Chương 3

# Bài thực hành-thí nghiệm 1: Thiết kế bộ cộng 2 số 4 bit

Bài thực hành-thí nghiệm mở đầu sẽ hướng dẫn người đọc cách xây dựng cấu trúc thư mục cho một dự án thiết kế vi mạch, cũng như bước đầu làm quen với toàn bộ quy trình thiết kế vi mạch bao gồm việc vẽ mô tả thiết kế, viết lập trình phần cứng cùng mô phỏng chức năng RTL thông qua phần mềm VCS, tổng hợp (Synthesis) bằng phần mềm DC, kiểm định netlist bằng phần mềm Formality, và Place & Route bằng phần mềm IC Compiler

## 3.1. Lý thuyết

Về cơ bản, khi thiết kế vi mạch, ta cần hình thành tư duy xây dựng theo dạng các khối cùng các kết nối của chúng để tạo thành một thể cấu tạo hoàn chỉnh. Mỗi khối như vậy, tài liệu này sẽ gọi là 1 module để tương đồng với định nghĩa module khi lập trình trên Verilog.

### 3.1.1. Mục tiêu thiết kế

Mục tiêu thiết kế trong bài thực hành-thí nghiệm này là một khối cộng 4 bit có các yêu cầu như sau:

- **Ngõ vào**: hai số 4 bit
- **Ngõ ra:** tổng của hai số trên, kèm với 1 bit nhớ ( Carry)

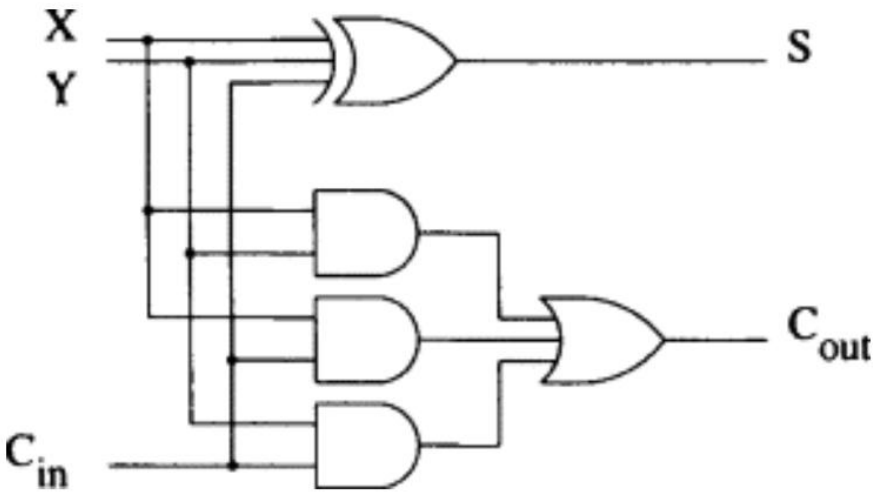
Ví dụ: Cho ngõ vào là hai số 4'b0110 (='d6) và 4'b1100 (='d12). Kết quả phép cộng sẽ là 5'b10010 (='d18), với 4 bit ngõ ra là 4'b0010 và bit nhớ Carry là 1'b1.

Bảng 3.1: Bảng sự thật cho khối cộng 1 bit.

| X | Y | $C_{in}$ | S | $C_{out}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

### 3.1.2. Bộ cộng 1 bit

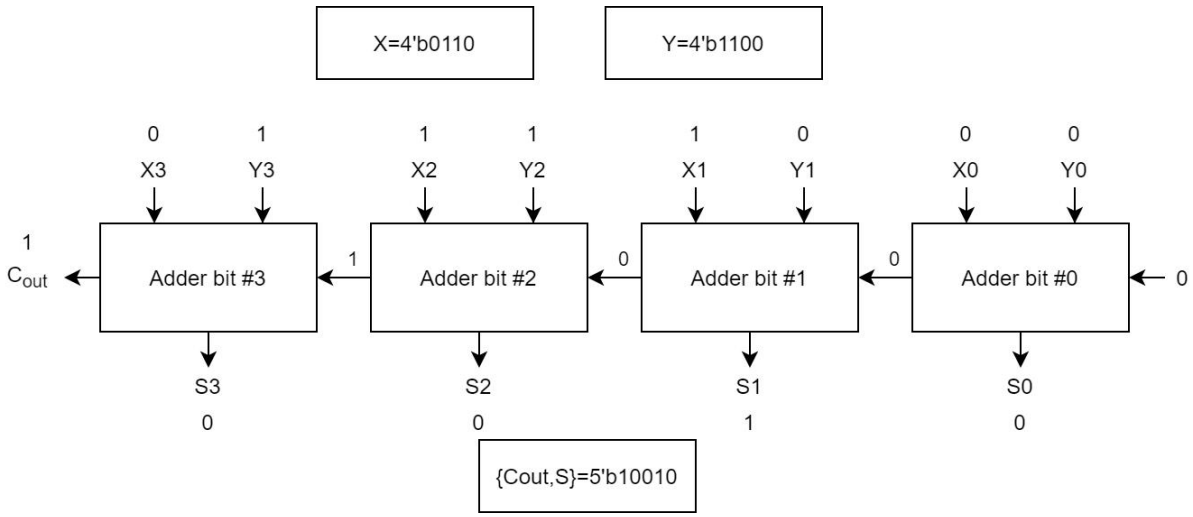
Để xây dựng bộ cộng 4 bit, trước hết ta xây dựng các bộ cộng 1 bit trước, ở mức cấp cổng. Các ngõ vào lần lượt là hai số X, Y cần tìm tổng, một số Cin (số nhớ carry) dùng cho phép cộng nhiều hơn 1 bit; các ngõ ra là số S thể hiện tổng, cùng bit Cout (thể hiện số nhớ cho phép cộng tiếp theo). Sơ đồ khối cho khối cộng 1 bit được thể hiện trong hình 3.1, với đơn giản là các cổng XOR, AND và OR. Để xây dựng được mạch logic như trên, người ta thực hiện rút gọn bìa Karnaugh thông qua bảng sự thật (bảng 3.1), tuy nhiên tài liệu này sẽ lược bỏ phần đó không hướng dẫn do đây là kiến thức căn bản.




***Hình 3.1.*** *Sơ đồ khối cho bộ cộng 1 bit.*

### 3.1.3 Kết nối thành bộ cộng 4 bit

Thực hiện cộng 2 số 4 bit cũng tương tự như khi ta thực hiện cộng hai số thập phân ở bậc trung học: cộng dồn từ các bit thấp nhất (LSB), nếu có nhớ thì cộng dồn vào phép cộng ở bit cao hơn. Do đó, để thiết kế bô cộng 4 bit, ta đơn giản chỉ cần nối tiếp 4 bộ cộng 1 bit đã thiết kế ở trên thành 1 chuỗi (hình 3.2), trong đó, bit nhớ $C_{in}$ của bộ cộng thấp nhất được nối bằng 0 do không còn bit nào thấp hơn, và bit $C_{out}$của bộ cộng cao nhất cũng chính là bit nhớ của toàn bộ phép cộng.




***Hình 3.2.*** *Sơ đồ khối cho bộ cộng 4 bit*

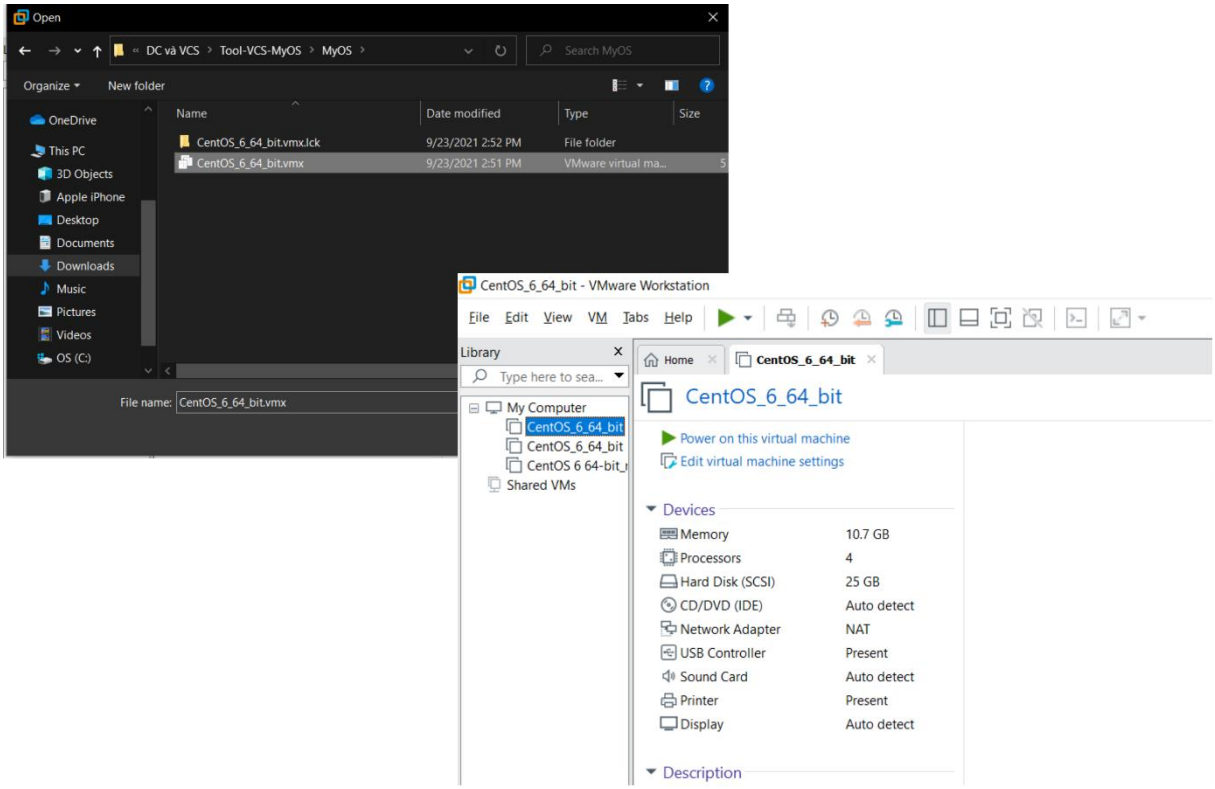
## 3.2. Thực hiện thiết kế

Ở từng công đoạn, tài liệu sẽ cung cấp một bản mô tả ngắn gọn những yêu cầu về tài liệu đầu vào, kết quả đầu ra, cũng như các bước nhỏ hơn để hoàn thành được công đoạn đó.

### 3.2.1. Tạo dựng môi trường và các chuẩn bị khác

Trong tài liệu này, gần như toàn bộ quy trình thiết kế vi mạch được thực hiện trên các máy ảo (thông qua phần mềm VMWare Station). Mục này sẽ hướng dẫn người đọc cách chuẩn bị các máy ảo trên cùng một số yêu cầu liên quan để có môi trường làm việc hiệu quả.

Như đã trình bày, quá trình thiết kế vi mạch thường được thực hiện trên nền tảng môi trường Linux, tuy nhiên hệ điều hành phổ biến nhất hiện tại cho người dùng là Windows. Do đó, tài liệu này cung cấp cho người đọc cách thức mô phỏng hệ điều hành Linux trên máy tính chạy hệ điều hành Windows thông qua hình thức máy ảo. Một máy ảo sẽ chứa đầy đủ tính năng cần thiết cũng như khả năng lưu trữ, tương tác với người dùng. Sau khi cài đặt **VMWare WorkStation** (phiên bản tùy chọn phù hợp với tình hình hiện tại) và khởi động, ta chọn nút **Open Virtual Machine** trên giao diện hoặc chọn **File/Open....** Sau đó vào đường dẫn lưu máy ảo có định dạng đuôi là **.vmx**. Một tab cho máy ảo vừa chọn sẽ được mở ra, người dùng chọn **Power on this virtual machine** để khởi động hệ điều hành (hình 3.3). Cần lưu ý rằng một máy ảo như trên cần được cài đặt các phần mềm cho thiết kế vi mạch bởi nhà cung cấp Synopsys để thực hiện các bước tiếp theo trong tài liệu này. Quá trình cài đặt cụ thể sẽ không được mô tả trong tài liệu này, bởi các ràng buộc về bản quyền cũng như tùy thuộc vào từng thời điểm mà người đọc tiếp cận.




***Hình 3.3.*** *Ví dụ cho khởi động máy ảo*

Sau khi máy ảo vào màn hình chờ, người dùng sẽ được yêu cầu nhập tên người dùng (user) và mật khẩu tương ứng, sau đó chọn **Log In**. Một giao diện Desktop thông thường của Linux sẽ hiện lên. Người đọc tiếp tục chọn mở chương trình **Terminal** trên màn hình Desktop, hoặc nhấn chuột phải vào một khoảng trống bất kì trên Desktop, chọn **Open in Terminal**, sau đó tải lên package tương ứng cho các tool synopsys (ví dụ thông qua gõ lệnh **LoadDC** để tải tool DC). Sau khi lấy được license giúp người đọc được cấp quyền chạy các tool, người đọc có thể sử dụng các tool trên mà không bị hạn chế về mặt truy cập.

Sau khi đã lấy được license cho tool, các bạn dùng lệnh **gedit ~/.bashrc** để mở file bashrc nhằm thêm biến môi trường. Ở giữa file, các bạn có thể thấy các dòng lệnh **$PATH**. Đây là các lệnh giúp ta thiết lập biến môi trường nhằm tạo điều kiện cho Linux biết lệnh ta dùng có thể được tìm kiếm ở đâu. Các bạn sẽ thêm vào một dòng lệnh sau:

**export PATH=$PATH:/home/albert/MyPrograms/synopsys/M2017.03-SP2/bin/**

Lệnh này có tác dụng trỏ thêm vào thư mục bin, nơi có một số lệnh ta sẽ cần dùng sau trong quá trình thiết kế. Sau khi sửa file bashrc, các bạn dùng lệnh **source ~/.bashrc** để thực thi, biến môi trường sẽ được cập nhật mới (hình 3.4).

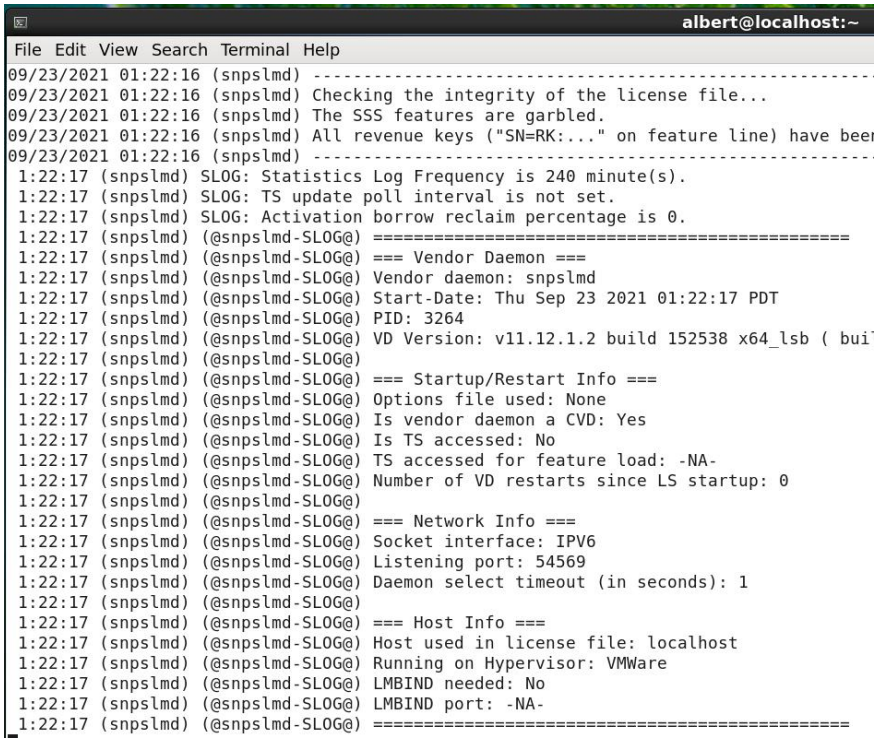
```
## User specific aliases and functions
export SNPSLMD_LICENSE_FILE=27020@localhost.localdomain
export LM_LICENSE_FILE=/home/albert/MyPrograms/synopsys/L-2016.03-SP1/Synopsys.dat
export MGLS_LICENSE_FILE=/home/albert/MyPrograms/synopsys/L-2016.03-SP1/Synopsys.dat
export SYNOPSYS=/home/albert/MyPrograms/synopsys/L-2016.03-SP1
export PATH=$PATH:/home/albert/MyPrograms/synopsys/L-2016.03-SP1/linux64/bin
export PATH=$PATH:/home/albert/MyPrograms/synopsys/L-2016.03-SP1/bin
export PATH=$PATH:/home/albert/MyPrograms/11.9/amd64/bin
export PATH=$PATH:/home/albert/MyPrograms/synopsys/M-2017.03-SP2/amd64/bin
export VCS_HOME=/home/albert/MyPrograms/synopsys/M-2017.03-SP2
export VCS_TARGET_ARCH=amd64
export VCS_ARCH_OVERRIDE=linux

export PATH=$PATH:/home/albert/MyPrograms/synopsys/M-2016.12/bin
export PATH=$PATH:/home/albert/MyPrograms/synopsys/M-2016.12/admin/install/lc/bin
export LC_HOME=/home/albert/MyPrograms/synopsys/M-2016.12
export PATH=$PATH:/home/albert/MyPrograms/synopsys/M-2017.03-SP2/bin/
# verdi
export PATH=$PATH:/home/albert/MyPrograms/synopsys/Verdi3_L-2016.06-1/bin


alias LoadDC='lmgrd -c $LM_LICENSE_FILE'
```

***Hình 3.4.*** *Nội dung file* ***bashrc.***

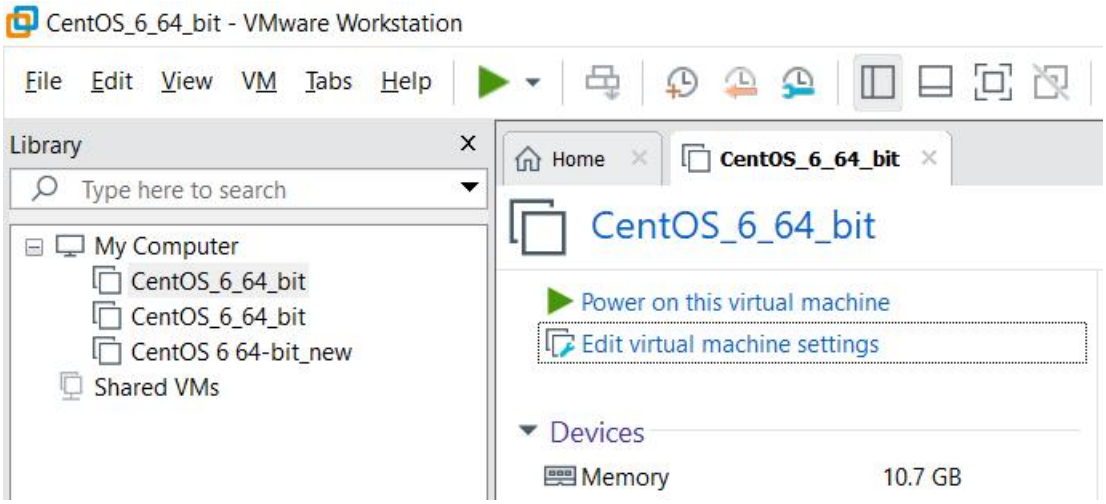
Trong một số trường hợp nhất định, các phần mềm được lưu và cài đặt trong những máy ảo khác nhau. Do đó, để đồng bộ dữ liệu giữa các bước, tài liệu này sẽ hướng dẫn thêm cách dùng thư mục chia sẻ (shared folder) giữa các máy ảo nói trên, đồng thời với cả với máy Windows đang sử dụng. Để làm được điều này, đầu tiên ta cần ở trạng thái trước khi bật (power on) máy ảo, hoặc nếu đang mở thì phải tắt (power off) nó đi, khi đó ta sẽ có mặt ở giao diện ban đầu của VMware Station. Trên cửa sổ làm việc của nó, chọn máy ảo cần chia sẻ quyền truy cập, chọn **Edit virtual machine settings** (hình 3.6). Trong cửa sổ hiện lên, chọn tab **Options**, sau đó chọn **Shared Folders**. Trong phần bên phải của cửa sổ, nhấn chọn **Always enabled**, nhấn phím **Add...** ở bên dưới (hình 3.7. Một cửa sổ sẽ xuất hiện, chọn **Next >**, sau đó nhập đường dẫn vào mục **Host path** (lưu ý là đường dẫn trên máy Windows), đường dẫn có thể chon bất kì thư mục nào, miễn là tiện cho việc sử dụng và nằm trên máy Windows. Sau đó, nhập tên thư mục (tên trong máy ảo) ở mục **Name** (hình 3.8, rồi nhấn phím **Next**, rồi chọn **Finish**. Nhấn **OK** để hoàn tất quá trình.

Sau khi bật máy ảo, để truy cập thư mục trên, trên Desktop chọn **Computer→Computer→Filesystem→mnt→hgfs**.Thư mục được chia sẻ sẽ nằm ở đây.

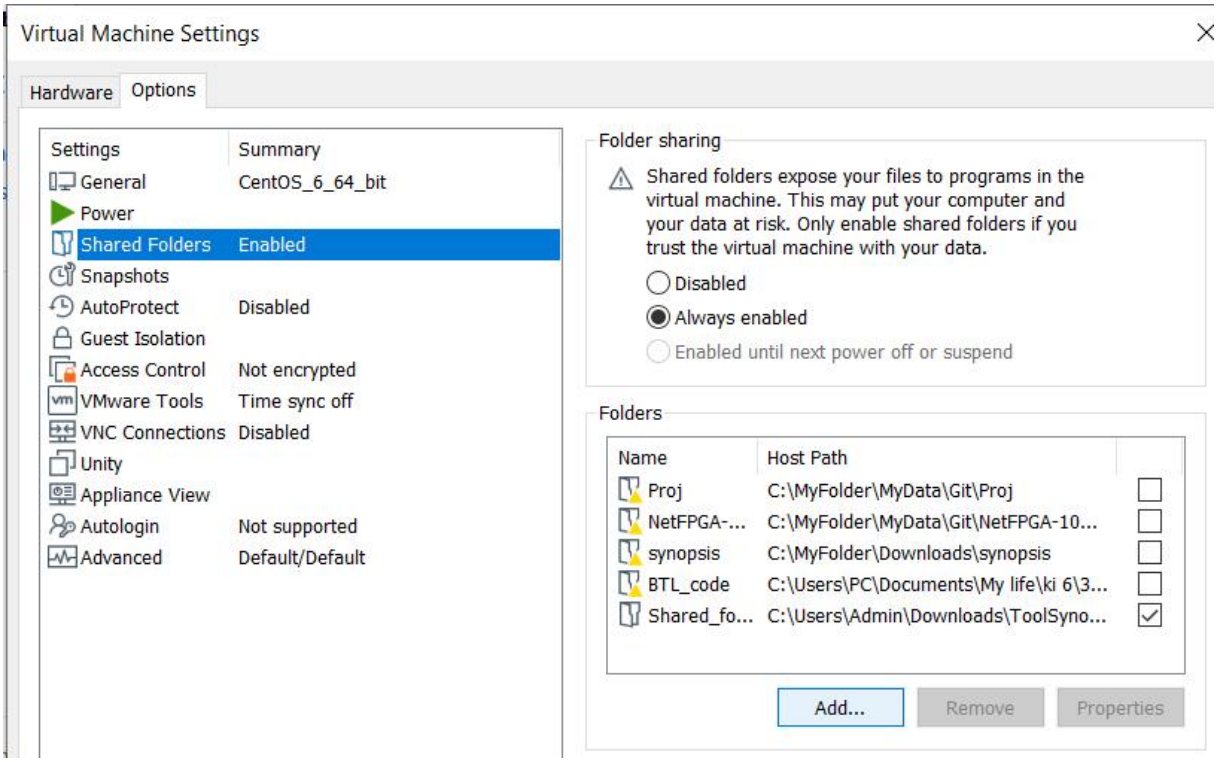
| Lưu ý: Với các thư mục được nén lại bên trong máy ảo, khi được chia sẻ qua lại giữa các máy ảo thì tốt nhất nên giải nén cũng bên trong máy ảo thay vì ngoài Windows để tránh mất mát nội dung các file bên trong. |
|---|



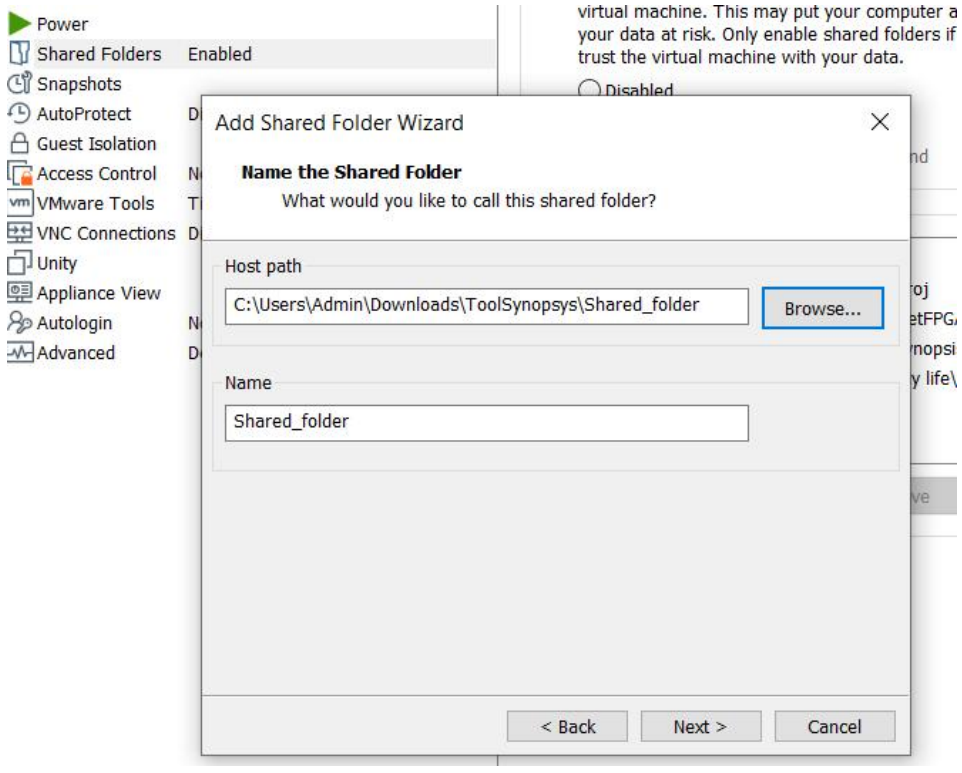

***Hình 3.5.*** *Ví dụ tải package tool DC và VCS thông qua việc lấy license*




***Hình 3.6.*** *Thêm thư mục chia sẻ (1).*




***Hình 3.7.*** *Thêm thư mục chia sẻ (2).*



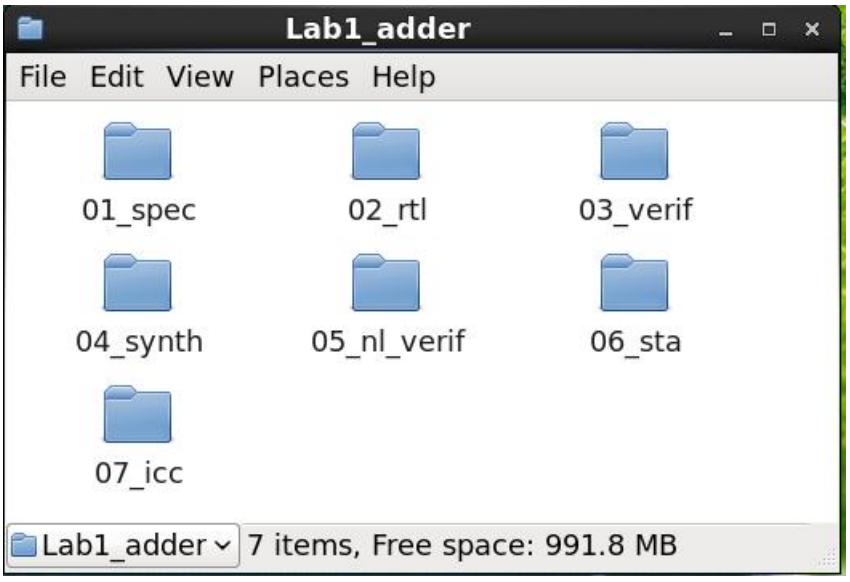

***Hình 3.8.*** *Thêm thư mục chia sẻ (3).*

Sau khi thực hiện các thao tác liên quan, chúng ta sẽ quay lại quy trình thiết kế vi mạch, bắt đầu từ việc xây dựng cây thư mục làm việc cho bài thực

hành-thí nghiệm 1, được chia sẻ giữa các máy ảo thông qua thư mục chia sẻ trình bày ở trên. Trên thực tế, môi trường thư mục làm việc dự án cũng cần được sắp xếp một cách khoa học, đặc biệt là khi mà quá trình thiết kế sẽ có nhiều thay đổi (chỉnh sửa lại thiết kế cho phù hợp). Người ta sẽ chia thư mục dự án thành nhiều phần thư mục riêng, ứng với từng công đoạn trong quá trình thiết kế, thuộc nhiệm vụ của các nhóm kĩ sư khác nhau.

Tài liệu này sẽ hướng dẫn người đọc xây dựng một thư mục làm việc theo từng phân đoạn, thể hiện trong hình 3.9. Trong đó:

- **01_spec:** Thư mục chứa các bản vẽ sơ đồ khối, hay các document thể hiện mô tả thiết kế.
- **02_rtl:** Thư mục chứa các mã lập trình phần cứng sử dụng Verilog.
- **03_verif:** Thư mục dành cho việc kiểm định RTL.
- **04_synth:** Thư mục dành cho việc tổng hợp RTL
- **05_nl_verif:** Thư mục dành cho kiểm định netlist sau tổng hợp RTL
- **06_sta:** Thự mục dành cho STA (phân tích thời gian tĩnh)
- **07_icc:** Thư mục dành cho việc Place&Route cũng như xuất file GDSII.




***Hình 3.9.*** *Cấu trúc thư mục cho bài thực hành-thí nghiệm 1.*

### 3.2.2 Thiết kế cấp độ hệ thống

Tài liệu này sẽ hướng dẫn người đọc xây dựng một thư mục làm việc theo từng phân đoạn, thể hiện trong hình 3.9. Trong đó:

***Bảng 3.2.*** *Mô tả bước thiết kế cấp độ hệ thống*

| Đầu vào | Đầu ra | Các công đoạn |
|---|---|---|
| Ý tưởng thiết kế | Mô tả cụ thể thiết kế (gọi chung là Specification), có thể là file Word, hình vẽ sơ đồ khối,… | - Nắm rõ mục đích của thiết kế<br>- Xây dự thiết kế nhằm phục vụ mục đích trên<br>- Sửa chữa lỗi và hoàn thiện |

Bảng 3.2 là mô tả khái quát của bước thiết kế cấp độ hệ thống

Lý thuyết trình bày ở trên chính là thiết kế ở cấp độ hệ thống. Trên thực tế, các thiết kế phức tạp cần nhiều thời gian và công sức hơn thế này nhiều. Để thiết kế được ở cấp độ hệ thống, thông thường người ta thể hiện dưới dạng sơ đồ khối (như trên), hoặc tập hợp nhiều sơ đồ khối (cho các khối con) cùng một sơ đồ khối tổng. Đôi khi, người ta có thể dùng từ ngữ hay bảng tính để mô tả cặn kẽ thiết kế, tuy nhiên dùng sơ đồ hình ảnh vẫn mang tính trực quan hơn nhiều. Cách tốt nhất để thực hiện thiết kế ở cấp độ hệ thống chính là vừa dùng hình ảnh vừa dùng từ ngữ để mô tả, trong đó hình ảnh giúp người đọc nắm bắt tổng quát, còn từ ngữ, bảng biểu giúp họ đi vào chi tiết khi cần.

### 3.2.3 Mô tả thiết kế bằng verilog

Bảng 3.3 là mô tả khái quát của bước thiết kế cấp độ RTL

| Đầu vào | Đầu ra | Các công đoạn |
|---|---|---|
| Specification | Các file mã lập trình đuôi .v, trong đó mô tả các khối trong thiết kế cùng kết nối giữa chúng thông qua ngôn ngữ Verilog | - Dùng Verilog mô tả lần lượt các khối trong thiết kế<br>- Kế nối các khối con và instance chúng trong các khối lớn hơn<br>- Kết nối các khối lớn hơn cho đến cấp độ cao nhất |

Dựa trên sơ đồ khối đã vẽ ở trên, ta sẽ xây dựng hai file Verilog lần lượt là **adder_1bit.v** và **adder_4bit.v**. Nội dung các file này được thể hiện trong hình 3.10, với lập trình cơ bản sử dụng toán tử logic và gọi instance các module con.

Sau khi lưu 2 file trên trong thư mục 02_rtl, các bạn thêm vào một file

lab_rtl.flist, có tác dụng như một bảng dẫn địa chỉ của tất cả các file RTL. File flist này thường chỉ dùng trong các dự án lớn, khi mà hàng trăm, hàng nghìn file Verilog cần được dẫn địa chỉ thông qua các file flist này. Tuy nhiên, trong bài thực hành-thí nghiệm này, tài liệu cũng sẽ hướng dẫn làm file flist cho người đọc quen dần với môi trường. Hình 3.11 thể hiện nội dung cũng như vị trí file flist trong thư mục 02_rtl.

adder_1bit.v

```
module adder_1bit(X,Y,Cin,S,Cout);
input X,Y,Cin;
output S,Cout;

assign S = X^Y^Cin;
assign Cout = (X&Y)|(Y&Cin)|(X&Cin);

endmodule
```

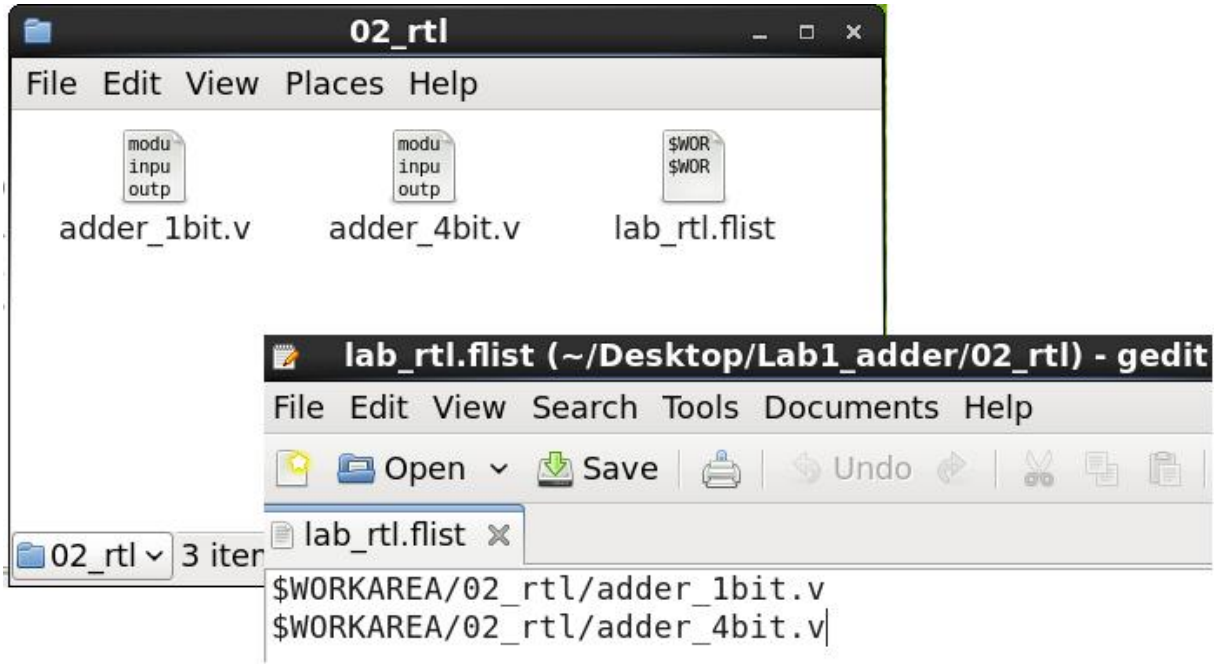
Dùng các toán tử logic AND, OR, XOR

adder_4bit.v

```
module adder_4bit(X,Y,S,Cout);
input [3:0] X, Y;
output [3:0] S;
output Cout;
wire C0, C1, C2;

adder_1bit A0(X[0],Y[0],1'b0,S[0],C0);
adder_1bit A1(X[1],Y[1],C0,  S[1],C1);
adder_1bit A2(X[2],Y[2],C1,  S[2],C2);
adder_1bit A3(X[3],Y[3],C2,  S[3],Cout);

endmodule
```

Instance 4 module adder_1bit và nối chúng lại thông qua các wire C0, C1, C2

***Hình 3.10.*** *Mã lập trình Verilog cho các bộ cộng*




***Hình 3.11.*** *File flist làm đường dẫn cho các file Verilog khác*

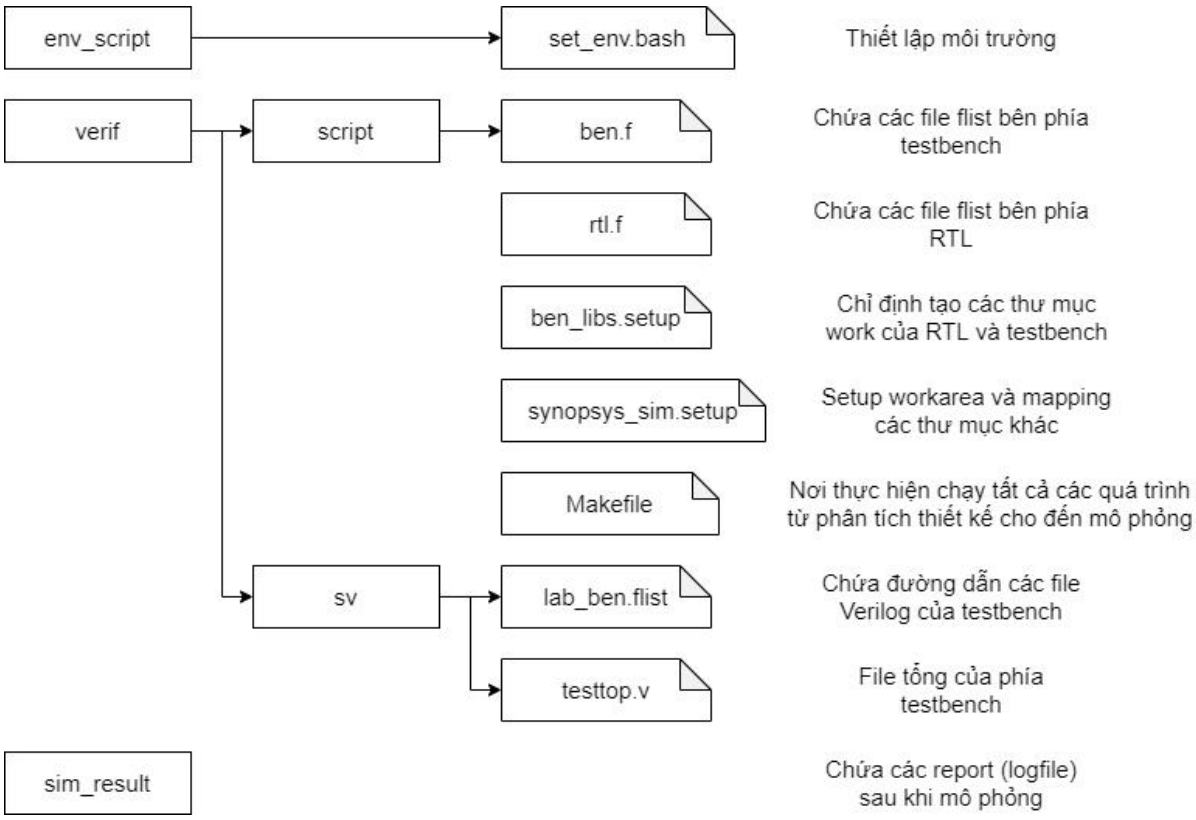
### 3.2.4 Thực hiện viết testbench

Bảng 3.4: Mô tả bước kiểm tra thiết kế cấp độ RTL (Verification).

| Đầu vào | Đầu ra | Các công đoạn |
|---|---|---|
| - Các file mã lập trình đuôi .v<br>- Các file testbench cũng đuôi .v hay .sv (SystemVerilog) | - Các báo cáo (report) về quá trình mô phỏng - Các file dạng sóng (waveform) nhằm kiểm tra mô phỏng thiết kế | - Xây dựng môi trường cho việc kiểm tra - Kiểm tra lỗi cú pháp của RTL (các file .v) và testbench<br>- Chạy mô phỏng và xuất ra dạng sóng<br>- Kiểm tra các file báo cáo (report) và dạng sóng nhằm đảm bảo mô phỏng RTL sạch lỗi |

Bảng 3.4 là mô tả khái quát của bước kiểm tra thiết kế cấp độ RTL (Verification).

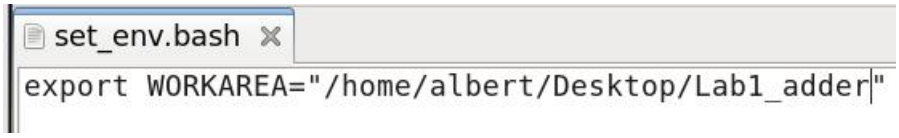
Việc tiếp theo cần làm là xây dựng môi trường kiểm định các file RTL đã thiết kế ở trên. Môi trường cần xây dựng giúp tối ưu khả năng sử dụng lại, tức là người thiết kế không cần chỉnh sửa quá nhiều để thực hiện kiểm định với một thiết kế RTL khác. Để làm được điều đó, thư mục 03_verif sẽ được xây dựng với cấu trúc như hình 3.12, bao gồm thư mục env_script để thiết đặt môi trường, thư mục verif để viết testbench, chạy phân tích và mô phỏng RTL, và thư mục sim_result để lưu các báo cáo (report) trong quá trình chạy dưới dạng logfile.




***Hình 3.12.*** *Cấu trúc chung cho thư mục 03_verif*

Trước hết, ta sẽ đi vào xây dựng file **set_env.bash** (hình 3.13) để hiểu cách thiết đặt môi trường của nó. Nội dung file này rất đơn giản là set một biến môi trường tên là **WORKAREA** theo đường dẫn của dự án hiện tại. Khi đó, các file, các thư mục khác có thể sử dụng biến này thay vì sử dụng đường dẫn tuyệt đối. Nói cách khác, khi chuyển đổi qua dự án khác, ta chỉ cần sửa lại đường dẫn của **WORKAREA** mà không làm ảnh hưởng nhiều đến nội dung các file khác. Ví dụ, trong file **lab_rtl.flist** như ở trên cũng có sử dụng biến **WORKAREA** này.

Tiếp theo, ta sẽ tiến hành xây dựng testbench cũng bằng ngôn ngữ Verilog ở thư mục verif/sv. Testbench này sẽ được đặt trong file **testtop.v**, có nội dung và mô tả được thể hiện trong hình 3.14.




***Hình 3.13.*** *Nội dung file set_env.bash*

testtop.v

```
module testtop;

reg [3:0] X,Y;
wire [3:0] S;
wire Cout;

adder_4bit adder_inst(X,Y,S,Cout);
//adder_4bit adder_inst( .X(X), .Y(Y), .S(S), .Cout(Cout));

initial begin
        X=4'b0000; Y=4'b0000; #10
        X=4'b0110; Y=4'b1100; #10
        X=4'b1100; Y=4'b0101; #10
        X=4'b1010; Y=4'b0010; #10
        #100;
        $finish;
end

always @(X or Y) begin
        #1;
        $display("%t Nhi phan: X = %b, Y = %b, Cout = %b, S = %b",$time,X,Y,Cout,S);
        $display("%t Thap phan: X = %d, Y = %d, {Cout,S} = %d",$time,X,Y,{Cout,S});
end

endmodule
```

Khai báo tín hiệu tương tự với RTL cần test, với input khai báo là reg, output khai báo là wire

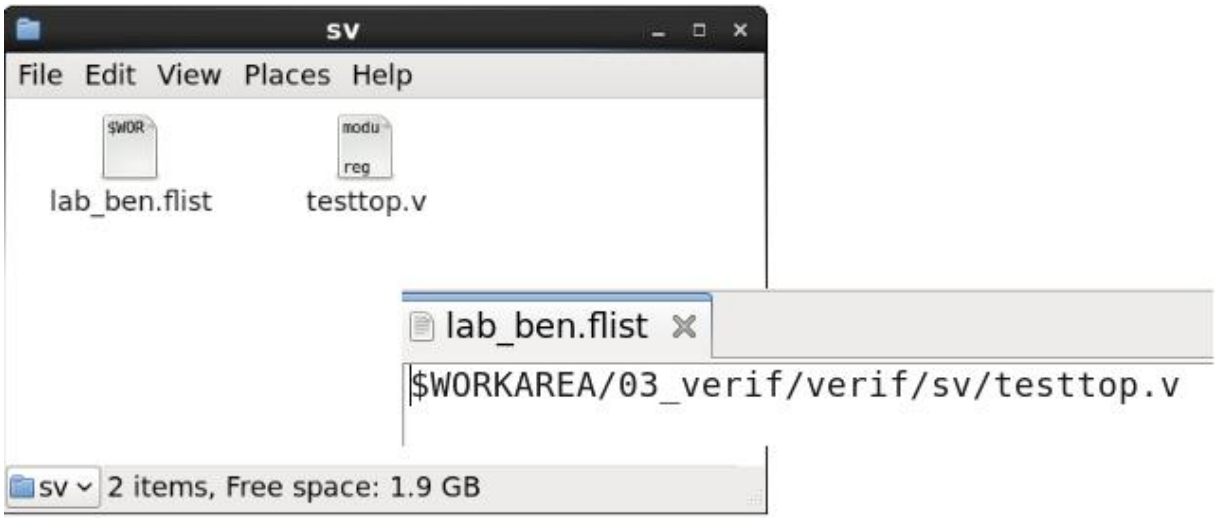
Instance module RTL tổng cần test, nối với các wire và reg của testtop cho phù hợp

Thay đổi input của RTL qua thời gian thông qua các reg ở trên

Xuất output khi input của RTL thay đổi ra Terminal để tiện theo dõi, có kèm theo mốc thời gian thông qua biến $time

***Hình 3.14.*** *Nội dung file testtop.v*

Sau khi viết xong file **testtop.v**, ta cũng thêm vào bên cạnh một file **lab_ben.flist** (tương tự như khi thiết kế RTL) với đường dẫn trỏ đến file **testtop.v** ở trên (hình 3.15).



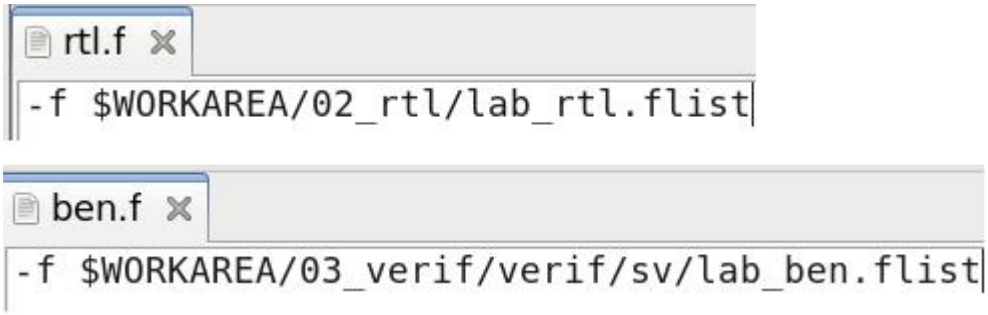

***Hình 3.15.*** *Nội dung file flist của testbench*

Tiếp theo, ta thực hiện xây dựng thư mục **verif/script**. Đây là thư mục chứa các file giúp tương tác với tool VCS của Synopsys cũng như góp phần tự

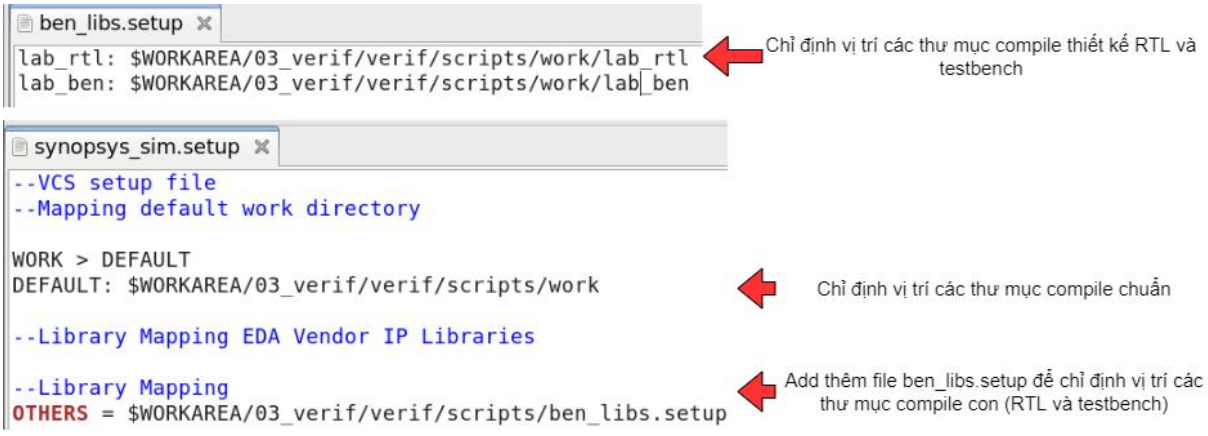
động hóa quá trình kiểm tra (test) để những lần sau làm những dự án khác không cần phải chỉnh sửa nhiều. Trong thự mục này, trước hết ta quan tâm đến các file **ben.f** và **rtl.f**; các file này đơn giản chỉ là chứa các file flist mà ta đã tạo trong các bước trước. Vậy ý nghĩa của chúng là gì? Trong khi các file flist giúp ta trỏ tới một hay nhiều file Verilog, tức tác dụng như một gói package, các file ben.f và rtl.f sẽ có nhiệm vụ chọn những package cần thiết trong quá trình kiểm tra, hay chọn những file flist phù hợp. Ở bài thực hành-thí nghiệm này, cả thiết kế và testbench đều rất đơn giản nên việc thực hiện nhiều file dẫn như thế này có thể là không cần thiết; tuy nhiên nếu trong các dự án lớn, khi mà có nhiều các file flist, ta phải sử dụng các file **ben.f** và **rtl.f** một cách phù hợp, tùy vào ta muốn kiểm tra những gì, kiểm tra toàn thể thiết kế hay kiểm tra cục bộ một khối nhỏ bên trong,... Trong bài thực hành-thí nghiệm này, tình huống kiểm tra đơn giản là kiểm tra chức năng toàn bộ thiết kế, với nội dung của chúng được thể hiện qua hình 3.16.

Tiếp đến, ta sẽ xem qua nội dung các file **ben_libs.setup** và **synopsys_sim.setup** (hình 3.17). Các file này giúp tool VCS của Synopsys xác định được vị trí để trả kết quả compile RTL và testbench; các kết quả compile này sẽ là thư viện để tool VCS chạy mô phỏng RTL.

Cuối cùng trong danh sách cần xét là file Makefile. Makefile chính là một trong những ngôn ngữ script đề cập trong phần trước. Ở đây, ta sẽ sử dụng nó để chạy các quá trình bao gồm phân tích (analyse) và mô phỏng (simulation). Nội dung của file này được thể hiện trong hình 3.18. Cụ thể:




***Hình 3.16.*** *Nội dung file ben.f và rtl.f*



***Hình 3.17.*** *Nội dung file ben_libs.setup và synopsys_sim.setup*

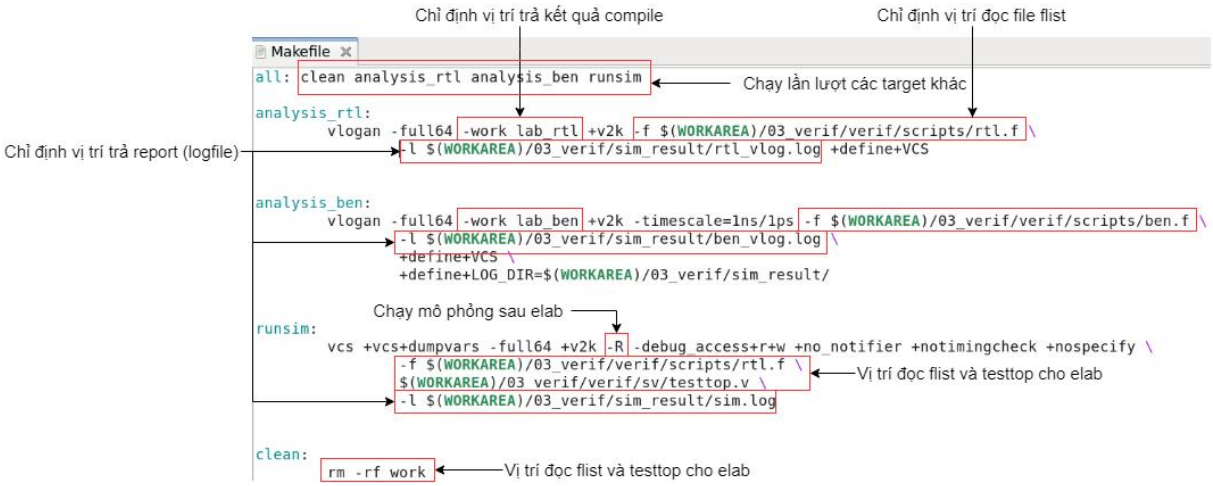
- Các chữ màu xanh lá cây trước dấu ":" được gọi là target. Mỗi target có thể chứa 1 hay nhiều lệnh bên trong. Khi chạy Makefile thông qua lệnh **make**, ta có thể chọn 1 hay nhiều target bên trong file, các lệnh bên trong các target sẽ được thực thi theo thứ tự. Nếu không đi kèm target khi chạy lệnh **make**, mặc định target được chọn là **all**.

- Lệnh **vlogan** là lệnh của VCS có tác dụng phân tích các file Verilog nhằm tìm lỗi về cú pháp (Syntax). Một số option sẽ được giải thích trong hình 3.18.

- Lệnh **vcs** là lệnh dùng để **elaborate**, tức là tìm ra sai sót nếu có về các mối quan hệ giữa các file Verilog mà không liên quan đến cú pháp. Khi thêm option "-R", lệnh này thực hiện chạy luôn mô phỏng thông qua file **testtop.sv** mà ta mô tả ở trên.

- Target **all** sẽ chạy lần lượt các target con bao gồm **clean** giúp xóa bỏ dữ liệu của lần chạy cũ, **analysis_rtl** và **analysis_ben** giúp phân tích các file Verilog ở cả RTL và testbench, và **runsim** giúp mô phỏng thiết kế RTL.

- Các kết quả sau khi phân tích và mô phỏng sẽ được lưu dưới dạng logfile trong thư mục sim_result theo đường dẫn được chỉ định qua các bước.




***Hình 3.18.*** *Nội dung file Makefile*

Đó là tất cả các file cần chuẩn bị cho quá trình kiểm định RTL. Để thực hiện chạy phân tích và mô phỏng RTL, bắt đầu từ thư mục gốc, ta thực hiện chạy các lệnh như hình 3.19. Trước hết là trỏ vào thư mục 03_verif, source lại file **set_env.bash** để thiết đặt môi trường, sau đó vào trong thư mục có chứa file Makefile, nhập lệnh **make** để thực hiện toàn bộ quá trình. Có thể thấy khâu

chuẩn bị càng kỹ lưỡng thì quá trình làm việc sẽ càng nhanh chóng hơn rất nhiều. Kết quả chạy mô phỏng được thể hiện trong hình 3.20, hiện lên trong Terminal; ngoài ra ta cũng có thể xem thêm các logfile trong thư mục **sim_result** (chính là quá trình chạy trên Terminal nhưng được lưu lại).

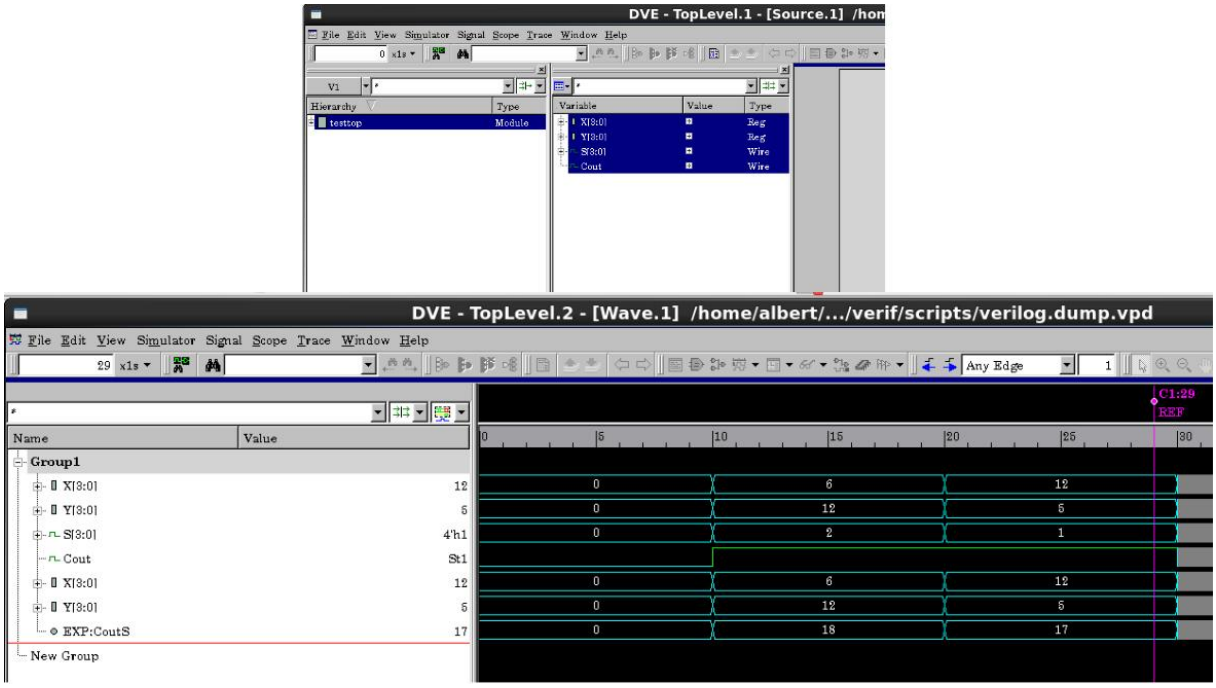
```
[albert@localhost Lab1_adder]$ pwd
/home/albert/Desktop/Lab1_adder
[albert@localhost Lab1_adder]$ cd 03_verif/
[albert@localhost 03_verif]$ source env_scripts/set_env.bash
[albert@localhost 03_verif]$ cd verif/scripts/
[albert@localhost scripts]$ make
```

***Hình 3.19.*** *Các lệnh để chạy thiết kế và mô phỏng RTL*

Sau khi mô phỏng và đọc các logfile, đôi khi vẫn chưa đủ để ta kiểm chứng tính đúng đắn của RTL, khi đó người ta có nhu cầu xem dạng sóng, tức giá trị tín hiệu qua thời gian dưới dạng hình ảnh. Để làm được điều đó, sau khi mô phỏng, cũng trên Terminal, người đọc nhập lệnh **dve&** (Dấu & ở đây có tác dụng làm cho chương trình **DVE** có thể chạy trong background và ta có thể tiếp tục sử dụng Terminal mà không cần chờ **DVE** phải tắt). Giao diện của **DVE** khá dễ sử dụng, người dùng chỉ cần lựa chọn **File/OpenDatbase**,chọn **verilog.dump** trong thư mục **verif/script** là mở được dạng sóng của RTL vừa được mô phỏng. Cửa sổ bên trái cho ta một mô hình cây phân cấp các khối, giúp ta lựa chọn khối cần xem, cửa sổ ngay bên cạnh cho phép ta lựa chon tín hiệu của khối tương ứng. Sau khi lựa chọn, người đọc có thể click chuột phải, chọn **Add to Wave** hoặc nhấn tổ hợp **Ctrl+4** để mở dạng sóng và quan sát (hình 3.21).

```
17.03-SP2/linux64/lib/vcs_save_restore_new.o -ldl  -lc -lm -lpthread -ldl
../simv up to date
make[1]: Leaving directory `/home/albert/Desktop/Lab1_adder/03_verif/verif/scripts/csrc'
Notice: timing checks disabled with +notimingcheck at compile-time
Chronologic VCS simulator copyright 1991-2017
Contains Synopsys proprietary information.
Compiler version M-2017.03-SP2_Full64; Runtime version M-2017.03-SP2_Full64;  Sep 23 04:06 2021
                  1 Nhi phan: X = 0000, Y = 0000, Cout = 0, S = 0000
                  1 Thap phan: X =  0, Y =  0, {Cout,S} =  0
                 11 Nhi phan: X = 0110, Y = 1100, Cout = 1, S = 0010
                 11 Thap phan: X =  6, Y = 12, {Cout,S} = 18
                 21 Nhi phan: X = 1100, Y = 0101, Cout = 1, S = 0001
                 21 Thap phan: X = 12, Y =  5, {Cout,S} = 17
                 31 Nhi phan: X = 1010, Y = 0010, Cout = 0, S = 1100
                 31 Thap phan: X = 10, Y =  2, {Cout,S} = 12
$finish called from file "/home/albert/Desktop/Lab1_adder/03_verif/verif/sv/testtop.v", line 16.
$finish at simulation time                  140
          V C S   S i m u l a t i o n   R e p o r t
Time: 140
CPU Time:      0.210 seconds;       Data structure size:   0.0Mb
Thu Sep 23 04:06:04 2021
CPU time: .093 seconds to compile + .006 seconds to elab + .131 seconds to link + .228 seconds in simulation
[albert@localhost scripts]$
```

***Hình 3.20.*** *Kết quả mô phỏng RTL*




***Hình 3.21.*** *Quan sát dạng sóng bằng phần mềm DVE.*

### 3.2.5 Tổng hợp (Synthesis)

Bảng 3.5: Mô tả bước Synthesis (tổng hợp) thiết kế.

| Đầu vào | Đầu ra | Các công đoạn |
|---|---|---|
| - Các file mã lập trình đuôi .v<br>- Các file thư viện đuôi .db nhằm cung cấp các cell tham khảo cho quá trình chuyển RTL sang các cell này<br>- Một số ràng buộc về timing, area hay power | - Các báo cáo (report) về quá trình tổng hợp<br>- Các file dùng cho các công đoạn tiếp theo trong quy trình thiết kế (.sdc, .sdf, .ddc,...)<br>- File RTL (.v) đã tổng hợp (đã chuyển về các cell trong thư viện tham khảo), thường gọi là netlist | - Viết script cho quá trình tổng hợp, bao gồm việc lựa chọn thư viện, lựa chọn RTL, thêm các ràng buộc, tổng hợp và xuất kết quả<br>- Chạy script tổng hợp<br>- Kiểm tra và sửa lỗi sau khi tổng hợp (nếu có) |

Bảng 3.5 là mô tả khái quát của bước Synthesis (tổng hợp) thiết kế.

Công đoạn tiếp theo là Synthesis (tổng hợp) RTL ta vừa mới thiết kế ở trên. Như đã trình bày, quá trình Synthesis sẽ chuyển RTL về dạng các cổng (các cell) theo một thư viện cho trước, đồng thời tối ưu các kết nối và logic của RTL kể trên. Ở đây ta sẽ sử dụng phần mềm DC của Synopsys. Và thư mục **04_synth** sẽ được tổ chức như sau (hình 3.22):

- **lib**: Đây là thư mục chứa các thư viện, bao gồm các cổng với đặc tính

đã được chuẩn hóa. RTL sẽ được chuyển về dạng bao gồm các cổng này.

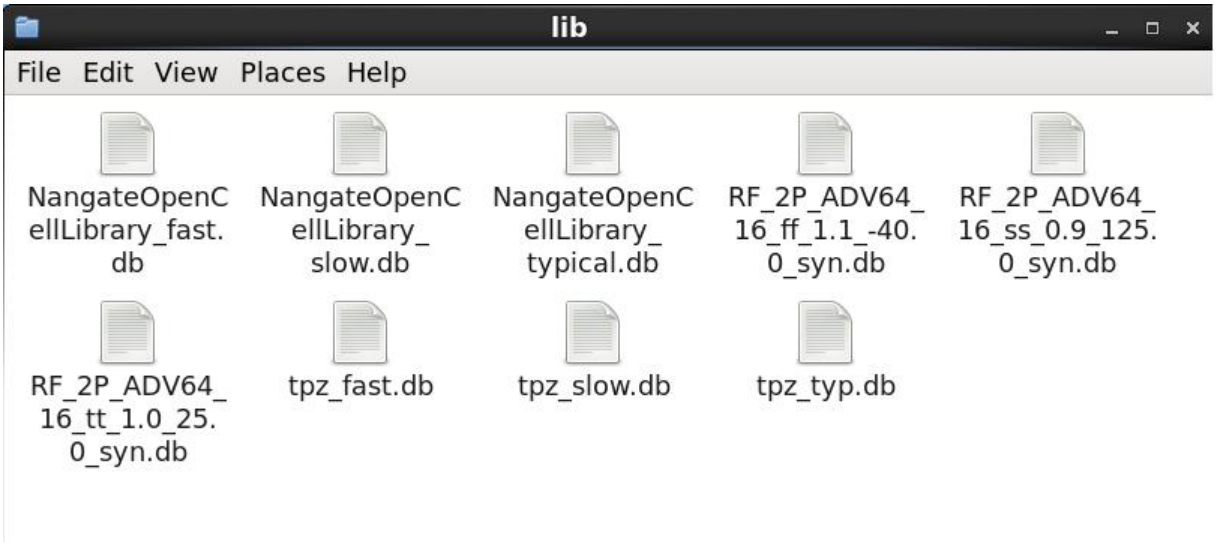
- **report**: Thư mục này chứ các thông tin báo cáo về diện tích, timing, chất lượng,...
- **run**: Thư mục chứa các file hệ thống trong quá trình chạy Synthesis cũng như kết quả RTL sau Synthesis.
- **dc_command.src**: Đây là file chứa các lệnh chạy quá trình Synthesis.

Trước hết, người đọc cần import thư viện vào trong thư mục **lib** (hình 3.23). Các thư viện này được cung cấp bởi các nhà máy, người dùng sau khi có được chúng thì lưu các thư viện này (đuôi .db) và thư mục **lib** nói trên.

Tiếp đến, ta tiến hành xây dựng file **dc_command.src** nhằm tự động hóa quá trình Synthesis. Mô tả file này được thể hiện trong hình 3.24. Chú ý thư viện người viết tài liệu sử dụng (NangateOpenCellLibrary_typical.db) có thể được thay thế bởi bất kì thư viện nào khác mà người đọc nhận thấy sử dụng phù hợp.

Để chạy tổng hợp, người đọc thực hiện **cd** vào thư mục **run**, sau đó nhập lệnh **dc_shell** để khởi động chương trình **DC**. Sau đó, nhập lệnh **source ../dc_command.src** và chờ quá trình Synthesis hoàn tất (hình 3.25).


***Hình 3.22.*** *Tổ chức thư mục 04_synth.*




***Hình 3.23.*** *Ví dụ các file thư viện để chạy Synthesis.*

```
#!/bin/bash

#============== SET DIRECTORY =================
set search_path "./../lib"
set osearch_path [ concat $search_path \
            ]
#============== ADD THE LIBRARY ================
set target_library "NangateOpenCellLibrary_typical.db"
set link_library "* $target_library"
set synthesis_library standard.sldb

#============= ANALYSE DESIGN ==================
analyze -format verilog "./../../02_rtl/adder_1bit.v"
analyze -format verilog "./../../02_rtl/adder_4bit.v"
elaborate adder_4bit
current_design adder_4bit

#============= CONSTRAIN FOR DESIGN ============

#============= SYNTHESIZE=======================
compile ultra

#============= REPORT PERFORMANCE ==============
report_area > ./../report/report.area
report_timing > ./../report/report.timing
report_constraint > ./../report/report.constraint
report_qor > ./../report/report.qor
write -f ddc -o ./../report/report.ddc
write -format verilog -hierarchy -output ./../report/lab_synth.netlist.v
write_sdf ./../report/report.sdf
write_sdc ./../report/report.sdc

quit
```

Chỉ định đường dẫn tìm kiếm thư viện

Chọn thư viện dùng cho Synthesis

Chọn các file RTL cần phân tích

Thường chọn file RTL tổng

Chọn file RTL cần Synthesis

Thiết kế phần này đơn giản nên chưa cần ràng buộc

Chạy Synthesis

Lưu các file report + RTL đã Synthesis (file lab_synth.netlist.v)

***Hình 3.24.*** *Nội dung file dc_command.src và giải thích.*

```
[albert@localhost run]$ pwd
/home/albert/Desktop/Lab1_adder/04_synth/run
[albert@localhost run]$ dc_shell

                        Design Compiler Graphical
                              DC Ultra (TM)
                               DFTMAX (TM)
                           Power Compiler (TM)
                              DesignWare (R)
                              DC Expert (TM)
                            Design Vision (TM)
                            HDL Compiler (TM)
                            VHDL Compiler (TM)
                               DFT Compiler
                            Design Compiler(R)

                    Version L-2016.03-SP1 for linux64 - Apr 18, 2016

                         Copyright (c) 1988 - 2016 Synopsys, Inc.
   This software and the associated documentation are proprietary to Synopsys,
 Inc. This software may only be used in accordance with the terms and conditions
 of a written license agreement with Synopsys, Inc. All other use, reproduction,
            or distribution of this software is strictly prohibited.
Initializing...
Initializing gui preferences from file  /home/albert/.synopsys_dv_prefs.tcl
dc_shell> source ../dc_command.src
```

***Hình 3.25.*** *Chạy Synthesis trên Terminal bằng phần mềm DC.*

Nếu quan sát file RTL sau tổng hợp (**report/lab_synth.netlist.v**), ta có thể thấy được các cổng của RTL ban đầu đã được thay thế bởi các cell trong thư viện, đồng thời số cổng logic cũng được giảm xuống (hình 3.26). Ngoài ra, ta có thể vào thư mục **report** để quan sát các báo cáo khác liên quan đến diện tích, timing, chất lượng của RTL và đảm bảo không có vi phạm nào xảy ra.

```
module adder_4bit ( X, Y, S, Cout );
  input [3:0] X;
  input [3:0] Y;
  output [3:0] S;
  output Cout;
  wire   \A0/N0 , n4, n5, n6;
  assign S[0] = \A0/N0 ;

  XOR2_X1 U7 ( .A(Y[0]), .B(X[0]), .Z(\A0/N0 ) );
  AND2_X1 U8 ( .A1(Y[0]), .A2(X[0]), .ZN(n6) );
  FA_X1 U9 ( .A(X[3]), .B(Y[3]), .CI(n4), .CO(Cout), .S(S[3]) );
  FA_X1 U10 ( .A(X[2]), .B(Y[2]), .CI(n5), .CO(n4), .S(S[2]) );
  FA_X1 U11 ( .A(Y[1]), .B(X[1]), .CI(n6), .CO(n5), .S(S[1]) );
endmodule
```

***Hình 3.26.*** *Kết quả RTL sau khi Synthesis.*

Nếu quan sát file RTL sau tổng hợp (report/lab_synth.netlist.v), ta có thể thấy được các cổng của RTL ban đầu đã được thay thế bởi các module trong thư viện (hình 3.26). Ngoài ra, ta có thể vào thư mục report để quan sát các báo

cáo khác liên quan đến diện tích, timing, chất lượng của RTL và đảm bảo không có vi phạm nào xảy ra.

### 3.2.6 Kiểm tra netlist

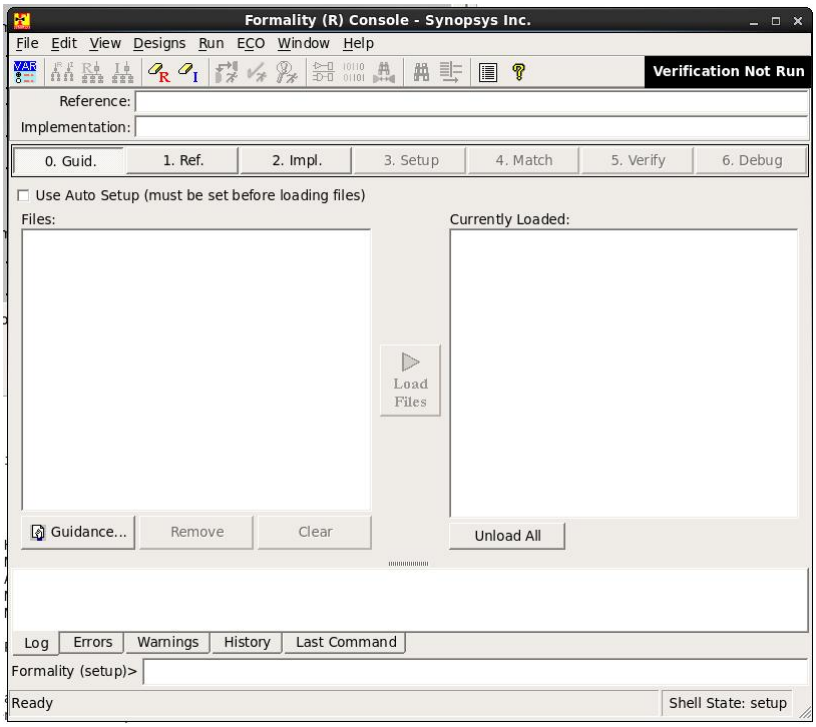
Bảng 3.6: Mô tả bước kiểm tra netlist.

| Đầu vào | Đầu ra | Các công đoạn |
|---|---|---|
| - Các file mã lập trình RTL đuôi .v<br>- Các file RTL đã tổng hợp (netlist) đuôi .v | - Các báo cáo (report) về quá trình Matching<br>- Các báo cáo (report) về quá trình Verify | - Lựa chọn các file thiết kế RTL<br>- Lựa chọn file RTL đã tổng hợp cùng thư viện tham khảo đi kèm<br>- Thực hiệm kiểm tra Matching và sửa lỗi (nếu có)<br>- Thực hiệm kiểm tra Verify và sửa lỗi (nếu có) |

| Bảng 5.6 là mô tả khái quát của bước kiểm tra netlist. |
|---|

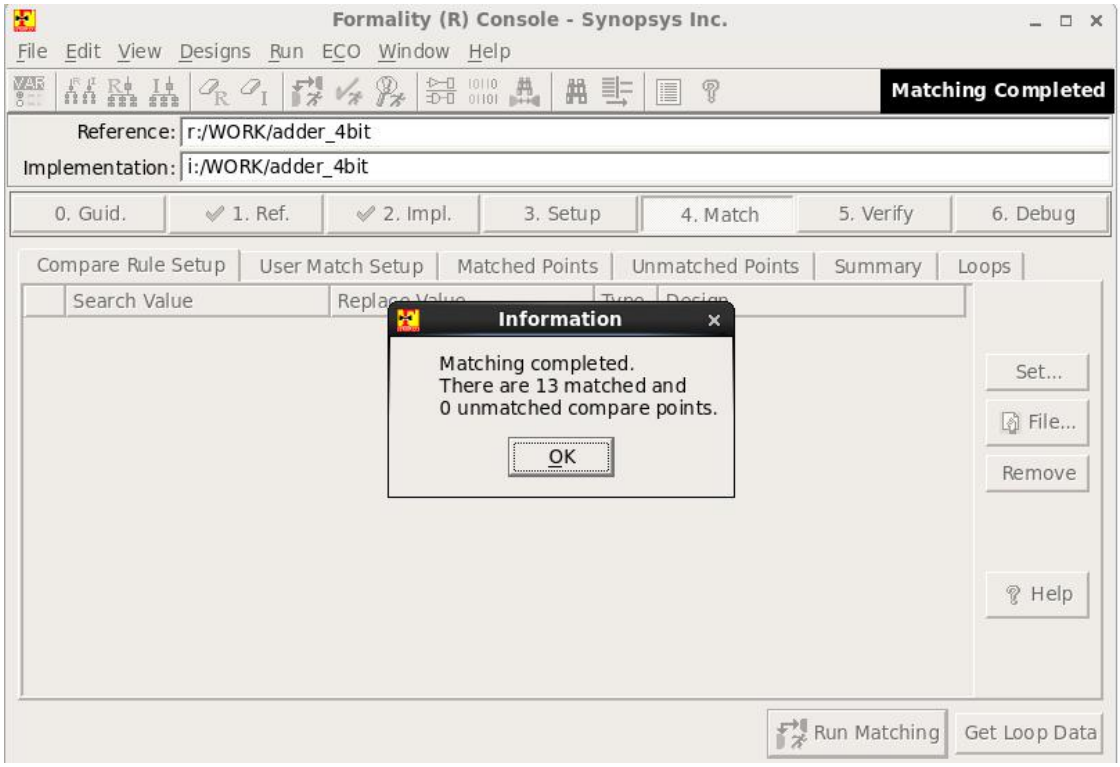
Để thực hiện kiểm tra netlist, ta cần dùng đến tool Formality của Synopsys. Trong **Terminal**, trỏ đường dẫn vào trong thư mục **05_nl_verif**. Sau đó nhập lệnh **formality&** để mở giao diện tool Formality (hình 3.27).

Các bước tiếp theo nhằm kiểm định netlist sẽ được thực hiện như sau:

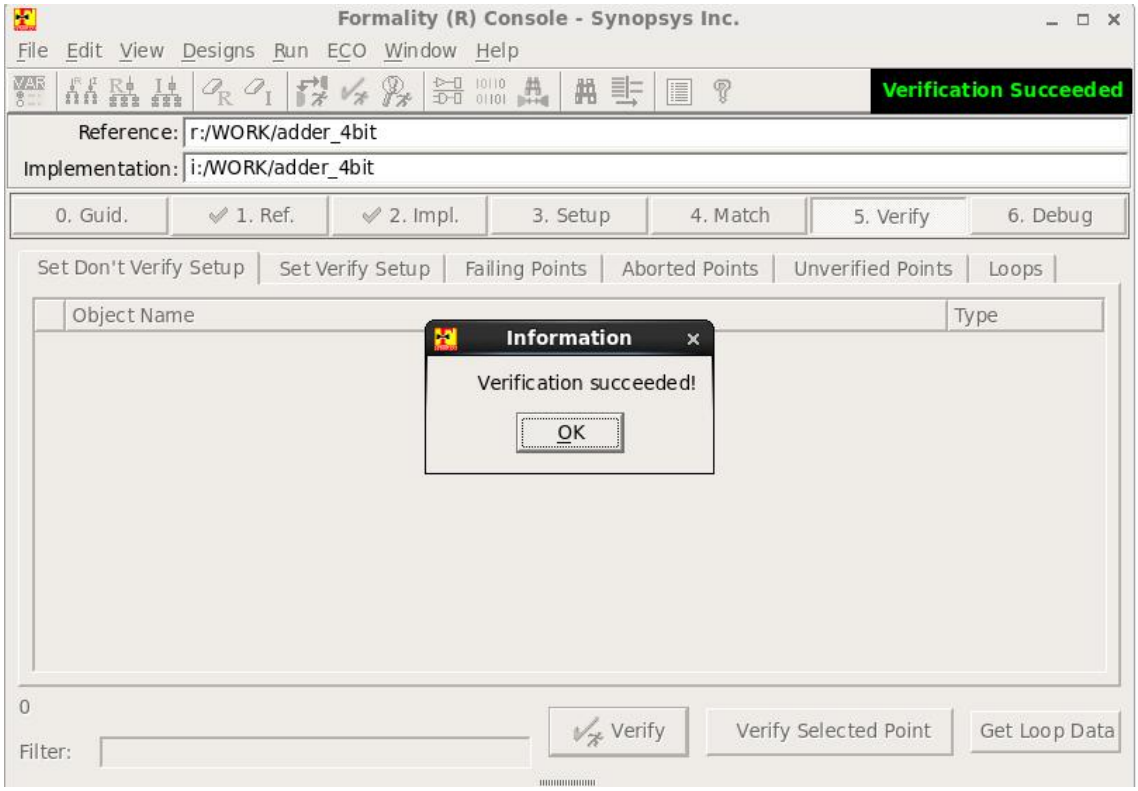



***Hình 3.27.*** *Giao diện của tool Formality.*

- Nhấn chọn tab **1.Ref**., nhấn nút **Verilog....** Chọn tất cả các file Verilog trong thư mục **02_rtl**. Sau đó nhấn **Load Files**. Chọn sang tab **3.Set Top Design** (ở dưới) và chọn module **adder_4bit** là top design (thiết kế tổng), rồi nhấn **Set Top.**
- Nhấn chọn tab **2.Impl.**, nhấn nút **Verilog....** Chọn file **lab_synth.netlist.v** trong thư mục **04_synth/report**. Sau đó nhấn **Load Files**. Chọn sang tab **2.Read DB Libraries**, nhấn nút **DB...**, chọn thư viện dùng cho Synthesize trong thư mục **04_synth/l**ib (ở đây là thư viện NangateOpen CellLibrary_typical.db). Sau đó nhấn **Load Files**. Chọn sang tab **3.Set Top Design** (ở dưới) và chọn module **adder_4bit** là top design (thiết kế tổng), rồi nhấn **Set Top**.
- Nhấn chọn tab **4.Match.**, nhấn chọn **Run Matching** để kiểm chứng sự tương đồng của RTL khi thiết kế và sau khi Synthesis. (hình 3.28)
- Nhấn chọn tab **5.Verify**., nhấn chọn **Verify** để kiểm chứng chức năng của RTL sau khi Synthesis. (hình 3.29).
- Nhấn chọn tab **6.Debug.**. Chức năng này được sử dụng khi có lỗi xảy ra trong quá trình so sánh RTL trước và sau Synthesize.




***Hình 3.28.*** *Kiểm tra tương đồng giữa RTL trước và sau khi Synthesize.*




***Hình 3.29.*** *Kiểm tra chức năng RTL sau khi Synthesize.*

### 3.2.7 Design For Test (DFT)

Design For Test là bước chèn thêm các khối kiểm tra vào trong thiết kế, nhằm kiểm tra lỗi của thiết kế sau khi đã ra thành phẩm. Bước này chỉ sử dụng đối với các thiết kế lớn, ở đó người ta cần sử dụng DFT để khoanh vùng, phân tích lỗi cục bộ của vi mạch. Đối với các thiết kế trong tài liệu này, bước này sẽ được bỏ qua do quy mô thiết kế không lớn và bước này là không cần thiết.

### 3.2.8 Phân tích thời gian tĩnh (STA)

Phân tích thời gian tĩnh, hay STA, có tác dụng kiểm chứng timing của thiết kế. Ở bài thực hành-thí nghiệm 1 này, do thiết kế còn đơn giản, hoàn toàn là mạch tổ hợp, chưa sử dụng đến tín hiệu clock nên bước STA này sẽ được bỏ qua. Trình bày cụ thể hơn sẽ được thể hiện trong bài thực hành-thí nghiệm 2.

### 3.2.9 Place & Route

Bảng 3.7 là mô tả khái quát của bước Place & Route.

Công đoạn cuối cùng của thiết kế vi mạch là Place & Route, nói cách khác là đặt các cell và nối dây, thực hiện thông qua tool IC Compiler của Synopsys. Thư mục **07_icc** sẽ là thư mục làm việc chính ở bước này. Trong thư mục này ta sẽ tạo 1 file có tên **icc_setup.TCL** nhằm setup cho bài thực hành-thí nghiệm trước khi đi vào thực hiện Place & Route. Mô tả của file này được thể hiện trong hình 3.30. Trong quá trình thực hiện Place & Route, dữ liệu sẽ

được lưu trữ dưới dạng thư viện MilkyWay.

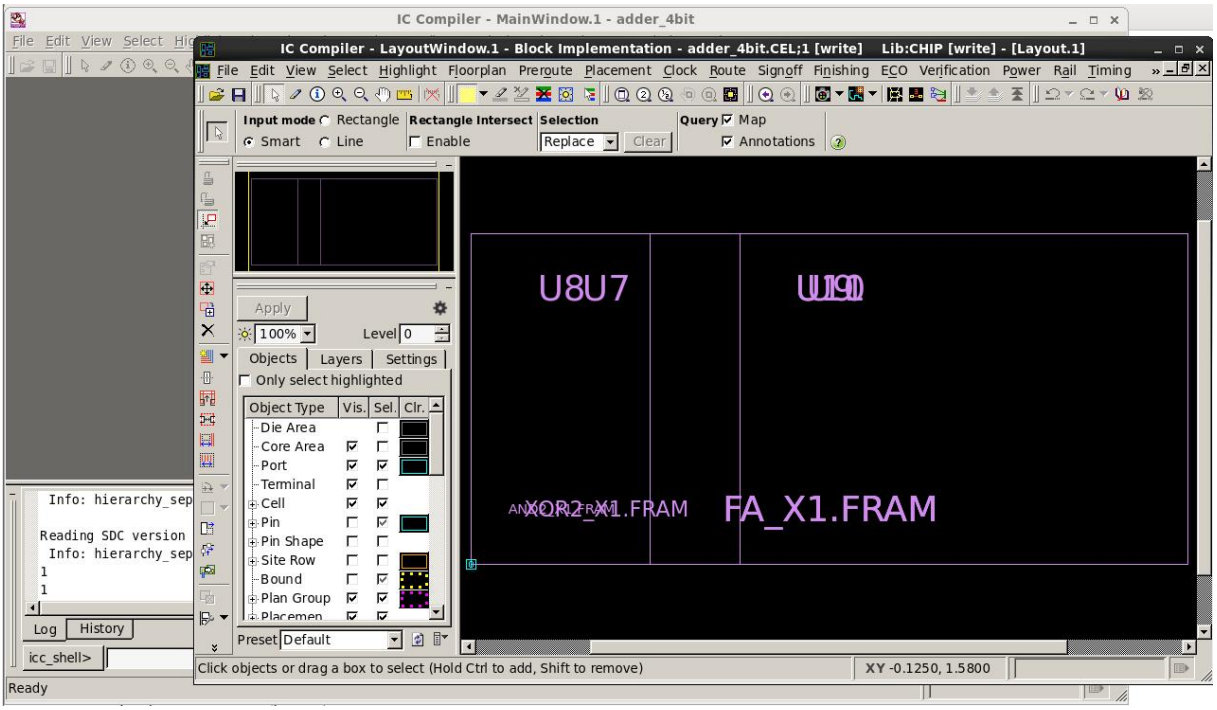

***Hình 3.30.*** *Mô tả nội dung file* ***icc_setup.TCL***

Sau khi bật Terminal, dùng lệnh **cd** để trỏ vào thư mục **07_icc**, sau đó nhập lệnh **icc_shell-gui-f icc_setup.TCL** để khởi động phần mềm IC Compiler đồng thời chạy setup thông qua file **icc_setup.TCL** (hình 3.31). Trong trường hợp tiếp tục làm việc trên một thiết kế đã tạo từ trước, ta sẽ không thêm option **-f icc_setup.TCL** bởi lưu ý là file này khi chạy sẽ xóa thư viện MilkyWay cũ và khi đó ta cần bắt đầu lại toàn bộ quá trình Place & Route.

Bảng 3.7: Mô tả bước Place & Route.

| Đầu vào | Đầu ra | Các công đoạn |
|---|---|---|
| - Các file RTL đã tổng hợp (netlist) đuôi .v<br>- Thư viện tham khảo ở cấp độ vật lý (phải trùng khớp với thư viện dùng cho Synthesis)<br>- File công nghệ đuôi .tf đi theo thư viện trên<br>- Các ràng buộc cung cấp bởi nhà máy (các file rule về DRC và LVS) | - Các báo cáo (report) về kiểm tra DRC<br>- Các báo cáo (report) về kiểm tra LVS<br>- File GDSII đuôi .gds để gửi đến nhà máy sản xuất | - Tạo thư viện Milkyway và import thiết kế<br>- Floor Planning: đặt các thành phần nền tảng lên bề mặt thiết kế (I/O, nguồn, đất,...)<br>- Placement: đặt các cell (trong thư viện tham khảo) ứng với netlist lên Floor vừa tạo<br>- Clock Tree Synthesis: tổng hợp, đi dây cho toàn bộ hệ thống clock, kiểm tra các vi phạm về timing và sửa lỗi (nếu có)<br>- Route: nối dây cho tất cả các kết nối còn lại trong thiết kế<br>- Xuất các file GDSII, .sdf nhằm kiểm tra các bước |

| | | |
|---|---|---|
| | | cuối cùng<br>- Kiểm tra DRC (Design Rule Check) nhằm tìm ra vi phạm với ràng buộc của nhà máy sản xuất (nếu có)<br>- Kiểm tra LVS (Layout vs Schematic) nhằm tìm ra khác biệt giữa Layout (sau khi Place & Route) và netlist trước đó<br>- Nếu không có vấn đề gì, file GDSII sẽ được gửi đến nhà máy để tiến hành sản xuất |

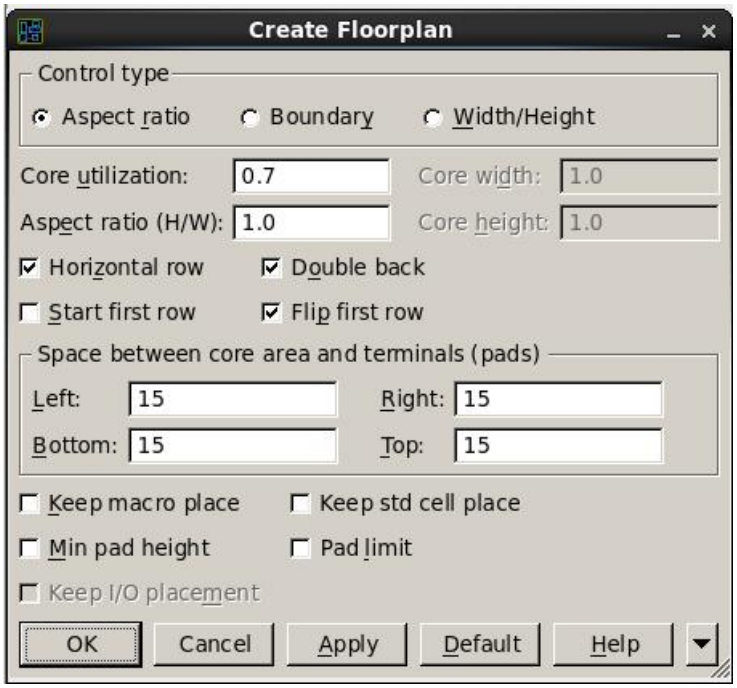



***Hình 3.31.*** *Khởi động phần mềm IC Compiler.*

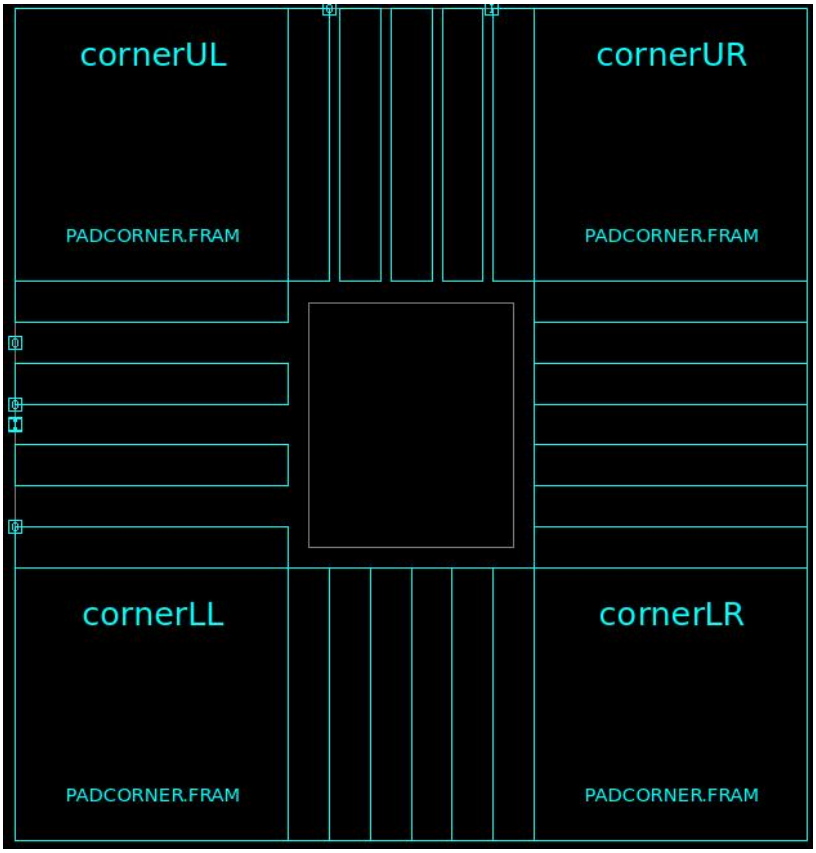
Sau khi khởi động phần mềm IC Compiler, ta sẽ thấy có hai cửa sổ xuất hiện, MainWindow và LayoutWindow. Trong đó MainWindow cho phép ta nhập lệnh tương tự trên Terminal, và LayoutWindow là nơi ta thực hiện hầu hết các tác vụ của Place & Route. Trong cửa sổ của LayoutWindow, ta có thể kéo chuột giữa, hoặc dùng các cặp phím "I"/"O" nhằm phóng to/thu nhỏ. Nhấn phím "F" cho phép ta căn chỉnh vừa vặn toàn bộ thiết kế trong cửa sổ.

Bước đầu tiên cần làm trong Place & Route là Floorplanning, trong đó các I/O, nguồn, đất được đặt trước tiên. Trong thư mục thí nghiệm (**icc_lab**) đã có thiết đặt sẵn cho 1 chip phổ thông và ta sẽ dùng chúng thông qua tác vụ sau: **File→Execute Script...**, sau đó chọn file **create_phy_cell.TCL** trong đường

dẫn **icc_lab/scripts** và nhấn **Open**. Sau đó chọn **Floorplan→Read Pin/Pad Physical Constraints...**, rồi nhấn phím chọn file bên cạnh box **Input file name** để chọn file **io.tdf** trong đường dẫn **icc_lab/design_data** để thiết lập các ràng buộc khi đặt I/O lên Floor, sau đó nhấn **Open** và **OK**. Người ta tiến hành Floorplannning bằng cách chọn **Floorplan→CreateFloorplan....** Một cửa sổ hiện ra, ta nhấn chọn tick cho box **Flip first row**, đồng thời thiết lập toàn bộ 15 cho phần **Space between core area and terminals (pads)** (đây là thiết đặt khoảng cách giữa phần core của vi mạch với các I/O) theo hình 3.32 rồi chọn **OK**. Kết quả Floorplanning được thể hiện trong hình 3.33.




***Hình 3.32.*** *Thiết lập cho Floorplanning.*

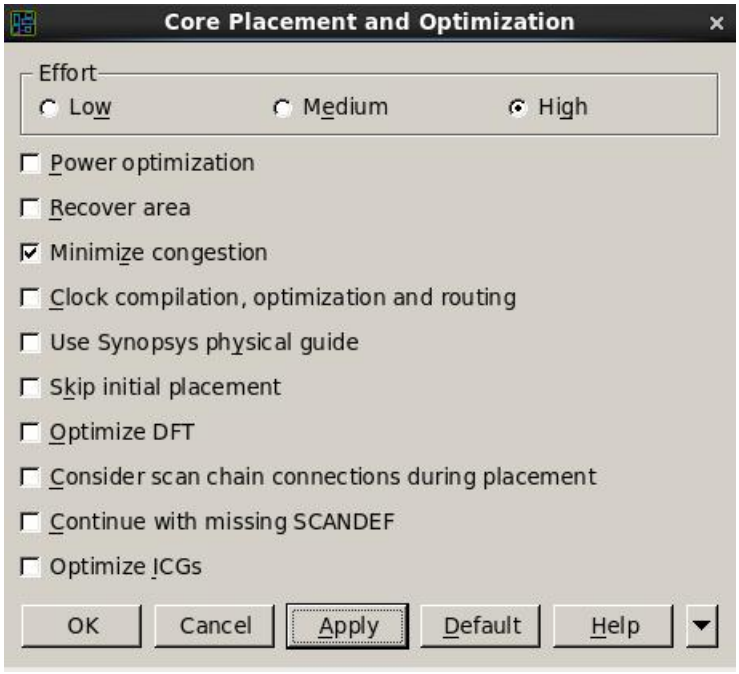



***Hình 3.33.*** *Kết quả chạy Floorplanning*

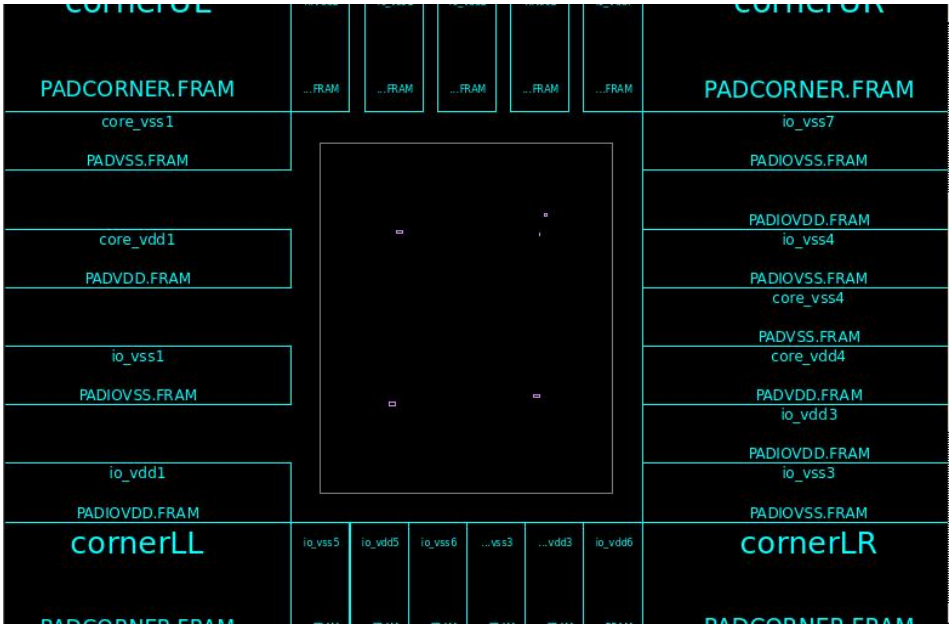
Việc tiếp theo cần làm là đặt các cell vào trong vùng core của vi mạch, điều này có thể được thực hiện thông qua chọn **Placement→Core Placement and Optimization**. Một cửa sổ hiện ra, ta nhấn chọn **High** ở phần **Effort** nhằm tăng hiệu quả của quá trình Placement. Ở các option phía dưới, nhấn chọn **Minimize congestion** nhằm hạn chế chồng lấn (hình 3.34), sau đó chọn **OK**. Kết quả quá trình Placement được thể hiện trên hình 3.35, trong đó các khối màu hồng tượng trưng cho các cell ứng với thư viện vật lý.

Công việc tiếp đến là đặt và nối trước các tín hiệu nguồn và đất:

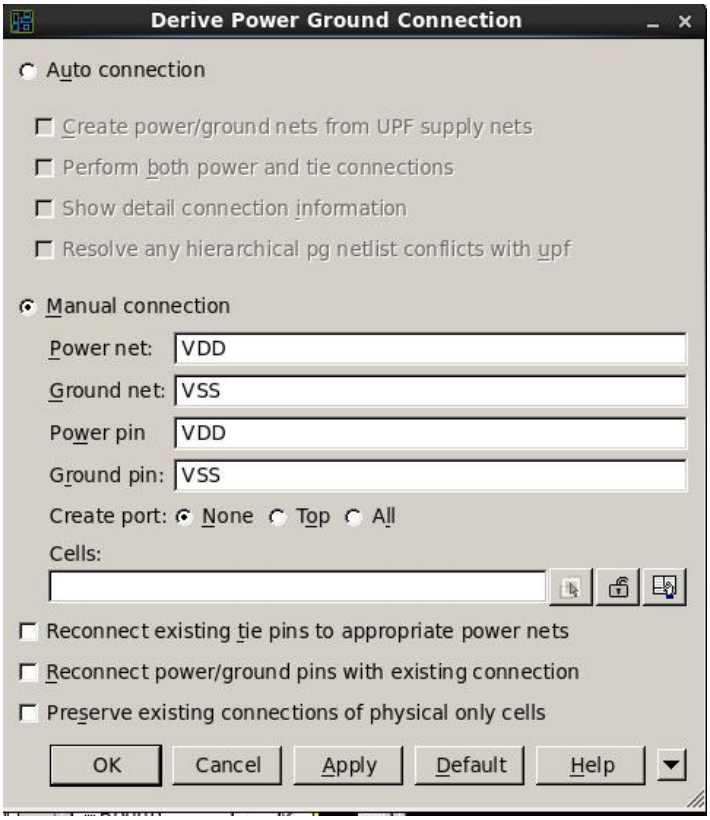
- Chọn **Preroute→Derive PG Connection...**,sau đó thiết lập như hình 3.36, và kết quả thể hiện trên Terminal như hình 3.37.
- Chọn **Preroute→Preroute Standard Cells...**, sau đó thiết lập như hình 3.38, và kết quả thể hiện trên LayoutWindow như hình 3.39 (các dây nguồn và đất được nối trước).




***Hình 3.34.*** *Thiết lập cho Placement.*



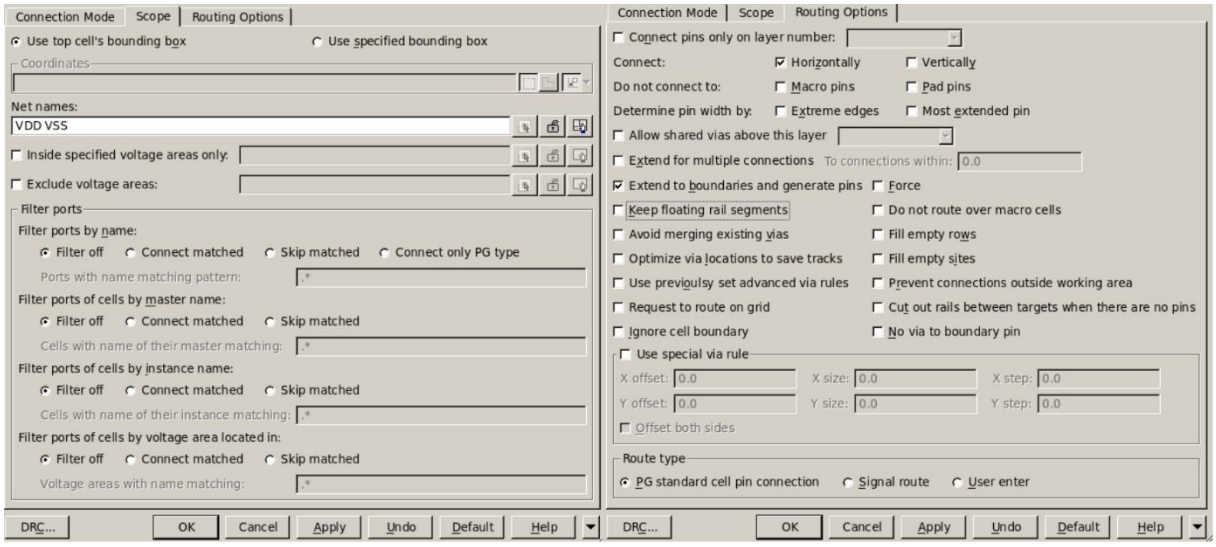

***Hình 3.35.*** *Kết quả chạy Placement.*



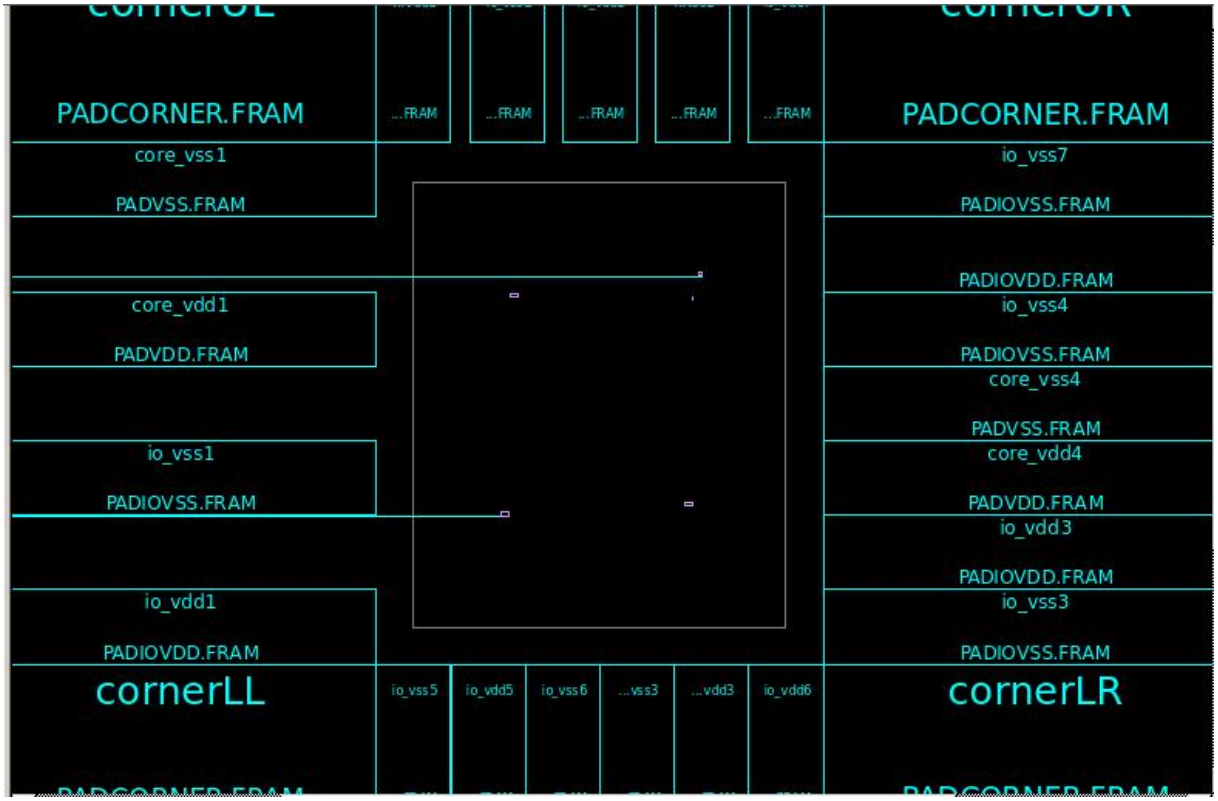

***Hình 3.36.*** *Thiết lập cho nhận tín hiệu nguồn/đất.*

```
icc_shell> derive_pg_connection -power_net {VDD} -ground_net {VSS} -power_pin {VDD} -ground_pin {VSS}
Information: connected 9 power ports and 9 ground ports
```

***Hình 3.37.*** *Kết quả chạy nhận tín hiệu nguồn/đất.*




***Hình 3.38.*** *Thiết lập cho nối trước tín hiệu nguồn/đất.*




***Hình 3.39.*** *Kết quả nối trước tín hiệu nguồn/đất.*

Công đoạn tiếp theo thông thường là tổng hợp cây clock (Clock Tree Syn thesis), tuy nhiên do thiết kế của thực hành-thí nghiệm 1 còn đơn giản, chưa có tín hiệu clock nên sẽ bỏ qua bước này.

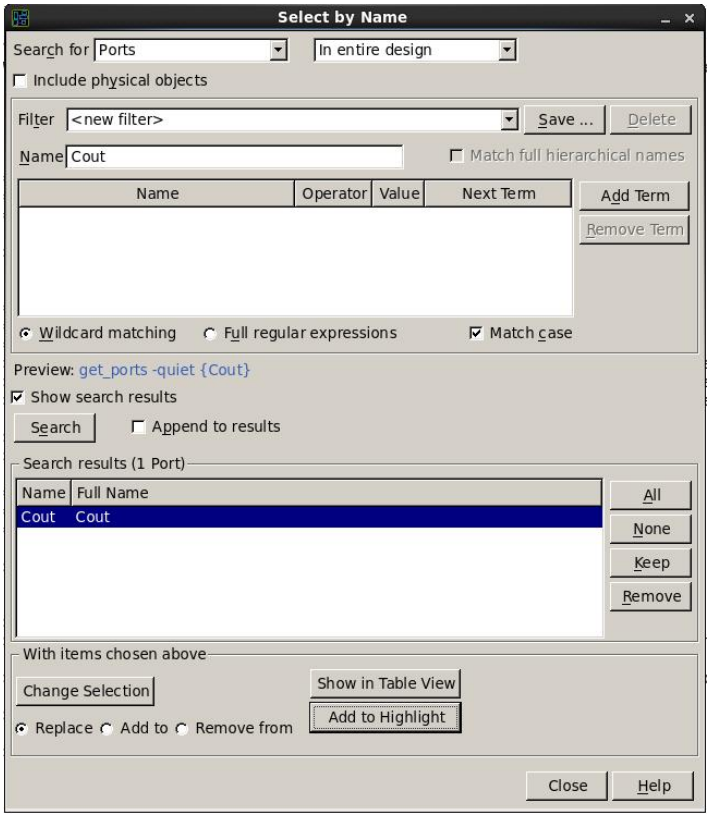
Công đoạn cuối cùng trong Place & Route là Routing, hay nói cách khác là nối tất cả các dây còn lại:

- Trước khi tiến hành Routing, ta cần kiểm tra khả năng cho phép đi dây thông qua chọn **Route→Check Routability...**, sau đó chọn OK và xem kết quả kiểm tra trong Terminal (hình 3.40).
- Trong trường hợp port hay cell bị blocked (tức bị cản trở), ta cần thay đổi vị trí của chúng. Để thuận tiện cho việc tìm Port hay Cell bị lỗi, chọn **Se lect→By Name...**, sau đó chọn loại tìm kiếm, tên→nhấn **Search**→chọn tín hiệu cần quan sát→nhấn **Add to Highlight** (hình 3.41). Cell hay Port bị lỗi sẽ được thể hiện sáng lên giúp ta dễ dàng tìm kiếm được chúng trên Floor. Để thay đổi vị trí của chúng, nhấn chuột phải lên LayoutWin dow, chọn **Window Stretch**, sau đó kéo thả sao cho bao phủ Cell hay Port cần di chuyển, sau đó kéo thả chính Cell hay Port đó đến vị trí thích hợp.
- Nếu kiểm tra Routing cho kết quả khả thi, nhấn chọn **Route→Core Routing and Optimization**. Sau đó thiết lập theo hình 3.42. Sau khi kiểm tra Terminal để đảm bảo không có lỗi trong khi chạy Routing, ta quan sát kết quả Routing như thể hiện trong hình 3.43.

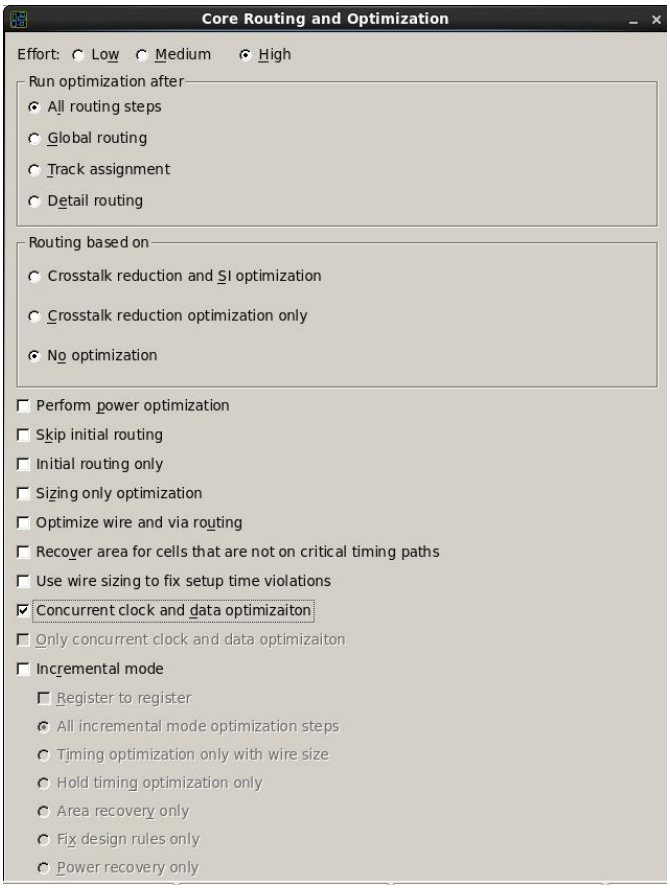
Sau khi Routing, ta cần kiểm tra DRC (Design Rule Check) nhằm kiểm tra ràng buộc theo yêu cầu nhà máy và LVS (Layout vs Schematic) nhằm kiểm tra liệu có khác biệt giữa netlist và mạch sau khi Place & Route. Tuy nhiên bài thực hành-thí nghiệm này sẽ chưa đề cập đến cách thực hiện hai tác vụ kiểm tra trên. Ta xem như mạch sau khi Routing đã hoàn hảo và sẵn sàng cho đóng gói cuối cùng. Để gửi thông tin về vi mạch đến nhà máy sản xuất, ta sẽ xuất file có định dạng GDSII (đuôi .gds) bằng cách chọn **File→Export→WriteStream**, sau đó chọn thự viện MilkyWay, Cell và tên file output như hình 3.44, và nhấn **OK**

```
==========================================
==    Check for out-of-boundary ports     ==
==========================================
>>> Port  S[1] at [230.070,609.930..230.140,610.000] metal2 is not in any grid region
>>> Port  S[0] at [274.530,609.930..274.600,610.000] metal2 is not in any grid region
>>> Port  Y[3] at [348.630,609.930..348.700,610.000] metal2 is not in any grid region

>>>>>> Warning: 3 ports not in any grid region
>>>>>> No out-of-boundary error found

==========================================
==        Check for blocked ports         ==
==========================================

>>>>>> Port blocked by layer constraints - min/max and freeze layer settings


>>>>>> Port blocked by check port access

>>>>>> No blocked port found

>>>>>> Net blocked by layer constraints - min/max and freeze layer settings

>>>>>> No blocked net found

End of check zrt routability
```

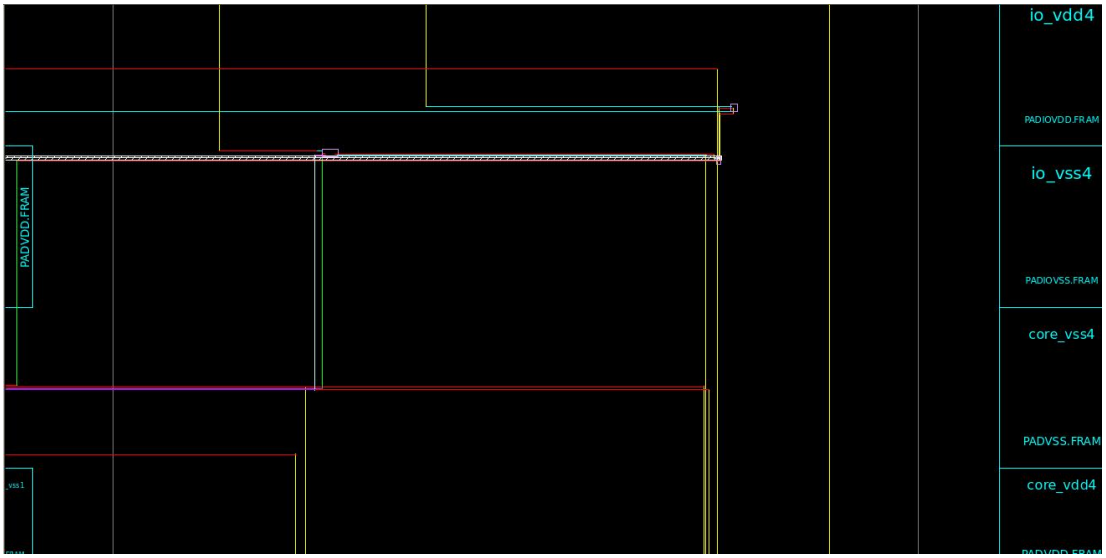
***Hình 3.40.*** *Kết quả kiểm tra khả năng Routing.*




***Hình 3.41.*** *Tìm Cell hay Port trên Floor.*




***Hình 3.42.*** *Thiết lập quá trình Routing.*




***Hình 3.43.*** *Kết quả chạy Routing.*

Đến đây là kết thúc hướng dẫn cho bài thực hành-thí nghiệm 1.