**National Sun Yat-sen University**

**Department of Electrical Engineering**

**112-1 Practical Digital System Design**

**HW4 Traffic Light Controller**

**Name: 胡庭翊**

**Student ID: B103015006**

1. **Algorithm**
2. **Sync Counter**
3. **How to Design**

To make a counter, I need to be assured that it can be controlled not only by the clock but also inputs of reset and enable.

The states needed to be changed with the clock, so that the waiting time for each state would depend on the frequency of the clock itself.

Also, it should have the function of car detection to control whether the converting process conduct or not, lastly, it should have a function of reset so as it can be used to be manually controlled.

1. **Features**

A counter in this circuit can help user notice current state, and it will also be easier for user to modify the operation time for each states: i.e. if user wishes to have a shorter or longer time of state-convert.

1. **Operation Method**

The input should be reset, enable, and clock for 1 bit of each, and the output should be the current state after convert, so as it can be used in the latter module. The size of state should be 3 bits since we have 8 temporary states: ST0, ST1, ST2, ST3, ST4, ST5, ST6, ST7.

Triggered at every positive edge of clock, if there is car on the farm road being detected, then the convert process will be started, current state=ST0, and the state will +1 in every positive edge clock. However, if reset=1, than current state will be set to ST0 no matter enable is 1 or not.

Last but not last, when enable=0, the convert process will be stopped.

1. **Traffic Light Controller**
2. **How to Design**

To make a traffic light controller, I need to make sure that it can be controlled by a counter, and then output the states of traffic light on highway and farm road.

The states of traffic lights are red light, yellow light, and green light, which means, to operate a traffic light control circuit, we should have 6 different types of outputs.

Also, it should be known that red light, yellow light, and green light cannot happen at both side of the roads, otherwise, it may caused car accidents.

1. **Features**

By controlling 2 sides of traffic light together, we can avoid nonsense situation from happens, which is, two same lights happens at the same time. Moreover, we can easily control both traffic light system and force them to be controlled at the same time, there won’t be any chances that two traffic light system operate under different clocks.

1. **Operation Method**

When there are cars on farm road, an ideal state convert of me is:

High way: green light; Farm road: red light.

(If there are cars on farm road: )

High way: green light; Farm road: red light.

High way: yellow light; Farm road: red light.

High way: red light; Farm road: yellow light.

High way: red light; Farm road: green light.

(Wait for 5 clocks for car to pass.)

(Then convert back to highway again.)

High way: red light; Farm road: yellow light.

High way: yellow light; Farm road: red light.

High way: green light; Farm road: red light.

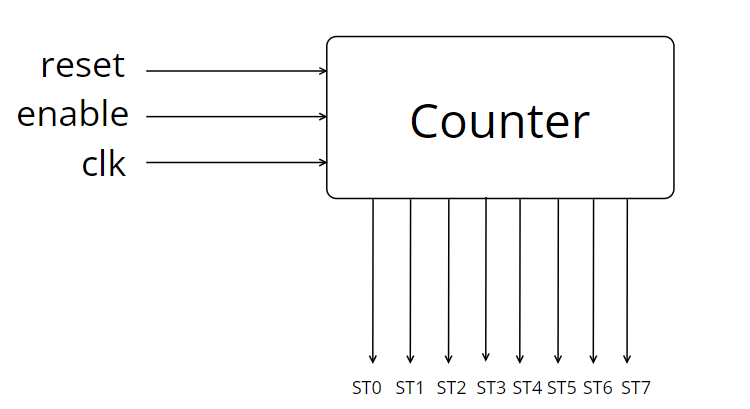
(Wait for 5 clocks for car to pass.)

(Detect if there is any car on farm road again)

(If yes, then transfer the traffic light states again.)

(If no, high way traffic light will remain green while farm road remain red.)

1. **Circuit Architecure**
2. **Sync Counter**

****

The counter will count+1 at every cycle of clock, and will be reset if reset=1, start counting when enable=1.

When counter count to 0, it outputs ST0;

when counter count to 1, it outputs ST1;

when counter count to 2, it outputs ST2;

when counter count to 3, it outputs ST3;

when counter count from 4~8, it outputs ST4;

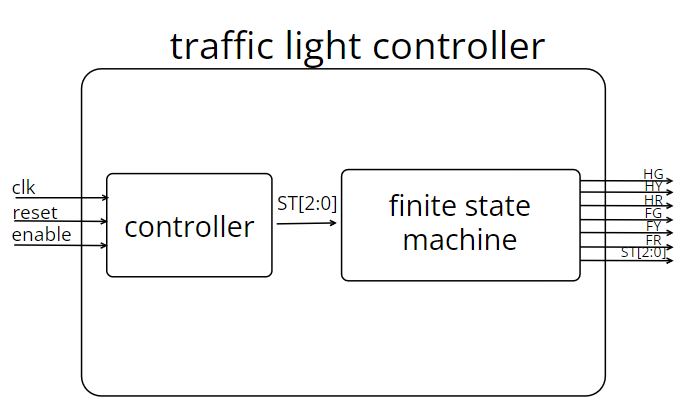
when counter count to 9, it outputs ST5;

when counter count to 10, it outputs ST6;

when counter count to 11~15, it outputs ST7.

After a cycle, the counter will count to 0 again.

1. **Communication**

****

Here, the meaning of the output state of counter is:

ST0: State of S0 in finite state machine;

ST1: State of S1 in finite state machine;

ST2: State of S2 in finite state machine;

ST3: State of S3 in finite state machine;

ST4: State of S4 in finite state machine;

ST5: State of S3 in finite state machine;

ST6: State of S2 in finite state machine;

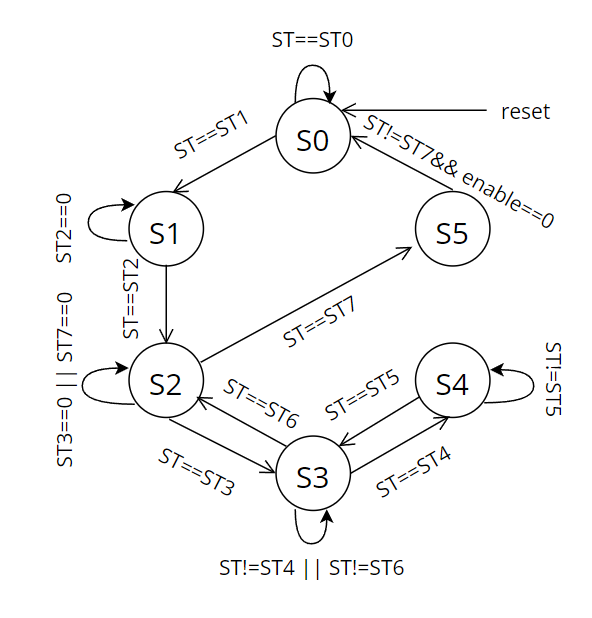
ST7: State of S5 in finite state machine.

The reason why here we have multiple assignments to the same output state is that, although we have a total number of 8 states, some of them are actually repeated. In order to simplify and smaller our area, here we assign and tidy them up together.

1. **Traffic Light Controller**

The traffic light controller is composed of a counter and a finite state machine.

The state diagram of the finite state machine is as follow:

****

The meaning of each state in finite state machine is:

S0: Highway green light; Farm road: red light.

(This is the initial state when there is no car on farm road.)

S1: Highway green light; Farm road: red light.

(There is car on farm road, the counter starts to count.)

S2: Highway yellow light; Farm road: red light.

S3: Highway red light; Farm road: yellow light.

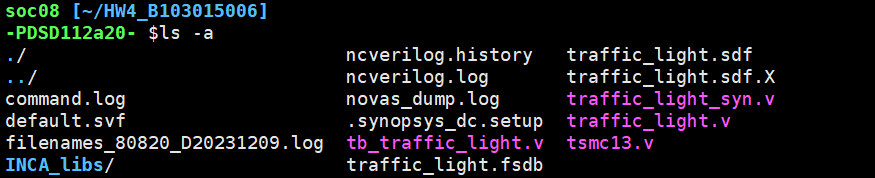
S4: Highway red light; Farm road: green light.

S5: Highway green light; Farm road: red light.

Where HR means highway red light, HY means highway yellow light, HG means highway green light; FR means farm road red light, FY means farm road yellow light, FG means farm road green light.

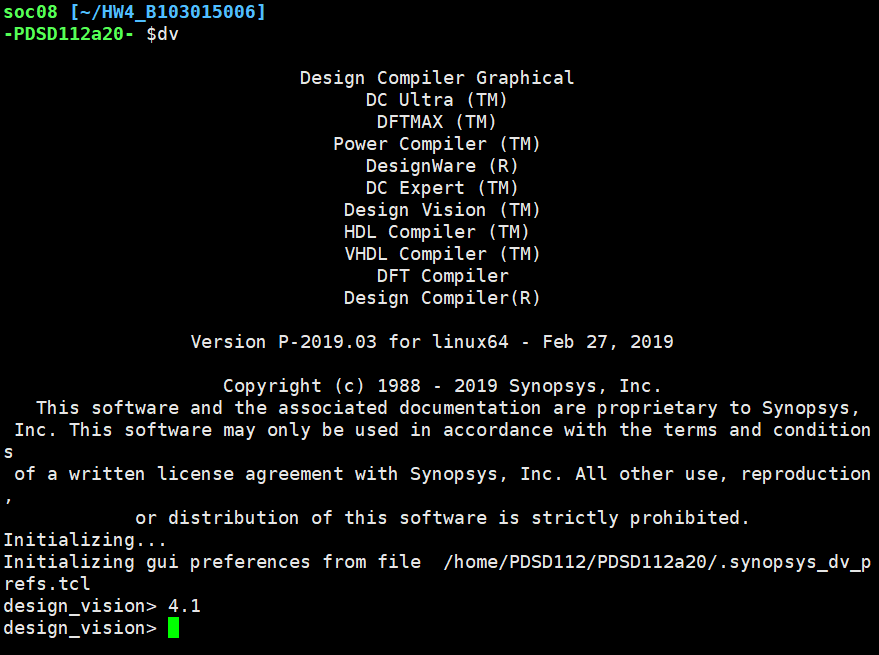
1. **Synthesis Process**
2. **Check Setup File**

First open XShell and cd to our target folder, check the folder in which the design compiler is to be executed contains the .synopsys\_dc.setup file by typing ‘ls –a’.

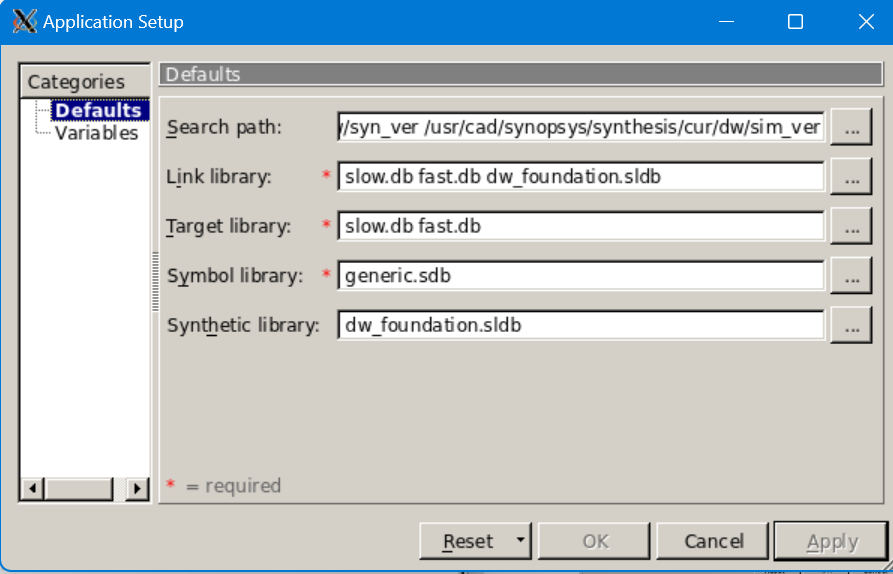
****

1. **Invoke Design Compiler**

Type ‘dv’ to invoke the design compiler.

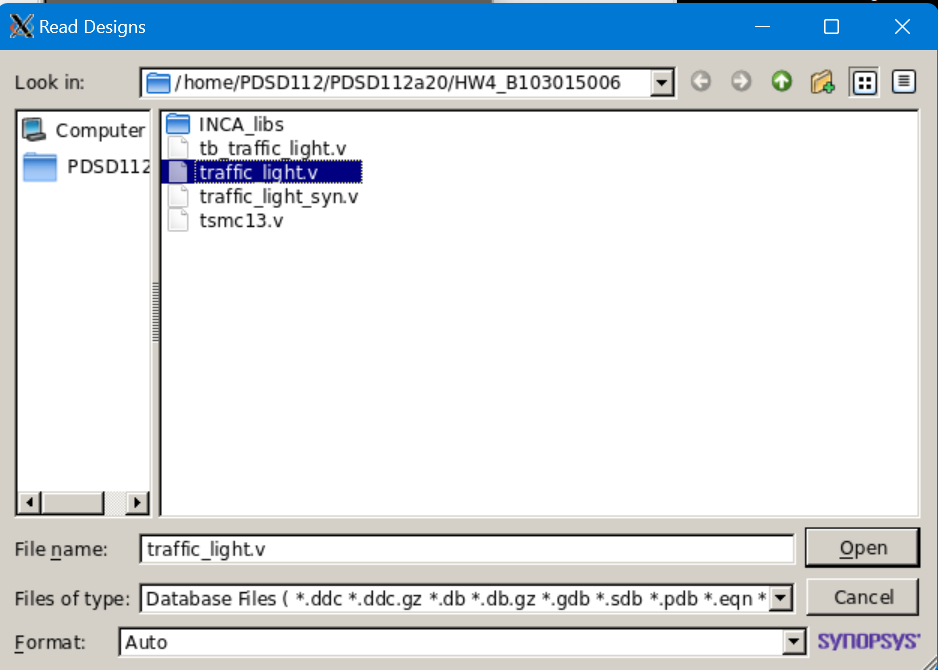
****

1. **Check if library is correctly loaded**

****

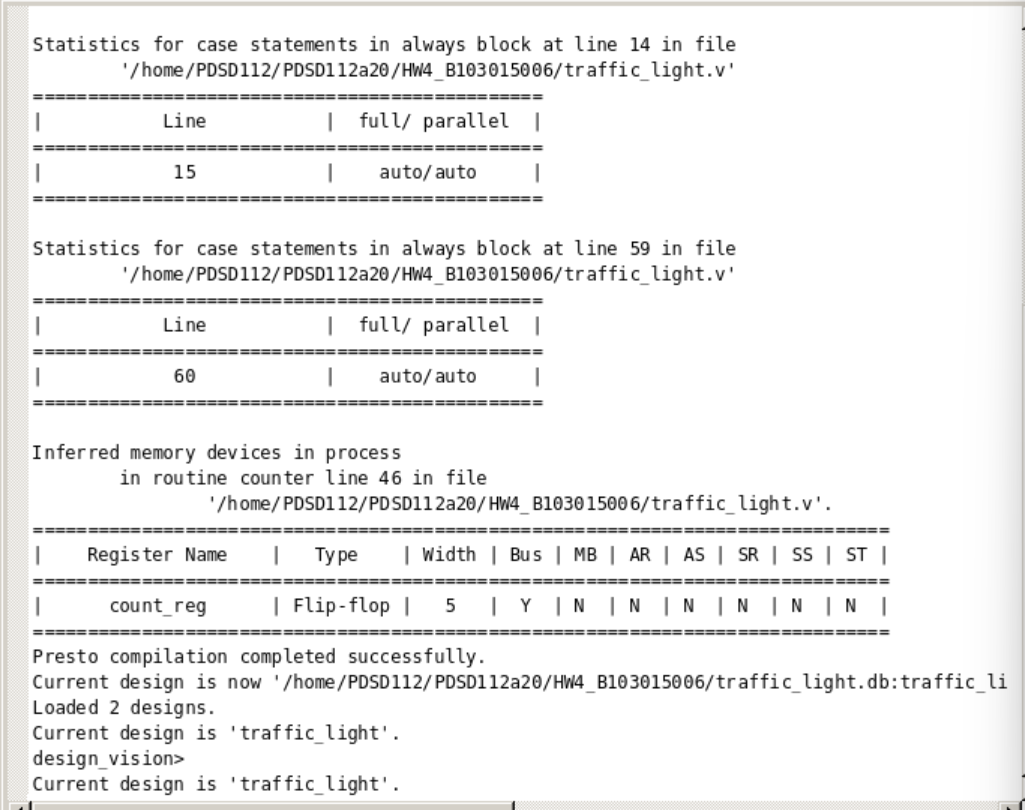
The library is correctly loaded.

1. **Read file**

****

Check if any errors or warnings appear.

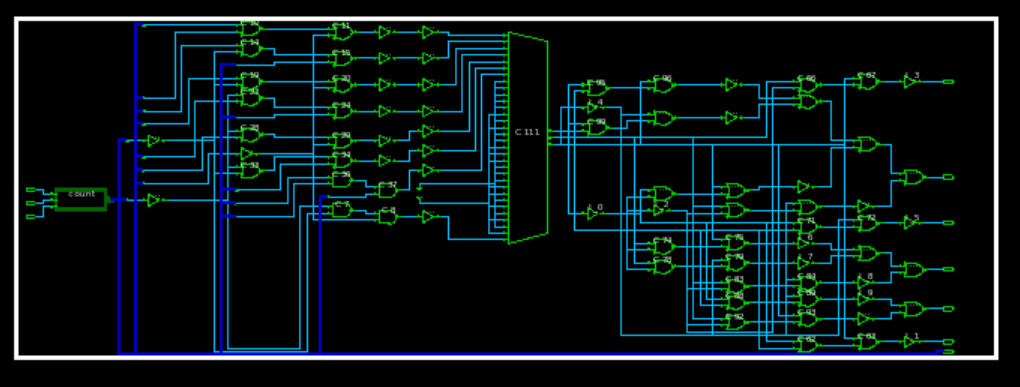
The design is correctly read without latches.



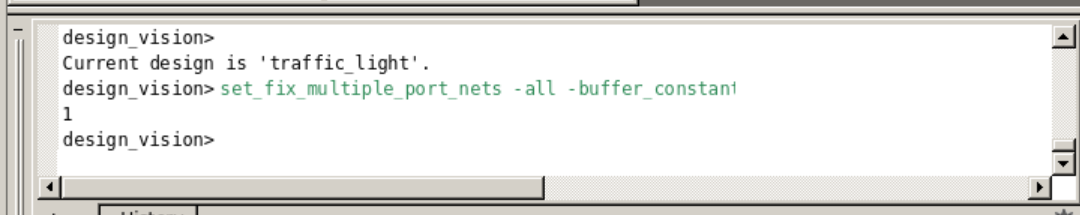
1. **Symbol View**

****

1. **Schematic View**

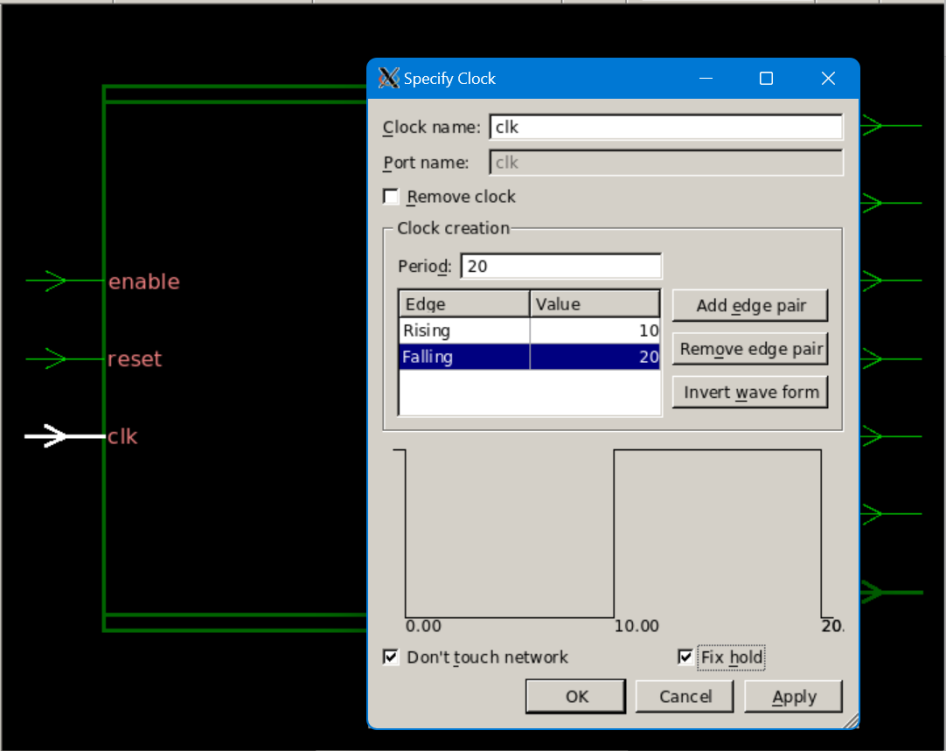
****

1. **Prevent Possible Assign Statement Problem**

****

Type ‘set\_fix\_multiple\_port\_nets -all -buffer\_constants’ in the command line.

1. **Specify Clock**

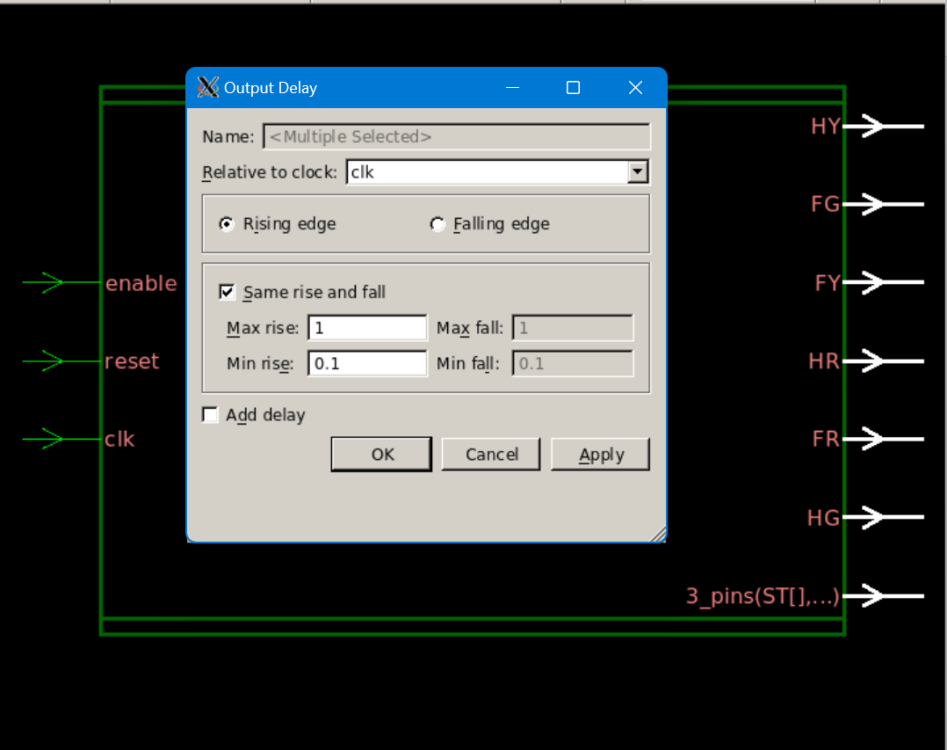
****

1. **Set Input/Output Delay**

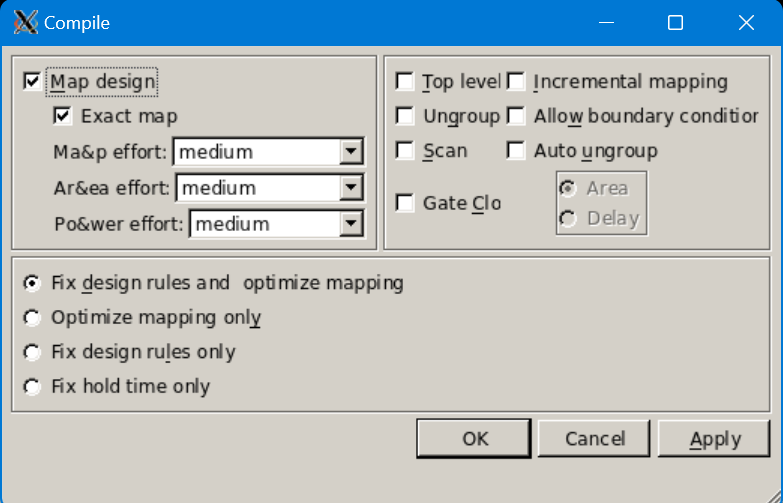
Attribute→Operating Environment

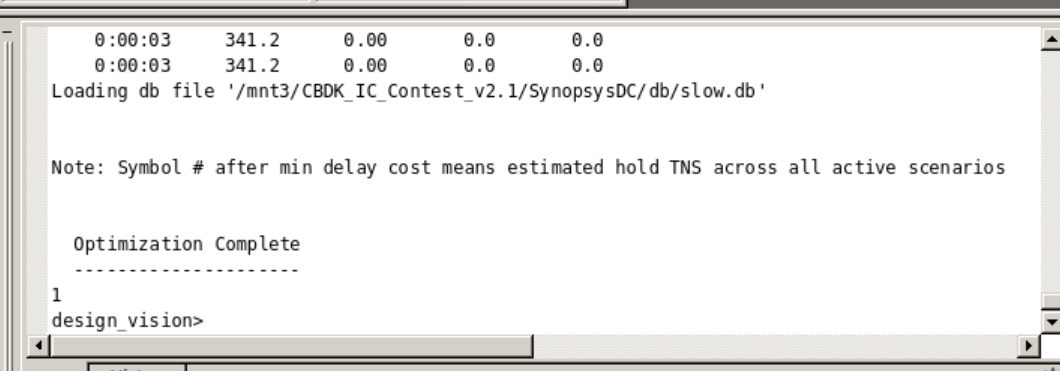
Select the input and output pins respectively and set the min and max rise time.

****

****

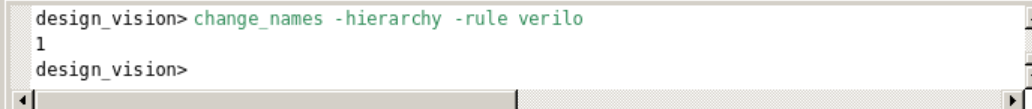
1. **Compile Design**

****

****

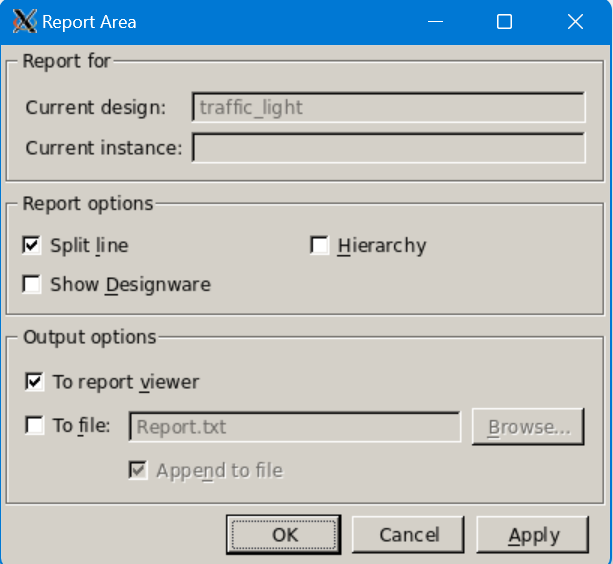
The compile process completed successfully.

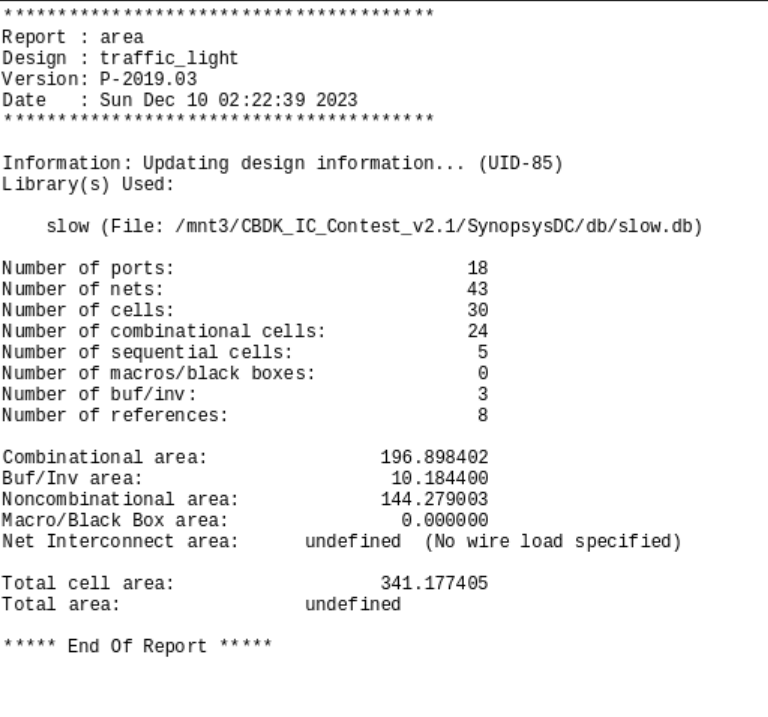
Enter ‘change\_names -hierarchy -rule verilog’ in the command line in order to remove assign in the circuit.



1. **Area Report**

Design→Report Area

****

The output result of the area report will be shown as below. ****

Unit: umum

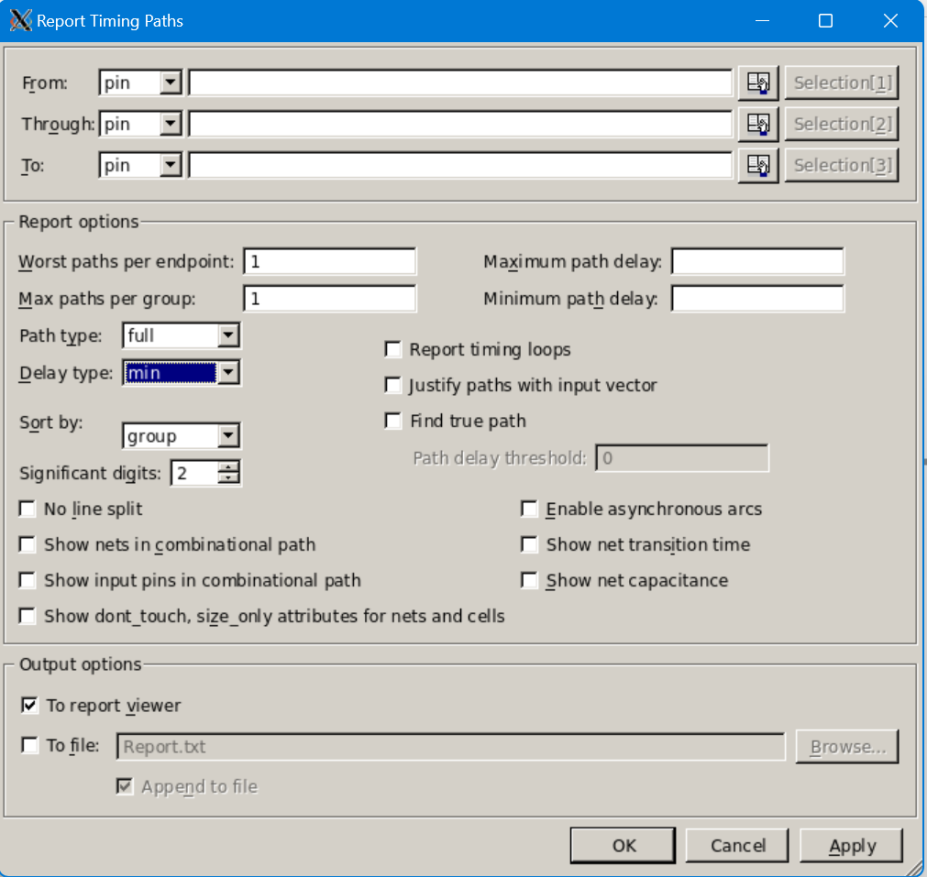
Gate count = reported area/ area of a NAND2 gate

Area of a NAND2 gate is approximately:

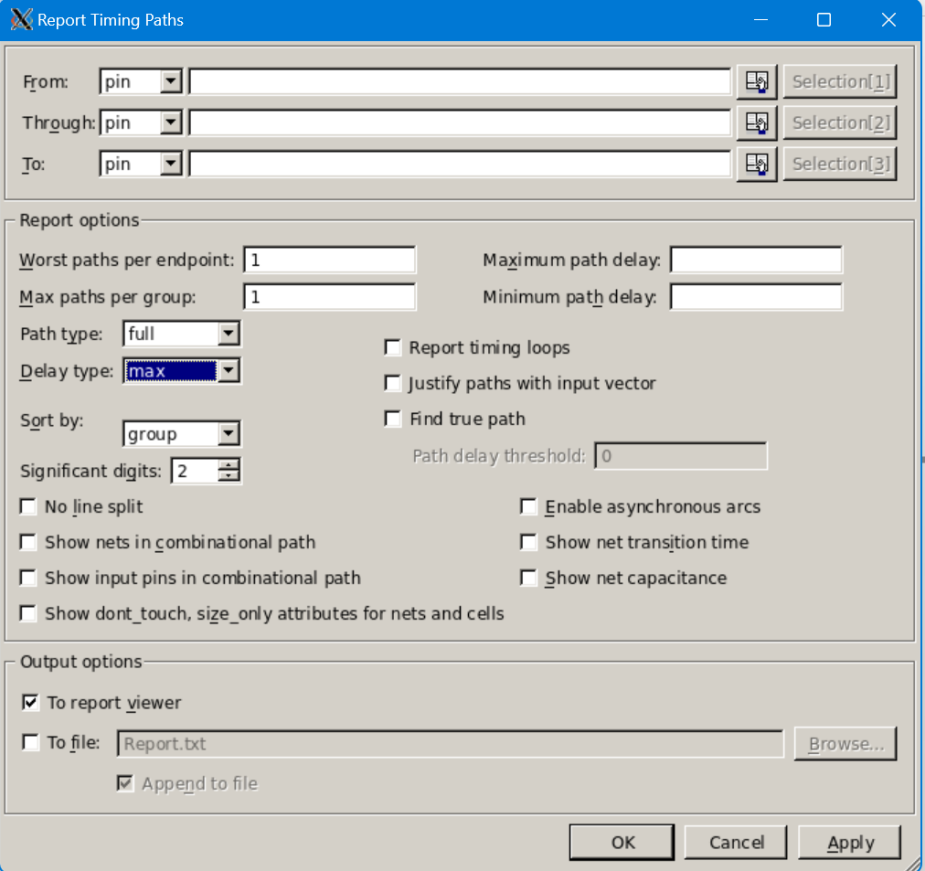
* 5 umum for a 0.13um technology
* 10 umum for a 0.18um technology

1. **Timing Report**

Timing→Report Timing Path

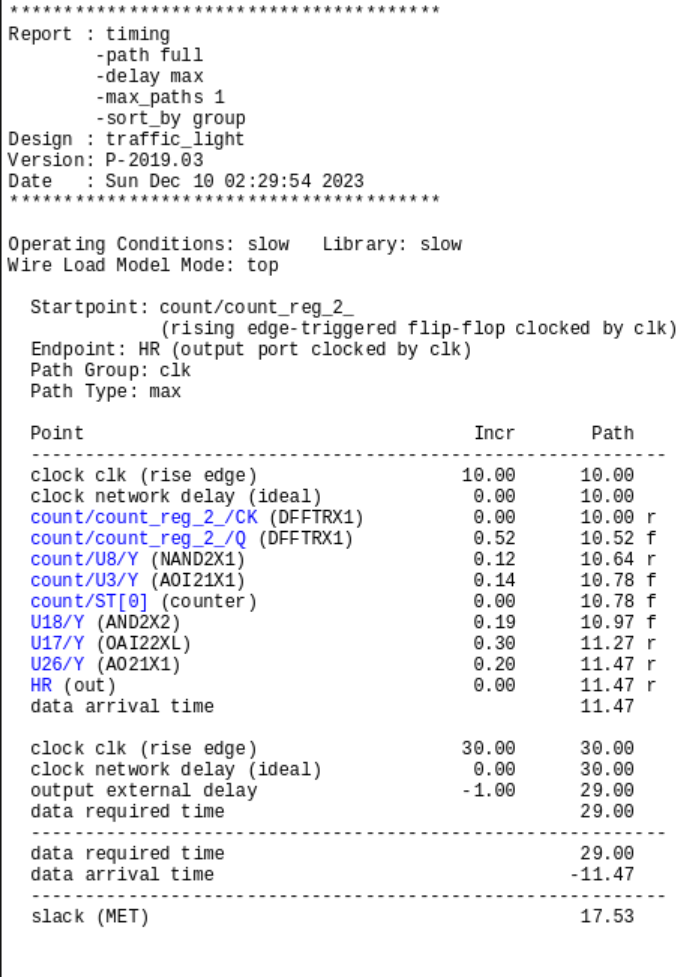
****

Set delay time to max to see the information of setup time.

****

Set delay time to min to see the information of hold time.

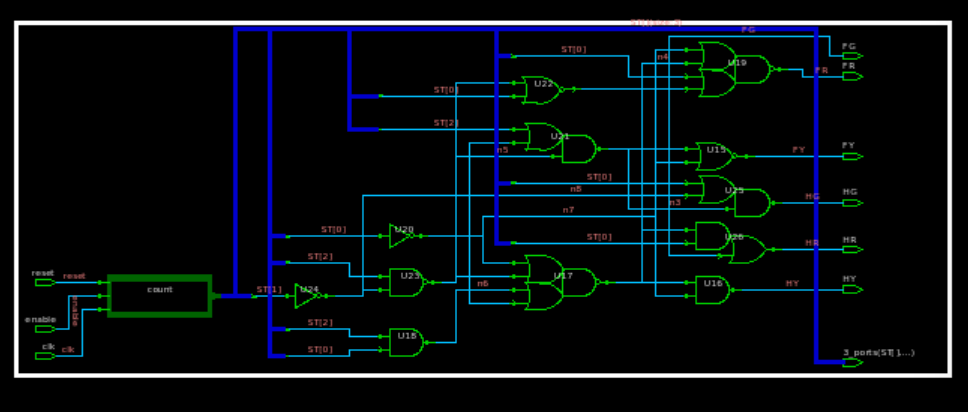
The output result of the timing report will be shown as below.

****

****

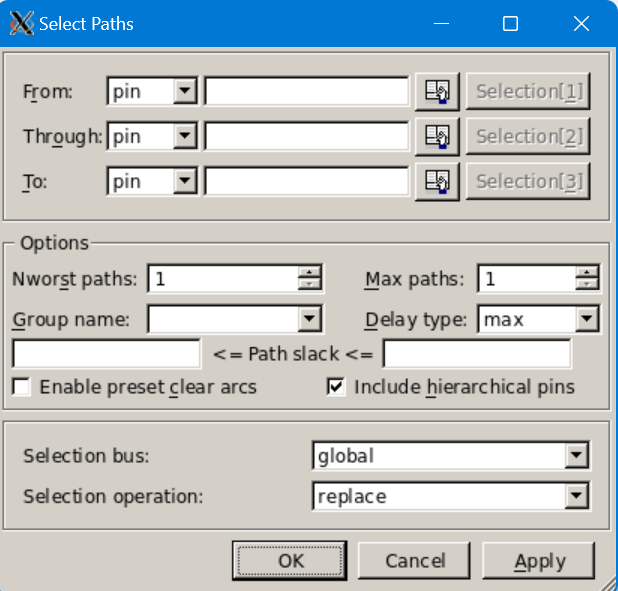
1. **Critical Path Highlight**

Following is the circuit after compiled:

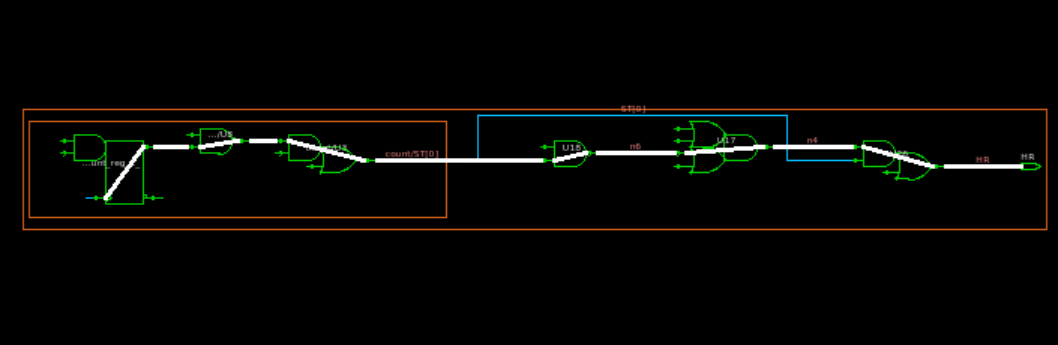


One of the methods is: Select->Paths From/Through/to

We set the delay type to max.

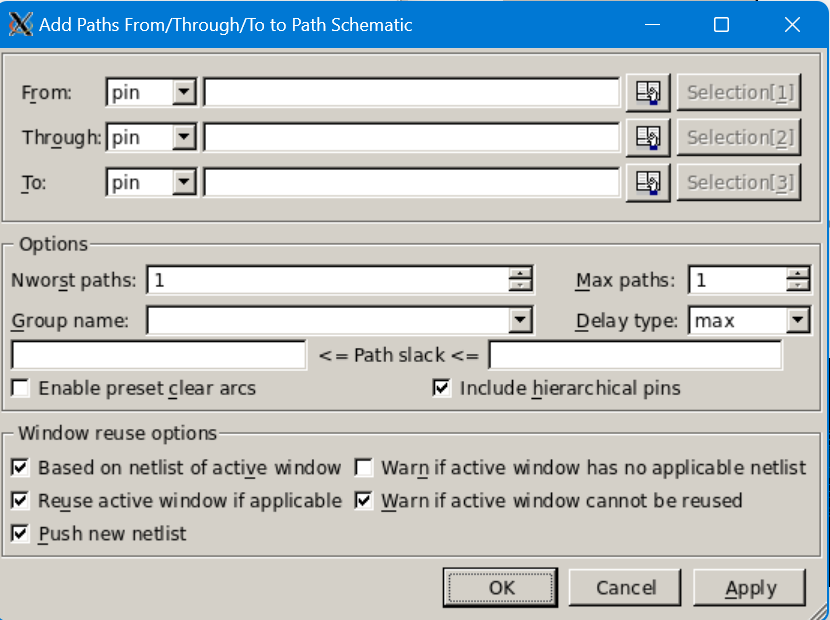


The critical path will be highlighted on the schematic window using a white line.

****

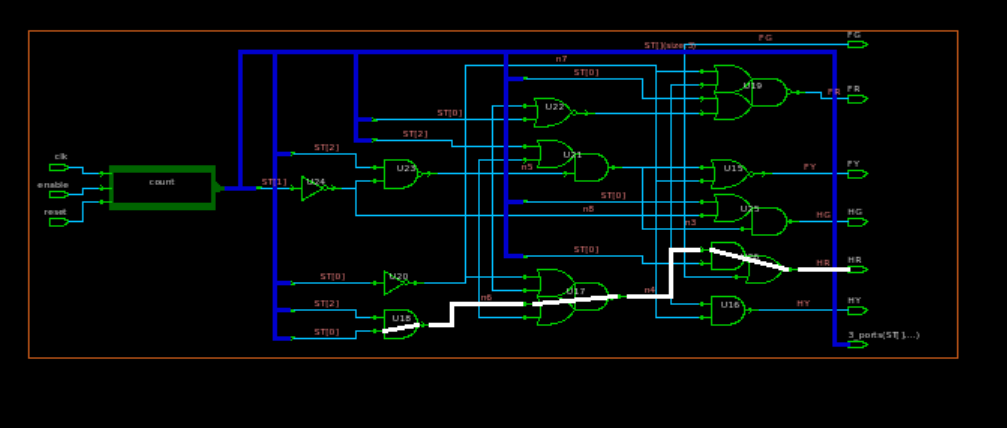
The other method is: Schematic->Paths From/Through/to

Set the delay type to max.

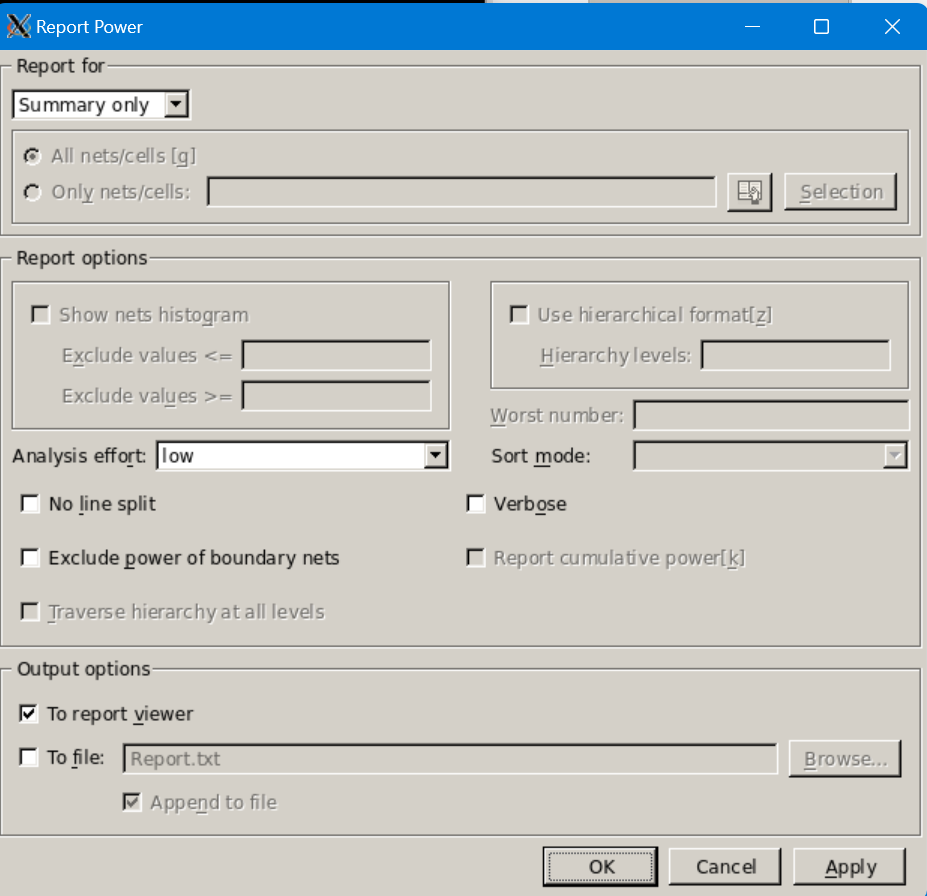


Set the delay type to max.

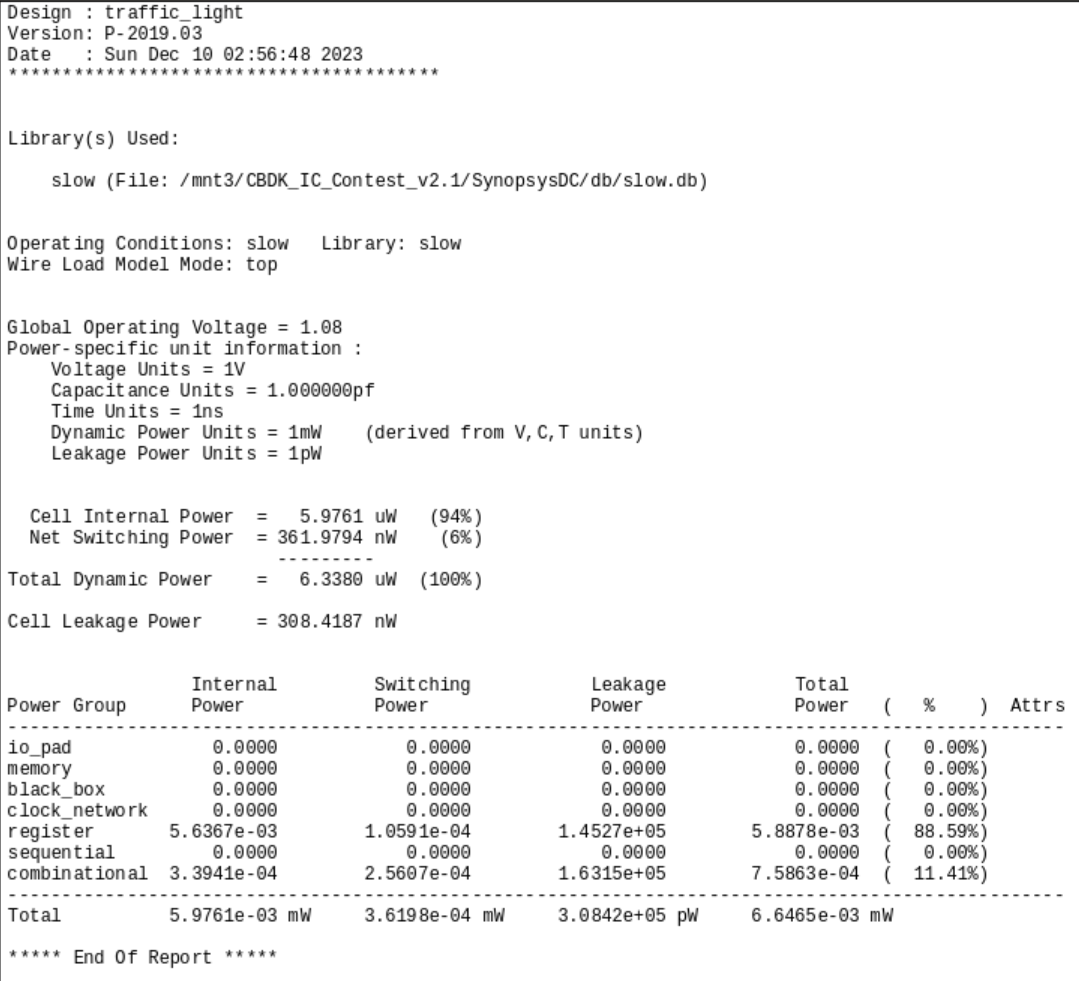
A new window will open up showing the entire critical path.

****

1. **Power Report**

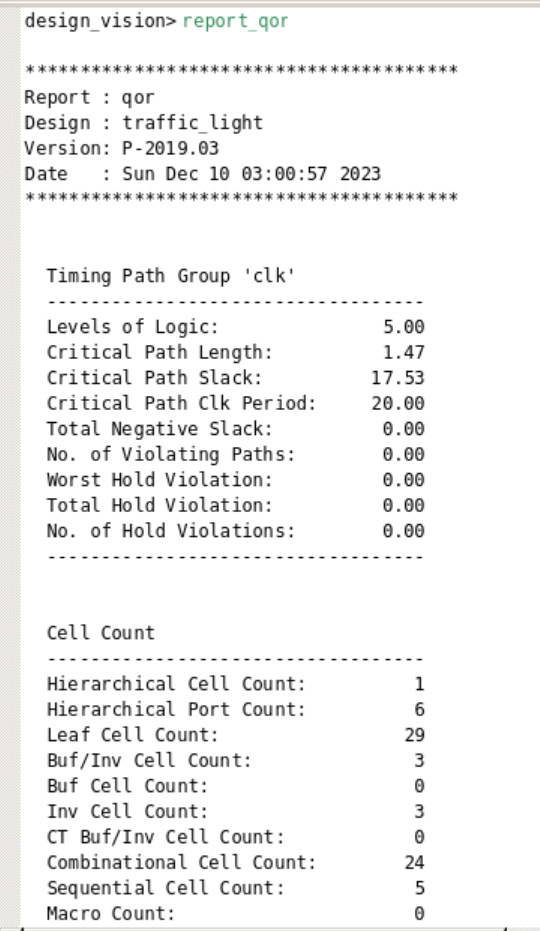
****

****

****

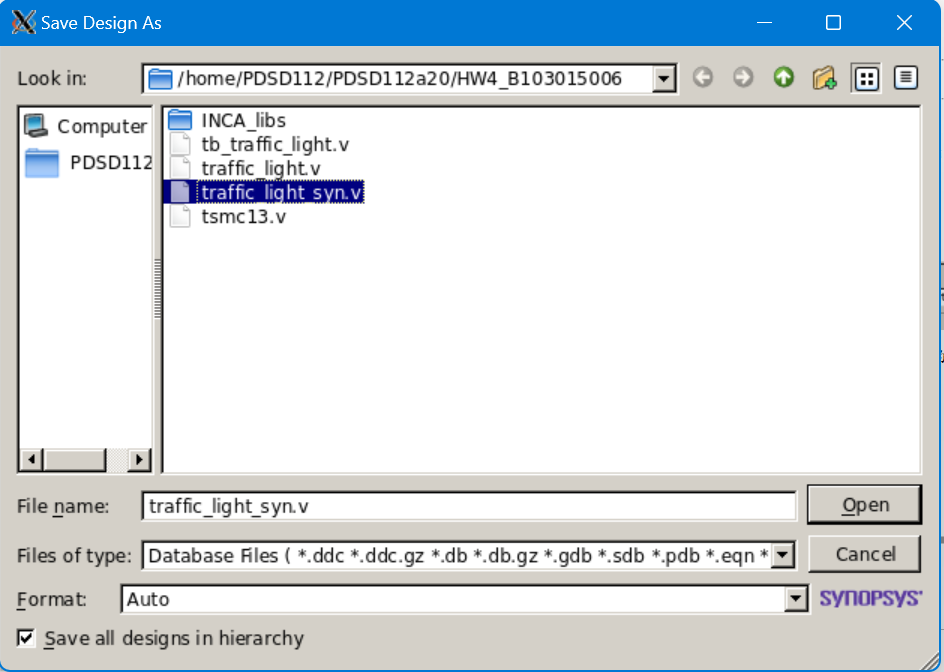
1. **Report Qor**

If we want to fast access the information of the synthesized circuit, then we can enter ‘report\_qor’ in the command line.

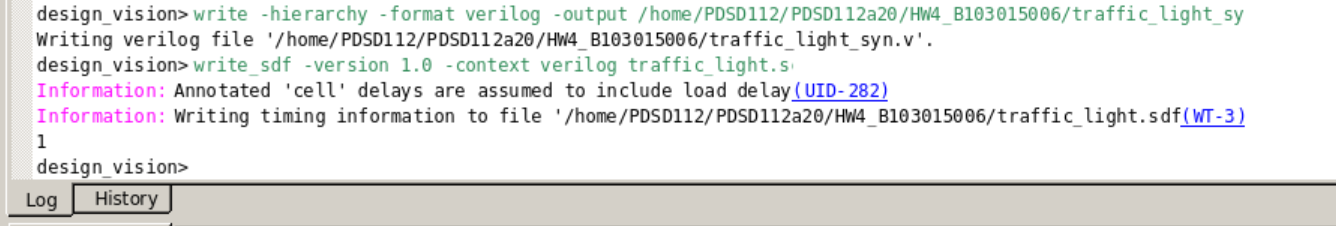


1. **Save Design Circuit and Timing Files**

****

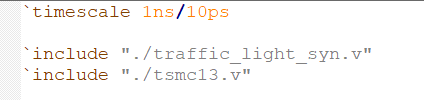
****

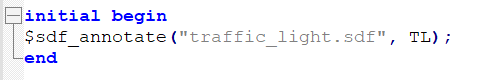
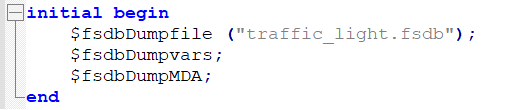
Type ‘wrie\_sdf –version 1.0 –context verilog (filename).sdf’.

****

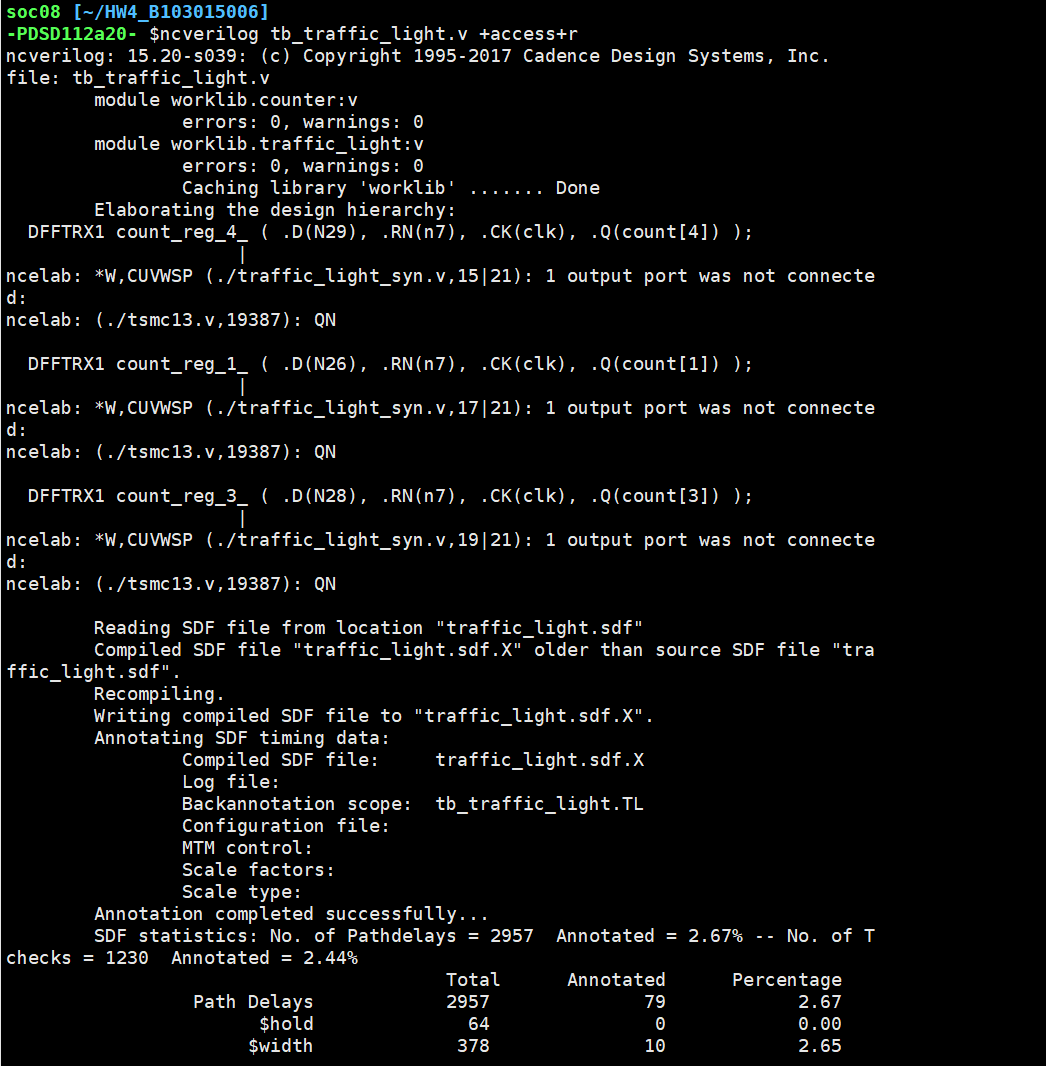
1. **Rerun Verilog Simulation**

Modify the testbench:



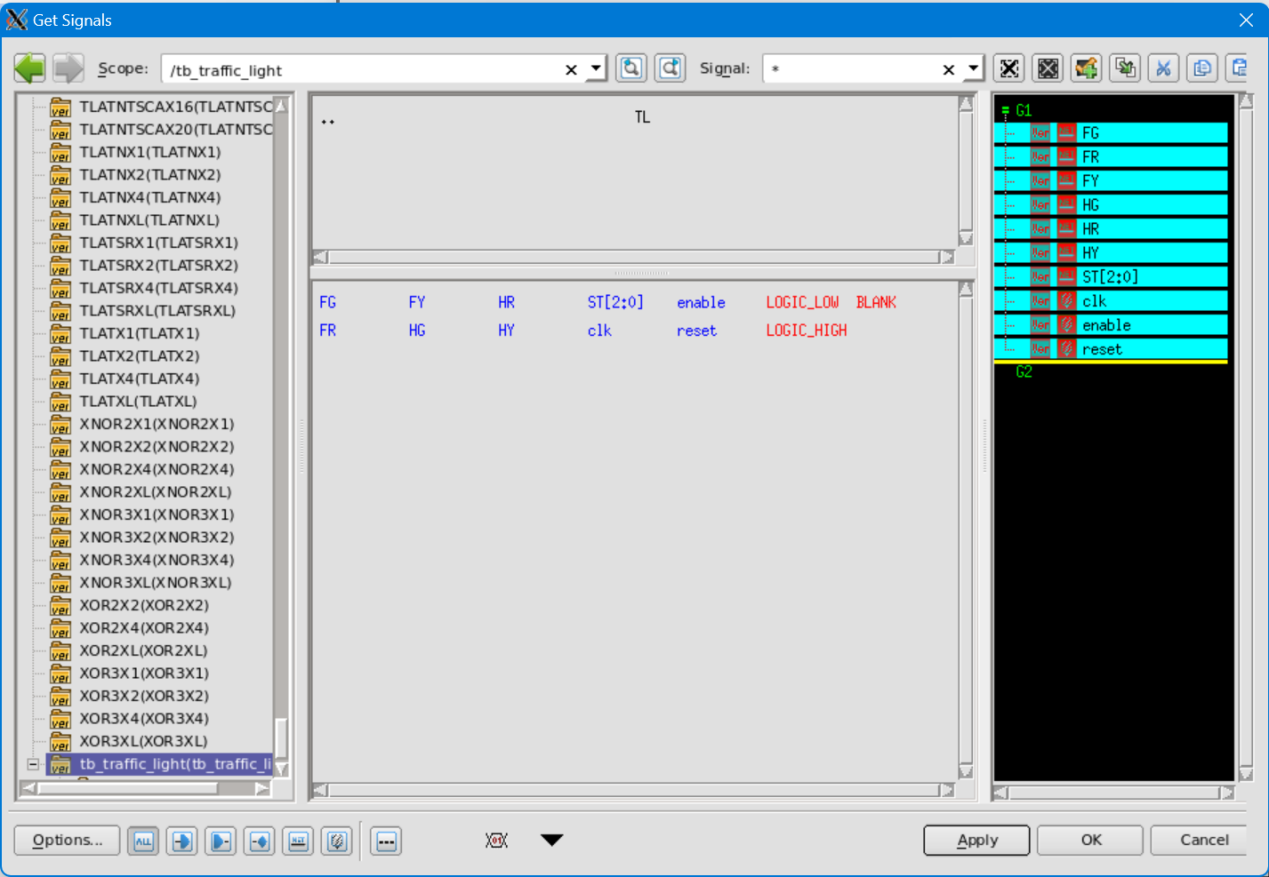
  


Enter ‘ncverilog tb.v +access+r’ in the workstation command line to initiate the simulation.

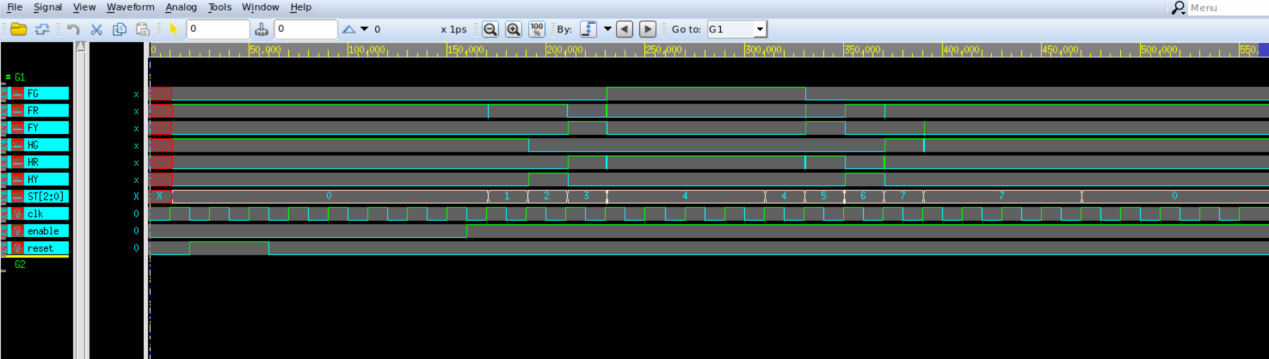


Make sure ‘Annotation completed successfully’ is in terminal.

Open nWave and find testbench module name.

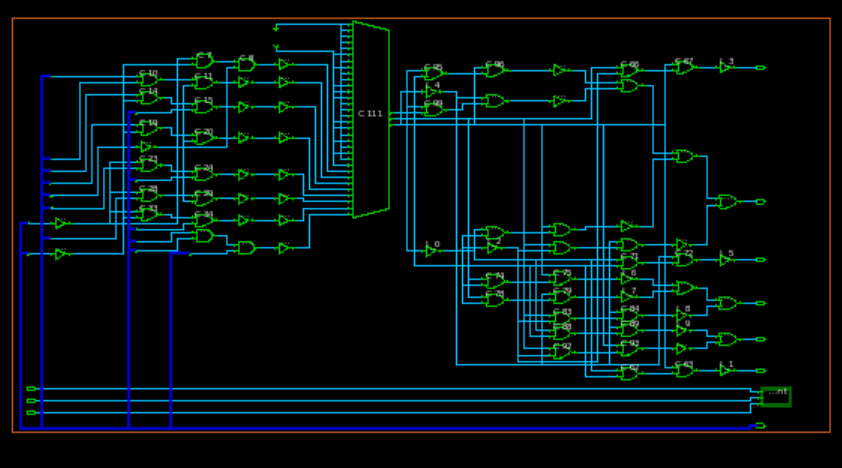


Select signals to be observed.

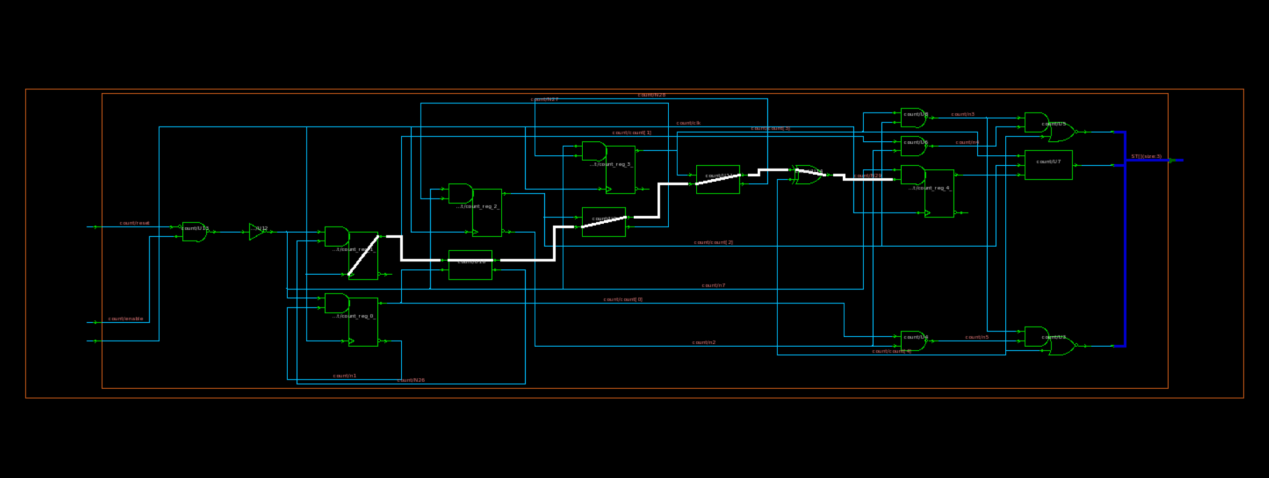


Delay effect can be observed.

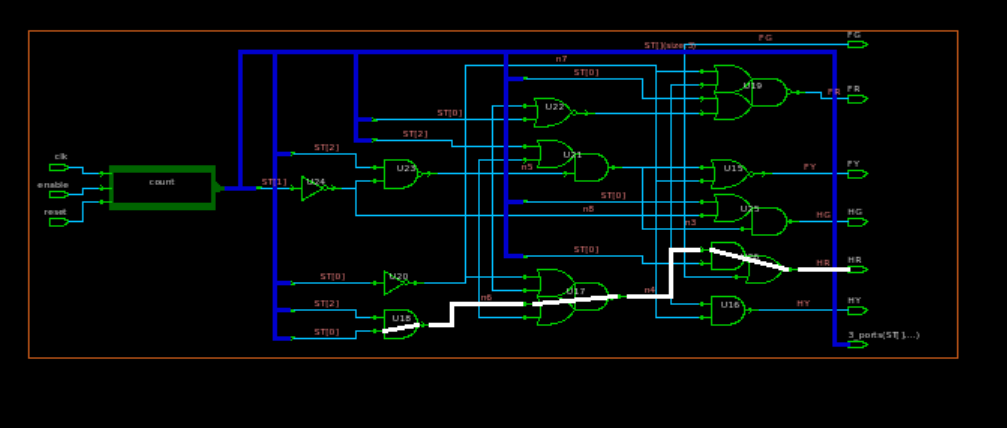
1. **Circuit Analysis**
2. **Critical Path**
3. **Before Synthesis**

****

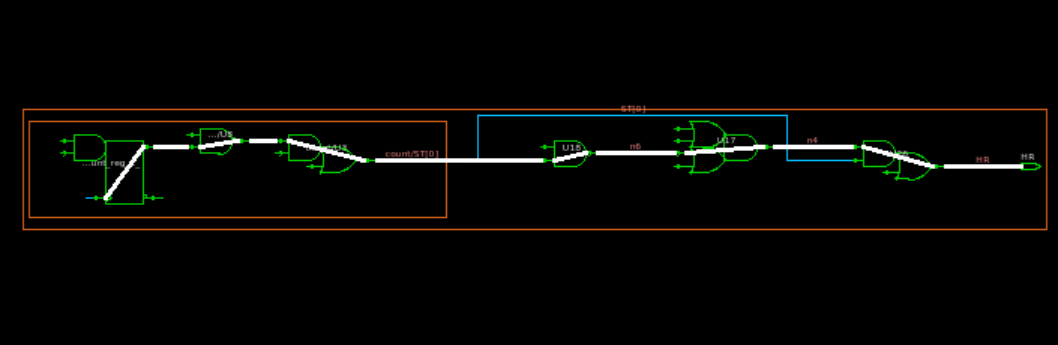
Longest path:

****

1. **After Synthesis**

****

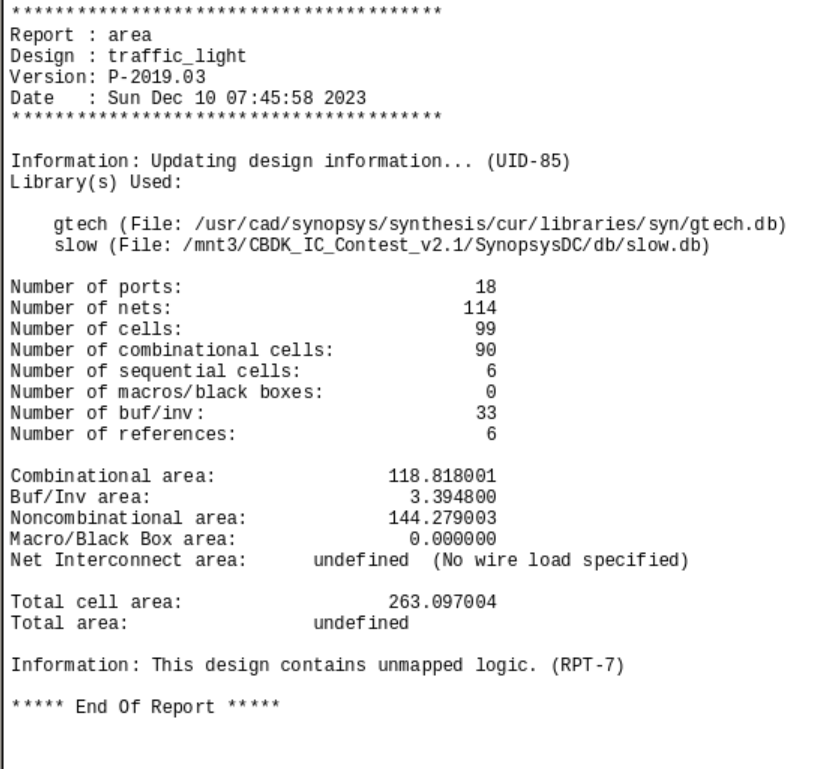
Longest path:



1. **Comparison**

There is no difference of the critical path before and after synthesis.

1. **Area**
2. **Before Synthesis**

****

Unit: 263.097004 umum

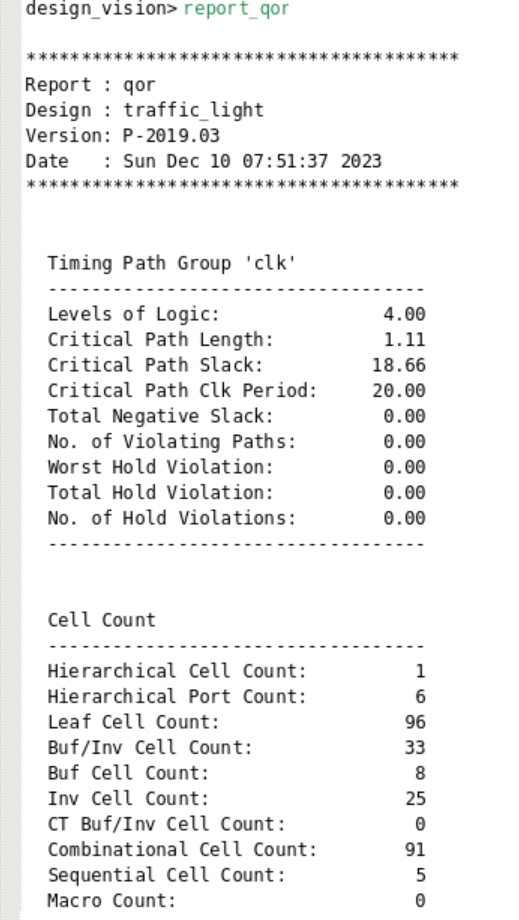
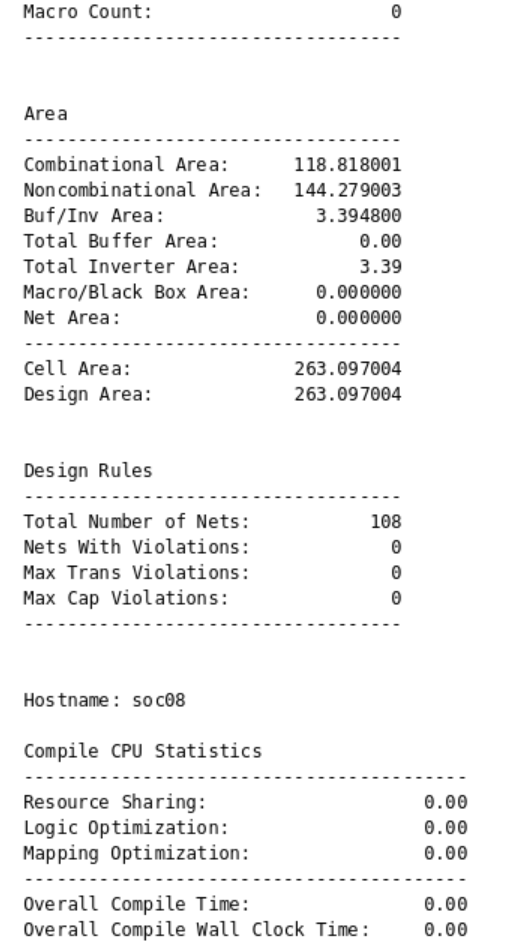
Area of a NAND2 gate is approximately:

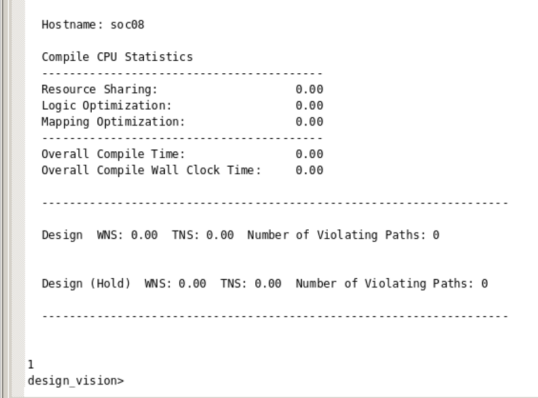
1. 5 umum for a 0.13um technology
2. 10 umum for a 0.18um technology

Gate count = reported area/ area of a NAND2 gate

1. 263.097004/5 = 52.6194008 in 0.13um technology
2. 263.097004/10 = 26.3097004 in 0.18um technology

Report Qor



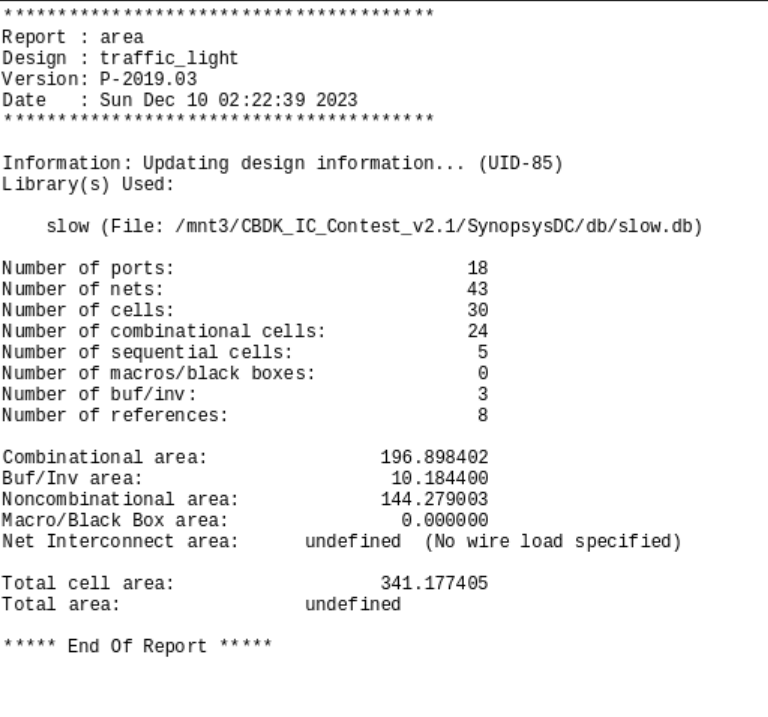
Longest path: 1.11

Sequential Cell Count: 5

Combinational Cell Count: 91

Number of logic gates layers: 4

1. **After Synthesis**

The output result of the area report will be shown as below. ****

Unit: 341.177405 umum

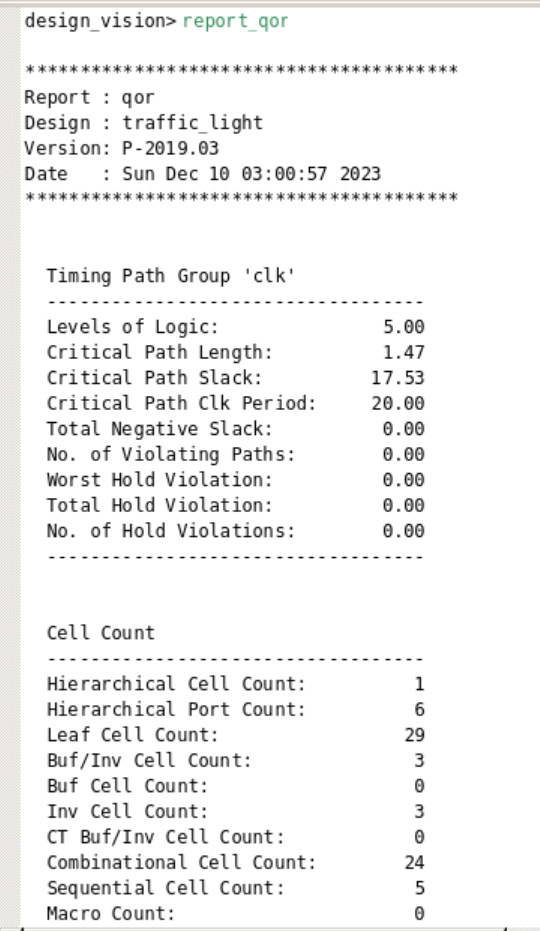
Area of a NAND2 gate is approximately:

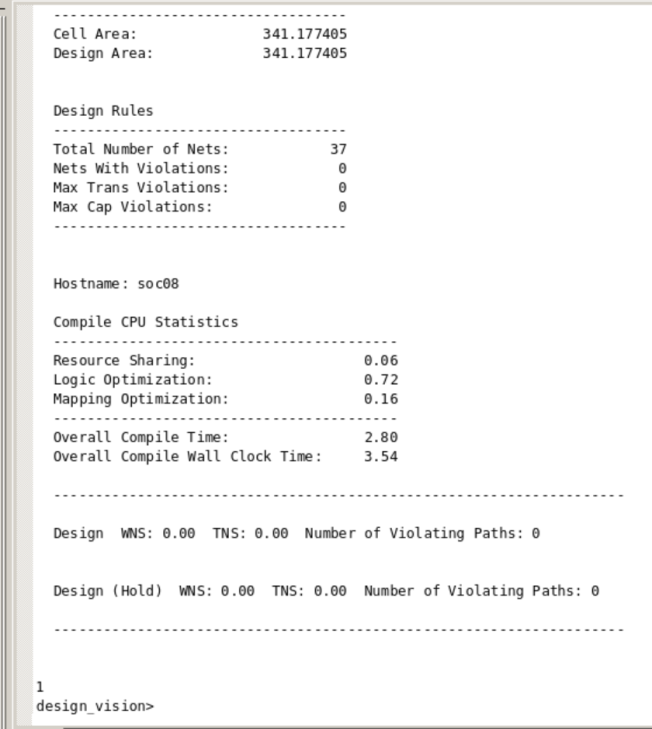
1. 5 umum for a 0.13um technology
2. 10 umum for a 0.18um technology

Gate count = reported area/ area of a NAND2 gate

1. 341.177405/5 = 68.235481 in 0.13um technology
2. 341.177405/10 = 34.1177405 in 0.18um technology

Report Qor



Longest path: 1.47

Sequential Cell Count: 5

Combinational Cell Count: 24

Number of logic gates layers: 5

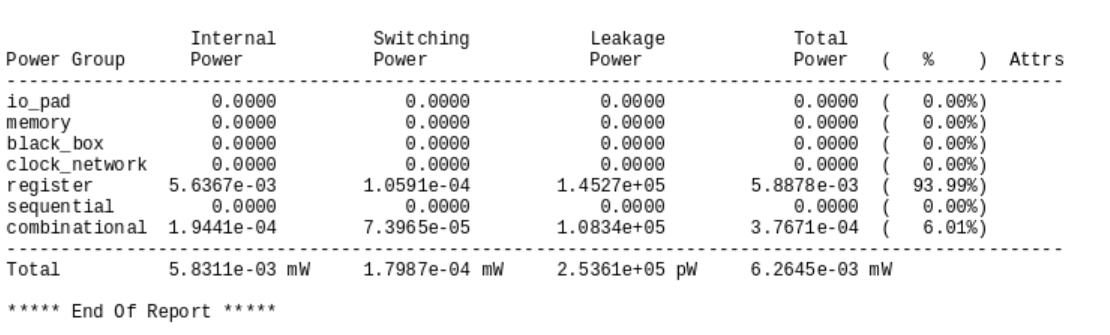
1. **Comparison**

The area after synthesis is bigger than before synthesis, which is due to the reason of delay setting.

With same number of logic gates layers and sequential cell, the circuit after synthesis has less number of combinational circuits but with a bit longer critical path that might happens in the counter also because of the input and output delay setting.

1. **Power**
2. **Before Synthesis**

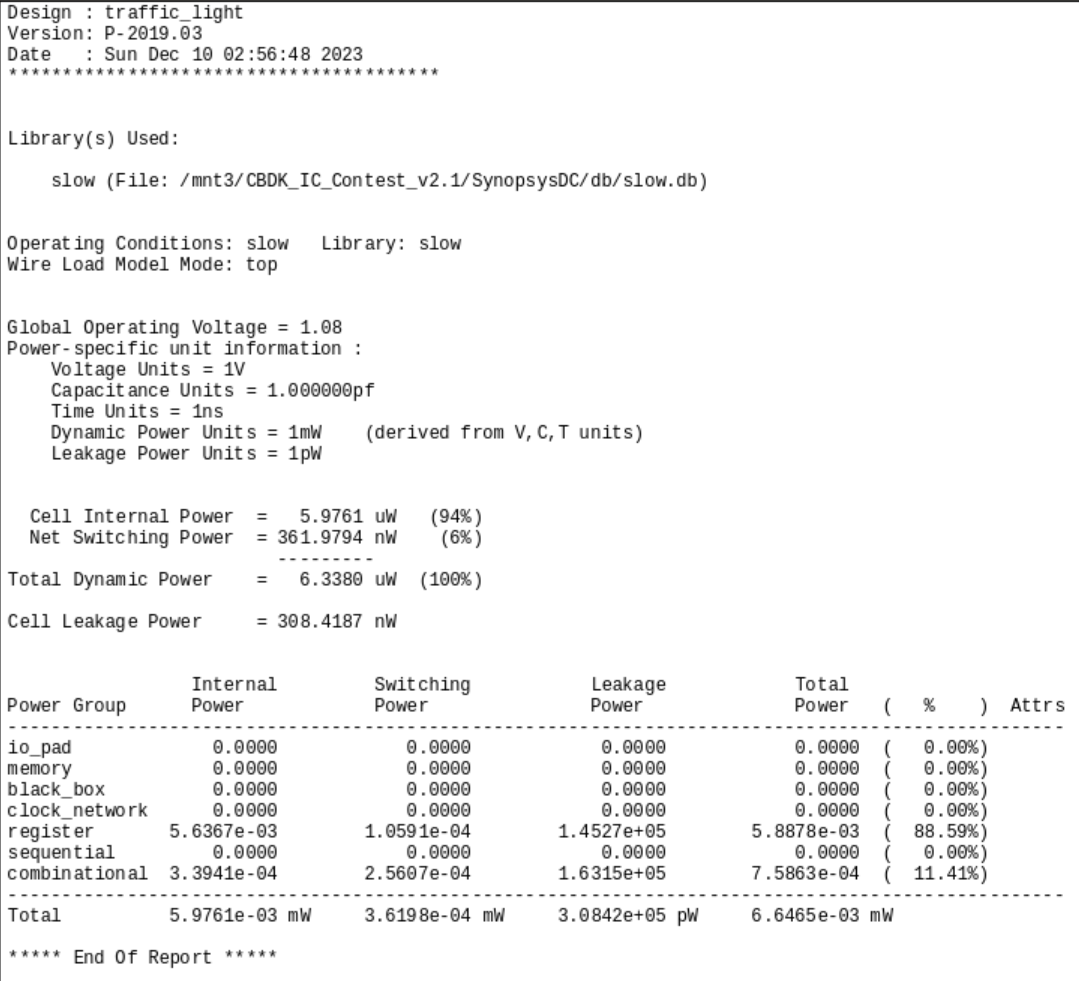
****

****

Total power: 6.2645e-03 mW

1. **After Synthesis**

****

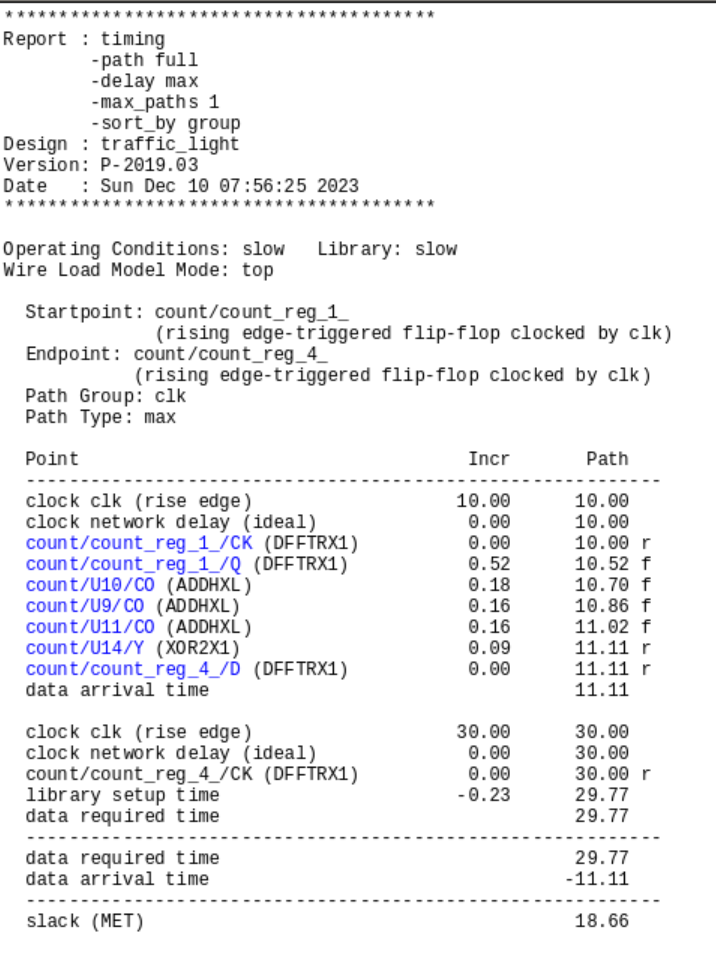
****

Total power: 6.6465e-03 mW

1. **Comparison**

The circuit after logic synthesis has larger power consumption.

1. **Delay**
2. **Before Synthesis**

****

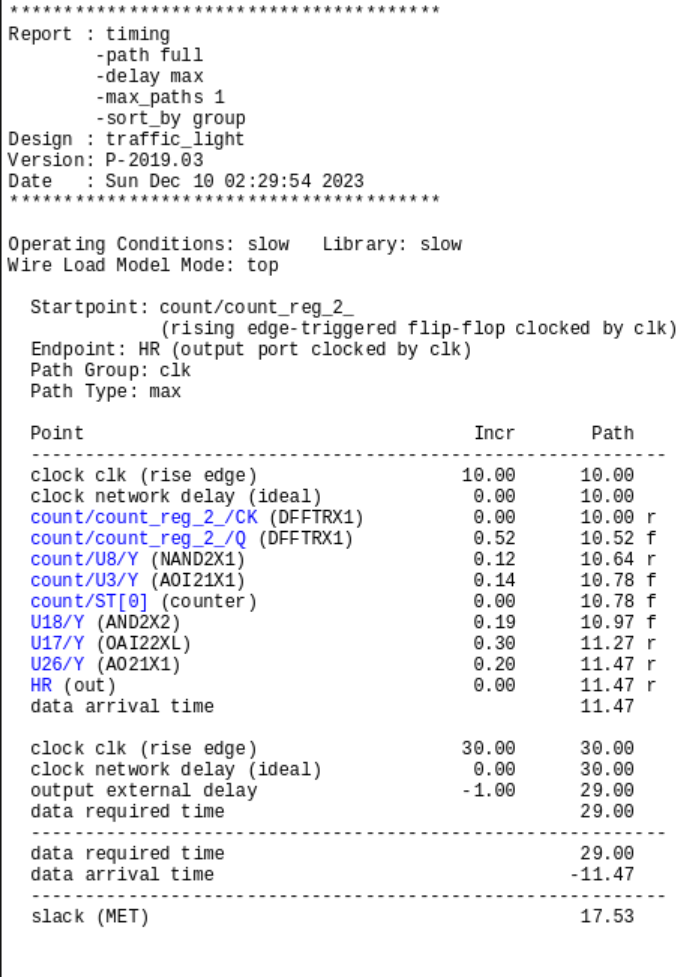
Delay type: max

Data arrival time: 11.11(ns)

Setup time: 29.77(ns)

Slack: 18.66(ns)

1. **After Synthesis**



Delay type: max

Data arrival time: 11.47 (ns)

Setup time: 29.00(ns)

Slack: 17.53(ns)

1. **Comparison**

The slack after synthesis has a smaller slack compare to the one before synthesis.

1. **Testbench and Verification**

Following are the main function of our circuit, we check each function separately and if they all functioned correctly, then it shows that the circuit can normally work.

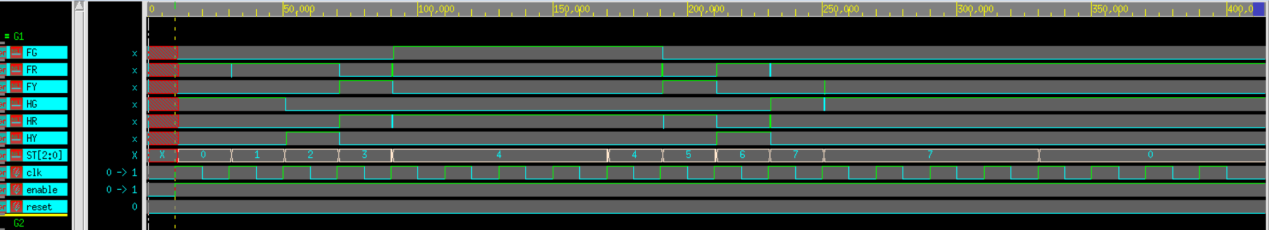
1. **When there are cars on farm road, S0->S1**



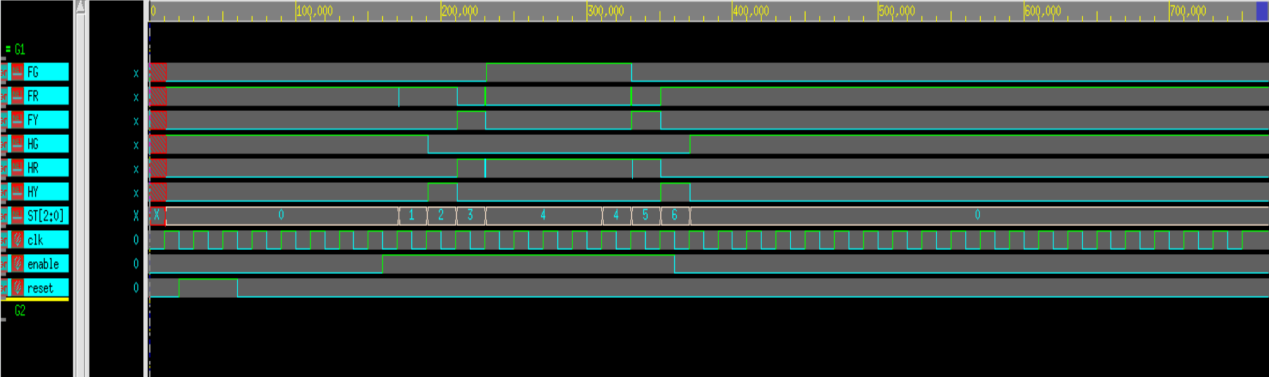
When enable(car detect)=1, the state will change from 0 to 1 after the posedge of clk arrive.

And the states continue normally.

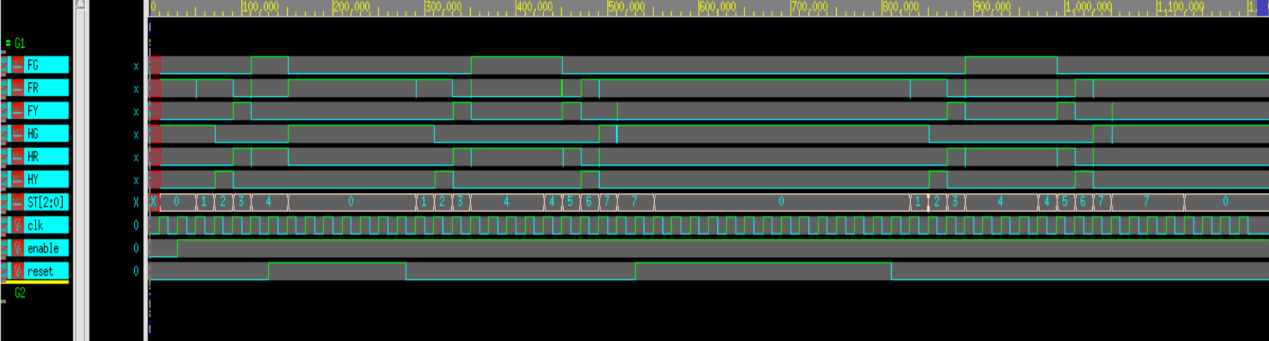
1. **After one turn of the light conversion, if there still have cars on the road, S7->S0**



1. **After the light conversion, if there is no car on the road anymore, current state->S0**



1. **When reset=1, current state-> S0**



1. **Discussion and Reflection**
2. **Discussion**

The reason for the delay effect arises from the difference in the time it takes for signals to reach the output port. In other words, before the signal on the longest path completes its calculation and reaches the output port, signals from other paths with shorter delays will already have arrived.

This causes continuous fluctuations in the output values. In the context of simulating the waveform in the traffic light, this delay effect resembles a signal glitch.

1. **Reflection**

I feel a big difference between writing HW1 and HW2 with this HW4. Writing combinational circuit and writing sequential circuit has huge difference in the data type and its way of sending signals, to write a sequential circuit, one should be very clear about the circuit they are designing. I got confused with my code several times in the designing process, however, thanks to the help of error message, I figured out my mistake immediately and successfully complete the work.

After having the experience of designing the traffic light controller, I fell that I can really design something that can be apply in the real world, it is a cool thing to be honest, however, this work is really time consuming, I spent most of my time not on writing code but on logic synthesis…. Can’t imagine how much effort will it take on our final project.