|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  | | |  |  |
|  |  | **TRƯỜNG ĐẠI HỌC BÁCH KHOA**  **ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH KHOA ĐIỆN – ĐIỆN TỬ**  🙞**···☼···**🙜    **BÁO CÁO BÀI TẬP LAB3:**  **LOGIC SYNTHESIS AND VLSI FRONT-END FLOW**  **Môn: THIẾT KẾ VI MẠCH**  Giảng Viên Hướng Dẫn: TRẦN HOÀNG QUÂN | | |  |  |
|  | | |
|  | | |  |
|  |  |  | | |  |  |

**MỤC LỤC**

1. [LOGIC SYNTHESIS 3](#_bookmark0)

[Lab 2: 3](#_bookmark1)

* 1. [Synthesize folder. 3](#_bookmark2)
  2. [Simulation 7](#_bookmark3)
  3. [High level Arichitecture of the design 8](#_bookmark4)
  4. [Report 9](#_bookmark5)
  5. [Waveform of RTL and netlist simulation 14](#_bookmark6)

1. [VLSI FRONT-END FLOW 16](#_bookmark7)

[Lab 1:Thiết kế 4-bit counter từ các T-Flip Flop. 16](#_bookmark8)

* 1. [Verification plan: 17](#_bookmark9)
  2. [Code tạo file testbench.v 18](#_bookmark10)
  3. [Kết quả mô phỏng 19](#_bookmark11)

[Lab 2: ALU 20](#_bookmark12)

1. [Code VHDL 20](#_bookmark13)
2. [Code testbench 21](#_bookmark14)
3. [Verification plan 24](#_bookmark15)
4. [Simulation 24](#_bookmark16)

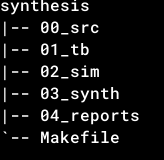
# LOGIC SYNTHESIS

## Lab 2:

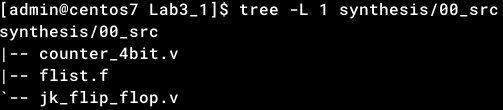
Design and synthesize “4-bit counter” specified in the previous RTL and verification lab, use verification plan in previous lab.

## Synthesize folder.

- Danh sách các file trong thư mục “Synthesis”:



+ Folder 00\_src chứa code RTL. Với file flist.f là danh sách tất cả các file code của thiết kế.

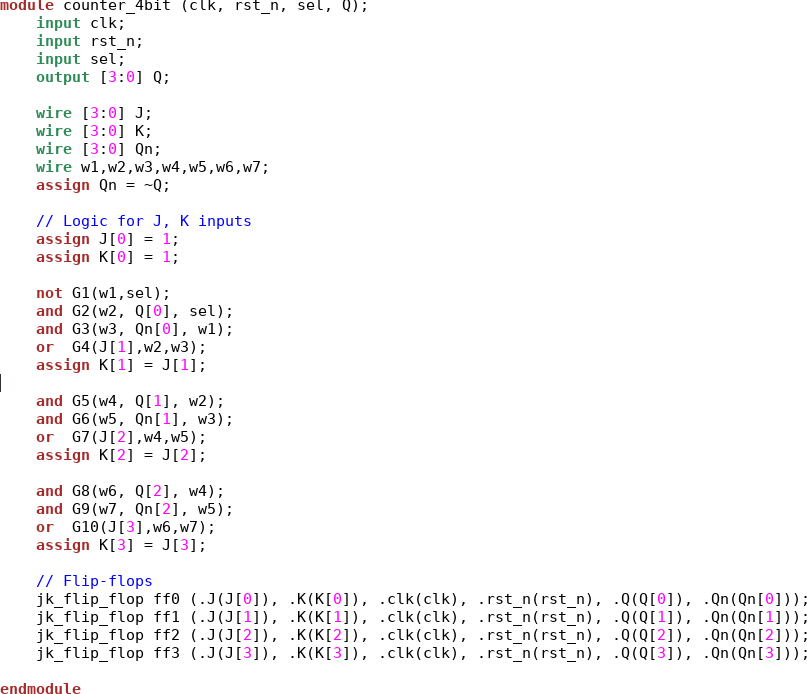


Bên trong file flist.f, đường dẫn của các file code được liệt kê từ module thấp đến module cao.



File counter\_4bit.v là top-module của thiết kế synthesis mà bài làm sẽ đăng ký tất cả các ngõ vào và ngõ ra của top-module với thiết kế được khởi tạo.

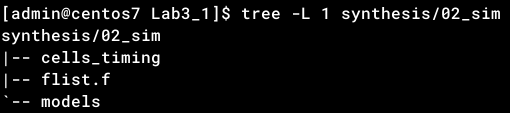
* + - Code Verilog top-module counter\_4bit.v



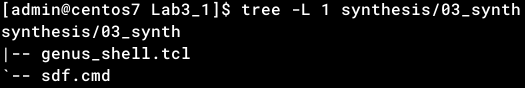
+ Folder 01\_tb sẽ chứa file testbench.v dùng để kiểm tra code RTL và xác minh synthesized netlist.

A black background with white text  Description automatically generated

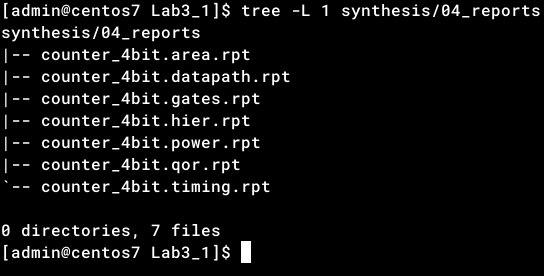
+ Folder 02\_sim chứa các thư viện được khởi tạo bên trong synthesized netlist. Và file flist.f liệt kê tất cả các đường dẫn.



+ Folder 03\_synth chứa synthesis script genus\_shell.tcl để điều khiển toàn bộ quá trình synthesis, file sdf.cmd sẽ cấu hình cho mô phỏng nếu cần mô phỏng toàn bộ thời gian của netlist.



+ Folder 04\_report chứa tất cả các báo cáo liên quan đến thiết kế. Ta có thể kiểm tra tất cả các thông tin cần thiết để đánh giá và phân tích thiết kế.



+ Makefile là tập lệnh chính dùng tự động mọi thao tác trong quá trình làm việc thay vì gõ từng dòng lệnh vào Terminal.

Dưới đây là tập lệnh Makefile trong bài lab này:

################################################## #

#QuocThai

#HCMUT LAB 3 counter\_4bit #

##################################################

PHONY: syn syn: clean

genus -f ./03\_synth/genus\_shell.tcl ##################################################

#GUI = -gui

tb\_file := 01\_tb/testbench.v

PHONY: sim sim:

xrun $(GUI) +xm64bit -sv \

-vlogext .sv \

-f 00\_src/flist.f \

$(tb\_file) \

-timescale 1ns/10ps \

+access+rcw ##################################################

#GUI = -gui

delay\_mode := -delay\_mode punit

#sdf\_file := -sdf\_cmd\_file 03\_synth/sdf.cmd netlist := 03\_synth/\*\_gate.v

test\_file := 01\_tb/testbench.v

PHONY: verify verify:

xrun $(GUI) +xm64bit -sv \

$(netlist) \

$(test\_file) \

-vlogext .sv \

-f 02\_sim/flist.f \

$(sdf\_file) \

$(delay\_mode) \

-timescale 1ns/10ps \

+access+rcw

##################################################

PHONY: report report:

@echo "with Frequency at: "

@grep "set FREQ\_GHz" 03\_synth/genus\_shell.tcl @grep "Path " 04\_reports/\*.timing.rpt

##################################################

PHONY: clean clean:

@rm -rf genus.cmd genus.log xrun.history xrun.key xcelium.d \*.shm

##################################################

PHONY: clean-all

clean-all: clean-rpt clean-syn

@rm -rf fv genus.cmd genus.log xrun.history xrun.log xrun.key counter\_4bit.sdf.X xcelium.d \*.shm

##################################################

PHONY: clean-rpt

clean-rpt:

@rm -rf 04\_reports/\*.rpt

##################################################

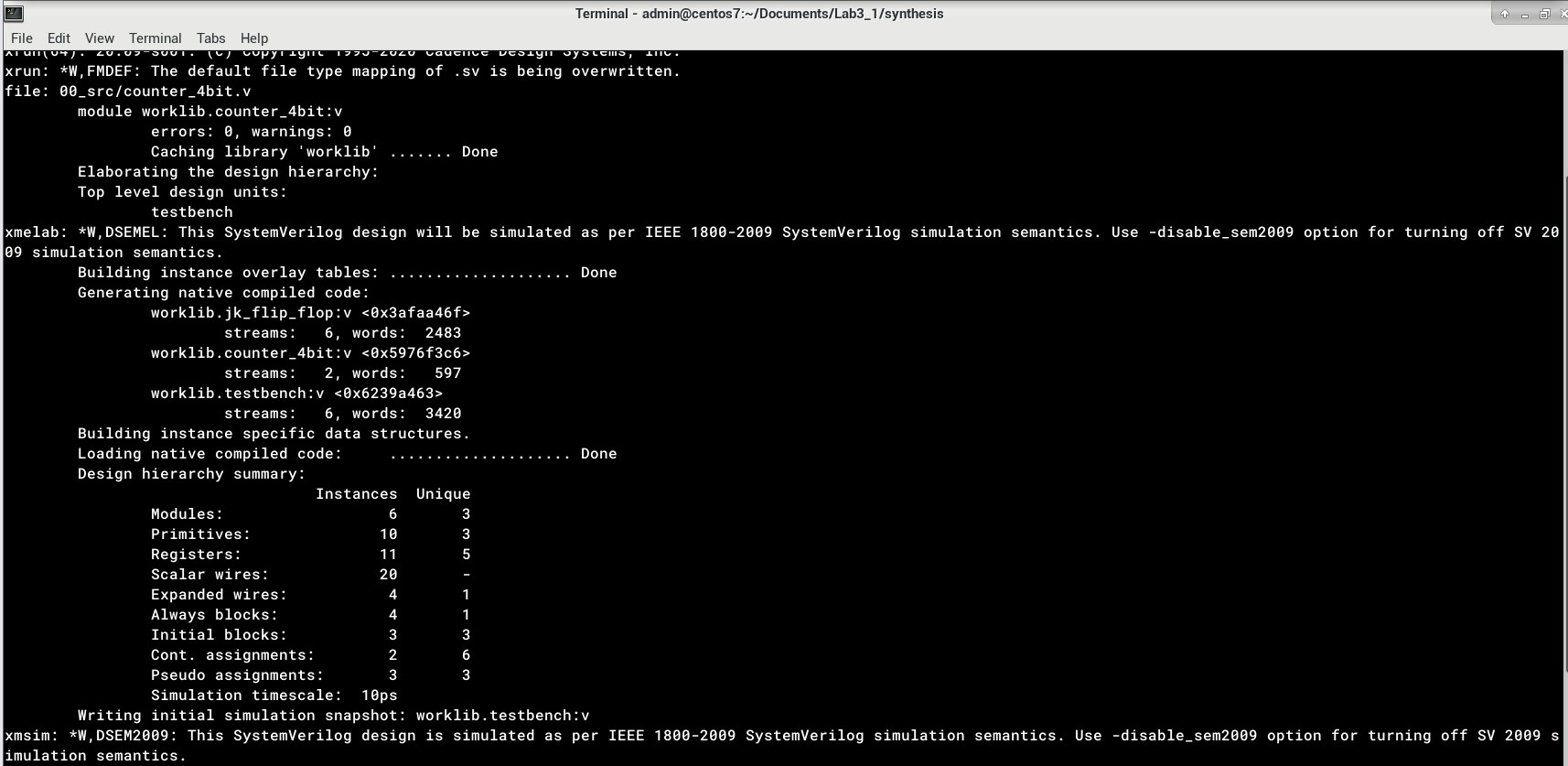
PHONY: clean-syn clean-syn:

@rm -rf 03\_synth/\*.v 03\_synth/\*.sdf

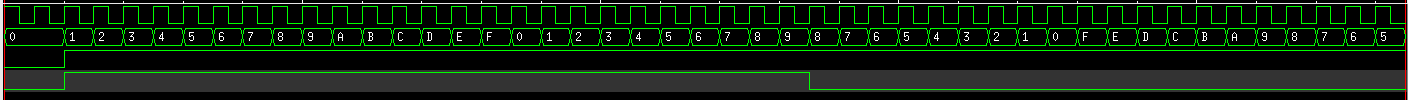
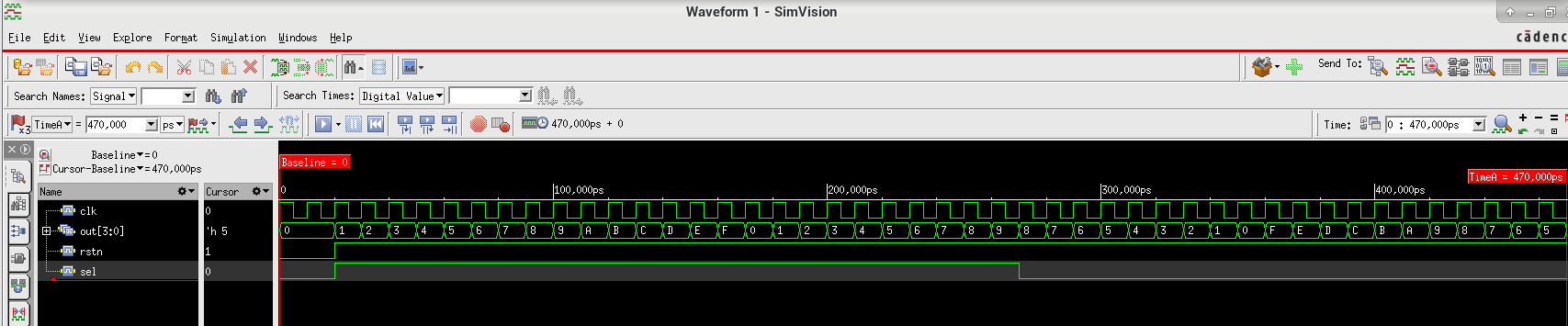
##################################################

## Simulation

* Result make sim:



* Kết quả mô phỏng:

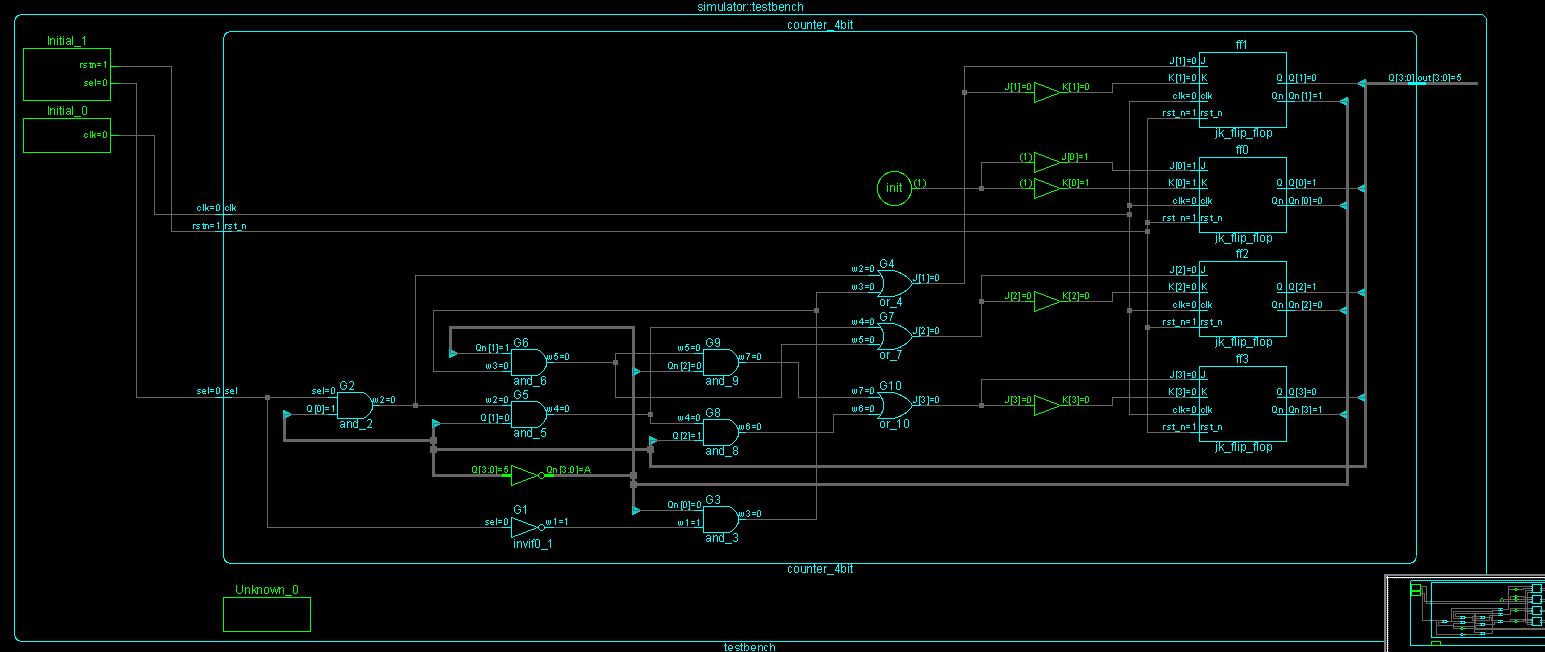


Verification plan:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Section | Item | Descreption | Testcase name | Owner | Status |
| 1 | Reset | When rst\_n is asserted, the output is 0, when it de- asserts, the output start to count up each positive edge  of clock | cnt\_rst\_test | Thai | PASS |
| 2 | Max count | When output is 4’b1111 and the counter is counting up, the next positive edge clock  output will be 4’b0000 | cnt\_max\_test | Thai | PASS |
| 3 | Min count | When output is 4’b0000 and the counter is counting down, the next positive edge  clock output will be 4’b1111 | cnt\_min\_test | Thai | PASS |

## High level Arichitecture of the design

- Schematic của netlist:

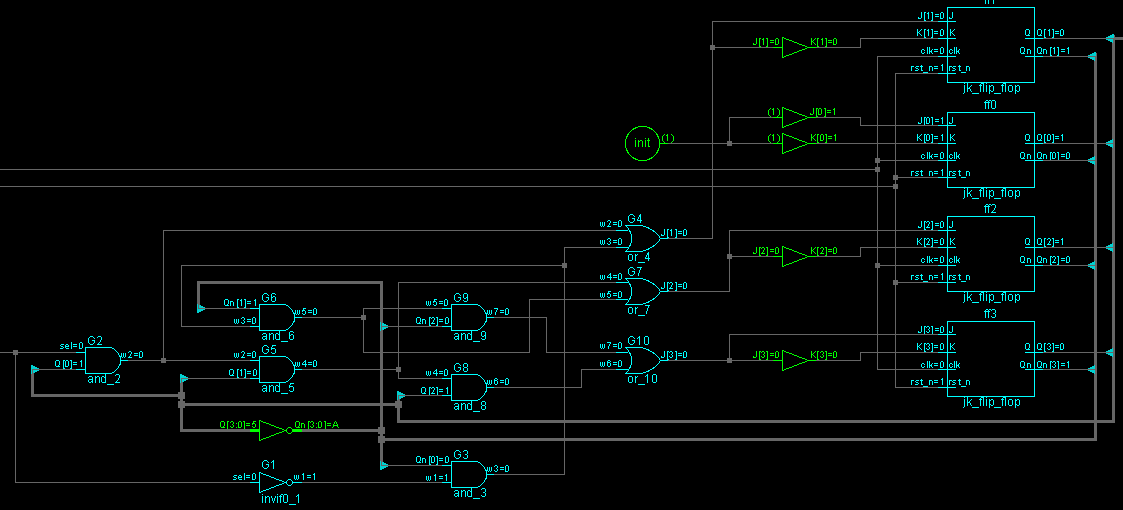


- Mô tả ngõ vào/ra:

|  |  |  |  |
| --- | --- | --- | --- |
| Signal | Width | Type | Description |
| Clk | 1 | input | Clock signal |

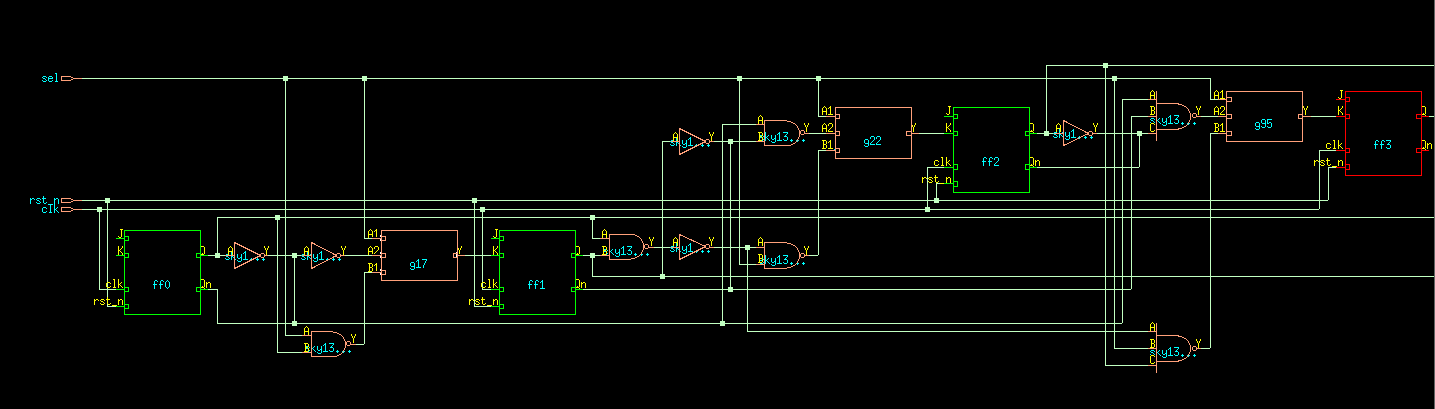
|  |  |  |  |
| --- | --- | --- | --- |
| Rst\_n | 1 | input | Negative edge reset. If rst\_n = 0, output will be set  to 0. Else, it will start the normal operation. |
| Sel | 1 | input | Mode selection signal. If sel = 1, the design will start  counting up. Else, it will start counting down from the current output value. |
| Out | 4 | output | Result of the counter. |

* Sử dụng Flip Flop JK và các cổng Not, And, Or để thực hiện đếm lên hoặc đếm xuống trong bộ counter\_4bit.

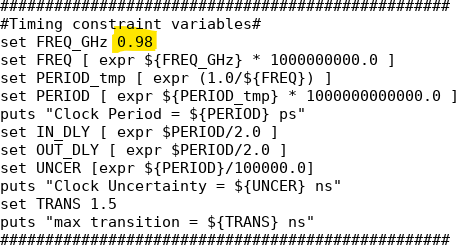


## Report

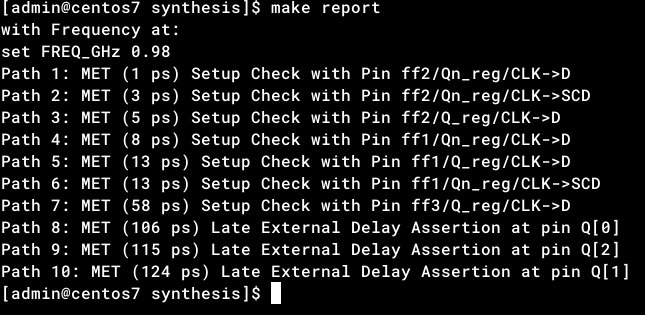
* + Schematic Viewer (main)



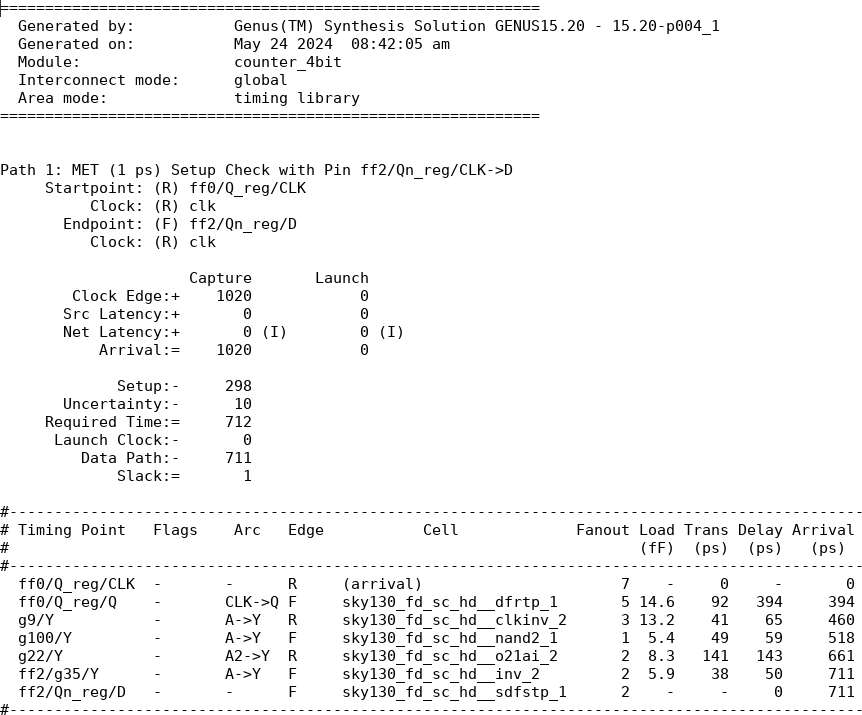
* + - Tần số cao nhất mà thiết kế có thể đạt được là 0.98 GHz



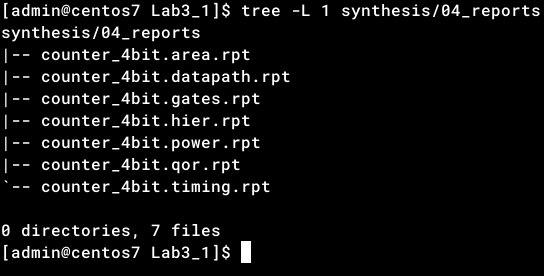
* + - Báo cáo tiến độ thiết kế:



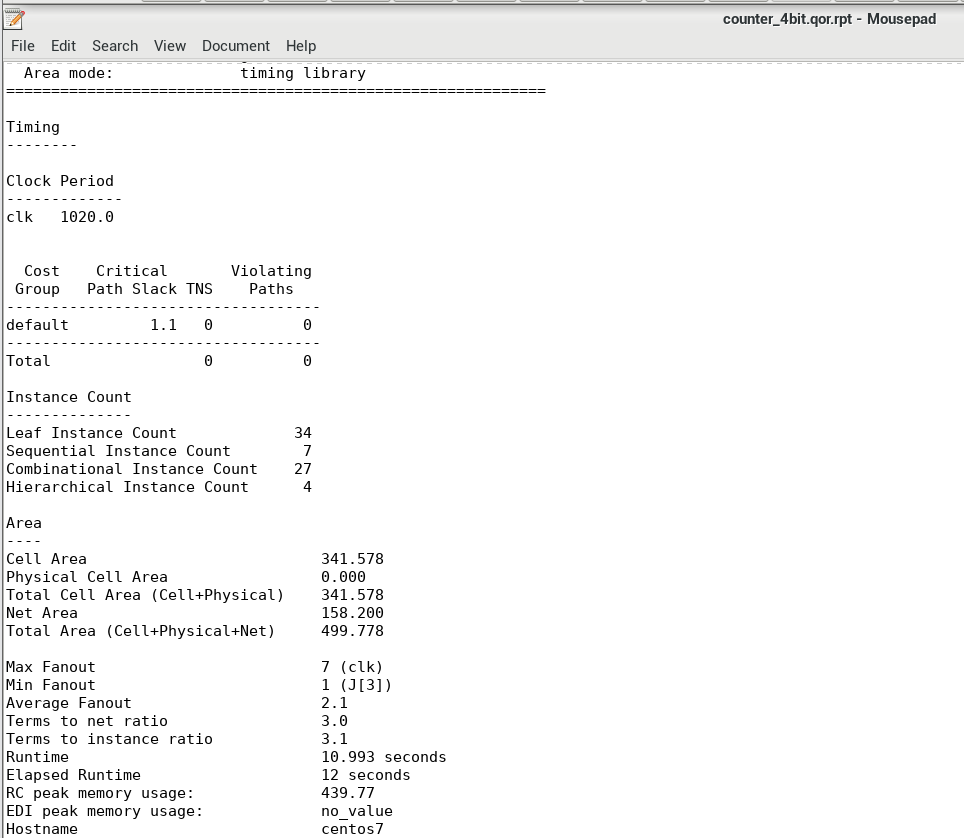
* + - Báo cáo cũng có thể được tìm thấy trong 04\_report/counter\_4bit.timing.rpt



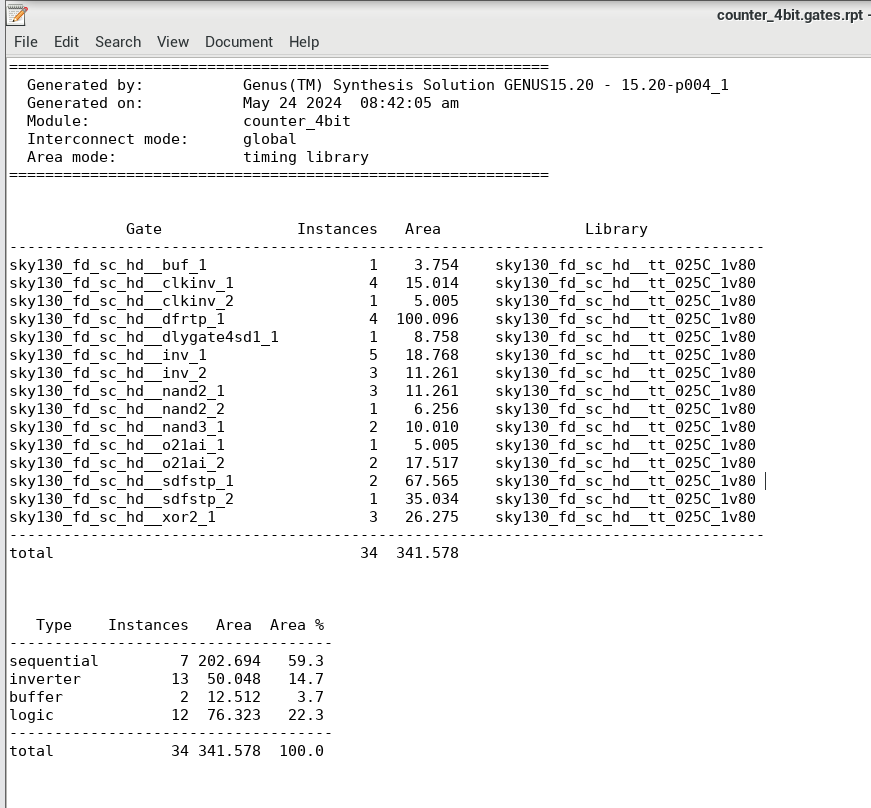
* + We can also see other aspects of the synthesized design in other reports inside the 04\_reports folder.



* + Tổng diện tích thiết kế và tài nguyên cổng được sử dụng trong thiết kế:



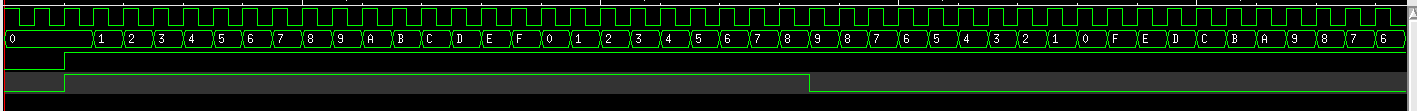
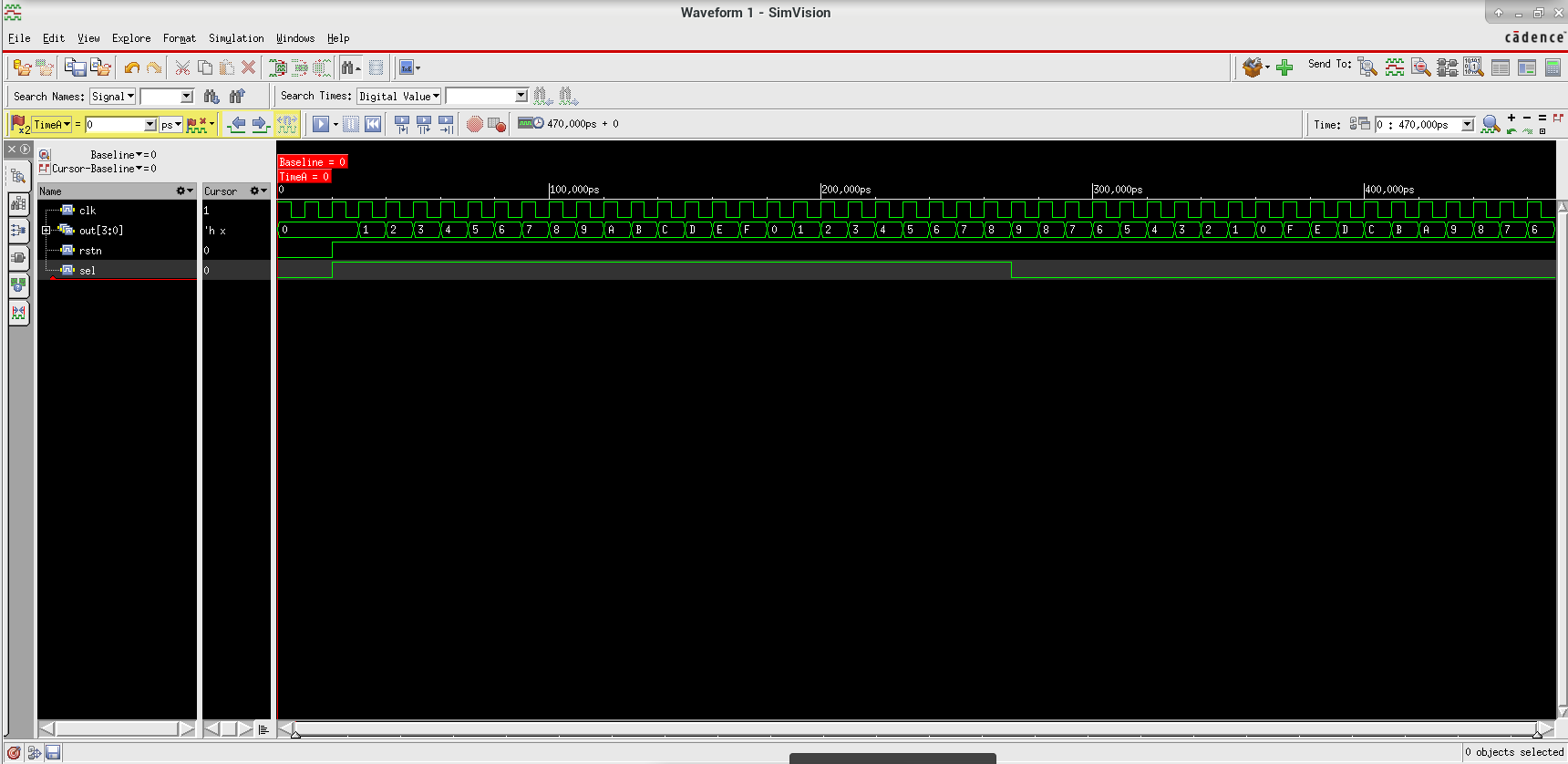
## Quality of Results report (QoR)



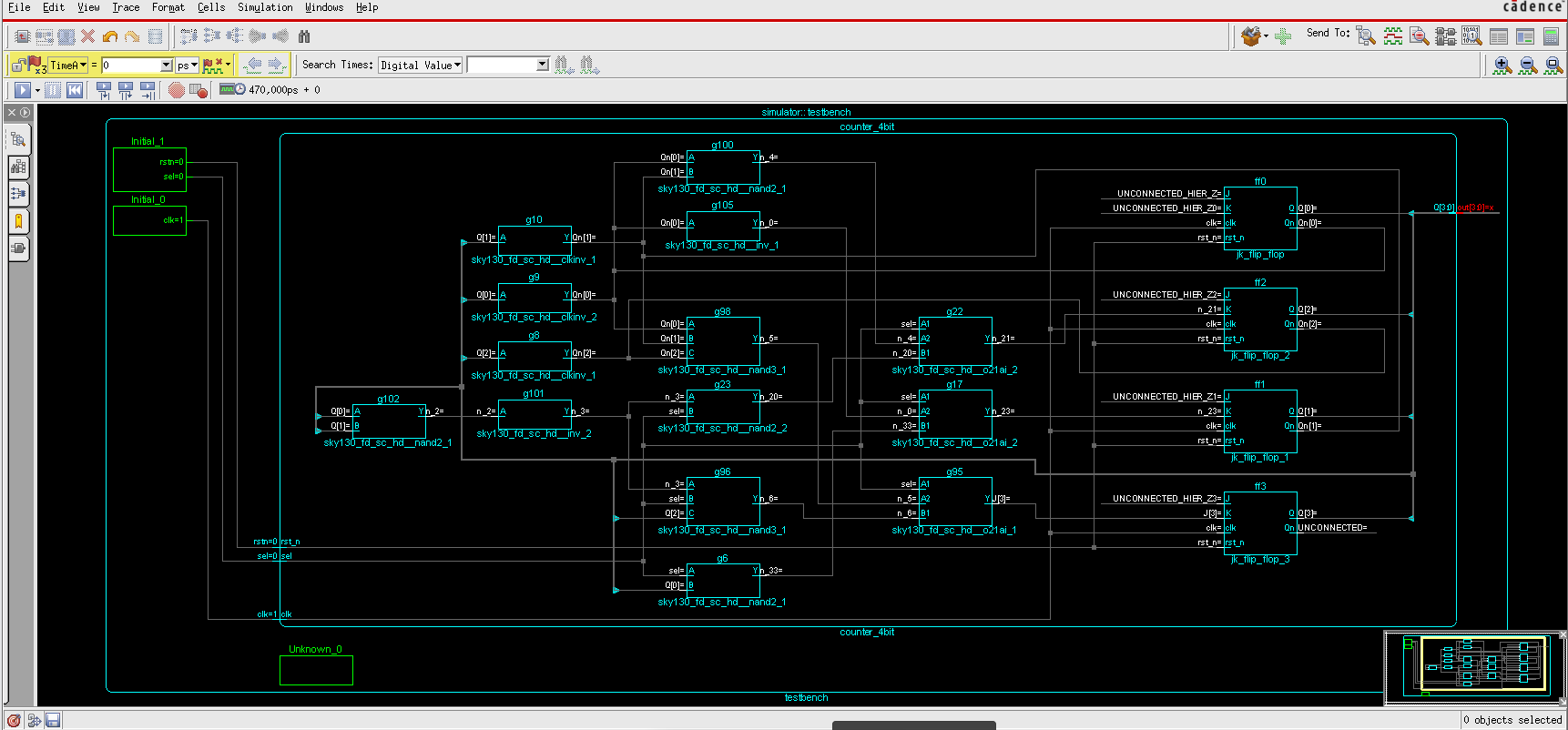
**Gate usage of the design**

## Waveform of RTL and netlist simulation

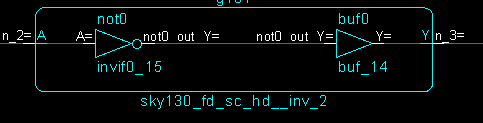
* + Dạng sóng của kết quả mô phỏng RTL:

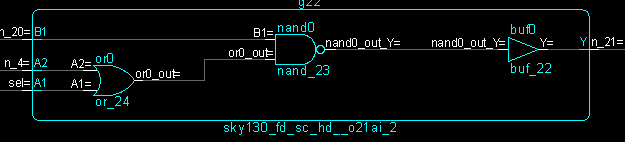
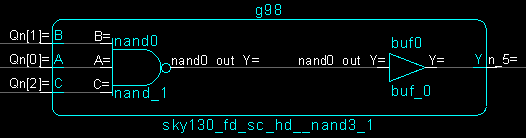


* + Schematic của netlist:



* + Các cổng logic được dùng trong thiết kế:



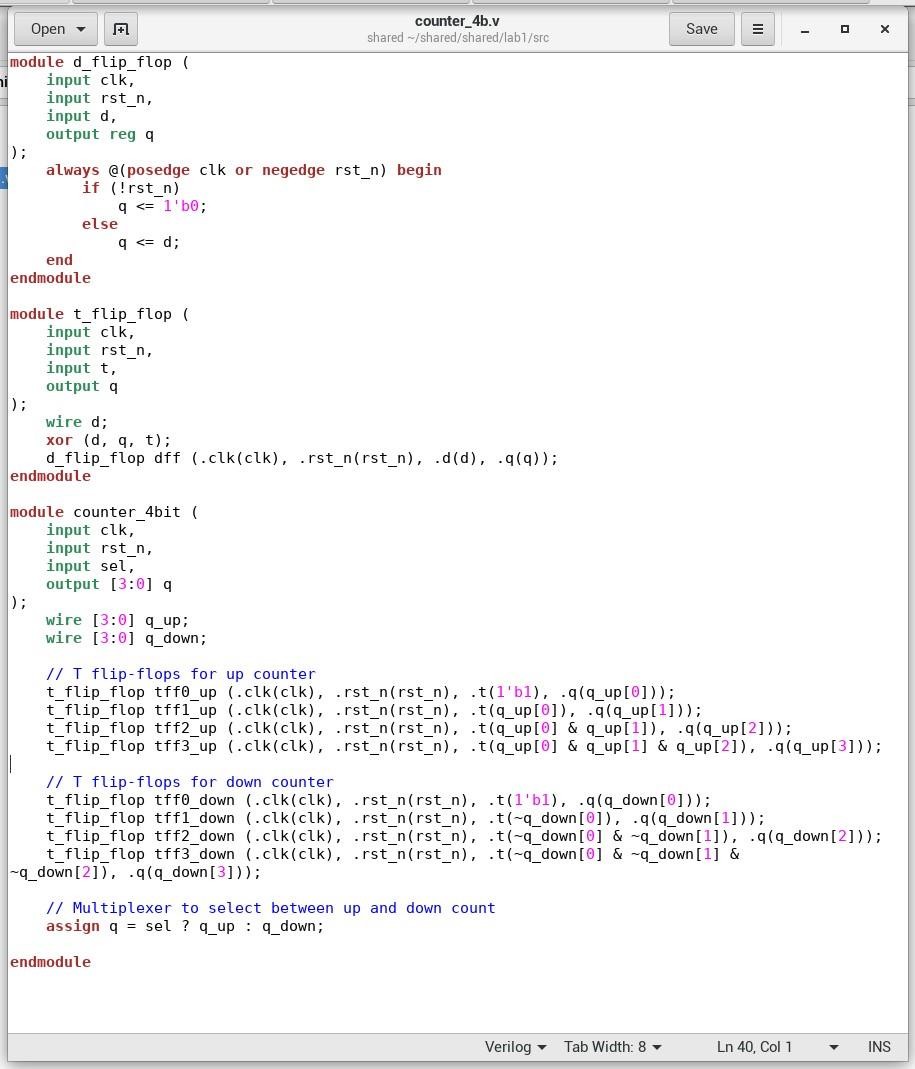


## Nhận xét:

* Kết quả mô phỏng dạng sóng (Waveform) chính xác so với yêu cầu đề bài.
* Phần schematic của netlist có một vài chỗ bị “Unconnected” dù rằng kết quả mô phỏng dạng sóng (Waveform) vẫn chính xác.

# VLSI FRONT-END FLOW

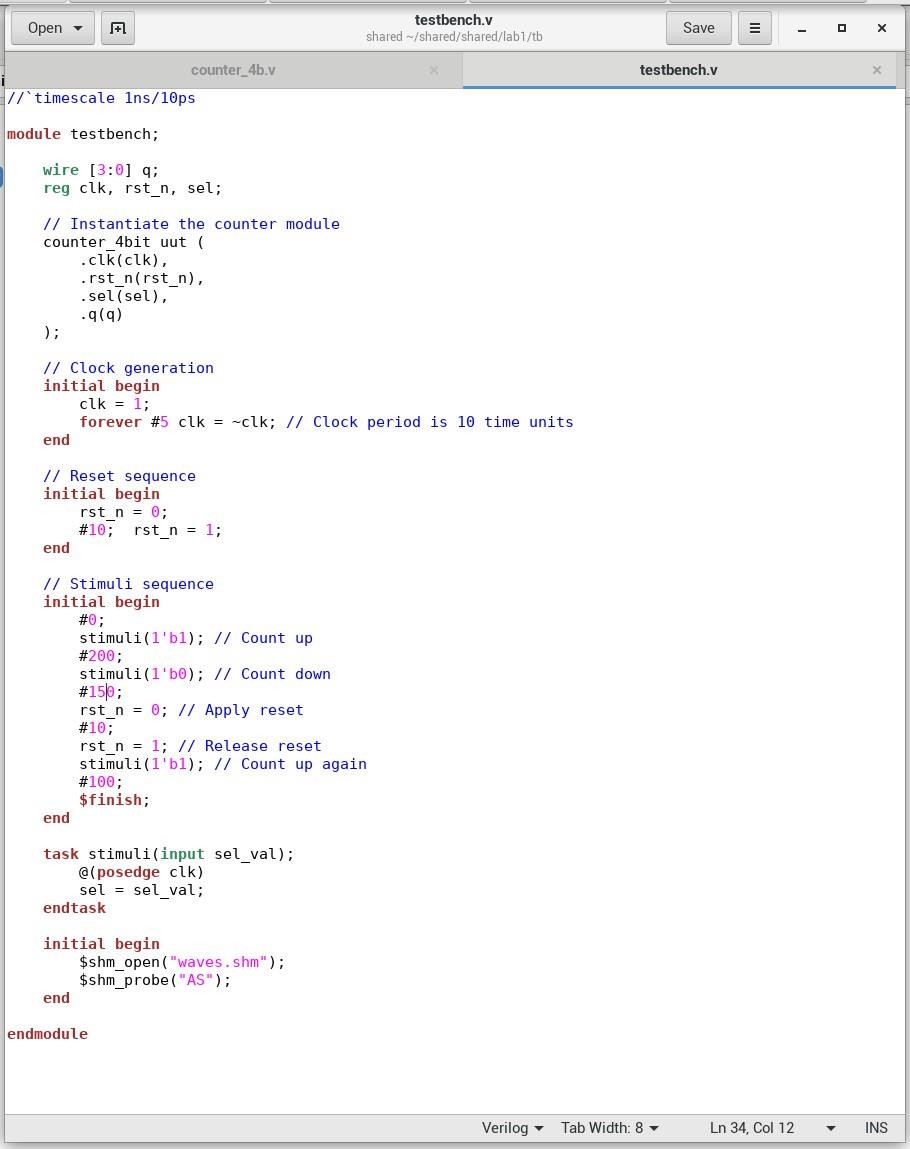
## Lab 1:Thiết kế 4-bit counter từ các T-Flip Flop.



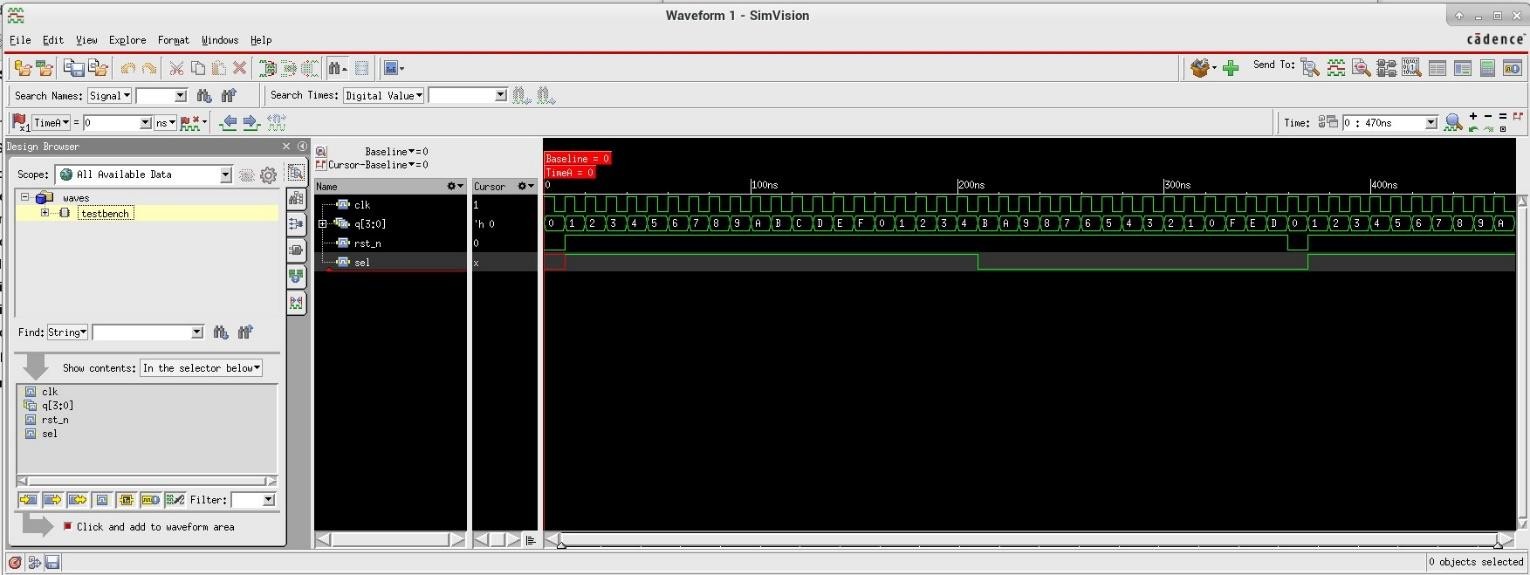
## Verification plan:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Section | Item | Description | Testcase name | Owner | Status |
| 1 | Reset | When rst\_n is asserted, the output is 0, when it de-asserts, the output start to count up each positive edge of clock | cnt\_rst\_test | Kiet | Pass |
| 2 | Max count | When the output is 4'b1111 and the counter is counting up , the next positive edge clock , output will be 4'b0000 | cnt\_max\_test | Kiet | Pass |
| 3 | Min count | When the output is 4'b0000 and the counter is counting down, the next positive edge clock , output will be 4'b1111 | cnt\_min\_test | Kiet | Pass |
| 4 | Count up | When sel is 1, the counter should count up on each positive edge of clock | cnt\_up\_test | Kiet | Pass |
| 5 | Count down | When sel is 0, the counter should count down on each positive edge of clock | cnt\_down\_test | Kiet | Pass |

## Code tạo file testbench.v



* 1. **Kết quả mô phỏng:**



Kết quả mô phỏng đúng như yều cầu của testbench và pass các điều kiện của verification plan

## Lab 2: ALU

## Code VHDL

module synth\_wrapper #(parameter WIDTH=4)( input clk,

input rst\_n,

input [WIDTH-1:0] a, input [WIDTH-1:0] b, input [2:0] op,

output reg [WIDTH-1:0] result, output reg carry

);

always @(posedge clk or negedge rst\_n) begin if (!rst\_n) begin

result <= 4'b0; carry <= 1'b0; end else begin case (op)

3'b000: begin // Addition

{carry, result} <= a + b; end

3'b001: begin // Subtraction

{carry, result} <= a - b; end

3'b010: begin // AND carry <= 1'b0;

result <= a & b; end

3'b011: begin // OR carry <= 1'b0; result <= a | b;

end

3'b100: begin // XOR carry <= 1'b0; result <= a ^ b;

end

3'b101: begin // NOT carry <= 1'b0;

result <= ~a; end

3'b111: begin // Logical shift left carry <= 1'b0;

result <= a << b; // Shift A left by B bits end

3'b110: begin // Logical shift right carry <= 1'b0;

result <= a >> b; // Shift A right by B bits end

default: begin result <= 4'b0; carry <= 1'b0; end

endcase end

end

endmodule

## Code testbench

Code testbench ( test add, sub, and, or, xor, not)

module testbench;

parameter WIDTH=4; reg clk;

reg rst\_n;

reg [WIDTH-1:0] a;

reg [WIDTH-1:0] b;

reg [2:0] op;

wire [WIDTH-1:0] result; wire carry;

synth\_wrapper u\_alu (

.clk (clk),

.rst\_n (rst\_n),

* 1. (a),
  2. (b),

.op (op),

.result (result),

.carry (carry)

);

initial begin

#0;

clk = 0;

forever #5 clk = ~clk;

end

initial begin

test();

end

initial begin

$shm\_open("tb.shm");

$shm\_probe("AS");

end

task test(); #0;

rst\_n = 0;

#50;

repeat (5) @(posedge clk); rst\_n = 1;

//add

op = 3'b000;

a = 5;

//sub

b = 3;

#30

a = 11;

b = 9;

#30

op = 3'b001;

a = 10;

b = 7;

#30

a = 7;

b = 10;

#30

//and

op = 3'b010;

a = 3;

b = 5;

#30

//or

op = 3'b011;

a = 5;

b = 3;

#30

//xor

op = 3'b100;

a = 3;

b = 5;

#30

//not

op = 3'b101;

a = 5;

#30

//rst rst\_n = 0;

#80

$finish; endtask

endmodule

## Code testbench ( test shift right, shift left)

module testbench;

parameter WIDTH=4; reg clk;

reg rst\_n;

reg [WIDTH-1:0] a;

reg [WIDTH-1:0] b;

reg [2:0] op;

wire [WIDTH-1:0] result; wire carry;

synth\_wrapper u\_alu (

.clk (clk),

.rst\_n (rst\_n),

1. (a),
2. (b),

.op (op),

.result (result),

.carry (carry)

);

initial begin

#0;

clk = 0;

forever #5 clk = ~clk;

end

initial begin

test();

end

initial begin

$shm\_open("tb.shm");

$shm\_probe("AS");

end

task test(); #0;

rst\_n = 0;

#50;

repeat (5) @(posedge clk); rst\_n = 1;

//shift right

op = 6;

a = 11;

b = 1;

#30

op = 6;

a = 11;

b = 2;

#30

op = 6;

a = 11;

b = 3;

#30

op = 6;

a = 11;

b = 4;

#30

//shift left

op = 7;

a = 11;

b = 1;

#30

op = 7;

a = 11;

b = 2;

#30

op = 7;

a = 11;

b = 3;

#30

op = 7;

a = 11;

b = 4;

#30

//rst

rst\_n = 0;

#80

$finish; endtask

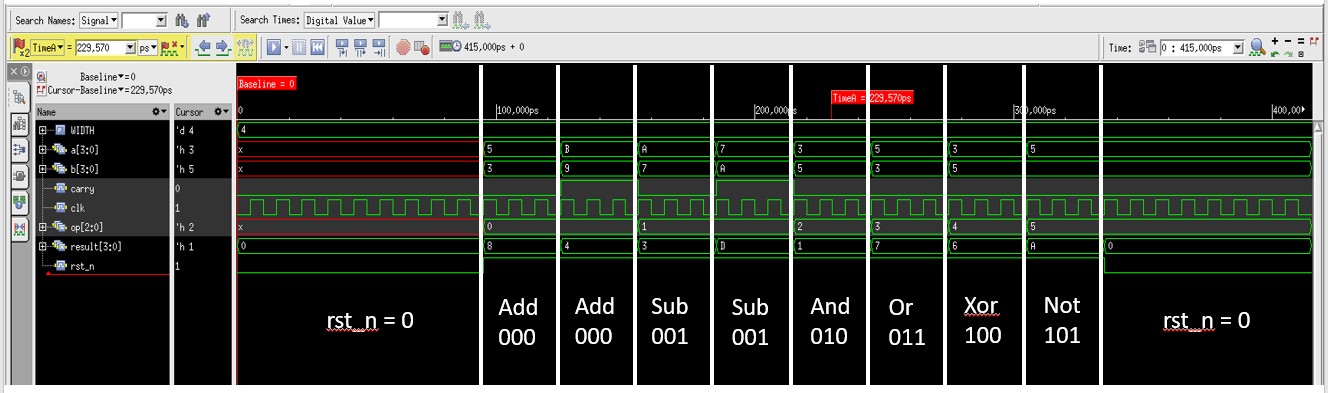
endmodule

## Verification plan

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Section | Item | Decription | Testcase name | Owner | Status |
| 1 | Reset | Khi rst\_n = 0, thì giá trị ngõ ra và cờ carry = 0, khi rst\_1 = 1 thì mạch hoạt động bình thường. | alu\_rst\_test | Tien Dat | PASS |
| 2 | Max add | Khi thực hiện việc cộng 2 số 4 bit, thì giá trị ngõ ra có thể bị tràn bit. Lúc bị  tràn thì cờ carry sẽ lên 1. | alu\_flag\_add | Tien Dat | PASS |
| 3 | Min sub | Khi thực hiện việc trừ 2 số 4 bit, thì giá trị ngõ ra có thể bị âm. Lúc giá trị âm thì cờ carry sẽ lên 1 | alu\_flag\_sub | Tien Dat | PASS |
| 4 | And, or, xor, not | Thực hiện and, or, xor, not hai số 4 bit bất kì | alu\_operator\_1 | Tien Dat | PASS |
| 5 | Shift | Thực hiện shift right hoặc left hai số 4 bit bất kì | alu\_operator\_2 | Tien Dat | PASS |

## Simulation

Simulation (test add, sub, and, or, xor, not)



Simulation (test shift right, shift left)

