

ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN – ĐIỆN TỬ
BỘ MÔN ĐIỆN TỬ



LUẬN VĂN TỐT NGHIỆP ĐẠI HỌC

**Thiết Kế Kiến Trúc Vi Mạch Mạng Dữ Liệu Trên Chip
(Network on Chip – NoC) Trên Nền Công Nghệ 130nm**

GVHD: ThS. Phạm Đăng Lâm

SVTH: Nguyễn Anh Hào

MSSV: 41000821

TP. HỒ CHÍ MINH, THÁNG 12 NĂM 2015

-----☆-----

-----☆-----

Số: _____ /BKĐT

Khoa: **Điện – Điện tử**

Bộ Môn: **Điện Tử**

NHIỆM VỤ LUẬN VĂN TỐT NGHIỆP

1. HỌ VÀ TÊN: Nguyễn Anh Hào

MSSV: 41000821

2. NGÀNH: **ĐIỆN TỬ - VIỄN THÔNG**

LỚP : DD10DV2

3. Đề tài: Thiết Kế Kiến Trúc Vi Mạch Mạng Dữ Liệu Trên Chip (Network on Chip -NoC)
Trên Nền Công Nghệ 130nm

4. Nhiệm vụ (Yêu cầu về nội dung và số liệu ban đầu):

.....
.....
.....
.....
.....
.....

5. Ngày giao nhiệm vụ luận văn:

6. Ngày hoàn thành nhiệm vụ:

7. Họ và tên người hướng dẫn:

Phản hướng dẫn

.....
.....

Nội dung và yêu cầu LVTN đã được thông qua Bộ Môn.

Tp.HCM, ngày..... tháng..... năm 20

CHỦ NHIỆM BỘ MÔN

NGƯỜI HƯỚNG DẪN CHÍNH

PHẦN DÀNH CHO KHOA, BỘ MÔN:

Người duyệt (chấm sơ bộ):.....

Đơn vị:.....

Ngày bảo vệ :

Điểm tổng kết:

Nơi lưu trữ luận văn:

LỜI CẢM ƠN

Lời đầu tiên em xin cảm ơn Ban lãnh đạo trường ĐH Bách Khoa cùng các thầy cô trong khoa Điện – Điện tử đã tạo điều kiện tốt nhất cho em có thể hoàn thành Luận văn tốt nghiệp. Xin cảm ơn các thầy cô ĐH Bách Khoa nói chung và khoa Điện – Điện tử nói riêng, những người đã mang lại cho em nền tảng kiến thức, khả năng tư duy, những bài học quý báu trong suốt hơn 4 năm qua làm hành trang cho em khi bước vào đời.

Xin gửi lời cảm ơn chân thành đến thầy Phạm Đăng Lâm đã tận tình hướng dẫn và giúp đỡ em trong suốt quá trình làm Luận văn tốt nghiệp. Thông qua quá trình hoàn thành Luận văn tốt nghiệp, Thầy đã bổ sung nhiều kiến thức mới và hoàn thiện thêm nhưng kiến thức đã học được. Thầy cũng đã giúp em nắm bắt được mối liên hệ mật thiết giữa thực tế và lý thuyết để em có thể hoàn thành Luận văn một cách đạt yêu cầu nhất.

Luận văn tốt nghiệp xem như một môn học cuối cùng của sinh viên. Quá trình thực hiện luận văn này đã giúp em tổng hợp tất cả kiến thức đã học ở trường trong suốt hơn 4 năm qua, đồng thời tìm hiểu thêm nhiều kiến thức mới. Đây là thời gian quý giá để em làm quen với công tác thiết kế vi mạch, nắm bắt các kiến thức công nghệ chủ yếu để sử dụng trong tương lai.

Em cũng xin dành những lời cảm ơn đến ba mẹ, những người luôn ủng hộ và tạo điều kiện tốt nhất để em có được ngày hôm nay.

Luận văn này là một công trình nhỏ đầu tay của mỗi sinh viên tự mình phải hoàn thiện trước khi ra trường. Khi đó đòi hỏi người sinh viên phải nỗ lực không ngừng để học hỏi. Em hoàn thành luận văn này trước hết nhờ sự chỉ bảo kỹ càng, tận tình của các thầy cô và sự giúp đỡ nhiệt tình của các bạn.

Mặc dù rất cố gắng trong quá trình thực hiện luận văn nhưng vì kinh nghiệm và quỹ thời gian hạn chế nên không tránh khỏi sai sót. Em kính mong được sự chỉ dẫn thêm rất nhiều từ quý thầy cô.

Xin chân thành cảm ơn!

Tp. Hồ Chí Minh, ngày tháng năm .

Sinh viên

TÓM TẮT LUẬN VĂN

Mô hình mạng trên chip (**Network on Chip – NoC**) được xem là sự thay thế mô hình chia sẻ bus trong việc truyền dẫn dữ liệu trên chip. Trong mô hình NoC, dữ liệu truyền dẫn giữa các phần cứng không thông qua các bus dữ liệu và các cầu nối (bus bridge) thông thường mà thông qua các router, do đó việc thiết kế các router giữ vai trò quan trọng trong việc đánh giá hiệu quả của mô hình NoC. Thông qua kiến trúc router, các giải thuật định tuyến, điều khiển luồng, công suất, độ trễ của một mô hình NoC được thể hiện. Cũng như các giao thức bus trước đây, để dữ liệu được truyền dẫn cần trì hoãn một số chu kỳ xung clock nhất định cho việc thiết lập đường truyền giữa các phần cứng. Với các router trong mô hình NoC, điều này cũng diễn ra tương tự. Để tài tiếp cận mô hình NoC cơ bản với khả năng định tuyến thích nghi. Để đạt được điều này, một kiến trúc router mới được giới thiệu nhằm đánh giá tính năng được cải thiện. Kiến trúc router mới không chỉ tối ưu về số chu kỳ clock mà dữ liệu phải trải qua khi truyền dẫn giữa hai router với nhau mà còn có khả năng phát hiện lỗi và tự động định tuyến nhằm đảm bảo chức năng của mạng dữ liệu. Sự xuất hiện của mạng trên chip (NoC) là sự phát triển của việc truyền thông tin cho hệ thống trên chip (SoC) dựa trên giao diện chuẩn cho việc tích hợp của IP với các yêu cầu truyền thông đa dạng. Các giao diện này phải đơn giản và đáp ứng nhanh. Luận văn trình bày thiết kế giao diện mạng dựa trên giao thức AHB được tích hợp với kiến trúc hiện tại để sử dụng với lõi ARM (ARM core).

Luận văn gồm 5 chương:

Chương 1 mô tả tổng quan về mạng trên chip, các hướng nghiên cứu và đặc tả giao thức của AMBA bus (AHB, APB).

Chương 2 mô tả kiến trúc mạng NoC và Network Interface (NI).

Chương 3 tiến hành mô phỏng kiến trúc router và mô hình NoC 4x4.

Chương 4 mô tả kết quả của thiết kế ở mức vật lý.

Chương 5 tổng kết và đánh giá.

MỤC LỤC

CHƯƠNG 1 TỔNG QUAN	1
1.1 Mạng trên chip	1
1.2 Các hướng nghiên cứu và các khái niệm liên quan đến NoC.....	1
1.2.1 Mô hình kiến trúc mạng NoC (topology).....	1
1.2.2 Thủ tục định tuyến (routing protocol)	7
1.2.3 Kỹ thuật chuyển mạch (switching technology)	9
1.2.4 Kiến trúc router (router architecture)	14
1.2.5 Điều khiển luồng (flow control).....	14
1.2.6 Kênh ảo (virtual channel).....	16
1.2.7 Giải thuật thích nghi.....	17
1.3 Giao thức AMBA AHB.....	20
1.3.1 Giới thiệu.....	20
1.3.2 Mô tả các tín hiệu	23
1.3.3 Quá trình truyền	26
1.3.4 Kết nối bên trong bus	35
1.3.5 Đáp ứng truyền của slave	36
1.4 Giao thức AMBA APB	38
1.4.1 Giới thiệu.....	38
1.4.2 Mô tả các tín hiệu	39
1.4.3 Quá trình truyền	40
1.4.4 Các trạng thái hoạt động.....	44
CHƯƠNG 2 THIẾT KẾ KIẾN TRÚC MẠNG NOC.....	46
2.1 Chọn lựa mô hình mạng	46
2.2 Lựa chọn giải thuật định tuyến	47
2.3 Xây dựng cơ chế điều khiển luồng.....	48

2.4	Mô hình kiến trúc bên ngoài router	49
2.5	Mô hình kiến trúc bên trong router	50
2.5.1	Giao diện một liên kết truyền nhận giữa hai router.....	50
2.5.2	Giao thức REQ/ANS	51
2.5.3	Cấu trúc khối bên trong của router.....	53
2.5.4	Cấu trúc dữ liệu	55
2.5.5	Cấu trúc và giao thức của khối Fifo	56
2.5.6	Cấu trúc và giao thức của khối CtrlInFifoIn	57
2.5.7	Cấu trúc và giao thức của khối CtrlOutFifoIn.....	58
2.5.8	Cấu trúc và giao thức của khối DataPath	61
2.5.9	Cấu trúc và giao thức của khối ChannelCtrl	62
2.5.10	Cấu trúc và giao thức của khối CtrlOutFifoOut	64
2.5.11	Đánh giá cấu trúc và giao thức truyền nhận của router	66
2.6	Khả năng và kiến trúc NoC tự sửa một lỗi sau khi sản xuất	66
2.6.1	Phương thức xác định lỗi trên mô hình NoC hai chiều	68
2.6.2	Phương thức tránh lỗi	68
2.7	Giao diện mạng (Network Interface - NI)	69
2.7.1	Mô hình kiến trúc bên ngoài NI	70
2.7.2	Mô hình kiến trúc bên trong NI.....	71
2.8	Tạo mạng dữ liệu tự động	77
CHƯƠNG 3 MÔ PHỎNG KIẾN TRÚC ROUTER & MÔ HÌNH NOC 4x4.....		79
3.1	Mô hình mô phỏng	79
3.2	Ngôn ngữ mô phỏng.....	81
3.3	Công cụ mô phỏng	82
3.4	Kiểm tra và đánh giá kết quả mô phỏng trên một router đơn.....	82
3.4.1	Router đơn nhận một luồng dữ liệu.....	83
3.4.2	Router đơn nhận hai luồng dữ liệu	87
3.4.3	Router đơn nhận ba luồng dữ liệu	91

3.4.4	Router đơn nhận bốn luồng dữ liệu	93
3.5	Kiểm tra và đánh giá kết quả mô phỏng trên mô hình NoC 4x4.....	96
3.6	Kiểm tra và đánh giá kết quả tránh lỗi trong khi truyền dữ liệu.....	100
3.7	Mô hình kiểm tra NoC 4x4 trên FPGA	102
CHƯƠNG 4 KẾT QUẢ THIẾT KẾ VẬT LÝ		103
4.1	Kết quả tổng hợp từ cấp độ RTL xuống lớp công	103
4.2	Kết quả đặt và đi dây cho thiết kế	110
CHƯƠNG 5 TỔNG KẾT		131
5.1	Quá trình triển khai luận văn	131
5.2	Kết quả đạt được	131
5.3	So sánh các khái quát	131
5.3.1	Tính cấp thiết trong khả năng sửa lỗi của mạng NoC	131
5.3.2	Mô hình mạng NoC.....	132
5.4	Ý nghĩa khoa học đề tài	132
5.5	Đề nghị hướng phát triển đề tài	133

DANH SÁCH HÌNH VẼ

<i>Hình 1.1 Các mô hình kết nối trong NoC [2]</i>	2
<i>Hình 1.2 Phân nhánh mô hình NoC theo cấu trúc mạng [2]</i>	3
<i>Hình 1.3 Mô hình mạng dạng MESH và TORUS [2]</i>	3
<i>Hình 1.4 Các mô hình dạng lưới hai chiều [3]</i>	4
<i>Hình 1.5 Phân loại các giải thuật định tuyến [1]</i>	8
<i>Hình 1.6 Phân loại kỹ thuật chuyển mạch [1]</i>	10
<i>Hình 1.7 Cấu trúc đơn giản của dữ liệu dạng gói [2]</i>	11
<i>Hình 1.8 Mô tả cơ chế chuyển mạch Store & Forward [2]</i>	12
<i>Hình 1.9 Mô tả cơ chế chuyển mạch Wormhole [2]</i>	13
<i>Hình 1.10 Phân loại cơ chế điều khiển luồng [1]</i>	15
<i>Hình 1.11 Thí dụ hoạt động của hai kênh ảo A và B [2]</i>	17
<i>Hình 1.12 Các trường hợp lỗi</i>	18
<i>Hình 1.13 Các trường hợp định tuyến với lỗi trên OX</i>	19
<i>Hình 1.14 Các trường hợp định tuyến với lỗi trên OY</i>	20
<i>Hình 1.15 Sơ đồ khối AHB-Lite [25]</i>	21
<i>Hình 1.16 Giao diện master [25]</i>	21
<i>Hình 1.17 Giao diện slave [25]</i>	22
<i>Hình 1.18 Quá trình đọc [25]</i>	26
<i>Hình 1.19 Quá trình ghi [25]</i>	27
<i>Hình 1.20 Quá trình đọc với trạng thái chờ [25]</i>	27
<i>Hình 1.21 Quá trình ghi với trạng thái chờ [25]</i>	28
<i>Hình 1.22 Nhiều quá trình truyền [25]</i>	28
<i>Hình 1.23 Ví dụ về kiểu truyền [25]</i>	29
<i>Hình 1.24 4-beat wrapping burst [25]</i>	32
<i>Hình 1.25 4-beat incrementing burst [25]</i>	32

<i>Hình 1.26 8-beat wrapping burst [25]</i>	33
<i>Hình 1.27 8-beat incrementing burst [25]</i>	33
<i>Hình 1.28 Truyền theo khói với độ dài không xác định [25]</i>	34
<i>Hình 1.29 Các tín hiệu lựa chọn slave [25]</i>	35
<i>Hình 1.30 Bộ phân kênh tín hiệu và các kết nối [25]</i>	36
<i>Hình 1.31 Đáp ứng ERROR [25]</i>	38
<i>Hình 1.32 Quá trình ghi cơ bản không có trạng thái chờ [26]</i>	40
<i>Hình 1.33 Quá trình ghi có trạng thái chờ [26]</i>	41
<i>Hình 1.34 Quá trình đọc cơ bản không có trạng thái chờ [26]</i>	42
<i>Hình 1.35 Quá trình đọc có trạng thái chờ [26]</i>	42
<i>Hình 1.36 Ví dụ quá trình ghi không thành công [26]</i>	43
<i>Hình 1.37 Ví dụ quá trình đọc không thành công [26]</i>	44
<i>Hình 1.38 Sơ đồ trạng thái [26]</i>	44
<i>Hình 2.1 Mô hình NoC 4x4</i>	46
<i>Hình 2.2 Gói dữ liệu truyền theo giải thuật định tuyến XY</i>	48
<i>Hình 2.3 Kiến trúc bên ngoài của router đơn</i>	49
<i>Hình 2.4 Giao diện liên kết giữa hai router</i>	50
<i>Hình 2.5 Kiến trúc bên ngoài của router đơn</i>	50
<i>Hình 2.6 Giao thức req/ans</i>	52
<i>Hình 2.7 Mô hình khối bên trong router</i>	53
<i>Hình 2.8 Kết nối chi tiết bên trong router</i>	54
<i>Hình 2.9 Cấu trúc khung dữ liệu</i>	56
<i>Hình 2.10 Giao diện khối Fifo</i>	56
<i>Hình 2.11 Giao diện khối CtrlInFifoIn</i>	57
<i>Hình 2.12 Giao thức khối CtrlInFifoIn</i>	58
<i>Hình 2.13 Giao diện khối CtrlOutFifoIn</i>	59
<i>Hình 2.14 Giao thức khối CtrlOutFifoIn</i>	59
<i>Hình 2.15 Giao diện khối CtrlInFifoIn và CtrlOutFifoIn</i>	60

<i>Hình 2.16 Đường truyền dữ liệu bên trong router</i>	61
<i>Hình 2.17 Giao diện khói DataPath và các khói ChannelCtrl.....</i>	62
<i>Hình 2.18 Giao diện khói ChannelCtrl</i>	62
<i>Hình 2.19 Giao thức khói ChannelCtrl</i>	64
<i>Hình 2.20 Giao diện khói CtrlOutFifoOut</i>	64
<i>Hình 2.21 Giao thức khói CtrlOutFifoOut</i>	65
<i>Hình 2.22 Các trường hợp tránh 1 lỗi</i>	67
<i>Hình 2.23 Phương thức xác định router lỗi</i>	68
<i>Hình 2.24 Cấu trúc khung dữ liệu cho việc tránh lỗi</i>	68
<i>Hình 2.25 Sơ đồ kết nối của NI trong NoC</i>	70
<i>Hình 2.26 Kiến trúc bên ngoài của NI</i>	71
<i>Hình 2.27 Mô hình khói bên trong của NI</i>	71
<i>Hình 2.28 Giao diện khói Buffer</i>	73
<i>Hình 2.29 Giao diện khói AHBInterface</i>	74
<i>Hình 2.30 Giao diện khói CtrlOut.....</i>	75
<i>Hình 2.31 Giao diện khói CtrlIn</i>	76
<i>Hình 2.32 Giao diện phần mềm tạo mạng dữ liệu tự động</i>	77
<i>Hình 3.1 Luồng thiết kế chip cơ bản</i>	79
<i>Hình 3.2 Mô hình kiểm tra thiết kế NoC 4x4</i>	81
<i>Hình 3.3 Các trường hợp kiểm tra router đơn nhận một luồng dữ liệu</i>	83
<i>Hình 3.4 Dữ liệu truyền qua 1 router đơn.....</i>	86
<i>Hình 3.5 Mô phỏng 5 đường truyền không tranh chấp nhau</i>	87
<i>Hình 3.6 Các trường hợp kiểm tra router đơn nhận hai luồng dữ liệu.....</i>	87
<i>Hình 3.7 Mô phỏng hai luồng dữ liệu ra cùng một hướng.....</i>	90
<i>Hình 3.8 Trường hợp kiểm tra router đơn nhận ba luồng dữ liệu</i>	91
<i>Hình 3.9 Giải đố xung 3 gói dữ liệu tranh chấp 1 đường truyền</i>	93
<i>Hình 3.10 Trường hợp kiểm tra router đơn nhận bốn luồng dữ liệu</i>	93
<i>Hình 3.11 Giải đố mô phỏng 4 luồng dữ liệu tranh chấp một đường truyền</i>	96

<i>Hình 3.12 Truyền dữ liệu từ router 0000 đến router 1111</i>	97
<i>Hình 3.13 Quan hệ giữa số gói dữ liệu – thời gian truyền nhận.....</i>	98
<i>Hình 3.14 Gửi từ West sang East nhưng chuyển hướng sang North</i>	100
<i>Hình 3.15 Gửi từ West sang North nhưng quay lại West.....</i>	101
<i>Hình 3.16 Gửi từ North sang South nhưng chuyển hướng sang West</i>	101
<i>Hình 3.17 Mô hình kiểm tra trên FPGA.....</i>	102
<i>Hình 4.1 Biểu đồ diện tích NoC 4x4 khi thay đổi tần số tổng hợp</i>	109
<i>Hình 4.2 Kết quả đặt và đi dây toàn bộ chip.....</i>	110
<i>Hình 4.3 Kết quả đặt và đi dây toàn bộ chip (chi tiết)</i>	111
<i>Hình 4.4 Kết quả đặt và đi dây tất cả lớp metal.....</i>	111
<i>Hình 4.5 Kết quả đặt và đi dây Core Area</i>	112
<i>Hình 4.6 Kết quả đặt và đi dây Core Area_Port</i>	112
<i>Hình 4.7 Kết quả đặt và đi dây Core Area_Port_Terminal</i>	113
<i>Hình 4.8 Kết quả đặt và đi dây Die Area</i>	113
<i>Hình 4.9 Kết quả đặt và đi dây Port hrdata</i>	114
<i>Hình 4.10 Kết quả đặt và đi dây Core_Port_Pin</i>	114
<i>Hình 4.11 Kết quả đặt và đi dây Core_Port_Cell</i>	115
<i>Hình 4.12 Kết quả đặt và đi dây Core_Port_Pin_Cell</i>	115
<i>Hình 4.13 Kết quả đặt và đi dây Core_Port_All power signals.....</i>	116
<i>Hình 4.14 Kết quả đặt và đi dây Core_Port_All metal</i>	116
<i>Hình 4.15 Kết quả đặt và đi dây Via tất cả Metal</i>	117
<i>Hình 4.16 Kết quả đặt và đi dây lớp Metal 1</i>	117
<i>Hình 4.17 Kết quả đặt và đi dây Metal 1_Via 1</i>	118
<i>Hình 4.18 Kết quả đặt và đi dây Metal 2_Via 2</i>	118
<i>Hình 4.19 Kết quả đặt và đi dây Metal 3_Via 3</i>	119
<i>Hình 4.20 Kết quả đặt và đi dây Metal 4_Via 4</i>	119
<i>Hình 4.21 Kết quả đặt và đi dây Metal 5_Via 5</i>	120
<i>Hình 4.22 Kết quả đặt và đi dây Metal 6_Via 6</i>	120

<i>Hình 4.23 Kết quả đặt và đi dây từ Metal 6 đến Metal 9</i>	121
<i>Hình 4.24 Kết quả đặt và đi dây Metal 2 và Metal 3</i>	121
<i>Hình 4.25 Kết quả đặt và đi dây Clock</i>	122
<i>Hình 4.26 Kết quả đặt và đi dây VDD và VSS</i>	122
<i>Hình 4.27 Kết quả đặt và đi dây Power_Ground_Clock</i>	123
<i>Hình 4.28 Kết quả đặt và đi dây Power và Cell</i>	123
<i>Hình 4.29 Kết quả đặt và đi dây Cell và Clock tree</i>	124
<i>Hình 4.30 Kết quả đặt và đi dây Cell Combination</i>	124
<i>Hình 4.31 Kết quả đặt và đi dây Clock net</i>	125
<i>Hình 4.32 Kết quả đặt và đi dây VDD power ring</i>	126
<i>Hình 4.33 Kết quả đặt và đi dây VSS power ring</i>	126
<i>Hình 4.34 Kết quả đặt và đi dây VDD rail</i>	127
<i>Hình 4.35 Kết quả đặt và đi dây VSS rail</i>	127
<i>Hình 4.36 Kết quả đặt và đi dây khoảng cách giữa Die và Core</i>	128
<i>Hình 4.37 Kết quả đặt và đi dây chiều cao của Cell</i>	128
<i>Hình 4.38 Kết quả đặt và đi dây chiều rộng của Power</i>	129
<i>Hình 4.39 Kết quả đặt và đi dây Cell Density</i>	129
<i>Hình 4.40 Kết quả đặt và đi dây Pin Density</i>	130

DANH SÁCH BẢNG BIỂU

<i>Bảng 1-1 Bảng so sánh các đặc tả của các mô hình lưới khác nhau [3]</i>	7
<i>Bảng 1-2 Các tín hiệu toàn cục [25]</i>	23
<i>Bảng 1-3 Các tín hiệu của master [25]</i>	24
<i>Bảng 1-4 Các tín hiệu của slave [25]</i>	25
<i>Bảng 1-5 Các tín hiệu của bộ giải mã [25]</i>	25
<i>Bảng 1-6 Các tín hiệu của bộ phân kênh tín hiệu [25]</i>	26
<i>Bảng 1-7 Các kiểu truyền [25]</i>	29
<i>Bảng 1-8 Các kích thước truyền [25]</i>	30
<i>Bảng 1-9 Các kiểu truyền theo khối [25]</i>	31
<i>Bảng 1-10 Tín hiệu HRESP [25]</i>	36
<i>Bảng 1-11 Đáp ứng truyền [25]</i>	37
<i>Bảng 1-12 Các tín hiệu của giao thức APB [26]</i>	39
<i>Bảng 3-1 Các trường hợp kiểm tra router đơn nhận một luồng dữ liệu</i>	84
<i>Bảng 3-2 Quá trình truyền một gói dữ liệu từ West → South</i>	85
<i>Bảng 3-3 Quá trình truyền một gói dữ liệu từ West → North</i>	85
<i>Bảng 3-4 Quá trình truyền một gói dữ liệu từ West → IP</i>	86
<i>Bảng 3-5 Quá trình truyền một gói dữ liệu từ West, IP → East</i>	88
<i>Bảng 3-6 Quá trình truyền một gói dữ liệu từ West, IP → North</i>	89
<i>Bảng 3-7 Quá trình truyền một gói dữ liệu từ West, East → IP, North</i>	90
<i>Bảng 3-8 Quá trình truyền một gói dữ liệu từ West, East, IP → North</i>	92
<i>Bảng 3-9 Quá trình truyền một gói dữ liệu từ West, East, IP, South → North</i>	95
<i>Bảng 3-10 Tổng hợp các trường hợp mô phỏng trên một router</i>	96
<i>Bảng 3-11 Kết quả truyền dữ liệu từ router 0000 đến router 1111</i>	98
<i>Bảng 3-12 Công thức tính băng thông cho bus 32 bit dữ liệu của mô hình NoC 4x4</i>	99
<i>Bảng 4-1 Nền tảng tổng hợp khối NoC 4x4 xuống lớp cổng</i>	103

Bảng Biểu

<i>Bảng 4-2 Bảng đặc tả cây thư mục môi trường tổng hợp</i>	105
<i>Bảng 4-3 Bảng khảo sát tương tác giữa tần số và diện tích</i>	109

CHƯƠNG 1 TỔNG QUAN

1.1 Mạng trên chip

Với kiến trúc bus thông thường trước đây, các nghiên cứu đã cho thấy sự giới hạn về hiệu quả cũng như tính phức tạp trong thiết kế của nó [1] khi mà mật độ tích hợp các đơn vị xử lý tăng lên. Một hướng nghiên cứu khác được xem là rất có tiềm năng trong tương lai để giải quyết vấn đề đặt ra là “mạng trên chip – Network on Chip (**NoC**)”. Trong một kiến trúc NoC, các lõi xử lý hay các IP giao tiếp không thông qua các bus thông dụng mà thông qua các router. Các hướng nghiên cứu về vấn đề này được giới thiệu ở mục 1.2 tiếp sau đây.

1.2 Các hướng nghiên cứu và các khái niệm liên quan đến NoC

Dựa trên ý tưởng giao tiếp giữa các IP thông qua các router được kết nối với nhau. Các hướng nghiên cứu sau được đưa ra và gặt hái nhiều thành quả.

- Kiến trúc NoC (mô hình, giải thuật định tuyến và phân xử...).
- Các phương pháp kết nối chuyển mạch bên trong router và giữa các router.
- Những vấn đề về năng lượng.
- Những vấn đề về đảm bảo chất lượng (lỗi và khả năng sửa lỗi, băng thông giới hạn ...).
- Các phương pháp thiết kế (trình tự các bước thiết kế), đánh giá (mô phỏng, kiểm tra), phần mềm hỗ trợ... cho thiết kế NoC.
- Hệ điều hành và khả năng lập trình trên NoC.
- Vấn đề về tích hợp các kỹ thuật bên trong NoC (mạng noron, logic mờ...).
- Vấn đề về đánh giá hiệu suất NoC trên FPGA và ASIC.
- Vấn đề về điện tích.

Do NoC là một thuật ngữ mang tính khái quát nên các lĩnh vực nghiên cứu về NoC nêu trên mang ý nghĩa bao hàm nhiều mảng kiến thức rộng lớn khác nhau. Bởi thế, Luận văn chỉ đề cập sâu vào những hướng nghiên cứu mà đề tài sẽ tiếp cận. Đầu tiên những khái niệm cơ bản nhất của một kiến trúc NoC được mô tả.

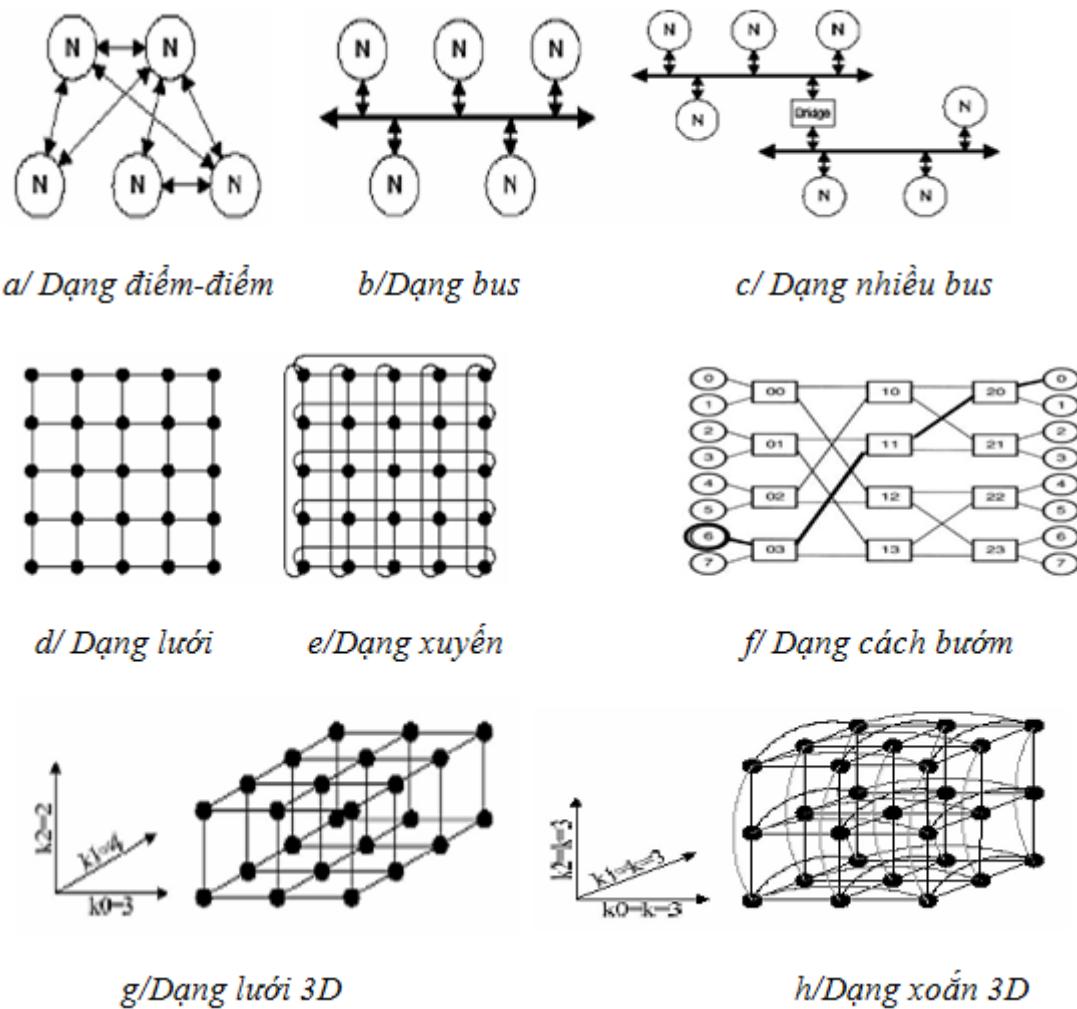
1.2.1 Mô hình kiến trúc mạng NoC (topology)

Xuất phát từ các quan điểm khác nhau, các nghiên cứu đã đưa ra các mô hình mạng thông dụng NoC khác nhau như dạng điểm đối (point-to-point), dạng bus, dạng xuyến (**Torus**),

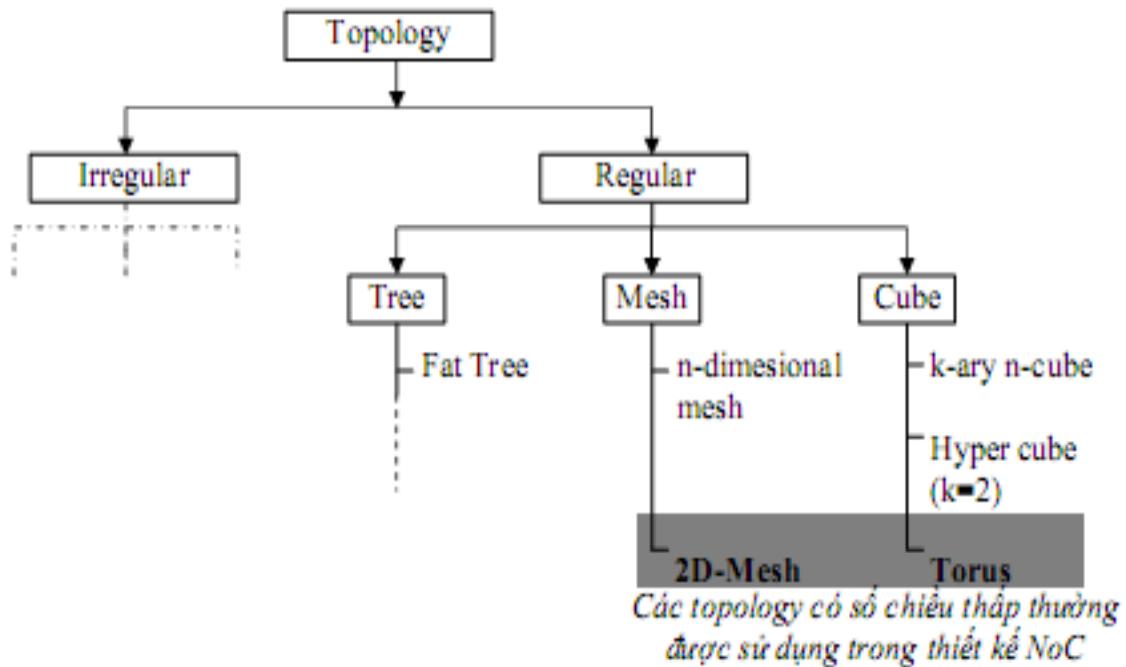
dạng lưới (**Mesh**), dạng vòng (**Ring**), dạng hình cánh bướm (**Butterfly**), dạng không theo quy luật (**Irregular**) ...

Cấu hình mạng là sự sắp xếp các router chuyên mạch và các kênh liên kết giữa các router đó. Lựa chọn mô hình mạng là công việc đầu tiên cần làm khi thiết kế một hệ thống mạng, cho dù là mạng máy tính hay mạng trên chip. Việc chọn lựa mô hình mạng có ảnh hưởng rất lớn đến các kỹ thuật định tuyến (**routing**) và điều khiển luồng (**flow control**) sẽ được nhắc đến ở các phần sau.

Mô hình mạng thường được mô tả dưới dạng sơ đồ hay đồ hình. *Hình 1.1 [2]* mô tả một số mô hình mạng cơ bản đã được phát triển và đưa vào ứng dụng trong thực tế. Hình 1.2 [2] mô tả các mô hình mạng NoC phổ biến. Trong đó mạng dạng lưới (**MESH**) và dạng xuyên (**TORUS**) được sử dụng phổ biến nhất.

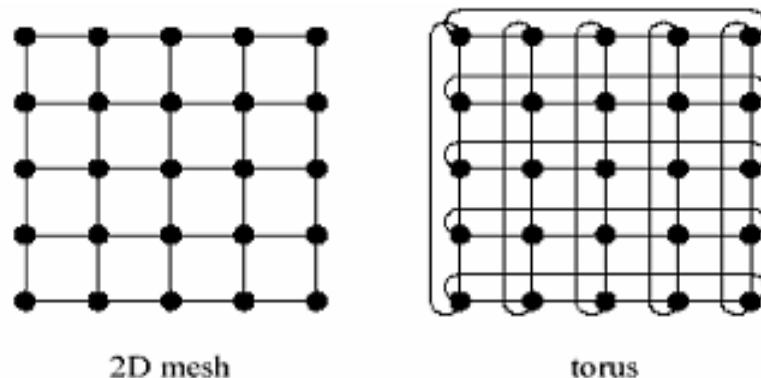


Hình 1.1 Các mô hình kết nối trong NoC [2]



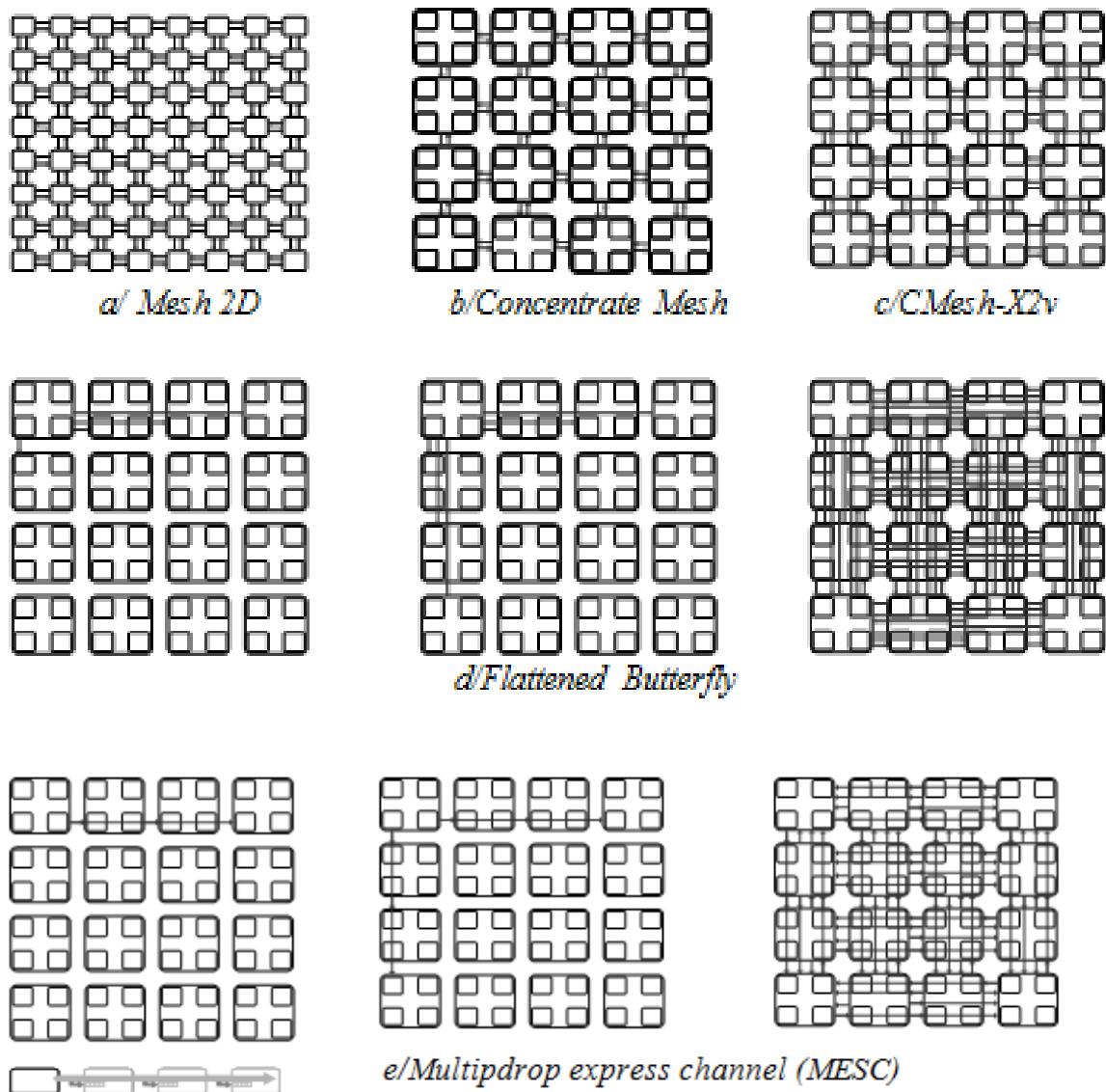
Hình 1.2 Phân nhánh mô hình NoC theo cấu trúc mạng [2]

Sau đây là những phân tích về mô hình mạng **MESH** và **TORUS** hai chiều. Sự khác biệt giữa hai mô hình mạng này được mô tả bởi Hình 1.3 [2].



Hình 1.3 Mô hình mạng dạng MESH và TORUS [2]

Dựa trên Hình 1.3, mô hình dạng lưới mang tính đối xứng cao rất phù hợp và tiện lợi trong quá trình sản xuất. Chính điều này đã làm cho mô hình dạng lưới trở thành mô hình được lựa chọn nhiều nhất trong các hướng nghiên cứu. Sau đây là những biến đổi của mô hình dạng lưới được phân tích chi tiết.

*Hình 1.4 Các mô hình dạng lưới hai chiều [3]*

Theo như Hình 1.4 [3], các ưu và khuyết điểm của mỗi mô hình được trình bày.

a/ Mesh 2D (Hình 1.4.a)

Thuận lợi:

- Cấu tạo các router đơn giản tốc độ cao.
- Thuận lợi trong quá trình sắp đặt cấu trúc lên chip (**place & route**).

Không thuận lợi:

- Chiếm diện tích lớn.
- Độ trễ cao.

- Tiêu hao nhiều năng lượng.

b/ Concentrate Mesh (Hình 1.4.b)

Là sự cải thiện của **Mesh-2D** bằng cách nhóm 4 router thành 1 router. Với sự kết hợp này đã cho thấy các đặc điểm cải thiện sau:

Thuận lợi:

- Chia sẻ các ngõ vào/ra dùng chung trong một nhóm 4 router.
- Giảm độ trễ hơn so với **Mesh-2D** do có thể rút ngắn số router trung gian phải đi qua.
- Tốc độ tăng khi giao tiếp giữa các router trong cùng một nhóm.

Không thuận lợi:

- Độ phức tạp trong thiết kế các kiến trúc bên trong router.
- Độ trễ cao.
- Tiêu hao nhiều năng lượng.

c/ CMesh-X2 (Hình 1.4.c)

Mô hình **CMesh-X2** được xem như một phiên bản cải thiện của **Concentrate Mesh** với sự tăng cường các port giao tiếp nhằm tăng thông lượng truyền dẫn và giảm độ phức tạp trong việc thiết kế kiến trúc bên trong của router.

d/ Flattened Butterfly (Hình 1.4.d)

Dựa trên nền tảng **Concentrate Mesh**, mỗi router có tất cả các đường bus truyền đi đến các router khác trên phương x và y.

Trong khi đó, bản thân mỗi router cũng nhận các đường bus từ tất cả các router khác trên phương x và y. Chính cải thiện này đã mang đến những thuận lợi và bất lợi sau.

Thuận lợi:

- Tiện lợi giao tiếp giữa các router trên các phương x và y mà không thông qua router trung gian.
- Giảm độ trễ hơn so với **Mesh-2D**, **Concentrate Mesh** do có thể rút ngắn tổng số router phải đi qua nhiều nhất là 2.

Không thuận lợi:

- Độ phức tạp trong thiết kế thành phần điều khiển bên trong router.
- Khi lưu lượng dữ liệu ít, tài nguyên trên mạng dư thừa lãng phí.
- Tiêu hao nhiều năng lượng và các vấn đề về số lượng kênh truyền trở nên khó khăn khi mạng mở rộng với nhiều IP.

e/ Multidrop express channel-MESC (Hình 1.4.e)

Dựa trên nền tảng **Flattened Butterfly**, sự cải thiện trong vấn đề kênh truyền được chú ý. Số lượng kênh truyền sẽ bằng chính số router trên một phương x hoặc y. Ngoài ra, đặc tính kênh truyền trong mô hình này còn cho phép một router truyền dữ liệu đồng thời cho nhiều router khác nhờ cơ chế **multidrop** [3]. Chính cải thiện này đã mang đến những thuận lợi và bất lợi sau.

Thuận lợi:

- Tiện lợi giao tiếp giữa các router trên các phương x và y mà không thông qua router trung gian đồng thời hỗ trợ cơ chế **multidrop** trên một đường truyền.
- Giảm độ trễ so với **Mesh-2D**, **Concentrate Mesh** do có thể rút ngắn số router trung gian phải đi qua nhiều nhất là 2.
- Thích hợp cho các mô hình thời gian thực.

Không thuận lợi:

- Độ phức tạp trong thiết kế thành phần điều khiển bên trong router.
- Khi lưu lượng dữ liệu ít, tài nguyên trên mạng dư thừa lãng phí.
- Tính không đối xứng trong thiết kế.

Để so sánh chi tiết về những khác biệt của các mô hình NoC dạng lưới và những biến đổi của nó, kết quả so sánh tài nguyên giữa các mô hình được giới thiệu trong Bảng 1-1 [3]. Dựa vào sự khác biệt giữa các thông số của Bảng 1-1, các điểm thuận lợi và bất lợi của các mô hình được chọn được thể hiện.

Ví dụ tiêu biểu như số router tối đa mà gói dữ liệu phải đi qua chỉ là 2 router so với mô hình **MESC** và **FBfly**. Nhưng đối với **CMesh**, số router sẽ tăng theo quy mô của mạng. Ngoài ra khi tăng tài nguyên trong mạng NoC, một số chỉ số về tài nguyên sẽ thay đổi, một số khác được giữ cố định tùy theo đặc tính cấu hình mạng. Việc phân tích các thông số khác không được đề cập chi tiết ở đây. Phần phụ lục tham khảo có đưa ra các nghiên cứu thể

hiện các đánh giá chi tiết các mô hình cần quan tâm [3], [5], [6]. Phần kế tiếp Luận văn giới thiệu chi tiết về các kỹ thuật được sử dụng trong mạng NoC như : kỹ thuật chuyển mạch, kỹ thuật định tuyến, kỹ thuật điều khiển luồng và cấu trúc tổng quan router... Trước hết, kỹ thuật định tuyến được giới thiệu trước.

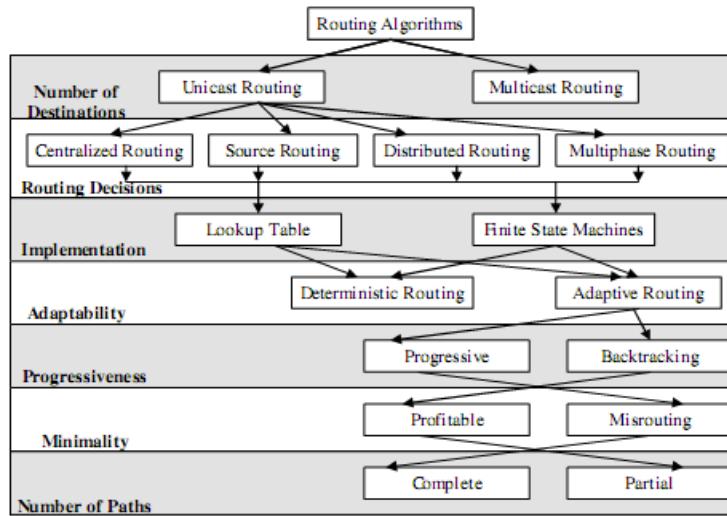
Bảng 1-1 Bảng so sánh các đặc tả của các mô hình lưới khác nhau [3]

	CMesh		FBfly		MESC	
Tổng số router	64	256	64	256	64	256
Số router/ phương	4	8	4	8	4	8
Số router tối đa gói dữ liệu đi qua	6	14	2	2	2	2
Số kênh truyền/phương	2	2	8	32	4	8
Độ rộng kênh	576	1152	144	72	288	288
Số ngõ vào router	4	4	6	14	6	14
Số ngõ ra router	4	4	6	14	4	4

1.2.2 Thủ tục định tuyến (routing protocol)

Dựa trên mô hình mạng tương ứng sẽ có những thủ tục định tuyến phù hợp. Ở đây Đề tài chỉ đưa ra các định tuyến thông dụng. Việc sử dụng giải thuật định tuyến tương ứng với mô hình mạng cụ thể không được đề xuất. Bạn đọc cần tham khảo các đề tài chi tiết để biết thêm. Bản thân các giải thuật định tuyến cũng thay đổi theo từng mô hình mạng khác nhau. Mô hình ở Hình 1.5 [1] mô tả cách phân loại các giải thuật định tuyến đã và đang nghiên cứu.

Dựa trên các mô hình mạng và mục đích khác nhau mà đưa ra các giải thuật định tuyến thích hợp. Đầu tiên căn cứ vào số lượng đích đến (**destinations**) mà các giải thuật được phân chia thành đơn mục tiêu (**unicast routing**) và đa mục tiêu (**multicast routing**). Trong đó, giải thuật định tuyến đơn mục tiêu được ứng dụng rộng rãi hơn nhờ tính phù hợp với đặc tính kết hợp điểm điểm trong mô hình nhiều IP. Trong mô hình đơn mục tiêu, người ta chia nhỏ thành bốn loại khác bao gồm: Định tuyến nguồn (**source routing**), định tuyến phân tán (**distributed routing**), định tuyến trung tâm (**centralized routing**) và định tuyến dựa trên từng pha khác nhau (**multiphase routing**).



Hình 1.5 Phân loại các giải thuật định tuyến [1]

Trong định tuyến trung tâm, một bộ phận điều khiển trung tâm sẽ thực hiện nhiệm vụ này để điều khiển dữ liệu trong hệ thống. Trong định tuyến nguồn, quyết định giải thuật định tuyến được thực hiện tại nơi (**router**) mà dữ liệu bắt đầu được truyền đi. Trong khi đó định tuyến phân tán quyết định đường truyền (các đường đi của các gói/khung-**packet/flit** dữ liệu) dựa trên các router trung gian. Sự kết hợp của định tuyến nguồn và định tuyến phân tán chính là định tuyến theo từng pha khác nhau.

Các mô hình định tuyến đơn mục tiêu nói trên đều dựa trên hai nguyên lý là bảng định tuyến hoạt động hoặc máy trạng thái. Dựa trên hai đặc tính này, các router có thể biết được trạng thái của chúng và định tuyến các luồng dữ liệu theo các hướng đi hợp lý.

Tiếp tục dựa trên hai đặc tính của việc định tuyến là bảng định tuyến và máy trạng thái, việc phân loại định tuyến được phân chia theo hai hướng khác là định tuyến thích nghi và không thích nghi. Ở định tuyến không thích nghi, mọi hoạt động định tuyến mà dựa trên bảng định tuyến hay máy trạng thái đều theo một trình tự cố định được thiết lập trước đó. Ngược lại với định tuyến thích nghi, các quá trình này có thể thay đổi theo một nguyên lý nào đó nhằm phù hợp với sự thay đổi trạng thái của mạng lúc bấy giờ (mạng đang rỗi hay đang nghẽn).

Các giải thuật định tuyến thích nghi nhằm làm giảm bớt các sự cố có thể xảy ra khi hệ thống hoạt động ở công suất cao và lúc này thường xảy ra các sự cố nghẽn mạch. Trong định tuyến thích nghi người ta lại chia ra nhiều giải thuật nhỏ khác nhằm thích nghi với trạng thái mạng hiện hành như: Backtracking, Progressive, Profitable, Misrouting, Complete, Partial... Các vấn đề chi tiết về các giải thuật không được đề cập trong Đề tài này. Tuy nhiên, trong

hầu hết các mô hình mạng dạng lưới, giải thuật định tuyến theo phương XY được ứng dụng phổ biến nhất. Về chi tiết giải thuật sẽ được trình bày ở các phần sau. Giữa các giải thuật định tuyến luôn tồn tại những vấn đề cần giải quyết khác nhau [4]. Do đó việc đưa ra giải thuật là một trong những bước quan trọng sau khi đã chọn lựa mô hình mạng nhằm quyết định hiệu quả mong muốn của hệ thống.

Ở đây, một số vấn đề liên quan đến giải thuật định tuyến mà yêu cầu khi router hoạt động đảm bảo không mắc phải.

a/ Deadlock:

Là trường hợp gói tin chờ đợi một sự kiện nào đó nhưng sự kiện đó sẽ không bao giờ xảy ra. Sự kiện đó có thể là sự giải phóng tài nguyên của router nhằm phục vụ cho một gói tin đang chờ. Trường hợp này gây ra việc mất gói tin.

b/ Livelock:

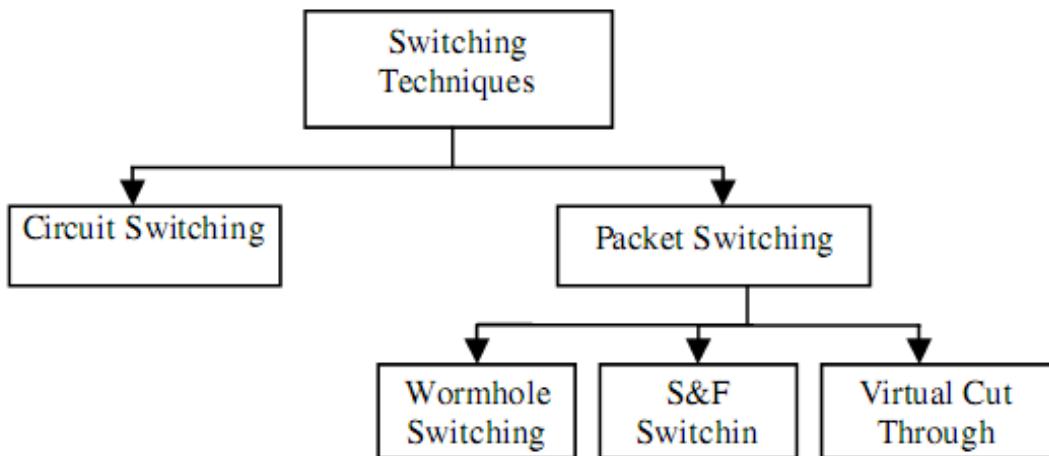
Là trường hợp một gói tin không bao giờ đến được đích. Hầu hết các NoC mắc phải lỗi này là do các thủ tục định tuyến thích nghi được sử dụng. Tuy nhiên chính sự không tối ưu của thủ tục định tuyến này làm gói tin được truyền đi lòng vòng nhưng không đến được đích cần thiết.

c/ Indefinite Postponement (trì hoãn không xác định):

Là trường hợp mà gói tin chờ đợi một sự kiện về mặt lý thuyết là có thể xảy ra nhưng trên thực tế không bao giờ diễn ra. Sự kiện này cũng như deadlock là sự giải phóng tài nguyên của một router. Sự thật, tài nguyên này đã được giải phóng, tuy nhiên sau đó lại được phục vụ cho một gói tin khác. Gói tin đang chờ có độ ưu tiên thấp sẽ không bao giờ được phục vụ. Vấn đề ưu tiên trước sau là nguyên nhân gây nên lỗi này.

1.2.3 Kỹ thuật chuyển mạch (switching technology)

Kỹ thuật chuyển mạch được đặc tả dựa trên đặc trưng của gói dữ liệu. Dựa trên đặc điểm này, kỹ thuật chuyển mạch được phân chia theo Hình 1.6 [1].



Hình 1.6 Phân loại kỹ thuật chuyển mạch [1]

Đầu tiên căn cứ vào gói dữ liệu truyền và mạng truyền dẫn mà phân chia thành hai loại chính là chuyển mạch mạch (**circuit switching**) và chuyển mạch gói (**packet switching**).

Chuyển mạch mạch được thực hiện như một mạch vật lý theo một hướng đi cụ thể từ nguồn đến đích đã được định sẵn trước đó. Trong khi đó đối với chuyển mạch gói, dữ liệu được truyền đi đến các router kế cận mà không biết trước các hướng đi chắc chắn sau đó.

Trong chuyển mạch gói, người ta dựa trên ba kỹ thuật chính để phân loại chúng là: Wormhole Switching, Store&Forward Switching và Virtual Cut Through. Để phân tích các giải thuật này đầu tiên cấu trúc các gói dữ liệu được giới thiệu.

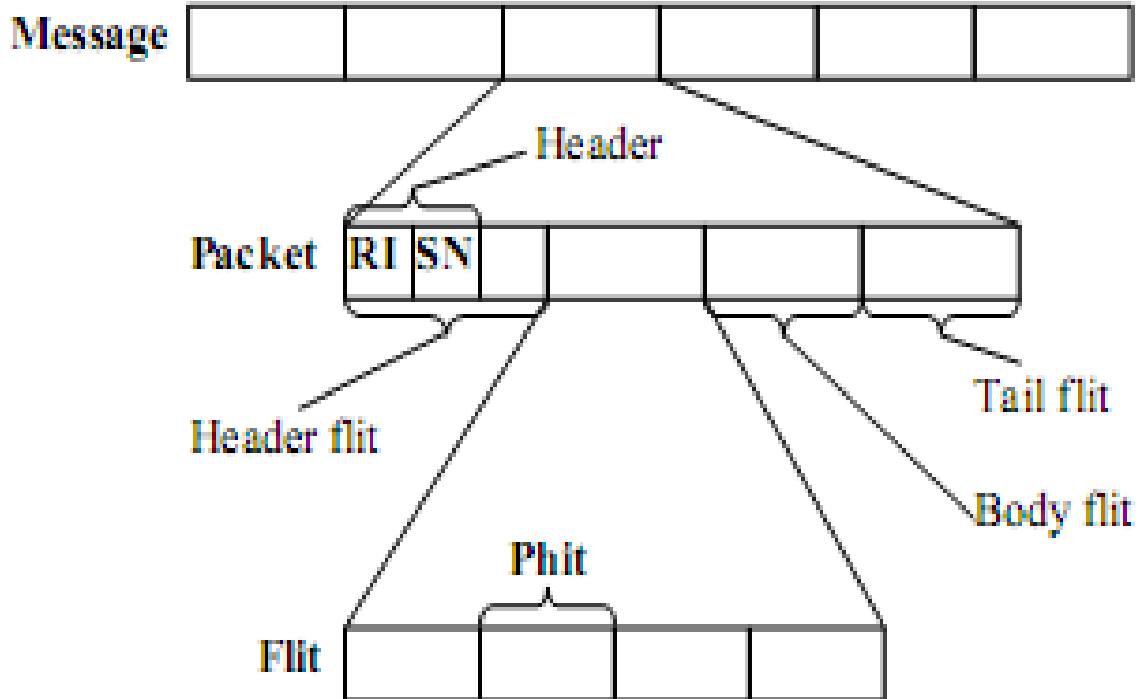
a/ Cấu trúc dữ liệu trong chuyển mạch gói

Trong chuyển mạch gói, dữ liệu cần truyền được phân chia thành từng gói nhỏ với kích thước được quy định trước (packet).

Trong mỗi gói nhỏ lại được chia thành các đơn vị dữ liệu nhỏ hơn gọi là khung (**flit**). Khung đầu tiên của gói gọi là **header flit**. Khung kết thúc gói dữ liệu gọi là **tail flit**.

Các khung giữa gọi là **body flit**. **Header flit** và **tail flit** chủ yếu mang thông tin thông báo mở đầu và kết thúc của một gói. Các **body flit** mang thông tin chính cần truyền.

Việc đóng gói dữ liệu theo một khung cho trước được thực hiện bởi Network Interface (NI) giữa các IP và router. Ta có thể thấy được cấu trúc đơn giản của một gói dữ liệu dựa trên Hình 1.7 [2].



Hình 1.7 Cấu trúc đơn giản của dữ liệu dạng gói [2]

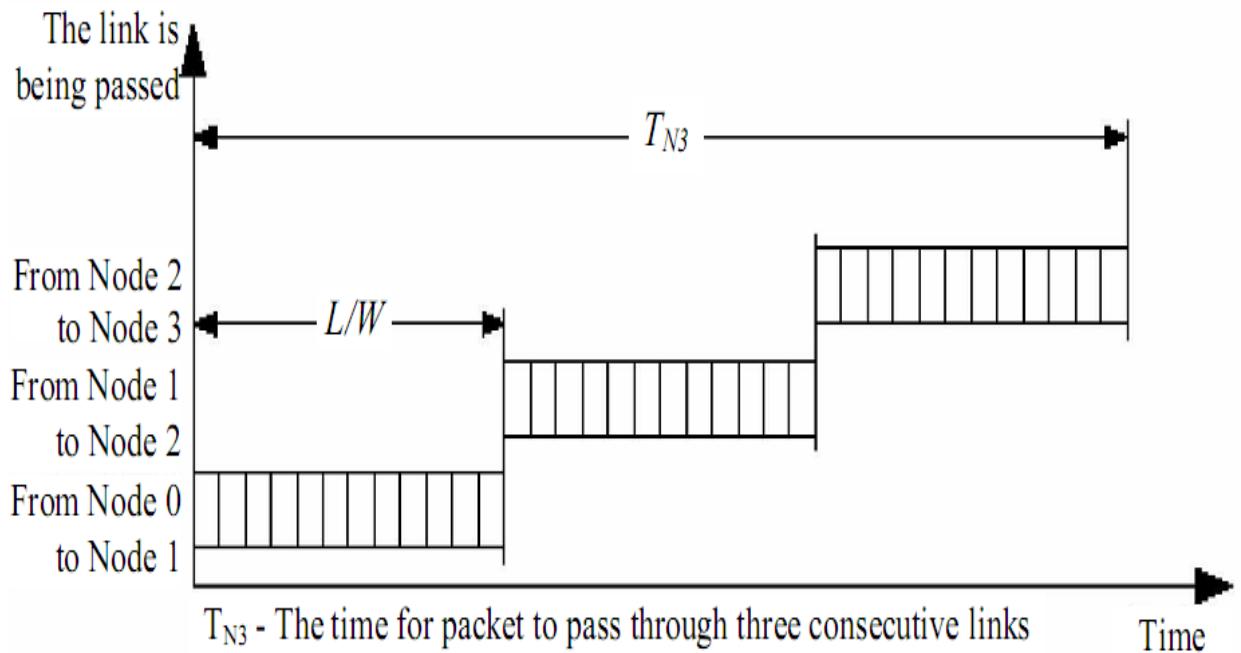
b/ Store & Forward Switching

Trong cơ chế Store&Forward một gói dữ liệu được thu nhận toàn bộ tại mỗi router trung gian trước khi được chuyển tới router kế tiếp theo như Hình 1.8 [2].

Nếu **L** là độ dài của gói dữ liệu, **W** là độ rộng của kênh và **T_c** là chu kỳ gói dữ liệu đi qua kênh truyền thì thời gian truyền một message qua **n** kênh liên tiếp theo công thức [2]:

$$T = \left(n \cdot \frac{L}{W} \right) T_c \quad (1.1)$$

Rõ ràng độ trễ khi sử dụng phương thức này tỷ lệ với chiều dài của gói dữ liệu. Đây chính là nhược điểm lớn của phương thức này.



Hình 1.8 Mô tả cơ chế chuyển mạch Store & Forward [2]

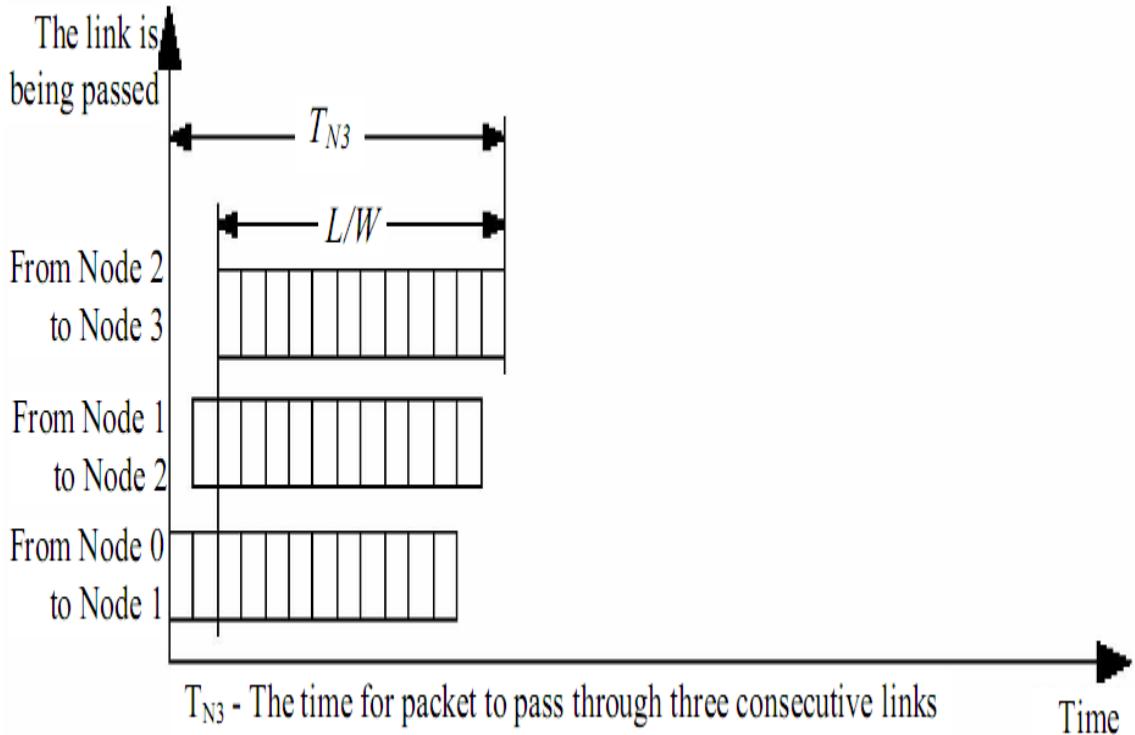
c/ Virtual cut through

Với kỹ thuật Virtual cut through, các khung có thể được chia thành các khung nhỏ hơn (**phit**). Tuy nhiên, cũng tương tự như Store & Forward, toàn bộ gói dữ liệu được truyền đến bộ đệm của một router.

Do đó, dung lượng bộ đệm không được cải thiện. Điều khác biệt duy nhất của phương pháp này với Store & Forward là phương pháp này cho phép lưu trữ các gói dữ liệu ở router trung gian khi router kế tiếp đang bận. Phương pháp này không có nhiều các nghiên cứu về nó vì sự cải thiện không đáng kể và việc thực thi phức tạp hơn Store & Forward.

d/ Wormhole Routing

Với Wormhole Routing, thay vì truyền toàn bộ các khung cùng lúc đến các router trung gian thì đầu tiên chỉ một khung đầu được truyền. Chính khung này mở ra một đường dẫn giữa hai router, các khung còn lại đi tiếp trên đường dẫn ấy (Hình 1.9 [2]).



Hình 1.9 Mô tả cơ chế chuyển mạch Wormhole [2]

Chính cơ chế này làm cho các khung đầu luôn đi trước mà không chờ các khung sau đó. Điều này làm giảm bộ đệm cho các router trung gian vì không bao giờ tồn tại tất cả các flit trên một router trung gian nếu mạng không bị nghẽn. Khung cuối cùng có nhiệm vụ giải phóng đường dẫn.

Sử dụng các ký hiệu tương tự Store & Forward, ta có thể tính được độ trễ của phương pháp này như sau [2]:

$$T = \left((n+1) + \frac{L}{W} \right) T_c \quad (1.2)$$

Rõ ràng kích thước một khung nhỏ hơn một gói nên cơ chế Wormhole cải thiện đáng kể băng thông, độ trễ và kích thước bộ đệm. Nhược điểm chính của phương pháp này là khi khung mở đầu mở ra một đường dẫn cho các khung sau thì tài nguyên trên đường dẫn sẽ bị chiếm dụng cho đến khi khung cuối cùng giải phóng nó. Ngoài ra khi hiện tượng nghẽn mạng xảy ra. Điều này có nghĩa là khung mở đường bị ngưng lại tại một router trung gian nào đó, nếu thời gian trì hoãn đủ lâu thì toàn bộ gói tin sẽ được lưu trữ trên một router giống như các phương thức trước. Có rất nhiều các nghiên cứu so sánh phương pháp này với mô hình

chuyển mạch mạch để chỉ rõ các thuận lợi và bất lợi của chúng nhằm đưa ra các mô hình phù hợp nhất.

Giải pháp để khắc phục hai nhược điểm trên là sử dụng kỹ thuật kênh ảo (**virtual channel**) sẽ được đề cập ở phần sau.

1.2.4 Kiến trúc router (router architecture)

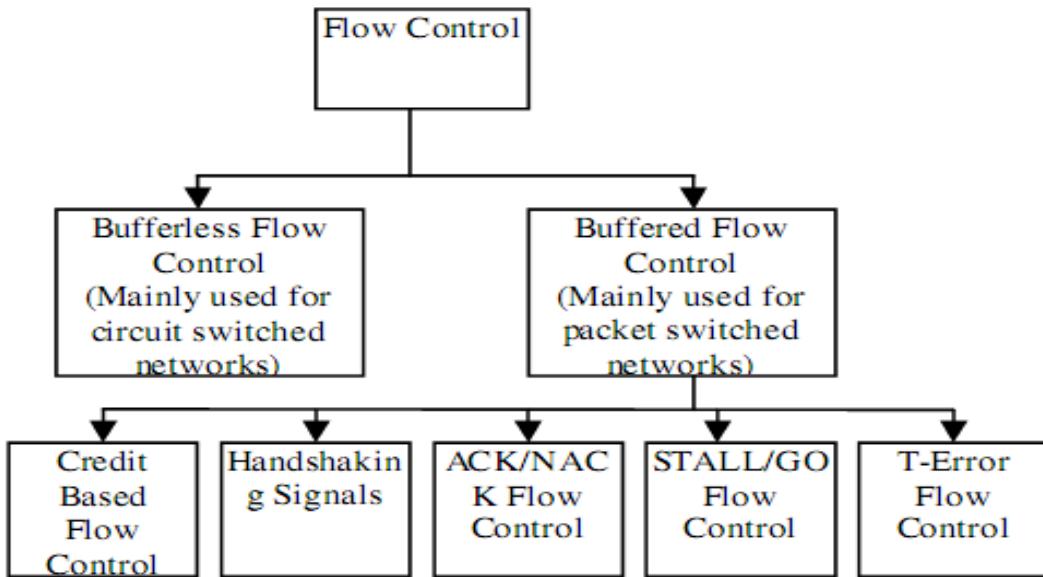
Kiến trúc router phần lớn dựa trên kỹ thuật chuyển mạch nêu trên [1]. Theo như hai phương pháp chính chuyển mạch mạch và chuyển mạch gói, cấu trúc router cũng được phân chia làm hai hướng chính là tăng cường bộ đệm (buffer) hoặc giảm bớt bộ đệm. Trong chuyển mạch mạch, các gói dữ liệu được truyền đi mà không phải xếp hàng đợi ở các router trung gian do đường truyền bị chiếm giữ. Do đó số lượng bộ đệm cũng giảm đáng kể. Trong chuyển mạch gói, việc phân chia các gói dữ liệu thành từng khung đặt ra yêu cầu tăng cường các bộ đệm trên mỗi router nhằm tránh tình trạng tắc nghẽn hay mật độ truyền dẫn tăng cao.

Tuy nhiên, cũng tùy vào các mục đích khác nhau của hệ thống mà các kiến trúc bên trong router cũng trở nên khác nhau. Số lượng bộ đệm dùng cho các ngõ vào, ngõ ra, hay bộ nhớ tạm của router cũng sẽ không giống nhau theo nhiều ứng dụng. Bạn đọc có thể tham khảo sơ lược về hai cấu trúc router cho chuyển mạch mạch và chuyển mạch gói qua [16] và [17].

1.2.5 Điều khiển luồng (flow control)

Điều khiển luồng quyết định tài nguyên của router cần dùng như dung lượng bộ đệm, kích cỡ router, băng thông và sự hao tổn năng lượng mà router phải sử dụng. Cơ chế điều khiển luồng là cơ chế giao tiếp giữa các router với nhau.

Cũng dựa trên việc sử dụng bộ đệm nhiều hay ít mà việc phân chia cơ chế điều khiển luồng thành hai nhóm riêng biệt theo Hình 1.10 [1].



Hình 1.10 Phân loại cơ chế điều khiển luồng [1]

Dựa trên việc sử dụng hay không sử dụng các bộ đệm mà phân chia thành điều khiển luồng có đệm (**buffered flow control**) và điều khiển luồng không đệm (**bufferless flow control**). Do đặc tính yêu cầu hệ thống hoạt động với băng thông và khối lượng dữ liệu cao, việc tồn tại các bộ đệm trở nên cần thiết trong mạng NoC. Do đó, các hướng nghiên cứu phát triển nhiều cho vấn đề điều khiển luồng dựa trên các bộ đệm. Dựa vào đó, cơ chế điều khiển luồng có đệm được phân chia thành nhiều cơ chế nhỏ cụ thể khác như: Credit Based, HandShaking, ACK/NAK, STALL/GO, T-ERROR.

a/ Credit Based Flow Control

Trên một luồng truyền dẫn, các router truyền (**upstream**) đếm số gói dữ liệu cần truyền và sau đó giải phóng các ô nhớ khi số gói dữ liệu đã truyền qua hết.

Cơ chế này được thực hiện dựa trên việc cập nhật số gói dữ liệu được quy định sẵn [8], [9].

b/ HandShaking Flow Control

Router truyền sẽ gửi một khung đầu cho router nhận. Nếu router nhận đã nhận khung này và chấp nhận gói dữ liệu tiếp theo đó khi đã có sẵn vùng đệm, nó sẽ gửi một tín hiệu thông báo cho router truyền. Dựa vào đó quá trình truyền dẫn được thiết lập. Việc kết thúc một gói dữ liệu cũng được thể hiện tương tự bằng cách gửi đi một tín hiệu xác nhận [10].

c/ ACK/NAK Flow Control

Thủ tục này tương tự như HandShaking, khung đầu sẽ được giữ trong bộ đệm router truyền cho đến khi tín hiệu ACK từ router nhận được truyền về router truyền. Khi đó, khung này sẽ được xóa khỏi bộ đệm trên router truyền. Nếu như một tín hiệu NAK được nhận, khung này vẫn được lưu trữ trên bộ đệm và truyền lại lần thứ hai nhằm mong muốn thiết lập đường truyền. Theo đó, việc chờ đợi tín hiệu ACK từ router nhận được tiếp tục [11], [12].

d/ STALL/GO Flow Control

Trong cơ chế này, giữa các router cần thêm hai dây dẫn khác. Khi một router có bộ đệm trống, nó sẽ truyền tín hiệu “GO” đi. Nếu như bộ đệm đã được sử dụng, tín hiệu “STALL” sẽ được truyền. Hiện tại chưa tìm thấy một NoC nào ứng dụng cơ chế này.

e/ T-ERROR Flow Control

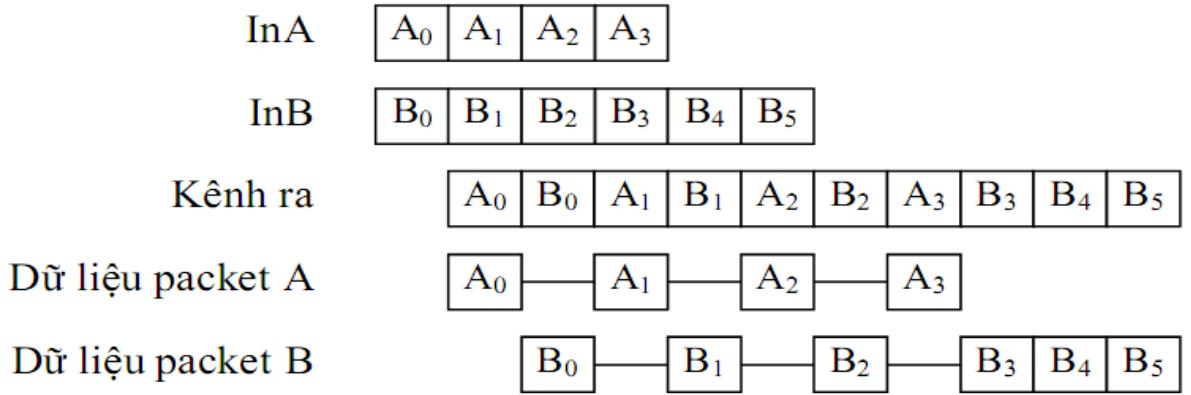
Tương tự STALL/GO, cơ chế này chưa được tìm thấy trong ứng dụng NoC chính bởi tính phức tạp của nó. Do đó đề tài không đề cập chi tiết.

1.2.6 Kênh ảo (virtual channel)

Cơ chế điều khiển luồng dùng kênh ảo (**virtual channel flow control**) thực hiện kết hợp một vài kênh ảo trên cùng một kênh vật lý bằng cách tận dụng khoảng thời gian rỗi giữa các gói dữ liệu. Điều này cho phép nhiều gói dữ liệu từ các bộ đệm khác nhau chia sẻ cùng một tài nguyên vật lý giúp nâng cao hiệu suất sử dụng kênh truyền.

Tương tự như cơ chế Wormhole Routing, trong cơ chế điều khiển luồng có sử dụng kênh ảo, khung đầu sẽ yêu cầu cấp phát một kênh ảo trong số các kênh ảo đang rỗi, sau đó các khung còn lại tiếp tục được truyền đi trên kênh ảo đã được cấp phát này.

Điểm khác biệt so với cơ chế Wormhole Routing là các khung trên kênh ảo cần cạnh tranh với các kênh ảo khác để có thể được cấp phát đường truyền vật lý và tiếp tục truyền đi. Do đó khi sử dụng thêm cơ chế kênh ảo trong điều khiển luồng, các router cần thêm các bộ điều khiển. Chính điều này tạo nên sự hạn chế trong vấn đề sử dụng kênh ảo. Có thể hình dung quá trình vận hành hai kênh ảo trên một kênh vật lý như Hình 1.11 [2].



Hình 1.11 Thí dụ hoạt động của hai kênh ảo A và B [2]

Hình 1.11 [2] minh họa trường hợp hai gói A và B thuộc hai kênh ảo cùng chia sẻ một kênh vật lý. Dựa vào chính sách phân phối tài nguyên mà các khung của gói A (A₀, A₁ ...) và các khung của gói B (B₀, B₁ ...) truyền đi tuần tự theo các thời điểm khác nhau.

Thông thường cơ chế Wormhole Routing và Virtual Channel được kết hợp để giải quyết các vấn đề về độ trễ, deadlock, livelock ... Chúng được tìm thấy nhiều trong các nghiên cứu về cấu tạo router [1, 13, 14].

1.2.7 Giải thuật thích nghi

Giải thuật thích nghi trong quá trình định tuyến đã đạt được những thành công nhất định [20, 21, 22] trong việc giải quyết những vấn đề liên quan.

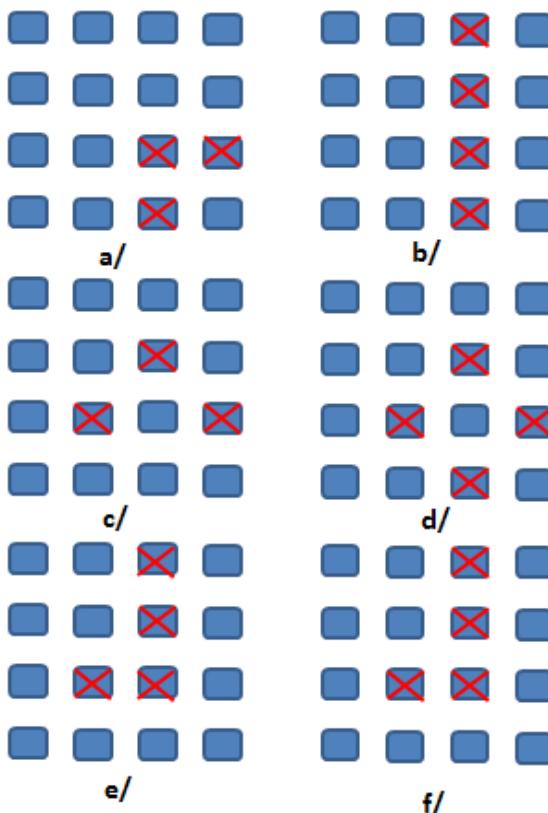
- Tăng chỉ số sản phẩm có khả năng sử dụng khi lỗi do sản xuất.
- Đảm bảo mô hình truyền dữ liệu được tiếp tục không bị nghẽn mạch.

Để tài tiếp cận giải thuật thích nghi nhằm giải quyết vấn đề thứ nhất là tăng khả năng sử dụng mạng khi lỗi sản xuất xảy ra. Một trong những lý do tiếp cận vấn đề này là vì kiến trúc chi tiết bên trong của mạng NoC được đề xuất luôn giải quyết được vấn đề tắc nghẽn mạng vì quy luật ưu tiên luồng dữ liệu được thiết kế với cơ chế mặt nạ được giới thiệu ở chương 2 cho phép việc chờ đợi phục vụ sẽ được đáp ứng sau một khoảng thời gian nhất định. Phương pháp này xuất phát từ ý tưởng như việc xử lý các ngắt trong hệ thống điều khiển. Tất cả các nguồn ngắt có độ ưu tiên khác nhau và được xử lý tuần tự cho đến hết và quay lại với nguồn ngắt có độ ưu tiên cao nhất. Các luồng dữ liệu cũng được xử lý một cách tương tự nhằm tránh việc tắc nghẽn luồng dữ liệu.

Tiếp cận với vấn đề tăng khả năng sử dụng khi lỗi do sản xuất, một số vấn đề phát sinh được thể hiện:

- Làm sao để biết được router nào không hoạt động.
- Số nút mạng không hoạt động tối đa là bao nhiêu thì có thể tái sử dụng sản phẩm.
- Các trường hợp lỗi làm cho một nút mạng không thể sử dụng được.

Giải quyết vấn đề kiểm tra router không hoạt động tương đối được kiểm soát một cách dễ dàng khi thiết lập một đường truyền dữ liệu với dữ liệu đã được kiểm soát, việc không nhận được hay nhận sai các dữ liệu ở các thiết bị đầu cuối cho phép xác định sự chính xác của nút mạng cần kiểm tra. Vấn đề là làm sao giúp cho các nút mạng kể cận biết được điều này nhằm tránh việc truyền dữ liệu qua những nút mạng này. Một trong những vấn đề này sinh khác là giả sử số nút mạng lỗi quá nhiều thì có thể sử dụng mạng dữ liệu nữa không hay chúng ta chỉ sử dụng được một phần nào đó của mô hình mà không phải sử dụng hoàn toàn hết mạng dữ liệu. Một số trường hợp được mô tả bởi Hình 1.12 cho thấy những tình trạng có thể không mong muốn với số lỗi nút mạng lớn hay rơi vào những tình huống không thể khắc phục ở cục bộ một khu vực nào đó.

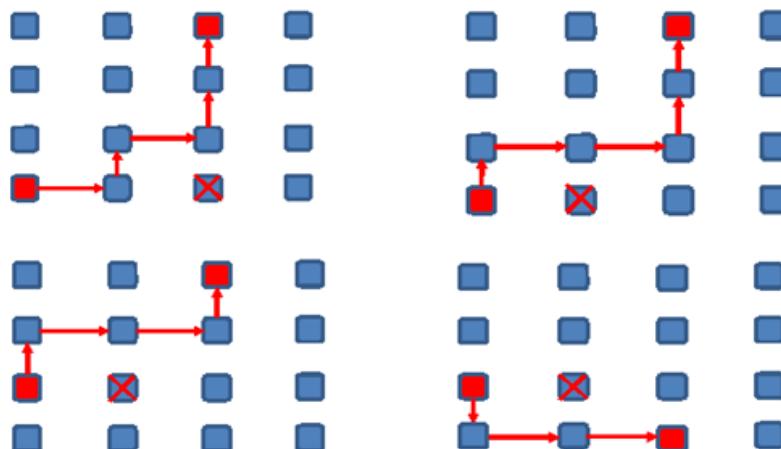


Hình 1.12 Các trường hợp lỗi

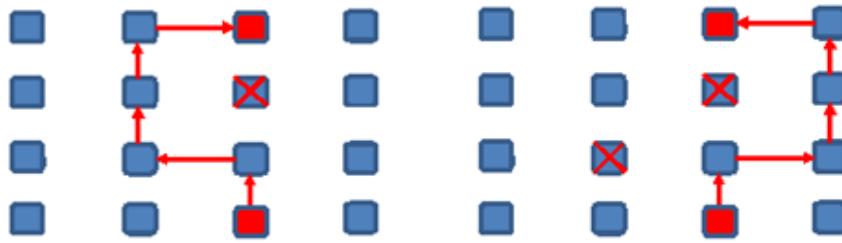
Trong quá trình truyền dữ liệu ở các trường hợp kể trên, các nút mạng không bị ván đề đều có thể giao tiếp với nhau (a/, e/, f/, c/) nhưng rất khó để định tuyến một cách tự động. Trường hợp thứ hai (d/, b/) cho thấy khả năng không thể giao tiếp với tất cả các nút mạng. Một số nút mạng chỉ có thể giao tiếp cục bộ với nhau vì đường truyền đã bị cắt đứt hẳn do ván đề sản xuất.

Mỗi nút mạng trong đề tài tiếp cận định tuyến thích nghi chọn lựa giải pháp lập trình sẵn. Chi tiết được thực hiện như sau:

- Các nút mạng lỗi được phát hiện đã trình bày ở trên.
- Dựa trên các nút mạng lỗi này, các thanh ghi nhận biết lỗi trong từng nút mạng được cập nhật.
- Mỗi nút mạng có thể truyền đi 4 hướng đông/tây/nam/bắc khác nhau một cách tổng quát. Dựa trên các giá trị của thanh ghi lỗi này, việc truyền dữ liệu theo hướng nào được quyết định.
- Một ví dụ minh họa như sau, nếu hướng đông bị lỗi thì dựa trên giá trị thanh ghi này cho phép các đường truyền dữ liệu theo hướng đông sẽ được truyền theo hướng khác.
- Việc cập nhật/lập trình cho các thanh ghi thông qua việc truyền gói dữ liệu được gọi là gói dữ liệu lập trình ngay sau khi kiểm tra để biết được toàn bộ hệ thống với các nút mạng lỗi.



Hình 1.13 Các trường hợp định tuyến với lỗi trên OX



Hình 1.14 Các trường hợp định tuyến với lõi trên OY

1.3 Giao thức AMBA AHB

1.3.1 Giới thiệu

a/ Về giao thức

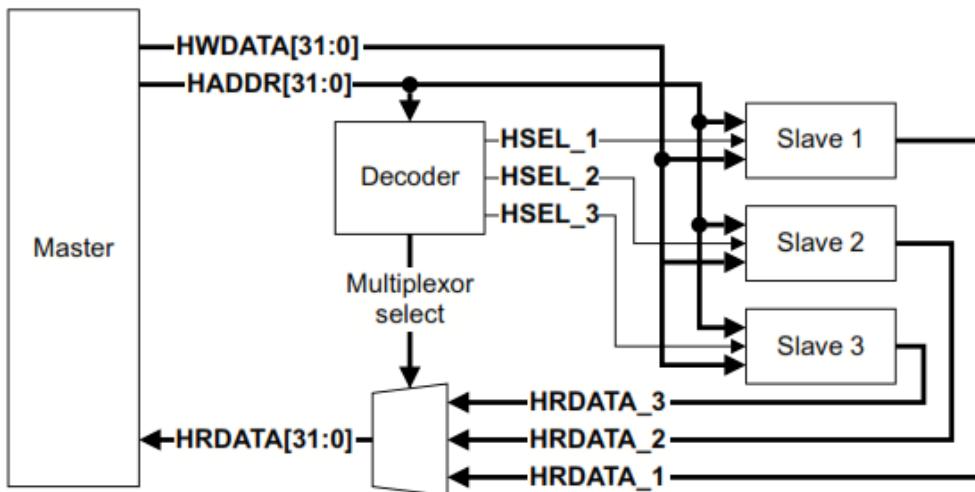
AMBA AHB-Lite giải quyết các yêu cầu thiết kế hiệu năng cao. Nó là giao diện bus hỗ trợ đơn bus master và cung cấp hoạt động băng thông cao.

AHB-Lite thực hiện các tính năng cần thiết cho hệ thống hiệu năng cao, tần số clock cao bao gồm:

- Truyền theo khối
- Hoạt động theo đơn cạnh clock
- Không dùng chuyển mức ba trạng thái
- Cấu hình bus dữ liệu rộng, 64, 128, 256, 512, và 1024 bit.

AHB-Lite slave phổ biến nhất là các thiết bị bộ nhớ trong, các giao diện bộ nhớ ngoài, thiết bị ngoại vi băng thông cao. Mặc dù thiết bị ngoại vi băng thông thấp có thể bao gồm AHB-Lite slave nhưng vì hiệu năng hệ thống nên thường dùng với AMBA Advanced Peripheral Bus (APB).

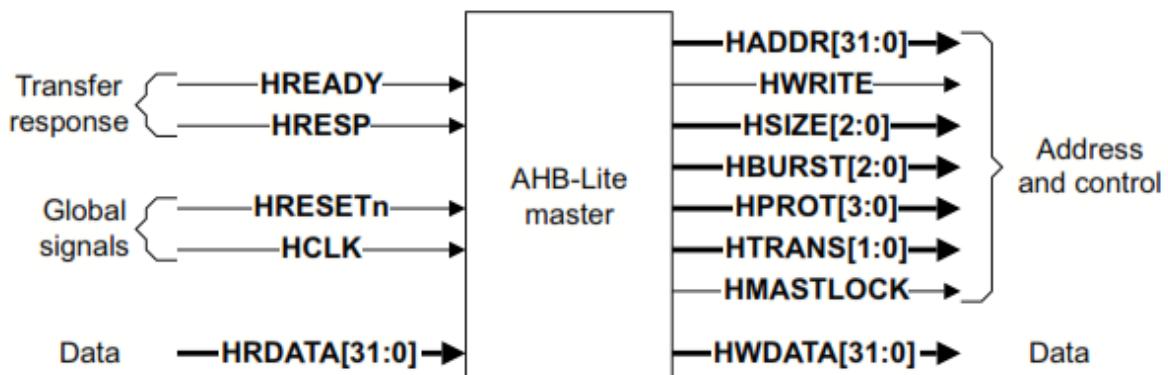
Hình 1.15 biểu diễn một hệ thống AHB-Lite đơn master được thiết kế với một AHB-Lite master và ba AHB-Lite slave. Logic kết nối bên trong bus bao gồm một bộ giải mã địa chỉ và một bộ phân kênh tín hiệu từ slave đến master. Bộ giải mã sẽ lấy địa chỉ từ master để chọn slave thích hợp và bộ phân kênh tín hiệu sẽ chọn dữ liệu ngõ ra từ slave tương ứng đến master.



Hình 1.15 Sơ đồ khái AHB-Lite [25]

➤ Master

Một AHB-Lite master cung cấp địa chỉ và thông tin điều khiển để bắt đầu hoạt động đọc và ghi. Hình 1.16 biểu diễn một giao diện AHB-Lite master.



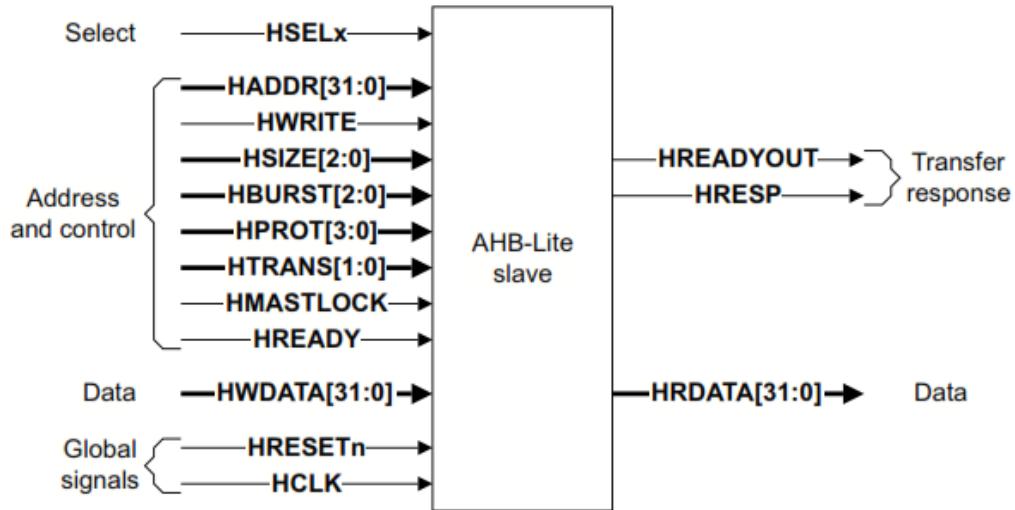
Hình 1.16 Giao diện master [25]

➤ Slave

Một AHB-Lite slave sẽ đáp ứng quá trình truyền từ master trong hệ thống. Slave sử dụng tín hiệu lựa chọn **HSELx** từ bộ giải mã để điều khiển khi nó đáp ứng một quá trình truyền. Slave sẽ báo lại cho master biết:

- Quá trình truyền thành công.
- Quá trình truyền thất bại.
- Hoặc đang chờ truyền dữ liệu.

Hình 1.17 biếu diễn một giao diện AHB-Lite slave.



Hình 1.17 Giao diện slave [25]

➤ **Bộ giải mã (Decoder)**

Thành phần này giải mã địa chỉ của mỗi quá trình truyền và cung cấp một tín hiệu lựa chọn cho slave tham gia vào quá trình truyền. Nó cũng cung cấp một tín hiệu điều khiển đến bộ phân kênh tín hiệu.

Một bộ giải mã cần thiết trong tất cả hiện thực AHB-Lite sử dụng hai hoặc nhiều slave.

➤ **Bộ phân kênh tín hiệu (Multiplexor)**

Một bộ phân kênh tín hiệu từ slave đến master để chọn bus dữ liệu đọc và các tín hiệu đáp ứng từ các slave đến master. Bộ giải mã sẽ cung cấp tín hiệu điều khiển cho bộ phân kênh tín hiệu.

Một bộ phân kênh tín hiệu cần thiết trong tất cả hiện thực AHB-Lite sử dụng hai hoặc nhiều slave.

b/ Hoạt động

Master bắt đầu quá trình truyền bằng cách gửi địa chỉ và các tín hiệu điều khiển. Các tín hiệu này cung cấp thông tin về địa chỉ, hướng truyền, độ rộng của quá trình truyền và báo hiệu nếu đây là một quá trình truyền theo khối. Quá trình truyền có thể là:

- Truyền đơn (single burst)
- Truyền theo khối với địa chỉ tăng (incrementing burst)

- Truyền theo khối với địa chỉ bị cuộn lại khi đến địa chỉ biên (wrapping burst)

Bus dữ liệu ghi sẽ chuyển dữ liệu từ master đến một slave, bus dữ liệu đọc sẽ chuyển dữ liệu từ một slave đến master.

Mỗi quá trình truyền chứa:

- **Pha địa chỉ:** một địa chỉ và một chu kỳ điều khiển
- **Pha dữ liệu:** một hoặc nhiều chu kỳ cho dữ liệu

Một slave không thể yêu cầu pha địa chỉ mở rộng, do đó tất cả slave phải có khả năng lấy mẫu địa chỉ trong thời gian này. Tuy nhiên, một slave có thể yêu cầu master kéo dài pha dữ liệu bằng cách sử dụng tín hiệu **HREADY**. Tín hiệu này ở mức **thấp** khi ở trạng thái chờ để được đưa vào quá trình truyền và cho phép slave có thêm thời gian để cung cấp hoặc lấy mẫu dữ liệu.

Slave sử dụng tín hiệu **HRESP** để cho biết quá trình truyền thành công hay thất bại.

1.3.2 Mô tả các tín hiệu

a/ Tín hiệu toàn cục

Bảng 1-2 liệt kê các tín hiệu toàn cục của giao thức.

Bảng 1-2 Các tín hiệu toàn cục [25]

Tên	Nguồn	Mô tả
HCLK	Nguồn clock	Clock định thời gian cho tất cả quá trình truyền trên bus. Tất cả các tín hiệu đều liên quan đến cạnh lên của HCLK .
HRESETn	Bộ điều khiển reset	Tín hiệu reset được tích cực thấp sẽ reset hệ thống và bus. Đây là tín hiệu tích cực thấp duy nhất trong giao thức AHB-Lite.

b/Các tín hiệu của master

Bảng 1-3 liệt kê các tín hiệu của giao thức được tạo ra bởi một master.

Bảng 1-3 Các tín hiệu của master [25]

Tên	Dịch	Mô tả
HADDR[31:0]	Slave và bộ giải mã	Bus địa chỉ 32-bit.
HBURST[2:0]	Slave	Cho biết quá trình truyền là đơn hay là một phần của truyền theo khối. Chiều dài khối được hỗ trợ là 4,8 và 16 gói.
HMASTLOCK	Slave	Khi ở mức cao, tín hiệu này cho biết quá trình truyền hiện tại không liên tục.
HPROT[3:0]	Slave	Cung cấp thêm thông tin về việc truy cập bus và được sử dụng bởi bất kỳ module khác tùy theo mức độ bảo vệ.
HSIZE[2:0]	Slave	Cho biết kích thước truyền, thường là byte, halfword hay word. Giao thức cho phép kích thước truyền rộng hơn lên đến tối đa là 1024 bit.
HTRANS[1:0]	Slave	Cho biết kiểu truyền của quá trình truyền hiện tại. Có thể là: <ul style="list-style-type: none"> • IDLE • BUSY • NONSEQUENTIAL • SEQUENTIAL
HWDATA[31:0]	Slave	Bus dữ liệu ghi từ master đến các slave thông qua quá trình ghi. Kích thước bus dữ liệu nhỏ nhất là 32 bit được đề nghị, tuy nhiên có thể được mở rộng để hoạt động ở băng thông rộng hơn.
HWRITE	Slave	Cho biết hướng truyền. Tín hiệu này cho biết quá trình ghi khi ở mức cao và quá trình đọc khi ở mức thấp.

c/ Các tín hiệu của slave

Bảng 1-4 liệt kê các tín hiệu của giao thức được tạo ra bởi một slave.

Bảng 1-4 Các tín hiệu của slave [25]

Tên	Đích	Mô tả
HRDATA[31:0]	Bộ phân kênh tín hiệu	Trong quá trình đọc, bus dữ liệu đọc truyền dữ liệu từ slave được lựa chọn đến bộ phân kênh tín hiệu. Bộ phân kênh tín hiệu sau đó sẽ truyền dữ liệu đến master. Kích thước bus dữ liệu nhỏ nhất là 32 bit được đề nghị, tuy nhiên có thể được mở rộng để hoạt động ở băng thông rộng hơn.
HREADYOUT	Bộ phân kênh tín hiệu	Khi ở mức cao, tín hiệu này cho biết một quá trình truyền đã kết thúc trên bus. Tín hiệu này có thể ở mức thấp để kéo dài việc truyền.
HRESP	Bộ phân kênh tín hiệu	Tín hiệu này sau khi qua bộ phân kênh tín hiệu sẽ cung cấp cho master thêm thông tin về trạng thái của một quá trình truyền. Khi ở mức thấp, tín hiệu này cho biết trạng thái truyền là OKAY. Khi ở mức cao, tín hiệu này cho biết trạng thái truyền là ERROR.

d/ Các tín hiệu của bộ giải mã

Bảng 1-5 liệt kê các tín hiệu của giao thức được tạo ra bởi bộ giải mã.

Bảng 1-5 Các tín hiệu của bộ giải mã [25]

Tên	Đích	Mô tả
HSELx	Slave	Mỗi AHB-Lite slave có tín hiệu lựa chọn riêng HSELx và tín hiệu này cho biết quá trình truyền hiện tại được thực hiện với slave được chọn. Khi slave mới được lựa chọn phải theo dõi tình trạng của tín hiệu HREADY để đảm bảo rằng quá trình truyền trước đó đã hoàn tất, trước khi đáp ứng với quá trình truyền hiện tại. Tín hiệu HSELx là một tổ hợp giải mã của bus địa chỉ.

e/ Các tín hiệu của bộ phân kênh tín hiệu

Bảng 1-6 liệt kê các tín hiệu của giao thức được tạo ra bởi bộ phân kênh tín hiệu.

Bảng 1-6 Các tín hiệu của bộ phân kênh tín hiệu [25]

Tên	Đích	Mô tả
HRDATA[31:0]	Master	Bus dữ liệu đọc, được chọn bởi bộ giải mã.
HREADY	Master và slave	Khi ở mức cao, tín hiệu này cho master và tất cả các slave biết quá trình truyền trước đó đã hoàn thành.
HRESP	Master	Đáp ứng của quá trình truyền, được chọn bởi bộ giải mã.

1.3.3 Quá trình truyền

a/ Quá trình truyền cơ bản

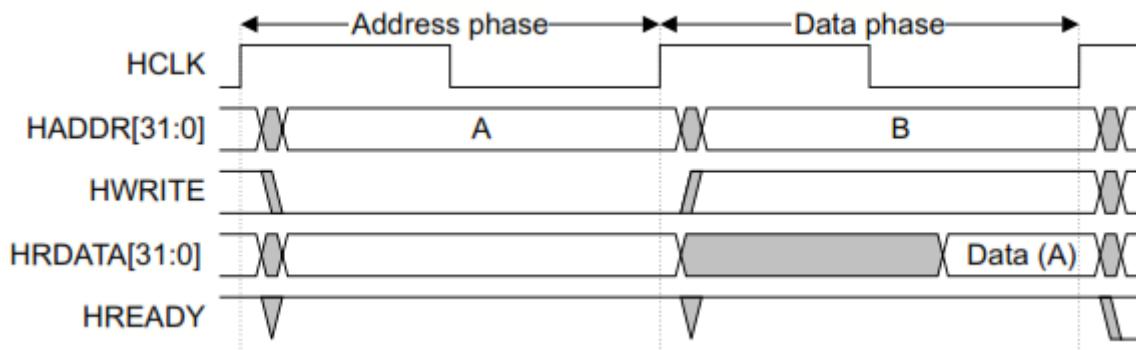
Một quá trình truyền AHB-Lite chứa 2 pha:

- Pha địa chỉ: Kéo dài một chu kỳ **HCLK**, có thể được mở rộng bởi bus transfer trước đó.
- Pha dữ liệu: Có thể kéo dài nhiều chu kỳ **HCLK**. Sử dụng tín hiệu **HREADY** để điều khiển số chu kỳ xung clock được yêu cầu để hoàn thành một quá trình truyền.

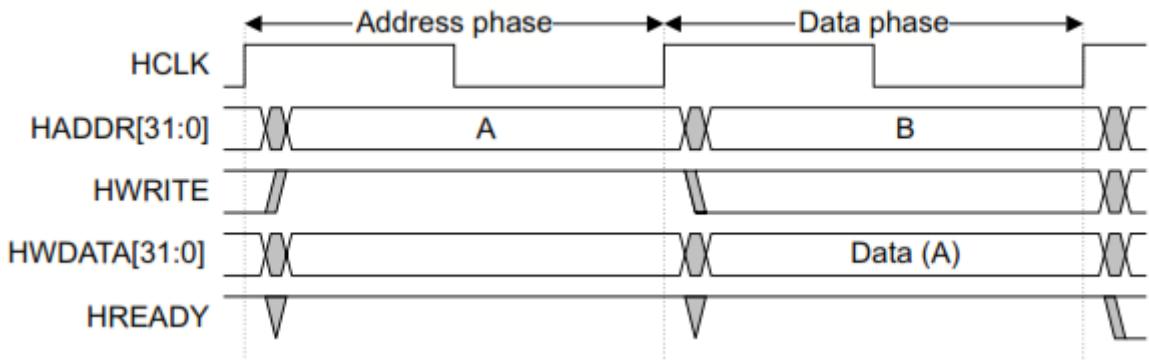
Tín hiệu **HWRITE** điều khiển hướng truyền từ master hoặc đến master. Do đó khi:

- HWRITE** ở mức cao cho biết quá trình ghi và master sẽ扩散 dữ liệu trên bus dữ liệu ghi **HWDATA[31:0]**.
- HWRITE** ở mức thấp cho biết quá trình đọc và slave phải tạo ra dữ liệu trên bus dữ liệu đọc **HRDATA[31:0]**.

Quá trình truyền đơn giản nhất không có trạng thái chờ, do đó chỉ chứa một chu kỳ của pha địa chỉ và một chu kỳ của pha dữ liệu. Hình 1.18 biểu diễn một quá trình đọc đơn giản và Hình 1.19 biểu diễn một quá trình ghi đơn giản.



Hình 1.18 Quá trình đọc [25]



Hình 1.19 Quá trình ghi [25]

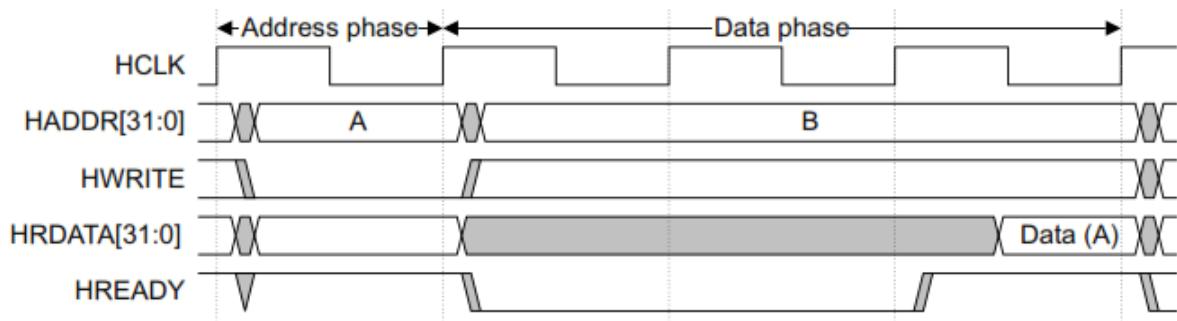
Trong một quá trình truyền đơn giản không có trạng thái chờ:

1. Master đưa địa chỉ và các tín hiệu điều khiển đến bus sau cạnh lén của **HCLK**.
2. Sau đó slave sẽ lấy mẫu địa chỉ và thông tin điều khiển ở cạnh lén **HCLK** tiếp theo.
3. Sau khi đã lấy mẫu địa chỉ và thông tin điều khiển, slave bắt đầu đáp ứng với **HREADY** thích hợp. Đáp ứng này sẽ được master lấy mẫu ở cạnh lén **HCLK** thứ ba.

Ví dụ đơn giản trên minh họa pha địa chỉ và pha dữ liệu trong các chu kì xung clock khác nhau. Pha địa chỉ của một quá trình truyền diễn ra trong pha dữ liệu của quá trình truyền trước đó. Việc chồng lén nhau của pha địa chỉ và pha dữ liệu này cho phép hoạt động hiệu năng cao trong khi vẫn cung cấp đủ thời gian cho slave đáp ứng.

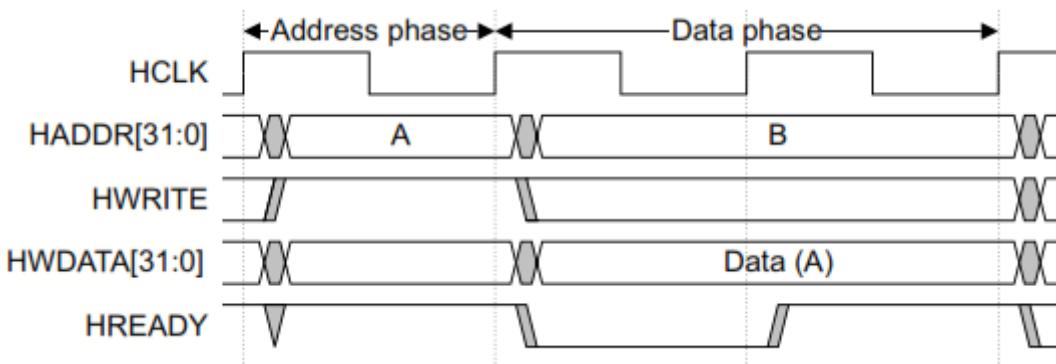
Một slave có thể chèn trạng thái chờ vào bất kỳ quá trình truyền nào để cho phép thêm thời gian để hoàn thành.

Hình 1.20 biêt diễn một quá trình đọc với hai trạng thái chờ.



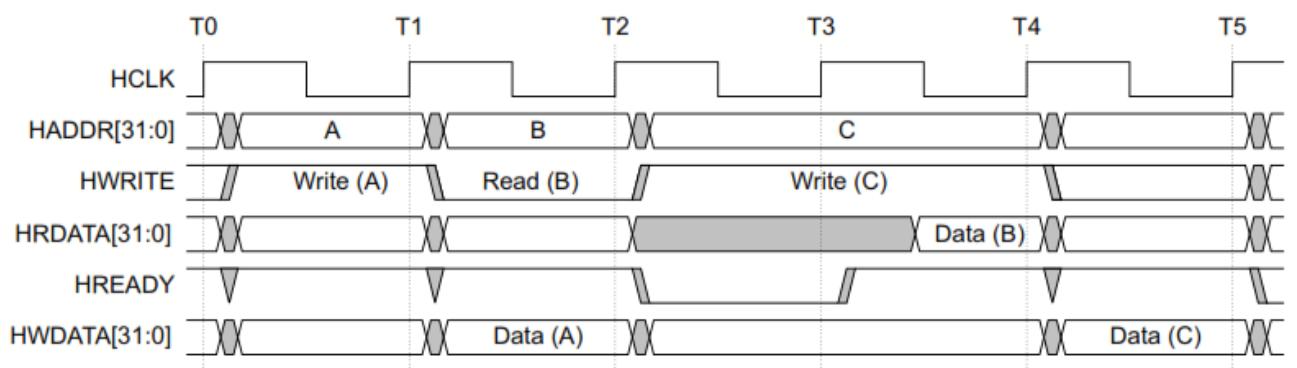
Hình 1.20 Quá trình đọc với trạng thái chờ [25]

Hình 1.21 biêt diêt một quá trình ghi với một trạng thái chờ.



Hình 1.21 Quá trình ghi với trạng thái chờ [25]

Khi một quá trình truyền được mở rộng theo cách trên sẽ làm kéo dài pha địa chỉ của quá trình truyền tiếp theo. Hình 1.22 biêt diêt ba quá trình truyền cho ba địa chỉ riêng A, B, C với pha địa chỉ C kéo dài.



Hình 1.22 Nhiều quá trình truyền [25]

Trên Hình 1.22:

- Quá trình truyền đến địa chỉ A và C không có trạng thái chờ.
- Quá trình truyền đến địa chỉ B có một trạng thái chờ.
- Việc mở rộng pha dữ liệu của quá trình truyền đến địa chỉ B làm mở rộng pha địa chỉ của quá trình truyền đến địa chỉ C.

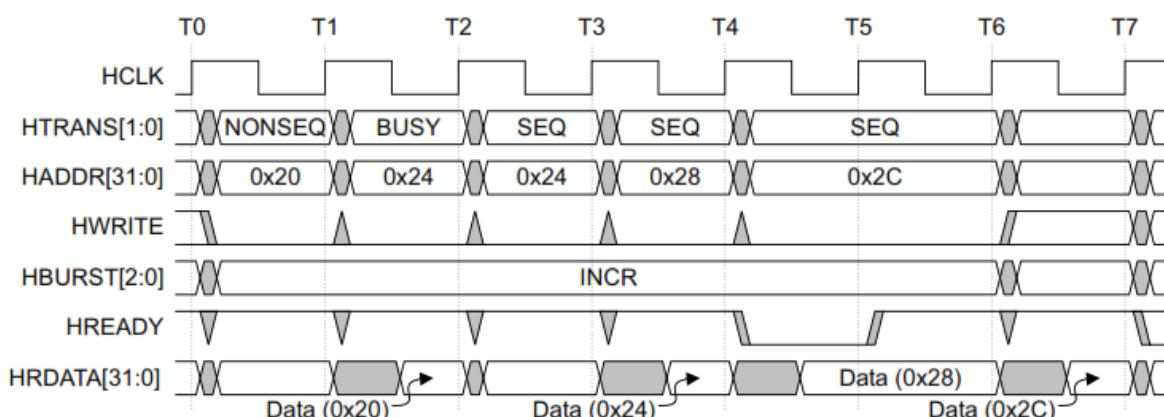
b/ Kiểu truyền

Kiểu truyền có thể chia làm bốn loại, điều khiển bởi tín hiệu **HTRANS[1:0]**.

Bảng 1-7 Các kiểu truyền [25]

HTRANS[1:0]	Kiểu	Mô tả
b00	IDLE	Cho biết không có quá trình truyền nào được yêu cầu. Master sử dụng IDLE khi không muốn truyền quá trình truyền này hoặc chấm dứt một quá trình truyền bị khóa. Slave phải luôn cung cấp đáp ứng OKAY và bỏ qua quá trình truyền này.
b01	BUSY	Cho phép master chèn thêm chu kỳ không hoạt động giữa một khối. Kiểu truyền này cho biết master tiếp tục truyền với một khối nhưng quá trình truyền tiếp theo không thể thực hiện ngay lập tức. Khi master sử dụng kiểu truyền BUSY, địa chỉ và các tín hiệu điều khiển phải được đưa tới quá trình truyền tiếp theo của khối. Chỉ những khối không xác định độ dài mới có quá trình truyền BUSY ở chu kỳ cuối cùng của khối. Slave phải luôn cung cấp đáp ứng OKAY và bỏ qua quá trình truyền này.
b10	NONSEQ	Cho biết quá trình truyền đơn hoặc quá trình truyền đầu tiên của truyền theo khối. Địa chỉ và các tín hiệu điều khiển không liên quan đến quá trình truyền trước đó.
b11	SEQ	Những quá trình truyền còn lại của truyền theo khối là tuần tự và địa chỉ sẽ liên quan đến địa chỉ trước đó. Thông tin điều khiển sẽ giống như quá trình truyền trước đó. Địa chỉ sẽ bằng địa chỉ của quá trình truyền trước cộng với kích thước của quá trình truyền điều khiển bởi tín hiệu HSIZE[2:0] .

Hình 1.23 Biểu diễn việc sử dụng các kiểu truyền NONSEQ, BUSY và SEQ.



Hình 1.23 Ví dụ về kiểu truyền [25]

Trên Hình 1.23:

- **T0-T1:** Quá trình đọc 4 beat bắt đầu với quá trình truyền NONSEQ.
- **T1-T2:** Master không thể thực hiện beat thứ hai nên chèn quá trình truyền BUSY để hoãn lại việc bắt đầu beat thứ hai. Slave cung cấp dữ liệu đọc cho beat đầu tiên.
- **T2-T3:** Master sẵn sàng bắt đầu beat thứ hai, do đó quá trình truyền SEQ được bật. Master bỏ qua data mà slave cung cấp trên bus dữ liệu đọc.
- **T3-T4:** Master thực hiện beat thứ ba. Slave cung cấp dữ liệu đọc cho beat thứ hai.
- **T4-T5:** Master thực hiện beat cuối cùng. Slave không cho phép kết thúc quá trình truyền nên sử dụng **HREADY** để chèn một trạng thái chờ.
- **T5-T6:** Slave cung cấp dữ liệu đọc cho beat thứ ba.
- **T6-T7:** Slave cung cấp dữ liệu đọc cho beat cuối cùng.

c/ Khóa truyền

Master yêu cầu khóa truy cập bằng tín hiệu **HMASTLOCK**. Tín hiệu này cho slave biết quá trình truyền hiện tại không thể chia ra, vì vậy phải được thực hiện trước quá trình truyền khác.

Hầu hết slave không có yêu cầu thực hiện **HMASTLOCK** vì chỉ có thể thực hiện quá trình truyền theo thứ tự được nhận.

d/ Kích thước truyền

Tín hiệu **HSIZE[2:0]** cho biết kích thước của dữ liệu được truyền. Bảng 1-8 liệt kê các kích thước truyền có thể có.

Bảng 1-8 Các kích thước truyền [25]

HSIZE[2]	HSIZE[1]	HSIZE[0]	Kích thước (bit)	Mô tả
0	0	0	8	Byte
0	0	1	16	Halfword
0	1	0	32	Word
0	1	1	64	Doubleword
1	0	0	128	4-word line
1	0	1	256	8-word line
1	1	0	512	-
1	1	1	1024	-

Sử dụng **HSIZE** kết hợp với **HBURST** để xác định ranh giới cho wrapping burst.

Tín hiệu **HSIZE** có cùng timing với bus địa chỉ nhưng phải liên tục trong một khối.

e/ Quá trình truyền theo khối

Khối gồm 4, 8, và 16 beat, khối có độ dài không xác định, và quá trình truyền đơn được định nghĩa trong giao thức này. Có 2 loại truyền theo khối:

- Truyền theo khối với địa chỉ tăng (incrementing burst)
- Truyền theo khối với địa chỉ bị cuộn lại khi đến địa chỉ biên (wrapping burst)

Số beat được điều khiển bởi **HBURST** và kích thước được điều khiển bởi **HSIZE**.

Thí dụ, cho một wrapping burst với 4 beat có kích thước là word (4 byte), nó sẽ truy cập vào ranh giới 16 byte. Nếu địa chỉ bắt đầu là 0x34 thì sẽ gồm bốn quá trình truyền lần lượt tại các địa chỉ 0x34, 0x38, 0x3C và 0x30.

Bảng 1-9 liệt kê các loại truyền theo khối có thể có.

Bảng 1-9 Các kiểu truyền theo khối [25]

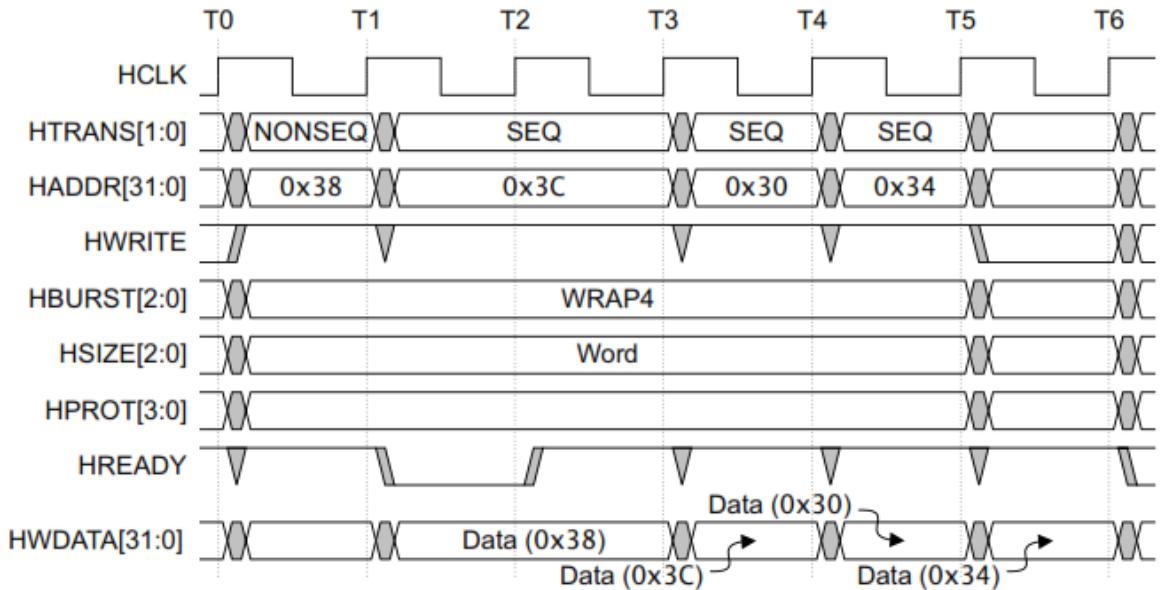
HBURST[2:0]	Kiểu	Mô tả
b000	SINGLE	Single burst
b001	INCR	Incrementing burst với độ dài không xác định
b010	WRAP4	4-beat wrapping burst
b011	INCR4	4-beat incrementing burst
b100	WRAP8	8-beat wrapping burst
b101	INCR8	8-beat incrementing burst
b110	WRAP16	16-beat wrapping burst
b111	INCR16	16-beat incrementing burst

Tất cả quá trình truyền trong một khối phải có địa chỉ giới hạn bằng với kích thước truyền. Thí dụ, quá trình truyền có kích thước word thì địa chỉ giới hạn phải là word (**HADDR[1:0] = b00**), quá trình truyền có kích thước halfword thì địa chỉ giới hạn phải là halfword (**HADDR[0] = 0**).

Một số ví dụ về truyền theo khối:

➤ 4-beat wrapping burst, WRAP4

Hình 1.24 biểu diễn một quá trình ghi với 4-beat wrapping burst, với một trạng thái chờ thêm vào quá trình truyền đầu tiên.

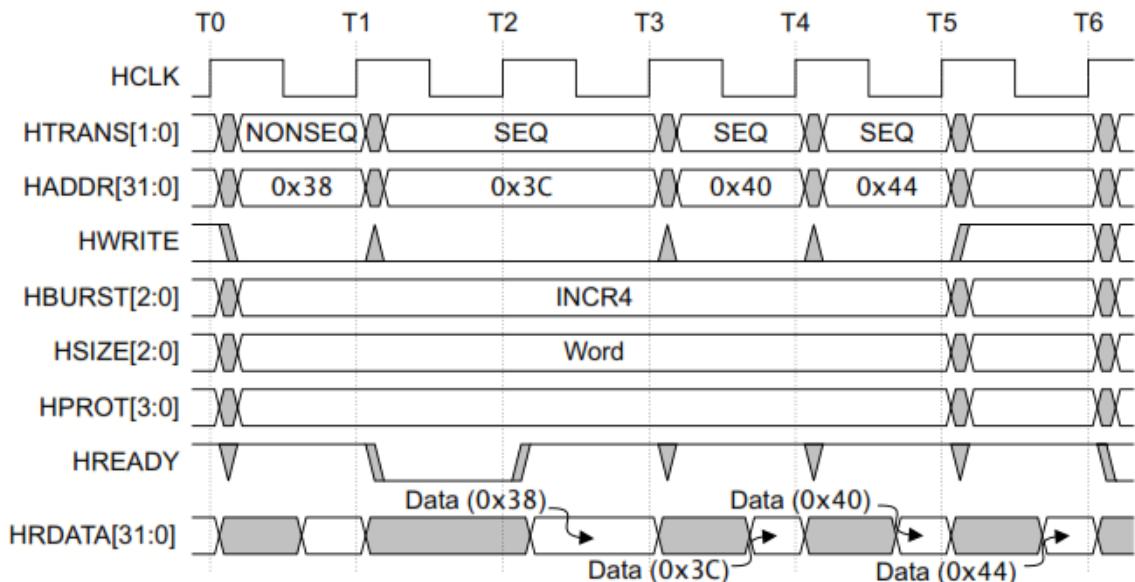


Hình 1.24 4-beat wrapping burst [25]

Vì khôi có 4 beat với kích thước truyền là word nên địa chỉ sẽ cuộn lại trong 16 byte, do đó địa chỉ của quá trình truyền tiếp theo địa chỉ 0x3C là 0x30.

➤ 4-beat incrementing burst, INCR4

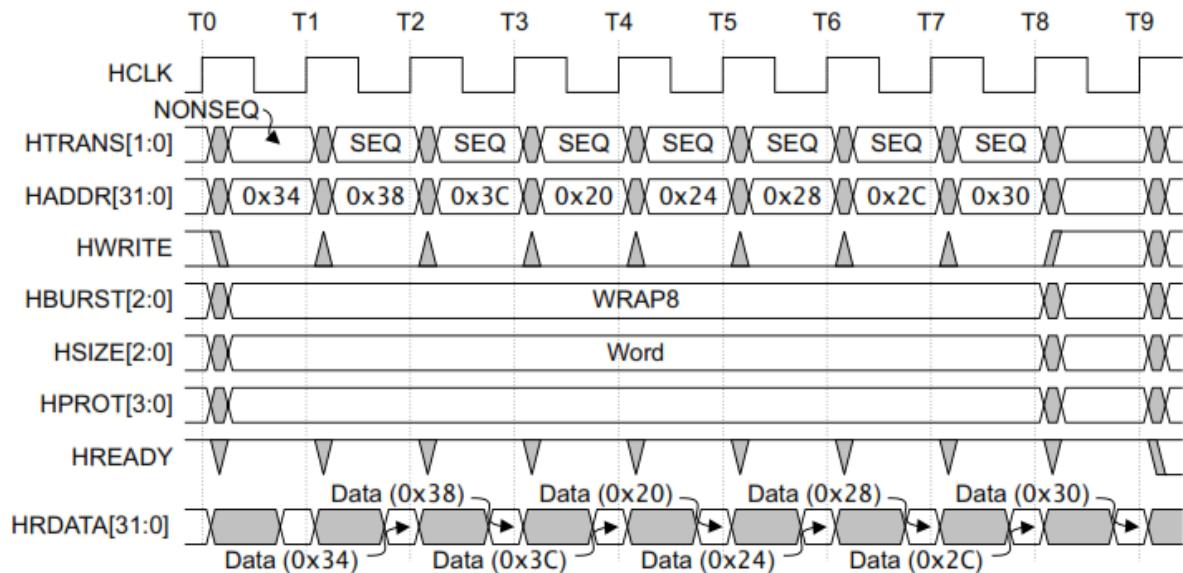
Hình 1.25 biểu diễn một quá trình đọc với 4-beat incrementing burst, với một trạng thái chờ thêm vào quá trình truyền đầu tiên. Trong trường hợp này, địa chỉ sẽ không cuộn lại trong giới hạn 16 byte, do đó địa chỉ của quá trình truyền tiếp theo địa chỉ 0x3C là 0x40.



Hình 1.25 4-beat incrementing burst [25]

➤ **8-beat wrapping burst, WRAP8**

Hình 1.26 biếu diễn một quá trình đọc với 8-beat wrapping burst.

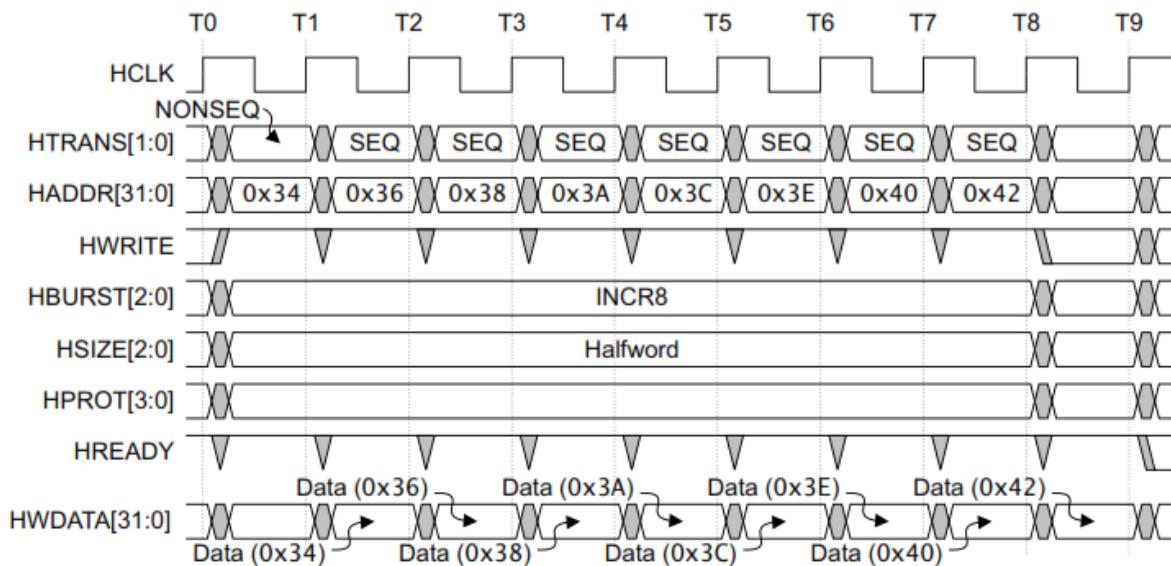


Hình 1.26 8-beat wrapping burst [25]

Vì khói có 8 beat với kích thước truyền là word nên địa chỉ sẽ cuộn lại trong 32 byte, do đó địa chỉ của quá trình truyền tiếp theo địa chỉ 0x3C là 0x20.

➤ **8-beat incrementing burst, INCR8**

Hình 1.27 biếu diễn một quá trình ghi với 8-beat incrementing burst.

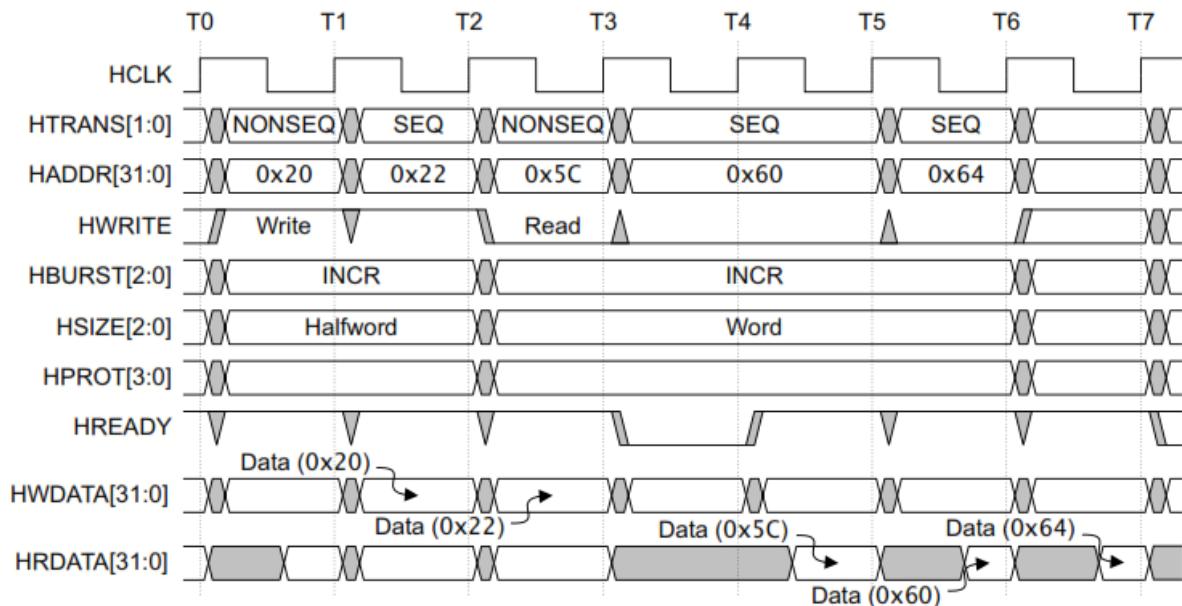


Hình 1.27 8-beat incrementing burst [25]

Khối này có kích thước truyền là halfword, do đó địa chỉ sẽ tăng lên 2. Vì là incrementing burst nên địa chỉ tiếp tục tăng vượt qua giới hạn địa chỉ 16 byte.

➤ Incrementing burst với độ dài không xác định, INCR

Hình 1.28 biểu diễn quá trình truyền incrementing burst với độ dài không xác định.



Hình 1.28 Truyền theo khối với độ dài không xác định [25]

Hình 1.28 có 2 khối:

- Khối thứ nhất để ghi với kích thước truyền là halfword bắt đầu tại địa chỉ 0x20. Địa chỉ sẽ tăng lên 2.
- Khối thứ hai để đọc với kích thước truyền là word bắt đầu tại địa chỉ 0x5C. Địa chỉ sẽ tăng lên 4.

f/ Tín hiệu bảo vệ

Tín hiệu điều khiển bảo vệ **HPROT[3:0]** cung cấp thêm thông tin về việc truy cập bus và được sử dụng bởi bất kỳ module khác tùy theo mức độ bảo vệ.

Tín hiệu này cho biết quá trình truyền là:

- Tìm nạp mã lệnh hay truy cập dữ liệu
- Truy cập chế độ đặc quyền hay truy cập chế độ người sử dụng

Với master có đơn vị quản lý bộ nhớ, tín hiệu này cho biết truy cập bộ nhớ cache hay bộ đệm.

1.3.4 Kết nối bên trong bus

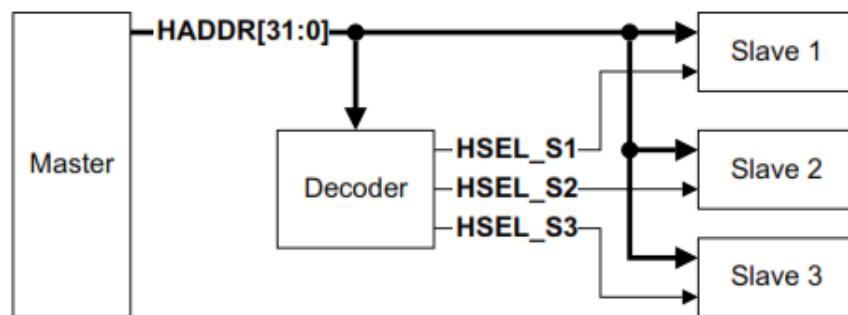
a/ Giải mã địa chỉ

Bộ giải mã địa chỉ cung cấp một tín hiệu điều khiển **HSELx** cho mỗi slave trên bus. Phương thức giải mã đơn giản này được khuyến khích để tránh logic giải mã phức tạp và đảm bảo hoạt động tốc độ cao.

Slave chỉ lấy mẫu **HSELx**, địa chỉ và các tín hiệu điều khiển khi **HREADY** ở mức cao.

Không gian địa chỉ tối thiểu có thể được phân bổ cho một slave là 1KB. Tất cả master được thiết kế để không thể thực hiện các quá trình truyền vượt quá giới hạn địa chỉ 1KB. Điều này đảm bảo truyền theo khối sẽ không bao giờ đi qua một giới hạn địa chỉ.

Hình 1.29 biểu diễn các tín hiệu **HSELx** được tạo ra bởi bộ giải mã.

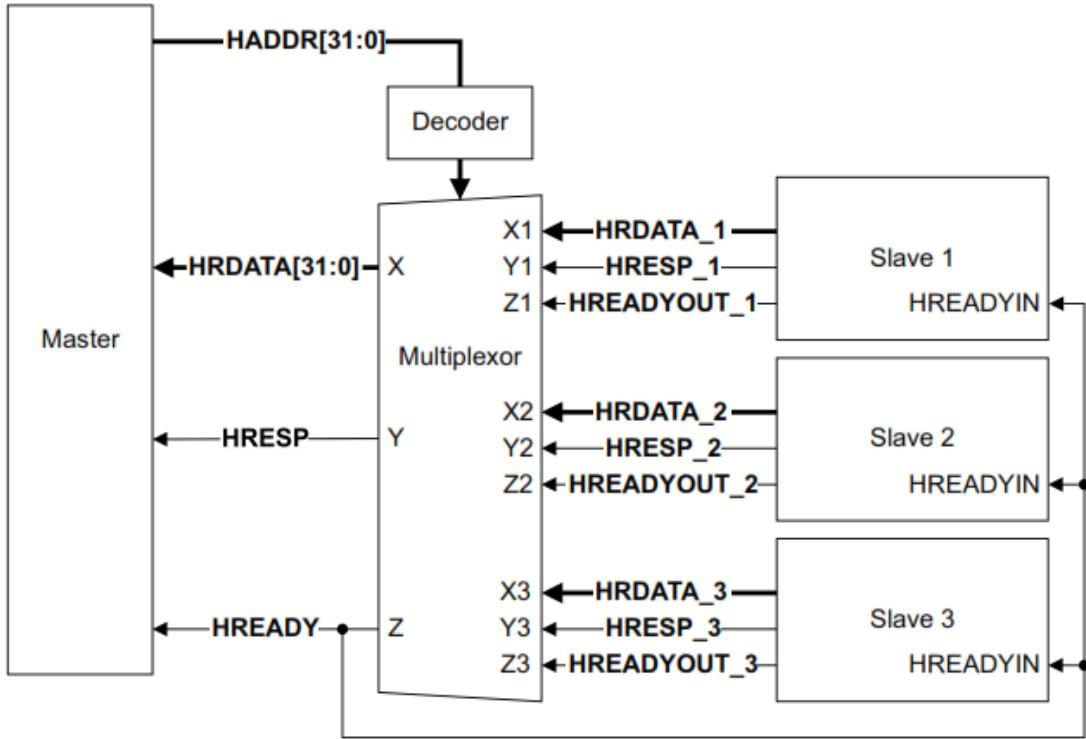


Hình 1.29 Các tín hiệu lựa chọn slave [25]

b/ Kết nối bên trong bus

Giao thức AHB-Lite sử dụng một bộ phân kênh tín hiệu của dữ liệu đọc. Master gửi địa chỉ và các tín hiệu điều khiển đến tất cả slave, bộ giải mã sẽ chọn slave thích hợp. Bất kỳ dữ liệu đọc từ slave được chọn phải đi qua bộ phân kênh tín hiệu mới đến master.

Hình 1.30 biểu diễn cấu trúc của bộ phân kênh tín hiệu với các kết nối được thiết kế với ba slave.



Hình 1.30 Bộ phân kênh tín hiệu và các kết nối [25]

1.3.5 Đáp ứng truyền của slave

Sau khi master bắt đầu một quá trình truyền, slave sẽ điều khiển việc truyền như thế nào. Master không thể hủy bỏ quá trình truyền khi nó đã bắt đầu.

Slave phải cung cấp đáp ứng cho biết trạng thái của quá trình truyền khi nó được truy cập. Trạng thái truyền được biểu diễn bởi tín hiệu **HRESP**. Bảng 1-10 liệt kê các trạng thái của **HRESP**.

Bảng 1-10 Tín hiệu HRESP [25]

HRESP	Đáp ứng	Mô tả
0	OKAY	Quá trình truyền thành công hoặc thêm các chu kỳ để slave hoàn thành yêu cầu. Tín hiệu HREADY cho biết quá trình truyền chưa xong hoặc đã hoàn thành.
1	ERROR	Có lỗi xảy ra trong quá trình truyền. Đáp ứng gồm hai chu kỳ được yêu cầu trong điều kiện lỗi với HREADY được xác nhận trong chu kỳ thứ hai.

Đáp ứng truyền là tổ hợp của tín hiệu **HRESP** và **HREADY**. Bảng 1-11 liệt kê các đáp ứng truyền dựa trên trạng thái của hai tín hiệu này.

Bảng 1-11 Đáp ứng truyền [25]

HRESP	HREADY	
	0	1
0	Quá trình truyền chưa xong	Quá trình truyền đã thành công
1	Quá trình truyền bị lỗi, chu kì thứ nhất	Quá trình truyền bị lỗi, chu kì thứ hai

Slave có thể kết thúc quá trình truyền theo ba cách:

- Hoàn thành quá trình truyền ngay lập tức.
- Chèn một hay nhiều trạng thái chờ để có thêm thời gian hoàn thành quá trình truyền.
- Gửi đáp ứng ERROR cho biết quá trình truyền thất bại.

a/ Quá trình truyền kết thúc

Quá trình truyền kết thúc thành công khi **HREADY** ở mức cao và **HRESP** là OKAY.

b/ Quá trình truyền chưa xong

Slave sử dụng **HREADY** để chèn số trạng thái chờ thích hợp vào pha dữ liệu của quá trình truyền. Quá trình truyền sau đó sẽ kết thúc với tín hiệu **HREADY** ở mức cao và đáp ứng OKAY cho biết quá trình truyền thành công.

c/ Đáp ứng lỗi

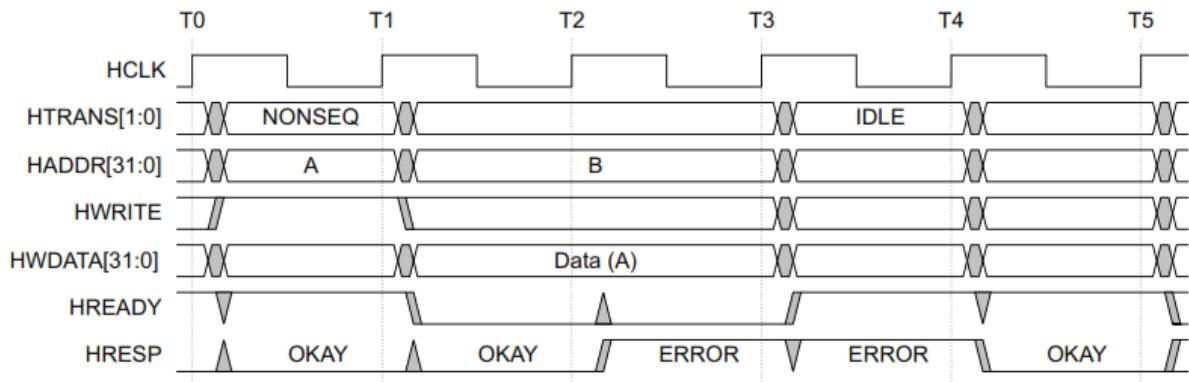
Slave sử dụng đáp ứng ERROR để thông báo các trường hợp lỗi trong quá trình truyền.

Mặc dù đáp ứng OKAY có thể đưa ra trong một chu kì đơn nhưng đáp ứng ERROR cần hai chu kì. Để bắt đầu đáp ứng ERROR, slave sẽ đưa **HRESP** lên mức cao để thông báo lỗi, đồng thời đưa **HREADY** xuống mức thấp để mở rộng quá trình truyền thêm một chu kì. Trong chu kì tiếp theo **HREADY** lên mức cao để kết thúc quá trình truyền, đồng thời **HRESP** vẫn duy trì ở mức cao để báo lỗi.

Đáp ứng cần hai chu kì là cần thiết do pipeline tự nhiên của bus. Tại thời điểm slave bắt đầu đưa ra đáp ứng ERROR thì địa chỉ của quá trình truyền tương ứng cũng quảng bá trên bus. Đáp ứng cần hai chu kì để cung cấp thời gian cần thiết cho master hủy bỏ truy cập kế tiếp và đưa **HTRANS[1:0]** sang IDLE trước khi bắt đầu quá trình truyền kế tiếp.

Nếu slave yêu cầu nhiều hơn hai chu kỳ để cung cấp đáp ứng ERROR thì sẽ chèn thêm các trạng thái chờ lúc bắt đầu quá trình truyền. Trong khoảng thời gian này **HREADY** ở mức thấp và đáp ứng là OKAY.

Hình 1.31 biểu diễn một quá trình truyền với đáp ứng ERROR.



Hình 1.31 Đáp ứng ERROR [25]

Trong Hình 1.31:

- T1-T2: Slave chèn một trạng thái chờ và cung cấp đáp ứng OKAY.
- T2-T3: Slave cung cấp đáp ứng ERROR. Đây là chu kỳ đầu tiên của đáp ứng ERROR vì **HREADY** ở mức thấp.
- T3-T4: Slave cung cấp đáp ứng ERROR. Đây là chu kỳ cuối cùng của đáp ứng ERROR vì **HREADY** bây giờ ở mức cao. Master thay đổi kiểu truyền sang IDLE. Điều này sẽ hủy bỏ một hoạt động truyền tiếp theo đến địa chỉ B.
- T4-T5: Slave cung cấp đáp ứng OKAY.

1.4 Giao thức AMBA APB

1.4.1 Giới thiệu

Advanced Peripheral Bus (APB) là một trong những kiến trúc bus tiên tiến dùng cho vi điều khiển (AMBA-Advanced Microcontroller Bus Architecture). Nó định nghĩa một giao diện chi phí thấp được tối ưu hóa cho tiêu thụ công suất thấp và giảm độ phức tạp giao diện.

Giao thức APB không có pipeline, dùng để kết nối với các ngoại vi băng thông thấp không yêu cầu hiệu năng cao.

Giao thức APB liên quan đến sự thay đổi của tín hiệu với cạnh lên xung clock. Mỗi quá trình truyền cần ít nhất hai chu kỳ.

APB có thể giao tiếp với:

- *AMBA Advanced High-performance Bus (AHB)*
- *AMBA Advanced High-performance Bus Lite (AHB-Lite)*
- *AMBA Advanced Extensible Interface (AXI)*
- *AMBA Advanced Extensible Interface Lite (AXI4-Lite)*

Có thể sử dụng APB để truy cập các thanh ghi điều khiển của các thiết bị ngoại vi.

1.4.2 Mô tả các tín hiệu

Bảng 1-12 liệt kê các tín hiệu của giao thức APB.

Bảng 1-12 Các tín hiệu của giao thức APB [26]

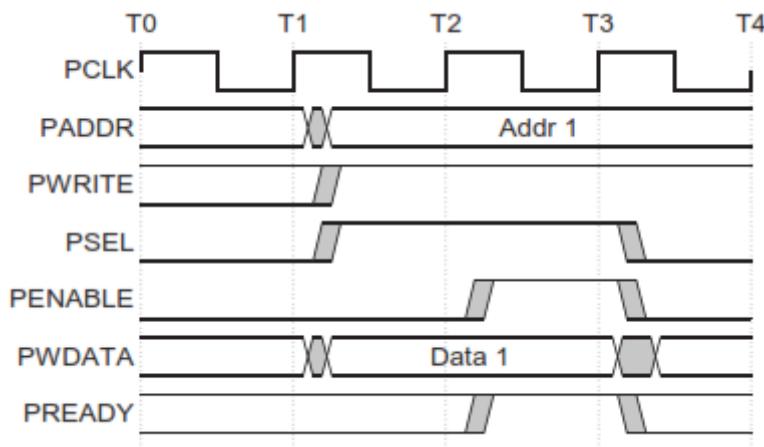
Tên	Nguồn	Mô tả
PCLK	Nguồn clock	Clock. Cạnh lên của PCLK định thời gian cho tất cả quá trình truyền.
PRESETn	Bus hệ thống tương đương	Tín hiệu reset tích cực thấp. Tín hiệu này thường được kết nối trực tiếp với tín hiệu reset bus hệ thống.
PADDR	Cầu APB	Đây là bus địa chỉ, có thể rộng đến 32 bit.
PPROT	Cầu APB	Tín hiệu này cho biết mức độ bảo vệ bình thường, đặc quyền hay an toàn của quá trình truyền và quá trình truyền là truy cập dữ liệu hay truy cập lệnh.
PSELx	Cầu APB	Tín hiệu này để lựa chọn slave và một quá trình truyền dữ liệu được yêu cầu. Mỗi slave có một tín hiệu PSELx riêng.
PENABLE	Cầu APB	Tín hiệu này cho biết chu kỳ thứ hai của quá trình truyền.
PWRITE	Cầu APB	Cho biết hướng truyền. Tín hiệu này cho biết quá trình ghi khi ở mức cao và quá trình đọc khi ở mức thấp.
PWDATA	Cầu APB	Bus dữ liệu ghi, sử dụng khi PWRITE ở mức cao. Có thể rộng đến 32 bit.
PSTRB	Cầu APB	Tín hiệu này cho biết byte nào để cập nhật trong quá trình ghi. PSTRB[n] tương ứng với PWDATA[(8n + 7):(8n)] . Tín hiệu này không được sử dụng trong quá trình đọc.
PREADY	Giao diện slave	Slave sử dụng tín hiệu này để mở rộng quá trình truyền.
PRDATA	Giao diện slave	Bus dữ liệu đọc, sử dụng khi PWRITE ở mức thấp. Có thể rộng đến 32 bit.
PSLVERR	Giao diện slave	Tín hiệu này cho biết quá trình truyền thất bại.

1.4.3 Quá trình truyền

a/ Quá trình ghi

➤ Không có trạng thái chờ

Hình 1.32 biểu diễn một quá trình ghi cơ bản không có trạng thái chờ.



Hình 1.32 Quá trình ghi cơ bản không có trạng thái chờ [26]

Tại T1, một quá trình ghi bắt đầu với địa chỉ **PADDR**, dữ liệu ghi **PWDATA**, tín hiệu ghi **PWRITE**, tín hiệu lựa chọn **PSEL** được lấy mẫu tại cạnh lên của **PCLK**. Đây gọi là pha thiết lập (Setup phase) của quá trình ghi.

Tại T2, tín hiệu cho phép **PENABLE** và tín hiệu sẵn sàng **PREADY** được lấy mẫu tại cạnh lên của **PLCK**.

Khi được xác nhận, **PENABLE** cho biết bắt đầu pha truy cập (Access phase) của quá trình truyền.

Khi được xác nhận, **PREADY** cho biết slave có thể hoàn thành quá trình truyền tại cạnh lên tiếp theo của **PLCK**.

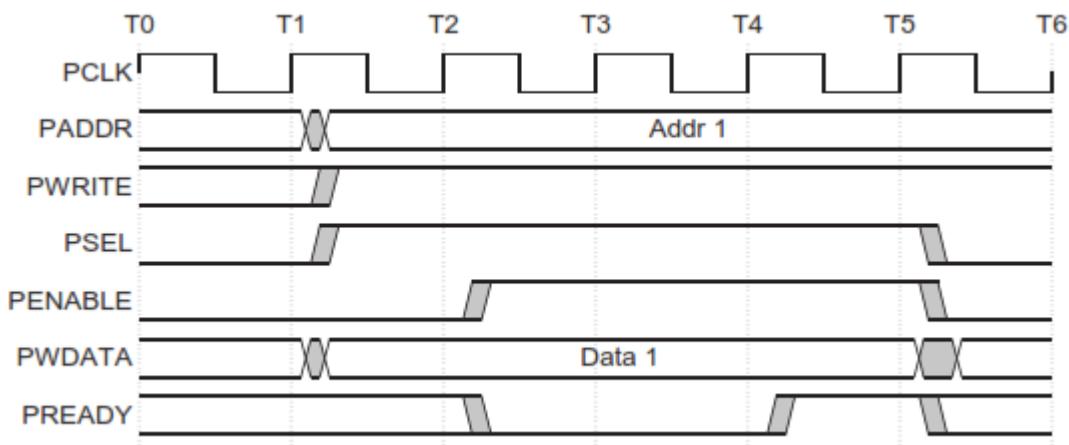
Địa chỉ **PADDR**, dữ liệu ghi **PWDATA** và các tín hiệu điều khiển được duy trì đến khi quá trình truyền hoàn thành ở T3, kết thúc pha truy cập.

Tín hiệu cho phép **PENABLE** được xác nhận lại ở cuối quá trình truyền. Tín hiệu lựa chọn **PSEL** cũng được xác nhận lại trừ khi quá trình truyền được tiếp tục với quá trình truyền tiếp theo.

➤ Có trạng thái chờ

Hình 1.33 biểu diễn cách slave sử dụng tín hiệu **PREADY** để mở rộng quá trình truyền. Trong pha truy cập, khi **PENABLE** ở mức **cao**, slave mở rộng quá trình truyền bằng cách lái **PREADY** ở mức **thấp**. Những tín hiệu sau sẽ duy trì không đổi khi **PREADY** giữ ở mức **thấp**:

- **PADDR**
- **PWRITE**
- **PSEL**
- **PENABLE**
- **PWDATA**
- **PSTRB**
- **PPROT**

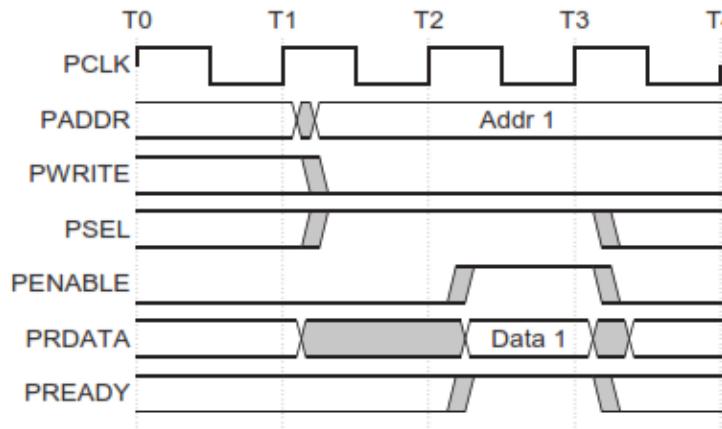


Hình 1.33 Quá trình ghi có trạng thái chờ [26]

b/ Quá trình đọc

➤ Không có trạng thái chờ

Hình 1.34 biểu diễn một quá trình đọc. Slave phải cung cấp dữ liệu trước khi kết thúc quá trình đọc.



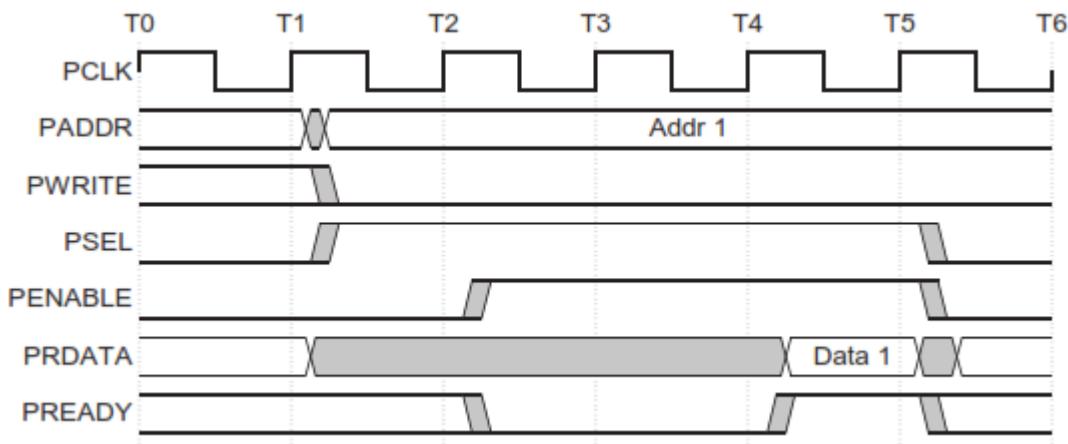
Hình 1.34 Quá trình đọc cơ bản không có trạng thái chờ [26]

➤ Có trạng thái chờ

Hình 1.35 biểu diễn cách slave sử dụng tín hiệu **PREADY** để mở rộng quá trình truyền. Quá trình truyền được mở rộng nếu **PREADY** được lái ở mức **thấp** trong pha truy cập. Những tín hiệu sau sẽ duy trì không đổi khi **PREADY** giữ ở mức **thấp**:

- **PADDR**
- **PWRITE**
- **PSEL**
- **PENABLE**
- **PPROT**

Hình 1.35 biểu diễn hai chu kỳ được thêm vào sử dụng tín hiệu **PREADY**. Tuy nhiên có thể thêm vào một số chu kỳ bất kỳ.



Hình 1.35 Quá trình đọc có trạng thái chờ [26]

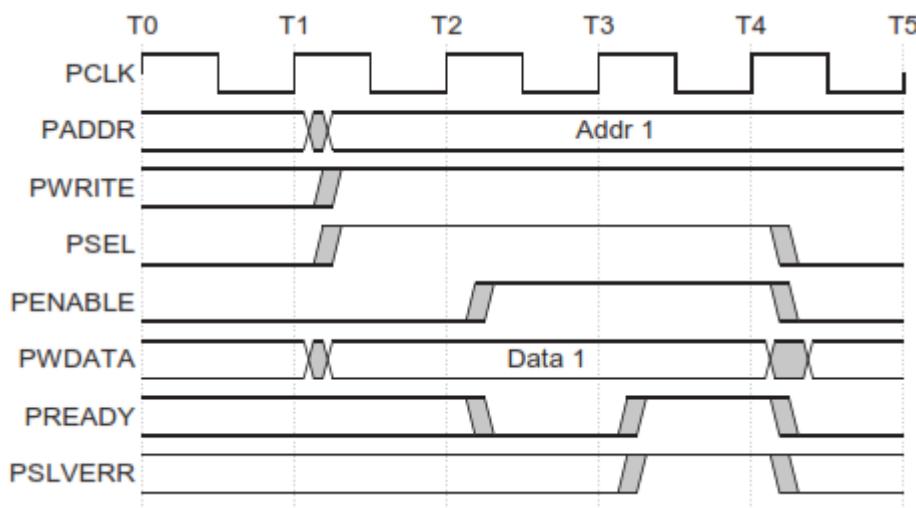
c/ Đáp ứng lỗi

Có thể sử dụng **PSLVERR** để thông báo điều kiện lỗi của quá trình truyền. Điều kiện lỗi có thể xảy ra cả quá trình ghi hoặc quá trình đọc.

PSLVERR chỉ có hiệu lực trong chu kỳ cuối của quá trình truyền khi **PSEL**, **PENABLE**, **PREADY** đều ở mức **cao**.

➤ Quá trình ghi

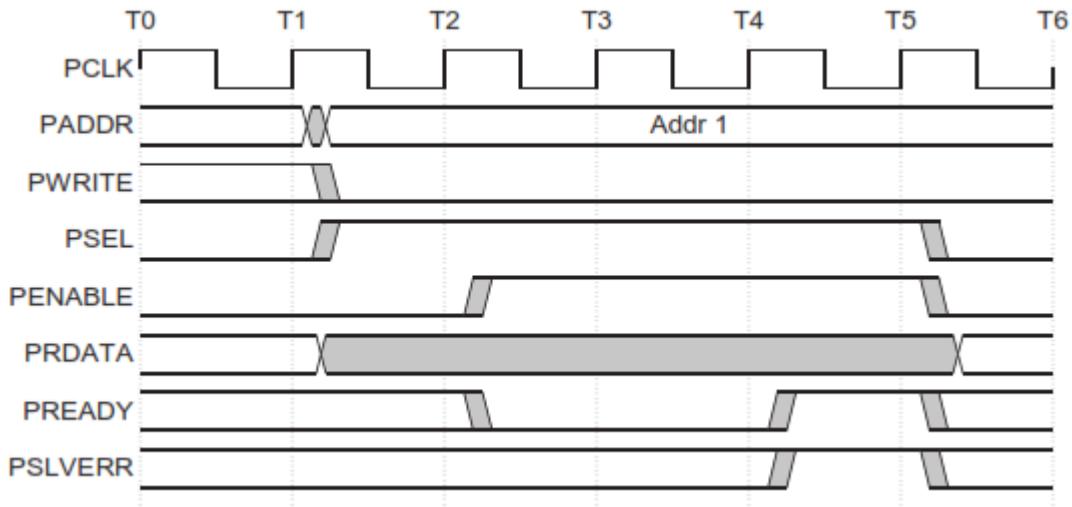
Hình 1.36 biểu diễn ví dụ cho quá trình ghi không thành công với thông báo lỗi.



Hình 1.36 Ví dụ quá trình ghi không thành công [26]

➤ Quá trình đọc

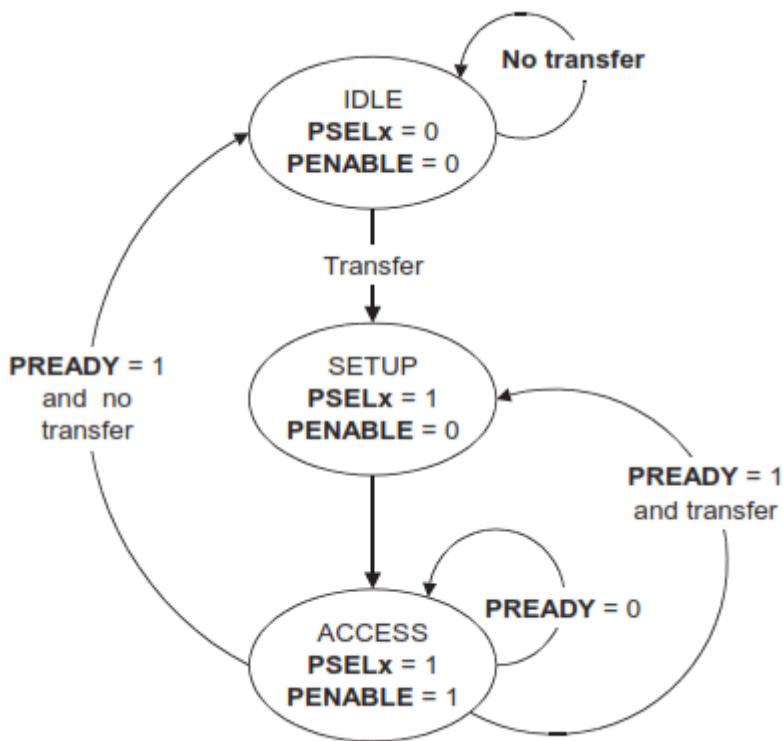
Một quá trình đọc cũng có thể hoàn thành với thông báo lỗi, cho biết không có dữ liệu đọc hợp lệ. Hình 1.37 biểu diễn một quá trình đọc không thành công với thông báo lỗi.



Hình 1.37 Ví dụ quá trình đọc không thành công [26]

1.4.4 Các trạng thái hoạt động

Hình 1.38 biểu diễn các trạng thái hoạt động của APB.



Hình 1.38 Sơ đồ trạng thái [26]

Máy trạng thái hoạt động với các trạng thái sau:

IDLE: Đây là trạng thái mặc định của APB.

SETUP: Khi một quá trình truyền yêu cầu bus chuyển sang trạng thái SETUP, tại đó tín hiệu lựa chọn **PSEL_x** thích hợp được xác nhận. Bus chỉ duy trì trạng thái SETUP trong một chu kỳ và luôn chuyển sang trạng thái ACCESS ở cạnh lên của xung clock kế tiếp.

ACCESS: Tín hiệu cho phép **PENABLE** được xác nhận ở trạng thái này. Địa chỉ, dữ liệu ghi và các tín hiệu điều khiển phải được giữ nguyên từ trạng thái SETUP sang ACCESS. Thoát khỏi trạng thái ACCESS bằng cách điều khiển tín hiệu **PREADY** từ slave:

- Nếu **PREADY** ở mức **thấp** thì bus sẽ tiếp tục ở trạng thái ACCESS.
- Nếu **PREADY** ở mức **cao** thì trạng thái ACCESS sẽ kết thúc và bus sẽ quay về trạng thái IDLE nếu không còn quá trình truyền nào được yêu cầu. Mặt khác, bus sẽ chuyển trực tiếp sang trạng thái SETUP nếu có quá trình truyền tiếp theo.

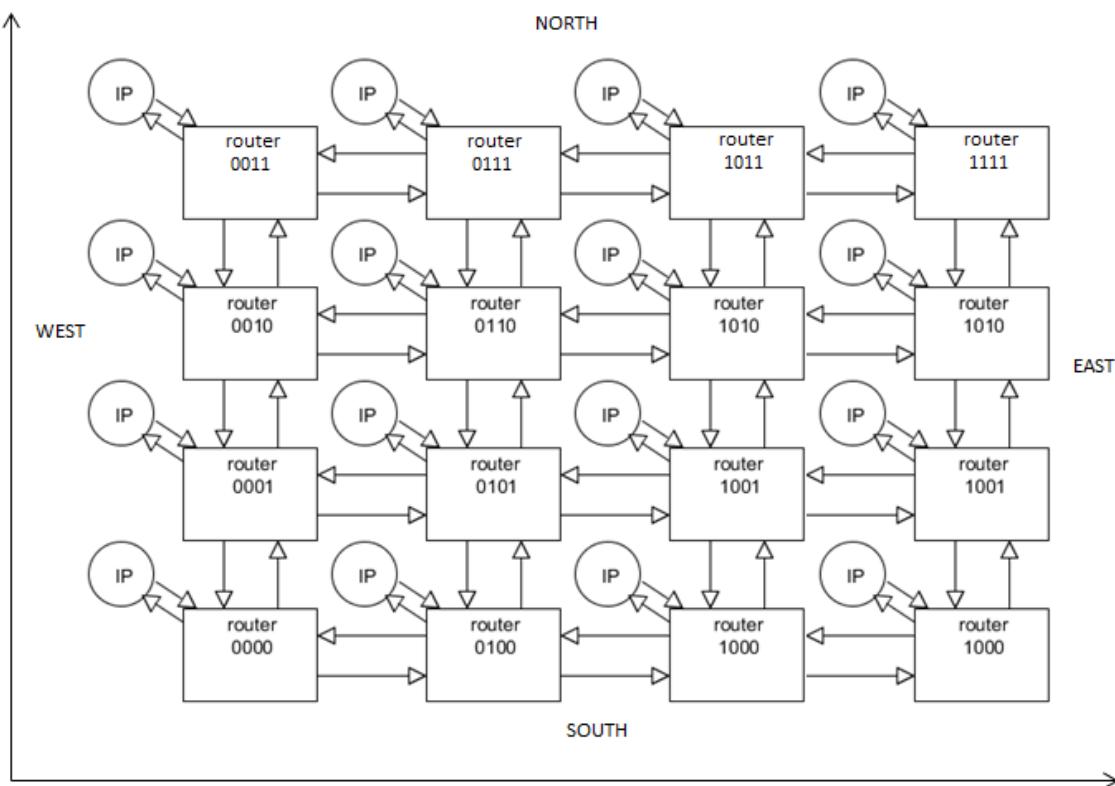
CHƯƠNG 2 THIẾT KẾ KIẾN TRÚC MẠNG NOC

Đề tài sẽ tiếp cận vấn đề độ trễ của router dựa trên hướng giải quyết rút ngắn số trạng thái mà một khung dữ liệu sẽ trải qua trên một router trung gian đồng thời kết hợp với cơ chế pipeline nhằm tăng cường hiệu quả cho mạng NoC.

Một khía cạnh khác được khai thác là định tuyến thích nghi cũng được tiếp cận nhằm tăng hiệu năng hệ thống khi lỗi xảy ra trên tầm mô hình mạng.

2.1 Chọn lựa mô hình mạng

Mục tiêu của đề tài là xây dựng một cấu hình NoC 4x4 sao cho mỗi khung dữ liệu chỉ mất ba chu kỳ xung clock để đi qua một router. Chính vì vậy, việc lựa chọn mô hình (topology) là hết sức quan trọng. Mô hình mạng được lựa chọn thực hiện cần đảm bảo ràng buộc về công nghệ chế tạo cũng như thuận lợi cho việc cài đặt các giải thuật định tuyến và cơ chế điều khiển luồng. Như phân tích ở chương 1, mô hình mạng lưới 2D được xem là mô hình mạng cơ bản để phát triển lên các mô hình mạng phức tạp khác. Chính vì lẽ đó, mô hình mạng lưới 2D đơn giản được chọn lựa nhằm kiểm tra và đánh giá ý tưởng như Hình 2.1.



Hình 2.1 Mô hình NoC 4x4

Như vậy trong Hình 2.1, hệ trục Oxy cho mô hình hai chiều được chọn lựa và các nút router được đặt trên hai chiều này. Mỗi router sẽ có một vị trí cố định được mã hoá bởi các bit. Ví dụ khi router trong mô hình NoC 2D 4x4 ở vị trí 1001 thì có nghĩa là vị trí trên trục Ox là 3 và Oy là 2 hay có thể nói xét về hàng ngang (trục Ox) ta có router thứ 3 từ trái qua và xét về hàng dọc (Oy) ta có router thứ 2 từ dưới lên.

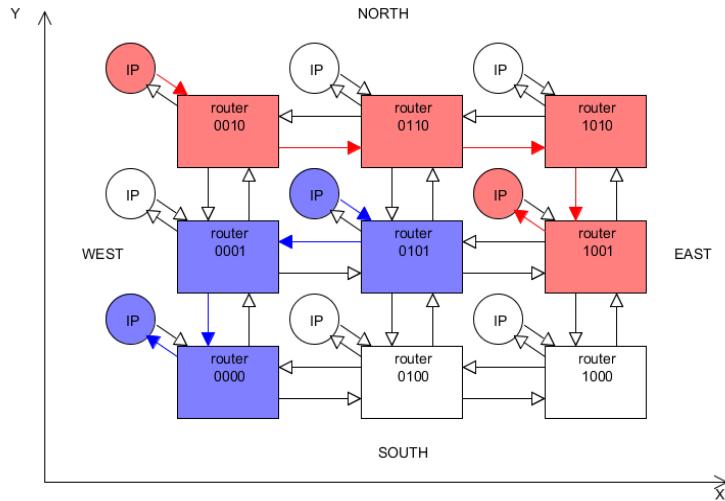
Công việc tiếp theo là quyết định số nút mạng là bao nhiêu? Để đơn giản cho thiết kế, số nút mạng được giới hạn vừa đủ để đánh giá tất cả các trường hợp có thể xảy ra của mô hình thiết kế. Để tài lựa chọn xây dựng mô hình 16 nút mạng như Hình 2.1. Trong mô hình mạng này, tất cả các trường hợp tranh chấp mạng đều có thể được giả lập. Điều này giúp cho việc kiểm tra mô hình được chính xác.

2.2 Lựa chọn giải thuật định tuyến

Giải thuật định tuyến đơn giản được ưa thích trong mô hình lưới 2D là giải thuật định tuyến XY. Tương tự, đề tài cũng sử dụng giải thuật định tuyến này áp dụng cho mô hình NoC 4x4 nhằm mục đích đơn giản hóa trong vấn đề thiết kế và kiểm định hiệu năng của mô hình.

Như đã thấy trong Hình 2.1 các router trong mạng NoC 4x4 được liên kết với nhau thông qua các ngõ vào và ngõ ra. Ngoài liên kết với các router khác, mỗi router còn liên kết với một IP. Tuy nhiên trong mô hình kiểm tra, các IP được thay thế bằng các khói kiến trúc **InputAdaptive** và **OuputAdaptive**. Các router được đánh số từ 0000 cho đến 1010 tương ứng 16 router trong mạng NoC 4x4. Hai chữ số đầu của số liệu này thể hiện vị trí của router trên trục Ox. Hai chữ số tiếp theo của số liệu thể hiện vị trí của router trên trục Oy. Số liệu này sẽ được giới thiệu tiếp ở phần sau khi nói về cấu trúc dữ liệu vì chúng giúp các router biết cần chuyển dữ liệu đến router kề cận nào là chính xác. Ví dụ cho thấy router_0100 cho biết ở vị trí 1 trên trục Ox và vị trí 0 trên trục Oy. Khi muốn mở rộng mô hình lên NoC nxn với giải thuật này cần nhiều bit hơn để mã hóa cho từng vị trí của các router (nút mạng).

Theo như giải thuật định tuyến XY, khi các gói dữ liệu muốn đi từ nút mạng này đến nút mạng khác, gói dữ liệu sẽ được định tuyến truyền đi theo phương Ox trước. Khi gói dữ liệu đến được router có vị trí theo phương Ox trùng với vị trí phương Ox mà gói dữ liệu mong muốn, gói dữ liệu sẽ được định tuyến để được truyền đi theo phương Oy.



Hình 2.2 Gói dữ liệu truyền theo giải thuật định tuyến XY

Hình 2.2 mô tả hai đường truyền riêng biệt tuân theo định tuyến XY đã nói trên: Đường truyền màu đỏ và đường truyền màu xanh. Hình 2.2 cho thấy trên đường truyền thứ nhất (màu đỏ), gói dữ liệu đi theo trình tự: IP nguồn → router_0010 → router_0110 → router_1010 → router_1001 → IP đích. Gói dữ liệu trên đường truyền thứ hai (màu xanh) đi theo trình tự: IP nguồn → router_0101 → router_0001 → router_0000 → IP đích.

2.3 Xây dựng cơ chế điều khiển luồng

Cơ chế điều khiển luồng đảm bảo việc cấp phát các tài nguyên sao cho các gói dữ liệu có thể đi qua mạng trên chip từ IP nguồn đến IP đích. Trong đề tài này sẽ sử dụng cơ chế Wormhole Routing. Trong cơ chế này gói dữ liệu được phân chia thành nhiều khung dữ liệu nhỏ.

Đối với điều khiển luồng ứng dụng cho mạng trên chip, gói dữ liệu là đơn vị cơ bản để thực hiện việc định tuyến và sắp xếp thứ tự, trong khi đó các khung dữ liệu là đơn vị cơ bản để thực hiện việc cấp phát băng thông và bộ đệm. Đặc điểm của cơ chế Wormhole Routing là việc cấp phát băng thông và bộ đệm đều được thực hiện với đơn vị dữ liệu là khung, đó là lý do mà cơ chế này giảm thiểu được thời gian trễ và tận dụng được tài nguyên mạng.

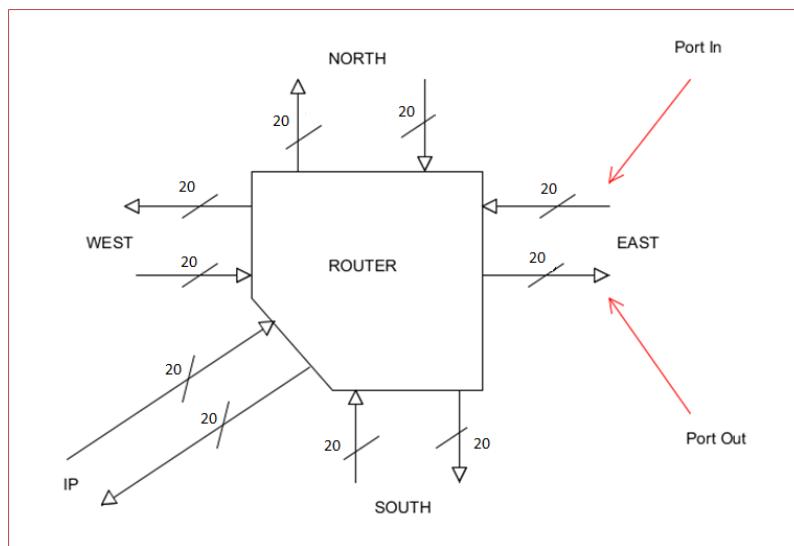
Tuy nhiên, bên cạnh những ưu điểm thì nhược điểm của Wormhole Routing là khả năng nghẽn mạng có thể xảy ra nếu một khung dữ liệu không được cấp phát tài nguyên (bộ đệm), và do đó không được chuyên đi như mong muốn. Điều này dẫn đến treo toàn bộ kênh truyền tại nút mạng đó. Để khắc phục và giảm khả năng tắc nghẽn, việc áp dụng cơ chế điều khiển luồng kết hợp kỹ thuật khen ảo đã được kiểm chứng kết quả.

Mô hình kênh ảo giúp làm giảm khả năng tắc nghẽn mạng trên chip. Càng nhiều kênh ảo được hỗ trợ, khả năng tắc nghẽn mạng càng giảm xuống. Tuy nhiên việc tăng số kênh ảo cũng làm tăng công việc khôi xử lý bên trong router. Do đó, để tài chỉ thực hiện mô hình được hỗ trợ với 2 kênh ảo nhằm hạn chế diện tích cũng như tính phức tạp của router mong muốn nhưng vẫn đảm bảo được yêu cầu đề ra ban đầu. Như vậy, có thể tóm tắt các yếu tố cần thiết để thực hiện mô hình NoC mong muốn.

- Mô hình lưới 2D.
- Cơ chế định tuyến XY.
- Điều khiển luồng Wormhole Routing kết hợp kỹ thuật kênh ảo.
- Mô hình kiểm định NoC 4x4.

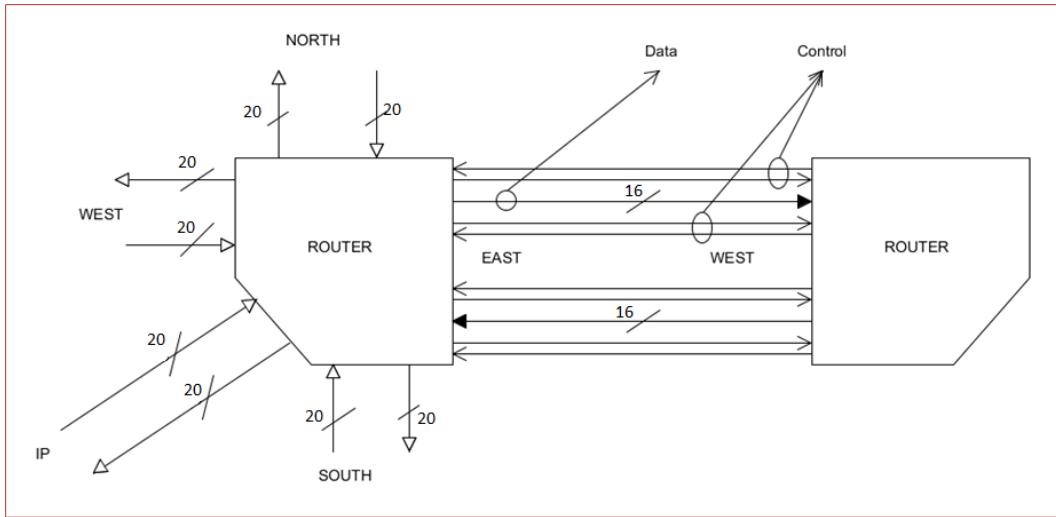
2.4 Mô hình kiến trúc bên ngoài router

Mô hình bên ngoài của router được mô tả như trong Hình 2.3, một router gồm 5 ngõ vào và 5 ngõ ra tạo thành 5 cặp ngõ vào ra cho các hướng khác nhau. Bốn hướng đông, tây, nam và bắc liên kết với các ngõ vào ra khác của các router kế cận trong mạng NoC 4x4. Hướng còn lại cho liên kết với IP tương ứng của router đó. Trong mô hình router này, số chân cho mỗi ngõ vào và ngõ ra là như nhau. Mô tả chi tiết các chân được thực hiện ở phần sau.



Hình 2.3 Kiến trúc bên ngoài của router đơn

Dựa trên Hình 2.4, các ngõ vào/ra được chia thành hai nhóm chính. Nhóm thứ nhất gồm các dây điều khiển (4 dây). Các dây này giữ vai trò điều khiển liên lạc giữa hai router. Nhóm thứ hai gồm các dây còn lại là các đường chia dữ liệu. Lý do chọn số lượng dây cho mỗi nhóm được trình bày chi tiết khi đi vào bên trong cấu trúc của router.

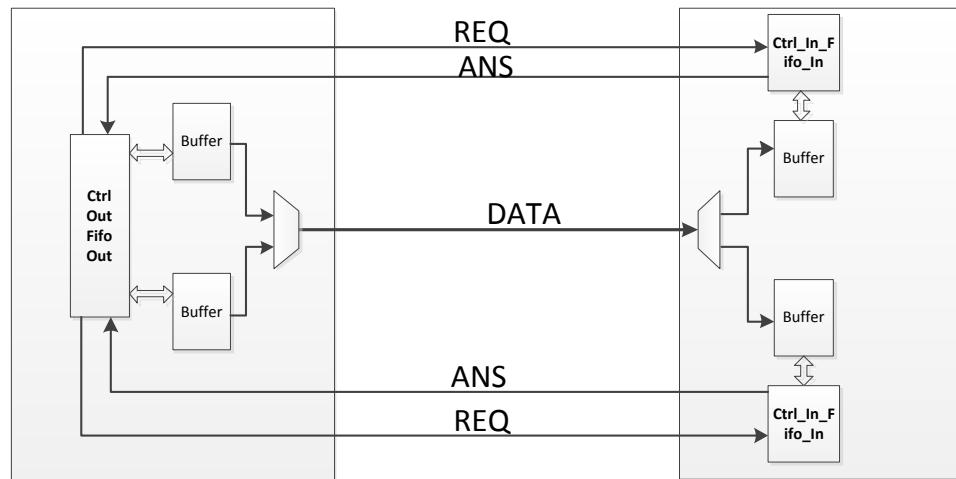


Hình 2.4 Giao diện liên kết giữa hai router

2.5 Mô hình kiến trúc bên trong router

2.5.1 Giao diện một liên kết truyền nhận giữa hai router

Để dễ dàng mô tả các kết nối tín hiệu bên trong một router, một liên kết truyền nhận giữa hai router được phân tích sâu hơn.



Hình 2.5 Kiến trúc bên ngoài của router đơn

Giữa hai router có hai liên kết truyền nhận, ngõ ra của router này là ngõ vào của router liên kết. Hình 2.5 phân tích một liên kết truyền nhận của hai router. Để gói dữ liệu được

truyền nhận giữa hai router, ngoài đường dữ liệu còn cần 4 đường điều khiển (2 tín hiệu **req** và 2 tín hiệu **ans**). Các đường điều khiển này giúp phân xử gói dữ liệu biết được lấy từ nguồn Fifo nào của router truyền và nhận vào ở Fifo nào của router đích.

Như đã nói trên, kiến trúc router được hỗ trợ bởi kỹ thuật kênh ảo. Số lượng kênh ảo được hỗ trợ là 2 cho mỗi ngõ vào/ra. Với số lượng kênh ảo là 2, ở mỗi ngõ vào/ra của router sẽ có hai bộ đệm (2 Fifo) chứa dữ liệu. Tại mỗi thời điểm truyền nhận dữ liệu, chỉ một bộ đệm được xử lý. Mỗi bộ đệm cần hai tín hiệu điều khiển (**ans** và **req**).

Như vậy với mỗi một liên kết truyền nhận giữa hai router có hỗ trợ hai kênh ảo cần 2 cặp tín hiệu **ans** và **req**. Mỗi cặp tín hiệu này phục vụ cho một kênh ảo nhằm giúp hai router hiểu được gói dữ liệu được truyền và nhận bởi những Fifo của cùng một kênh ảo.

Như vậy cứ mỗi liên kết truyền nhận (vào/ra) giữa hai router, kiến trúc router được thiết kế cần 20 đường tín hiệu. 16 đường tín hiệu cho dữ liệu và 4 đường tín hiệu cho điều khiển.

Do đặc tính cấu trúc trên và sự hỗ trợ 2 kênh ảo cho mỗi ngõ vào/ra, mỗi router có giao diện như sau:

- 5 luồng (bus) dữ liệu vào **data_in**, mỗi luồng dữ liệu 16 bit.
- 5 luồng (bus) dữ liệu ra **data_out**, mỗi luồng dữ liệu 16 bit.
- 10 đường điều khiển **ans** ngõ vào điều khiển dữ liệu vào.
- 10 đường điều khiển **req** ngõ vào điều khiển dữ liệu vào.
- 10 đường điều khiển **ans** ngõ ra điều khiển dữ liệu ra.
- 10 đường điều khiển **req** ngõ ra điều khiển dữ liệu ra.

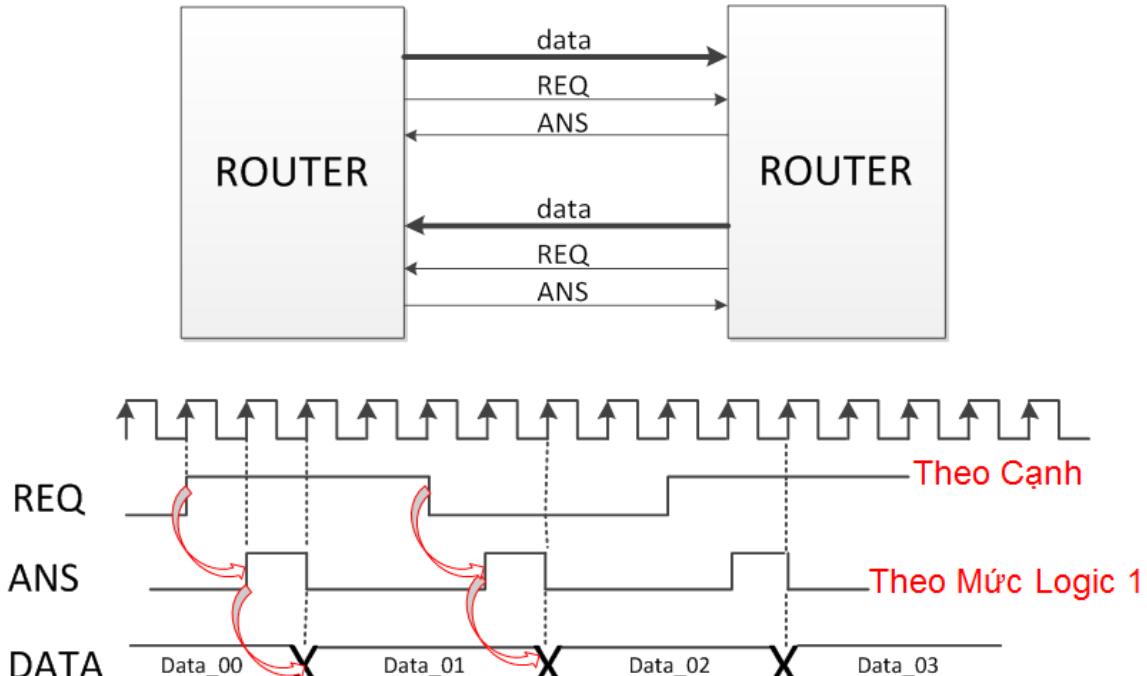
2.5.2 Giao thức REQ/ANS

Khảo sát giao thức **req/ans** cho một quá trình truyền nhận dữ liệu được thể hiện trong Hình 2.6.

Giao thức **req/ans** được sử dụng xuyên suốt trong quá trình giao tiếp giữa các router với nhau trong mạng NoC cũng như giao tiếp giữa các khôi bên trong router.

Liên kết truyền nhận giữa hai router (Liên kết giữa khôi **CtrlOutFifoOut** của router truyền với khôi **CtrlInFifoIn** của router nhận).

Liên kết giữa khôi **CtrlOutFifoIn** với các khôi điều khiển khác trong router.



Hình 2.6 Giao thức req/ans

Khi một khung dữ liệu cần truyền, khói điều khiển ngõ ra **CtrlOutFifoIn** phát đi tín hiệu **req** tại cạnh lên của xung clock. Ở khói điều khiển ngõ nhận, khi **CtrlInFifoOut** bắt được sự thay đổi của tín hiệu **req**, nếu đã sẵn sàng nhận dữ liệu, khói này truyền ngược lại tín hiệu **ans** tích cực cạnh lén sau cạnh lén xung clock kế đó. Khói điều khiển ngõ phát nhận tín hiệu phản hồi **ans** tích cực cạnh lén sẽ truyền dữ liệu đi ở cạnh lén xung clock tiếp theo.

Tại cạnh lén xung clock kế tiếp, một thiết lập truyền nhận kế tiếp được thực hiện bởi việc thay đổi mức logic của tín hiệu **req**. Cũng chính tại thời điểm này dữ liệu truyền ở thiết lập truyền nhận trước đó được ghi vào ở phía nhận dữ liệu.

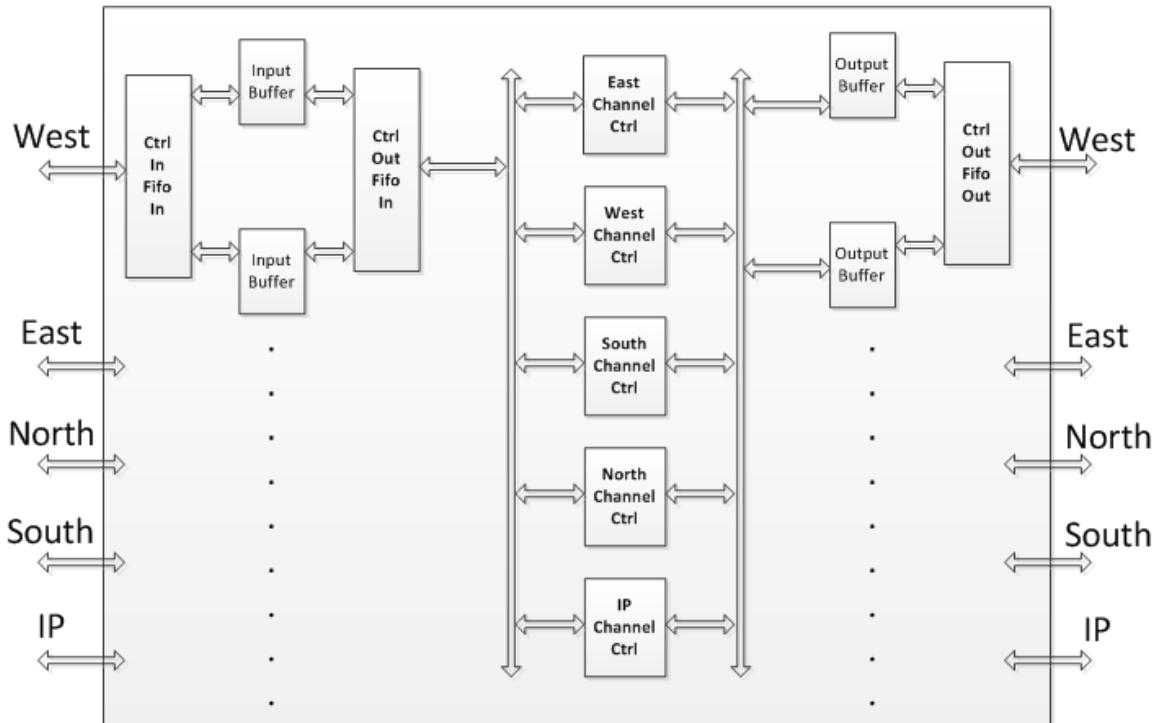
Như vậy có thể rút ra một số đặc điểm chính của giao thức như sau:

- Tín hiệu yêu cầu truyền dữ liệu tích cực theo cạnh (**lên/xuống**).
- Tín hiệu báo chấp nhận truyền dữ liệu tích cực theo mức (**mức 1**).
- Thời gian thiết lập, truyền và nhận dữ liệu chỉ tốn hai chu kỳ xung clock. Chính đặc tính này của giao thức đã góp phần làm giảm đáng kể thời gian trì hoãn giao tiếp bên trong và bên ngoài router.

2.5.3 Cấu trúc khối bên trong của router

Cấu trúc khối bên trong cho biết các thông tin tổng quan của router như sau:

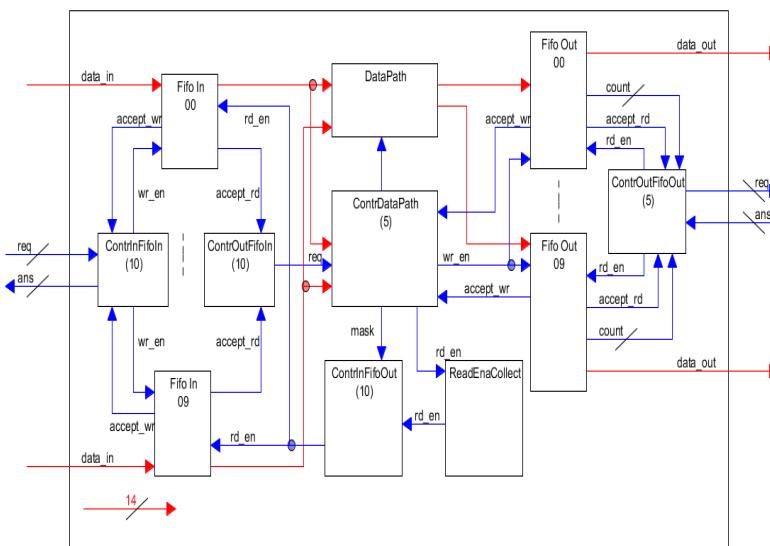
- Các đường dữ liệu ngõ vào (**16 bit**) được truyền từ các router liên kết đến các bộ đệm ngõ vào (**FifoIn**). Có 5 ngõ vào ở năm hướng khác nhau. Mỗi hướng hỗ trợ hai kênh ảo nên tổng cộng có 10 khối **FifoIn**.



Hình 2.7 Mô hình khói bên trong router

- Sau khi các gói dữ liệu ngõ vào được lưu trên bộ đệm ngõ vào, chúng sẽ được tiếp tục truyền đến khói **DataPath**. Tại đây, các hướng đi của gói dữ liệu được phân xử bởi các tín hiệu từ các khói **ChannelCtrl** nhằm giúp chúng đến đúng hướng và đến các bộ đệm ngõ ra một cách chính xác.
- Các tín hiệu điều khiển các đường dữ liệu ở ngõ vào/ra, ở bên trong router nhận được từ các khói **ChannelCtrl**. Khối này gửi các tín hiệu đến khói **DataPath** điều khiển các luồng dữ liệu đi đúng hướng đến các bộ đệm ngõ ra. Do có 5 ngõ ra ở 5 hướng khác nhau nên sẽ có 5 khói: **EastChannelCtrl**, **WestChannelCtrl**, **SouthChannelCtrl**, **NorthChannelCtrl**, **IPChannelCtrl**). Mỗi khói đảm nhiệm mỗi hướng.

- Khối **CtrlInFifoIn**: Khối này nhận các tín hiệu điều khiển **ans** và **req** từ các router liên kết nhằm điều khiển quá trình nhận luồng dữ liệu vào từ các router liên kết khác. Do có 10 khối **FifoIn** nên sẽ có tương ứng 10 khối **CtrlInFifoIn**.
- Khối **CtrlOutFifoIn**: Khối này điều khiển việc cho phép đọc ra các dữ liệu bên trong bộ đệm ngõ vào để truyền đến khối **DataPath**. Do có 10 khối **FifoIn** nên cũng sẽ có tương ứng 10 khối **CtrlOutFifoIn**.
- Khối **CtrlOutFifoOut**: Khối này nhận các tín hiệu điều khiển **ans** và **req** từ các router liên kết nhằm điều khiển quá trình xuất dữ liệu đến các router liên kết khác. Khác với các khối điều khiển khác, chỉ cần 5 khối **CtrlOutFifoOut** cho 5 hướng khác nhau.
- Các đường dữ liệu ngõ ra xuất phát từ các bộ đệm ngõ ra (**FifoOut**) được truyền đi đến ngõ vào của các router liên kết. Do có 5 ngõ ra ở 5 hướng khác nhau, mỗi hướng hỗ trợ hai kênh ảo nên tổng cộng có 10 khối **FifoOut**.



Hình 2.8 Kết nối chi tiết bên trong router

Phân tích Hình 2.8 cho thấy kết nối chi tiết các khối bên trong router như sau:

- Có tất cả 5 ngõ vào, 5 ngõ ra
- 10 khối **FifoIn**, 10 khối **ContrInFifoIn**, 10 khối **ContrOutFifoIn**.
- 10 khối **FifoOut**, 5 khối **ContrOutFifoOut**.
- 1 khối **DataPath**.
- 5 khối **ChannelCtrl**.

2.5.4 Cấu trúc dữ liệu

Như đã nói trên, với cơ chế Wormhole Routing, một gói dữ liệu (**packet**) được chia thành các khung dữ liệu (**flit**) được truyền từ router nguồn đến router đích.

Để hiểu được cấu trúc và sự cần thiết của các khung bên trong router, khung dữ liệu được phân tích chi tiết ở phần này.

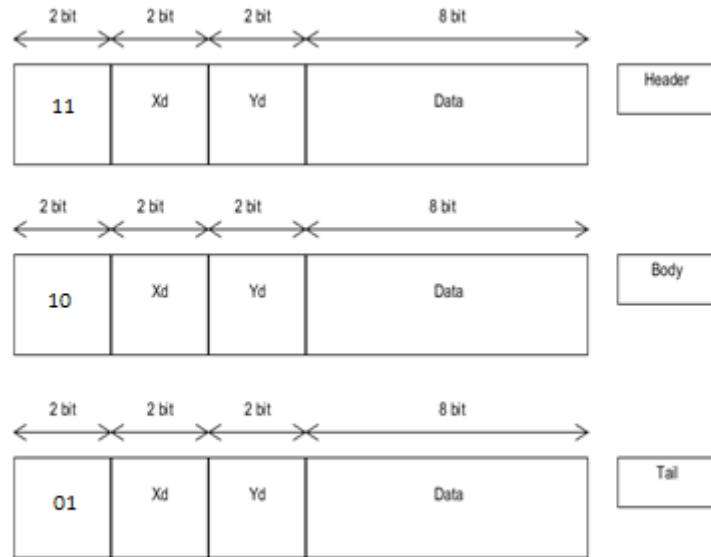
Để mở đầu và kết thúc một gói dữ liệu, khung dữ liệu được chia làm ba loại: Khung mở đầu (**header flit**), khung thân (**body flit**) và khung kết thúc (**tail flit**). Để phân biệt ba loại khung dữ liệu cần 2 bit mã hóa. Đầu tiên mã hóa khung mở đầu 11, khung thân mã hóa 10, khung kết thúc mã hóa 01.

Trên mỗi khung dữ liệu còn chứa thông tin chỉ vị trí của router đích mà nó mong muốn đến. Do mô hình mạng thiết kế là NoC 4x4 nên cần hai bit mã hóa vị trí tương ứng trực Ox và 2 bit mã hóa vị trí tương ứng trực Oy.

Ngoài các bit mã hóa nói trên, dữ liệu trên một khung được chọn là 8 bit. Số lượng bit này có thể thay đổi tùy mô hình mà không ảnh hưởng đến quá trình điều khiển luồng dữ liệu trên mô hình cố định NoC 4x4.

Như vậy một khung dữ liệu tương ứng tín hiệu dữ liệu bao gồm 14 bit:

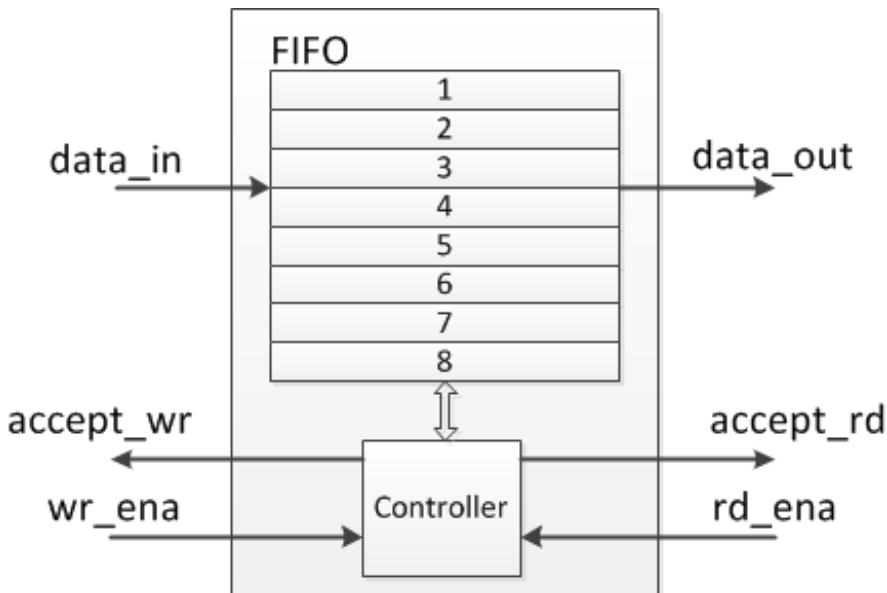
- 2 bit cho mã hóa loại khung dữ liệu.
- 4 bit cho mã hóa vị trí trên hai phương Ox và Oy.
- 8 bit cho dữ liệu truyền tải.



Hình 2.9 Cấu trúc khung dữ liệu

2.5.5 Cấu trúc và giao thức của khối Fifo

Cấu trúc của khối **FifoIn** và **FifoOut** giống nhau và được mô tả bởi Hình 2.10.



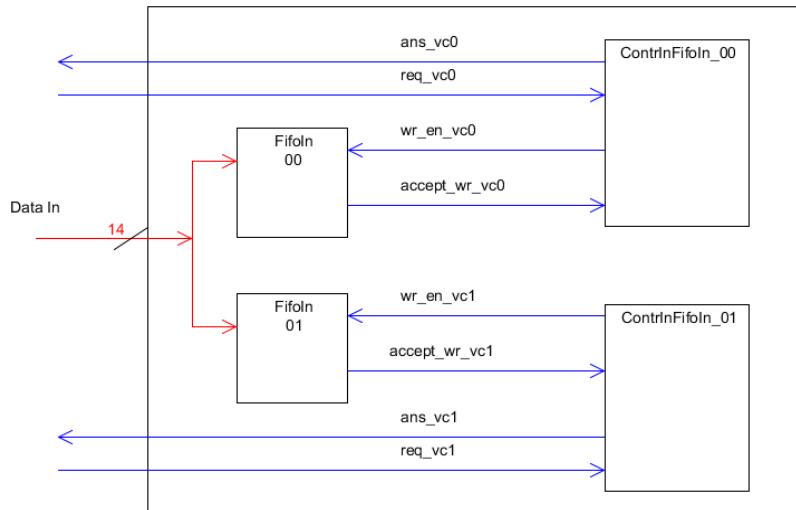
Chân **rst_n** là chân reset cho toàn bộ các router. Khi reset được tích cực (tích cực mức thấp), tất cả các dữ liệu trong khối **Fifo** và các ngõ ra được đưa về giá trị 0.

Chân **wr_en** cho phép dữ liệu **data_in** được ghi vào khối **Fifo**. Chân **rd_en** cho phép dữ liệu trong khối **Fifo** được đọc ra ở chân **data_out**. Chân **accept_rd** tích cực mức cao khi khối **Fifo** có chứa dữ liệu mới được ghi vào và ngược lại khi khối **Fifo** không chứa dữ liệu nào. Chân **accept_wr** tích cực mức cao khi **Fifo** vẫn còn chỗ trống để đưa dữ liệu vào và ngược lại khi khối **Fifo** đã chứa đầy dữ liệu.

Fifo hoạt động như một bộ đệm, các dữ liệu được ghi vào trước sẽ được đọc ra trước.

Khả năng lưu trữ của tất cả các bộ đệm trong thiết kế router được chọn lựa là 8 ô nhớ, mỗi ô nhớ chứa các bit dữ liệu. Việc chọn lựa số ô nhớ nhỏ giúp cải thiện đáng kể diện tích của router.

2.5.6 Cấu trúc và giao thức của khối **CtrlInFifoIn**



Hình 2.11 Giao diện khối **CtrlInFifoIn**

Khối **CtrlInFifoIn** như Hình 2.11 dùng để giao tiếp với khối **router** khác thông qua các tín hiệu điều khiển nhằm quyết định tín hiệu dữ liệu ngõ vào được nhận vào khối **FifoIn** hay không. Hình 2.12 mô tả giao thức hoạt động của khối **CtrlInFifoIn**.

Khi tín hiệu **req** từ các router liên kết tích cực (tích cực cạnh lén hoặc cạnh xuồng) ở cạnh lén của xung clock nhằm đề nghị truyền dữ liệu, nếu khối **FifoIn** còn trống (tín hiệu **accept_wr** tích cực mức cao) thì khối **FifoIn** cho phép nhận dữ liệu từ ngõ vào. Khi đó khối **CtrlInFifoIn** sẽ trả về tín hiệu **ans** tích cực mức cao và xuất ra ngoài đến router liên kết nhằm thông báo khối **FifoIn** đã sẵn sàng nhận dữ liệu.

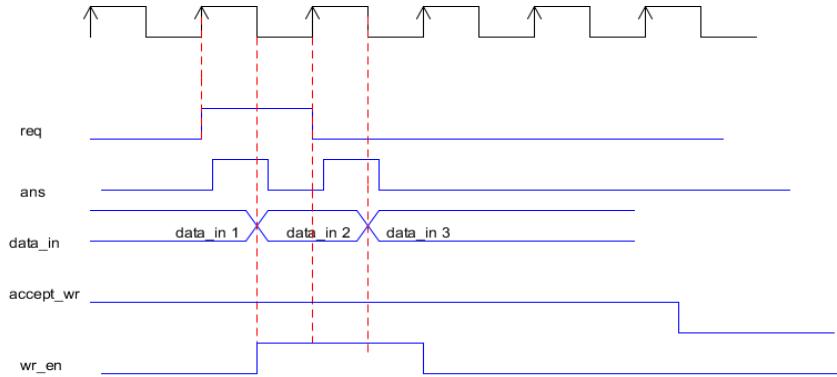
Ở cạnh lén xung clock tiếp theo đó, dữ liệu mới được truyền đến khối **FifoIn**. Cũng tại thời điểm này, tín hiệu **wr_en** được tích cực mức cao bởi khối **CtrlInFifoIn** gửi đến khối **FifoIn**.

Ở cạnh lén xung clock tiếp theo, dữ liệu mới được ghi vào khối **FifoIn**. Cũng tại cạnh lén xung clock này, tín hiệu **req** của router liên kết được tích cực nhằm thiết lập một chu kỳ truyền khung dữ liệu thứ hai.

Như vậy cơ chế truyền nhận được thực thi liên tục cho đến khi tất cả các khung dữ liệu của gói dữ liệu được truyền hết. Quá trình truyền nhận có thể bị gián đoạn do khối **FifoIn** đã đầy.

Trong trường hợp khối **FifoIn** đã đầy, tín hiệu **req** yêu cầu truyền dữ liệu được bỏ qua. Các router liên kết sẽ vẫn liên tục gửi các tín hiệu **req** ở xung clock kế tiếp.

Khi khối **FifoIn** đủ điều kiện để nhận dữ liệu, tín hiệu **ans** sẽ được tích cực tương ứng với tín hiệu **req** trước đó. Khi không có tín hiệu **req** tích cực, router hiểu dữ liệu không được truyền tiếp nữa.

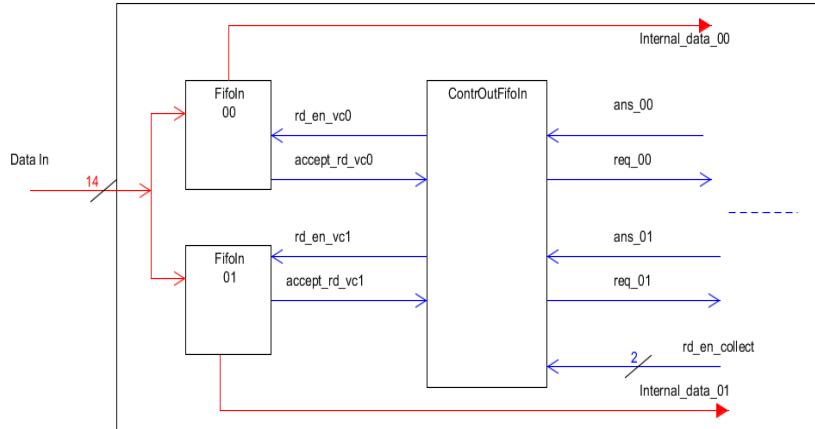


Hình 2.12 Giao thức khối **CtrlInFifoIn**

2.5.7 Cấu trúc và giao thức của khối **CtrlOutFifoIn**

Khối **CtrlOutFifoIn** đảm nhận vai trò giao tiếp của khối **FifoIn** với các khối khác bên trong router nhằm truyền dữ liệu ở ngõ vào đã lưu trong khối **FifoIn** đi đến các khối khác. Kiến trúc và giao thức của khối **CtrlOutFifoIn** được thể hiện qua Hình 2.13 và Hình 2.14.

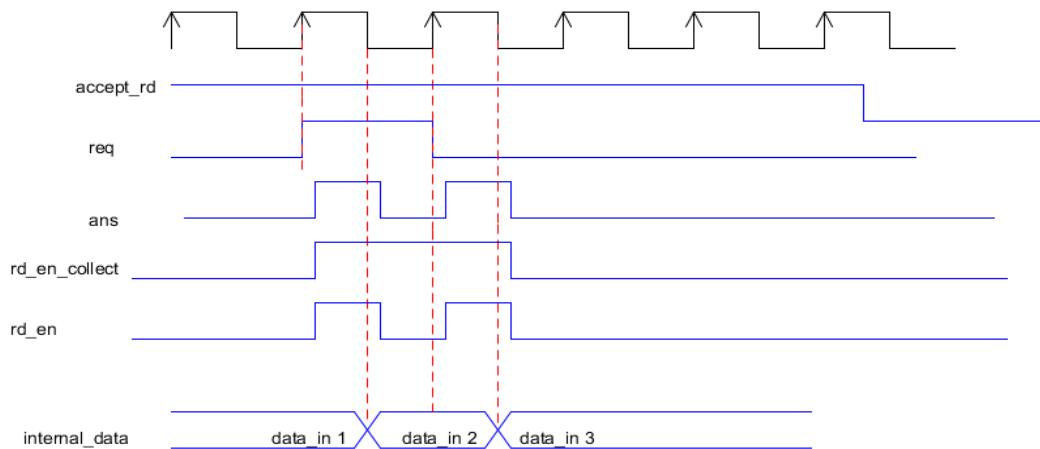
Khi khối **FifoIn** còn chứa dữ liệu chưa được đọc ra (tín hiệu **accept_rd** tích cực mức cao), khối **CtrlOutFifoIn** gửi liên tục các tín hiệu **req** ở các cạnh lên của xung clock. Khi tín hiệu **req** này được chấp nhận, tín hiệu **ans** sẽ được trả về ngay tức khắc cùng với tín hiệu **rd_en_collect**.



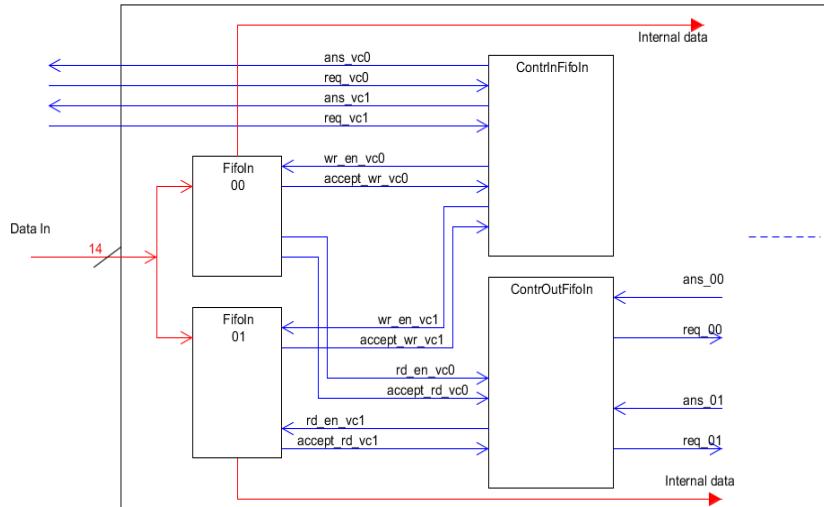
Hình 2.13 Giao diện khối **CtrlOutFifoIn**

Khi nhận được tín hiệu **rd_en_collect**, khối **CtrlOutFifoIn** sẽ truyền tín hiệu **rd_en** cho khối **FifoIn** để cho phép khối này xuất dữ liệu ở cạnh xuống xung clock kế đó.

Khi khối **FifoIn** vẫn còn dữ liệu chưa được đọc, tín hiệu **req** được thay đổi liên tục ở cạnh lên xung clock và quá trình trên lại tiếp diễn liên tục cho đến khi nào khối **FifoIn** không còn dữ liệu (tín hiệu **accept_rd** ở mức thấp).



Hình 2.14 Giao thức khối **CtrlOutFifoIn**

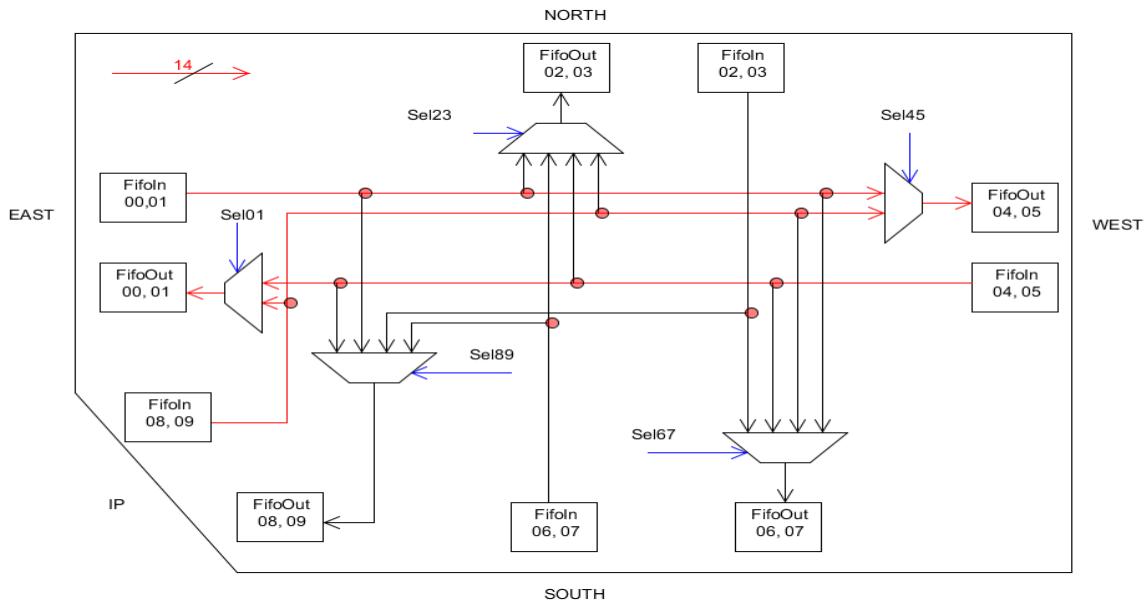


Hình 2.15 Giao diện khói CtrlInFifoIn và CtrlOutFifoIn

Hình 2.15 mô tả liên kết các khói **CtrlInFifoIn**, **CtrlOutFifoIn** và **Fifo** nhằm giúp bạn đọc dễ hình dung giao thức truyền nhận của khói **FifoIn** với router liên kết và với các khói bên trong. Có thể tóm gọn các chi tiết sau:

- Dữ liệu được lưu vào các bộ đệm ở cạnh lén xung clock khi tín hiệu **wr_en** tích cực.
- Dữ liệu được đọc ra ở cạnh lén xung clock khi tín hiệu **rd_en** tích cực.
- Các bộ đệm (**FifoIn**) có các tín hiệu cho phép đọc (**accept_rd**) và tín hiệu cho phép ghi (**accept_wr**) giúp các bộ điều khiển **CtrlInFifoIn** và **CtrlOutFifoIn** biết được trạng thái của bộ đệm mà đưa ra các quyết định chính xác.
- Mỗi một khói **FifoIn** tương ứng một cặp khói điều khiển ghi (**CtrlInFifoIn**) và đọc (**CtrlOutFifoIn**).
- Khối **CtrlInFifoIn** giao tiếp với các ngõ vào/ra của router liên kết. Khối **CtrlOutFifoIn** giao tiếp với các khói bên trong của router.

2.5.8 Cấu trúc và giao thức của khối DataPath



Hình 2.16 Đường truyền dữ liệu bên trong router

Khối **DataPath** điều khiển các đường dữ liệu từ khói **FifoIn** đi đến đúng khói **FifoOut** mong muốn.

Khối **DataPath** chủ yếu bao gồm các bộ mux phân định các luồng dữ liệu. Các tín hiệu điều khiển chọn lựa của bộ mux được nhận từ các khói **ChannelCtrl**.

Theo như Hình 2.16, mỗi router có 5 ngõ ra tương ứng với 5 bộ **mux**, tín hiệu ngõ ra của mỗi bộ **mux** sẽ được truyền đến hai khói **FifoOut** ở mỗi ngõ ra tương ứng hai kênh ảo. Tại một thời điểm, chỉ có một khói **FifoOut** được ghi.

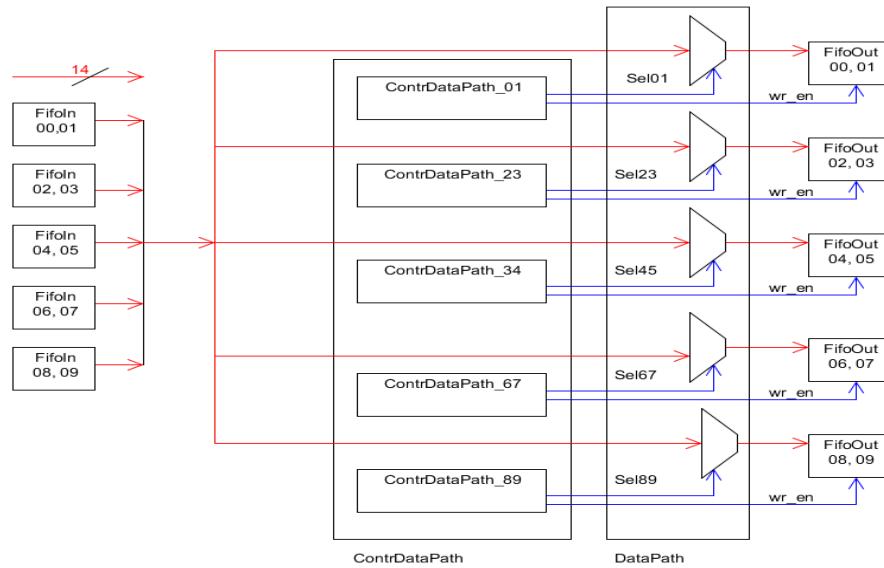
Do đặc điểm của giải thuật định tuyến XY, các gói dữ liệu sẽ được truyền theo phương X trước. Do đó, các gói dữ liệu ở ngõ vào hướng bắc và nam (phương Y) sẽ không được đưa tới các bộ **mux** cho các ngõ ra ở hướng đông và tây (phương X) và tất cả các dữ liệu từ hướng đông, hướng tây (phương X) sẽ được truyền đến tất cả các hướng.

Đối với ngõ vào/ra IP, các dữ liệu của **FifoIn** sẽ được truyền đi đến tất cả các hướng còn lại.

Ngược lại dữ liệu của khói **FifoIn** từ tất cả các hướng sẽ được truyền tới bộ **mux** của hướng IP.

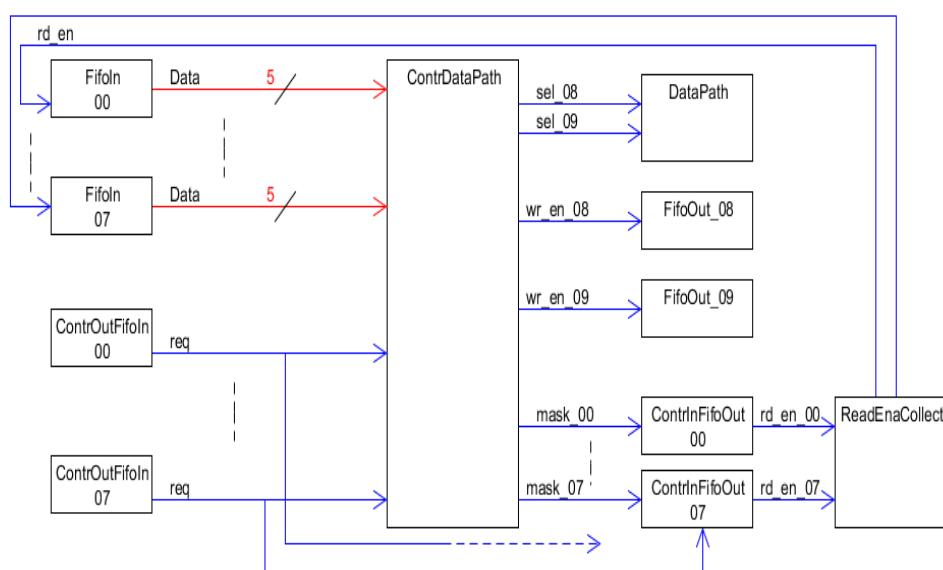
Với các đặc tính trên, các bộ **mux** cho hướng đông và hướng tây là các bộ **mux** hai sang một.

Các bộ **mux** cho hướng nam, hướng bắc và hướng IP là các bộ mux bốn sang một. Hình 2.17 làm rõ mối quan hệ khối **DataPath** và các khối **ChannelCtrl**.



Hình 2.17 Giao diện khối DataPath và các khối ChannelCtrl

2.5.9 Cấu trúc và giao thức của khối ChannelCtrl



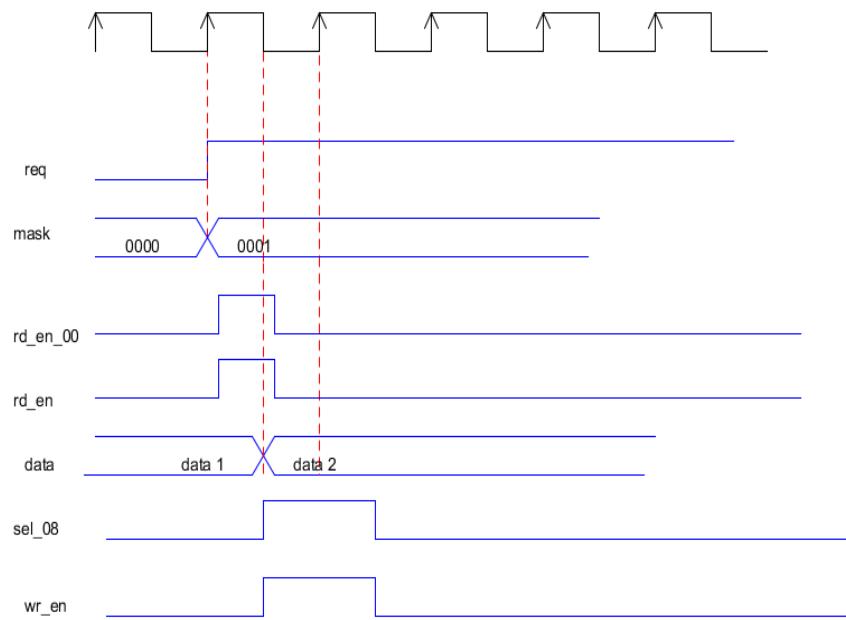
Hình 2.18 Giao diện khối ChannelCtrl

Việc phân xử các hướng là độc lập, do đó cần 5 khối **ChannelCtrl** cho 5 hướng khác nhau. Điều này giúp tránh các trường hợp xung đột khi các luồng dữ liệu đi ra các hướng khác nhau. Mô hình cấu trúc của các khối **ChannelCtrl** là gần như nhau cho tất cả các hướng.

Kiến trúc khối **ChannelCtrl** trong Hình 2.18 đảm nhận nhiều chức năng điều khiển của router được nêu ra như sau:

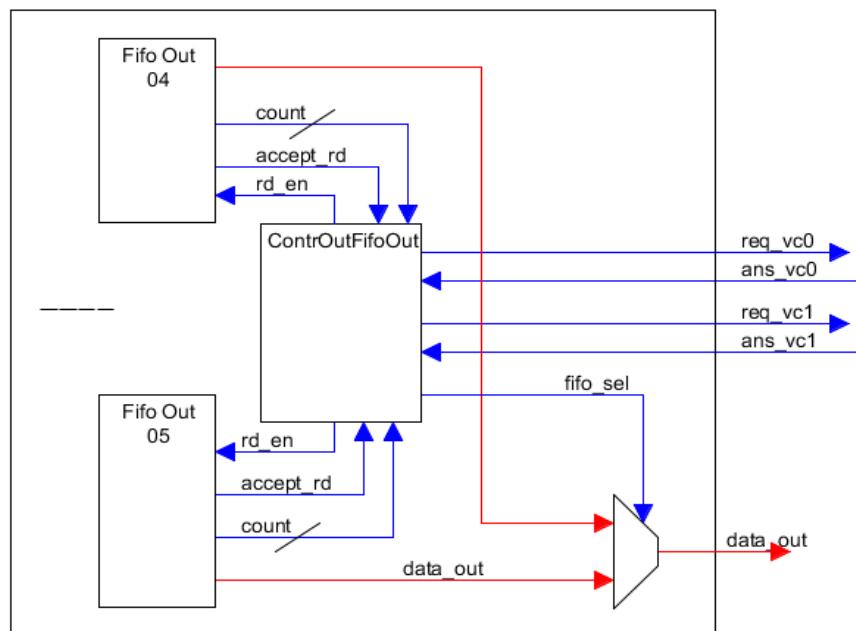
- Nhận trực tiếp một phần dữ liệu từ ngõ vào, dữ liệu này giúp khối **ChannelCtrl** hiểu được đây là khung dữ liệu mở đầu, thân hay cuối, phương tiện truyền của khung dữ liệu (đông, tây, nam, bắc hay IP).
- Nhận các tín hiệu yêu cầu **req** của các khối **CtrlOutFifoIn**. Các tín hiệu này giúp biết được yêu cầu được đề nghị từ bộ đệm ngõ vào nào, từ đó sẽ truyền tín hiệu **ans** nhằm điều khiển quá trình đọc dữ liệu của khối **FifoIn**.
- Xuất ra các tín hiệu cho phép ghi **wr_en** đến các khối **FifoOut** nhằm kiểm soát việc ghi dữ liệu lấy ra từ bộ đệm ngõ vào (**FifoIn**) lên bộ đệm ngõ ra (**FifoOut**).
- Xuất các tín hiệu chọn lựa **channel_sel** đến các bộ **mux** của khối **DataPath** nhằm chọn ra dữ liệu mong muốn.
- Điều khiển quá trình tránh lỗi (sẽ đề cập chi tiết ở phần sau).

Hình 2.19 mô tả tuần tự tín hiệu ngõ ra của khối **ChannelCtrl** được xuất ra.



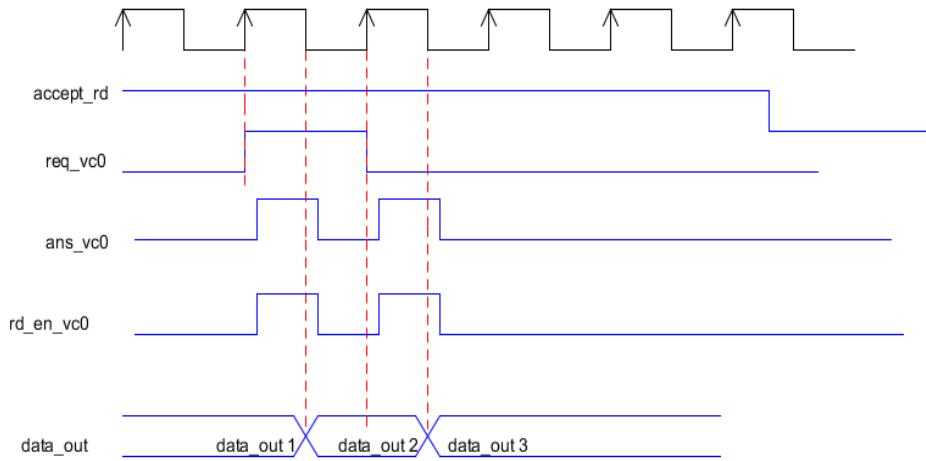
Hình 2.19 Giao thức khói ChannelCtrl

2.5.10 Cấu trúc và giao thức của khói CtrlOutFifoOut



Hình 2.20 Giao diện khói CtrlOutFifoOut

Khối **CtrlOutFifoOut** có nhiệm vụ tạo liên kết truyền nhận với các router liên kết để truyền đi các dữ liệu trên **FifoOut**. Đầu tiên dựa vào các tín hiệu **accept_rd** từ các khối **FifoOut**, khối **CtrlOutFifoOut** biết được bộ đệm ngõ ra (**FifoOut**) có chứa dữ liệu cần truyền hay không.



Hình 2.21 Giao thức khối **CtrlOutFifoOut**

Nếu bộ đệm ngõ ra cần truyền dữ liệu đến các router liên kết (tín hiệu **accept_rd** tích cực mức cao), khối **CtrlOutFifoOut** truyền tín hiệu **req** tương ứng của bộ đệm đó đến router liên kết tại cạnh lên của xung clock.

Khi nhận được tín hiệu phản hồi **ans** từ router liên kết ngay sau đó, khối **CtrlOutFifoOut** sẽ xuất tín hiệu **rd_en** đến bộ đệm ngõ ra. Dữ liệu từ bộ đệm ngõ ra sẽ được xuất ra ở cạnh lên xung clock kế đó.

Ngoài nhiệm vụ liên lạc và truyền nhận dữ liệu giữa hai router, khối **CtrlOutFifoOut** còn nhận vai trò phân xử dữ liệu từ hai bộ đệm ngõ ra trên cùng một hướng. Tại một thời điểm chỉ một dữ liệu từ một bộ đệm ngõ ra được truyền. Do ứng dụng kỹ thuật kênh ảo, dữ liệu từ hai bộ đệm ngõ ra được truy xuất xen kẽ dựa vào số lượng dữ liệu nằm trong bộ đệm nhiều hay ít.

Chính vì một khối **CtrlOutFifoOut** điều khiển việc đọc dữ liệu từ hai bộ đệm ngõ ra trên cùng một hướng nên chỉ cần 5 khối **CtrlOutFifoOut** trong một router.

2.5.11 Đánh giá cấu trúc và giao thức truyền nhận của router

Dựa vào kiến trúc và giao thức mà router được thiết kế như đã thảo luận ở trên, một số đặc điểm nổi bật được đúc kết:

- Giao thức **ans/req** đơn giản nhằm giảm thiểu tối đa quá trình liên kết giữa các router và quá trình truyền dữ liệu bên trong router.
- Kiến trúc **Fifo** được thiết kế phù hợp với việc sử dụng giao thức **ans/req**.
- Một phần dữ liệu được truyền trực tiếp đến các khối **ChannelCtrl** nhằm phân tích và định hướng cho gói dữ liệu mà không thông qua quá trình đọc của các khối **FifoIn**. Điều này giúp quá trình định tuyến các luồng dữ liệu được thực hiện sớm hơn khi dữ liệu chưa ghi vào bộ đệm ngõ vào (**FifoIn**).
- Router không thiết kế các vùng nhớ trung gian để lưu trữ dữ liệu, các dữ liệu chỉ được lưu trữ trên các **Fifo** ở ngõ vào và ngõ ra. Kiến trúc này giúp làm giảm việc sử dụng tài nguyên và rút gọn thời gian xử lý dữ liệu.

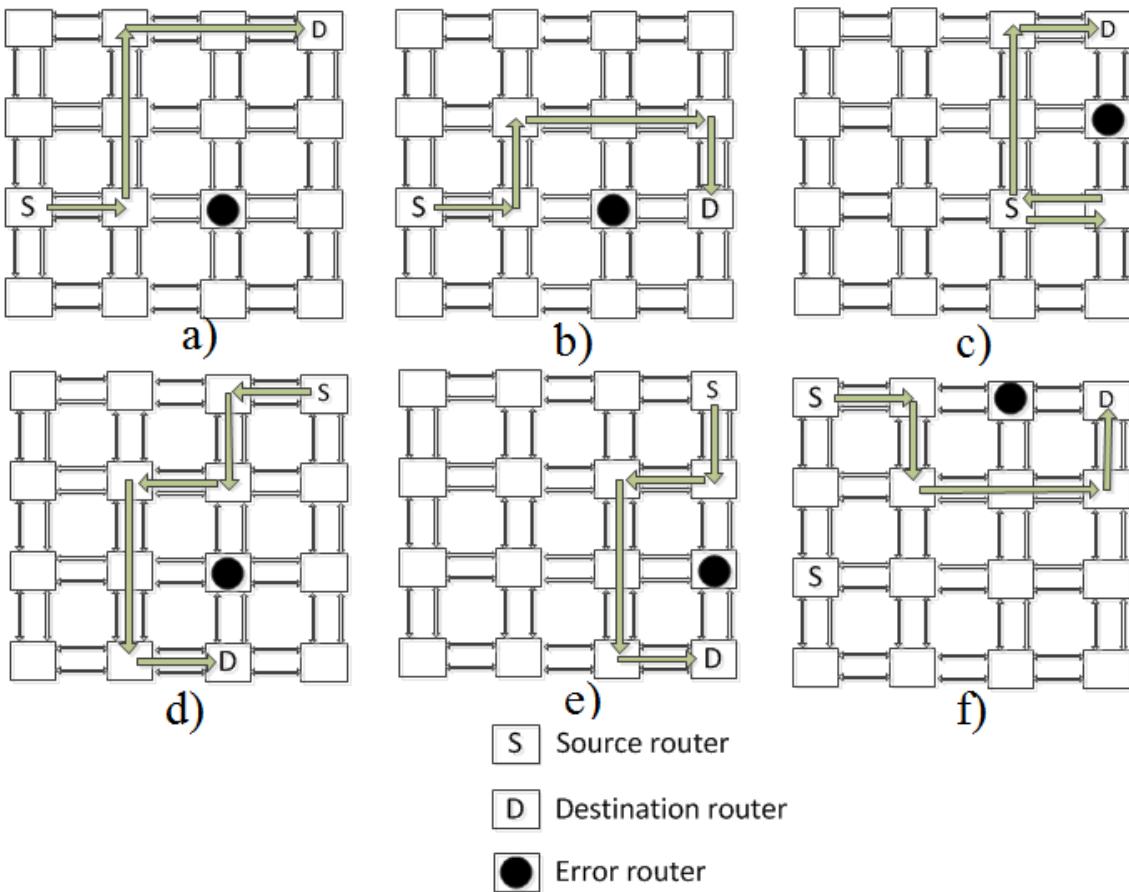
2.6 Khả năng và kiến trúc NoC tự sửa một lỗi sau khi sản xuất

Trong quá trình sản xuất, xác suất sai sót trong công nghệ sản xuất vì lý do cơ khí, bụi hay bất cứ một nguyên nhân nào làm giảm hiệu năng của quá trình sản xuất. Đối với bất cứ mạng NoC nào, việc sản xuất lỗi trên bất cứ một router nào cũng gây ảnh hưởng nghiêm trọng đến hiệu năng của toàn mạch.

Mô hình NoC tiếp cận dạng mạng hai chiều và các gói dữ liệu truyền theo quy tắc Oxy với việc ưu tiên truyền trên trục Ox trước, do đó nếu có một lỗi bất kỳ xảy ra, một số đường truyền nhất định sẽ không thể thực hiện được. Việc khắc phục lỗi sau khi sản xuất có thể được thực hiện bởi hai cách thức sau:

- Sửa bằng phần mềm, có nghĩa là đường truyền mà đi qua router lỗi sẽ bị cấm trong quá trình lập trình.
- Sửa bằng phần cứng, có nghĩa là đường gói dữ liệu có thể tránh được router lỗi.

Trong đề tài, kiến trúc NoC tiếp cận với các router đơn có thể tránh được một lỗi bất kỳ trên mạng NoC. Các hình vẽ minh họa sau mô tả các trường hợp gói dữ liệu đi và né tránh các router lỗi.



Hình 2.22 Các trường hợp tránh 1 lỗi

Hình 2.22 (a) và Hình 2.22 (b) mô tả trường hợp lỗi trên quá trình truyền theo phương Ox. Tuy nhiên đối với trường hợp Hình 2.22 (a), độ ưu tiên trực truyền bị thay đổi. Cụ thể đường truyền ưu tiên phương Oy trước rồi mới truyền Ox sau. Tuy nhiên với Hình 2.22 (b), trực Ox vẫn được ưu tiên sau khi gói dữ liệu né được router lỗi.

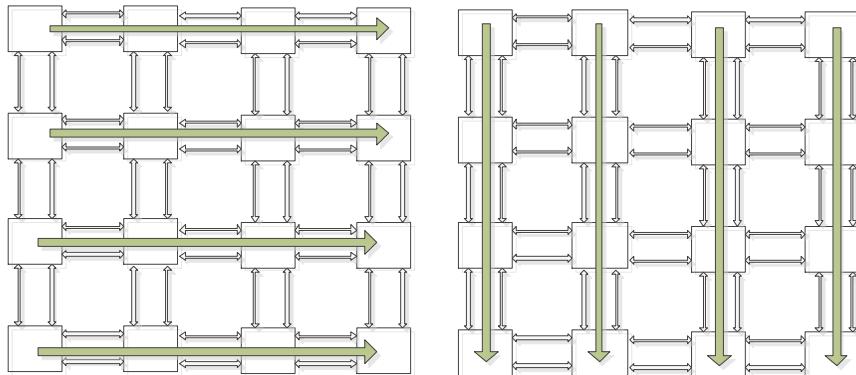
Hình 2.22 (d) và Hình 2.22 (e) mô tả trường hợp lỗi trên phương Oy, cũng tương tự như phân tích lỗi trên phương Ox sẽ có hai trường hợp xảy ra về việc thay đổi độ ưu tiên của gói dữ liệu theo phương Ox hoặc giữ nguyên phương Oy như trước đó.

Trường hợp còn lại được đề cập là các lỗi xảy ra ở các router biên như Hình 2.22 (c) và Hình 2.22 (f). Trong đó, Hình 2.22 (c) cho thấy sự khác biệt vì phải truyền ngược về router nguồn (Source router). Sau đó, hướng truyền sẽ bị thay đổi để có thể vừa né được router lỗi và đến được router đích.

Dựa vào tất cả các trường hợp lỗi có thể xảy ra như trên có thể kết luận rằng nếu một router có thể biết được router kế cận bị lỗi và có thể thay đổi được độ ưu tiên phương truyền thì có thể sẽ né được tất cả các trường hợp 1 lỗi bất kỳ.

2.6.1 Phương thức xác định lỗi trên mô hình NoC hai chiều

Đối với mô hình NoC tiếp cận, việc xác định lỗi sau khi sản xuất có thể được thực hiện bằng loạt gửi các gói dữ liệu giả lập theo chiều dọc và theo chiều ngang như hình 2.23.



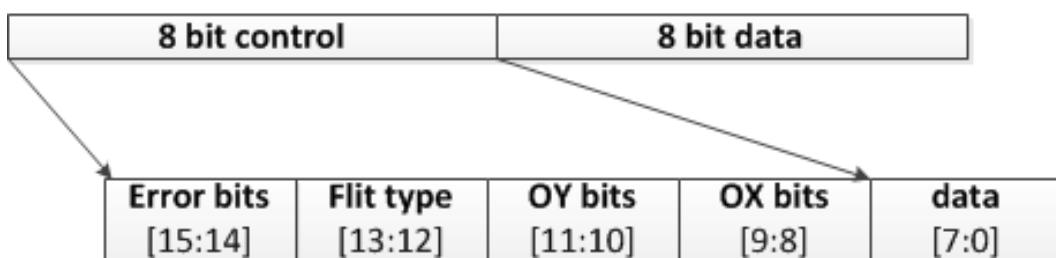
Hình 2.23 Phương thức xác định router lỗi

Việc đánh dấu các gói dữ liệu truyền đi thành công và không thành công khi truyền lần lượt qua các router và từng router kế cận nhau cho biết router lỗi một cách dễ dàng. Tuỳ thuộc vào vị trí router lỗi, nhóm bit xác định router lỗi sẽ được tích cực. Trong đề tài này mô hình NoC 4x4 được tiếp cận, do đó cần 4 bit xác định vị trí router lỗi với 2 bit cho phương Ox và 2 bit cho phương Oy.

4 bit này sẽ được thiết lập sau khi các gói dữ liệu được gửi và giữ cố định trong suốt quá trình hoạt động của mạng sau đó. Như vậy 4 bit này sẽ truyền vào tất cả các router trên mạng, do đó bất cứ router nào cũng có thể biết được có hay không router kế cận bị lỗi mà từ đó chuyển hướng đi cần thiết với gói dữ liệu.

2.6.2 Phương thức tránh lỗi

Để có thể tránh lỗi, khói điều khiển cần được thêm vào những logic cần thiết. Để làm được điều này, khung dữ liệu sẽ bị thay đổi như sau.



Hình 2.24 Cấu trúc khung dữ liệu cho việc tránh lỗi

- Dữ liệu 8 bit được giữ nguyên
- 4 bit định vị trí Ox và Oy của router đích
- 2 bit mô tả loại khung dữ liệu
- 2 bit báo lỗi

Như vậy khung dữ liệu chỉ thêm 2 bit lỗi với ý nghĩa được miêu tả như sau:

- 01: Giá trị ban đầu
- 10: Giá trị báo cho router biết khung này vừa chuyển hướng tránh router lỗi nhưng sẽ không thay đổi phương truyền
- 11: Giá trị báo cho router biết khung này vừa chuyển hướng tránh router lỗi và yêu cầu đổi phương truyền

Với việc thêm 2 bit lỗi thông tin này vào gói dữ liệu, mô hình NoC có những thuận lợi sau:

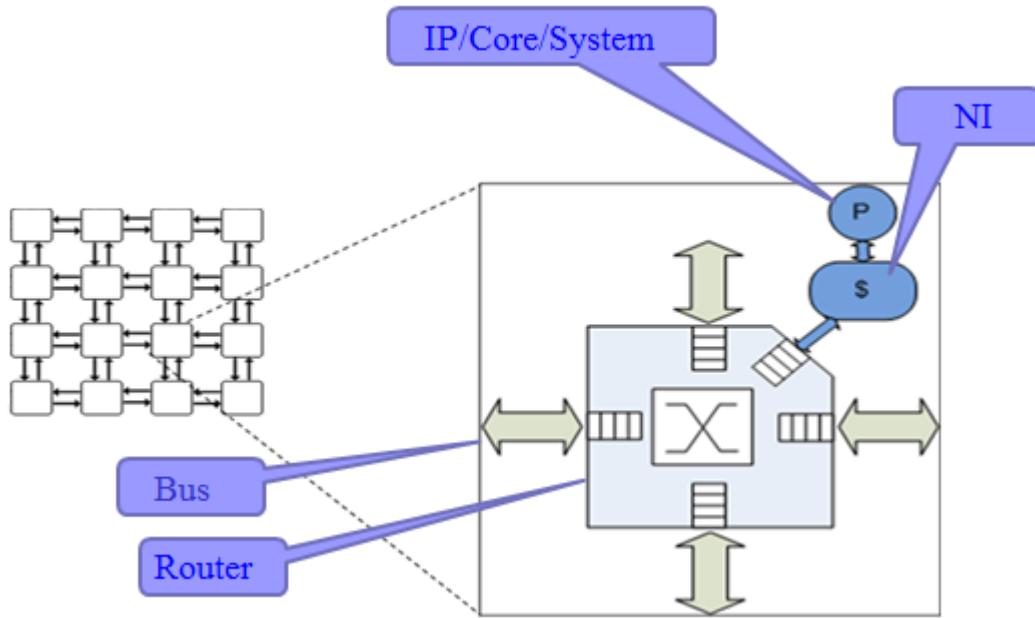
- Router nhận khung dữ liệu không quan tâm đến nội dung của khung, do đó việc điều khiển ngõ vào được đơn giản.
- Quá trình xử lý khung dữ liệu bên trong router khi hướng truyền mong muốn bị lỗi dựa trên 4 bit thông tin lỗi và 2 bit lỗi trong khung dữ liệu. Điều đó có nghĩa là ở mỗi router, việc tránh lỗi sẽ được xử lý độc lập mà không cần dựa trên các thông tin trao đổi qua lại giữa các router với nhau giảm tải quá trình điều khiển.
- Đối với các khung dữ liệu đã được đánh dấu 10 hay 11, 4 bit thông tin lỗi là 0 sẽ thông báo cho router biết các đường truyền kế tiếp sẽ không thể có lỗi vì bit thông tin thông báo không lỗi ($4'b0000$) hoặc lỗi đã được tránh trước đó. Từ đó, việc truyền dữ liệu sẽ được thực thi nhanh hơn vì không cần điều khiển tránh lỗi nữa.

Với những đặc tính thiết kế này, kiến trúc router sẽ đạt được yêu cầu như đề tài đề ra trước đó. Để đánh giá hoạt động và độ chính xác cũng như hiệu năng của kiến trúc router, chương 3 sẽ được giới thiệu.

2.7 Giao diện mạng (Network Interface - NI)

Sự xuất hiện của mạng trên chip (NoC) là sự phát triển của việc truyền thông tin cho hệ thống trên chip (SoC) dựa trên giao diện chuẩn cho việc tích hợp của IP với các yêu cầu truyền thông đa dạng. Các giao diện này phải đơn giản và đáp ứng nhanh. Luận văn trình bày

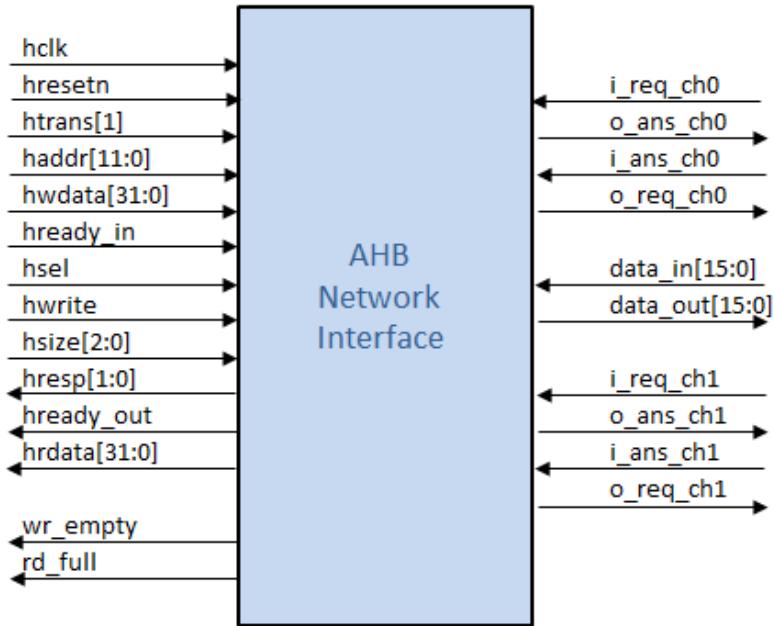
thiết kế giao diện mạng dựa trên giao thức AHB được tích hợp với kiến trúc hiện tại để sử dụng với lõi ARM (ARM core).



Hình 2.25 Sơ đồ kết nối của NI trong NoC

2.7.1 Mô hình kiến trúc bên ngoài NI

Mô hình bên ngoài của NI được mô tả như trong Hình 2.26. Mỗi NI gồm các tín hiệu AHB interface để liên kết với CPU và bus controller, ngoài ra còn có hai tín hiệu interrupt gửi đến CPU để thông báo khi nào cần đọc hoặc ghi dữ liệu. Mặt khác, mỗi NI bao gồm các tín hiệu **req/ans** và **data** để truyền hoặc nhận dữ liệu với router. Mô tả chi tiết các tín hiệu được thực hiện ở phần sau.

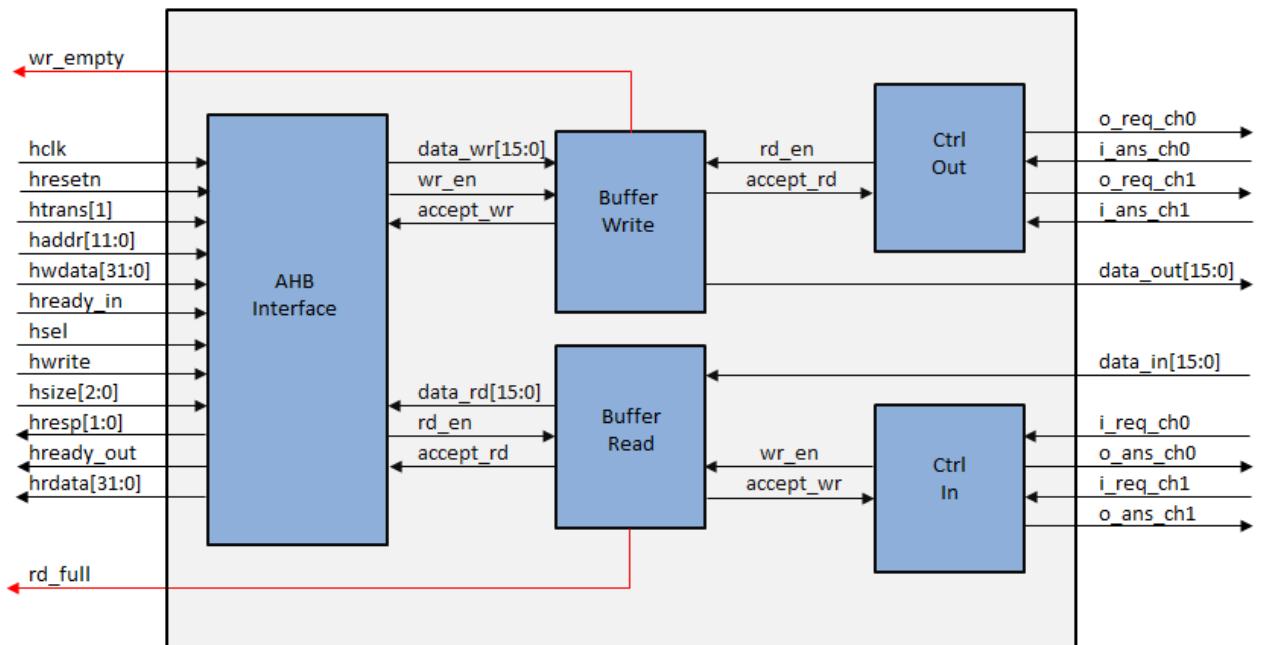


Hình 2.26 Kiến trúc bên ngoài của NI

2.7.2 Mô hình kiến trúc bên trong NI

a/ Cấu trúc khối bên trong NI

Mô hình bên trong của NI được mô tả như trong Hình 2.27.



Hình 2.27 Mô hình khối bên trong của NI

Cấu trúc khói bên trong cho biết các thông tin tổng quan của NI như sau:

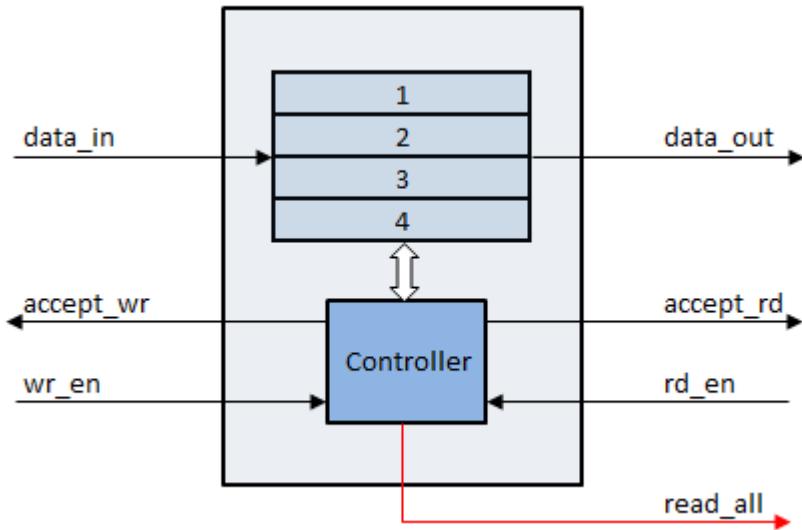
- NI gồm hai bộ đệm hỗ trợ cho việc đọc dữ liệu **BufferRead** và ghi dữ liệu **BufferWrite** từ CPU đến router.
- Khối **AHBInterface**: Khối này nhận các tín hiệu điều khiển từ CPU để thiết lập quá trình ghi dữ liệu vào bộ đệm **BufferWrite** hoặc đọc dữ liệu từ bộ đệm **BufferRead**.
- Khối **CtrlOut**: Khối này nhận các tín hiệu điều khiển **ans** và **req** từ router liên kết nhằm điều khiển quá trình xuất dữ liệu đến router liên kết.
- Khối **CtrlIn**: Khối này nhận các tín hiệu điều khiển **ans** và **req** từ router liên kết nhằm điều khiển quá trình nhận luồng dữ liệu vào từ router liên kết.

NI gồm hai hoạt động chính:

- Khi CPU bắt đầu một hoạt động ghi, các tín hiệu điều khiển được gửi đến khói **AHBInterface**. Khối này sẽ xác nhận yêu cầu ghi, đồng thời gửi dữ liệu ghi **hwdata** đến bộ đệm **BufferWrite**. Sau đó khói **CtrlOut** sẽ điều khiển quá trình xuất dữ liệu từ **BufferWrite** đến router liên kết. Khi dữ liệu từ **BufferWrite** đã được đọc ra, nó sẽ gửi tín hiệu interrupt **wr_empty** đến CPU để thông báo sẵn sàng cho hoạt động ghi kế tiếp.
- Khi có dữ liệu từ router liên kết gửi đến, khói **CtrlIn** sẽ điều khiển quá trình nhận luồng dữ liệu vào **BufferRead**. Khi bộ đệm **BufferRead** đã đầy dữ liệu, nó sẽ gửi tín hiệu interrupt **rd_full** đến CPU để thông báo sẵn sàng cho hoạt động đọc. CPU bắt đầu một hoạt động đọc bằng cách gửi các tín hiệu điều khiển đến khói **AHBInterface**. Khối này sẽ xác nhận yêu cầu đọc, đồng thời điều khiển quá trình xuất dữ liệu từ **BufferRead** đến CPU thông qua dữ liệu đọc **hrdata**.

b/ Cấu trúc và giao thức của khói Buffer

Cấu trúc của khói **BufferRead** và **BufferWrite** giống nhau và được mô tả bởi Hình 2.28.



Hình 2.28 Giao diện khối Buffer

Chân **rst_n** là chân reset cho toàn bộ các router. Khi reset được tích cực (tích cực mức thấp), tất cả các dữ liệu trong khối **Buffer** và các ngõ ra được đưa về giá trị 0.

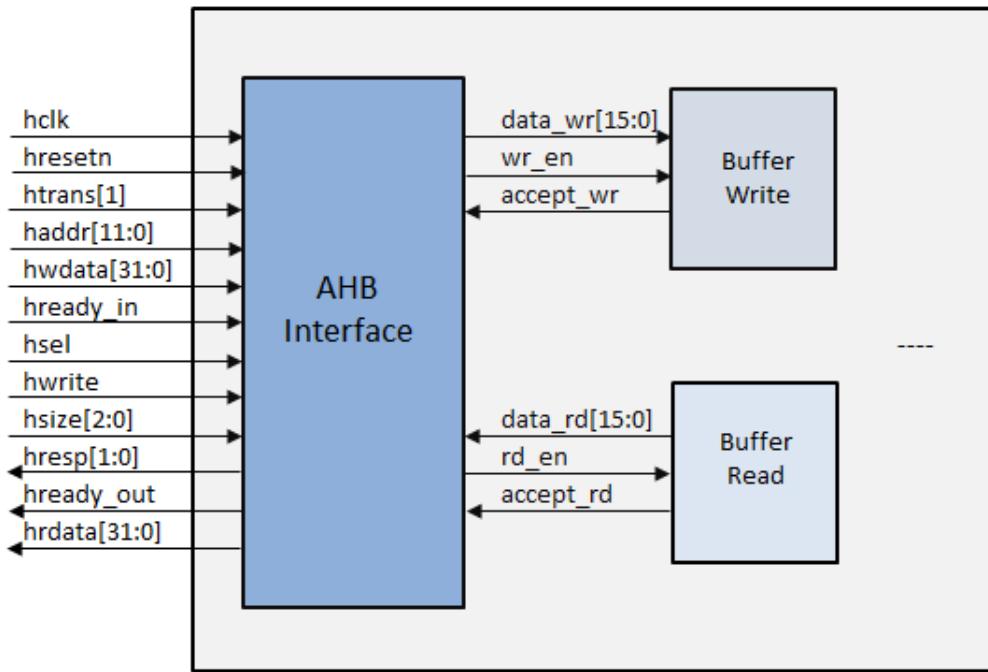
Chân **wr_en** cho phép dữ liệu **data_in** được ghi vào khối **Buffer**. Chân **rd_en** cho phép dữ liệu trong khối **Buffer** được đọc ra ở chân **data_out**. Chân **accept_rd** tích cực mức cao khi khối **Buffer** có chứa dữ liệu mới được ghi vào và ngược lại khi khối **Buffer** không chứa dữ liệu nào. Chân **accept_wr** tích cực mức cao khi **Buffer** vẫn còn chỗ trống để đưa dữ liệu vào và ngược lại khi khối **Buffer** đã chứa đầy dữ liệu.

Chân **read_all** thông báo khi dữ liệu trong **Buffer** đã được đọc xong, đóng vai trò là interrupt gửi đến CPU. Đối với **BufferWrite**, tín hiệu **wr_empty** thông báo sẵn sàng cho hoạt động ghi kế tiếp. Đối với **BufferRead**, tín hiệu **rd_full** thông báo sẵn sàng cho hoạt động đọc.

Buffer hoạt động như một bộ đệm, các dữ liệu được ghi vào trước sẽ được đọc ra trước.

Khả năng lưu trữ của các bộ đệm trong thiết kế NI được chọn lựa là 4 ô nhớ, tương ứng với 4 khung dữ liệu: khung mở đầu (**header flit**), 2 khung thân (**body flit**) và khung kết thúc (**tail flit**). Việc đóng gói dữ liệu theo một khung cho trước được thực hiện bởi khối **AHBInterface**.

c/ Cấu trúc và giao thức của khối AHBInterface



Hình 2.29 Giao diện khối AHBInterface

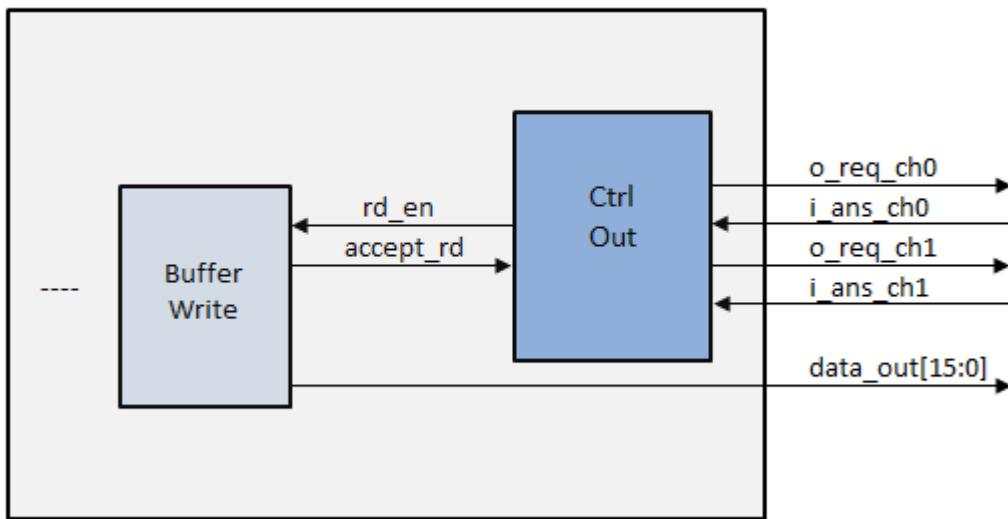
Khối **AHBInterface** như Hình 2.29 dùng để giao tiếp với CPU (ARM core) thông qua các tín hiệu điều khiển nhằm thiết lập quá trình ghi dữ liệu vào bộ đệm **BufferWrite** hoặc đọc dữ liệu từ bộ đệm **BufferRead**.

Khi CPU bắt đầu một hoạt động ghi, khối **AHBInterface** sẽ xác nhận yêu cầu ghi từ các tín hiệu điều khiển, sau đó thực hiện việc đóng gói dữ liệu theo một khung cho trước và điều khiển ghi từng khung dữ liệu vào bộ đệm **BufferWrite**. Với 32 bit dữ liệu từ **hwdata** sẽ được chia thành 4 khung, mỗi khung gồm 8 bit dữ liệu. Đồng thời CPU cũng sử dụng 4 bit thấp của địa chỉ **haddr** để chỉ vị trí của router đích. Mặt khác, đây là gói dữ liệu cho router nguồn và chưa tránh lỗi nên sẽ gán **error_bit** là **2'b0**. Do đó, ta có cấu trúc 16 bit của 4 khung dữ liệu như sau:

- Khung mở đầu: {**2'b0, 2'b11, haddr[3:2], haddr[1:0], hwdata[7:0]**}
- Khung thân thứ nhất: {**2'b0, 2'b10, haddr[3:2], haddr[1:0], hwdata[15:8]**}
- Khung thân thứ hai: {**2'b0, 2'b10, haddr[3:2], haddr[1:0], hwdata[23:16]**}
- Khung kết thúc: {**2'b0, 2'b01, haddr[3:2], haddr[1:0], hwdata[31:24]**}

Khi có dữ liệu từ router liên kết gửi đến bộ đệm **BufferRead**, nó sẽ gửi tín hiệu interrupt **rd_full** đến CPU để thông báo sẵn sàng cho hoạt động đọc. Khi CPU bắt đầu một hoạt động đọc, khối **AHBInterface** sẽ xác nhận yêu cầu đọc từ các tín hiệu điều khiển, đồng thời điều khiển quá trình đọc dữ liệu từ **BufferRead**, gộp các bit dữ liệu của mỗi khung thành 32 bit dữ liệu đọc **hrdata** và gửi đến CPU.

d/ Cấu trúc và giao thức của khối CtrlOut



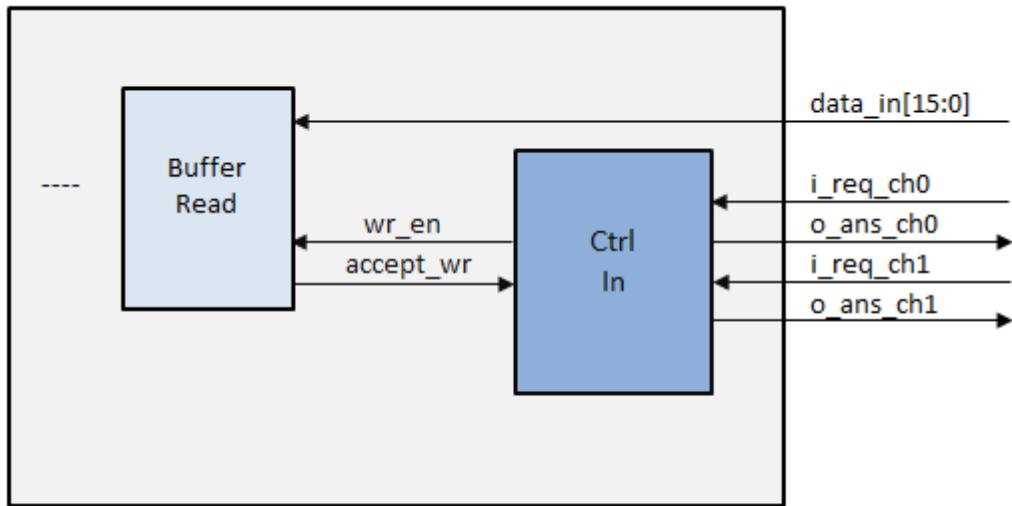
Hình 2.30 Giao diện khối CtrlOut

Khối **CtrlOut** như Hình 2.30 có nhiệm vụ tạo liên kết truyền nhận với router liên kết để truyền đi các dữ liệu trên **BufferWrite**. Đầu tiên dựa vào các tín hiệu **accept_rd** từ khối **BufferWrite**, khối **CtrlOut** biết được bộ đệm **BufferWrite** có chứa dữ liệu cần truyền hay không.

Nếu bộ đệm **BufferWrite** cần truyền dữ liệu đến router liên kết (tín hiệu **accept_rd** tích cực mức cao), khối **CtrlOut** truyền tín hiệu **req** tương ứng của bộ đệm đó đến router liên kết tại cạnh lên của xung clock.

Khi nhận được tín hiệu phản hồi **ans** từ router liên kết ngay sau đó, khối **CtrlOut** sẽ xuất tín hiệu **rd_en** đến bộ đệm **BufferWrite**. Dữ liệu từ bộ đệm sẽ được xuất ra ở cạnh lên xung clock kế đó.

e/ Cấu trúc và giao thức của khối CtrlIn



Hình 2.31 Giao diện khối CtrlIn

Khối **CtrlIn** như Hình 2.31 dùng để giao tiếp với **router** liên kết thông qua các tín hiệu điều khiển nhằm quyết định tín hiệu dữ liệu ngõ vào được nhận vào khối **BufferRead** hay không.

Khi tín hiệu **req** từ router liên kết tích cực (tích cực cạnh lén hoặc cạnh xuống) ở cạnh lén của xung clock nhằm đề nghị truyền dữ liệu, nếu khối **BufferRead** còn trống (tín hiệu **accept_wr** tích cực mức cao) thì khối **BufferRead** cho phép nhận dữ liệu từ ngõ vào. Khi đó khối **CtrlIn** sẽ trả về tín hiệu **ans** tích cực mức cao và xuất ra ngoài đến router liên kết nhằm thông báo khối **BufferRead** đã sẵn sàng nhận dữ liệu.

Ở cạnh lén xung clock tiếp theo đó, dữ liệu mới được truyền đến khối **BufferRead**. Cũng tại thời điểm này, tín hiệu **wr_en** được tích cực mức cao bởi khối **CtrlIn** gửi đến khối **BufferRead**.

Ở cạnh lén xung clock tiếp theo, dữ liệu mới được ghi vào khối **BufferRead**. Cũng tại cạnh lén xung clock này, tín hiệu **req** của router liên kết được tích cực nhằm thiết lập một chu kỳ truyền khung dữ liệu thứ hai.

Như vậy cơ chế truyền nhận được thực thi liên tục cho đến khi tất cả các khung dữ liệu của gói dữ liệu được truyền hết. Quá trình truyền nhận có thể bị gián đoạn do khói **BufferRead** đã đầy.

Trong trường hợp khói **BufferRead** đã đầy, tín hiệu **req** yêu cầu truyền dữ liệu được bỏ qua. Các router liên kết sẽ vẫn liên tục gửi các tín hiệu **req** ở xung clock kế tiếp.

Khi khói **BufferRead** đủ điều kiện để nhận dữ liệu, tín hiệu **ans** sẽ được tích cực tương ứng với tín hiệu **req** trước đó. Khi không có tín hiệu **req** tích cực, router hiểu dữ liệu không được truyền tiếp nữa.

2.8 Tạo mạng dữ liệu tự động

Để xây dựng phần mềm cho phép tạo mạng dữ liệu tự động (Soft IP) thích nghi với các ứng dụng khác nhau, luận văn sử dụng ngôn ngữ lập trình Perl.

Luận văn trình bày việc đánh giá trên mô hình NoC 4x4. Với phần mềm trên sẽ cho phép mở rộng mô hình NoC với số nút mạng n tùy ý phụ thuộc vào hai thông số số hàng và số cột. Tuy nhiên thiết kế router đơn chỉ hỗ trợ 2 bit cho địa chỉ Ox và 2 bit cho địa chỉ Oy, do đó phải thay đổi số bit địa chỉ của router phù hợp với số nút mạng yêu cầu.

Phần mềm cho phép tạo mạng dữ liệu tự động với 2 thông số do người dùng nhập vào:

- Số nút mạng hàng ngang
- Số nút mạng hàng dọc

```
[quanghan@localhost 01_Gen_Top]$ 
[quanghan@localhost 01_Gen_Top]$ ./run.sh 4 4
===== START GENERATING THE TOP FILE =====
===== NOC CONFIG : DIMENSION: 4; CHANNEL: 4 =====
The dimension of NoC is: 4; The channel number of NoC is: 2
DEFINE PORT LIST STEP
DEFINE PARAMETER STEP
DEFINE INPUT PORT STEP
DEFINE OUTPUT PORT STEP
DEFINE INTERNAL SIGNAL STEP
DEFINE ROUTER STEP
===== FINISH GENERATING THE TOP FILE =====
[quanghan@localhost 01_Gen_Top]$ █
```

Hình 2.32 Giao diện phần mềm tạo mạng dữ liệu tự động

Hình 2.32 mô tả giao diện phần mềm với lệnh run.sh tạo mạng dữ liệu tự động, ở đây ví dụ là 4x4 với số kênh ảo mặc định là 2.

Sau khi chạy xong đoạn Script sẽ tạo ra một file ‘top.v’ trong thư mục ‘output’ là top module của thiết kế chứa các router đơn và các kết nối giữa chúng.

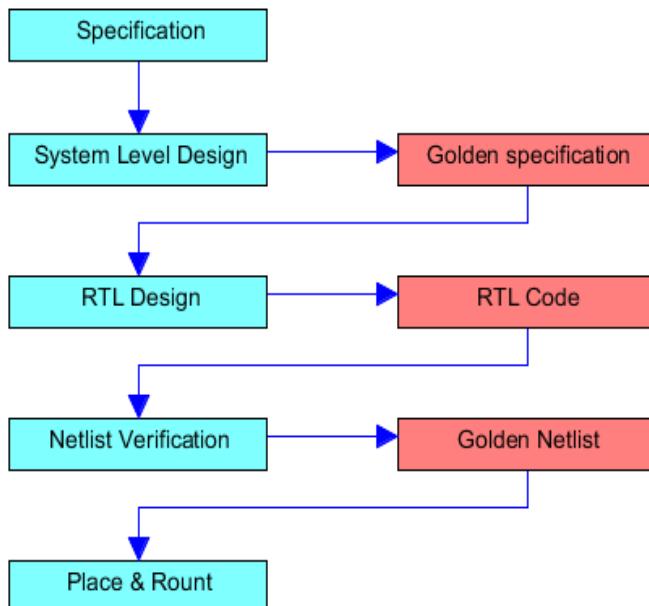
Với phần mềm tạo mạng dữ liệu tự động, việc thiết kế mô hình NoC trở nên đơn giản hơn và tránh sai sót trong quá trình kết nối các tín hiệu với nhau khi số nút mạng lớn. Ngoài ra, việc linh động số nút mạng giúp mô hình NoC thích nghi với các ứng dụng khác nhau cũng như giảm quy mô của thiết kế, tránh lãng phí những nút mạng không sử dụng.

CHƯƠNG 3 MÔ PHỎNG KIẾN TRÚC ROUTER & MÔ HÌNH NOC 4x4

Chương 3 sẽ tiến hành mô phỏng và kiểm tra hoạt động của router và mô hình NoC 4x4 nhằm đảm bảo ý tưởng tối ưu về số chu kỳ xung clock cho gói dữ liệu qua mỗi router có tính khả thi.

3.1 Mô hình mô phỏng

Để xác định rõ mô hình mô phỏng cho thiết kế, trước tiên luồng thiết kế vi mạch được xem xét.



Hình 3.1 Luồng thiết kế chip cơ bản

Hình 3.1 mô tả mô hình khái quát cho luồng thiết kế chip hiện nay. Dựa trên các bảng tóm tắt điều tra [1] về lĩnh vực NoC, mô hình NoC không nằm ngoài khuôn mẫu này.

- **System level design:** Đầu tiên để làm rõ các nghi vấn và kiểm định các khía cạnh của bản đặc tả, mô hình NoC được thiết kế bằng ngôn ngữ C++ với sự hỗ trợ của thư viện **systemC**. Với khả năng linh hoạt của ngôn ngữ C++, các giải thuật và đặc tính của NoC được mô tả khá hoàn chỉnh nhằm đúc kết các kết quả về hiệu năng và tính khả thi của cấu trúc NoC mong muốn. Sau khi đã thống nhất những yêu cầu và kết quả

mong muôn của mô hình NoC, bản đặc tả cuối cùng được đưa ra (**golden specification**) để thiết kế ở mức RTL.

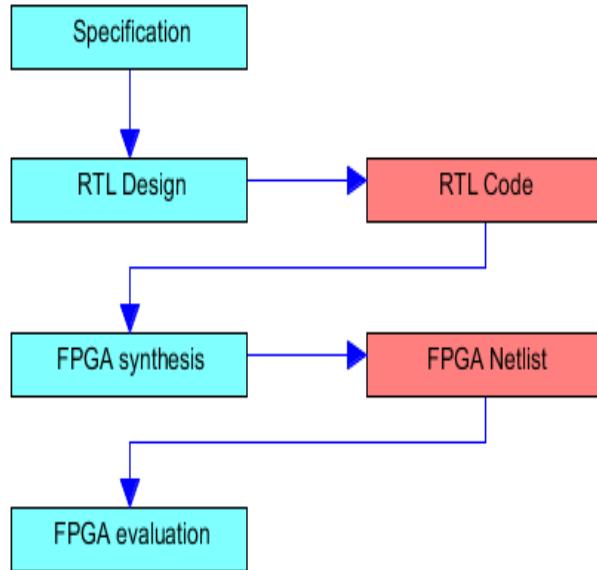
- **RTL design:** Dựa trên bảng đặc tả của bước thiết kế trước đó (**golden specification**), cấp độ RTL được thiết kế để mô tả lại cấu trúc đã được chi tiết hóa. Sau khi hoàn tất ở cấp độ RTL, thiết kế được kiểm tra sao cho thỏa mãn các chức năng mà bản đặc tả thiết kế (**golden specification**) đã đưa ra. Sau khi hoàn tất thiết kế ở cấp độ RTL, thiết kế được tổng hợp qua dạng **Netlist**.
- **Netlist Verification:** Netlist sau khi được tổng hợp từ mã RTL được kiểm tra và đánh giá những yêu cầu về thời gian thực và diện tích. Nếu các điều kiện này được thỏa mãn, Netlist được thực hiện bước **place & route**.
- **Place & route:** Netlist được sắp xếp lại và đánh giá lại về timing và diện tích.

Vì thời gian dành cho đề tài hạn chế cũng như mục đích của đề tài về việc làm rõ tính khả thi khi giảm số chu kỳ xung clock của gói dữ liệu khi đi qua router. Đề tài bỏ qua bước thiết kế ở cấp độ hệ thống (**system level design**), từ bản đặc tả và chi tiết giải thuật giao tiếp như đã nêu ở chương 2, cấp độ RTL được thiết kế. Đề tài lựa chọn ngôn ngữ Verilog để hiện thực thiết kế ở mức độ RTL.

Sau khi mã Verilog được thiết kế, tiến trình kiểm tra được tiến hành. Mô hình kiểm tra, các trường hợp kiểm tra cũng như công cụ để kiểm tra được thể hiện chi tiết.

Việc tổng hợp và kiểm tra ở cấp độ Netlist cũng sẽ được bỏ qua do yếu tố cơ sở vật chất. Thay vào đó, cấp độ RTL sẽ được tổng hợp trên FPGA. Thông qua FPGA, diện tích và sự hao tốn tài nguyên sẽ được thể hiện thay vì tổng hợp ở cấp độ Netlist với thư viện cho trước.

Hình 3.2 sẽ tóm tắt công việc mô phỏng thiết kế trong đề tài.



Hình 3.2 Mô hình kiểm tra thiết kế NoC 4x4

3.2 Ngôn ngữ mô phỏng

Verilog là một trong hai ngôn ngữ mô tả phần cứng chính (gồm VHDL và Verilog HDL) được người thiết kế phần cứng sử dụng để mô tả, thiết kế các hệ thống số, ví dụ như máy tính hay linh kiện điện tử.

Verilog dễ học và dễ sử dụng hơn VHDL. Verilog được chuẩn hoá theo chuẩn IEEE vào năm 1995 và 2001. Verilog rất giống ngôn ngữ C và được giới chuyên môn nghiên cứu, sử dụng nhiều.

Verilog HDL có thể được sử dụng để thiết kế hệ thống số ở nhiều mức khác nhau, ví dụ ở mức cao như các mô hình đặc trưng đến các mức thấp như mô hình bố trí dây, điện trở, transistor trên một mạch tích hợp; mô tả các công logic, flip_flop trong hệ thống số; mô tả thanh ghi và sự di chuyển dữ liệu giữa các thanh ghi (RTL - **Register Transfer Level**).

Hệ thống số là một hệ thống phức tạp bậc cao. Ở cấp độ chi tiết nhất, chúng có thể bao gồm hàng nghìn thành phần như: các transistor hoặc các công logic, cho nên với hệ thống số lớn, thiết kế ở mức công không còn sử dụng nữa. Qua nhiều thập kỷ, giản đồ logic của các thiết kế logic cũng không còn nhiều nữa. Ngày nay, sự phức tạp của phần cứng đã tăng lên ở một mức độ mà giản đồ của công logic hầu như vô ích khi nó chỉ biểu diễn một mạng lưới phức tạp các liên kết không theo chức năng của thiết kế. Từ những năm 1970, các kỹ sư điện và máy tính đổi hướng theo ngôn ngữ mô tả phần cứng (HDL). Hai ngôn ngữ mô tả phần

cứng nỗi bật trong kỹ thuật là Verilog và VHDL nhưng những nhà thiết kế công nghệ thích sử dụng Verilog hơn.

Verilog cho phép các nhà thiết kế logic thiết kế và mô tả hệ thống số ở nhiều mức độ khác nhau và có sự hỗ trợ từ các công cụ thiết kế bằng máy tính để giúp cho việc xử lý thiết kế ở những mức độ khác nhau.

Cách sử dụng cơ bản của Verilog HDL trong thiết kế mạch tích hợp là mô phỏng thiết kế và tạo mẫu trên FPGA trước khi chuyển sang sản xuất. Mục tiêu của Verilog không phải tạo ra những chip VLSI mà sử dụng Verilog để mô tả một cách chính xác chức năng của bất kỳ hệ thống số nào và nạp chương trình tạo mẫu lên FPGA, ví dụ như máy tính, các bộ vi xử lý,... tuy tốc độ chậm và lãng phí diện tích hơn. Những thiết kế mức thấp hơn trong Verilog được thực hiện trên VLSI để đạt đến tốc độ cực đại và có diện tích cực tiểu. Tuy nhiên sử dụng thiết kế dùng Verilog trên FPGA sẽ tiết kiệm chi phí và thời gian thiết kế.

3.3 Công cụ mô phỏng

Bộ công cụ VCS của công ty Synopsys được sử dụng để hỗ trợ trong quá trình mô phỏng. Mô hình cũng như các trường hợp kiểm tra được thể hiện ở mục 3.4 sau đây.

Để kiểm tra và đánh giá hiệu quả của mạng NoC 4x4, đầu tiên router đơn lẻ được kiểm tra để đảm bảo chất lượng truyền tải của một nút mạng. Trong tất cả các trường hợp kiểm tra một router đơn, chỉ một gói dữ liệu được truyền. Các trường hợp truyền nhiều gói dữ liệu được thực hiện kiểm tra trên mô hình NoC 4x4.

3.4 Kiểm tra và đánh giá kết quả mô phỏng trên một router đơn

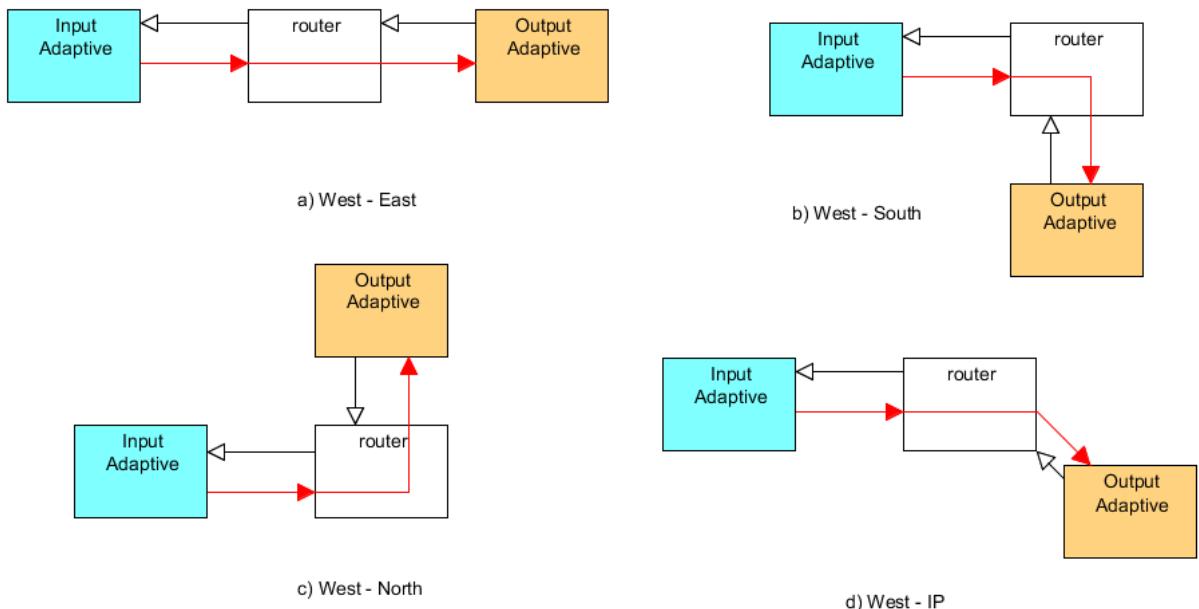
Ở mô hình kiểm tra một router đơn, chỉ một gói dữ liệu được gửi qua để kiểm tra router. Số khung dữ liệu trong gói dữ liệu và các luồng dữ liệu thay đổi theo từng trường hợp. Khi các trường hợp kênh truyền chia sẻ đường truyền, kỹ thuật kênh ảo được sử dụng và hai khái niệm **OutputAdaptive** sẽ được thiết kế để nhận dữ liệu từ một đường truyền ra duy nhất.

Do kiến trúc router có hỗ trợ hai kênh ảo ở mỗi ngõ vào/ra nên router có thể xử lý nhiều luồng dữ liệu đến từ các hướng khác nhau. Dựa trên số lượng luồng dữ liệu đến một router đơn có thể phân chia các trường hợp sau đây.

3.4.1 Router đơn nhận một luồng dữ liệu

Trường hợp router nhận duy nhất một luồng dữ liệu được thể hiện trong Hình 3.3.

- Dữ liệu truyền từ West → East: Hình 3.3.a.
- Dữ liệu truyền từ West → South: Hình 3.3.b.
- Dữ liệu truyền từ West → North: Hình 3.3.c.
- Dữ liệu truyền từ West → IP: Hình 3.3.d.



Hình 3.3 Các trường hợp kiểm tra router đơn nhận một luồng dữ liệu

Hình 3.3 cho thấy kết quả chạy mô phỏng quá trình truyền gói dữ liệu từ West đến East (Hình 3.3.a). Khi chân **rst_n** không còn tích cực (time = 150 ns), sau 5 chu kỳ xung clock, khung đầu tiên được ghi vào khối **OutputAdaptive**. Các bước tiếp theo được tiếp tục theo quy trình sau.

- Thiết lập quá trình truyền nhận giữa **InputAdaptive** và router.
- Dữ liệu được ghi vào router.
- Dữ liệu được giải mã để truyền đến ngõ ra của router.
- Dữ liệu được ghi vào ngõ ra của router đồng thời thiết lập đường truyền đến khối **OutputAdaptive**.
- Tại cạnh lên xung clock của chu kỳ kế tiếp dữ liệu được đọc ra. Tại cạnh lên của xung clock (cuối chu kỳ), dữ liệu được ghi vào khối **OutputAdaptive**.

Như vậy, tính từ lúc dữ liệu được ghi vào router và đọc ra khỏi router tốn 6 chu kỳ. Quá trình tạo kết nối giữa **InputAdaptive** và router chiếm 1 cycle. Quá trình tạo kết nối giữa **OutputAdaptive** và router không mất thời gian. Bảng 3-1, Bảng 3-2, Bảng 3-3 và Bảng 3-4 phân tích các kết quả đạt được cho các trường hợp mô tả ở Hình 3.3. Phân tích kết quả thu được quá trình truyền dữ liệu qua một router theo một luồng cố định. Nhờ cơ chế pipeline và cơ chế chuyển mạch gói, các khung dữ liệu khác trong gói dữ liệu không mất các chu kỳ xung clock dùng để thiết lập kết nối.

Do đó, số chu kỳ xung clock trung bình cho một khung dữ liệu trong gói dữ liệu 48 khung là 2.92~3.12 chu kỳ. Số liệu này thay đổi là do vấn đề đầy bộ nhớ các Fifo, một số chu kỳ xung clock trì hoãn để quá trình truyền được thực hiện nhằm giải phóng bộ nhớ cho các Fifo để chúng có thể nhận thêm các gói tin mới.

Bảng 3-1 Các trường hợp kiểm tra router đơn nhận một luồng dữ liệu

	time = 100	clk = 1	rst_n = 0	top_req = 1	data_out_check = 0000000000000000
	time = 110	clk = 0	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
	time = 150	clk = 1	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
	time = 200	clk = 0	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
1 period	time = 250	clk = 1	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
	time = 300	clk = 0	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
2 period	time = 350	clk = 1	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
	time = 400	clk = 0	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
3 period	time = 450	clk = 1	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
	time = 500	clk = 0	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
4 period	time = 550	clk = 1	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
	time = 600	clk = 0	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
	time = 650	clk = 1	rst_n = 1	top_req = 1	data_out_check = 0001100000000000
				
150 period	time = 15050	clk = 1	rst_n = 1	top_req = 1	data_out_check = 10011000100111
Kết luận	Tổng số khung dữ liệu : 48 khung; Tổng số chu kỳ clock trì hoãn : 150 chu kỳ Trung bình số chu kỳ clock trì hoãn cho một khung dữ liệu: 3.12 chu kỳ/khung				

Bảng 3-2 Quá trình truyền một gói dữ liệu từ West → South

	time = 100	clk = 1	rst_n = 0	top_req = 1	data_out_check = 0000000000000000
	time = 110	clk = 0	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
	time = 150	clk = 1	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
	time = 200	clk = 0	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
1 period	time = 250	clk = 1	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
	time = 300	clk = 0	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
2 period	time = 350	clk = 1	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
	time = 400	clk = 0	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
3 period	time = 450	clk = 1	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
	time = 500	clk = 0	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
4 period	time = 550	clk = 1	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
	time = 600	clk = 0	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
	time = 650	clk = 1	rst_n = 1	top_req = 1	data_out_check = 0010010000000000
				
140 period	time = 14050	clk = 1	rst_n = 1	top_req = 1	data_out_check = 10100100100111
Kết luận	Tổng số khung dữ liệu : 48 khung; Tổng số chu kỳ clock trì hoãn : 140 chu kỳ Trung bình số chu kỳ clock trì hoãn cho một khung dữ liệu: 2.92 chu kỳ/khung				

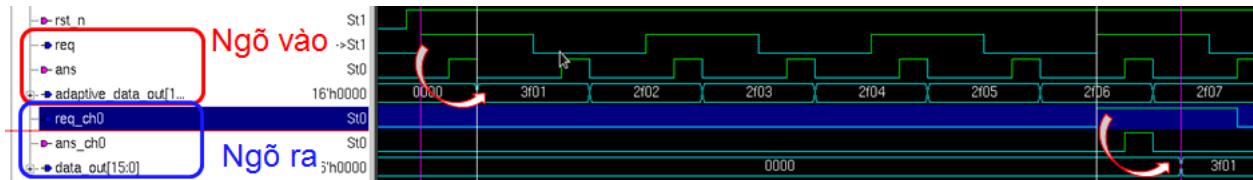
Bảng 3-3 Quá trình truyền một gói dữ liệu từ West → North

	time = 100	clk = 1	rst_n = 0	top_req = 1	data_out_check = 0000000000000000
	time = 110	clk = 0	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
	time = 150	clk = 1	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
	time = 200	clk = 0	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
1 period	time = 250	clk = 1	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
	time = 300	clk = 0	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
2 period	time = 350	clk = 1	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
	time = 400	clk = 0	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
3 period	time = 450	clk = 1	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
	time = 500	clk = 0	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
4 period	time = 550	clk = 1	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
	time = 600	clk = 0	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
	time = 650	clk = 1	rst_n = 1	top_req = 1	data_out_check = 0001000000000000
				
140 period	time = 14050	clk = 1	rst_n = 1	top_req = 1	data_out_check = 10010000100111
Kết luận	Tổng số khung dữ liệu : 48 khung; Tổng số chu kỳ clock trì hoãn : 140 chu kỳ Trung bình số chu kỳ clock trì hoãn cho một khung dữ liệu: 2.92 chu kỳ/khung				

Bảng 3-4 Quá trình truyền một gói dữ liệu từ West → IP

	time = 100	clk = 1	rst_n = 0	top_req = 1	data_out_check = 0000000000000000
	time = 110	clk = 0	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
	time = 150	clk = 1	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
	time = 200	clk = 0	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
1 period	time = 250	clk = 1	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
	time = 300	clk = 0	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
2 period	time = 350	clk = 1	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
	time = 400	clk = 0	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
3 period	time = 450	clk = 1	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
	time = 500	clk = 0	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
4 period	time = 550	clk = 1	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
	time = 600	clk = 0	rst_n = 1	top_req = 1	data_out_check = 0000000000000000
	time = 650	clk = 1	rst_n = 1	top_req = 1	data_out_check = 0001010000000000
				
150 period	time = 15050	clk = 1	rst_n = 1	top_req = 1	data_out_check = 00010100100111
Kết luận	Tổng số khung dữ liệu : 48 khung; Tổng số chu kỳ clock trì hoãn : 150 chu kỳ Trung bình số chu kỳ clock trì hoãn cho một khung dữ liệu: 3.12 chu kỳ/khung				

Hình 3.4 mô tả dữ liệu từ ngõ vào đến ngõ ra của một router đơn cho thấy các khung dữ liệu tuần tự truyền ra theo đúng giao thức REQ/ANS đã đề cập ở chương 2.



Hình 3.4 Dữ liệu truyền qua 1 router đơn

Ngoài ra, Hình 3.5 mô tả giản đồ xung cho trường hợp 5 đường truyền dữ liệu ra 5 hướng khác nhau không xung đột. Giản đồ xung xác nhận các gói dữ liệu ngõ ra cùng lúc cho 5 hướng khác nhau là phù hợp với đặc tả kiến trúc của router.



Hình 3.5 Mô phỏng 5 đường truyền không tranh chấp nhau

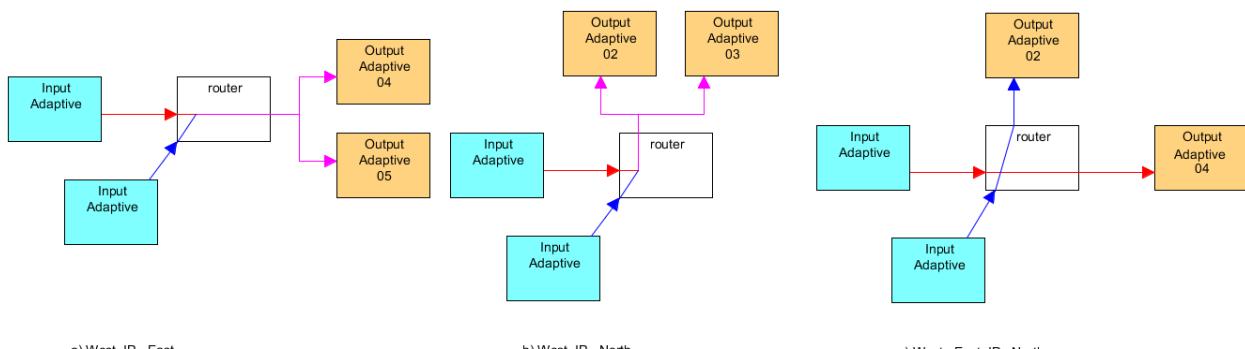
3.4.2 Router đơn nhận hai luồng dữ liệu

Trường hợp router nhận hai luồng dữ liệu từ hai hướng khác nhau được thể hiện trong Hình 3.6.

- Dữ liệu truyền từ West, IP → East: Hình 3.6.a.
- Dữ liệu truyền từ West, IP → North: Hình 3.6.b.
- Dữ liệu truyền từ West → East, IP → North: Hình 3.6.c.

Bảng 3-5, Bảng 3-6 và Bảng 3-7 phân tích các kết quả đạt được cho các trường hợp mô tả ở Hình 3.6. Trong trường hợp mô phỏng này, kỹ thuật kênh ảo được sử dụng cho ngõ ra của dữ liệu vì có hai luồng dữ liệu cần đi ra cùng một hướng. Dựa trên kỹ thuật kênh ảo, từng khung dữ liệu của mỗi hướng sẽ được truyền ra xen kẽ nhau theo mỗi chu kỳ xung clock.

Riêng trường hợp c, kỹ thuật kênh ảo không được sử dụng do hai luồng dữ liệu ra hai hướng khác nhau. Trường hợp này được kiểm tra nhằm đảm bảo việc phân luồng diễn ra độc lập không phụ thuộc lẫn nhau.



Hình 3.6 Các trường hợp kiểm tra router đơn nhận hai luồng dữ liệu

Phân tích kết quả thu được quá trình truyền dữ liệu qua một router theo hai luồng dữ liệu cùng một hướng cho thấy khung đầu tiên chỉ tồn 4 chu kỳ xung clock để đi qua router trung gian, việc thêm 1 chu kỳ xung clock là do quá trình phân xử của kênh ảo nhằm phân định độ ưu tiên của khung dữ liệu trên các **FifoOut**.

Nhờ cơ chế pipeline và cơ chế chuyển mạch gói, các khung dữ liệu khác trong gói dữ liệu không mất các chu kỳ xung clock dùng để thiết lập kết nối. Do đó, số chu kỳ xung clock trung bình cho một khung dữ liệu là 3.8 cho 17 khung dữ liệu.

Bảng 3-5 Quá trình truyền một gói dữ liệu từ West, IP → East

	time = 100	clk = 1	rst_n = 0	top_req = 1	data_out_check_04 = 0000000000000000 data_out_check_05 = 0000000000000000
	time = 110	clk = 0	rst_n = 1	top_req = 1	data_out_check_04 = 0000000000000000 data_out_check_05 = 0000000000000000
	time = 150	clk = 1	rst_n = 1	top_req = 1	data_out_check_04 = 0000000000000000 data_out_check_05 = 0000000000000000
	time = 200	clk = 0	rst_n = 1	top_req = 1	data_out_check_04 = 0000000000000000 data_out_check_05 = 0000000000000000
1 period	time = 250	clk = 1	rst_n = 1	top_req = 1	data_out_check_04 = 0000000000000000 data_out_check_05 = 0000000000000000
	time = 300	clk = 0	rst_n = 1	top_req = 1	data_out_check_04 = 0000000000000000 data_out_check_05 = 0000000000000000
2 period	time = 350	clk = 1	rst_n = 1	top_req = 1	data_out_check_04 = 0000000000000000 data_out_check_05 = 0000000000000000

7 period	time = 750	clk = 1	rst_n = 1	top_req = 1	data_out_check_04 = 0000000000000000 data_out_check_05 = 00110000000000
	time = 500	clk = 0	rst_n = 1	top_req = 1	data_out_check_04 = 0000000000000000 data_out_check_05 = 0011000000000000
8 period	time = 850	clk = 1	rst_n = 1	top_req = 1	data_out_check_04 = 0011011110000 data_out_check_05 = 0011000000000000
				
65 period	time = 6550	clk = 1	rst_n = 1	top_req = 1	data_out_check_04 = 01011011110111 data_out_check_05 = 10011000000111
	time = 6600	clk = 0	rst_n = 1	top_req = 1	data_out_check_04 = 01011011110111 data_out_check_05 = 10011000000111
66 period	time = 6650	clk = 1	rst_n = 1	top_req = 1	data_out_check_04 = 10011011111000 data_out_check_05 = 10011000000111
Kết luận	Tổng số khung dữ liệu từ West : 8 khung Tổng số khung dữ liệu từ IP : 9 khung Tổng số chu kỳ clock trì: 66 chu kỳ Trung bình số chu kỳ clock trì hoãn cho một khung dữ liệu: 3.8 chu kỳ				

Bảng 3-6 Quá trình truyền một gói dữ liệu từ West, IP → North

	time = 100	clk = 1	rst_n = 0	top_req = 1	data_out_check_02 = 0000000000000000 data_out_check_03 = 0000000000000000
	time = 110	clk = 0	rst_n = 1	top_req = 1	data_out_check_02 = 0000000000000000 data_out_check_03 = 0000000000000000
	time = 150	clk = 1	rst_n = 1	top_req = 1	data_out_check_02 = 0000000000000000 data_out_check_03 = 0000000000000000
	time = 200	clk = 0	rst_n = 1	top_req = 1	data_out_check_02 = 0000000000000000 data_out_check_03 = 0000000000000000
1 period	time = 250	clk = 1	rst_n = 1	top_req = 1	data_out_check_02 = 0000000000000000 data_out_check_03 = 0000000000000000
	time = 300	clk = 0	rst_n = 1	top_req = 1	data_out_check_02 = 0000000000000000 data_out_check_03 = 0000000000000000
2 period	time = 350	clk = 1	rst_n = 1	top_req = 1	data_out_check_02 = 0000000000000000 data_out_check_03 = 0000000000000000

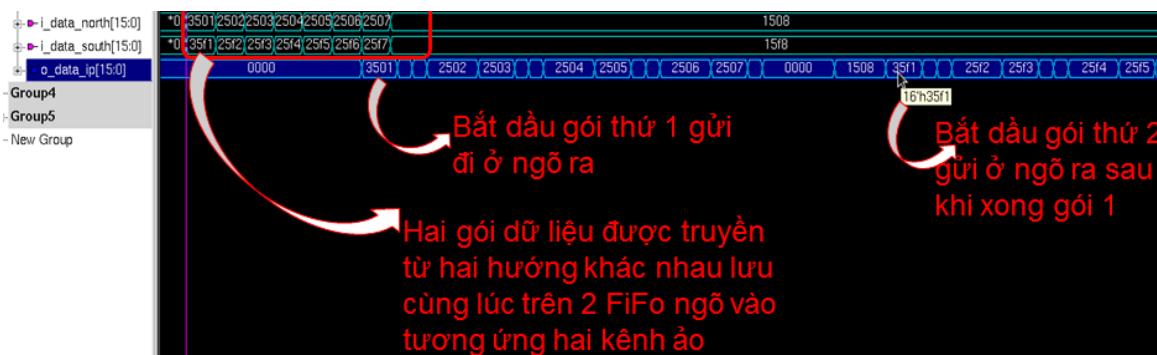
7 period	time = 750	clk = 1	rst_n = 1	top_req = 1	data_out_check_02 = 0000000000000000 data_out_check_03 = 0011000000000000
	time = 500	clk = 0	rst_n = 1	top_req = 1	data_out_check_02 = 0000000000000000 data_out_check_03 = 0011000000000000
8 period	time = 850	clk = 1	rst_n = 1	top_req = 1	data_out_check_02 = 0011011110000 data_out_check_03 = 0011000000000
				
65 period	time = 6550	clk = 1	rst_n = 1	top_req = 1	data_out_check_02 = 10011011110111 data_out_check_03 = 10011000000111
	time = 6600	clk = 0	rst_n = 1	top_req = 1	data_out_check_02 = 10011011110111 data_out_check_03 = 10011000000111
66 period	time = 6650	clk = 1	rst_n = 1	top_req = 1	data_out_check_02 = 10011011111000 data_out_check_03 = 10011000000111
Kết luận	Tổng số khung dữ liệu từ West : 8 khung Tổng số khung dữ liệu từ IP : 9 khung Tổng số chu kỳ clock trì: 66 chu kỳ Trung bình số chu kỳ clock trì hoãn cho một khung dữ liệu: 3.8 chu kỳ				

Bảng 3-7 Quá trình truyền một gói dữ liệu từ West, East → IP, North

	time = 100	clk = 1	rst_n = 0	top_req = 1	data_out_check_02 = 0000000000000000 data_out_check_04 = 0000000000000000
	time = 110	clk = 0	rst_n = 1	top_req = 1	data_out_check_02 = 0000000000000000 data_out_check_04 = 0000000000000000
	time = 150	clk = 1	rst_n = 1	top_req = 1	data_out_check_02 = 0000000000000000 data_out_check_04 = 0000000000000000
	time = 200	clk = 0	rst_n = 1	top_req = 1	data_out_check_02 = 0000000000000000 data_out_check_04 = 0000000000000000
1 period	time = 250	clk = 1	rst_n = 1	top_req = 1	data_out_check_02 = 0000000000000000 data_out_check_04 = 0000000000000000
	time = 300	clk = 0	rst_n = 1	top_req = 1	data_out_check_02 = 0000000000000000 data_out_check_04 = 0000000000000000
2 period	time = 350	clk = 1	rst_n = 1	top_req = 1	data_out_check_02 = 0000000000000000 data_out_check_04 = 0000000000000000
...
6 period	time = 650	clk = 1	rst_n = 1	top_req = 1	data_out_check_02 = 00011011110000 data_out_check_04 = 0010010000000000

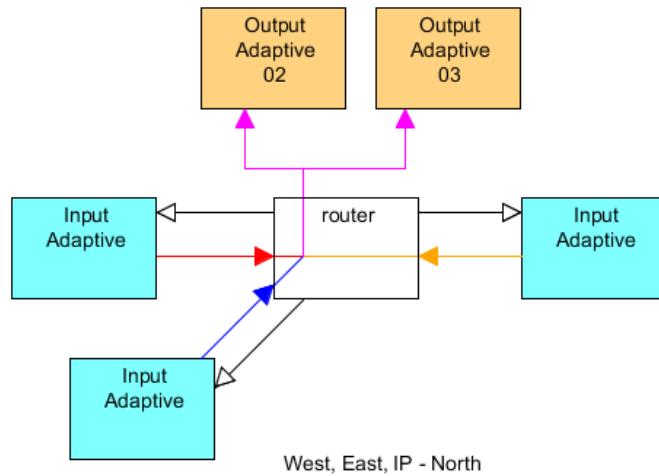
29 period	time = 2950	clk = 1	rst_n = 1	top_req = 1	data_out_check_02 = 10011011111000 data_out_check_04 = 10011000000111
Kết luận	Tổng số khung dữ liệu từ West : 8 khung Tổng số chu kỳ clock trì hoãn cho hướng từ West : 29 chu kỳ Tổng số khung dữ liệu từ IP : 9 khung Tổng số chu kỳ clock trì hoãn cho hướng từ West : 29 chu kỳ Trung bình số chu kỳ clock trì hoãn cho một khung dữ liệu: + Từ West: 3.625 chu kỳ/khung + Từ East: 3.22 chu kỳ/khung				

Hình 3.7 mô tả giản đồ xung cho trường hợp hai luồng dữ liệu yêu cầu ra cùng một hướng. Khi đó, hai luồng dữ liệu này sẽ xen kẽ truyền ra theo từng gói dữ liệu. Cụ thể khi gói dữ liệu của một luồng (8 khung dữ liệu cho một gói) được truyền đi thì gói của luồng khác được tiếp tục. Việc xen kẽ gói dữ liệu dựa trên hoạt động của hai kênh ảo cho phép sự chia sẻ tài nguyên đường dữ liệu truyền dẫn.



Hình 3.7 Mô phỏng hai luồng dữ liệu ra cùng một hướng

3.4.3 Router đơn nhận ba luồng dữ liệu



Hình 3.8 Trường hợp kiểm tra router đơn nhận ba luồng dữ liệu

Trường hợp router nhận ba luồng dữ liệu từ ba hướng khác nhau được thể hiện trong Hình 3.8. Trường hợp dữ liệu truyền từ West, East, IP → North được kiểm tra. Bảng 3-8 thể hiện quá trình mô phỏng của trường hợp này.

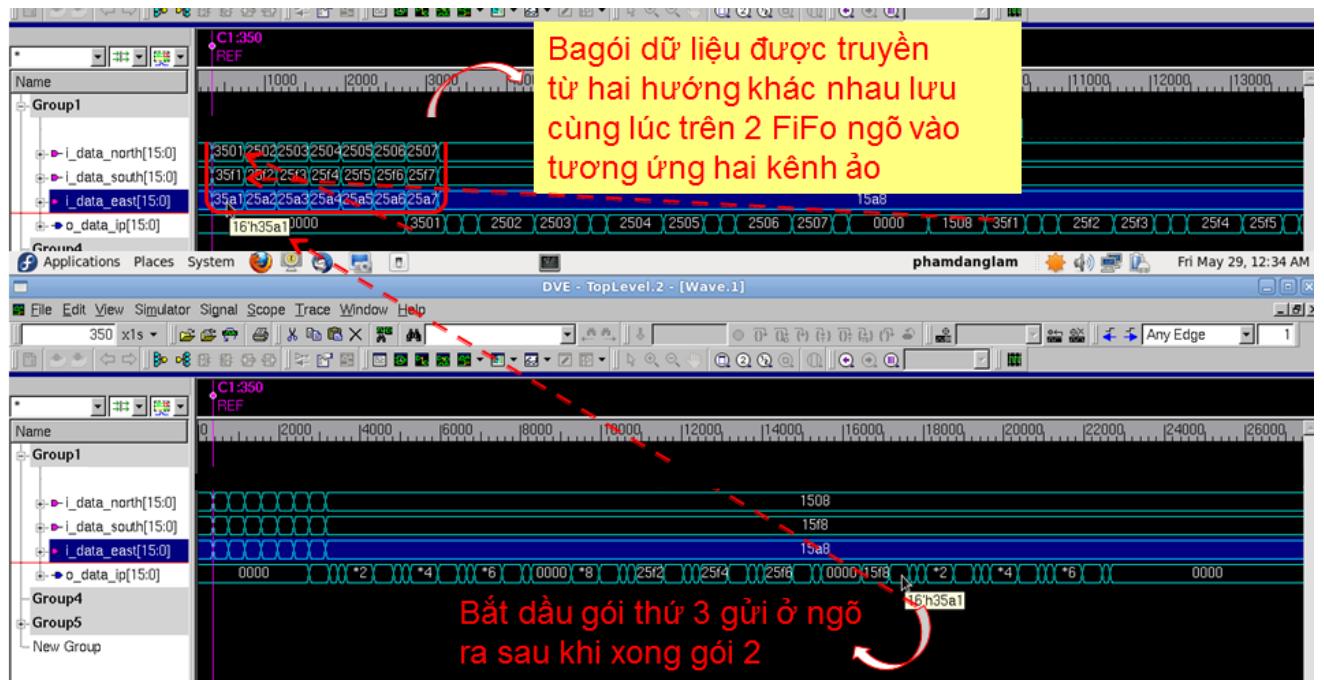
Phân tích kết quả thu được quá trình truyền dữ liệu qua một router theo ba luồng dữ liệu cùng một hướng cho thấy khung đầu tiên chỉ tồn 4 chu kỳ xung clock để đi qua router trung gian, việc thêm 1 chu kỳ xung clock là do quá trình phân xử của kênh ảo nhằm phân định độ ưu tiên của khung dữ liệu trên các **FifoOut**. Nhờ cơ chế pipeline và cơ chế chuyển mạch gói, các khung dữ liệu khác trong gói dữ liệu không mất các chu kỳ xung clock dùng để thiết lập kết nối. Do đó, số chu kỳ xung clock trung bình cho một khung dữ liệu là 3.4 cho 25 khung dữ liệu.

Bảng 3-8 Quá trình truyền một gói dữ liệu từ West, East, IP → North

	time = 100	clk = 1	rst_n = 0	top_req = 1	data_out_check_02 = 0000000000000000 data_out_check_03 = 0000000000000000
	time = 110	clk = 0	rst_n = 1	top_req = 1	data_out_check_02 = 0000000000000000 data_out_check_03 = 0000000000000000
	time = 150	clk = 1	rst_n = 1	top_req = 1	data_out_check_02 = 0000000000000000 data_out_check_03 = 0000000000000000
	time = 200	clk = 0	rst_n = 1	top_req = 1	data_out_check_02 = 0000000000000000 data_out_check_03 = 0000000000000000
1 period	time = 250	clk = 1	rst_n = 1	top_req = 1	data_out_check_02 = 0000000000000000 data_out_check_03 = 0000000000000000
	time = 300	clk = 0	rst_n = 1	top_req = 1	data_out_check_02 = 0000000000000000 data_out_check_03 = 0000000000000000
2 period	time = 350	clk = 1	rst_n = 1	top_req = 1	data_out_check_02 = 0000000000000000 data_out_check_03 = 0000000000000000
.....
7 period	time = 750	clk = 1	rst_n = 1	top_req = 1	data_out_check_02 = 0000000000000000 data_out_check_03 = 0001100000000000
	time = 800	clk = 0	rst_n = 1	top_req = 1	data_out_check_02 = 0000000000000000 data_out_check_03 = 0001100000000000
8 period	time = 850	clk = 1	rst_n = 1	top_req = 1	data_out_check_02 = 00011010100000 data_out_check_03 = 0001100000000000

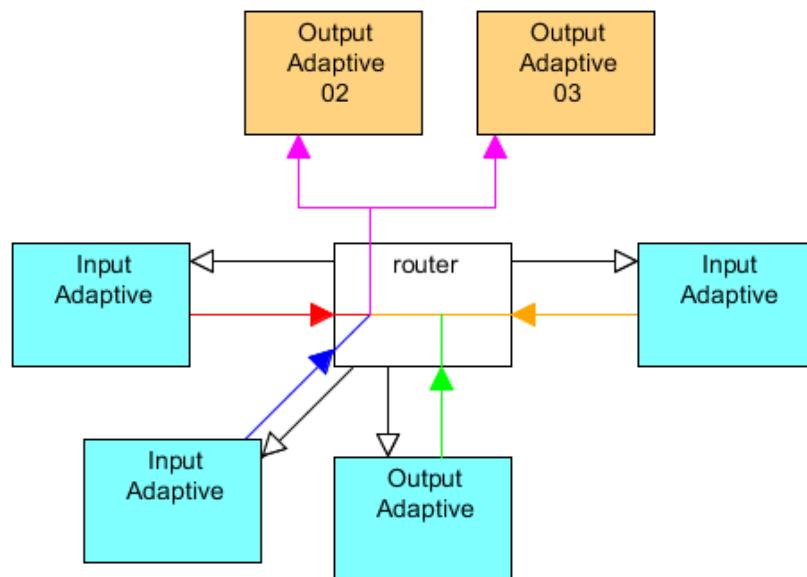
66 period	time = 6650	clk = 1	rst_n = 1	top_req = 1	data_out_check_02 = 10011010100111 data_out_check_03 = 01011000000101
	time = 6650	clk = 0	rst_n = 1	top_req = 1	data_out_check_02 = 10011010100111 data_out_check_03 = 01011000000101
67 period	time = 6750	clk = 1	rst_n = 1	top_req = 1	data_out_check_02 = 00011011110000 data_out_check_03 = 01011000000101
.....
85 period	time = 8550	clk = 1	rst_n = 1	top_req = 1	data_out_check_02 = 01011011110011 data_out_check_03 = 10011000000111
Kết luận	Tổng số khung dữ liệu từ West : 8 khung Tổng số khung dữ liệu từ East : 8 khung Tổng số khung dữ liệu từ IP : 9 khung Tổng số chu kỳ clock trì hoãn : 85 chu kỳ Trung bình số chu kỳ clock trì hoãn cho một khung dữ liệu: 3.4				

Hình 3.9 mô tả 3 gói dữ liệu tranh chấp cùng một đường truyền. Dựa trên độ ưu tiên được thiết lập từ trước, hai gói dữ liệu từ hướng bắc và nam được ưu tiên lần lượt truyền ra trước với hai kênh ảo. Sau khi hai gói này hoàn tất, gói dữ liệu từ phía đông được truyền tiếp tục.



Hình 3.9 Giản đồ xung 3 gói dữ liệu tranh chấp 1 đường truyền

3.4.4 Router đơn nhận bốn luồng dữ liệu



Hình 3.10 Trường hợp kiểm tra router đơn nhận bốn luồng dữ liệu

Chương 3: Mô Phỏng Kiến Trúc Router & Mô Hình NoC 4x4

Trường hợp router nhận bốn luồng dữ liệu từ bốn hướng khác nhau được thể hiện trong Hình 3.10. Chỉ một trường hợp kiểm tra dữ liệu truyền từ West, IP, South, East → North.

Phân tích số liệu Bảng 3-9 cho thấy kết quả độ trễ hoãn số chu kỳ xung clock cho mỗi khung dữ liệu có giá trị gần với các kết quả mô phỏng đã thực hiện. Điều này chứng tỏ độ trễ hoãn số chu kỳ xung clock trong quá trình nhận dữ liệu không phụ thuộc vào tình trạng của router.

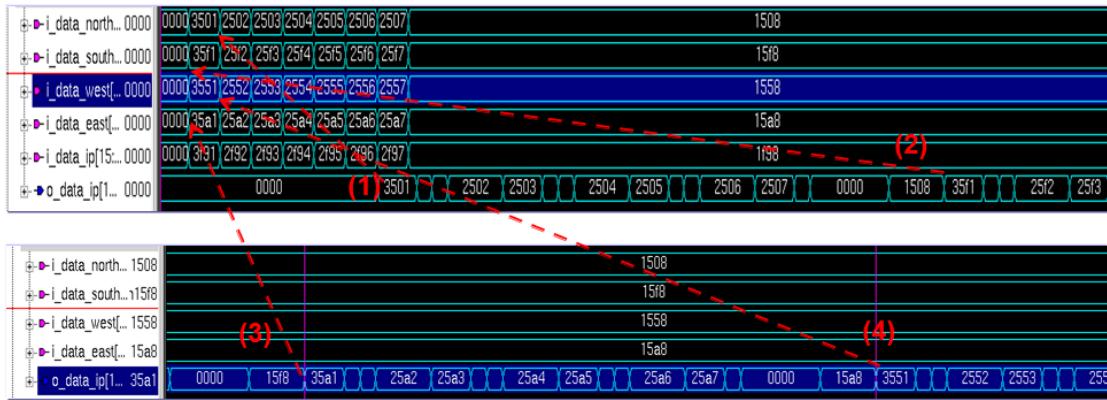
Có thể kết luận rằng khi các router truyền một luồng dữ liệu duy nhất (điều kiện tốt nhất khi router hoạt động) cũng như trường hợp tranh chấp bốn luồng dữ liệu ra cùng một hướng (điều kiện tranh chấp nhiều nhất), hoạt động của router vẫn ổn định theo cấu trúc đã đề nghị và số chu kỳ trễ hoãn cho một khung dữ liệu là gần như không đổi.

Bảng 3-9 Quá trình truyền một gói dữ liệu từ West, East, IP, South → North

	time = 100	clk = 1	rst_n = 0	top_req = 1	data_out_check_02 = 0000000000000000 data_out_check_03 = 0000000000000000
	time = 110	clk = 0	rst_n = 1	top_req = 1	data_out_check_02 = 0000000000000000 data_out_check_03 = 0000000000000000
	time = 150	clk = 1	rst_n = 1	top_req = 1	data_out_check_02 = 0000000000000000 data_out_check_03 = 0000000000000000
	time = 200	clk = 0	rst_n = 1	top_req = 1	data_out_check_02 = 0000000000000000 data_out_check_03 = 0000000000000000
1 period	time = 250	clk = 1	rst_n = 1	top_req = 1	data_out_check_02 = 0000000000000000 data_out_check_03 = 0000000000000000
	time = 300	clk = 0	rst_n = 1	top_req = 1	data_out_check_02 = 0000000000000000 data_out_check_03 = 0000000000000000
2 period	time = 350	clk = 1	rst_n = 1	top_req = 1	data_out_check_02 = 0000000000000000 data_out_check_03 = 0000000000000000
.....
7 period	time = 750	clk = 1	rst_n = 1	top_req = 1	data_out_check_02 = 0000000000000000 data_out_check_03 = 0011000000000000
	time = 800	clk = 0	rst_n = 1	top_req = 1	data_out_check_02 = 0000000000000000 data_out_check_03 = 0011000000000000
8 period	time = 850	clk = 1	rst_n = 1	top_req = 1	data_out_check_02 = 0011010101000000 data_out_check_03 = 0011000000000000

66 period	time = 6650	clk = 1	rst_n = 1	top_req = 1	data_out_check_02 = 10011010100111 data_out_check_03 = 0011001010000
	time = 6700	clk = 0	rst_n = 1	top_req = 1	data_out_check_02 = 10011010100111 data_out_check_03 = 0011001010000
.....
121 period	Time = 12150	clk = 1	rst_n = 1	top_req = 1	data_out_check_02 = 0011011110110 data_out_check_03 = 10011001010111
	Time = 12100	clk = 0	rst_n = 1	top_req = 1	data_out_check_02 = 01011011110110 data_out_check_03 = 10011001010111
122 period	Time = 12250	clk = 1	rst_n = 1	top_req = 1	data_out_check_02 = 10011011110111 data_out_check_03 = 10011001010111
Kết luận	Tổng số khung dữ liệu từ West : 8 khung; Tổng số khung dữ liệu từ East : 8 khung Tổng số khung dữ liệu từ South : 8 khung; Tổng số khung dữ liệu từ IP : 8 khung Tổng số chu kỳ clock trì hoãn : 122 chu kỳ Trung bình số chu kỳ clock trì hoãn cho một khung dữ liệu: 3.8				

Hình 3.11 mô tả 4 luồng dữ liệu tranh chấp cùng một đường truyền ngõ ra IP, dựa trên mức độ ưu tiên khác nhau (North-South-East-West-IP), các gói dữ liệu lần lượt được gửi ra theo thứ tự được đánh dấu. Ở đây, ngõ ra chính là IP.



Hình 3.11 Giản đồ mô phỏng 4 luồng dữ liệu tranh chấp một đường truyền

Qua các số liệu mô phỏng một router đơn, các số liệu được đúc kết theo Bảng 3-10 cho thấy độ trễ trung bình cho một khung dữ liệu là 3.12~3.8 chu kỳ xung clock. Khi số khung dữ liệu trong một gói dữ liệu càng tăng thì độ trễ trung bình càng giảm nhờ cơ chế pipeline.

Quá trình kiểm tra trên router đơn chưa đánh giá chính xác độ ổn định của độ trễ hoãn xung clock cho một khung dữ liệu. Các kết quả kiểm tra trên NoC 4x4 được xem xét để có những kết luận chính xác hơn.

Bảng 3-10 Tổng hợp các trường hợp mô phỏng trên một router

No.	Trường hợp	Số khung dữ liệu	Độ trễ hoãn trung bình
1	1 luồng dữ liệu ra 1 hướng	48	3.12
2	2 luồng dữ liệu ra 1 hướng	17	3.8
3	2 luồng dữ liệu ra 2 hướng	8.0/9.0	3.2/3.6
4	3 luồng dữ liệu ra 1 hướng	25	3.4
5	4 luồng dữ liệu ra 1 hướng	32	3.8

3.5 Kiểm tra và đánh giá kết quả mô phỏng trên mô hình NoC 4x4

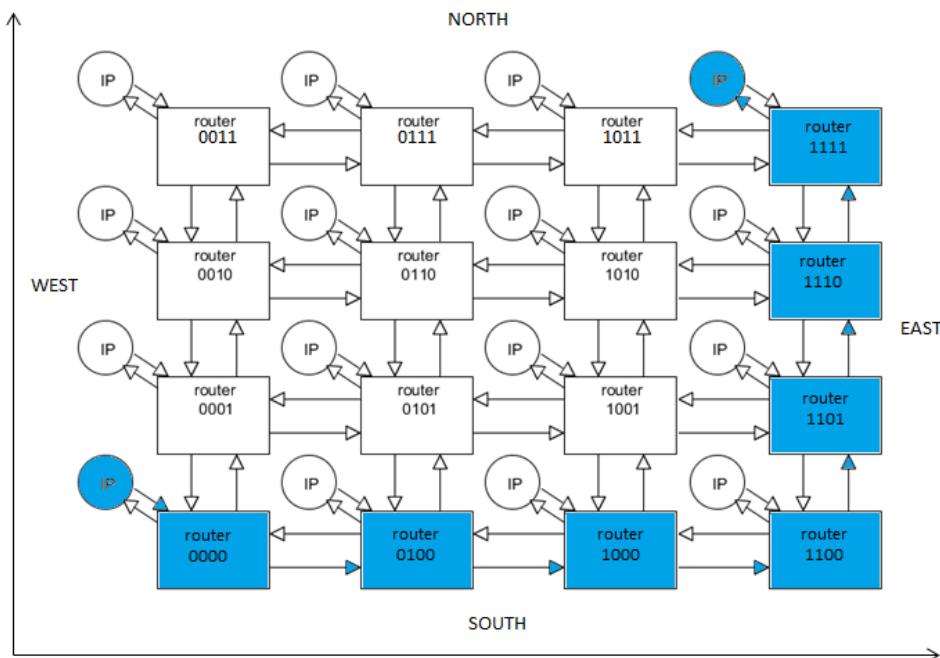
Sau khi đã kiểm tra và đảm bảo router hoạt động chính xác cho các trường hợp ở mục 3.4, các router được liên kết thành mô hình NoC 4x4.

Trong mô hình kiểm tra mạng NoC 4x4, một số đặc tính trong quá trình mô phỏng được giới thiệu.

- Số gói dữ liệu được tăng dần từ 100 đến 1000 khung. Mỗi lần tăng 100 khung dữ liệu. Tổng cộng có 10 lần đo.
- Mỗi gói dữ liệu bao gồm 4 khung dữ liệu.

- Chu kỳ xung clock trong quá trình mô phỏng được chọn là 10 ns.
- Đánh giá băng thông dựa trên chu kỳ xung clock này.

Để đánh giá mô hình NoC 4x4, ta sẽ đánh giá đường truyền dài nhất từ IP của router “0000” đến IP của router “1111”.



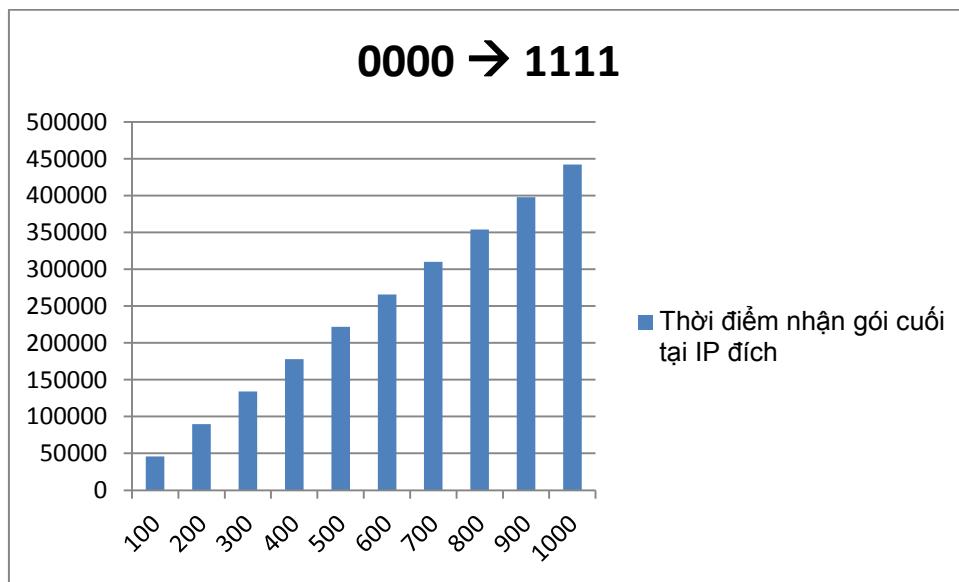
Hình 3.12 Truyền dữ liệu từ router 0000 đến router 1111

Trong mô hình này các gói dữ liệu được truyền qua 7 router liên tiếp để kiểm tra độ ổn định của mạng NoC 4x4. Quá trình mô phỏng và tổng kết dữ liệu được thể hiện ở Bảng 3-11.

Bảng 3-11 Kết quả truyền dữ liệu từ router 0000 đến router 1111

No.	Số khung dữ liệu	Thời điểm kết nối – reset inactive	Thời điểm nhận gói 1 tại IP đích	Thời điểm nhận gói cuối tại IP đích	Trung bình trì hoãn (chu kỳ/gói)
1	100	105 ns	2305 ns	45865 ns	43.56
2	200	105 ns	2305 ns	89865 ns	43.78
3	300	105 ns	2305 ns	133865 ns	43.853333
4	400	105 ns	2305 ns	177865 ns	43.89
5	500	105 ns	2305 ns	221865 ns	43.912
6	600	105 ns	2305 ns	265865 ns	43.926666
7	700	105 ns	2305 ns	309865 ns	43.937143
8	800	105 ns	2305 ns	353865 ns	43.945
9	900	105 ns	2305 ns	397865 ns	43.951111
10	1000	105 ns	2305 ns	441865 ns	43.956

Dựa trên kết quả truyền từ 100 gói dữ liệu đến 1000 gói dữ liệu ta có được biểu đồ cột Hình 3.13.



Hình 3.13 Quan hệ giữa số gói dữ liệu – thời gian truyền nhận

Xét gói dữ liệu đầu tiên đến được **IP đích** tốn 220 chu kỳ xung clock. Trong đó, 21 chu kỳ đầu tiên được sử dụng để truyền dữ liệu từ **IP nguồn**, đóng khung dữ liệu và tạo kết nối giữa khối **NI** và **router_0000**, 31 chu kỳ cuối cùng được sử dụng để tạo kết nối và truyền dữ

liệu từ **router_1111** đến khối **NI**, đóng khung dữ liệu và truyền đến **IP đích**, 168 chu kỳ còn lại chứng tỏ khung dữ liệu chỉ tồn 24 chu kỳ cho mỗi router mà nó đi qua.

Trong 10 lần chạy mô phỏng cho trường hợp này, độ trễ hoãn cho khung đầu tiên kể từ khi bắt đầu kết nối đến khi đến được đích giữ cố định không đổi.

Việc tăng dần gói dữ liệu đồng nghĩa với việc số chu kỳ xung clock cần dùng để truyền hết tất cả các gói dữ liệu cũng tăng theo. Với số gói dữ liệu tăng lên là 100 gói ở mỗi lần đo, ta thấy được 44.000 (ns) được cộng thêm vào khi gói dữ liệu cuối cùng tới được **IP đích**.

Việc các số liệu tăng tịnh tiến như Hình 3.13 thông kê đã cho thấy độ trễ hoãn trung bình của một gói dữ liệu cho một đường truyền qua 7 router rất ổn định (44 chu kỳ/gói). Điều này chứng tỏ mạng NoC 4x4 hoạt động rất ổn định trong trường hợp đường truyền dài. Qua trường hợp kiểm tra trên cũng khẳng định việc ứng dụng định tuyến XY hoạt động chính xác. Luồng dữ liệu phải đi qua hai hướng Ox trước và Oy sau như đã giới thiệu ở các chương trước.

Tiếp theo, việc tính toán băng thông của mô hình NoC được thực hiện. Do độ trễ hoãn số chu kỳ xung clock cho mỗi khung dữ liệu là không đổi.

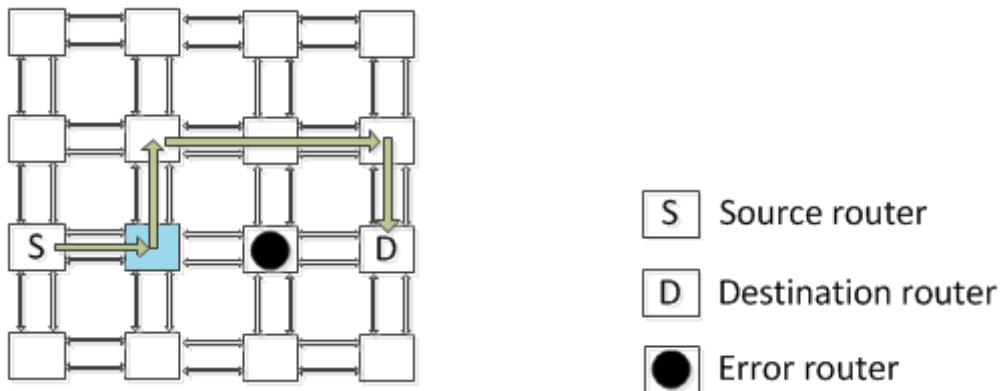
Bảng 3-12 Công thức tính băng thông cho bus 32 bit dữ liệu của mô hình NoC 4x4

Ký hiệu	Thông số	Công thức	Giá trị	Đơn vị
f	Tần số xung clock	Chọn lựa	800	Mhz
p	Chu kỳ xung clock	$p = 1/f$	1.25	ns
n	Số khung dữ liệu	Chọn lựa	1000	frame
b	Số bit dữ liệu trong khung dữ liệu	Chọn lựa	32	bit
t1	Thời điểm nhận khung dữ liệu đầu tiên	Kết quả mô phỏng	541	ns
t2	Thời điểm nhận khung dữ liệu cuối cùng	Kết quả mô phỏng	88453	ns
T	Thời gian trung bình một gói dữ liệu	$(t2 - t1) / n$	88	ns
V	Tốc độ trung bình cho một gói dữ liệu	$1/T$	11.25	MegaFrame/s
C	Băng thông cho bus với 32 bit dữ liệu	$V * b$	360	Mbps

Theo như Bảng 3-12, băng thông của bus trong mô hình NoC 4x4 là **360 Mbps**. Băng thông có thể tăng lên dựa vào sự cải thiện những chỉ số sau: Số bit dữ liệu trong một khung dữ liệu. Trong mô hình router đang giới thiệu, 32 bit dữ liệu cho một khung dữ liệu. Việc mở rộng số bit dữ liệu không ảnh hưởng đến kiến trúc của mạch. Nó chỉ ảnh hưởng đến diện tích và năng lượng của mạng.

3.6 Kiểm tra và đánh giá kết quả tránh lỗi trong khi truyền dữ liệu

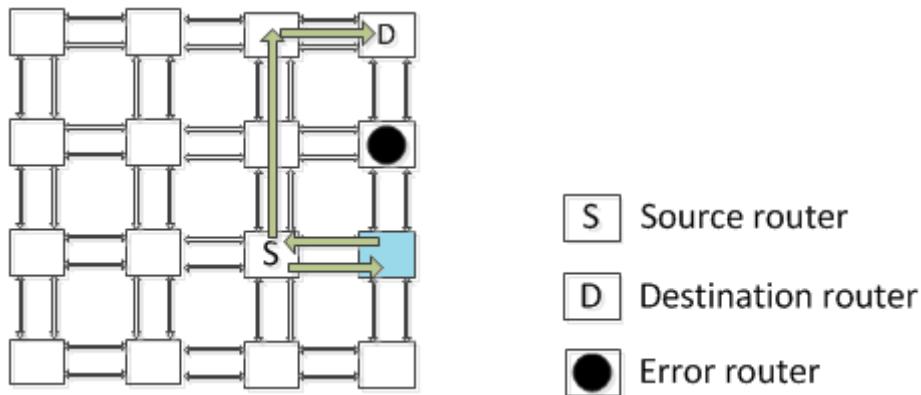
Để có thể kiểm tra và đánh giá được khả năng tránh lỗi trong quá trình truyền dữ liệu, một số trường hợp giả lập lỗi và giàn đồ xung mô phỏng sau đây được phân tích.



i _data_west[15:0]	16'h1758	0000	3751	2752	2753	2754	2755	2756	2757	1758
o _data_east[15:0]	16'h0000							0000		
o _data_north[15:0]	16'h0000					0000		3751	2752	0000

Hình 3.14 Gửi từ West sang East nhưng chuyển hướng sang North

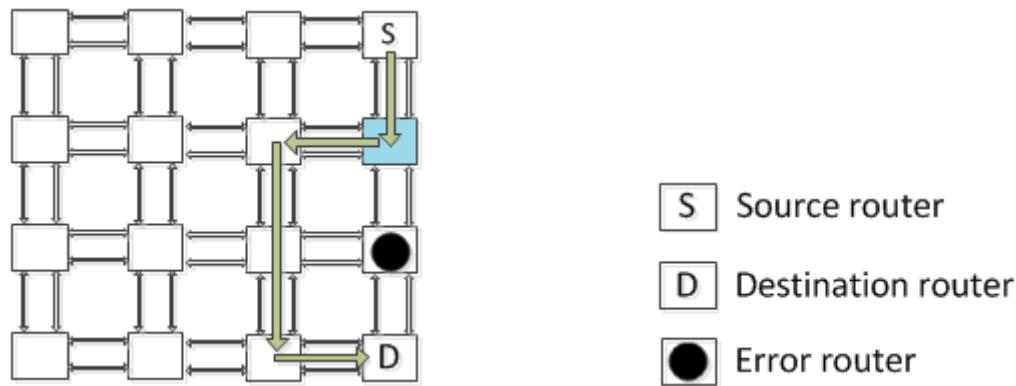
Hình 3.14 cho thấy dữ liệu đi từ West tới vị trí router đích 0111 (giá trị 7 trong mã hex) tương ứng Oy(01) và Ox(11). Vị trí router hiện tại là 0101 tương ứng Oy(01) và Ox(01), như vậy dữ liệu đầu tiên là 51(8 bit mã hex) sẽ được truyền từ West sang East (đi sang phải) vì vị trí router đích trên phương Ox là 11 lớn hơn router hiện hành là 01. Tuy nhiên router tương ứng ở vị trí 0110 bị lỗi nên dữ liệu được truyền đi và ra ở hướng North.



i data west[1...	0000	3f51	2f52	2f53	2f54	2f55	2f56	2f57		1f58
o data north[...									0000	
o data west[1...					0000		f51	ef52	*0	ef52

Hình 3.15 Gửi từ West sang North nhưng quay lại West

Hình 3.15 cho thấy dữ liệu đi từ West tới vị trí router đích 1111 (giá trị f trong mã hex) tương ứng Oy(11) và Ox(11). Vị trí router hiện tại là 0111 tương ứng Oy(01) và Ox(11), như vậy dữ liệu đầu tiên là 51(8 bit mã hex) sẽ được truyền từ West sang North (đi lên) vì vị trí router đích trên phương Oy là 11 lớn hơn router hiện hành là 01. Tuy nhiên hướng truyền bị lỗi nên dữ liệu phải truyền quay lại West vì lý do đây là biên phải của mô hình NoC 4x4 nên dữ liệu sẽ truyền ngược lại và đổi hướng. Giá trị 4 bit trọng số cao nhất f ở ngõ ra West so với 3 ở ngõ vào West cho thấy quyết định chuyển hướng.



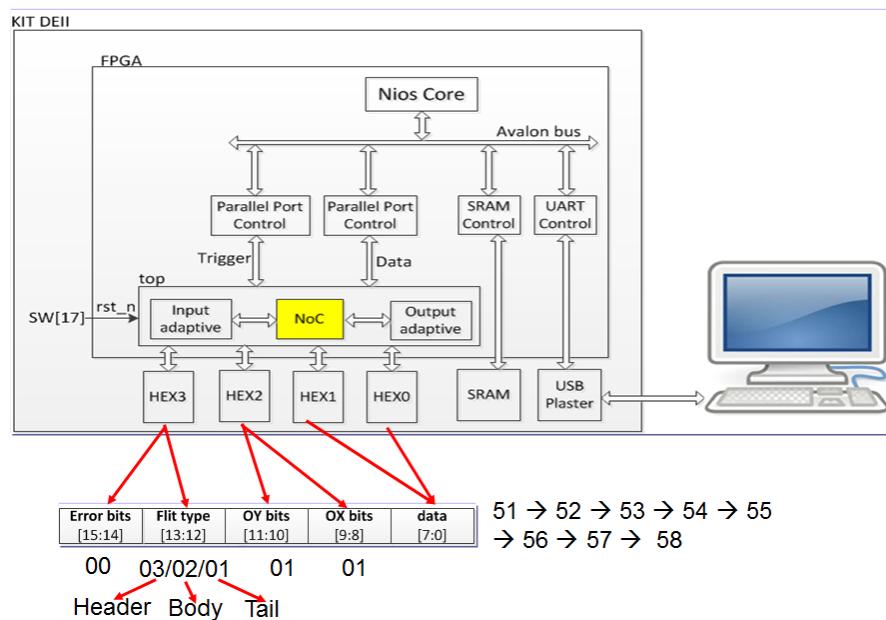
i data north[...	0000	3301	2302	2303	2304	2305	2306	2307		1308
o data south[...									0000	
o data west[1...					0000		f301	*2	*0	e302

Hình 3.16 Gửi từ North sang South nhưng chuyển hướng sang West

Hình 3.16 cho thấy dữ liệu đi từ North tới vị trí router đích 0011 (giá trị 3 trong mã hex) tương ứng Oy(00) và Ox(11). Vị trí router hiện tại là 1011 tương ứng Oy(10) và Ox(11), như vậy dữ liệu đầu tiên là 01(8 bit mã hex) sẽ được truyền từ North sang South (đi xuống dưới) vì vị trí router đích trên phương Oy là 00 nhỏ hơn router hiện hành là 10, trong khi đó vị trí Ox là bằng nhau và bằng 11. Tuy nhiên hướng truyền bị lỗi nên dữ liệu được truyền đi và ra ở hướng West.

3.7 Mô hình kiểm tra NoC 4x4 trên FPGA

Sau khi các mô phỏng trên giản đồ xung được hoàn tất đảm bảo tính chính xác của các chức năng đúng với những đặc tả kỹ thuật. Thiết kế được đổ lên FPGA và kiểm tra. Mô hình kết nối FPGA được thể hiện bởi Hình 3.17. Theo đó, mô hình NoC 4x4 được kết nối với các **InputAdaptive** và **OutputAdaptive**. Khi Nios core kích thích khôi **InputAdaptive** hoạt động, khôi này sẽ truyền vào NoC 4x4 và xuất ra ở **OutputAdaptive**. Dữ liệu ở **OutputAdaptive** sau đó sẽ được truyền ra các led 7 đoạn (HEX) và màn hình máy tính tương ứng giá trị gói dữ liệu.



Hình 3.17 Mô hình kiểm tra trên FPGA

CHƯƠNG 4 KẾT QUẢ THIẾT KẾ VẬT LÝ

4.1 Kết quả tổng hợp từ cấp độ RTL xuống lớp cổng

Để có thể tổng hợp kiến trúc NoC từ cấp độ RTL xuống lớp cổng, việc thiết lập môi trường và lựa chọn công cụ là cần thiết. Sau đây tóm tắt chi tiết các công cụ cũng như nền tảng tổng hợp cho khối kiến trúc NoC.

Bảng 4-1 Nền tảng tổng hợp khối NoC 4x4 xuống lớp cổng

Hệ điều hành	Linux (Fedora)
Ngôn ngữ tạo môi trường	Bash Shell, Perl, Tcl
Ngôn ngữ thiết kế	Verilog
Thư viện sử dụng	TSMC 130nm
Công cụ	Design compiler
Thiết kế	NoC 4x4
Tên file chính	vr.v
Các ràng buộc	+ Trì hoãn ngõ vào + Trì hoãn ngõ ra + Tần số xung clock + Sử dụng bộ nhớ nội + Reset bất đồng bộ + Trì hoãn clock

Thư viện chọn cho thiết kế là thư viện TSMC 130nm. Cấu trúc cơ bản của thư viện TSMC 130nm được thể hiện ở cây thư mục sau.

```

library/
`-- TSMC_130nm
  `-- sc-x
    |-- cell_list
    |-- symbols
    / |-- cadence
    / |-- edif
    / `-- synopsys
    |-- synopsys
    / |-- README
    / |-- scx2_tsmc_cl013g_ff_1p32v_0c.db
    / |-- scx2_tsmc_cl013g_ff_1p32v_0c.lib
    / |-- scx2_tsmc_cl013g_ff_1p32v_m40c.db
    / |-- scx2_tsmc_cl013g_ff_1p32v_m40c.lib
    / |-- scx2_tsmc_cl013g_ss_1p08v_125c.db
    / |-- scx2_tsmc_cl013g_ss_1p08v_125c.lib
    / |-- scx2_tsmc_cl013g_tt_1p2v_25c.db
    / `-- scx2_tsmc_cl013g_tt_1p2v_25c.lib
  `-- verilog
    |-- README
    |-- tsmc13.v

```

Mô hình hình cây sau mô tả cấu trúc môi trường tổng hợp cũng như môi trường kiểm tra.

```

./
|-- formality.log
|-- README
/ |-- README.DC-RM.txt
|-- reports
/ |-- vr.check_design.rpt
/ |-- vr.fmv_unmatched_points.rpt
/ |-- vr.mapped.area.rpt
/ |-- vr.mapped.clock_gating.rpt
/ |-- vr.mapped.power.rpt
/ |-- vr.mapped.qor.rpt
/ `-- vr.mapped.timing.rpt
|-- results
/ |-- vr.compile_ultra.ddc
/ |-- vr.elab.ddc
/ |-- vr.mapped.ddc
/ |-- vr.mapped.sdc
/ |-- vr.mapped.svf
/ `-- vr.mapped.v
|-- rm_dc_scripts
/ |-- dc.dft_ autofix_config.tcl
/ |-- dc.dft_occ_config.tcl
/ |-- dc.mcmm.scenarios.tcl
/ |-- dc.tcl
/ |-- dc_top.tcl
/ |-- fm_gate2gate.tcl
/ |-- fm.tcl

```

```

/ |-- fm_top.tcl
/ |-- mvrc.tcl
/ |-- mvrc_top.tcl
/ `-- vr.constraints.tcl
|-- rm_setup
/ |-- common_setup.tcl
/ |-- dc_setup_filenames_gate2gate.tcl
/ |-- dc_setup_filenames.tcl
/ |-- dc_setup_gate2gate.tcl
/ |-- dc_setup_minlib.tcl
/ |-- dc_setup.tcl
/ |-- default.svf
/ `-- vr.constraints.tcl
|-- rtl
/ `-- vr.v
|-- run_dc
|-- run_fm
/-- run_fm_gate
|-- wire_load

```

File “vr” là file chứa tất cả các mã Verilog mô tả kiến trúc NoC 4x4.

Bảng 4-2 Bảng đặc tả cây thư mục môi trường tổng hợp

STT	Tên	Đặc tả
1	README	Thư mục chứa các file hướng dẫn sử dụng môi trường
2	reports	Thư mục chứa các file tường trình các lỗi cũng như các thông số
3	results	Thư mục chứa các kết quả
4	rm_dc_scripts	Thư mục chứa các công cụ thực thi viết bởi ngôn ngữ TCL
5	rm_setup	Thư mục chứa các file thiết lập ràng buộc
6	rtl	Thư mục chứa thiết kế ở cấp độ RTL
7	run_dc	File thực thi tổng hợp xuống lớp cổng
8	run_fm	File thực thi kiểm tra thiết kế ở RTL và lớp cổng
9	run_fm_gate	File thực thi kiểm tra thiết kế ở lớp cổng trước và sau đi dây
10	wire_load	File chứa thông số ước lượng thời gian trì hoãn các dây dẫn

Các thiết lập ràng buộc được thể hiện ở các giá trị sau.

```

create_clock -name CLOCK_SYS [get_ports CLOCK] -period 1.25 -waveform {0 0.625}
set_clock_uncertainty 0.1 CLOCK_SYS
set_clock_latency 0.1 CLOCK_SYS
set_clock_transition 0.1 CLOCK_SYS

set_dont_touch_network CLOCK_SYS
set_dont_touch_rst_n
set_ideal_network rst_n

set_max_fanout 50 [all_inputs]
set_max_fanout 50 [all_designs]

set_input_delay -clock CLOCK_SYS 0.1 [all_inputs]
set_output_delay -clock CLOCK_SYS 0.1 [all_outputs]

```

Các file tường thuật cho phép thấy được kết quả diện tích ước lượng từ công cụ tổng hợp. Với diện tích công NAND2 (Công NOT-AND 2 ngõ vào) là 6.7 (Số liệu được lấy từ thư viện của TSMC tương ứng file *scx2_tsmc_cl013g_tt_1p2v_25c.lib*), có thể thấy tổng số công được ước lượng tương ứng là 250615 (Lấy số liệu diện tích 1679122.357517 từ kết quả tổng hợp chia cho giá trị 6.7).

```

*****
Report : area
Design : vr
*****
Library(s) Used:
  scx2_tsmc_cl013g_tt_1p2v_25c (File:
  /home/hieunguyen/TOOL/Library/01_TSMC130nm_frontend/sc-
  x/synopsys/scx2_tsmc_cl013g_tt_1p2v_25c.db)

Number of ports:      1431
Number of nets:       2286
Number of cells:      69
Number of references: 18

```

```

Combinational area:   558164.521320
Noncombinational area: 1120957.836197
Net Interconnect area: undefined (No wire load specified)

```

```

Total cell area:      1679122.357517
Total area:           undefined
1

```

File tường thuật sau cho thấy các yêu cầu về mặt thời gian được đáp ứng. Báo cáo tường thuật đường tín hiệu dài nhất được thỏa mãn với tần số tổng hợp cho trước. Chỉ số “SLACK” 0.01 dương cho phép các bước vật lý tiếp theo sau đó được tiến hành và mang tính khả thi cao.

Report : timing																																																																		
-path full																																																																		
-delay max																																																																		
-nets																																																																		
-max_paths 1																																																																		
-transition_time																																																																		
Design : vr																																																																		

<i>Operating Conditions: tt_1p2v_25c Library: scx2_tsmc_cl013g_tt_1p2v_25c</i>																																																																		
<i>Wire Load Model Mode: top</i>																																																																		
<i>Startpoint: full_router_1_3/router_01/FiFo_Out_South_ch0/addr_wr_reg_1_</i>																																																																		
<i>(rising edge-triggered flip-flop clocked by CLOCK_SYS)</i>																																																																		
<i>Endpoint: full_router_1_3/router_01/FiFo_Out_South_ch0/data_out_reg_6_</i>																																																																		
<i>(rising edge-triggered flip-flop clocked by CLOCK_SYS)</i>																																																																		
<i>Path Group: CLOCK_SYS</i>																																																																		
<i>Path Type: max</i>																																																																		
Attributes:																																																																		
<i>d - dont_touch</i>																																																																		
<i>u - dont_use</i>																																																																		
<i>mo - map_only</i>																																																																		
<i>so - size_only</i>																																																																		
<i>i - ideal_net or ideal_network</i>																																																																		
<table border="1"> <thead> <tr> <th>Point</th> <th>Fanout</th> <th>Trans</th> <th>Incr</th> <th>Path</th> <th>Attributes</th> </tr> </thead> <tbody> <tr> <td><i>clock CLOCK_SYS (rise edge)</i></td> <td></td> <td>0.00</td> <td>0.00</td> <td></td> <td></td> </tr> <tr> <td><i>clock network delay (ideal)</i></td> <td></td> <td>0.10</td> <td>0.10</td> <td></td> <td></td> </tr> <tr> <td><i>full_router_1_3/router_01/FiFo_Out_South_ch0/addr_wr_reg_1_</i>/CK (DFFRX2)</td> <td></td> <td></td> <td>0.10</td> <td>0.00 #</td> <td></td> </tr> <tr> <td><i>0.10 r</i></td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td><i>full_router_1_3/router_01/FiFo_Out_South_ch0/addr_wr_reg_1_</i>/Q (DFFRX2)</td> <td></td> <td>0.06</td> <td>0.34</td> <td></td> <td></td> </tr> <tr> <td><i>0.44f</i></td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td><i>full_router_1_3/router_01/FiFo_Out_South_ch0/addr_wr[1] (net)</i></td> <td>4</td> <td>0.00</td> <td>0.44f</td> <td></td> <td></td> </tr> <tr> <td><i>full_router_1_3/router_01/FiFo_Out_South_ch0/U242/Y (NAND4BX1)</i></td> <td></td> <td>0.09</td> <td>0.14</td> <td>0.59f</td> <td></td> </tr> <tr> <td><i>full_router_1_3/router_01/FiFo_Out_South_ch0/n344 (net)</i></td> <td>1</td> <td>0.00</td> <td>0.59f</td> <td></td> <td></td> </tr> <tr> <td><i>full_router_1_3/router_01/FiFo_Out_South_ch0/U273/Y (OA22X4)</i></td> <td></td> <td>0.06</td> <td>0.18</td> <td>0.77f</td> <td></td> </tr> </tbody> </table>	Point	Fanout	Trans	Incr	Path	Attributes	<i>clock CLOCK_SYS (rise edge)</i>		0.00	0.00			<i>clock network delay (ideal)</i>		0.10	0.10			<i>full_router_1_3/router_01/FiFo_Out_South_ch0/addr_wr_reg_1_</i> /CK (DFFRX2)			0.10	0.00 #		<i>0.10 r</i>						<i>full_router_1_3/router_01/FiFo_Out_South_ch0/addr_wr_reg_1_</i> /Q (DFFRX2)		0.06	0.34			<i>0.44f</i>						<i>full_router_1_3/router_01/FiFo_Out_South_ch0/addr_wr[1] (net)</i>	4	0.00	0.44f			<i>full_router_1_3/router_01/FiFo_Out_South_ch0/U242/Y (NAND4BX1)</i>		0.09	0.14	0.59f		<i>full_router_1_3/router_01/FiFo_Out_South_ch0/n344 (net)</i>	1	0.00	0.59f			<i>full_router_1_3/router_01/FiFo_Out_South_ch0/U273/Y (OA22X4)</i>		0.06	0.18	0.77f	
Point	Fanout	Trans	Incr	Path	Attributes																																																													
<i>clock CLOCK_SYS (rise edge)</i>		0.00	0.00																																																															
<i>clock network delay (ideal)</i>		0.10	0.10																																																															
<i>full_router_1_3/router_01/FiFo_Out_South_ch0/addr_wr_reg_1_</i> /CK (DFFRX2)			0.10	0.00 #																																																														
<i>0.10 r</i>																																																																		
<i>full_router_1_3/router_01/FiFo_Out_South_ch0/addr_wr_reg_1_</i> /Q (DFFRX2)		0.06	0.34																																																															
<i>0.44f</i>																																																																		
<i>full_router_1_3/router_01/FiFo_Out_South_ch0/addr_wr[1] (net)</i>	4	0.00	0.44f																																																															
<i>full_router_1_3/router_01/FiFo_Out_South_ch0/U242/Y (NAND4BX1)</i>		0.09	0.14	0.59f																																																														
<i>full_router_1_3/router_01/FiFo_Out_South_ch0/n344 (net)</i>	1	0.00	0.59f																																																															
<i>full_router_1_3/router_01/FiFo_Out_South_ch0/U273/Y (OA22X4)</i>		0.06	0.18	0.77f																																																														

<i>full_router_1_3/router_01/FiFo_Out_South_ch0/n345 (net)</i>	<i>2</i>	<i>0.00</i>	<i>0.77 f</i>			
<i>full_router_1_3/router_01/FiFo_Out_South_ch0/U275/Y (NAND2X2)</i>		<i>0.09</i>	<i>0.06</i>	<i>0.83 r</i>		
<i>full_router_1_3/router_01/FiFo_Out_South_ch0/n536 (net)</i>	<i>4</i>	<i>0.00</i>	<i>0.83 r</i>			
<i>full_router_1_3/router_01/FiFo_Out_South_ch0/U52/Y (NOR2X2)</i>		<i>0.12</i>	<i>0.09</i>	<i>0.92 f</i>		
<i>full_router_1_3/router_01/FiFo_Out_South_ch0/n451 (net)</i>	<i>16</i>	<i>0.00</i>	<i>0.92 f</i>			
<i>full_router_1_3/router_01/FiFo_Out_South_ch0/U375/Y (NAND2XL)</i>		<i>0.09</i>	<i>0.09</i>	<i>1.01 r</i>		
<i>full_router_1_3/router_01/FiFo_Out_South_ch0/n384 (net)</i>	<i>1</i>	<i>0.00</i>	<i>1.01 r</i>			
<i>full_router_1_3/router_01/FiFo_Out_South_ch0/U220/Y (OAI21IX1)</i>		<i>0.10</i>	<i>0.07</i>	<i>1.08 f</i>		
<i>full_router_1_3/router_01/FiFo_Out_South_ch0/n386 (net)</i>	<i>1</i>	<i>0.00</i>	<i>1.08 f</i>			
<i>full_router_1_3/router_01/FiFo_Out_South_ch0/U234/Y (AO21X4)</i>		<i>0.03</i>	<i>0.12</i>	<i>1.19 f</i>		
<i>full_router_1_3/router_01/FiFo_Out_South_ch0/n586 (net)</i>	<i>1</i>	<i>0.00</i>	<i>1.19 f</i>			
<i>full_router_1_3/router_01/FiFo_Out_South_ch0/data_out_reg_6/_D (DFFRX2)</i>		<i>0.03</i>	<i>0.00</i>			
<i>1.19 f</i>						
<i>data arrival time</i>		1.19				
<i>clock CLOCK_SYS (rise edge)</i>	<i>1.25</i>	<i>1.25</i>				
<i>clock network delay (ideal)</i>	<i>0.10</i>	<i>1.35</i>				
<i>clock uncertainty</i>	<i>-0.10</i>	<i>1.25</i>				
<i>full_router_1_3/router_01/FiFo_Out_South_ch0/data_out_reg_6/_D (DFFRX2)</i>		<i>0.00</i>	<i>1.25 r</i>			
<i>library setup time</i>	<i>-0.05</i>	<i>1.20</i>				
<i>data required time</i>		<i>1.20</i>				
<hr/>						
<i>data required time</i>		1.20				
<i>data arrival time</i>		-1.19				
<hr/>						
<i>slack (MET)</i>		0.01				

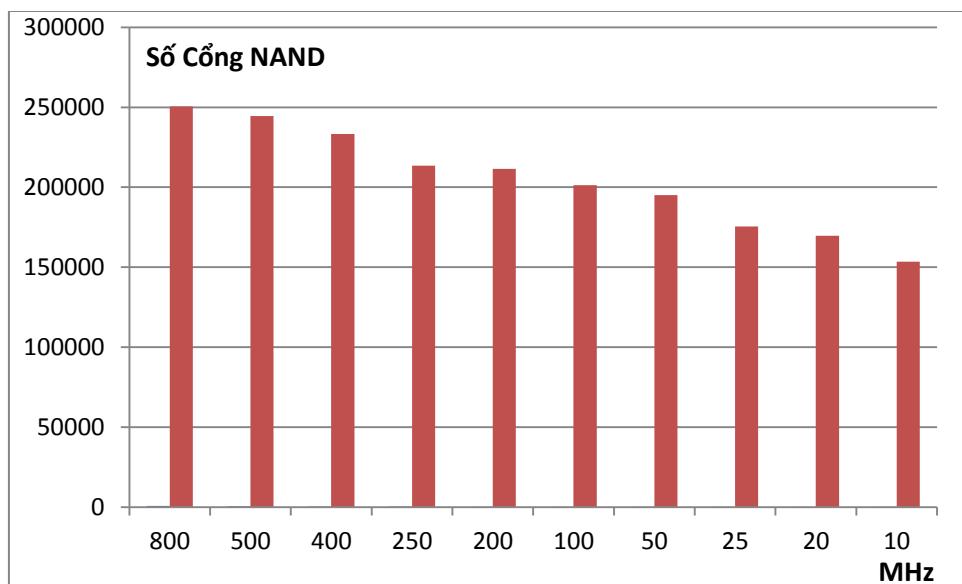
File tường trình sau cho biết công suất tiêu thụ của kiến trúc NoC 4x4 được chi tiết hóa.

```
*****
Report : power -analysis_effort low
Design : vr
*****
Library(s) Used:
scx2_tsmc_cl013g_tt_1p2v_25c (File:
/home/hieunguyen/TOOL/Library/01_TSMC130nm_frontend/sc-
x/synopsys/scx2_tsmc_cl013g_tt_1p2v_25c.db)
Operating Conditions: tt_1p2v_25c Library: scx2_tsmc_cl013g_tt_1p2v_25c
Wire Load Model Mode: top
Global Operating Voltage = 1.2
Power-specific unit information :
Voltage Units = 1V
Capacitance Units = 1.000000pf
Time Units = 1ns
Dynamic Power Units = 1mW (derived from V,C,T units)
Leakage Power Units = 1pW
Cell Internal Power = 504.4463 mW (100%)
Net Switching Power = 834.6915 uW (0%)
Total Dynamic Power = 505.2809 mW (100%)
Cell Leakage Power = 327.4334 uW
```

Phân tích sự tương quan giữa thời gian và diện tích cho thấy việc tăng tần số sẽ làm diện tích tăng lên cho thấy tốc độ và diện tích tỉ lệ nghịch đúng với quy luật thiết kế.

Bảng 4-3 Bảng khảo sát tương tác giữa tần số và diện tích

No.	Tần số (MHz)	Diện tích (Số Cổng NAND2)
1	800	250615
2	500	244515
3	400	233389
4	250	213448
5	200	211445
6	100	201288
7	50	195065
8	25	175410
9	20	169741
10	10	153429



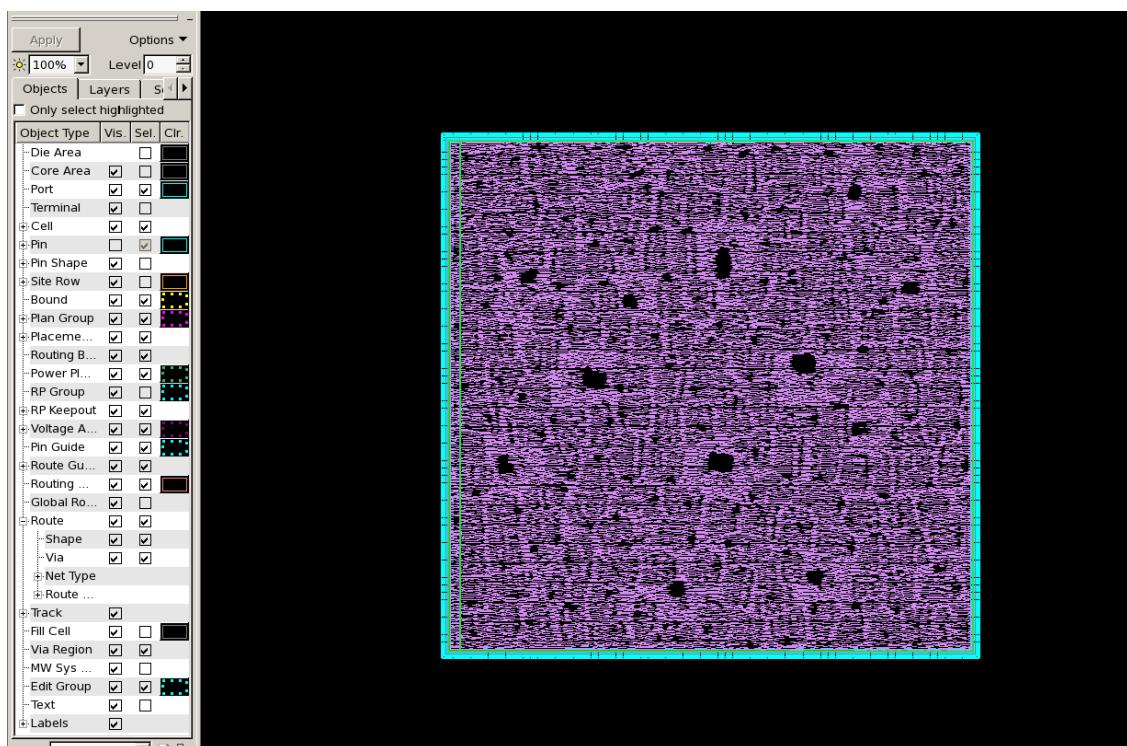
Hình 4.1 Biểu đồ diện tích NoC 4x4 khi thay đổi tần số tổng hợp

Bảng 4-3 và Hình 4.1 mô tả chi tiết kết quả tổng hợp xuống lớp công với các tần số tương ứng khác nhau. Kết quả khảo sát trên biểu đồ cho thấy việc ánh hưởng qua lại giữa diện tích và tần số. Cụ thể khi tăng tần số mong muốn trong quá trình tổng hợp, chỉ số diện tích thay đổi ở những thời điểm cụ thể. Khi cân bằng giữa tần số và diện tích không được kiểm soát (chỉ số slack bé hơn không), để có thể đạt được yếu tố cân bằng này bắt buộc phải sửa đổi lại thiết kế ở cấp độ thanh ghi (RTL). Một số phương pháp cải thiện được liệt kê:

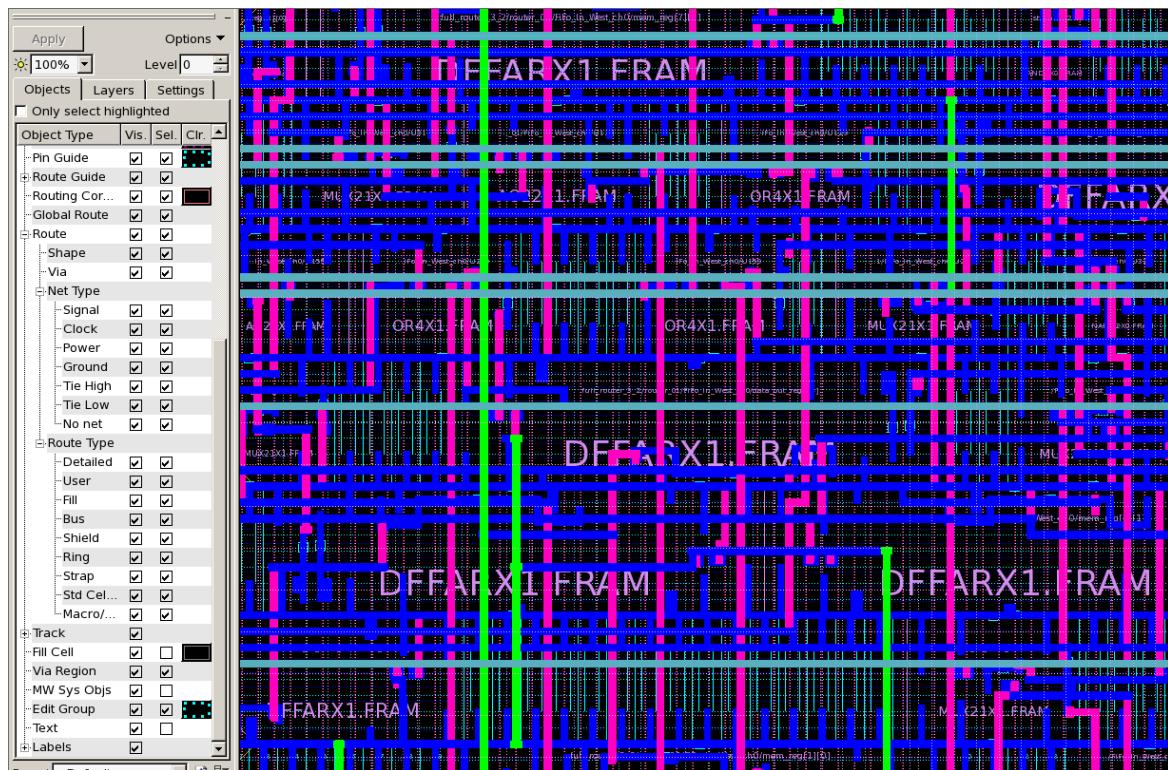
- Thay đổi giải thuật thiết kế sao cho việc trì hoãn trên các đường dữ liệu là thấp nhất có thể (bước thiết kế RTL).
- Chèn thêm các DFF vào các đường dữ liệu nhằm chia nhỏ trì hoãn trên đường dữ liệu thành các trì hoãn nhỏ hơn nhằm thỏa mãn tần số thiết kế.
- Can thiệp vào các cell đệm ở bước thiết kế vật lý (bước P&R) nhằm tăng tốc các cell thành phần nằm trên đường truyền dữ liệu.

4.2 Kết quả đặt và đi dây cho thiết kế

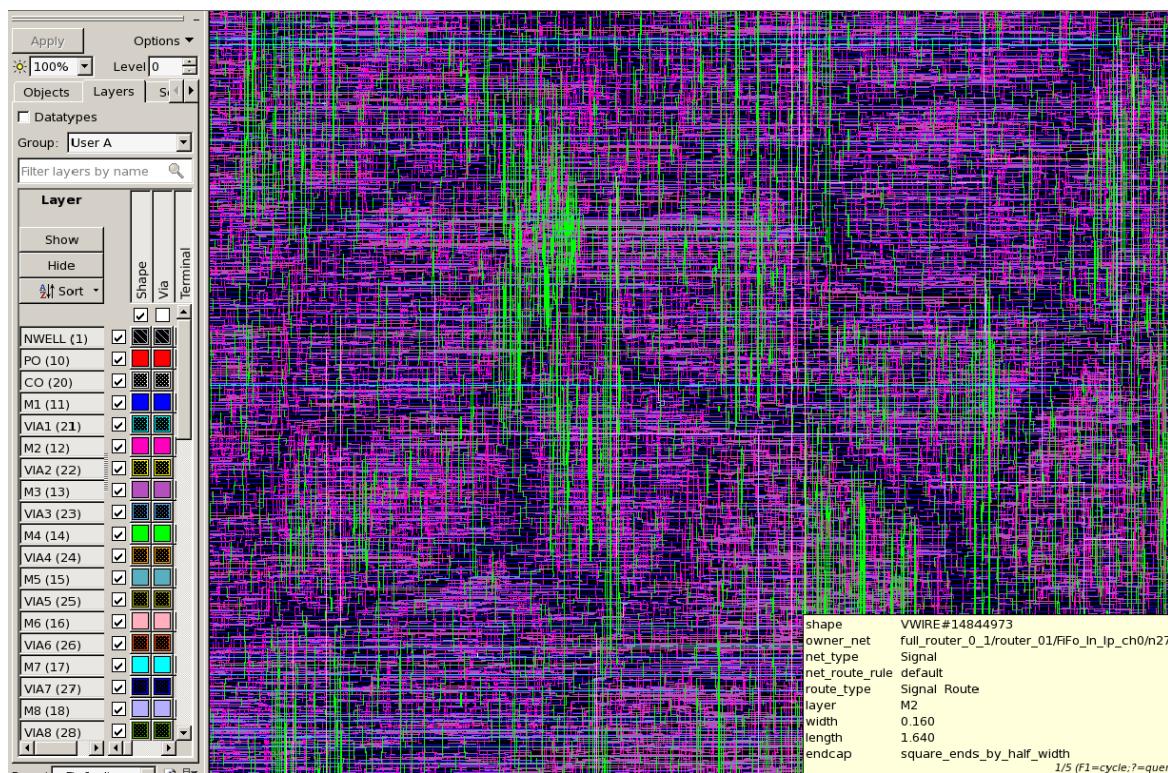
Thiết kế ở mức công sau khi được đảm bảo chính xác bởi các kiểm tra cần thiết, toàn bộ thiết kế được thực hiện đặt và đi dây nhằm tạo thiết kế vật lý cuối cùng cho sản xuất. Các hình vẽ sau cho thấy các kết quả của việc đặt và đi dây.



Hình 4.2 Kết quả đặt và đi dây toàn bộ chip



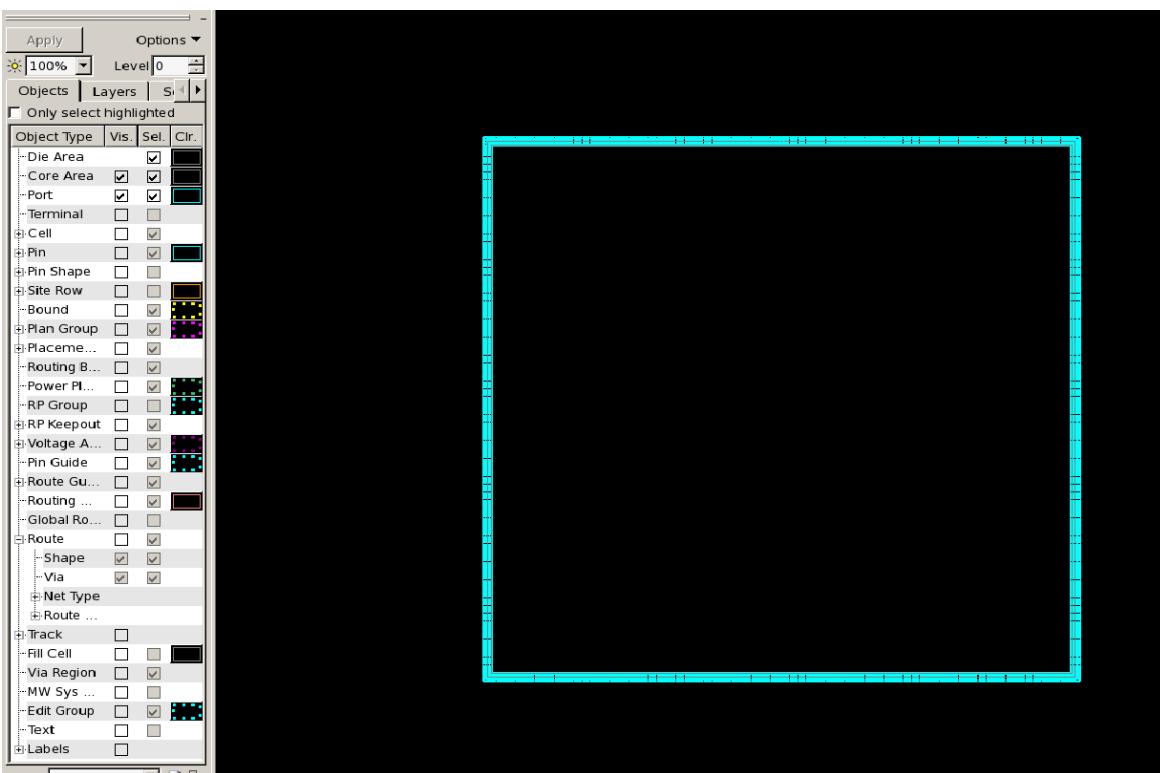
Hình 4.3 Kết quả đặt và đi dây toàn bộ chip (chi tiết)



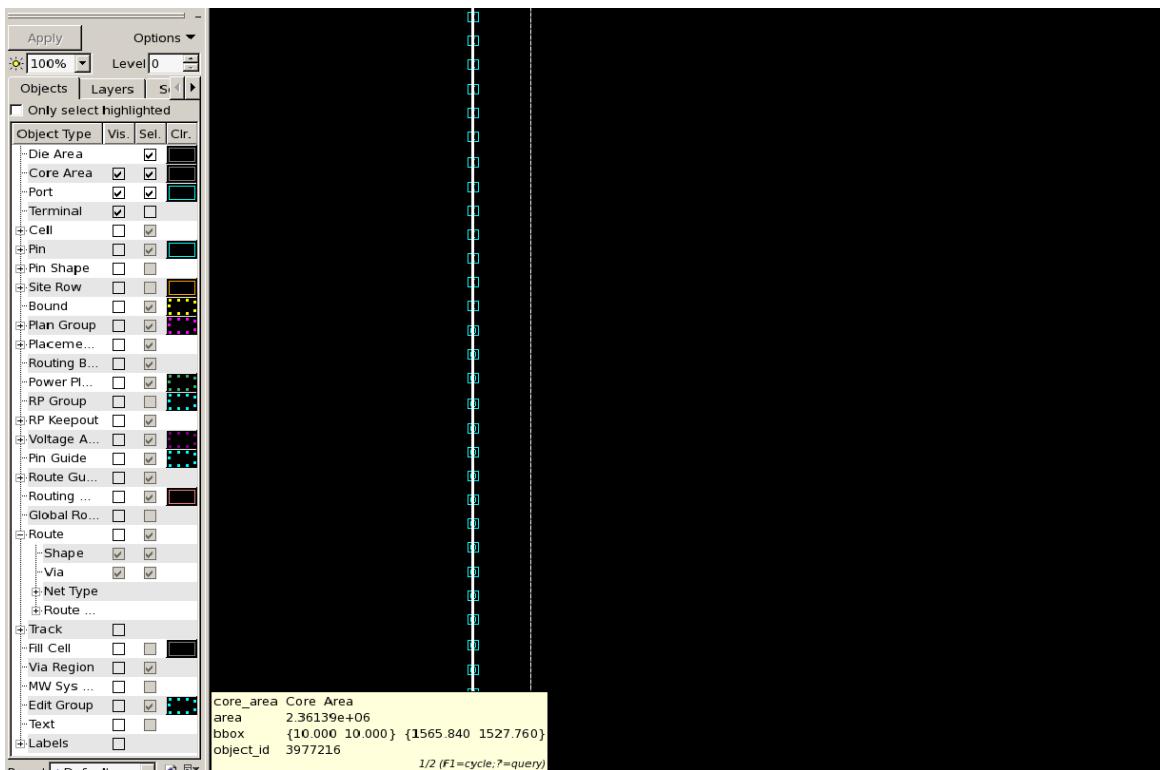
Hình 4.4 Kết quả đặt và đi dây tắt cả lớp metal



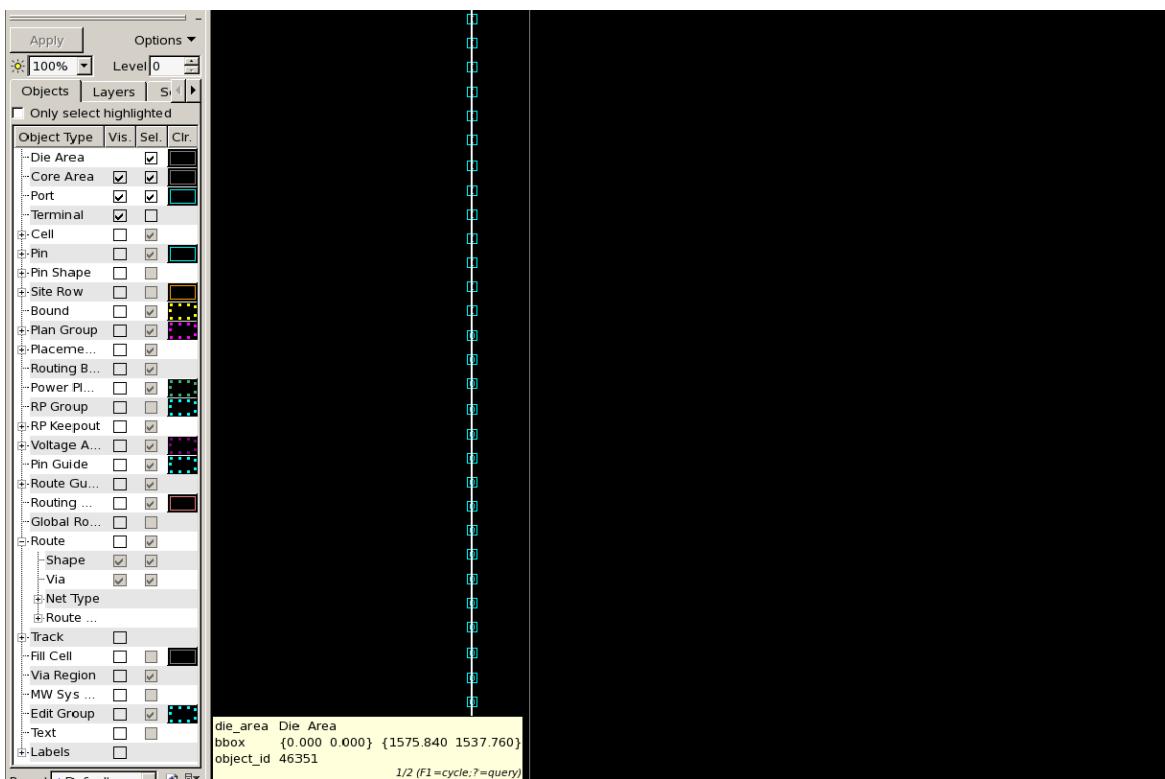
Hình 4.5 Kết quả đặt và đi dây Core Area



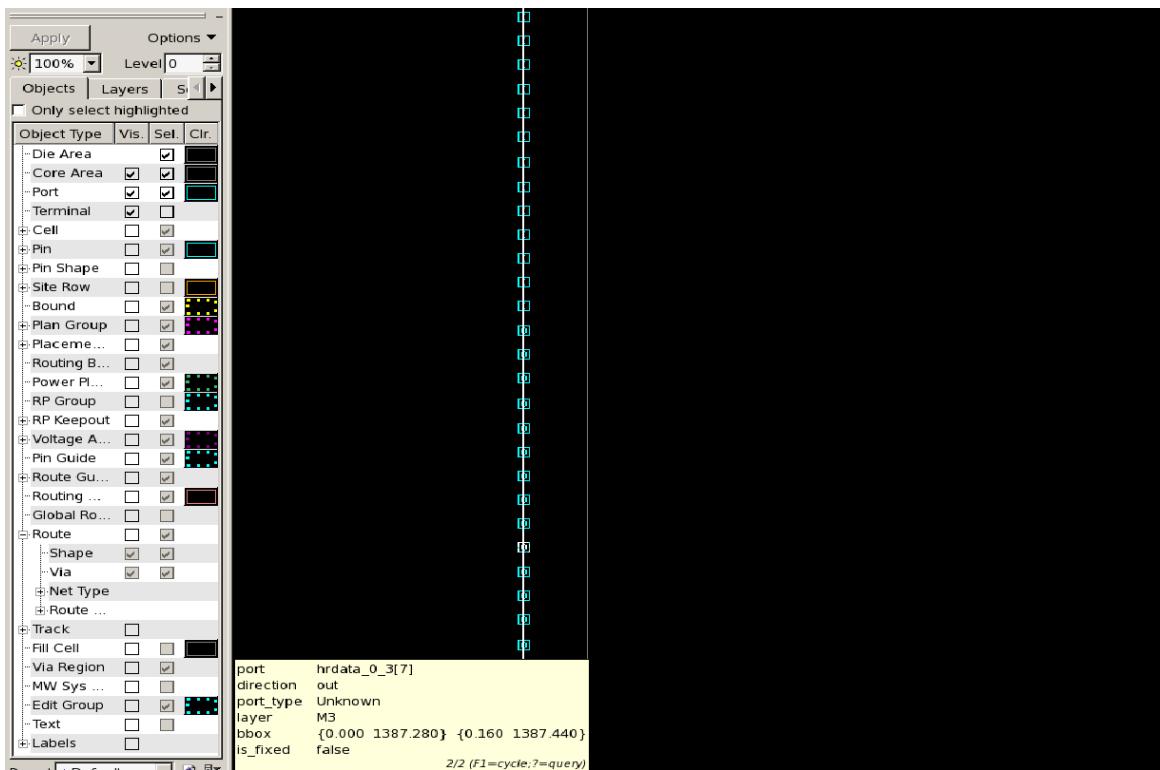
Hình 4.6 Kết quả đặt và đi dây Core Area_Port



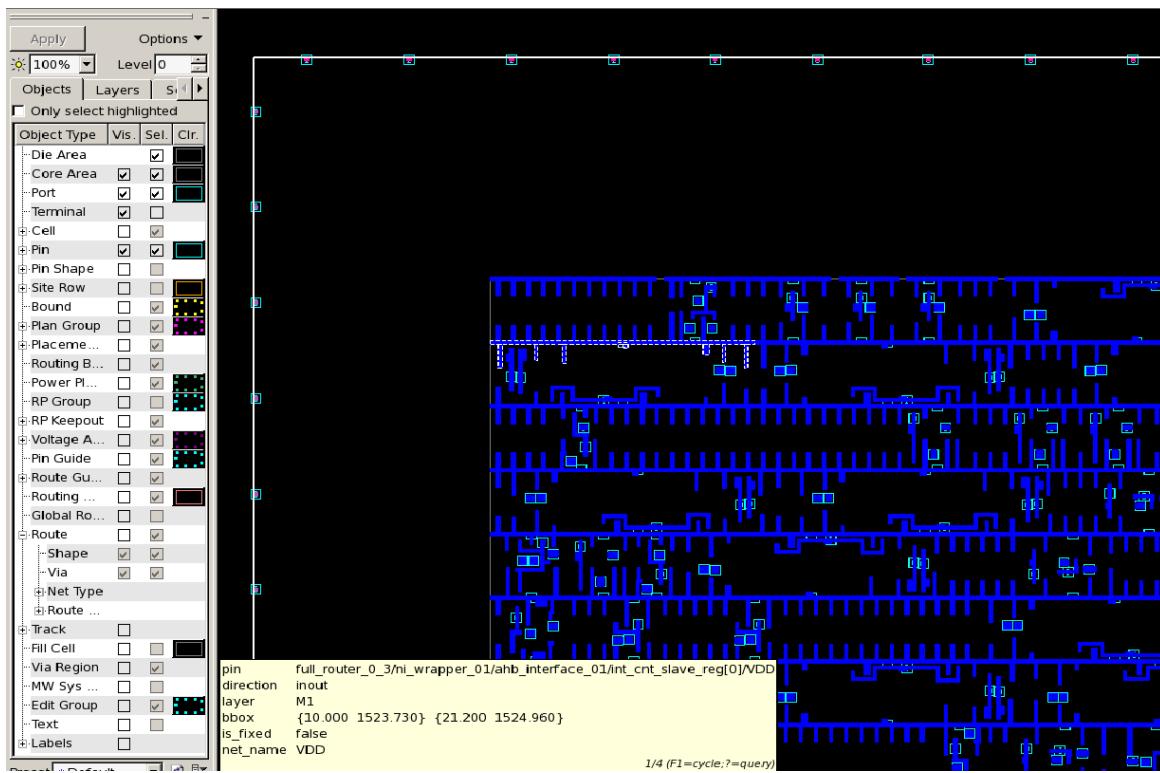
Hình 4.7 Kết quả đặt và di dây Core Area_Port_Terminal



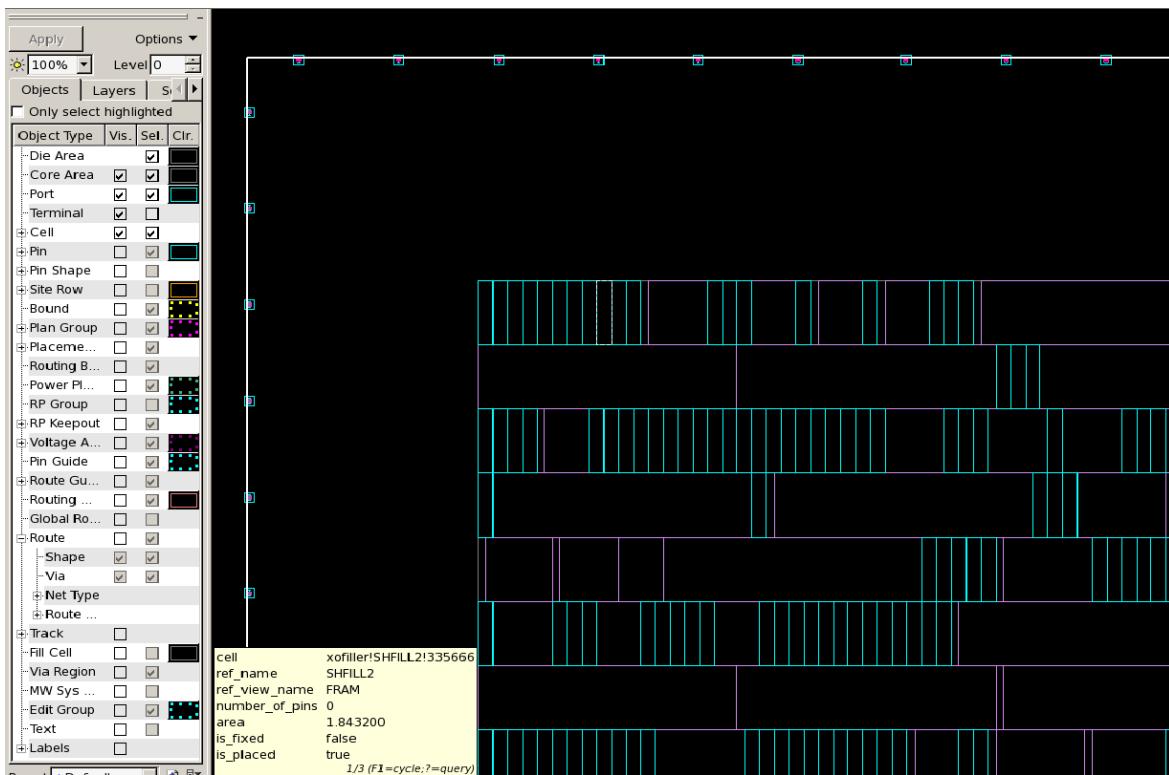
Hình 4.8 Kết quả đặt và di dây Die Area



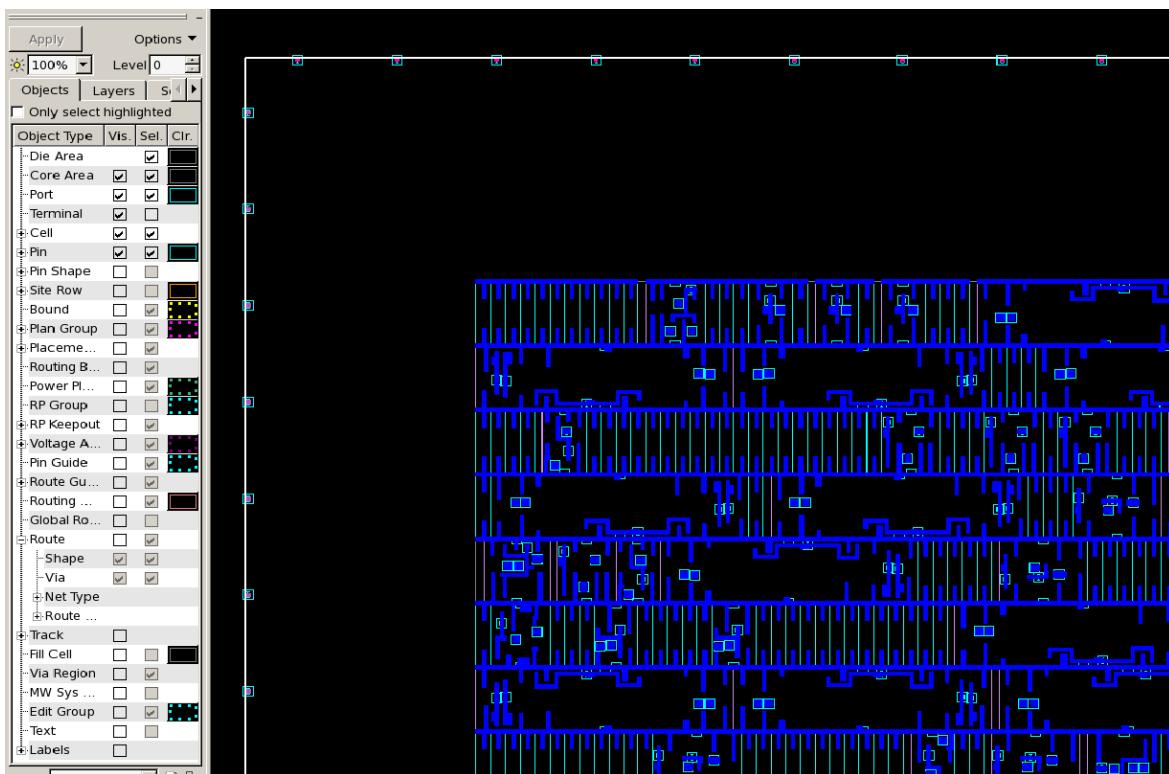
Hình 4.9 Kết quả đặt và đi dây Port hrdata



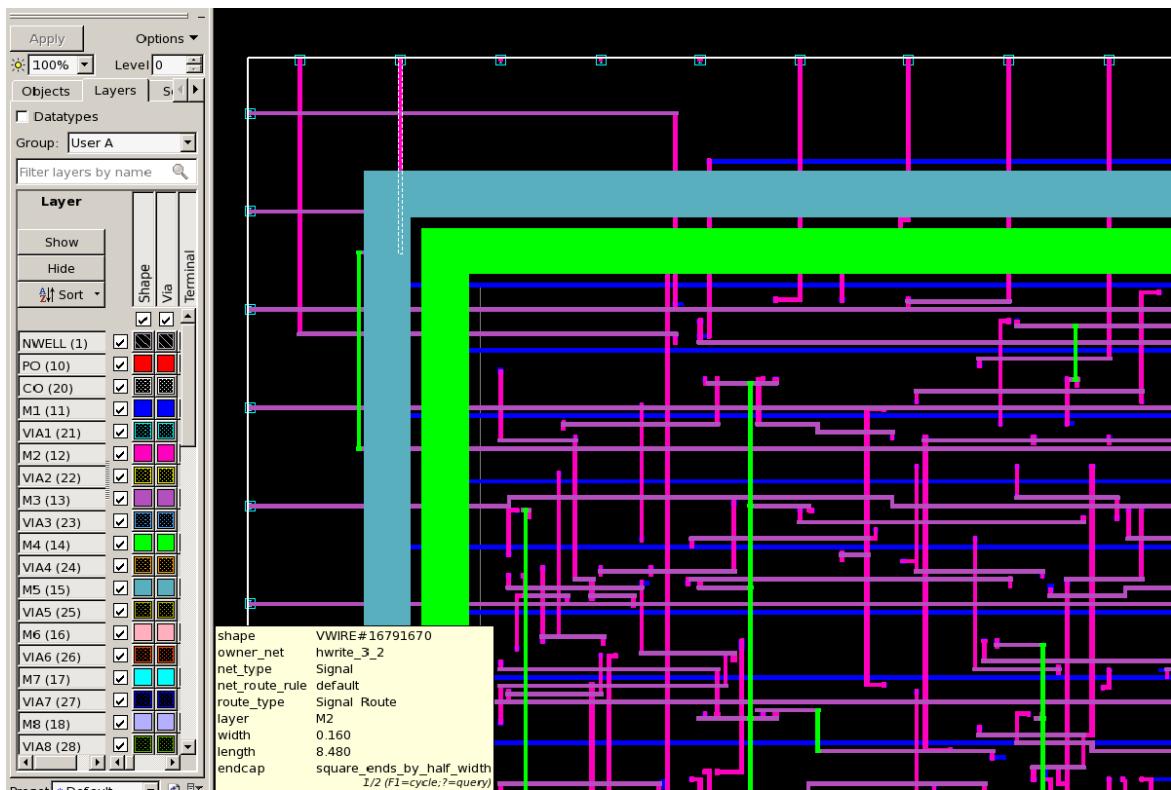
Hình 4.10 Kết quả đặt và đi dây Core_Port_Pin



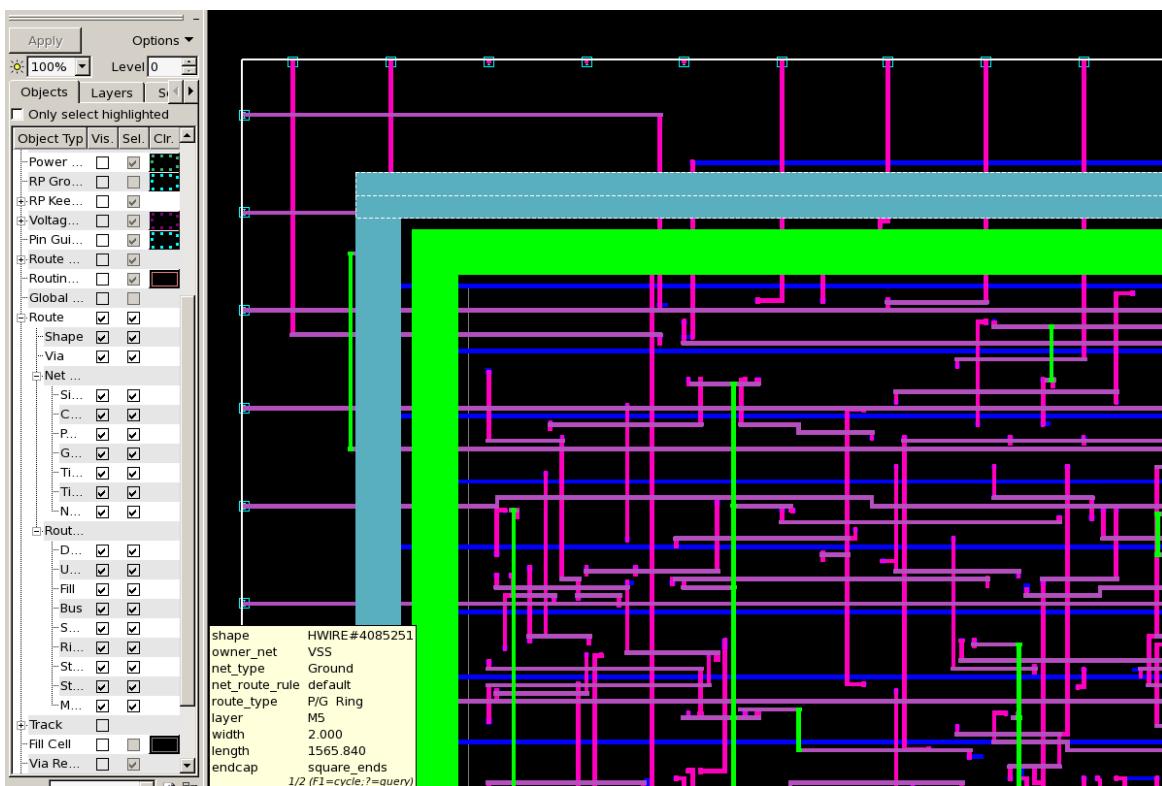
Hình 4.11 Kết quả đặt và đi dây Core_Port_Cell



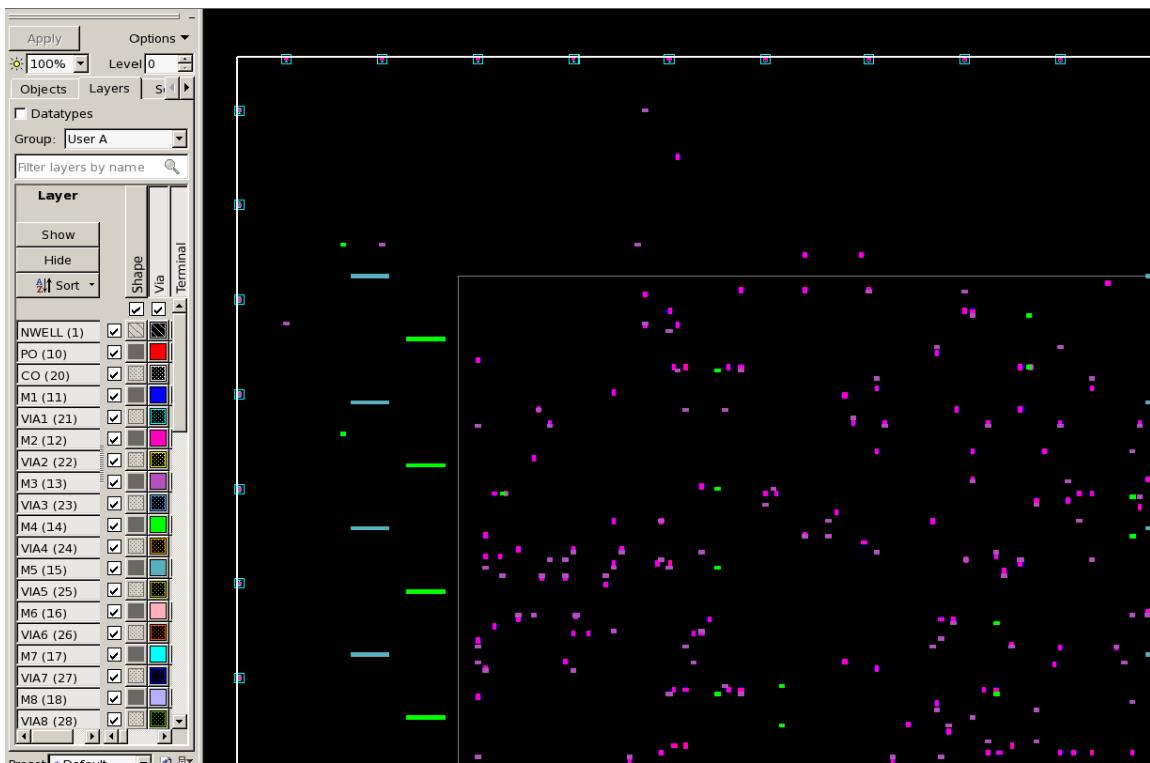
Hình 4.12 Kết quả đặt và đi dây Core_Port_Pin_Cell



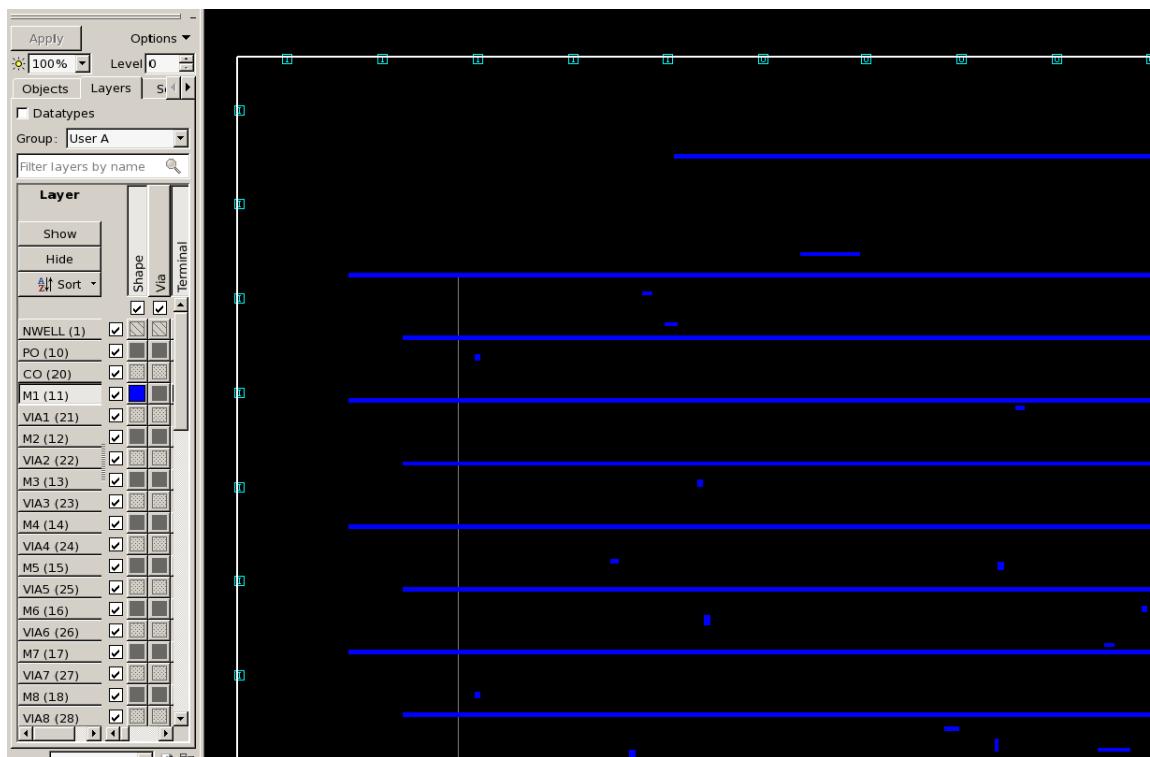
Hình 4.13 Kết quả đặt và đi dây Core_Port_All power signals



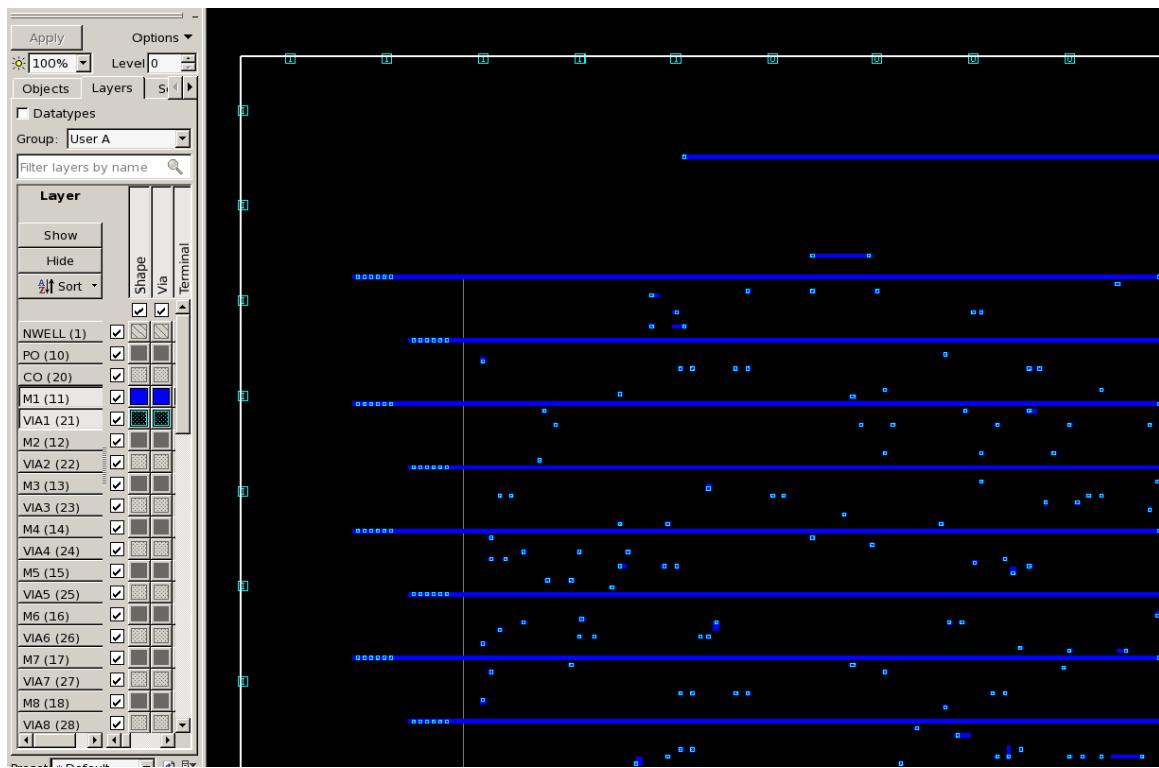
Hình 4.14 Kết quả đặt và đi dây Core_Port_All metal



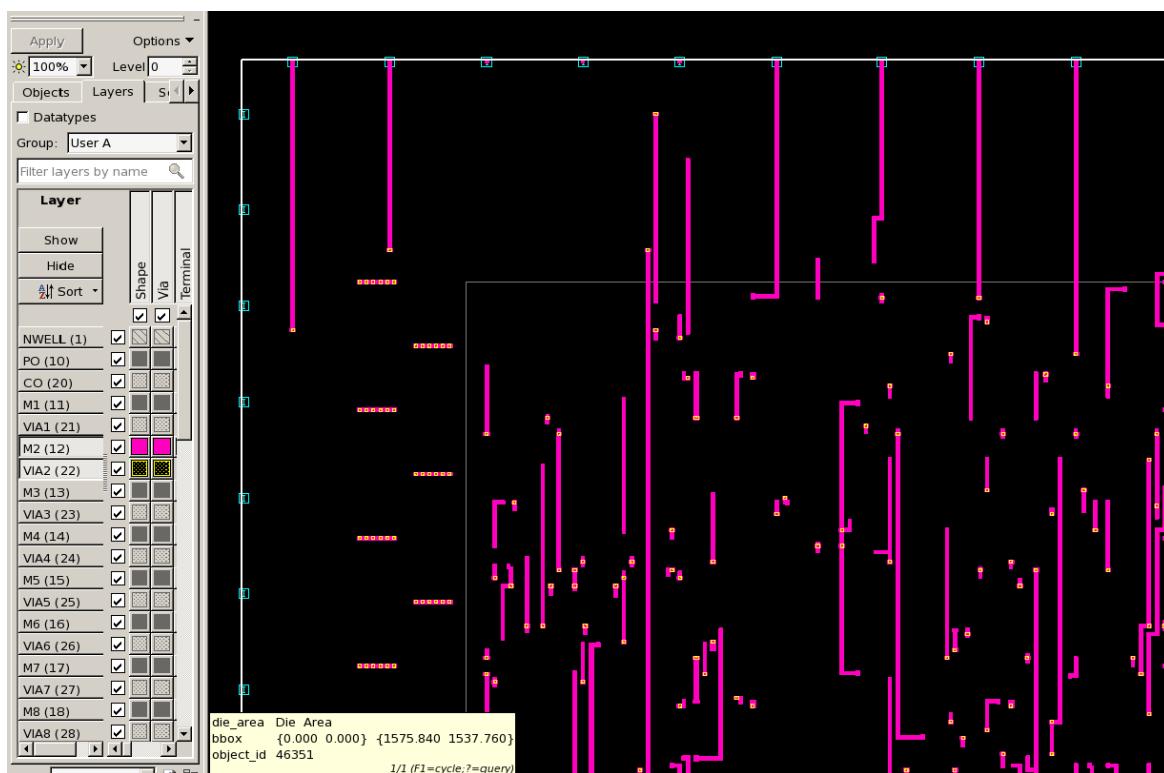
Hình 4.15 Kết quả đặt và di dây Via tắt cả Metal



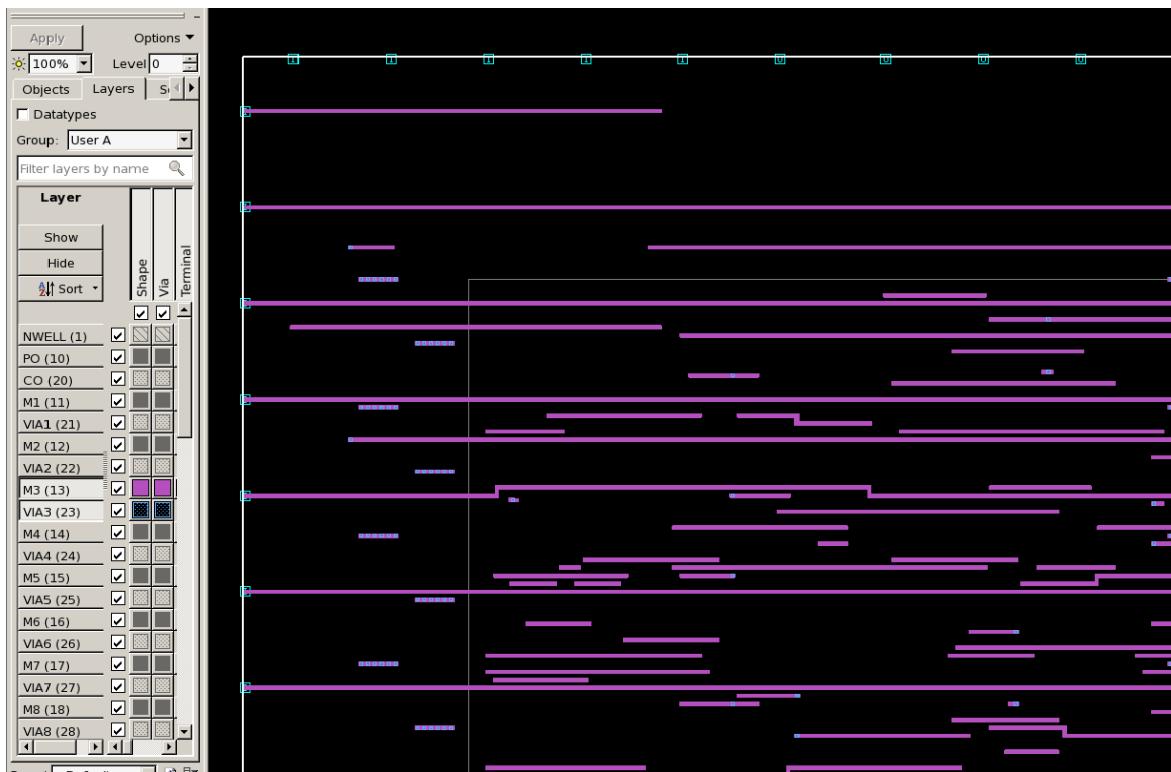
Hình 4.16 Kết quả đặt và di dây lớp Metal 1



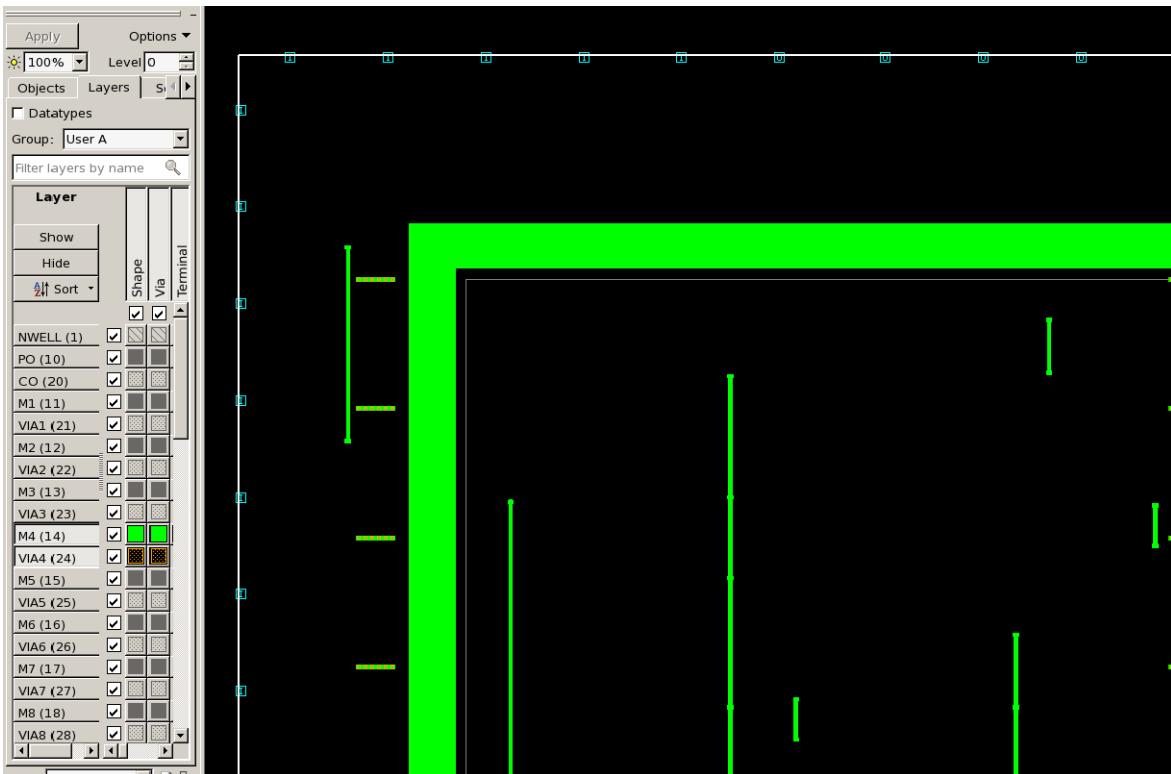
Hình 4.17 Kết quả đặt và đi dây Metal 1_Via 1



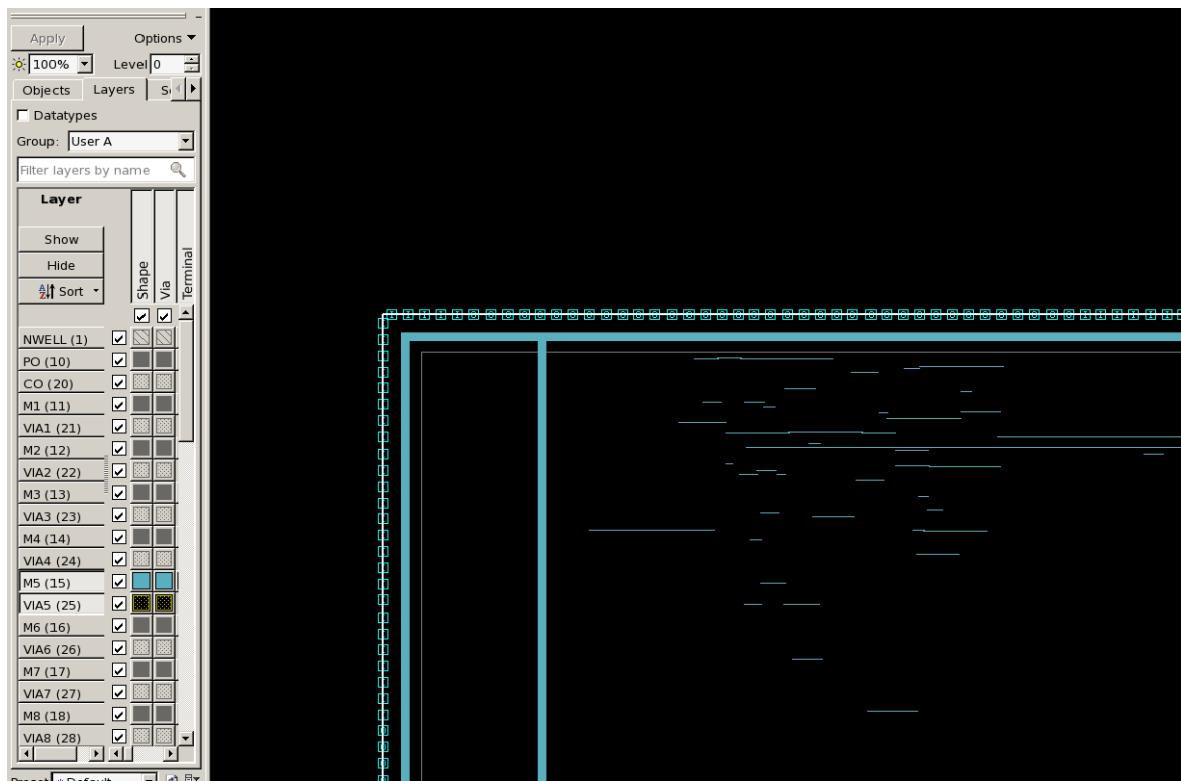
Hình 4.18 Kết quả đặt và đi dây Metal 2_Via 2



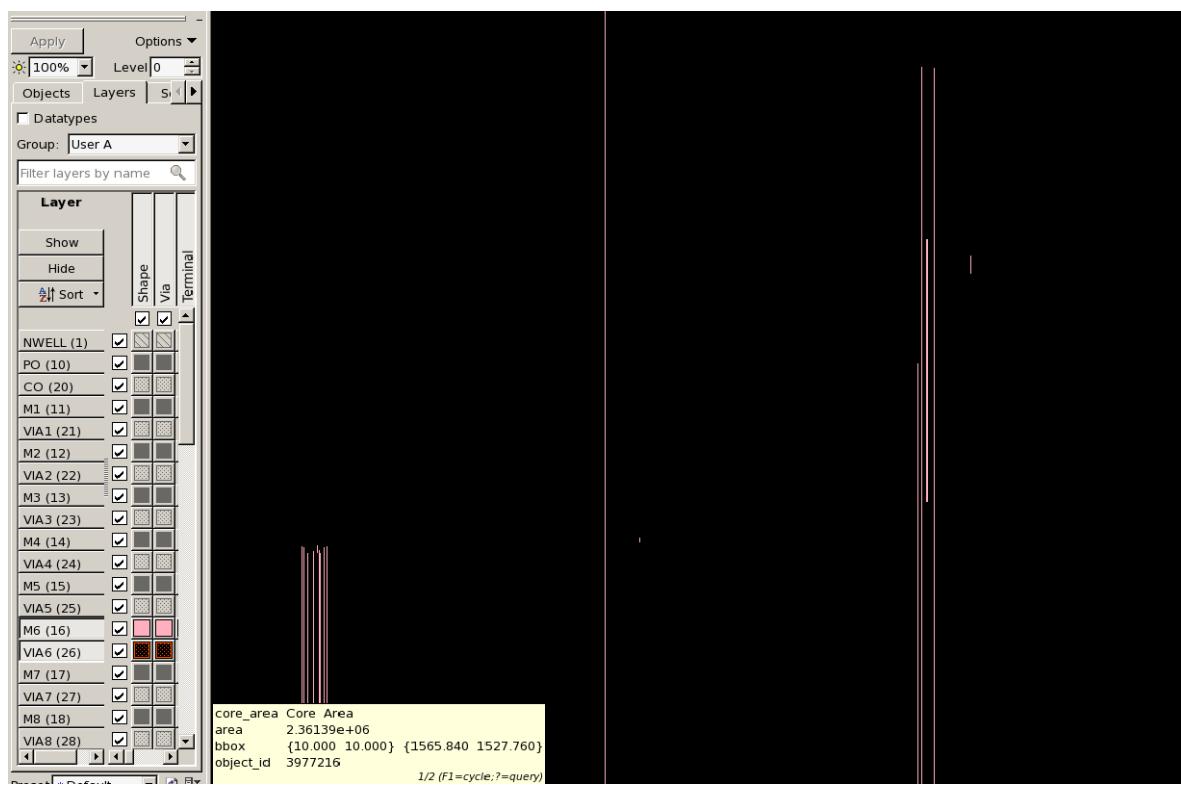
Hình 4.19 Kết quả đặt và đi dây Metal 3_Via 3



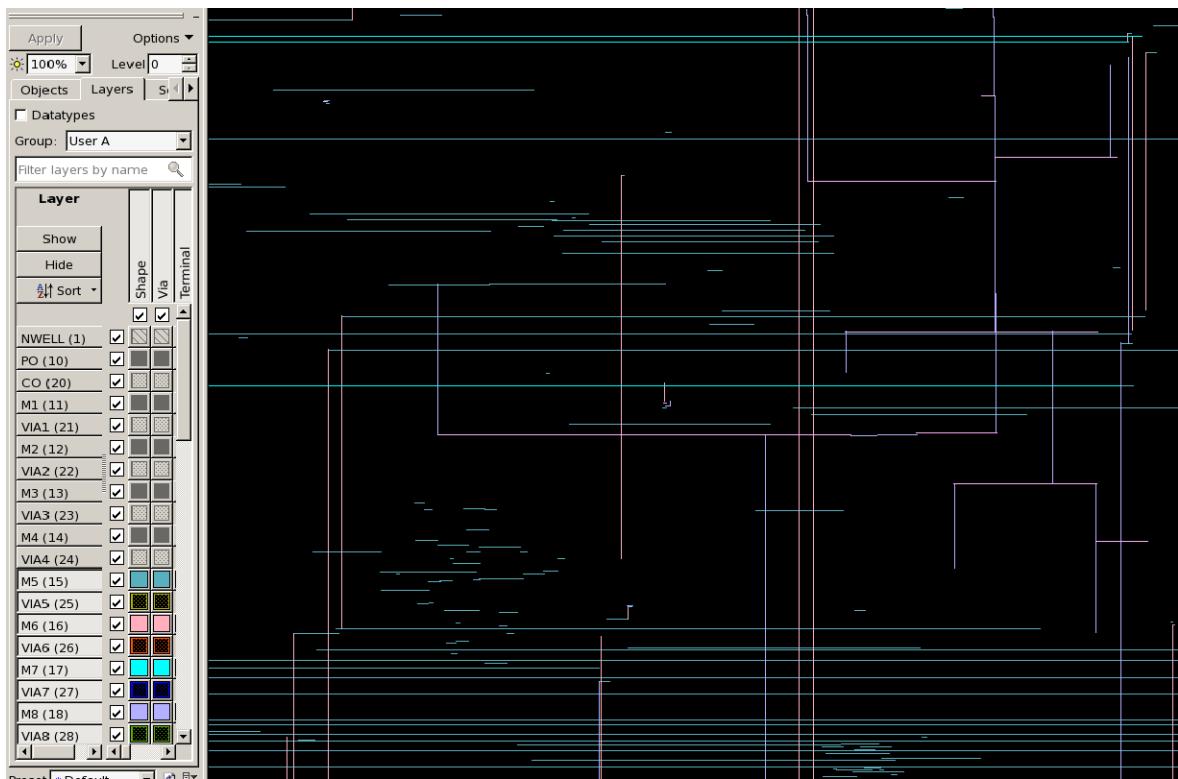
Hình 4.20 Kết quả đặt và đi dây Metal 4_Via 4



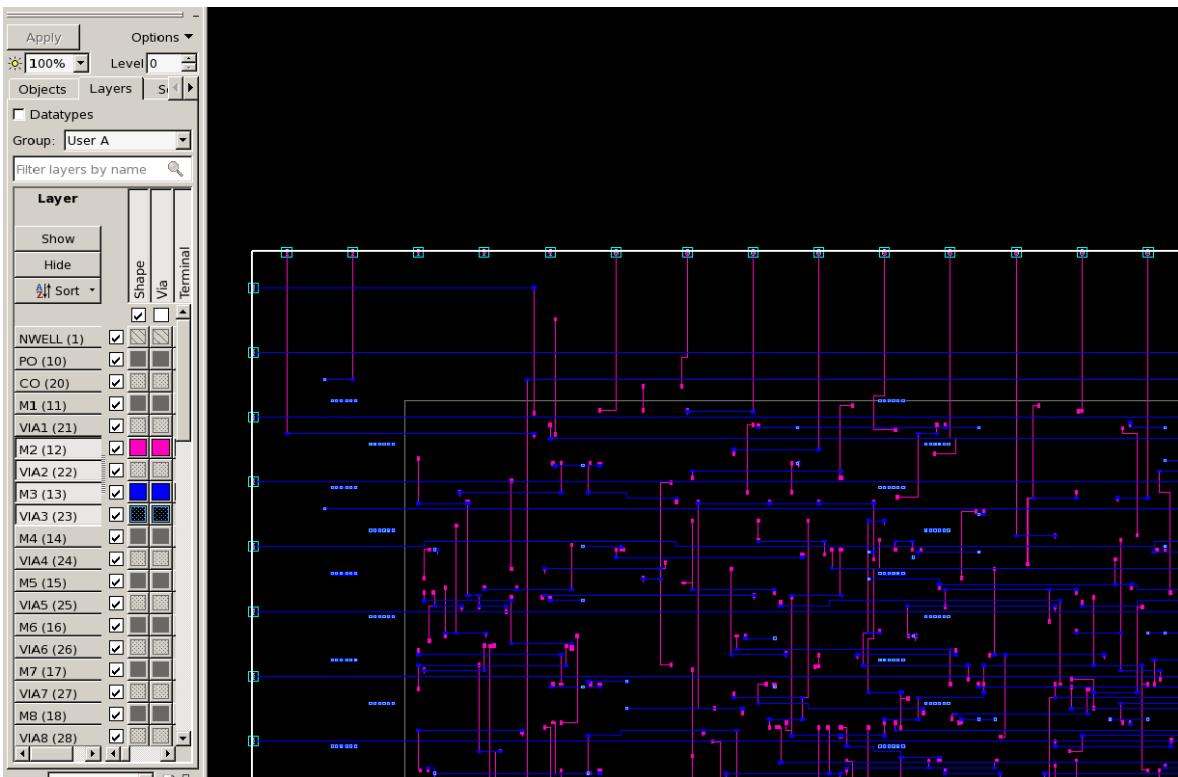
Hình 4.21 Kết quả đặt và đi dây Metal 5_Via 5



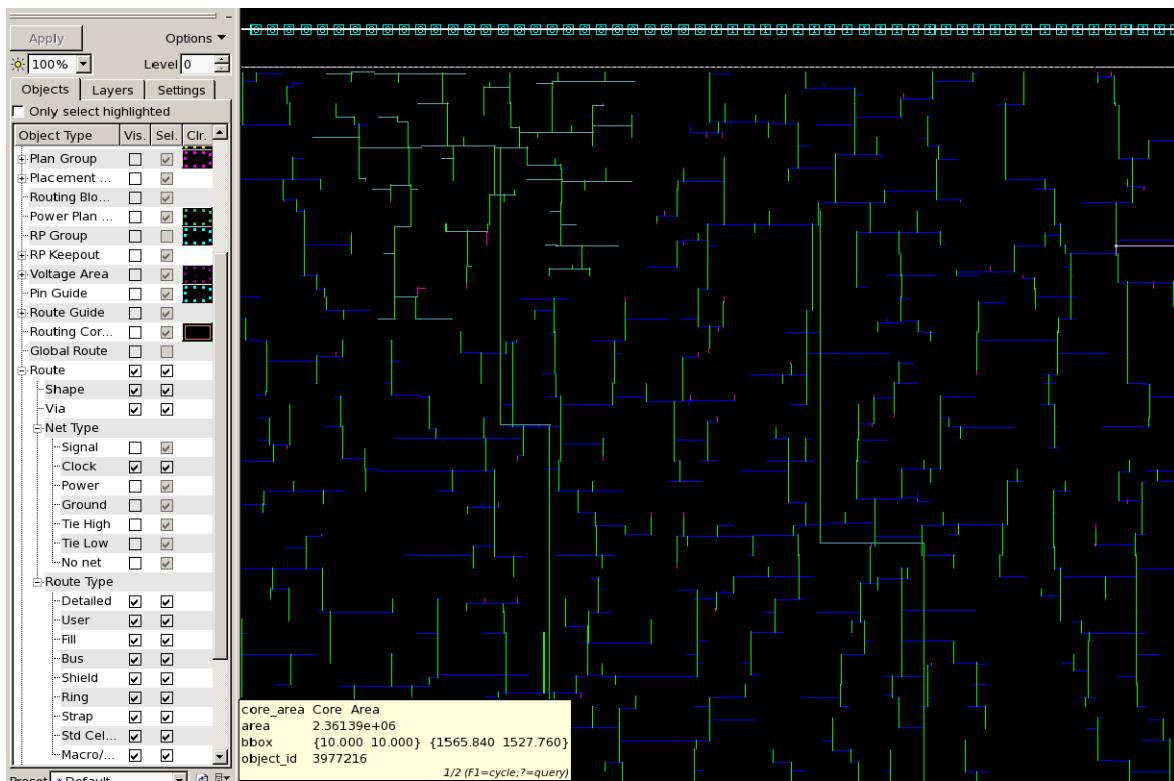
Hình 4.22 Kết quả đặt và đi dây Metal 6_Via 6



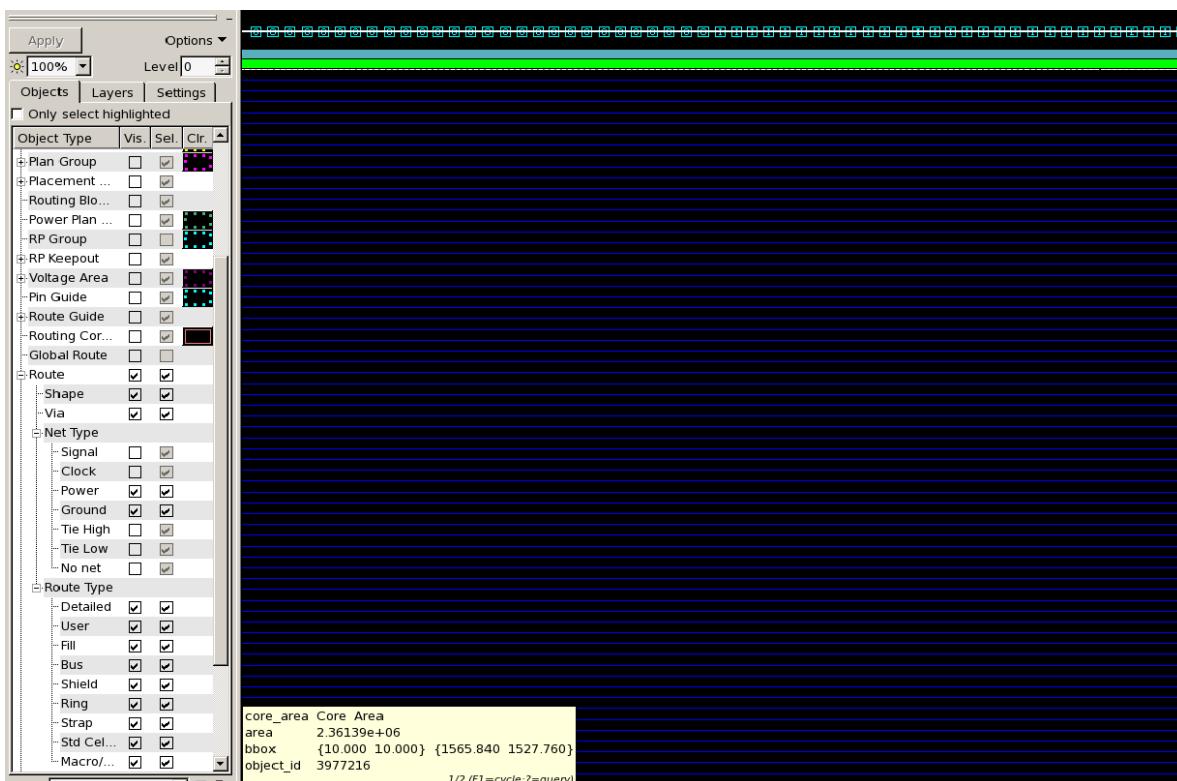
Hình 4.23 Kết quả đặt và đi dây từ Metal 6 đến Metal 9



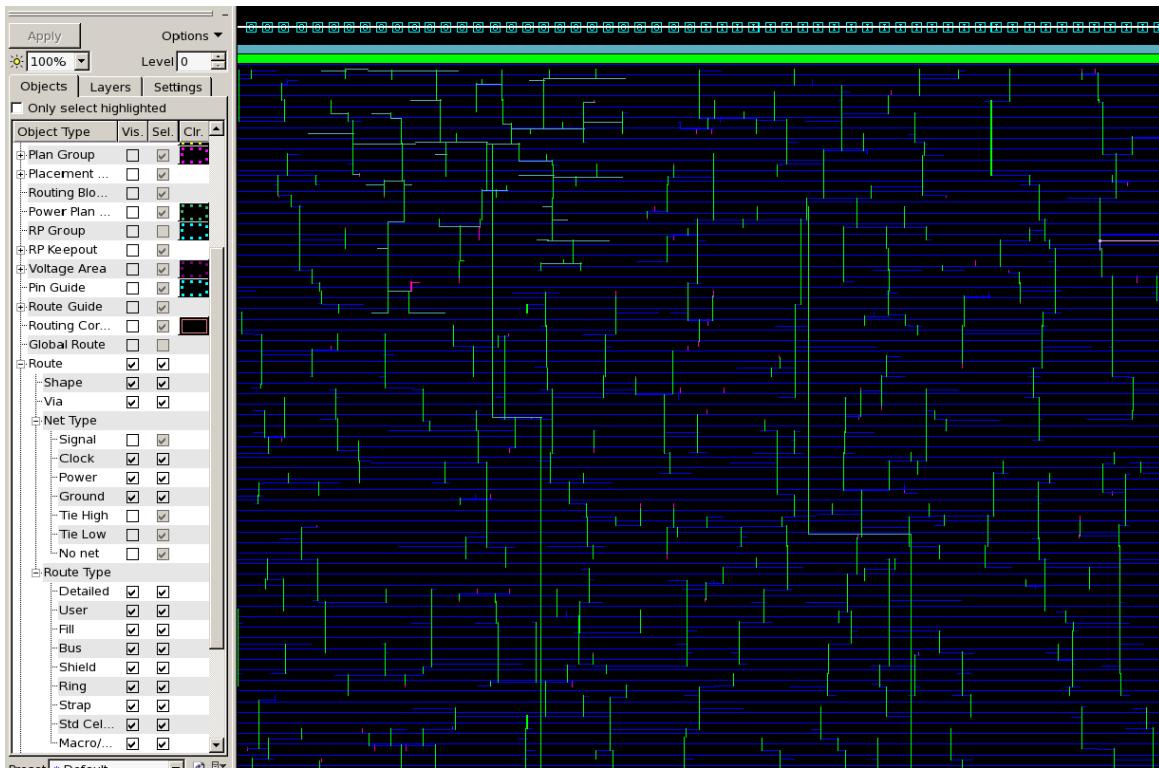
Hình 4.24 Kết quả đặt và đi dây Metal 2 và Metal 3



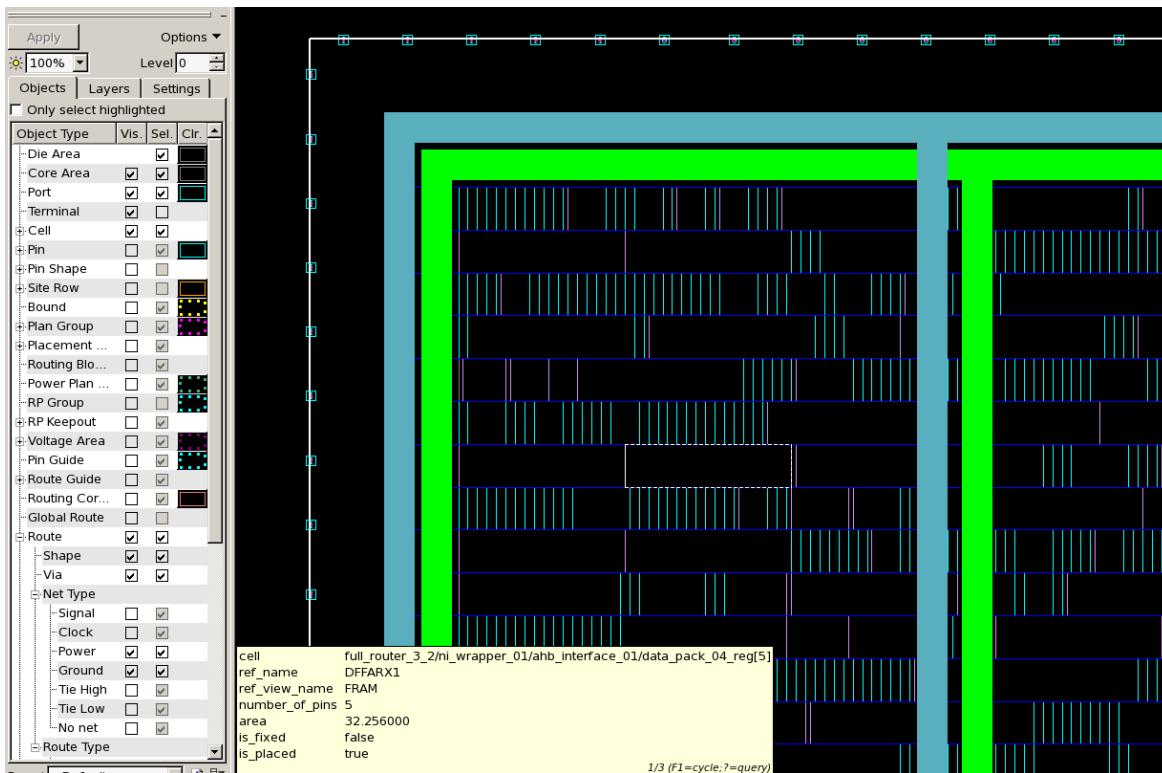
Hình 4.25 Kết quả đặt và đi dây Clock



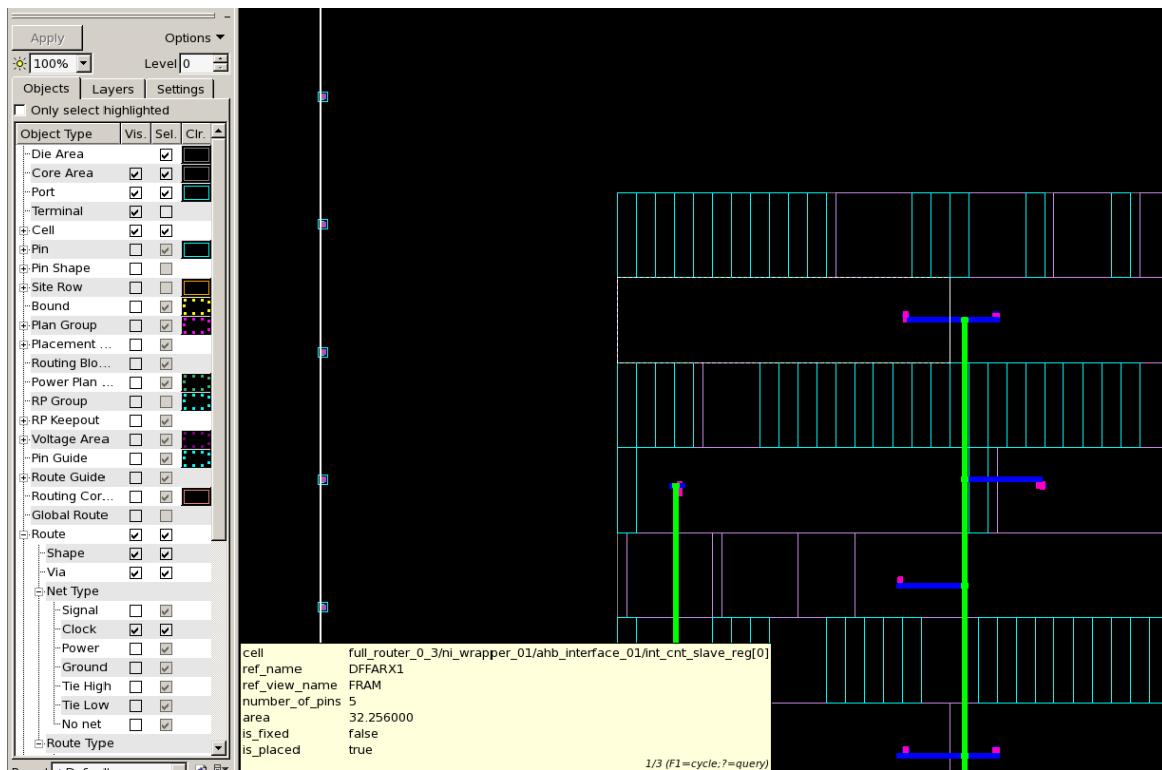
Hình 4.26 Kết quả đặt và đi dây VDD và VSS



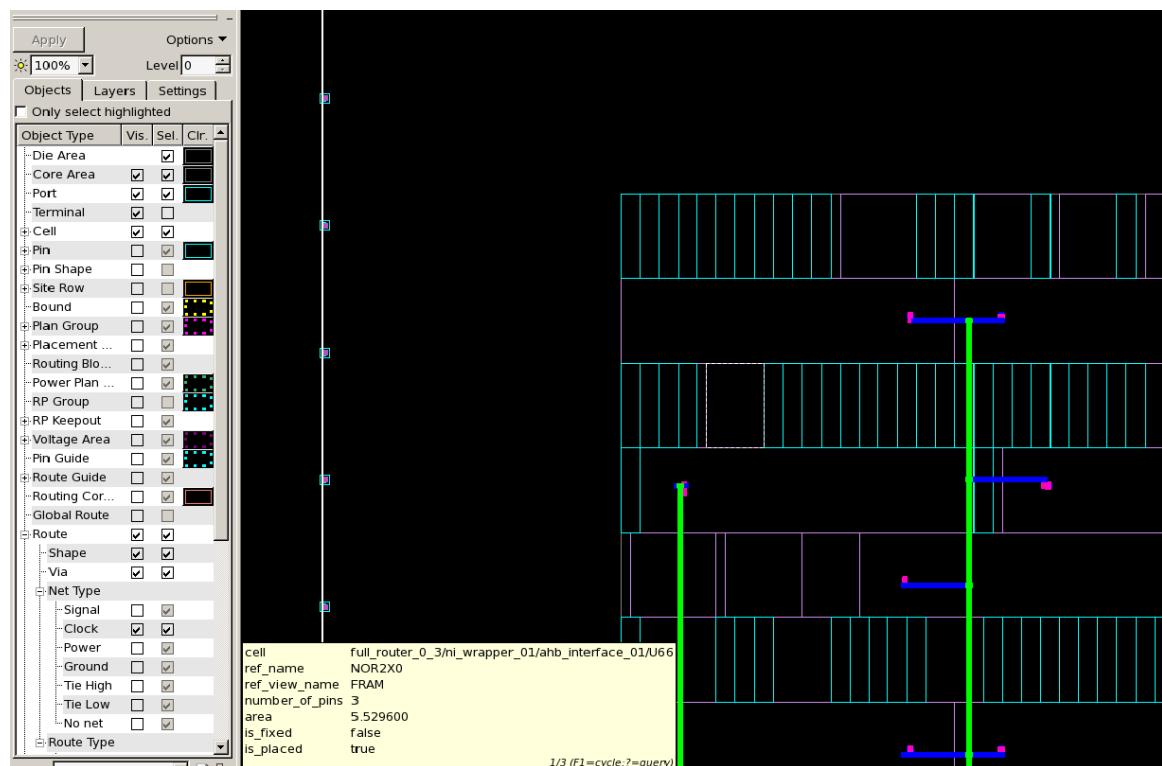
Hình 4.27 Kết quả đặt và đi dây Power_Ground_Clock



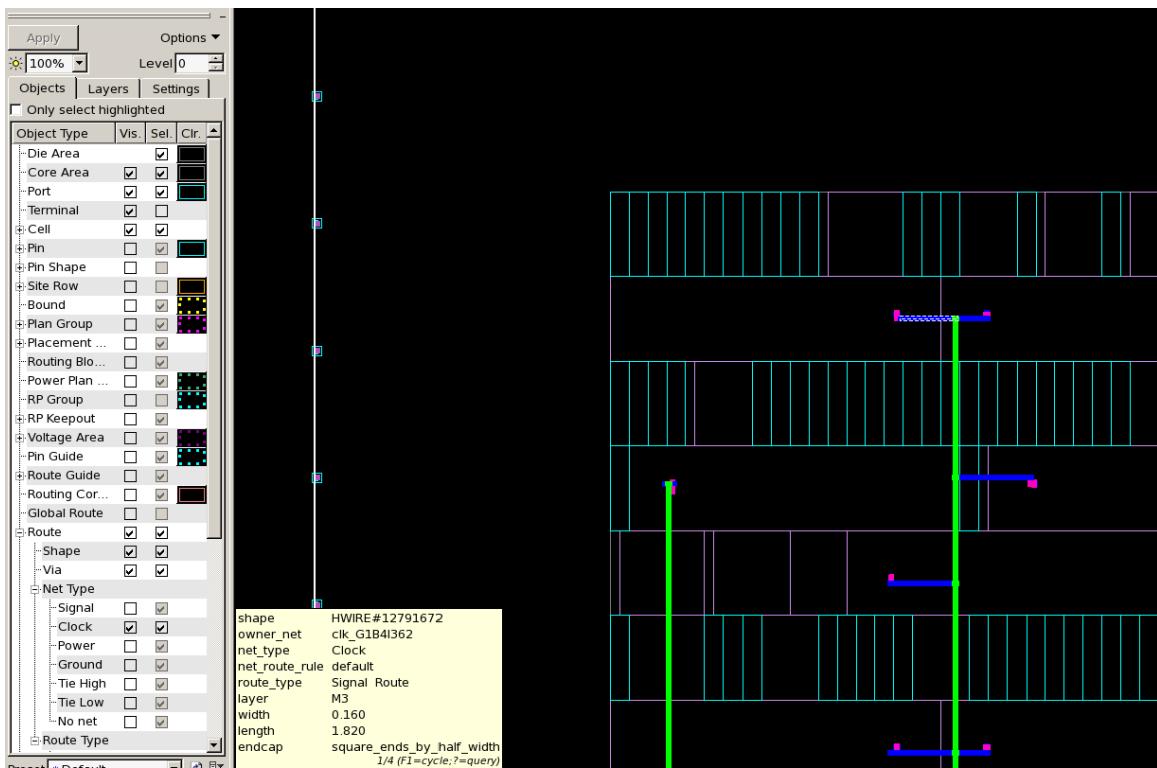
Hình 4.28 Kết quả đặt và đi dây Power và Cell



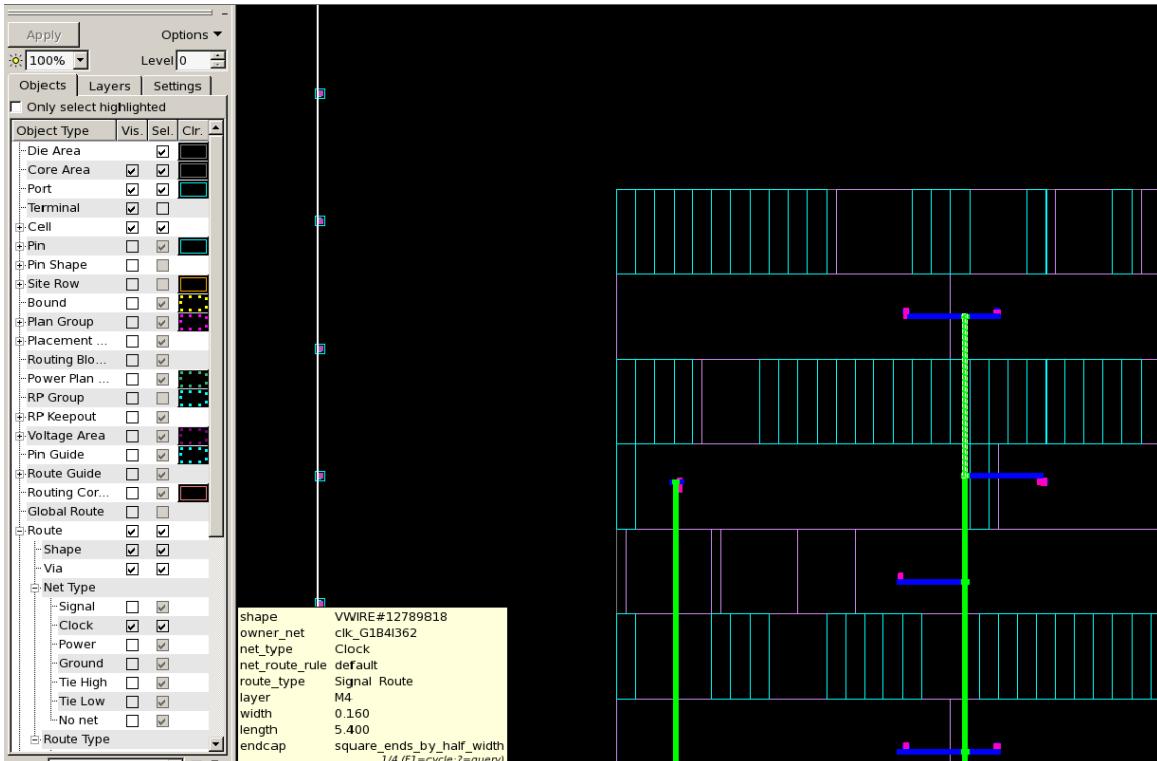
Hình 4.29 Kết quả đặt và đi dây Cell và Clock tree



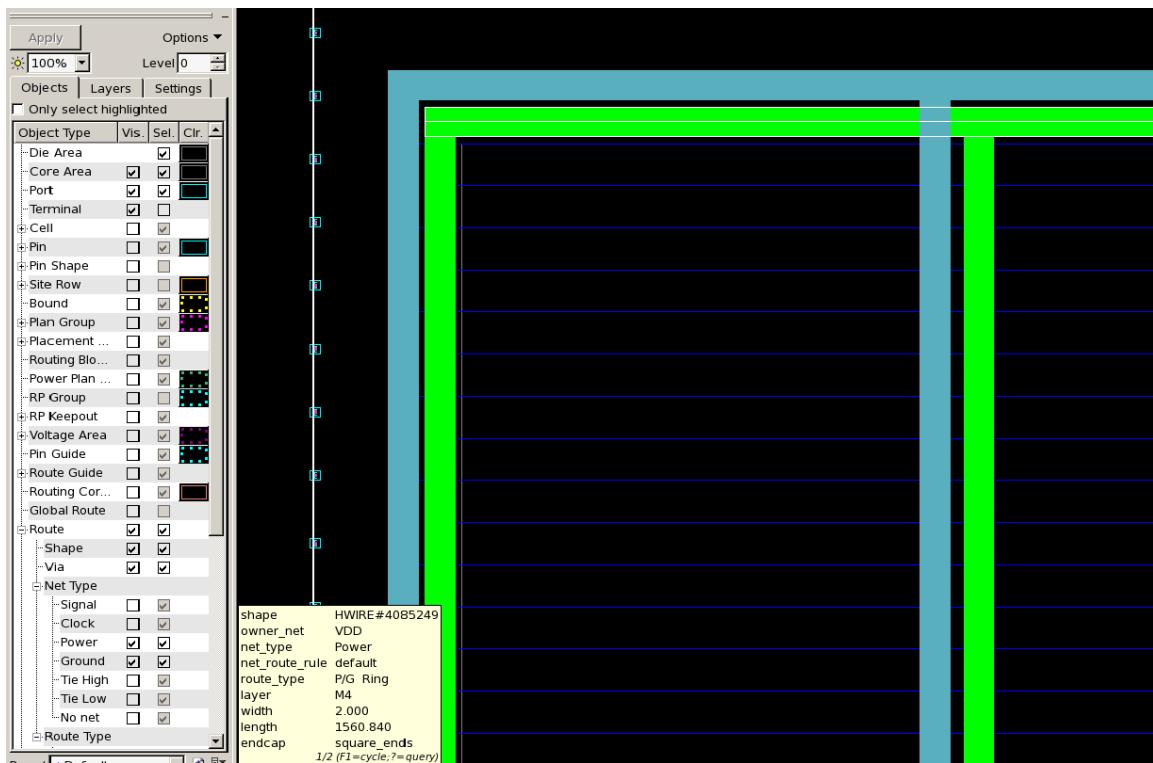
Hình 4.30 Kết quả đặt và đi dây Cell Combination



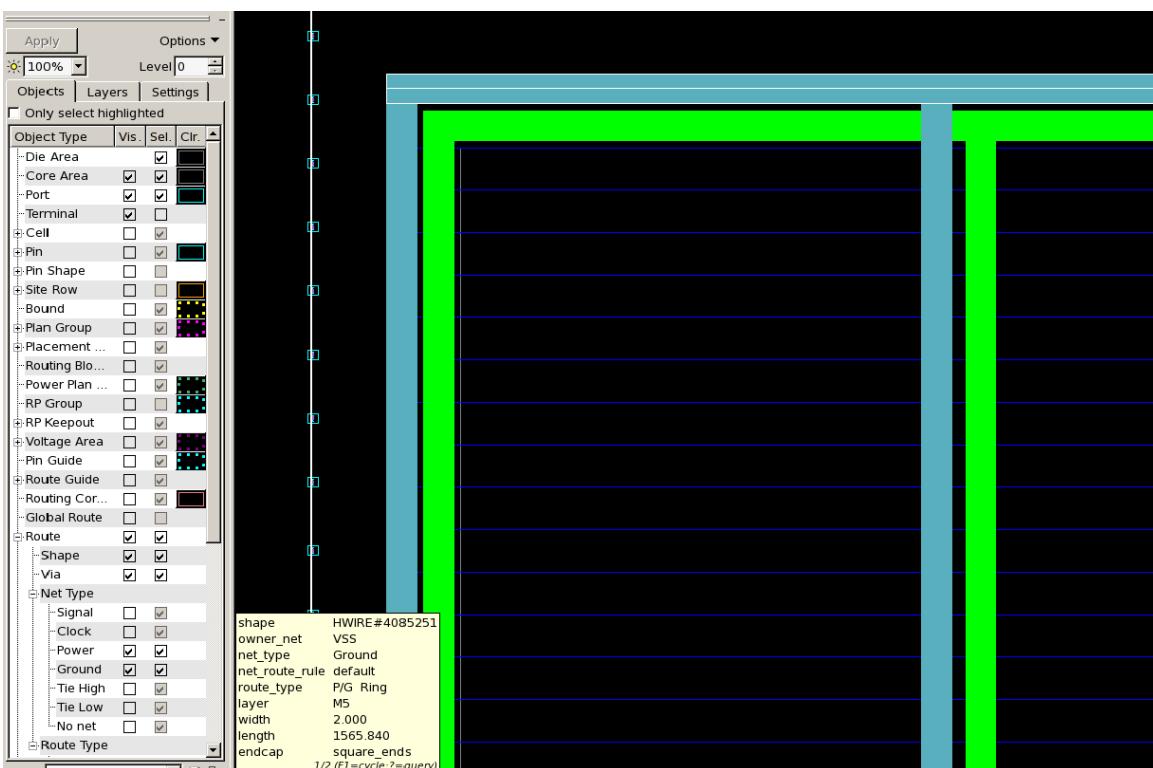
Hình 4.31 Kết quả đặt và đi dây Clock net



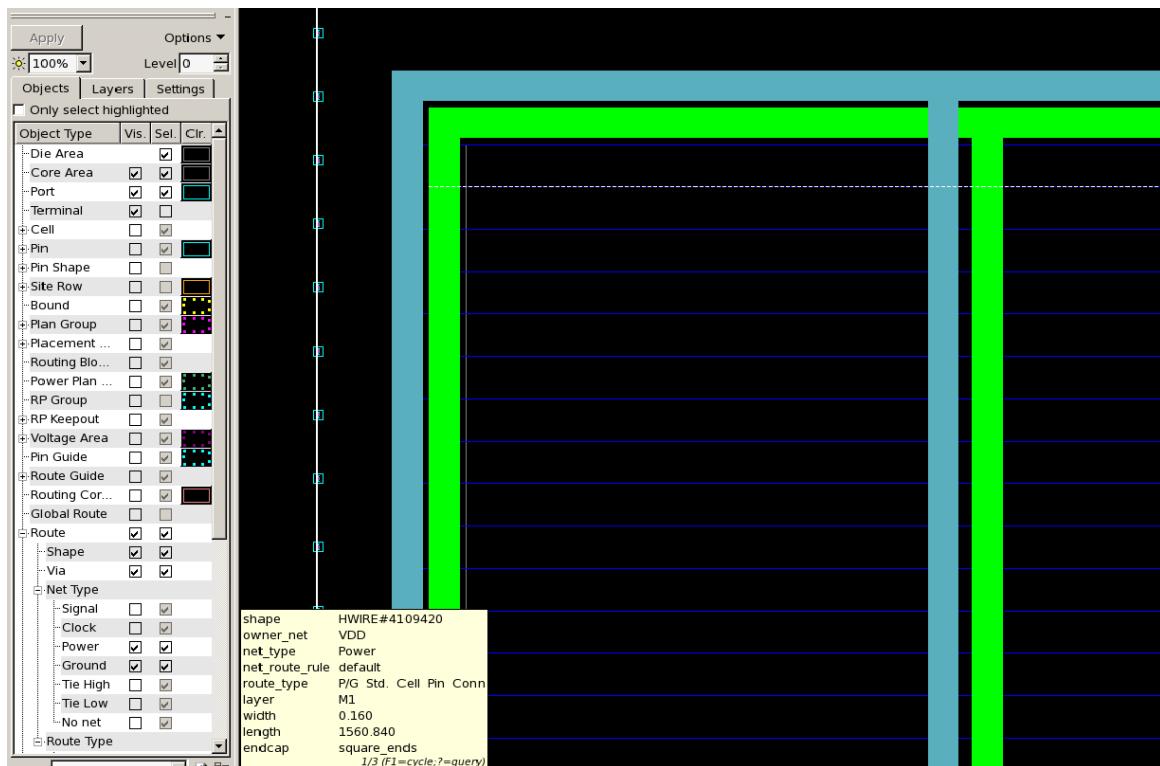
Kết quả đặt và đi dây Clock net (tt)



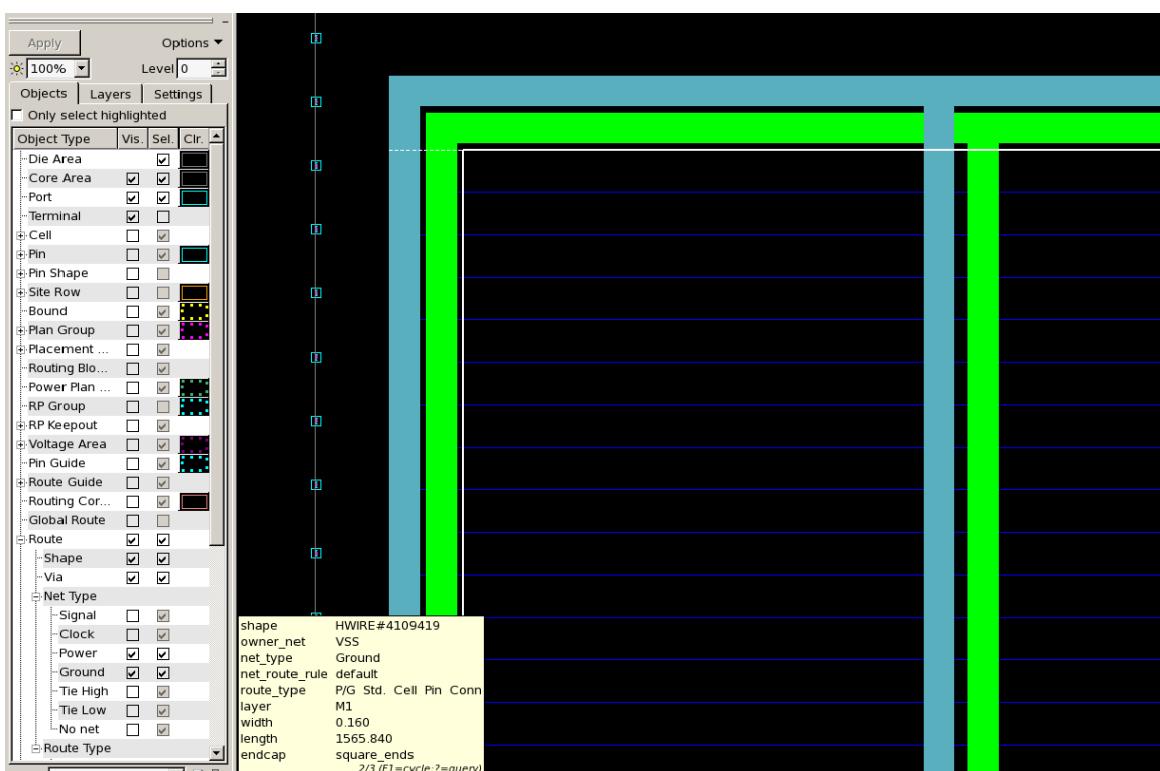
Hình 4.32 Kết quả đặt và đi dây VDD power ring



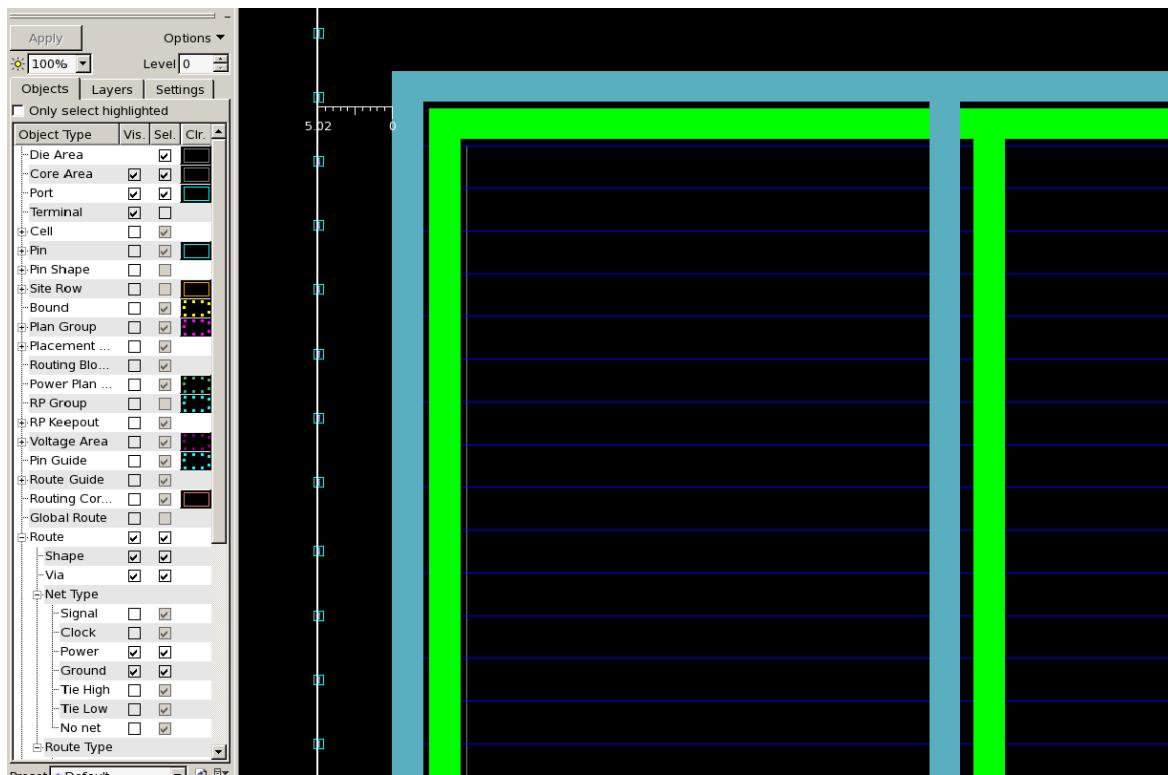
Hình 4.33 Kết quả đặt và đi dây VSS power ring



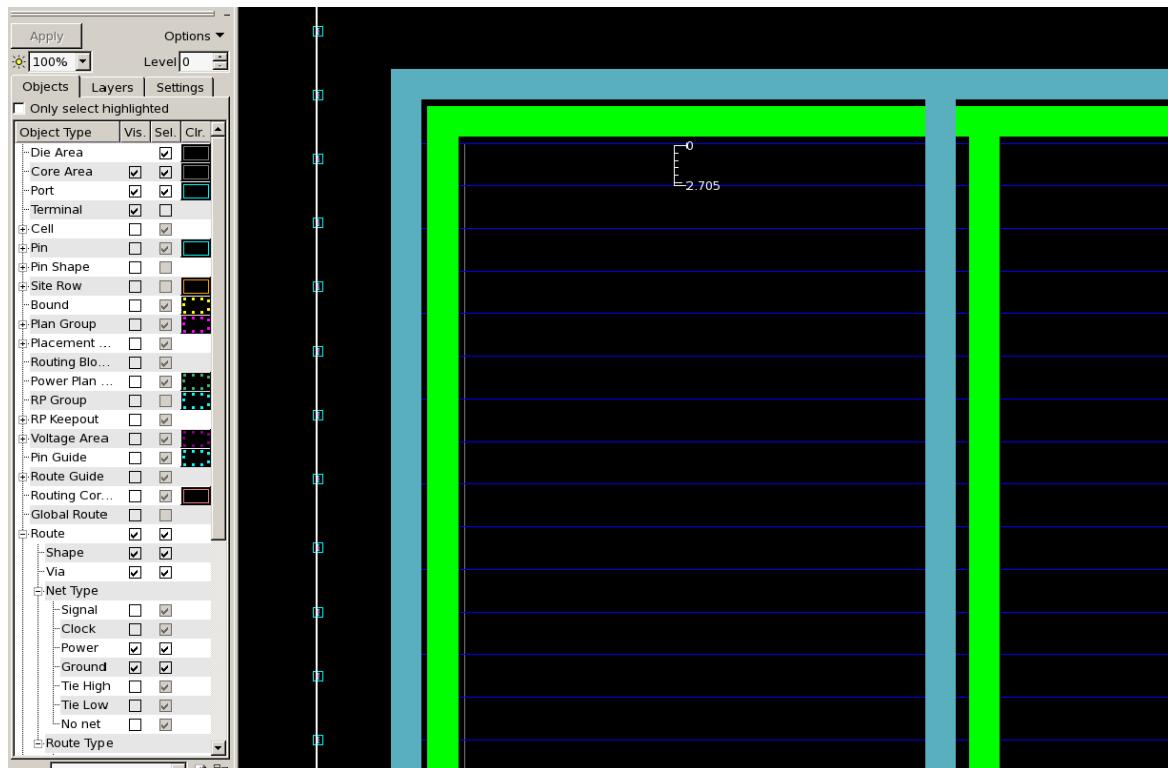
Hình 4.34 Kết quả đặt và di dây VDD rail



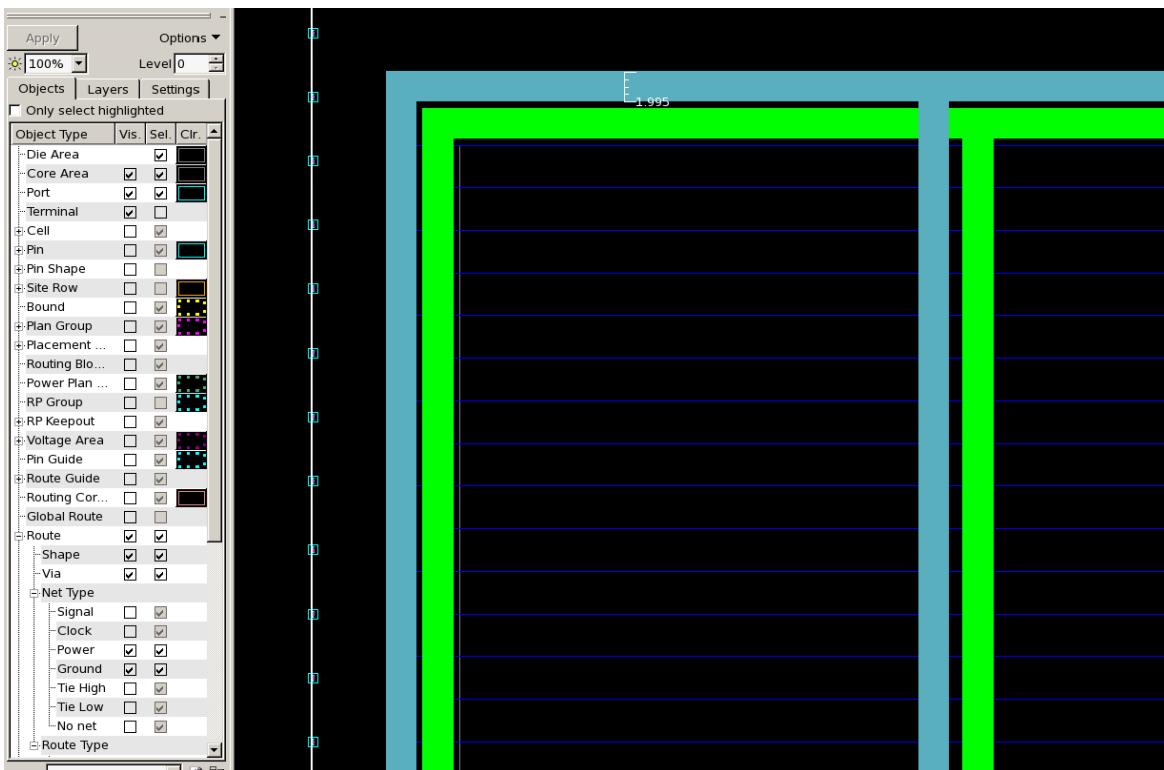
Hình 4.35 Kết quả đặt và di dây VSS rail



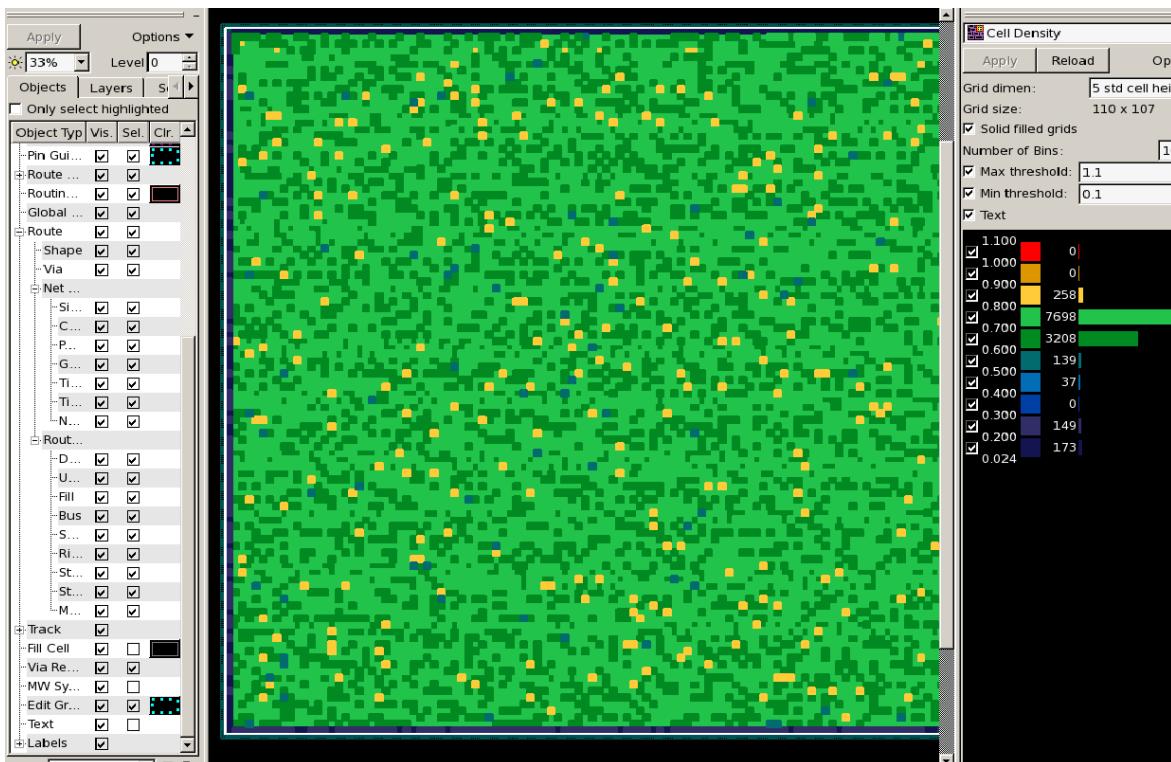
Hình 4.36 Kết quả đặt và đi dây khoảng cách giữa Die và Core



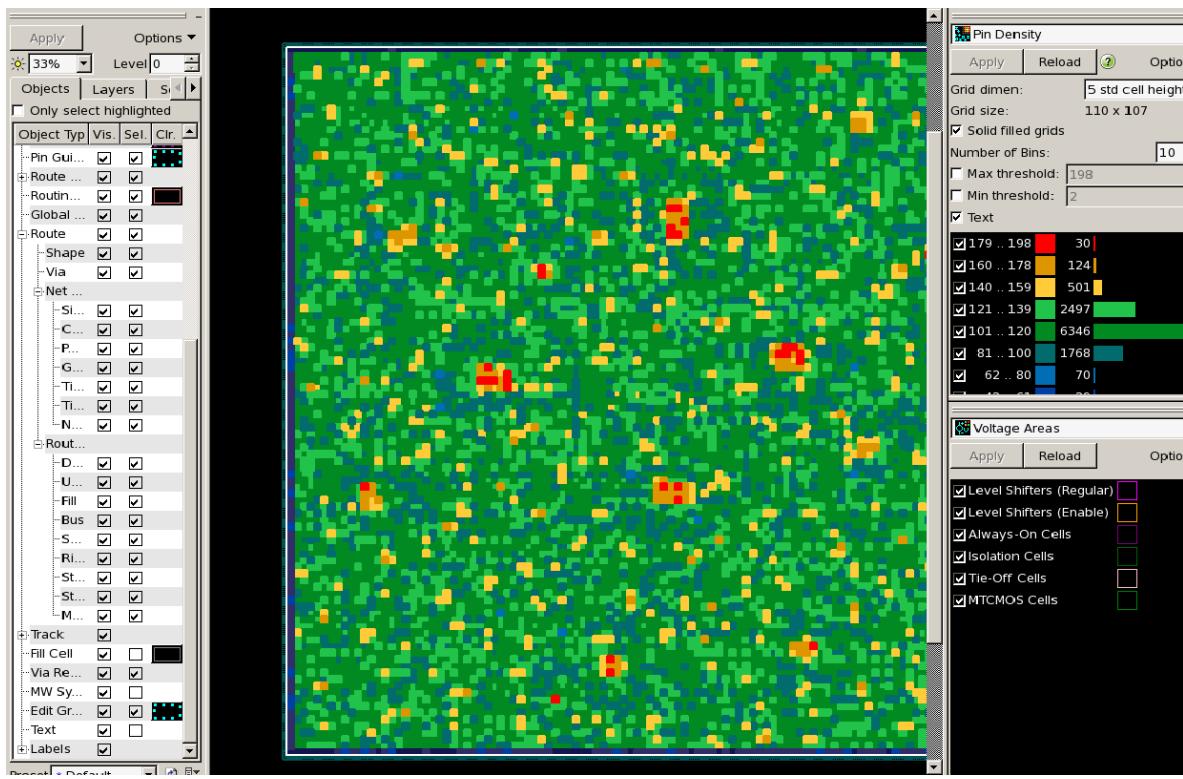
Hình 4.37 Kết quả đặt và đi dây chiều cao của Cell



Hình 4.38 Kết quả đặt và đi dây chiều rộng của Power



Hình 4.39 Kết quả đặt và đi dây Cell Density



Hình 4.40 Kết quả đặt và đi dây Pin Density

CHƯƠNG 5 TỔNG KẾT

5.1 Quá trình triển khai luận văn

Luận văn được triển khai và hoàn thành theo đúng tiến độ đề ra.

Quy trình làm việc được thực hiện và giám sát chặt chẽ nhằm đảm bảo thời gian thực thi luận văn.

5.2 Kết quả đạt được

Với những kết quả đạt được cho thấy những thành quả trong việc tiếp cận kiến trúc vi mạch mạng dữ liệu trên chip (NoC):

- Xây dựng thành công kiến trúc router cải thiện độ trì hoãn số chu kỳ xung thấp.
- Mô phỏng thành công hoạt động của router và mạng NoC 4x4 trong các trường hợp tiêu biểu bằng công cụ VCS, qua đó xác định được giá trị băng thông của bus cho phép trên mạng NoC 4x4.
- Dựa trên nền tảng kiến trúc mạng NoC cơ bản, tiến hành xây dựng một mạng dữ liệu hoàn chỉnh bao gồm cả khối giao tiếp NI (Network interface) dựa trên giao thức AHB được tích hợp để sử dụng với lõi ARM (ARM core) nhằm đạt được những đánh giá tổng quan nhất về hiệu năng của thiết kế hoàn chỉnh.
- Xây dựng mô hình giao diện phần mềm cho phép tạo mạng dữ liệu tự động (Soft IP) thích nghi với các ứng dụng khác nhau.
- Đánh giá và so sánh chi tiết các sản phẩm cuối cùng trên nền công nghệ 130nm.

5.3 So sánh các khái quát

5.3.1 Tính cáp thiết trong khả năng sửa lỗi của mạng NoC

Đối với đề xuất sửa một lỗi sau quá trình sản xuất của kiến trúc NoC, khả năng này cho phép tăng cường hiệu suất trong quá trình sản xuất. Tuy nhiên điều này chỉ phù hợp với một số công nghệ tiên tiến hiện thời (công nghệ 14nm hay 28nm) vì kỹ thuật chưa được hoàn thiện tốt nhất. Đối với các công nghệ 130nm trở lên, khi mà xác suất lỗi trong sản xuất gần như gần bằng không, khả năng sửa lỗi không mang lại những tác dụng thiết thực. Tuy nhiên, trong môi trường công nghiệp, các kỹ thuật sử dụng cho việc sửa lỗi cho các mạch tích hợp (LBIST cho công logic và MBIST cho bộ nhớ) là gần như bắt buộc trong các luồng thiết kế

cho dù thiết kế có sản xuất ở bất cứ công nghệ nào đi nữa. Do đó, việc tiếp cận khả năng sửa lỗi cho thiết kế là thật sự cần thiết và được tiêu chuẩn hóa trong khâu thiết kế kiến trúc vi mạch.

Việc tiếp cận khả năng sửa nhiều lỗi dựa trên định tuyến thích nghi đang được đề cập nhiều trong các báo cáo khoa học. Với hướng tiếp cận hiện tại của mạng NoC được đề cập trong báo cáo, việc tăng khả năng sửa lỗi ảnh hưởng không nhỏ đến cấu trúc nội bộ trong của router. Tuy nhiên, dựa trên tính kế thừa mô hình hiện có, việc mở rộng khả năng tránh nhiều lỗi trên mô hình NoC là khả thi và sẽ được tiếp tục thực hiện.

5.3.2 Mô hình mạng NoC

Việc chọn lựa mô hình mạng NoC như thế nào cho phù hợp là một trong những vấn đề luôn được đề cập đến trong nhiều báo cáo khoa học. Những báo cáo từ các tham khảo [23, 24] hay một so sánh cần thiết (<http://www.design-reuse.com/articles/10496/a-comparison-of-network-on-chip-and-busses.html>) tuy không đưa ra giá trị xác định cấu hình cụ thể của mạng NoC nhưng những so sánh cần thiết với các cấu hình khác nhau (từ 9 routers trở lên) đều cho thấy sự vượt trội so với các mô hình bus truyền thống được so sánh.

Ở mô hình NoC đề tài tiếp cận có cấu hình 4x4 tương đương với 16 router. Tuy nhiên việc mở rộng mô hình lớn hơn được thực hiện dễ dàng với công cụ tự động và tính khả thi trong việc tái sử dụng kiến trúc router đơn. Mặt khác, tần số tối đa có thể tổng hợp được cho thiết kế là 800 MHz cho thấy khả năng tích hợp vào các bus tốc độ cao của hệ thống hay các IP mang tính khả thi cao.

Tuy nhiên để có thể so sánh nhiều hơn về mặt công suất cũng như các giá trị khác một cách cụ thể, một hệ thống đầy đủ bao gồm các core IP kết nối trên NoC cũng như một hệ thống tương tự với mô hình bus truyền thống cùng với các thực nghiệm cần thiết trên phần mềm và trên phần cứng là cần thiết. Đây cũng là một trong những hạn chế của đề tài liên quan đến vấn đề tài nguyên và thời gian thực hiện.

5.4 Ý nghĩa khoa học đề tài

- Thành công trong việc tối ưu kiến trúc mô hình NoC cơ bản đặt nền tảng cho việc phát triển các mạng NoC đa dạng khác với việc kế thừa các đặc tính kỹ thuật của mạng NoC cơ bản.

- NoC là một trong những hướng giải quyết thiết yếu trong tương lai với mô hình đa nhân. Do đó, việc phát triển đào tạo và đưa vào giảng dạy kiến trúc NoC trở nên thiết thực và có tính ứng dụng cao.
- Kết quả phát triển kiến trúc NoC cơ bản thành công cho phép phát triển các ứng dụng đa nhân trong thực tế cũng như nghiên cứu.

5.5 Đề nghị hướng phát triển đề tài

- Dựa trên các kết quả đạt được, mô hình NoC dự định được phát triển với các kiến trúc khác nhau nhưng vẫn giữ các đặc tính kỹ thuật bên trong nhằm tăng cường hiệu năng và có những đánh giá và so sánh chính xác so với kiến trúc cơ bản đã tiếp cận.
- Tiếp cận và tích hợp các giải thuật thích nghi nhằm tăng cường tỉ lệ thành công trong quá trình sản xuất IC.

TÀI LIỆU THAM KHẢO

- [1] Ankur Agarwa, Boca Raton, Cyril Iskander, Ravi Shankar, “**Survey of Network on Chip (NoC) Architecture & Contributions**”, Engineering, Computing and Architecture ISSN 1934-7197 Volume 3, Issue 1, 2009, Boca Raton Univ, USA.
- [2] ARUN JANARTHANAN, “**Networks-On-Chip Based High Performance, Communication Architectures For FPGAs**”, Doctor of Philosophy Thesis.
- [3] Boris Grot, “**Express Cube Topologies for On-chip Interconnects**”, High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium, Dept. of Comput. Sci., Univ. of Texas at Austin, Austin, TX.
- [4] F. Moraes and N. Calazan, “**An infrastructure for low area overhead packet-switching network on chip**”, Integration - The VLSI Journal, vol. 38, Issue 1, pp. 69-93, October 2004, Pontifícia Universidade Católica do Rio Grande do Sul (FACIN-PUCRS), Av. Ipiranga, 6681, Prédio 30/BLOCO 4, 90619-900 Porto Alegre, RS, Brazil.
- [5] Mohammad Arjomand, Hamid Sarbazi-Azad, “**Performance Evaluation of Butterfly on-Chip Network for MPSoCs**”, SoC Design Conference, 2008. ISOCC '08. International, Comput. Eng. Dept., Sharif Univ. of Technol., Tehran.
- [6] Chia-Hsin Owen Chen¹, Niket Agarwal, Tushar Krishna¹, Kyung-Hoae Koo, Li-Shiuan Peh¹, and Krishna C. Saraswat, “**Physical vs. Virtual Express Topologies with Low-Swing Links for Future Many-core NoCs**”, Networks-on-Chip (NOCS), 2010 Fourth ACM/IEEE International Symposium, Dept. of Electr. Eng. & Comput. Sci., Massachusetts Inst. of Technol., Cambridge, MA, USA.
- [7] Amit Kumar Li-Shiuan Peh, Parth Kundu, Niraj K.Jha, “**Toward Ideal On-Chip Communication Using Express Virtual Channel**”, micr-28-01-kuma.3d.IEEE 2008, Princeton University.
- [8] E. Bolotin, I. Cidon, R. Ginosar and A. Kolodny, “**Cost considerations in Network on Chip**”, Integration: The VLSI Journal, no. 38, 2004, pp. 19-42.
- [9] C. A. Zeferino and A. A. Susin, “**SoCIN: A parametric and scalable network-on-chip**”, Integrated Circuits and Systems Design, 2003. SBCCI 2003. Proceedings. 16th Symposium on, Instituto de Informatica, Univ. Fed. do Rio Grande do Sul, Porto Alegre, Brazil.

- [10] D. Bertozzi, A. Jalabert, S. Murali, R. Tamhankar, S. Stergiou, L. Benini, and G. De Micheli, “**NoC synthesis flow for customized domain specific multiprocessor systems-on-chip**”, IEEE Transactions on Parallel and Distributed Systems, vol. 16, no. 2, pp. 113-129, 2005, Bologna Univ., Italy.
- [11] M. Dall’Ossa, G. Biccari, L. Giovannini, L. D. Bertozzi, and L. Benini, “**XPIPES: A latency insensitive parameterized network-on-chip architecture for multiprocessor SoCs**”, Computer Design, 2003. Proceedings. 21st International Conference, DEIS, Bologna Univ., Italy.
- [12] Avinash Kodi, Ahmed Louri, Janet Wang2, “**Design of Energy-Efficient Channel Buffers with Router Bypassing for Network-on-Chips (NoC)**”, Department of Electrical and Computer Engineering, University of Arizona, Tucson, AZ 85721, ch. of Electr. Eng. & Comput. Sci., Ohio Univ., Athens, OH.
- [13] Tobias Bjerregaard and Jens Spars, “**A Router Architecture for Connection-Oriented Service Guarantees in the MANGO Clockless Network-on-Chip**”, Design, Automation and Test in Europe, 2005. Proceedings, Dept of Informatics & Math. Modelling, Tech. Univ. Denmark, Lyngby, Denmark.
- [14] James Balfour, William J. Dally, “**Design Tradeoffs for Tiled CMP On-Chip Networks**”, ICS06, June 28-30, Cairns, Queensland, Australia.
- [15] P. T. Wolkotte, G. J. M. Smit, G. K. Rauwerda, and L. T. Smit, “**An energy-efficient reconfigurable circuit-switched network-on-chip**”, Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International, University of Twente, The Netherlands.
- [16] J. W. Dally and B. Towles, “**Route packets, not wires: On-Chip interconnection networks**”, Proc. IEEE International Conference on Design and Automation, pp. 684-689, June 2001, Comput. Syst. Lab., Stanford Univ., CA, USA.
- [17] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, “**QNoC: QoS architecture and design process for network on chip**”, Journal of Systems Architecture, Volume 50, Issue 2-3 (Special Issue on Network on Chip), pp. 105-128, February 2004, Electrical Engineering Department, Technion, Israel Institute of Technology, Haifa 32000, Israel.
- [18] Kwanho Kim, Se-Joong Lee, Kangmin Lee, and Hoi-Jun Yoo, “**An Arbitration Look-Ahead Scheme for Reducing End-to-End Latency in Networks on chip**”, Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium, Dept. of Electr. Eng., KAIST, Daejeon, South Korea.

- [19] Jose DUATO, Sudhakar YALAMANCHILI, Lionel NI “**Interconnection Networks An Engineering Approach**” Ebook.
- [20] Ran Manevich, Israel Cidon, Avinoam Kolodny, Isask'har Walter, “**Centralized Adaptive Routing for NoCs**”, Computer Architecture Letter, Volume 9, Issue 2, 2010.
- [21] Sheng Ma, Natalie Enright Jerger, Zhiying Wang, “**Whole Packet Forwarding: Efficient Design of Fully Adaptive Routing Algorithms for Networks-on-Chip**”, High-Performance Computer Architecture, Pages 1-12, 2012.
- [22] Haibo Zhu, Partha Pratim Pande, Cristian Grecu, “**Performance Evaluation of Adaptive Routing Algorithms for achieving Fault Tolerance in NoC Fabrics**”, Application -specific Systems, Architectures and Processors, 2007.
- [23] Altera Corporation, “**Applying the Benefits of Network on a Chip Architecture to FPGA System Design**”, April, 2011.
- [24] Daniel Sanchez, George Michelogiannakis, and Christos Kozyrakis, “**An Analysis of On-Chip Interconnection Networks for Large-Scale Chip Multiprocessors**”, ACM Transactions on Architecture and Code Optimization, Vol. 7, No. 1, Article 4, April 2010.
- [25] ARM Corp., ”AMBA AHB-Lite Protocol Specification”, www.arm.com
- [26] ARM Corp., ”AMBA APB Protocol Specification”, www.arm.com