 45nm | | | |
| Timing constraint | 2ns | 5ns | 10ns |
| Area | Can we meet the time constraint?  [Y/N] | 869.82um2 | 856.52um2 |
| Critical path | 3.59ns | 4.84ns |
| Leakage power | 14.0404uW | 13.7464uW |
| Dynamic power | 90.4856uW | 44.5559uW |
| Total power | 104.5260uW | 58.3023uW |

Please briefly analyze the results.

**(Information to be included for Q.8: one table and brief analysis.)**

**Questions on Post-synthesis simulation**

For the following questions, you only need to run the simulation and P&R on the 64-bit CLA adder synthesized under 5ns timing constraint.

**Q.9.** Run the post synthesis simulation. You need to modify the VCS compilation scripts (i.e. run) according to your settings. Please also remember to modify your testbench to annotate the information in the “sdf” file. A sample result is shown as follows:

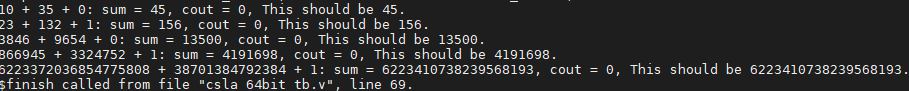
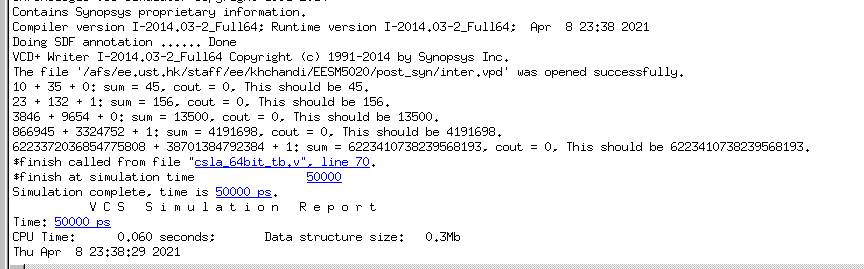


Fig. 3. Post-synthesis simulation (information printed in the terminal)



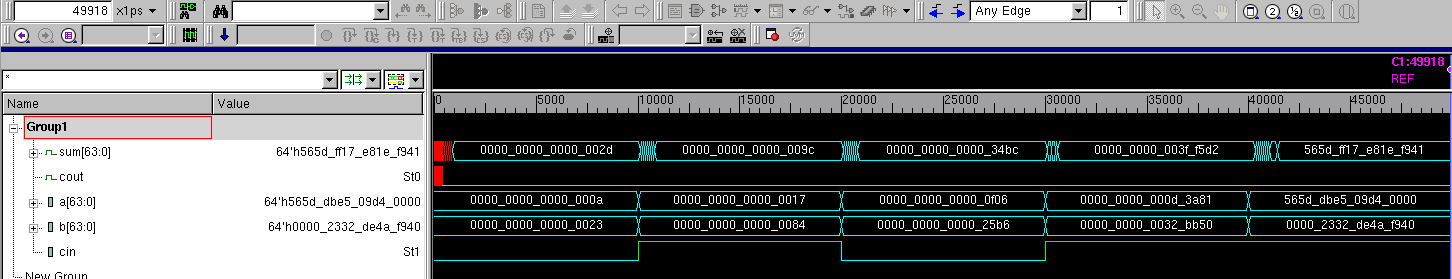
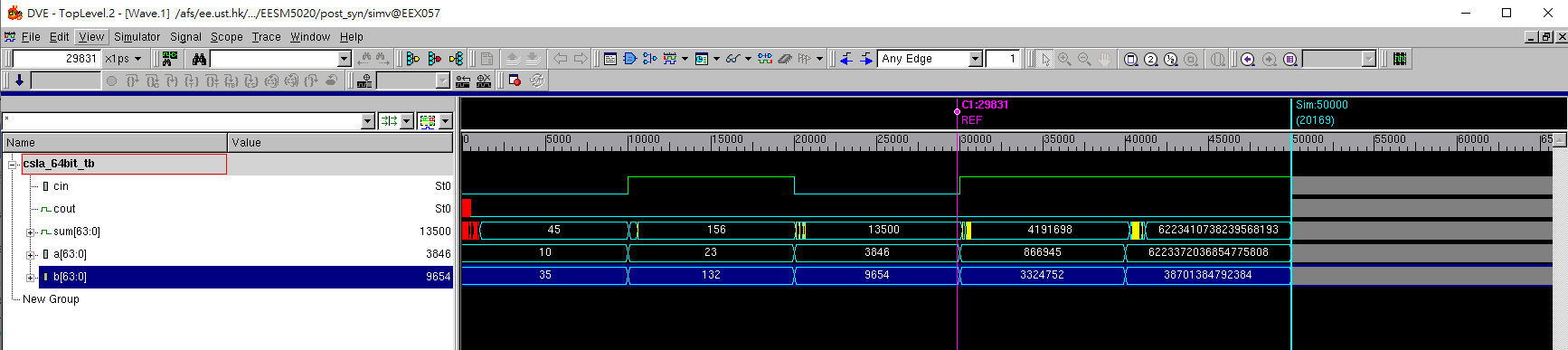


Fig. 4. Post-synthesis simulation (waveform)

As before, you should simulate the module with your own input stimulus. [Hint: modify the input stimulus in the csla\_64bit\_tb.v in syn\_sim directory]



**(Information to be included for Q.9: your own post-synthesis simulation results.)**

**Questions on P&R**

Do the P&R for the synthesized netlist from Design Compiler.

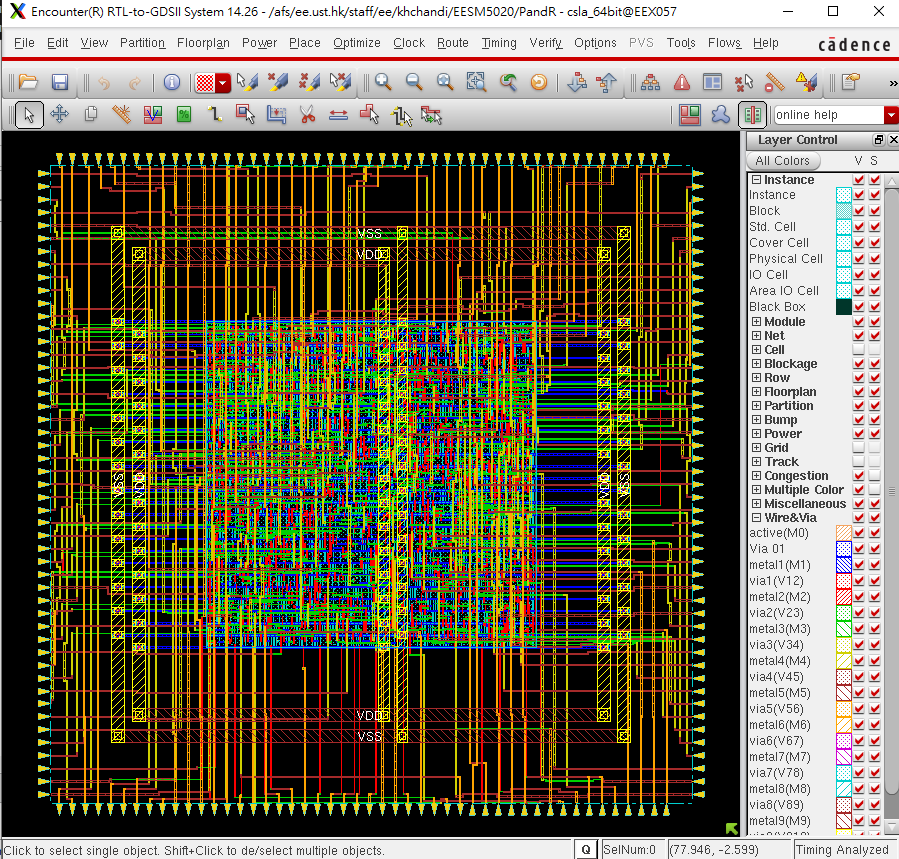
**Q.10.** Three reports should be handed after the layout step:

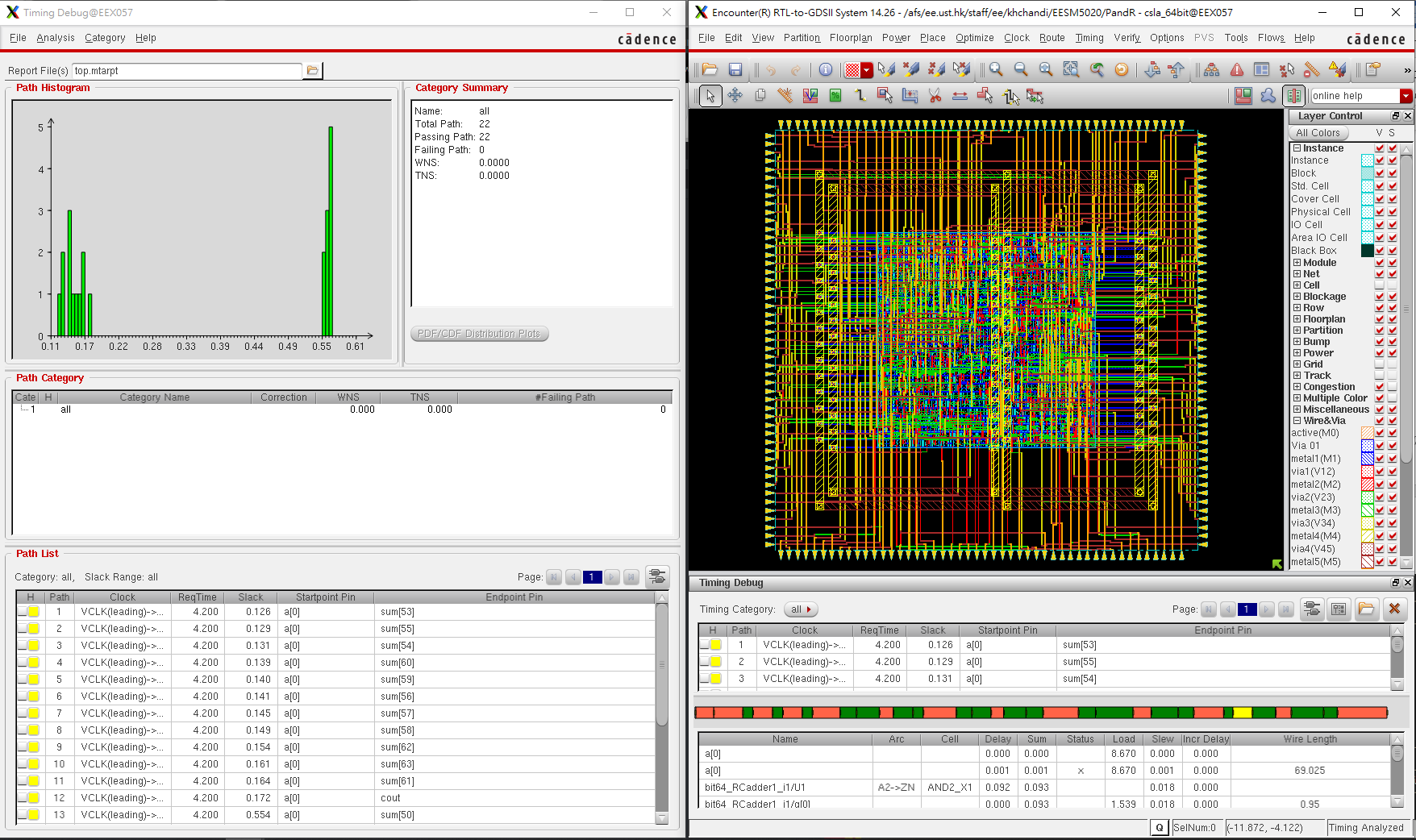
- Final layout view of your adder

- Setup timing report (histogram) of your adder

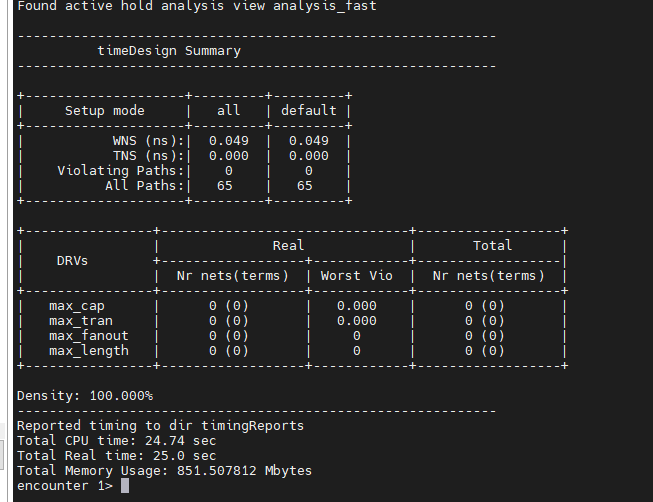
- Hold timing report (histogram) of your adder

**(Information to be included for Q.10: your own layout and timing report.)**

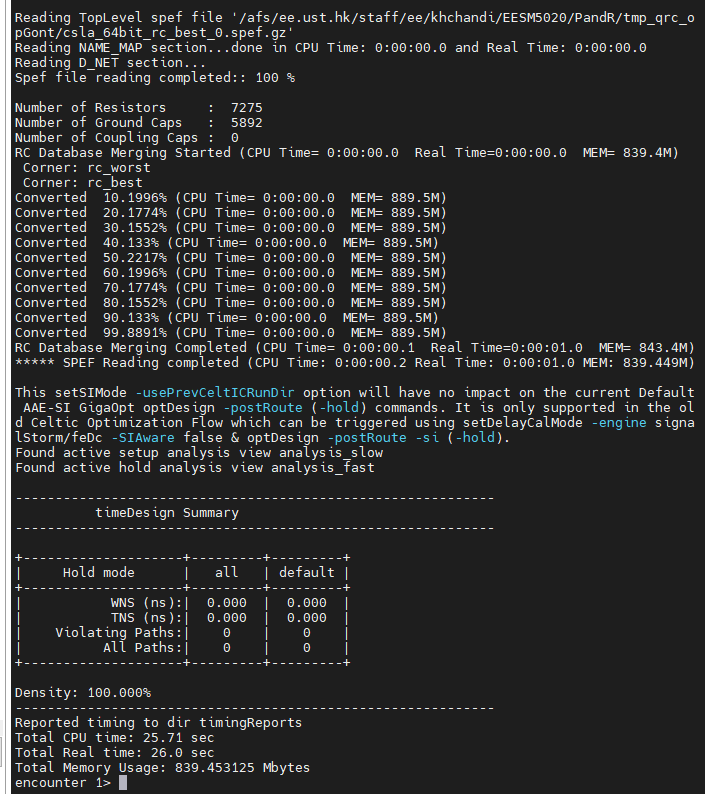


**Setup timing report (histogram)** 

**Setup Post-Route**



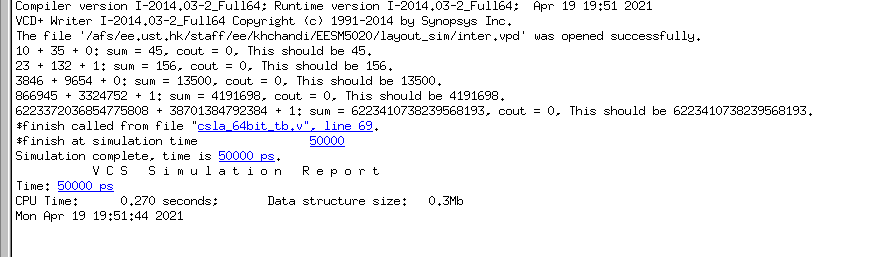
**Hold Post route**

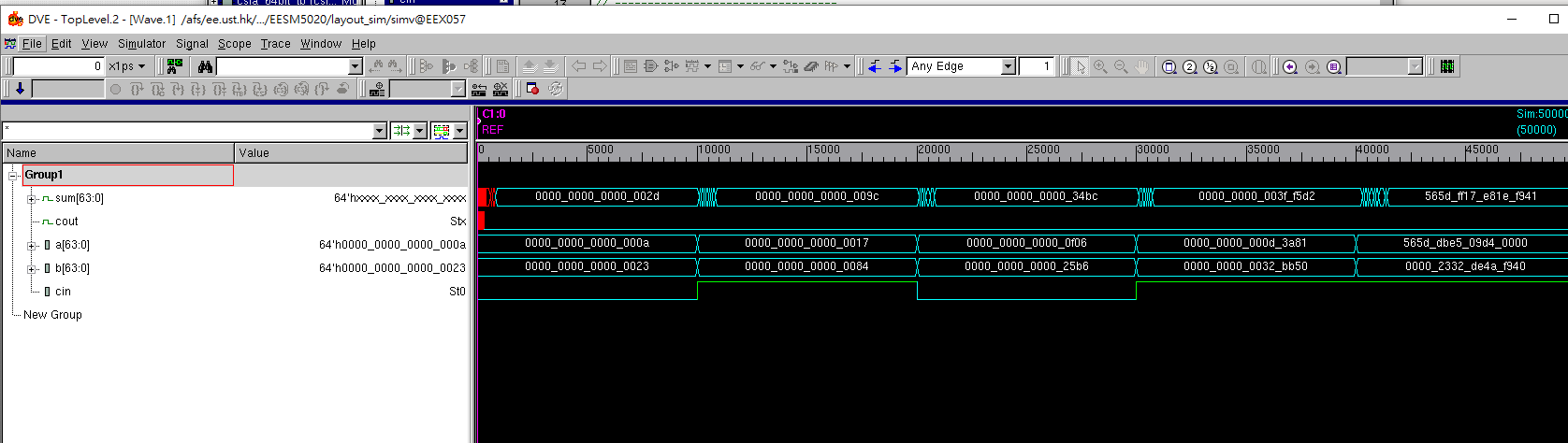


**Questions on Post-layout simulation**

**Q.11.** Run the post layout simulation under layout\_sim directory. As in Q.9, you should include your own waveform of the post-layout simulation.

**(Information to be included for Q.11: your own results of the post-layout simulation.)**

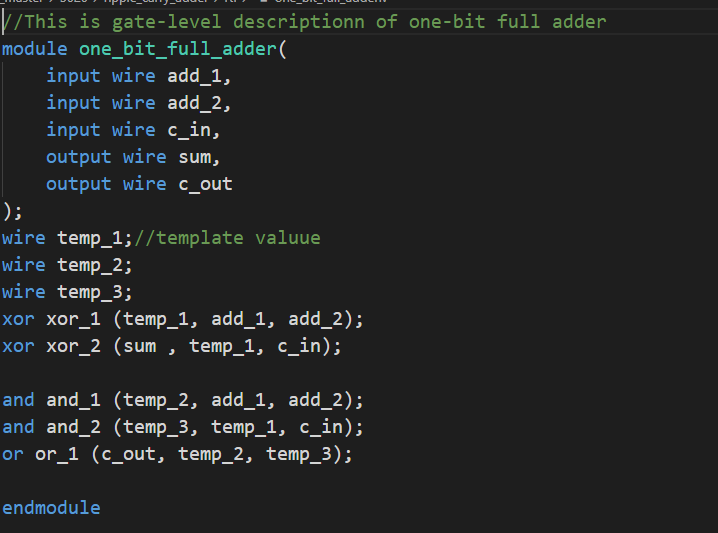
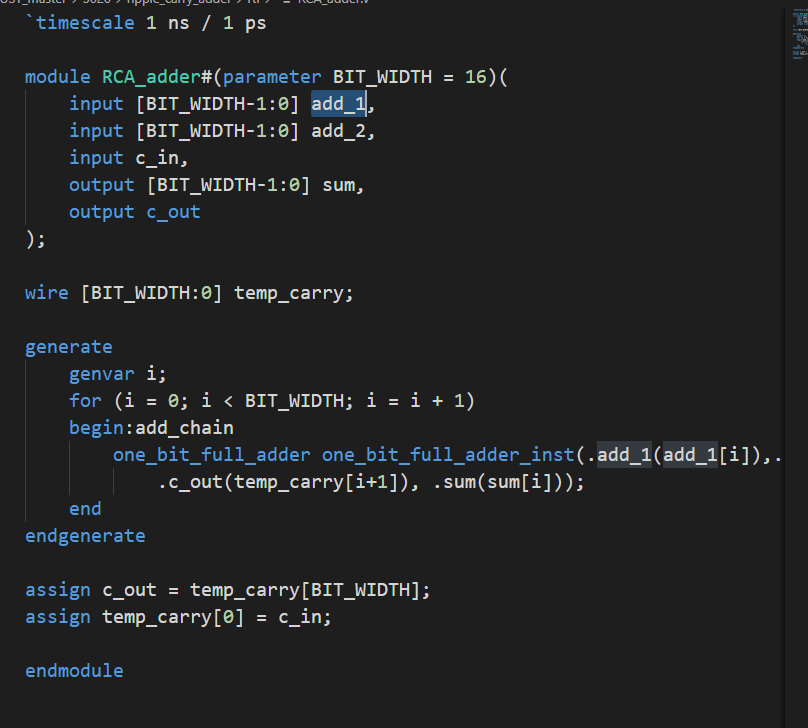


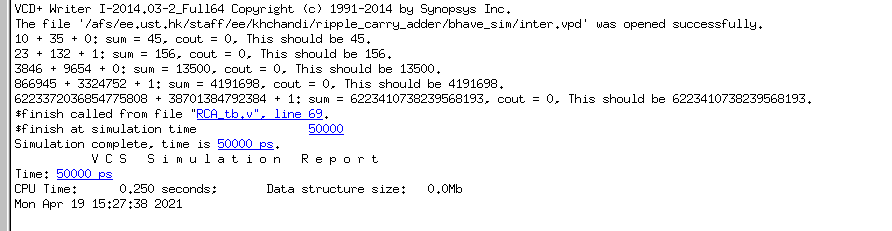


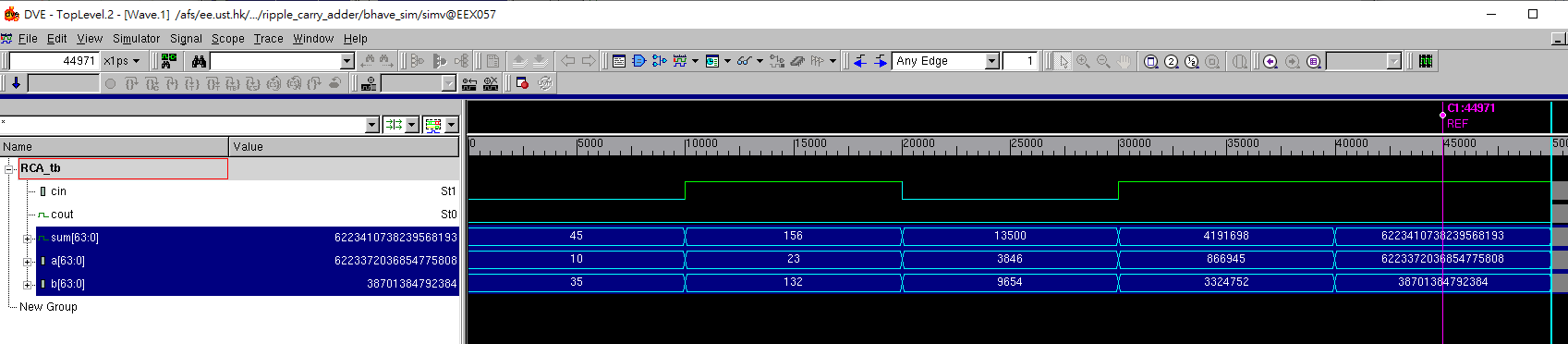
**Questions on Different Types of Adder**

**Q.12** Please design a 64-bit ripple carry adder and go through the design flow. Please compare the speed and area of the two 64-bit adder design.

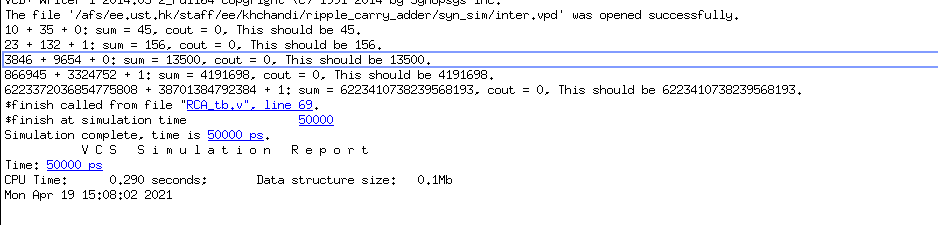
**(Information to be included for Q.12: RTL code, waveform of behavior-level simulation, speed and area analysis after synthesis, waveform of post-synthesis simulation, layout, waveform of post-layout simulation)**

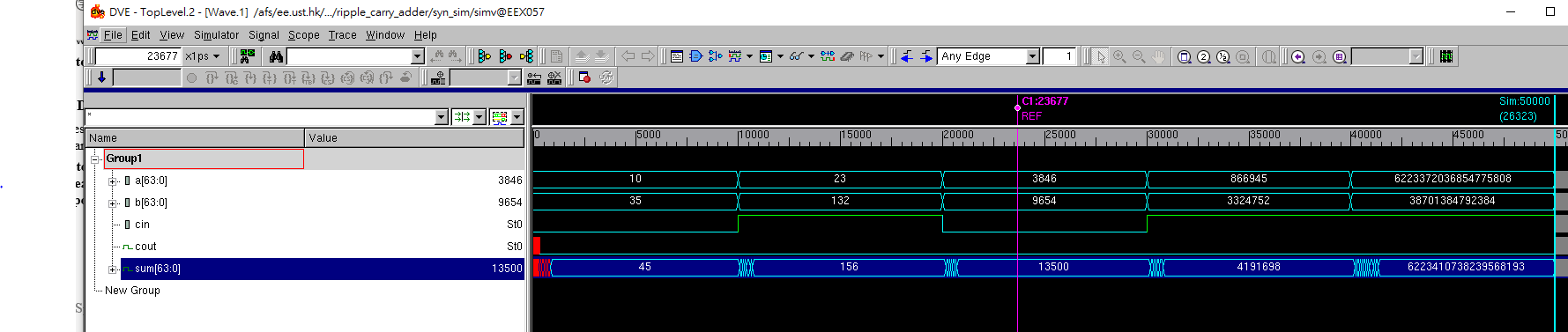
**.**

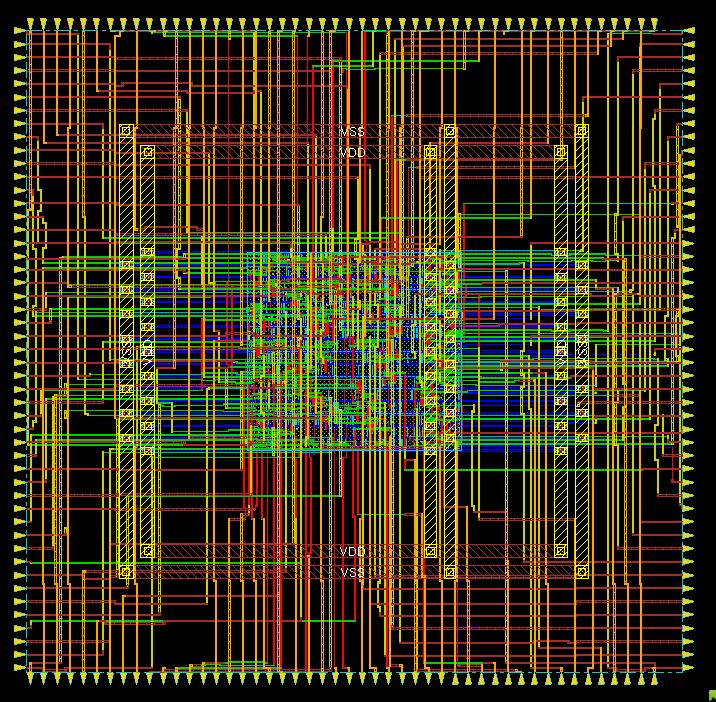




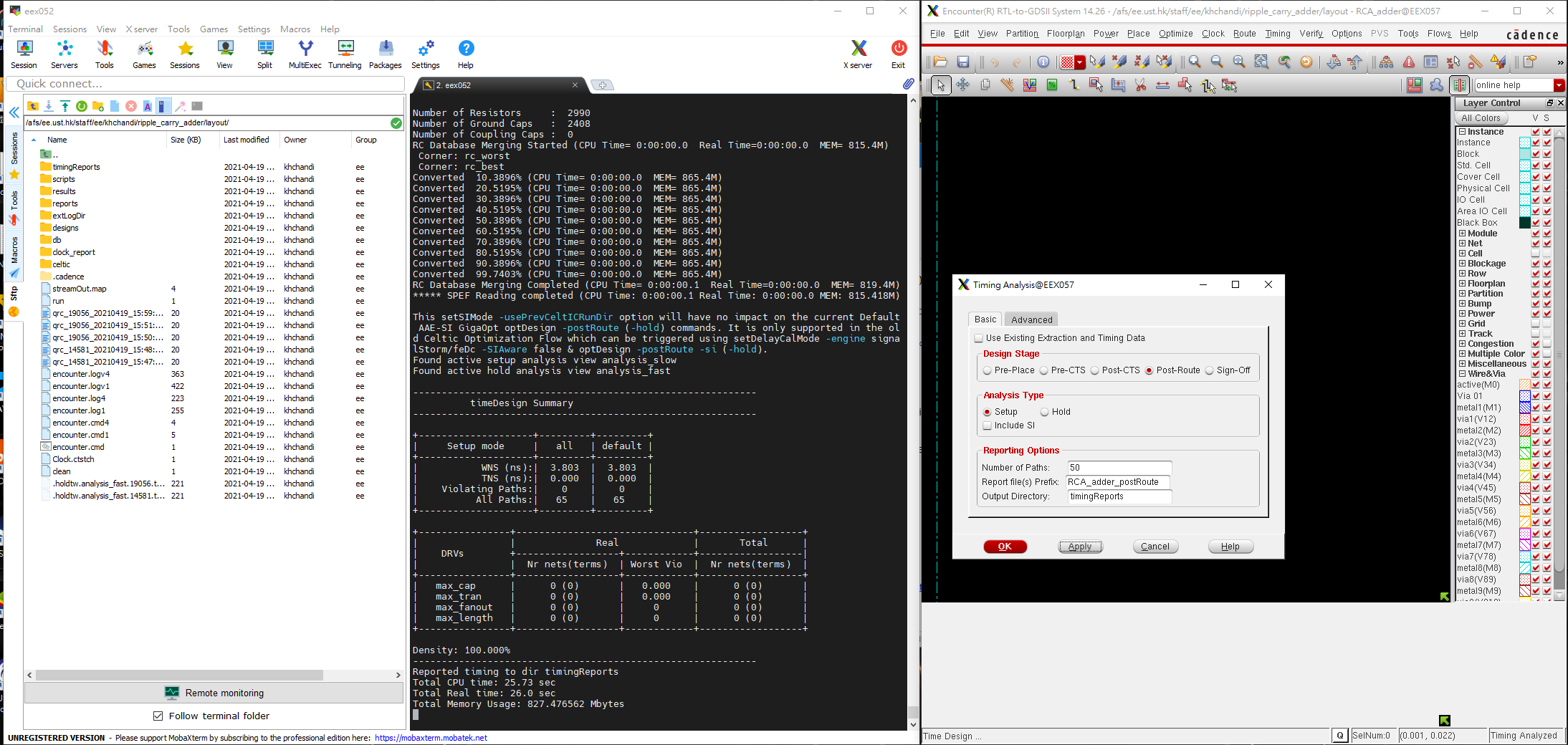
|  |  |
| --- | --- |
| Adder synthesized under timing constraint = 20ns | |
| Area | 323.455997um2 |
| Critical path [Hint 1] | 16.70ns |
| Leakage power | 14.0404uW |
| Dynamic power [Hint 2] | 8.5089uW |
| Total power | 5.0858 uW |







Post route setup time



Post route hold time

