**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**KHOA KỸ THUẬT MÁY TÍNH**



**BÁO CÁO BÀI THỰC HÀNH**

|  |  |
| --- | --- |
| **HỌ VÀ TÊN:** | **NGUYỄN GIA BẢO NGỌC – 21520366** |
| **LỚP:** | **CE409.O21** |

**GIẢNG VIÊN HƯỚNG DẪN:**

**PHẠM THANH HÙNG**

**TP. HỒ CHÍ MINH – Tháng 4 năm 2024**

**MỤC LỤC:**

[Danh mục ảnh: II](#_Toc169182173)

[Danh mục bảng III](#_Toc169182174)

[Chỉnh sửa sau báo cáo 1](#_Toc169182175)

[Thiết kế và mô phỏng bộ định tuyến 16 bit 13](#_Toc169182176)

[**1.** **Mô tả thiết kế** 13](#_Toc169182177)

[**2.** **Chi tiết thiết kế** 15](#_Toc169182178)

[**3.** **Kiểm tra dạng sóng bằng phần mềm Quartus** 24](#_Toc169182179)

[Kiểm tra chức năng của bộ định tuyến bằng môi trường System Verilog 27](#_Toc169182180)

[**1.** **Interface “router\_io”** 27](#_Toc169182181)

[**2.** **Program automatic test** 28](#_Toc169182182)

[**3.** **Kết quả kiểm tra thiết kế** 30](#_Toc169182183)

[Kết luận 31](#_Toc169182184)

# Danh mục ảnh:

[Hình 1 - Kết quả mô phỏng (kiểm tra lại sau báo cáo) 12](#_Toc169182251)

[Hình 2 - Hình minh họa cho thiết kế bộ định tuyến 16bit 13](#_Toc169182252)

[Hình 3 - Một phần góc nhìn minh họa bên trong thiết kế 14](#_Toc169182253)

[Hình 4 - Kết quả phân tích RTL 24](#_Toc169182254)

[Hình 5 - Kết quả mô phỏng dạng sóng bằng Quartus (1) 25](#_Toc169182255)

[Hình 6 - Kết quả mô phỏng dạng sóng bằng Quartus (2) 25](#_Toc169182256)

[Hình 7 - Kết quả mô phỏng dạng sóng bằng Quartus (3) 26](#_Toc169182257)

[Hình 8 - Môi trường System Verilog dùng để kiểm tra thiết kế 27](#_Toc169182258)

[Hình 9 - Kết quả tại TCL Console 31](#_Toc169182259)

[Hình 10 - Kết quả mô tả dạng sóng bằng Vivado 31](#_Toc169182260)

# Danh mục bảng

[Bảng 1 - Test bench sau chỉnh sửa 1](#_Toc169182331)

[Bảng 2 - Thiết kế module Router 15](#_Toc169182332)

[Bảng 3 - Module port\_in 17](#_Toc169182333)

[Bảng 4 - Module port\_out 21](#_Toc169182334)

[Bảng 5 - moudule classifi 23](#_Toc169182335)

[Bảng 6 - Interface “router\_io” 27](#_Toc169182336)

[Bảng 7 - Chương trình "test" 28](#_Toc169182337)

# 

# **Chỉnh sửa sau báo cáo**

Nội dung chỉnh sửa sau báo cáo chủ yếu đến từ việc thay đổi file test bench sau cho kiểm tra thỏa mãn các trường hợp sau:

* Từ một port-in bất kì đến port-out bất kì.
* Từ nhiều port-in bất kì đến các port-out bất kì. (không xảy ra xung đột)
* Từ nhiều port-in bất kì đến các port-out bất kì. (có xảy ra xung đột)

Sau đây là file test bench sau khi chỉnh sửa:

Bảng 1 - Test bench sau chỉnh sửa

|  |
| --- |
| interface router\_io(input bit clock);  logic reset\_n;  logic [15:0] din;  logic [15:0] frame\_n;  logic [15:0] valid\_n;  logic [15:0] dout;  logic [15:0] frameo\_n;  logic [15:0] valido\_n;  logic [15:0] busy\_n;  clocking cb @(posedge clock);  //default input #1ns output #1ns;  output reset\_n;  output din;  output frame\_n;  output valid\_n;  input dout;  input frameo\_n;  input valido\_n;  input busy\_n;  endclocking  modport TB(clocking cb, output reset\_n);  endinterface  program automatic test (router\_io rtr\_io);  bit[3:0] sa;  bit[3:0] da;  bit[3:0] sa1;  bit[3:0] da1;  bit[3:0] sa2;  bit[3:0] da2;  bit[3:0] sa3;  bit[3:0] da3;  bit[3:0] sa4;  bit[3:0] da4;  bit[3:0] sa5;  bit[3:0] da5;  bit[3:0] sa6;  bit[3:0] da6;  bit[3:0] sa7;  bit[3:0] da7;  bit[3:0] sa8;  bit[3:0] da8;  bit[3:0] sa9;  bit[3:0] da9;  bit[3:0] sa10;  bit[3:0] da10;  bit[3:0] sa11;  bit[3:0] da11;  bit[3:0] sa12;  bit[3:0] da12;  bit[3:0] sa13;  bit[3:0] da13;  bit[3:0] sa14;  bit[3:0] da14;  bit[3:0] sa15;  bit[3:0] da15;    logic[7:0] payload[$];  logic[7:0] payload1[$];  logic[7:0] payload2[$];  logic[7:0] payload3[$];  logic[7:0] payload4[$];  logic[7:0] payload5[$];  logic[7:0] payload6[$];  logic[7:0] payload7[$];  logic[7:0] payload8[$];  logic[7:0] payload9[$];  logic[7:0] payload10[$];  logic[7:0] payload11[$];  logic[7:0] payload12[$];  logic[7:0] payload13[$];  logic[7:0] payload14[$];  logic[7:0] payload15[$];    logic[7:0] pkt2cmp\_payload\_testcase1[$];  logic[7:0] temp1;    logic[7:0] pkt2cmp\_payload\_testcase2[$];  logic[7:0] temp2;    logic[7:0] pkt2cmp\_payload\_testcase3\_1[$];  logic[7:0] temp3;    logic[7:0] pkt2cmp\_payload\_testcase3\_2[$];  logic[7:0] temp4;  initial begin  repeat (1) begin  gen();  reset();  fork  send();  recv();  join  disable fork;  end  // send\_addrs();  // send\_pad();  // send\_payload();  //$monitor(rtr\_io.dout);  #1000 $stop;  end  task reset();  rtr\_io.TB.reset\_n=0;  rtr\_io.frame\_n = 0;  //rtr\_io.frame\_n = 65535;  rtr\_io.valid\_n = 65535;  rtr\_io.din= 16'd0;  #10;  rtr\_io.TB.reset\_n=1;  rtr\_io.cb.din <=16'b0;  // #150;  // rtr\_io.frame\_n = 0;  endtask  task gen();  sa=0;  da=0;  payload.delete();  //repeat ($urandom\_range(2,4))  repeat (4) payload.push\_back($urandom);    sa1=1;  da1=3;  payload1.delete();  //repeat ($urandom\_range(2,4))  repeat (4) payload1.push\_back($urandom);    sa2=2;  da2=3;  payload2.delete();  //repeat ($urandom\_range(2,4))  repeat (4) payload2.push\_back($urandom);    sa3=3;  da3=3;  payload3.delete();  //repeat ($urandom\_range(2,4))  repeat (4) payload3.push\_back($urandom);    sa4=4;  da4=3;  payload4.delete();  //repeat ($urandom\_range(2,4))  repeat (4) payload4.push\_back($urandom);    sa5=5;  da5=3;  payload5.delete();  //repeat ($urandom\_range(2,4))  repeat (4) payload5.push\_back($urandom);    sa6=6;  da6=3;  payload6.delete();  //repeat ($urandom\_range(2,4))  repeat (4) payload6.push\_back($urandom);    sa7=7;  da7=3;  payload7.delete();  //repeat ($urandom\_range(2,4))  repeat (4) payload7.push\_back($urandom);    sa8=8;  da8=3;  payload8.delete();  //repeat ($urandom\_range(2,4))  repeat (4) payload8.push\_back($urandom);    sa9=9;  da9=3;  payload9.delete();  //repeat ($urandom\_range(2,4))  repeat (4) payload9.push\_back($urandom);    sa10=10;  da10=3;  payload10.delete();  //repeat ($urandom\_range(2,4))  repeat (4) payload10.push\_back($urandom);    sa11=11;  da11=11;  payload11.delete();  //repeat ($urandom\_range(2,4))  repeat (4) payload11.push\_back($urandom);    sa12=12;  da12=12;  payload12.delete();  //repeat ($urandom\_range(2,4))  repeat (4) payload12.push\_back($urandom);    sa13=13;  da13=13;  payload13.delete();  //repeat ($urandom\_range(2,4))  repeat (4) payload13.push\_back($urandom);    sa14=14;  da14=15;  payload14.delete();  //repeat ($urandom\_range(2,4))  repeat (4) payload14.push\_back($urandom);    sa15=15;  da15=14;  payload15.delete();  //repeat ($urandom\_range(2,4))  repeat (4) payload15.push\_back($urandom);  endtask  task send ();  send\_addrs();  send\_pad ();  send\_payload();  endtask  task send\_addrs();    rtr\_io.din[sa] =da[3];  rtr\_io.din[sa1] =da1[3];  rtr\_io.din[sa2] =da2[3];  rtr\_io.din[sa3] =da3[3];  rtr\_io.din[sa4] =da4[3];  rtr\_io.din[sa5] =da5[3];  rtr\_io.din[sa6] =da6[3];  rtr\_io.din[sa7] =da7[3];  rtr\_io.din[sa8] =da8[3];  rtr\_io.din[sa9] =da9[3];  rtr\_io.din[sa10] =da10[3];  rtr\_io.din[sa11] =da11[3];  rtr\_io.din[sa12] =da12[3];  rtr\_io.din[sa13] =da13[3];  rtr\_io.din[sa14] =da14[3];  rtr\_io.din[sa15] =da15[3];      #10 rtr\_io.din[sa] =da[2];  rtr\_io.din[sa1] =da1[2];  rtr\_io.din[sa2] =da2[2];  rtr\_io.din[sa3] =da3[2];  rtr\_io.din[sa4] =da4[2];  rtr\_io.din[sa5] =da5[2];  rtr\_io.din[sa6] =da6[2];  rtr\_io.din[sa7] =da7[2];  rtr\_io.din[sa8] =da8[2];  rtr\_io.din[sa9] =da9[2];  rtr\_io.din[sa10] =da10[2];  rtr\_io.din[sa11] =da11[2];  rtr\_io.din[sa12] =da12[2];  rtr\_io.din[sa13] =da13[2];  rtr\_io.din[sa14] =da14[2];  rtr\_io.din[sa15] =da15[2];    #10 rtr\_io.din[sa] =da[1];  rtr\_io.din[sa1] =da1[1];  rtr\_io.din[sa2] =da2[1];  rtr\_io.din[sa3] =da3[1];  rtr\_io.din[sa4] =da4[1];  rtr\_io.din[sa5] =da5[1];  rtr\_io.din[sa6] =da6[1];  rtr\_io.din[sa7] =da7[1];  rtr\_io.din[sa8] =da8[1];  rtr\_io.din[sa9] =da9[1];  rtr\_io.din[sa10] =da10[1];  rtr\_io.din[sa11] =da11[1];  rtr\_io.din[sa12] =da12[1];  rtr\_io.din[sa13] =da13[1];  rtr\_io.din[sa14] =da14[1];  rtr\_io.din[sa15] =da15[1];    #10 rtr\_io.din[sa] =da[0];  rtr\_io.din[sa1] =da1[0];  rtr\_io.din[sa2] =da2[0];  rtr\_io.din[sa3] =da3[0];  rtr\_io.din[sa4] =da4[0];  rtr\_io.din[sa5] =da5[0];  rtr\_io.din[sa6] =da6[0];  rtr\_io.din[sa7] =da7[0];  rtr\_io.din[sa8] =da8[0];  rtr\_io.din[sa9] =da9[0];  rtr\_io.din[sa10] =da10[0];  rtr\_io.din[sa11] =da11[0];  rtr\_io.din[sa12] =da12[0];  rtr\_io.din[sa13] =da13[0];  rtr\_io.din[sa14] =da14[0];  rtr\_io.din[sa15] =da15[0];    #10;    endtask  task send\_pad ();    rtr\_io.valid\_n= 16'b0000011111111100;  endtask  task send\_payload();  for (integer i =0; i<payload.size(); i=i+1) begin    if (rtr\_io.valid\_n[sa]==1'b0) rtr\_io.din[sa] = payload[i][7];  if (rtr\_io.valid\_n[sa1]==1'b0) rtr\_io.din[sa1] = payload1[i][7];  if (rtr\_io.valid\_n[sa2]==1'b0) rtr\_io.din[sa2] = payload2[i][7];  if (rtr\_io.valid\_n[sa3]==1'b0) rtr\_io.din[sa3] = payload3[i][7];  if (rtr\_io.valid\_n[sa4]==1'b0) rtr\_io.din[sa4] = payload4[i][7];  if (rtr\_io.valid\_n[sa5]==1'b0) rtr\_io.din[sa5] = payload5[i][7];  if (rtr\_io.valid\_n[sa6]==1'b0) rtr\_io.din[sa6] = payload6[i][7];  if (rtr\_io.valid\_n[sa7]==1'b0) rtr\_io.din[sa7] = payload7[i][7];  if (rtr\_io.valid\_n[sa8]==1'b0) rtr\_io.din[sa8] = payload8[i][7];  if (rtr\_io.valid\_n[sa9]==1'b0) rtr\_io.din[sa9] = payload9[i][7];  if (rtr\_io.valid\_n[sa10]==1'b0) rtr\_io.din[sa10] = payload10[i][7];  if (rtr\_io.valid\_n[sa11]==1'b0) rtr\_io.din[sa11] = payload11[i][7];  if (rtr\_io.valid\_n[sa12]==1'b0) rtr\_io.din[sa12] = payload12[i][7];  if (rtr\_io.valid\_n[sa13]==1'b0) rtr\_io.din[sa13] = payload13[i][7];  if (rtr\_io.valid\_n[sa14]==1'b0)rtr\_io.din[sa14] = payload14[i][7];  if (rtr\_io.valid\_n[sa15]==1'b0) rtr\_io.din[sa15] = payload15[i][7];    #10 if (rtr\_io.valid\_n[sa]==1'b0) rtr\_io.din[sa] = payload[i][6];  if (rtr\_io.valid\_n[sa1]==1'b0) rtr\_io.din[sa1] = payload1[i][6];  if (rtr\_io.valid\_n[sa2]==1'b0) rtr\_io.din[sa2] = payload2[i][6];  if (rtr\_io.valid\_n[sa3]==1'b0) rtr\_io.din[sa3] = payload3[i][6];  if (rtr\_io.valid\_n[sa4]==1'b0) rtr\_io.din[sa4] = payload4[i][6];  if (rtr\_io.valid\_n[sa5]==1'b0) rtr\_io.din[sa5] = payload5[i][6];  if (rtr\_io.valid\_n[sa6]==1'b0) rtr\_io.din[sa6] = payload6[i][6];  if (rtr\_io.valid\_n[sa7]==1'b0) rtr\_io.din[sa7] = payload7[i][6];  if (rtr\_io.valid\_n[sa8]==1'b0) rtr\_io.din[sa8] = payload8[i][6];  if (rtr\_io.valid\_n[sa9]==1'b0) rtr\_io.din[sa9] = payload9[i][6];  if (rtr\_io.valid\_n[sa10]==1'b0) rtr\_io.din[sa10] = payload10[i][6];  if (rtr\_io.valid\_n[sa11]==1'b0) rtr\_io.din[sa11] = payload11[i][6];  if (rtr\_io.valid\_n[sa12]==1'b0) rtr\_io.din[sa12] = payload12[i][6];  if (rtr\_io.valid\_n[sa13]==1'b0) rtr\_io.din[sa13] = payload13[i][6];  if (rtr\_io.valid\_n[sa14]==1'b0) rtr\_io.din[sa14] = payload14[i][6];  if (rtr\_io.valid\_n[sa15]==1'b0) rtr\_io.din[sa15] = payload15[i][6];    #10 if (rtr\_io.valid\_n[sa]==1'b0) rtr\_io.din[sa] = payload[i][5];  if (rtr\_io.valid\_n[sa1]==1'b0) rtr\_io.din[sa1] = payload1[i][5];  if (rtr\_io.valid\_n[sa2]==1'b0) rtr\_io.din[sa2] = payload2[i][5];  if (rtr\_io.valid\_n[sa3]==1'b0) rtr\_io.din[sa3] = payload3[i][5];  if (rtr\_io.valid\_n[sa4]==1'b0) rtr\_io.din[sa4] = payload4[i][5];  if (rtr\_io.valid\_n[sa5]==1'b0) rtr\_io.din[sa5] = payload5[i][5];  if (rtr\_io.valid\_n[sa6]==1'b0) rtr\_io.din[sa6] = payload6[i][5];  if (rtr\_io.valid\_n[sa7]==1'b0) rtr\_io.din[sa7] = payload7[i][5];  if (rtr\_io.valid\_n[sa8]==1'b0) rtr\_io.din[sa8] = payload8[i][5];  if (rtr\_io.valid\_n[sa9]==1'b0) rtr\_io.din[sa9] = payload9[i][5];  if (rtr\_io.valid\_n[sa10]==1'b0) rtr\_io.din[sa10] = payload10[i][5];  if (rtr\_io.valid\_n[sa11]==1'b0) rtr\_io.din[sa11] = payload11[i][5];  if (rtr\_io.valid\_n[sa12]==1'b0) rtr\_io.din[sa12] = payload12[i][5];  if (rtr\_io.valid\_n[sa13]==1'b0) rtr\_io.din[sa13] = payload13[i][5];  if (rtr\_io.valid\_n[sa14]==1'b0) rtr\_io.din[sa14] = payload14[i][5];  if (rtr\_io.valid\_n[sa15]==1'b0) rtr\_io.din[sa15] = payload15[i][5];    #10 if (rtr\_io.valid\_n[sa]==1'b0) rtr\_io.din[sa] = payload[i][4];  if (rtr\_io.valid\_n[sa1]==1'b0) rtr\_io.din[sa1] = payload1[i][4];  if (rtr\_io.valid\_n[sa2]==1'b0) rtr\_io.din[sa2] = payload2[i][4];  if (rtr\_io.valid\_n[sa3]==1'b0) rtr\_io.din[sa3] = payload3[i][4];  if (rtr\_io.valid\_n[sa4]==1'b0) rtr\_io.din[sa4] = payload4[i][4];  if (rtr\_io.valid\_n[sa5]==1'b0) rtr\_io.din[sa5] = payload5[i][4];  if (rtr\_io.valid\_n[sa6]==1'b0) rtr\_io.din[sa6] = payload6[i][4];  if (rtr\_io.valid\_n[sa7]==1'b0) rtr\_io.din[sa7] = payload7[i][4];  if (rtr\_io.valid\_n[sa8]==1'b0) rtr\_io.din[sa8] = payload8[i][4];  if (rtr\_io.valid\_n[sa9]==1'b0) rtr\_io.din[sa9] = payload9[i][4];  if (rtr\_io.valid\_n[sa10]==1'b0) rtr\_io.din[sa10] = payload10[i][4];  if (rtr\_io.valid\_n[sa11]==1'b0) rtr\_io.din[sa11] = payload11[i][4];  if (rtr\_io.valid\_n[sa12]==1'b0) rtr\_io.din[sa12] = payload12[i][4];  if (rtr\_io.valid\_n[sa13]==1'b0) rtr\_io.din[sa13] = payload13[i][4];  if (rtr\_io.valid\_n[sa14]==1'b0) rtr\_io.din[sa14] = payload14[i][4];  if (rtr\_io.valid\_n[sa15]==1'b0) rtr\_io.din[sa15] = payload15[i][4];    #10 if (rtr\_io.valid\_n[sa]==1'b0) rtr\_io.din[sa] = payload[i][3];  if (rtr\_io.valid\_n[sa1]==1'b0) rtr\_io.din[sa1] = payload1[i][3];  if (rtr\_io.valid\_n[sa2]==1'b0) rtr\_io.din[sa2] = payload2[i][3];  if (rtr\_io.valid\_n[sa3]==1'b0) rtr\_io.din[sa3] = payload3[i][3];  if (rtr\_io.valid\_n[sa4]==1'b0) rtr\_io.din[sa4] = payload4[i][3];  if (rtr\_io.valid\_n[sa5]==1'b0) rtr\_io.din[sa5] = payload5[i][3];  if (rtr\_io.valid\_n[sa6]==1'b0) rtr\_io.din[sa6] = payload6[i][3];  if (rtr\_io.valid\_n[sa7]==1'b0) rtr\_io.din[sa7] = payload7[i][3];  if (rtr\_io.valid\_n[sa8]==1'b0) rtr\_io.din[sa8] = payload8[i][3];  if (rtr\_io.valid\_n[sa9]==1'b0) rtr\_io.din[sa9] = payload9[i][3];  if (rtr\_io.valid\_n[sa10]==1'b0) rtr\_io.din[sa10] = payload10[i][3];  if (rtr\_io.valid\_n[sa11]==1'b0) rtr\_io.din[sa11] = payload11[i][3];  if (rtr\_io.valid\_n[sa12]==1'b0) rtr\_io.din[sa12] = payload12[i][3];  if (rtr\_io.valid\_n[sa13]==1'b0) rtr\_io.din[sa13] = payload13[i][3];  if (rtr\_io.valid\_n[sa14]==1'b0) rtr\_io.din[sa14] = payload14[i][3];  if (rtr\_io.valid\_n[sa15]==1'b0) rtr\_io.din[sa15] = payload15[i][3];    #10 if (rtr\_io.valid\_n[sa]==1'b0) rtr\_io.din[sa] = payload[i][2];  if (rtr\_io.valid\_n[sa1]==1'b0) rtr\_io.din[sa1] = payload1[i][2];  if (rtr\_io.valid\_n[sa2]==1'b0) rtr\_io.din[sa2] = payload2[i][2];  if (rtr\_io.valid\_n[sa3]==1'b0) rtr\_io.din[sa3] = payload3[i][2];  if (rtr\_io.valid\_n[sa4]==1'b0) rtr\_io.din[sa4] = payload4[i][2];  if (rtr\_io.valid\_n[sa5]==1'b0) rtr\_io.din[sa5] = payload5[i][2];  if (rtr\_io.valid\_n[sa6]==1'b0) rtr\_io.din[sa6] = payload6[i][2];  if (rtr\_io.valid\_n[sa7]==1'b0) rtr\_io.din[sa7] = payload7[i][2];  if (rtr\_io.valid\_n[sa8]==1'b0) rtr\_io.din[sa8] = payload8[i][2];  if (rtr\_io.valid\_n[sa9]==1'b0) rtr\_io.din[sa9] = payload9[i][2];  if (rtr\_io.valid\_n[sa10]==1'b0) rtr\_io.din[sa10] = payload10[i][2];  if (rtr\_io.valid\_n[sa11]==1'b0) rtr\_io.din[sa11] = payload11[i][2];  if (rtr\_io.valid\_n[sa12]==1'b0) rtr\_io.din[sa12] = payload12[i][2];  if (rtr\_io.valid\_n[sa13]==1'b0) rtr\_io.din[sa13] = payload13[i][2];  if (rtr\_io.valid\_n[sa14]==1'b0) rtr\_io.din[sa14] = payload14[i][2];  if (rtr\_io.valid\_n[sa15]==1'b0) rtr\_io.din[sa15] = payload15[i][2];    #10 if (rtr\_io.valid\_n[sa]==1'b0) rtr\_io.din[sa] = payload[i][1];  if (rtr\_io.valid\_n[sa1]==1'b0) rtr\_io.din[sa1] = payload1[i][1];  if (rtr\_io.valid\_n[sa2]==1'b0) rtr\_io.din[sa2] = payload2[i][1];  if (rtr\_io.valid\_n[sa3]==1'b0) rtr\_io.din[sa3] = payload3[i][1];  if (rtr\_io.valid\_n[sa4]==1'b0) rtr\_io.din[sa4] = payload4[i][1];  if (rtr\_io.valid\_n[sa5]==1'b0) rtr\_io.din[sa5] = payload5[i][1];  if (rtr\_io.valid\_n[sa6]==1'b0) rtr\_io.din[sa6] = payload6[i][1];  if (rtr\_io.valid\_n[sa7]==1'b0) rtr\_io.din[sa7] = payload7[i][1];  if (rtr\_io.valid\_n[sa8]==1'b0) rtr\_io.din[sa8] = payload8[i][1];  if (rtr\_io.valid\_n[sa9]==1'b0) rtr\_io.din[sa9] = payload9[i][1];  if (rtr\_io.valid\_n[sa10]==1'b0) rtr\_io.din[sa10] = payload10[i][1];  if (rtr\_io.valid\_n[sa11]==1'b0) rtr\_io.din[sa11] = payload11[i][1];  if (rtr\_io.valid\_n[sa12]==1'b0) rtr\_io.din[sa12] = payload12[i][1];  if (rtr\_io.valid\_n[sa13]==1'b0) rtr\_io.din[sa13] = payload13[i][1];  if (rtr\_io.valid\_n[sa14]==1'b0) rtr\_io.din[sa14] = payload14[i][1];  if (rtr\_io.valid\_n[sa15]==1'b0) rtr\_io.din[sa15] = payload15[i][1];    #10 if (rtr\_io.valid\_n[sa]==1'b0) rtr\_io.din[sa] = payload[i][0];  if (rtr\_io.valid\_n[sa1]==1'b0) rtr\_io.din[sa1] = payload1[i][0];  if (rtr\_io.valid\_n[sa2]==1'b0) rtr\_io.din[sa2] = payload2[i][0];  if (rtr\_io.valid\_n[sa3]==1'b0) rtr\_io.din[sa3] = payload3[i][0];  if (rtr\_io.valid\_n[sa4]==1'b0) rtr\_io.din[sa4] = payload4[i][0];  if (rtr\_io.valid\_n[sa5]==1'b0) rtr\_io.din[sa5] = payload5[i][0];  if (rtr\_io.valid\_n[sa6]==1'b0) rtr\_io.din[sa6] = payload6[i][0];  if (rtr\_io.valid\_n[sa7]==1'b0) rtr\_io.din[sa7] = payload7[i][0];  if (rtr\_io.valid\_n[sa8]==1'b0) rtr\_io.din[sa8] = payload8[i][0];  if (rtr\_io.valid\_n[sa9]==1'b0) rtr\_io.din[sa9] = payload9[i][0];  if (rtr\_io.valid\_n[sa10]==1'b0) rtr\_io.din[sa10] = payload10[i][0];  if (rtr\_io.valid\_n[sa11]==1'b0) rtr\_io.din[sa11] = payload11[i][0];  if (rtr\_io.valid\_n[sa12]==1'b0) rtr\_io.din[sa12] = payload12[i][0];  if (rtr\_io.valid\_n[sa13]==1'b0) rtr\_io.din[sa13] = payload13[i][0];  if (rtr\_io.valid\_n[sa14]==1'b0) rtr\_io.din[sa14] = payload14[i][0];  if (rtr\_io.valid\_n[sa15]==1'b0) rtr\_io.din[sa15] = payload15[i][0];  #10;  end    endtask  function bit compare(logic[7:0] a[$], logic[7:0] b[$]);  for (integer i =0; i<a.size(); i=i+1) begin  if (a[i]==b[i])  compare = 1'b1;  else begin  compare = 1'b0;  break;  end  end  endfunction  task recv ();  get\_payload();  check();  endtask  task check();  if (compare (payload, pkt2cmp\_payload\_testcase1)) $display ("PASS TESTCASE 1");  else $display ("FAIL TESTCASE 1");  if (compare (payload1, pkt2cmp\_payload\_testcase2)) $display ("PASS TESTCASE 2");  else $display ("FAIL TESTCASE 2");  if (compare (payload15, pkt2cmp\_payload\_testcase3\_1)&&compare (payload14, pkt2cmp\_payload\_testcase3\_2)) $display ("PASS TESTCASE 3");  else $display ("FAIL TESTCASE 3");  endtask  task get\_payload();  logic enable;  pkt2cmp\_payload\_testcase1.delete();  pkt2cmp\_payload\_testcase2.delete();  @(negedge rtr\_io.frameo\_n[da])  begin  for (integer i =0; i<4; i=i+1) begin  temp1[7]= rtr\_io.dout[da];  temp2[7]= rtr\_io.dout[3];  temp3[7]= rtr\_io.dout[14];  temp4[7]= rtr\_io.dout[15];  #10 temp1[6]= rtr\_io.dout[da];  temp2[6]= rtr\_io.dout[3];  temp3[6]= rtr\_io.dout[14];  temp4[6]= rtr\_io.dout[15];  #10 temp1[5]= rtr\_io.dout[da];  temp2[5]= rtr\_io.dout[3];  temp3[5]= rtr\_io.dout[14];  temp4[5]= rtr\_io.dout[15];  #10 temp1[4]= rtr\_io.dout[da];  temp2[4]= rtr\_io.dout[3];  temp3[4]= rtr\_io.dout[14];  temp4[4]= rtr\_io.dout[15];  #10 temp1[3]= rtr\_io.dout[da];  temp2[3]= rtr\_io.dout[3];  temp3[3]= rtr\_io.dout[14];  temp4[3]= rtr\_io.dout[15];  #10 temp1[2]= rtr\_io.dout[da];  temp2[2]= rtr\_io.dout[3];  temp3[2]= rtr\_io.dout[14];  temp4[2]= rtr\_io.dout[15];  #10 temp1[1]= rtr\_io.dout[da];  temp2[1]= rtr\_io.dout[3];  temp3[1]= rtr\_io.dout[14];  temp4[1]= rtr\_io.dout[15];  #10 temp1[0]= rtr\_io.dout[da];  temp2[0]= rtr\_io.dout[3];  temp3[0]= rtr\_io.dout[14];  temp4[0]= rtr\_io.dout[15];  #10;  pkt2cmp\_payload\_testcase1.push\_back(temp1);  pkt2cmp\_payload\_testcase2.push\_back(temp2);  pkt2cmp\_payload\_testcase3\_1.push\_back(temp3);  pkt2cmp\_payload\_testcase3\_2.push\_back(temp4);  end  end  endtask  endprogram  module router\_test\_top();  parameter simulation\_cycle= 10;  bit SystemClock;  router\_io top\_io(SystemClock);  test t (top\_io);  Router dut (  .reset\_n (top\_io.reset\_n),  .frame\_n (top\_io.frame\_n),  .valid\_n (top\_io.valid\_n),  .din (top\_io.din),  .clock (top\_io.clock),  .frameo\_n(top\_io.frameo\_n),  .valido\_n(top\_io.valido\_n),  .busy\_n (top\_io.busy\_n),  .dout (top\_io.dout)  );  initial begin  SystemClock=0;  forever begin  #(simulation\_cycle/2);  SystemClock= ~SystemClock;  end  //$stop;  end  endmodule |

Cụ thể kế hoạch kiểm tra sẽ là:

* Port-in[0] truyền dữ liệu đến port\_out[0].
* Port\_in[1:10] truyền dữ liệu đến port\_out[3] ( kết quả mong đợi port\_out[3] sẽ nhận kết quả từ port\_in[1] do có độ ưu tiên cao nhất).
* Port\_in[11:13] truyền dữ liệu đến port\_out[11:13] theo thứ tự tương ứng.
* Port\_in[14] truyền dữ liệu đến port\_out[15] và port\_in[15] truyền dữ liệu đến port\_out[14].

A screenshot of a computer screen

Description automatically generated

Hình 1 - Kết quả mô phỏng (kiểm tra lại sau báo cáo)

Kết quả mô phỏng giống như kế hoạch kiểm tra mong muốn.

**Thiết kế và mô phỏng bộ định tuyến 16 bit**

1. **Mô tả thiết kế**

Bộ định tuyến bao gồm 16 cổng vào và 16 cổng ra. Mỗi cổng vào và cổng ra bao gồm 3 tín hiệu: “data”, “frame” và “valid”. Những tín hiệu này được biểu diễn dưới dạng bit-vector, din[15:0], frame\_n[15:0], valid\_n[15:0], dout[15:0], frameo\_n[15:0] và valido\_n[15:0].

Để lái dự liệu vào cổng vào, vị trí bit cụ thể tương ứng với số cổng phải được chỉ định. Ví dụ, nếu muốn điều khiển cổng vào số 3, thì các tín hiệu tương ứng sẽ là din[3], frame\_n[3] và valid\_n[3].

Để lấy mẫu tại cổng ra, vị trí bit cụ thể tương ứng với số cổng phải được chỉ định. Ví dụ, nếu muốn lấy mẫu cổng ra số 7, thì các tín hiệu tương ứng sẽ là dout[7], frameo\_n[7] và valido\_n[7].

A diagram of a router

Description automatically generated

Hình 2 - Hình minh họa cho thiết kế bộ định tuyến 16bit

Thiết kế hoạt động tích cực theo cạnh, dữ liệu đầu vào và đầu ra liên tục (1bit/1 clock). Các gói tin được truyền với độ dài tùy biến, bao gồm 2 phần là: Header và Payload. Mỗi gói tin có thể được truyền từ cổng vào bất kì đến cổng ra bất kì. Không có bộ đệm và lan truyền nội.

Các tín hiệu đầu vào:

* Frame\_n: cạnh xuống chỉ bit đầu tiên của dữ liệu ngõ vào, cạnh lên chỉ bit cuối cùng của dữ liệu từ ngõ vào.
* Din: gói tin được truyền tại gõ vào bao gồm Header (destination address và padding bit) và Payload.
* Valid\_n: mức thấp nếu như dữ liệu trong payload tồn tại, ngược lại trong các trường hợp khác.

Dữ liệu đầu ra bao gồm: dout, frame\_o, valid\_o. Dữ liệu đầu ra hợp lệ chỉ khi frame\_o và valid\_o mức thấp.

Tín hiệu “reset” bất đồng bộ, theo đó tín hiệu sẽ tồn tại trong ít nhất 1 chu kì và sau đó 15 chu kì thì bắt đầu quá trình tuyền dữ liệu.

A diagram of a network

Description automatically generated

Hình 3 - Một phần góc nhìn minh họa bên trong thiết kế

## **Chi tiết thiết kế**

Bộ định tuyến sẽ được thiết kế bằng ngôn ngữ Verilog, bao gồm các file “Router.v”, “port\_in.v”, “port\_out.v” và “classifi.v”. Theo đó “Router.v” là “top design” bao gồm tất cả các file khác.

Bảng 2 - Thiết kế module Router

|  |
| --- |
| `timescale 1ns / 1ps  module Router (reset\_n, frame\_n, valid\_n, din, clock, frameo\_n, valido\_n, busy\_n ,dout);  input reset\_n;  input[15:0] frame\_n;  input[15:0] valid\_n;  input[15:0] din;  input clock;  output[15:0] frameo\_n;  output[15:0] valido\_n;  output[15:0] busy\_n;  output[15:0] dout;    wire[15:0] dout0, dout1, dout2, dout3, dout4, dout5, dout6, dout7;  wire[15:0] dout8, dout9, dout10, dout11, dout12, dout13, dout14, dout15;  wire[15:0] di0, di1, di2, di3, di4, di5, di6, di7;  wire[15:0] di8, di9, di10, di11, di12, di13, di14, di15;  wire[3:0] request0, request1, request2, request3, request4, request5, request6, request7;  wire[3:0] request8, request9, request10, request11, request12, request13, request14, request15;  wire[15:0] fra\_in0, fra\_in1, fra\_in2, fra\_in3, fra\_in4, fra\_in5, fra\_in6, fra\_in7;  wire[15:0] fra\_in8, fra\_in9, fra\_in10, fra\_in11, fra\_in12, fra\_in13, fra\_in14, fra\_in15;    //(data\_out, reset, frame, valid, data\_in, clk, request);  port\_in Pin0 (dout0, reset\_n, frame\_n[0], valid\_n[0], din[0], clock, request0);  port\_in Pin1 (dout1, reset\_n, frame\_n[1], valid\_n[1], din[1], clock, request1);  port\_in Pin2 (dout2, reset\_n, frame\_n[2], valid\_n[2], din[2], clock, request2);  port\_in Pin3 (dout3, reset\_n, frame\_n[3], valid\_n[3], din[3], clock, request3);  port\_in Pin4 (dout4, reset\_n, frame\_n[4], valid\_n[4], din[4], clock, request4);  port\_in Pin5 (dout5, reset\_n, frame\_n[5], valid\_n[5], din[5], clock, request5);  port\_in Pin6 (dout6, reset\_n, frame\_n[6], valid\_n[6], din[6], clock, request6);  port\_in Pin7 (dout7, reset\_n, frame\_n[7], valid\_n[7], din[7], clock, request7);  port\_in Pin8 (dout8, reset\_n, frame\_n[8], valid\_n[8], din[8], clock, request8);  port\_in Pin9 (dout9, reset\_n, frame\_n[9], valid\_n[9], din[9], clock, request9);  port\_in Pin10 (dout10, reset\_n, frame\_n[10], valid\_n[10], din[10], clock, request10);  port\_in Pin11 (dout11, reset\_n, frame\_n[11], valid\_n[11], din[11], clock, request11);  port\_in Pin12 (dout12, reset\_n, frame\_n[12], valid\_n[12], din[12], clock, request12);  port\_in Pin13 (dout13, reset\_n, frame\_n[13], valid\_n[13], din[13], clock, request13);  port\_in Pin14 (dout14, reset\_n, frame\_n[14], valid\_n[14], din[14], clock, request14);  port\_in Pin15 (dout15, reset\_n, frame\_n[15], valid\_n[15], din[15], clock, request15);  assign di0 = {dout15[0], dout14[0], dout13[0], dout12[0], dout11[0], dout10[0], dout9[0], dout8[0], dout7[0], dout6[0], dout5[0], dout4[0], dout3[0], dout2[0], dout1[0], dout0[0]};    assign di1 = {dout15[1], dout14[1], dout13[1], dout12[1], dout11[1], dout10[1], dout9[1], dout8[1], dout7[1], dout6[1], dout5[1], dout4[1], dout3[1], dout2[1], dout1[1], dout0[1]};    assign di2 = {dout15[2], dout14[2], dout13[2], dout12[2], dout11[2], dout10[2], dout9[2], dout8[2], dout7[2], dout6[2], dout5[2], dout4[2], dout3[2], dout2[2], dout1[2], dout0[2]};  assign di3 = {dout15[3], dout14[3], dout13[3], dout12[3], dout11[3], dout10[3], dout9[3], dout8[3], dout7[3], dout6[3], dout5[3], dout4[3], dout3[3], dout2[3], dout1[3], dout0[3]};    assign di4 = {dout15[4], dout14[4], dout13[4], dout12[4], dout11[4], dout10[4], dout9[4], dout8[4], dout7[4], dout6[4], dout5[4], dout4[4], dout3[4], dout2[4], dout1[4], dout0[4]};  assign di5 = {dout15[5], dout14[5], dout13[5], dout12[5], dout11[5], dout10[5], dout9[5], dout8[5], dout7[5], dout6[5], dout5[5], dout4[5], dout3[5], dout2[5], dout1[5], dout0[5]};  assign di6 = {dout15[6], dout14[6], dout13[6], dout12[6], dout11[6], dout10[6], dout9[6], dout8[6], dout7[6], dout6[6], dout5[6], dout4[6], dout3[6], dout2[6], dout1[6], dout0[6]};    assign di7 = {dout15[7], dout14[7], dout13[7], dout12[7], dout11[7], dout10[7], dout9[7], dout8[7], dout7[7], dout6[7], dout5[7], dout4[7], dout3[7], dout2[7], dout1[7], dout0[7]};    assign di8 = {dout15[8], dout14[8], dout13[8], dout12[8], dout11[8], dout10[8], dout9[8], dout8[8], dout7[8], dout6[8], dout5[8], dout4[8], dout3[8], dout2[8], dout1[8], dout0[8]};    assign di9 = {dout15[9], dout14[9], dout13[9], dout12[9], dout11[9], dout10[9], dout9[9], dout8[9], dout7[9], dout6[9], dout5[9], dout4[9], dout3[9], dout2[9], dout1[9], dout0[9]};    assign di10 = {dout15[10], dout14[10], dout13[10], dout12[10], dout11[10], dout10[10], dout9[10], dout8[10], dout7[10], dout6[10], dout5[10], dout4[10], dout3[10], dout2[10], dout1[10], dout0[10]};  assign di11 = {dout15[11], dout14[11], dout13[11], dout12[11], dout11[11], dout10[11], dout9[11], dout8[11], dout7[11], dout6[11], dout5[11], dout4[11], dout3[11], dout2[11], dout1[11], dout0[11]};    assign di12 = {dout15[12], dout14[12], dout13[12], dout12[12], dout11[12], dout10[12], dout9[12], dout8[12], dout7[12], dout6[12], dout5[12], dout4[11], dout3[11], dout2[12], dout1[12], dout0[12]};    assign di13 = {dout15[13], dout14[13], dout13[13], dout12[13], dout11[13], dout10[13], dout9[13], dout8[13], dout7[13], dout6[13], dout5[13], dout4[13], dout3[13], dout2[13], dout1[13], dout0[13]};    assign di14 = {dout15[14], dout14[14], dout13[14], dout12[14], dout11[14], dout10[14], dout9[14], dout8[14], dout7[14], dout6[14], dout5[14], dout4[14], dout3[14], dout2[14], dout1[14], dout0[14]};  assign di15 = {dout15[15], dout14[15], dout13[15], dout12[15], dout11[15], dout10[15], dout9[15], dout8[15], dout7[15], dout6[15], dout5[15], dout4[15], dout3[15], dout2[15], dout1[15], dout0[15]};      classifi classi (fra\_in0, fra\_in1,fra\_in2, fra\_in3,fra\_in4,fra\_in5,fra\_in6,fra\_in7,  fra\_in8, fra\_in9,fra\_in10, fra\_in11,fra\_in12,fra\_in13,fra\_in14,fra\_in15,  request0, request1, request2, request3, request4, request5, request6, request7,  request8, request9, request10, request11, request12, request13, request14, request15);    port\_out Pout0 (frameo\_n[0], valido\_n[0], dout[0], di0, fra\_in0, frame\_n, valid\_n, busy\_n[0]);  port\_out Pout1 (frameo\_n[1], valido\_n[1], dout[1], di1, fra\_in1, frame\_n, valid\_n, busy\_n[1]);  port\_out Pout2 (frameo\_n[2], valido\_n[2], dout[2], di2, fra\_in2, frame\_n, valid\_n, busy\_n[2]);  port\_out Pout3 (frameo\_n[3], valido\_n[3], dout[3], di3, fra\_in3, frame\_n, valid\_n, busy\_n[3]);  port\_out Pout4 (frameo\_n[4], valido\_n[4], dout[4], di4, fra\_in4, frame\_n, valid\_n, busy\_n[4]);  port\_out Pout5 (frameo\_n[5], valido\_n[5], dout[5], di5, fra\_in5, frame\_n, valid\_n, busy\_n[5]);  port\_out Pout6 (frameo\_n[6], valido\_n[6], dout[6], di6, fra\_in6, frame\_n, valid\_n, busy\_n[6]);  port\_out Pout7 (frameo\_n[7], valido\_n[7], dout[7], di7, fra\_in7, frame\_n, valid\_n, busy\_n[7]);  port\_out Pout8 (frameo\_n[8], valido\_n[8], dout[8], di8, fra\_in8, frame\_n, valid\_n, busy\_n[8]);  port\_out Pout9 (frameo\_n[9], valido\_n[9], dout[9], di9, fra\_in9, frame\_n, valid\_n, busy\_n[9]);  port\_out Pout10 (frameo\_n[10], valido\_n[10], dout[10], di10, fra\_in10, frame\_n, valid\_n, busy\_n[10]);  port\_out Pout11 (frameo\_n[11], valido\_n[11], dout[11], di11, fra\_in11, frame\_n, valid\_n, busy\_n[11]);  port\_out Pout12 (frameo\_n[12], valido\_n[12], dout[12], di12, fra\_in12, frame\_n, valid\_n, busy\_n[12]);  port\_out Pout13 (frameo\_n[13], valido\_n[13], dout[13], di13, fra\_in13, frame\_n, valid\_n, busy\_n[13]);  port\_out Pout14 (frameo\_n[14], valido\_n[14], dout[14], di14, fra\_in14, frame\_n, valid\_n, busy\_n[14]);  port\_out Pout15 (frameo\_n[15], valido\_n[15], dout[15], di15, fra\_in15, frame\_n, valid\_n, busy\_n[15]);    endmodule |

Các module “port\_in” nhận dữ liệu vào là tín hiệu “din”, “frame\_n”, “valid\_ n”, “reset” và “clock” cho kết quả ngõ ra là “dout” (payload) và “request” (destination address). “dout” của “port-in” bao gồm 16 bit tương ứng với 16 tín hiệu sẽ được truyền vào 16 “port\_out”, tuy nhiên chỉ có tín hiệu “dout[i]” ứng với giá trị “request” là có dữ liệu. Sau đó các tín hiệu này sẽ được dùng để tạo ra tín hiệu “di”. Các tín hiệu “di” là các bit-vector, di thứ i tương ứng với các tín hiệu được truyền vào “port\_out” thứ i, ví dụ di0 là vector giá trị bit được gửi vào port\_out[0] từ 16 port\_in (có thể có dữ liệu hoặc mang tín hiệu trở kháng cao). Song song đó là bước xử lí của module “classifi” module này phân loại dựa trên các giá trị request của 16 port\_in để cho ra vector “fra\_in” cho biết port\_in nào đã truy cập vào một port\_out bất kì. Ví dụ, fra\_in5 = 0000000000000101 điều này có nghĩa là tại port\_out[5] có sự truyền dữ liệu từ port\_in[0] và port\_in[2]. Sau khi đã có dữ liệu là để xác định được dữ liệu được truyền vào port\_out (di) và truyền từ port\_in nào (fra\_in) ta có thể xác định được tín hiệu đầu ra tại ngõ ra của bộ định tuyến. Dưới đây lần lượt là thiết kế mức hành vi của các module “port\_in”, “port\_out” và “classifi”.

Bảng 3 - Module port\_in

|  |
| --- |
| module port\_in (data\_out, reset, frame, valid, data\_in, clk, request);  output reg[15:0] data\_out;  output reg [3:0] request;  input reset;  input frame;  input valid;  input data\_in;  input clk;    parameter[1:0] start= 0, header=1, pad=2, payload=3;  reg[1:0] state, next\_state;  reg[2:0] count;  reg dout;  reg[3:0] address;  wire din;  assign din= data\_in;  always @(posedge clk) begin  if (!reset) begin  state=0;  end  else state= next\_state;  end  always @(\*) begin  case (state)  start: begin  if (frame==0) begin  next\_state= header;  end  else next\_state= start;  end  header:  if (frame==1) next\_state= start;  else  begin  if (count<3) next\_state= header;  else  if(valid==1) next\_state= pad;  else next\_state= payload;  end  pad:  if (frame==1) next\_state=start;  else  if (valid==0) next\_state= payload;  else next\_state= pad;  payload:  if (frame==1) next\_state= start;  else next\_state= payload;  default: next\_state= start;  endcase  end  always @(posedge clk) begin  if (state==header) begin address= (address<<1)|{3'b0,din}; count= count +1'b1; end  else count=0;  if (state==start) address=0;  end  always @(\*) begin  case(state)  start: dout = 1'bz;  header: dout = 1'bz;  pad: dout = 1'bz;  payload: if (valid==1)  dout= 1'bz;  else dout= data\_in;  default: dout= 1'bz;  endcase  end  always @(\*) begin  if (state==payload||state==pad)  case(address)  4'b0000:  begin  data\_out={15'bz,dout};  request = 4'b0000;  end  4'b0001:  begin  data\_out= {14'bz,dout,1'bz};  request = 4'b0001;  end  4'b0010:  begin  data\_out= {13'bz,dout,2'bz};  request = 4'b0010;  end  4'b0011:  begin  data\_out= {12'bz,dout,3'bz};  request = 4'b0011;  end  4'b0100:  begin  data\_out= {11'bz,dout,4'bz};  request = 4'b0100;  end  4'b0101:  begin  data\_out= {10'bz,dout,5'bz};  request = 4'b0101;  end  4'b0110:  begin  data\_out= {9'bz,dout,6'bz};  request =4'b0110;  end  4'b0111:  begin  data\_out= {8'bz,dout,7'bz};;  request = 4'b0111;  end  4'b1000:  begin  data\_out= {7'bz,dout,8'bz};  request = 4'b1000;  end  4'b1001:  begin  data\_out= {6'bz,dout,9'bz};  request = 4'b1001;  end  4'b1010:  begin  data\_out= {5'bz,dout,10'bz};  request = 4'b1010;  end  4'b1011:  begin  data\_out= {4'bz,dout,11'bz};  request = 4'b1011;  end  4'b1100:  begin  data\_out= {3'bz,dout,12'bz};  request = 4'b1100;  end  4'b1101:  begin  data\_out= {2'bz,dout,13'bz};  request = 4'b1101;  end  4'b1110:  begin  data\_out= {1'bz,dout,14'bz};  request = 4'b1110;  end  4'b1111:  begin  data\_out= {dout,15'bz};  request = 4'b1111;  end  default:  begin  data\_out= 16'bz;  request = 4'bz;  end  endcase  else begin  data\_out= 16'bz;  request = 4'bz;  end  end  endmodule |

Module “port\_out” trong trường hợp có nhiều hơn 1 “port\_in” được truyền dữ liệu thì port\_out sẽ phân xử theo độ ưu tiên giảm dần từ port\_in0 đến port\_in15.

Bảng 4 - Module port\_out

|  |
| --- |
| module port\_out (frame\_out, valid\_out, data\_out, din, fra\_in, frame\_in, valid\_in, busy\_n);    input[15:0] din;  input[15:0] frame\_in;  input[15:0] valid\_in;  input[15:0] fra\_in;  output reg frame\_out;  output reg valid\_out;  output reg data\_out;  output reg busy\_n;      always@(\*) begin  if (!frame\_in[0]&&fra\_in[0]&&!valid\_in[0]) begin  data\_out= din[0];  frame\_out= frame\_in[0];  valid\_out= valid\_in[0];  busy\_n= !(frame\_in[0]);  end else  if (!frame\_in[1]&&fra\_in[1]&&!valid\_in[1]) begin  data\_out= din[1];  frame\_out= frame\_in[1];  valid\_out= valid\_in[1];  busy\_n= !(frame\_in[1]);  end else  if (!frame\_in[2]&&fra\_in[2]&&!valid\_in[2]) begin  data\_out= din[2];  frame\_out= frame\_in[2];  valid\_out= valid\_in[2];  busy\_n= !(frame\_in[2]);  end else  if (!frame\_in[3]&&fra\_in[3]&&!valid\_in[3]) begin  data\_out= din[3];  frame\_out= frame\_in[3];  valid\_out= valid\_in[3];  busy\_n= !(frame\_in[3]);  end else  if (!frame\_in[4]&&fra\_in[4]&&!valid\_in[4]) begin  data\_out= din[4];  frame\_out= frame\_in[4];  valid\_out= valid\_in[4];  busy\_n= !(frame\_in[4]);  end else  if (!frame\_in[5]&&fra\_in[5]&&!valid\_in[5]) begin  data\_out= din[5];  frame\_out= frame\_in[5];  valid\_out= valid\_in[5];  busy\_n= !(frame\_in[5]);  end else  if (!frame\_in[6]&&fra\_in[6]&&!valid\_in[6]) begin  data\_out= din[6];  frame\_out= frame\_in[6];  valid\_out= valid\_in[6];  busy\_n= !(frame\_in[6]);  end else  if (!frame\_in[7]&&fra\_in[7]&&!valid\_in[7]) begin  data\_out= din[7];  frame\_out= frame\_in[7];  valid\_out= valid\_in[7];  busy\_n= !(frame\_in[7]);  end else  if (!frame\_in[8]&&fra\_in[8]&&!valid\_in[8]) begin  data\_out= din[8];  frame\_out= frame\_in[8];  valid\_out= valid\_in[8];  busy\_n= !(frame\_in[8]);  end else  if (!frame\_in[9]&&fra\_in[9]&&!valid\_in[9]) begin  data\_out= din[9];  frame\_out= frame\_in[9];  valid\_out= valid\_in[9];  busy\_n= !(frame\_in[9]);  end else  if (!frame\_in[10]&&fra\_in[10]&&!valid\_in[10]) begin  data\_out= din[10];  frame\_out= frame\_in[10];  valid\_out= valid\_in[10];  busy\_n= !(frame\_in[10]);  end else  if (!frame\_in[11]&&fra\_in[11]&&!valid\_in[11]) begin  data\_out= din[11];  frame\_out= frame\_in[11];  valid\_out= valid\_in[11];  busy\_n= !(frame\_in[11]);  end else  if (!frame\_in[12]&&fra\_in[12]&&!valid\_in[12]) begin  data\_out= din[12];  frame\_out= frame\_in[12];  valid\_out= valid\_in[12];  busy\_n= !(frame\_in[12]);  end else  if (!frame\_in[13]&&fra\_in[13]&&!valid\_in[13]) begin  data\_out= din[13];  frame\_out= frame\_in[13];  valid\_out= valid\_in[13];  busy\_n= !(frame\_in[13]);  end else  if (!frame\_in[14]&&fra\_in[14]&&!valid\_in[14]) begin  data\_out= din[14];  frame\_out= frame\_in[14];  valid\_out= valid\_in[14];  busy\_n= !(frame\_in[14]);  end else  if (!frame\_in[15]&&fra\_in[15]&&!valid\_in[15]) begin  data\_out= din[15];  frame\_out= frame\_in[15];  valid\_out= valid\_in[15];  busy\_n= !(frame\_in[15]);  end else begin  frame\_out=1;  valid\_out=1;  data\_out= 1'bz;  busy\_n=0;  end  end  endmodule |

Sau đoạn mã sau đây minh họa cho hoạt động của module “classifi”:

Bảng 5 - moudule classifi

|  |
| --- |
| module classifi ( fra\_in0, fra\_in1,fra\_in2, fra\_in3,fra\_in4,fra\_in5,fra\_in6,fra\_in7,  fra\_in8, fra\_in9,fra\_in10, fra\_in11,fra\_in12,fra\_in13,fra\_in14,fra\_in15,  request0, request1, request2, request3, request4, request5, request6, request7,  request8, request9, request10, request11, request12, request13, request14, request15  );  output[15:0] fra\_in0, fra\_in1, fra\_in2, fra\_in3, fra\_in4, fra\_in5, fra\_in6, fra\_in7;  output[15:0] fra\_in8, fra\_in9, fra\_in10, fra\_in11, fra\_in12, fra\_in13, fra\_in14, fra\_in15;  input[3:0] request0, request1, request2, request3, request4, request5, request6, request7;  input[3:0] request8, request9, request10, request11, request12, request13, request14, request15;    assign fra\_in0[0] = (request0==4'b0000);  assign fra\_in0[1] = (request1==4'b0000);  assign fra\_in0[2] = (request2==4'b0000);  assign fra\_in0[3] = (request3==4'b0000);  assign fra\_in0[4] = (request4==4'b0000);  assign fra\_in0[5] = (request5==4'b0000);  assign fra\_in0[6] = (request6==4'b0000);  assign fra\_in0[7] = (request7==4'b0000);  assign fra\_in0[8] = (request8==4'b0000);  assign fra\_in0[9] = (request9==4'b0000);  assign fra\_in0[10] = (request10==4'b0000);  assign fra\_in0[11] = (request11==4'b0000);  assign fra\_in0[12] = (request12==4'b0000);  assign fra\_in0[13] = (request13==4'b0000);  assign fra\_in0[14] = (request14==4'b0000);  assign fra\_in0[15] = (request15==4'b0000);  //tương tự cho các output còn lại |

Sau khi hoàn thành thiết kế, tiến hành phân tích RTL cho ra kết quả như hình sau:

A computer screen shot of a circuit board

Description automatically generated

Hình 4 - Kết quả phân tích RTL

## **Kiểm tra dạng sóng bằng phần mềm Quartus**

Hai ảnh bên dưới là các anh minh chứng cho việc kiểm tra thiết kế bằng dạng sóng với phần mềm Quartus. Theo đó tất cả các port\_in từ 0 đến 15 sẽ truyền dữ liêu cho các port\_out từ 0 đến 15. Qua kiểm tra dạng sóng ta có thể thấy các port\_out đã truyền dữ liệu từ port\_in đúng như mong muốn.

A screenshot of a computer screen

Description automatically generated

Hình 5 - Kết quả mô phỏng dạng sóng bằng Quartus (1)

A screenshot of a computer

Description automatically generated

Hình 6 - Kết quả mô phỏng dạng sóng bằng Quartus (2)

Tiếp tục kiểm tra một trường hợp khác, khi cả hai ngõ vào din[0] và din[1] cùng muốn truyền dữ liệu đến dout[0] ta có thể thấy dout[0] sẽ nhận dữ liệu từ din[0] do là port\_in có độ ưu tiên cao hơn.

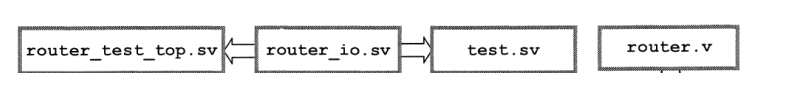
A screenshot of a computer screen

Description automatically generated

Hình 7 - Kết quả mô phỏng dạng sóng bằng Quartus (3)

# **Kiểm tra chức năng của bộ định tuyến bằng môi trường System Verilog**

Môi trường System Verilog dùng để kiểm tra thiết kế được tạo thành từ các thành phần như hình sau:



Hình 8 - Môi trường System Verilog dùng để kiểm tra thiết kế

## **Interface “router\_io”**

Là bộ phận dùng để kết nối DUT và chương trình “test”. Interface “router\_io” được tạo thành như bảng sau:

Bảng 6 - Interface “router\_io”

|  |
| --- |
| interface router\_io(input bit clock);  logic reset\_n;  logic [15:0] din;  logic [15:0] frame\_n;  logic [15:0] valid\_n;  logic [15:0] dout;  logic [15:0] frameo\_n;  logic [15:0] valido\_n;  logic [15:0] busy\_n;  clocking cb @(posedge clock);  //default input #1ns output #1ns;  output reset\_n;  output din;  output frame\_n;  output valid\_n;  input dout;  input frameo\_n;  input valido\_n;  input busy\_n;  endclocking  modport TB(clocking cb, output reset\_n);  endinterface |

## **Program automatic test**

Chương trình “test” thực hiện nhiệm vụ sinh ra các gói tin gửi đi và kiểm tra. Trong chương trình “test” bao gồm các task nhỏ hơn như

* gen(): nhiệm vụ của task này là sinh ra các gói tin ngẫu nhiên.
* reset(): nhiệm vụ của task này là đặt lại toàn bộ thiết kế về giá trị ban đầu.
* send(): nhiệm vụ chính của task này là gửi các gói tin. Trong task này bao gồm các task nhỏ hơn như:
  + send\_addrs(): nhiệm vụ để gửi các dữ liệu xác định địa chỉ cổng ra của dữ liệu.
  + send\_pad (): nhiệm vụ là gửi các padding bit.
  + send\_payload(): gửi dữ liệu đến cổng ra đã được xác định tại phần send\_addrs();
* recv(): nhiệm vụ của task này là nhận dữ liệu tại ngõ ra sau đó so sánh với các dữ liệu đã gửi để kiểm tra tính đúng đắn của thiết kế. Task này bao gồm 2 phần nhỏ hơn là:
  + get\_payload(): dùng để nhận các gói tin tại ngõ ra.
  + check(): kiểm tra giá trị tại ngõ ra có giống với giá trị đã gửi tại ngõ vào hay không thông qua hàm compare();

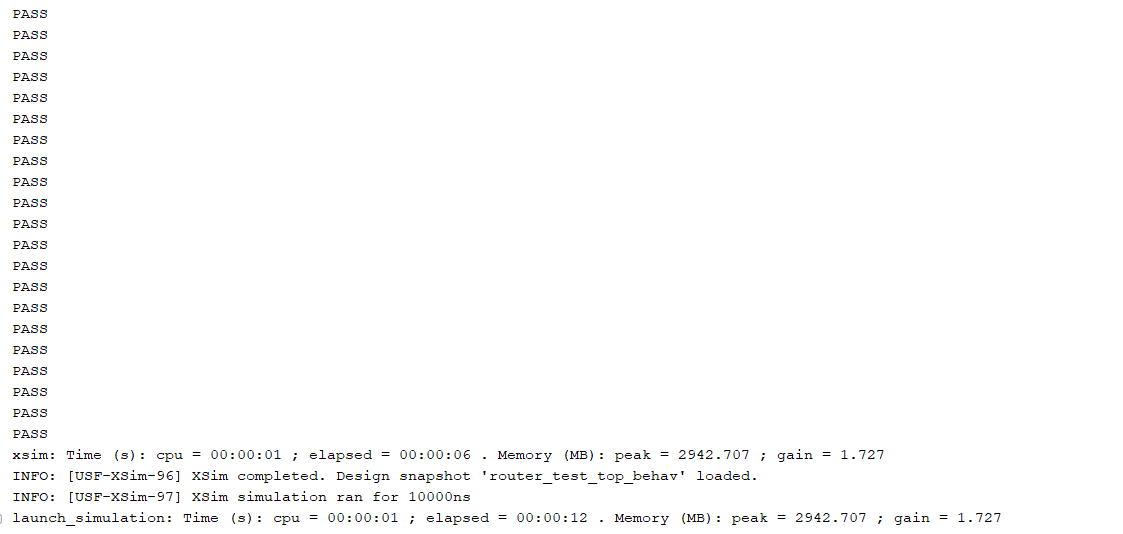
Chương trình test sẽ được thực thi 21 lần, nội dung chương trình test được trình bày như bảng sau:

Bảng 7 - Chương trình "test"

|  |
| --- |
| program automatic test (router\_io rtr\_io);  bit[3:0] sa;  bit[3:0] da;  logic[7:0] payload[$];  logic[7:0] pkt2cmp\_payload[$];  logic[7:0] temp;  initial begin  repeat (21) begin  gen();  reset();  fork  send();  recv();  join  disable fork;  end  // send\_addrs();  // send\_pad();  // send\_payload();  //$monitor(rtr\_io.dout);  #1000 $stop;  end  task reset();  rtr\_io.TB.reset\_n=0;  rtr\_io.frame\_n = 65535;  rtr\_io.valid\_n = 65535;  #15;  rtr\_io.TB.reset\_n=1;  rtr\_io.cb.din <=16'b0;  #150;  rtr\_io.frame\_n[sa] = 0;  endtask  task gen();  sa=3;  da=7;  payload.delete();  repeat ($urandom\_range(2,4))  payload.push\_back($urandom);  endtask  task send ();  send\_addrs();  send\_pad ();  send\_payload();  endtask  task send\_addrs();  rtr\_io.din[sa] =da[3];  #5 rtr\_io.din[sa] =da[2];  #5 rtr\_io.din[sa] =da[1];  #5 rtr\_io.din[sa] =da[0];    endtask  task send\_pad ();  //rtr.io.cb.valid\_n <=1'b1;  #30 rtr\_io.din[sa]= 1'b1;  rtr\_io.valid\_n[sa] = 0;  endtask  task send\_payload();  for (integer i =0; i<payload.size(); i=i+1) begin  rtr\_io.din[sa] = payload[i][7];  #10 rtr\_io.din[sa] = payload[i][6];  #10 rtr\_io.din[sa] = payload[i][5];  #10 rtr\_io.din[sa] = payload[i][4];  #10 rtr\_io.din[sa] = payload[i][3];  #10 rtr\_io.din[sa] = payload[i][2];  #10 rtr\_io.din[sa] = payload[i][1];  #10 rtr\_io.din[sa] = payload[i][0];  #10;  end  endtask  function bit compare(logic[7:0] a[$], logic[7:0] b[$]);  // $display(a);  // $display(b);  for (integer i =0; i<a.size(); i=i+1) begin  if (a[i]==b[i])  compare = 1'b1;  else begin  compare = 1'b0;  break;  end  end  endfunction  task recv ();  get\_payload();  check();  endtask  task check();  if (compare (payload, pkt2cmp\_payload )) $display ("PASS");  else $display ("FAIL");  endtask  task get\_payload();  logic enable;  pkt2cmp\_payload.delete();  @(negedge rtr\_io.frameo\_n[da])  begin  for (integer i =0; i<payload.size(); i=i+1) begin  #10 temp[7]= rtr\_io.dout[da];  #10 temp[6]= rtr\_io.dout[da];  #10 temp[5]= rtr\_io.dout[da];  #10 temp[4]= rtr\_io.dout[da];  #10 temp[3]= rtr\_io.dout[da];  #10 temp[2]= rtr\_io.dout[da];  #10 temp[1]= rtr\_io.dout[da];  #10 temp[0]= rtr\_io.dout[da];  pkt2cmp\_payload.push\_back(temp);  end  end  // $display(payload);  // $display(pkt2cmp\_payload);  endtask  endprogram |

## **Kết quả kiểm tra thiết kế**

Kết quả kiểm tra chức năng của mạch được xuất ra tại cửa sổ TCL Console:



Hình 9 - Kết quả tại TCL Console

Kết quả mô tả dạng sóng, theo đó các sóng theo thứ tự từ trên xuống lần lượt là: SystemClock, frame\_n[3], valid\_[3], din[3], dout[7] , frameo\_n[7], valido\_n[7], busy[7].

A green and black grid with blue lines

Description automatically generated

Hình 10 - Kết quả mô tả dạng sóng bằng Vivado

# **Kết luận**

* Qua quá trình làm bài thực hành, sinh viên đã nắm bắt được nội dung chức năng của bộ định tuyến 16 bit theo yêu cầu.
* Thực hiện được bộ định tuyến với chức năng theo yêu cầu.
* Đã kiểm tra thiết kế bằng môi trường SystemVerilog.

***\*Lưu ý: các đoạn mã Verilog mục đích chính là để minh họa, có thể không chính xác khi dùng vào chương trình thực tế. Các đoạn mã chính xác nằm bên trong thư mục đính kèm.***