ĐẠI HỌC QUỐC GIA TP. HCM
TRƯỜNG ĐẠI HỌC BÁCH KHOA

BÙI QUANG TÍN

THIẾT KẾ KIẾN TRÚC VI MẠCH MBIST CÓ TÍNH LINH HOẠT CAO CHO SRAM

Chuyên ngành: Kỹ thuật điện tử
Mã số: 60520203

LUẬN VĂN THẠC SĨ

TP. HỒ CHÍ MINH, tháng 12 năm 2016

Công trình được hoàn thành tại: **Trường Đại học Bách Khoa – ĐHQG-HCM**

Cán bộ hướng dẫn khoa học: .....

(Ghi rõ họ, tên, học hàm, học vị và chữ ký)

Cán bộ chấm nhận xét 1: .....
(Ghi rõ họ, tên, học hàm, học vị và chữ ký)

Cán bộ chấm nhận xét 2: .....
(Ghi rõ họ, tên, học hàm, học vị và chữ ký)

Luận văn thạc sĩ được bảo vệ tại Trường Đại học Bách Khoa, ĐHQG Tp. HCM ngày . . . . . tháng . . . . năm . . . . .

Thành phần Hội đồng đánh giá luận văn thạc sĩ gồm:

(Ghi rõ họ, tên, học hàm, học vị của Hội đồng chấm bảo vệ luận văn thạc sĩ)

1.....

2.....

3.....

4.....

5.....

Xác nhận của Chủ tịch hội đồng đánh giá LV và Trưởng khoa quản lý chuyên ngành sau khi luận văn đã được sửa chữa (nếu có).

CHỦ TỊCH HỘI ĐỒNG **TRƯỞNG KHOA**

LỜI CÁM ƠN

Luận văn được hoàn thành dưới sự hướng dẫn, giúp đỡ tận tình của PGS.TS Hoàng Trang, xin gửi tới thầy lòng biết ơn sâu sắc, những kiến thức, kinh nghiệm trong học tập, nghiên cứu khoa học và các bài học trong cuộc sống.

Tác giả cũng xin gửi lời cảm ơn chân thành tới Ban chủ nhiệm khoa Điện – Điện tử, Phòng Đào tạo sau Đại học, trường Đại học Bách Khoa- Đại học Quốc Gia Tp Hồ Chí Minh. Tôi cũng xin gửi lời cảm ơn chân thành tới các Thầy, Cô giáo trong Bộ môn Điện tử đã nhiệt tình giảng dạy trong suốt quá trình học tập tại Khoa.

Xin chân thành cảm ơn sự giúp đỡ, góp ý và động viên của bạn bè, người thân trong suốt quá trình học tập và thực hiện luận văn.

Do khả năng nhận thức của bản thân tác giả, luận văn còn nhiều hạn chế, thiếu sót. Kính mong nhận được các ý kiến đóng góp của thầy cô cùng các độc giả quan tâm tới luận văn này.

Xin chân thành cảm ơn!

TÓM TẮT LUẬN VĂN

Với xu hướng công nghệ CMOS sẽ ngày càng giảm kích thước, mật độ tích hợp lên trên các hệ thống vi mạch ngày càng nhiều để đáp ứng các yêu cầu của người sử dụng và nhà sản xuất, cho nên các bộ nhớ trên các hệ thống vi mạch sau khi sản xuất bị lỗi là điều không thể tránh khỏi. Do đó, kiến trúc tự động kiển tra lỗi trên bộ nhớ (Memory Build in self test) MBIST đã được sử dụng phát hiện ra những lỗi sai xuất hiện trên SRAM và được ứng dụng trong khối tự động sửa sai cho bộ nhớ.

Luận văn sẽ tập trung theo hướng ứng dụng để nâng cao tính hiệu quả của kiến trúc MBIST bằng việc tích hợp 2 giải thuât MarchC- và TLAPNPSF lên trên kiến trúc nhưng cũng đồng thời đảm bảo các điều kiện về mặt thiết kế vật lý, đồng thời xây dựng một môi trường kiểm tra thiết kế RTL để kiểm chứng tính hiệu quả của kiến trúc sau khi thiết kết ở mức độ thanh ghi. Với mục tiêu hướng tới tính linh hoạt của MBIST, thì môi trường tự động tạo ra kiến trúc MBIST với các giải thuật được xây dựng sẵn, cho phép người dùng chọn lựa những giải thuật được tích hợp (MarchA, MarchC-, TLAPNPSF) vào trong kiến trúc MBIST của bộ nhớ SRAM, đồng thời tính linh hoạt được thể quan việc mở rộng được dung lượng bộ nhớ kiểm tra thay đổi từ 1Kx8 tới 64Kx64 cũng như tùy chọn được số lượng bộ nhớ kết nối.

Với kiến trúc đề ra, phát huy được tính linh hoạt khi có phạm vi kiểm tra rộng, nhiều giải thuật được tích hợp lên kiến trúc, tạo ra sự hiệu quả của kiến trúc MBIST của đề tài so với các nghiên cứu khác trên thế giới.

SUMMARY

CMOS size have trend to decrease more and more, and the intergrated density on circuit and IC system is bigger to adapt with many required applications from user and manufacture as well. That's why, the failure mode on memory have frequently happened since some production or environment condition impact. Memory Built in Self Test – MBIST is a solution that had been applied on memory – SRAM to dectect the failure on SRAM and it will be came a part in the self repair system.

In this work, an effective architecture of MBIST for different SRAM memories is proposed for ensuring high ability of detecting memory faults supported by the most popular algorithms namely MARCH C- and TLAPNPSF but also satisfy strict silicon criteria, the tesing environment will be built to confirm the effeiveness of MBIST structure afte designing in RTL. To achive the dynamic of MBIST architecture, a environment generates the MBIST structure based on the intergrated algorithms (MarchC-, MarchA, TLAPNPSF). By the way, MBIST architecture can change the testing range of memory size from 1Kx8 to 64Kx64 and option to choose how many memories can be connected to MBIST.

The proposed MBIST can check with large range on SRAM with the intergrated algorithms on MBIST structure and doubly confirm the effectiveness with the other research on the world.

LỜI CAM ĐOAN

Tôi xin cam đoan đây là công trình nghiên cứu của riêng tôi. Các số liệu, kết quả nêu trong Luận văn là trung thực và chưa từng được ai công bố trong bất kỳ công trình nào khác.

Tôi xin cam đoan rằng mọi sự giúp đỡ cho việc thực hiện Luận văn này đã được cảm ơn và các thông tin trích dẫn trong Luận văn đã được chỉ rõ nguồn gốc.

MỤC LỤC

LỜI CÁM ƠN i
TÓM TẮT LUẬN VĂN ii
SUMMARY iii
LỜI CAM ĐOAN iv
MỤC LỤC v
LIỆT KÊ BẢNG vii
LIỆT KÊ HÌNH ix
LỜI MỞ ĐẦU xi
Chương 1: TỔNG QUAN 1
1.1 Kỹ thuật BIST 1
1.1.1 Tầm quan trọng của việc kiểm tra bộ nhớ 1
1.1.2 Các mô hình kiểm tra bộ nhớ hiện tại 2
1.2 Tình hình nghiên cứu trong và ngoài nước 4
Chương 2: CƠ SỞ LÝ THUYẾT VÀ ĐỀ XUẤT KIẾN TRÚC MBIST 9
2.1 Cấu trúc bộ nhớ bán dẫn 9
2.2 Các lỗi thường xảy ra trên bộ nhớ SRAM 11
2.2.1 Cấu trúc bộ nhớ SRAM và các nguyên nhân gây ra lỗi 11
2.2.2 Các mô hình lỗi trên bộ nhớ SRAM 13
2.2.3 Thuật toán được sử dụng để đi phát hiện lỗi của khối MBIST 20
Chương 3: KIẾN TRÚC MBIST CÓ TÍNH LINH HOẠT CAO 28
3.1 Kiến trúc cơ bản của MBIST 28
3.2 Kiến trúc đề xuất MBIST 30
3.2.1 Khối controller 31
3.2.2 Khối mbist_demux 34
3.2.3 Khối mbist_decoder 34
3.3 Môi trường kiểm tra độ chính xác của MBIST 53
3.4 Môi trường tạo tính linh hoạt cho MBIST 55
Chương 4 : KẾT QUẢ THỰC HIỆN 57
4.1 Kết quả verification 57

4.2 Kết quả tổng hợp từ cấp độ RTL xuống lớp cổng 62
4.3 Kết quả Place and Route.....69
Chương 5: Ý NGHĨA KHOA HỌC VÀ HƯỚNG PHÁT TRIỂN.....77
5.1 Ý nghĩa khoa học 77
5.2 Hướng phát triển của đề tài.....77
CÔNG TRÌNH CÔNG BỐ.....79
TÀI LIỆU THAM KHẢO 80

LIỆT KÊ BẢNG

Bảng 1.1 - Thống kê về các nghiên cứu MBIST từ 2005-2013 5
Bảng 1.2 - Thống kê các dạng lỗi có thể phát hiện được bằng giải thuật March 6
Bảng 1.3 - Khảo sát độ bao phủ của các thuật toán [1] [4] 7
Bảng 2.1 - Các mẫu dữ liệu của Phương pháp Hamiltonian 23
Bảng 2.2 - Các lỗi có thể xảy ra khi các ô nhớ kế cận thay đổi giá trị 27
Bảng 3.1 - Định nghĩa các tín hiệu của khối controller với 3 giải thuật 33
Bảng 3.2 - Các trạng thái MarchC- 35
Bảng 3.3 - Sự chuyển trạng thái của MarchC- 36
Bảng 3.4 - Định nghĩa tín hiệu trên khối marchc_decoder 36
Bảng 3.5 - Định nghĩa tín hiệu của khối marchc_controller 37
Bảng 3.6 - Định nghĩa các tín hiệu của khối marchc_datapath 39
Bảng 3.7 - Các trạng thái của TLAPNPSF 41
Bảng 3.8 - Sự chuyển trạng thái của TLAPNPSF 42
Bảng 3.9 - Định nghĩa các tín hiệu của apnpsf_decoder 43
Bảng 3.10 - Định nghĩa tín hiệu sơ đồ khối apnpsf_controller 44
Bảng 3.11 - Định nghĩa các tín hiệu của sơ đồ khối apnpsf_datapath 46
Bảng 3.12 - Các trạng thái của MarchA 49
Bảng 3.13 - Sự chuyển trạng thái của MARCHA 49
Bảng 3.14 - Sơ đồ tín hiệu khối marcha_decoder 50
Bảng 3.15 - Định nghĩa tín hiệu của khối marcha_controller 51
Bảng 3.16 - Định nghĩa các tín hiệu của khối marcha_datapath 52
Bảng 4.1- Sơ đồ cây thư mục của môi trường kiểm chứng giải thuật 57
Bảng 4.2 - Chức năng các thư mục trong cấu trúc cây thư mục 58
Bảng 4.3 - Kết quả chạy phân tích trên giải thuật MarchC- 58
Bảng 4.4 - Kết quả chạy phân tích trên giải thuật MarchA 59
Bảng 4.5 - Kết quả chạy phân tích trên giải thuật TLAPNPSF 60
Bảng 4.6 - Sơ đồ cây thư mục của môi trường tự tạo ra kiến trúc MBIST 61
Bảng 4.7 - Chức năng của các thư mục bên trong môi trường tự tạo kiến trúc MBIST 62

Bảng 4.8 - Các thông số bên trong môi trường thiết kế 63
Bảng 4.9 - Sơ đồ cây thư mục môi trường tổng hợp kiến trúc vật lý 63
Bảng 4.10 - Đặc tả cây thư mục của môi trường tổng hợp 65
Bảng 4.11 - Các ràng buộc trong quá trình tổng hợp kiến trúc MBIST 66
Bảng 4.12 - Phân tích kết quả diện tích của kiến trúc ở lớp vật lý 66
Bảng 4.13 - Phân tích công suất của kiến trúc MBIST ở lớp vật lý 67
Bảng 4.14 - Mối quan hệ giữa tần số và tổng diện tích các cell nhớ của thiết kế 68
Bảng 4.15 - So sánh kết quả tổng hợp lớp cổng của thiết kế với các nghiên cứu khác 69

LIỆT KÊ HÌNH

Hình 1.1- Kiến trúc kiểm tra MBOST....2
Hình 2.1 - Tốc độ tương ứng các bộ nhớ nội....9
Hình 2.2 - Kiến trúc tổng quát bên trong bộ nhớ....11
Hình 2.3 - Cấu trúc bên trong của bộ nhớ SRAM....12
Hình 3.4 - Các khối thiết kế bên trong kiến trúc MBIST....31
Hình 3.5 – Mô hình trạng thái máy cho khối controller....32
Hình 3.6 - Sơ đồ in-out của khối controller với 3 giải thuật....33
Hình 3.7 - Giao diện khối mbist_demux trong MBIST kiểm tra 8 bộ nhớ....34
Hình 3.8 - Mô hình trạng thái máy của thuật toán MarchC-....35
Hình 3.9 - Sơ đồ tín hiệu khối marchc_decoder....36
Hình 3.10 - Sơ đồ tín hiệu của khối controller....37
Hình 3.11 - Sơ đồ tín hiệu khối marchc_datapath....39
Hình 3.12 - Sơ đồ trạng thái máy của giải thuật TLAPNPSF....41
Hình 3.13 - Sơ đồ tín hiệu của khối apnpsf_decoder....43
Hình 3.14 - Sơ đồ khối apnpsf_controller....44
Hình 3.15 - Sơ đồ tín hiệu khối apnpsf_datapath....46
Hình 3.16 - Mô hình trạng thái máy của thuật toán MarchA....49
Hình 3.17- Sơ đồ tín hiệu khối marcha_decoder....50
Hình 3.18 - Sơ đồ tín hiệu khối marcha_controller....51
Hình 3.19 - Sơ đồ khối của các chân tín hiệu của marcha_datapath....52
Hình 3.20 - Môi trường test bench kiểm tra sự hiệu quả của kiến trúc MBIST....55
Hình 3.21 - Môi trường tạo ra kiến trúc MBIST....56
Hình 4.1 - Dạng sóng tín hiệu tại vị trí lỗi C350H (5000D)....59
Hình 4.2 - Dạng sóng tín hiệu tại vị trí lỗi 64H (100D)....60
Hình 4.3 - Dạng sóng tín hiệu tại vị trí lỗi 7E0H (2016D)....61
Hình 4.4 - Mối quan hệ giữa tần số và diện tích cell nhớ....68
Hình 4.5 Tạo core cho chip....70
Hình 4.6 – Tạo các chân kết nối (port)....70
Hình 4.7 – Các cổng trước khi đặt trên core và kết nối với port....70

Hình 4.9 - Chi tiết sau khi đặt vào bên trong các terminal 71
Hình 4.10 - Chi tiết của thiết kế sau khi đi dây ground 72
Hình 4.11 - Chi tiết sau khi đặt thêm dây power 72
Hình 4.12 - Chi tiết kết quả sau khi đi thêm xung clock 73
Hình 4.13 - Kết quả sau khi đi thêm các dây tín hiệu 73
Hình 4.14 - Kết quả sau khi tiến hành chỉ đi dây clock 74
Hình 4.15 - Kết quả sau khi đi dây clock 74
Hình 4.16 - Kết qua sau khi đi dây ground 75
Hình 4.17- Kết quả sau khi đi dây power 75
Hình 4.18 - Kết quả đặt và đi dây toàn bộ chip (chi tiết) 76

LỜI MỞ ĐẦU

Tính cấp thiết của đề tài

Với tốc độ phát triển nhanh của dung lượng bộ nhớ trong để đáp ứng với nhiều tác vụ được người dùng yêu cầu, thì quá trình sản xuất các bộ nhớ phải đòi hỏi phải đảm bảo một cách chính xác và đạt được tối ưu về mặt năng suất nhất. Thông thường, để kiểm tra một bộ nhớ có lỗi hay không thì một mạch kiểm tra mức logic bên ngoài theo từng loại bộ nhớ, theo dung lượng khác nhau được sử dụng để đảm bảo bộ nhớ không bị lỗi khi đến tay người dùng. Với mục đích nhằm làm tăng sự linh hoạt và hiệu quả của quy trình kiểm tra cũng như tiết kiệm chi phí, kiến trúc tự kiểm tra bộ nhớ - Memory build in self test - (MBIST) đã được xây dựng và tích hợp sẵn bên trong mỗi bộ nhớ để đảm bảo đầu ra luôn chính xác và đồng bộ quá trình kiểm tra. Tuy nhiên các đề tài nghiên cứu hiện nay liên quan nhiều tới các giải thuật, hay tối ưu về mặt công suất, diện tích chiếm dụng, nhưng bị giới hạn về dung lượng kiểm tra cũng như khả năng bao phủ hết tất cả các mô hình lỗi trên SRAM.

Mục đích của đề tài

Tìm hiểu được các mô hình lỗi thường hay xảy ra trên SRAM và các giải thuật có khả năng phát hiện ra các mô hình lỗi tương ứng. Kiến trúc MBIST mới sẽ dựa trên các giải thuật đó với khả năng bao phủ hết tất cả các mô hình lỗi thường hay xảy ra trên SRAM và cho phép người dùng, nhà sản xuất lựa chọn các giải thuật sẽ thực hiện việc kiểm tra, ngoài ra kiến trúc MBIST còn cho phép thay đổi dung lượng bộ nhớ và số lượng bộ nhớ được kết nối tới bộ nhớ trong quá trình kiểm tra. Đồng thời xây dựng được môi trường có khả năng tự tao ra kiến trúc MBIST ở cấp độ RTL với các giải thuật được tích hợp sẵn. Môi trường kiểm tra các mô hình lỗi được xây dựng để chứng minh tính hiệu quả của các giải thuật được tích hợp lên kiến trúc MBIST. Kiến trúc MBIST cũng sẽ được tổng hợp ở mức cổng vật lý bằng thư viện TSMC 90nm để thấy được khả năng ứng dụng của đề tài khi chạy thực tế.

Đối tượng và phạm vi nghiên cứu

Đối tượng nghiên cứu của đề tài là kiến trúc tự kiểm tra lỗi (MBIST) xuất hiện trên SRAM.

Phạm vi nghiên cứu của đề tài là tập trung nghiên cứu các mô hình lỗi thường hay xuất hiện trên SRAM với dung lượng bộ nhớ từ 1Kx8 tới 64Kx64. Thư viện TSMC 90nm được sử dụng để phân tích sự hiệu quả của kiến trúc ở lớp vật lý với những nghiên cứu khác.

Ý nghĩa khoa học và thực tiên của đề tài

Tính linh hoạt cao của kiến trúc mở ra hướng đi mới trong việc kiểm tra các mô hình lỗi thường hay xuất hiện trên SRAM, số lượng các thuật toán được tích hợp trên kiến trúc. Mở ra hướng đi mới nhằm tạo ra môi trường kiển tra tự động tạo ra kiến trúc hiệu quả nhất cho kiến trúc MBIST.

Kiến trúc MBIST với sự hiệu quả cao sẽ làm tiền đề cho sự phát triển của kiến trức tự động sửa lỗi đạt được hiệu quả tối ưu nhất. Đặc biệt trong định hướng phát triển của ngành vi mạch non trẻ tại Việt Nam thì kiến trúc mới MBIST là giải pháp để tiết kiệm chi phí và nâng cao chất lượng bộ nhớ sau khi thiết kế và sản xuất trong nước.

Kết cấu của luận văn

Luận văn gồm 5 chương:

Chương 1: Tổng quan kiến trúc MBIST

Chương 2: Cơ sở lý thuyết và đề xuất kiến trúc MBIST

Chương 3: Kiến trúc MBIST có tính linh hoạt cao

Chương 4: Kết quả thực hiện

Chương 5: Kết luận và kiến nghị

Chương 1: TỔNG QUAN

Có rất nhiều kỹ thuật được sử dụng để phát hiện lỗi trên các loại bộ nhớ, tuy nhiên kỹ thuật MBIST mang những đặc điểm ưu việt hơn nên đã được sử dụng nhiều hơn trong quy trình phát hiện lỗi trên bộ nhớ. Do đó, hướng nghiên cứu MBIST sẽ là một lĩnh vực mà được nhiều nhà khoa học tìm hiểu và nghiên cứu rộng rãi.

1.1 Kỹ thuật BIST

1.1.1 Tầm quan trọng của việc kiểm tra bộ nhớ

Theo đinh luật Moore's, kích thước CMOS trong nghành công nghệ vi mạch đã ngày càng nhỏ dần đi: từ 130nm năm 2002,…, 65nm năm 2006, 45nm năm 2008,… 14nm năm 2014 và theo xu hướng chung thì có thể được phát triển để giảm xuống nữa trên nền công nghệ 10nm năm 2017. Tuy nhiên trái ngược với sự phát triển của kích thước CMOS, thì ngành công nghiệp vi mạch sẽ đối diện với 3 vấn đề lớn:

+ Trong cùng quy trình sản xuất, năng suất sản xuất ngày càng thấp đi

+ Các mô hình lỗi cũng như khả năng xảy ra lỗi xuất hiện ngày càng nhiều.

+ Ảnh hưởng nhiều tới sự thay đổi của quá trình hoạt động, điện áp và nhiệt độ của mỗi chip sau khi sản xuất.

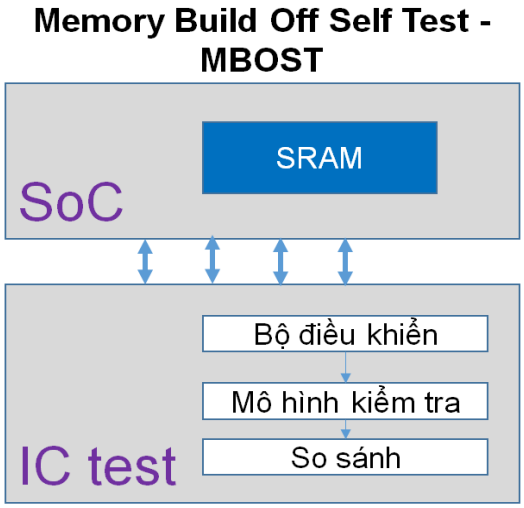
Đáp ứng với nhu cầu hiện nay của người sử dụng và nhà sản xuất, các hệ thống điện tử sẽ thực thi nhiều ứng dụng sẽ đòi hỏi ngày càng cao khối lượng tính toán, các vi mạch bán dẫn sẽ có khối lượng xử lý ngày càng nhiều, do đó hệ thống sẽ yêu cầu dung lượng bộ nhớ lớn để đáp ứng kịp thời các yêu cầu về khối lượng tính toán. Theo xu hướng phát triển của ngành công nghệ vi mạch thì trong những năm 4-5 gần đây hầu hết diện tích trên vi mạch bị chiếm bởi 94% bộ nhớ. Trong lĩnh vực điện tử thì chất lượng vi mạch phụ thuộc vào diện tích phần bán dẫn sử dụng trên chip. Do đó theo định hướng phát triển của ngành vi mạch thì chất lượng của chip sau khi sản xuất sẽ phụ thuộc rất nhiều chất lượng của bộ nhớ, nghĩa là phần diện tích chiếm dụng nhiều trên chip. Cho nên kích thước CMOS ngày càng giảm xuống thì số lỗi xuất hiện ngày càng nhiều, và đồng thời sẽ mở rộng thêm nhiều dạng mã lỗi mới xuất hiện khi các nhà thiết kế vi mạch có xu hướng cho ra đời nhiều kiến

trúc bộ nhớ mới. Cho nên việc kiểm tra chất lượng bộ nhớ là một phần quan trọng trong việc đảm bảo sự hoạt động tốt sau khi sản xuất.

1.1.2 Các mô hình kiểm tra bộ nhớ hiện tại

a. Kỹ thuật MBOST

Việc kiểm tra bộ nhớ từ bên ngoài MBOST (Memory Built Off Self test) là một phương pháp phổ biến trên thế giới để kiểm tra các thiết bị sau khi sản xuất, tuy nhiên đối với phương pháp này lại sẽ gặp phải một số nhược điểm khi kích thước CMOS ngày càng giảm đi: chi phí cao, chất lượng kiểm tra giảm dần, hạn chế việc truy cập tới các đường địa chỉ trực tiếp đến bộ nhớ.



Hình 1.1- Kiến trúc kiểm tra MBOST

Các thiết bị test MBOST sẽ được thiết kế một cách riêng biệt để kiểm tra thiết bị theo: cấu hình và dung lượng bộ nhớ, số cổng vào ra, mô hình tín hiệu hoạt động. Điều đó dẫn tới chi phí cho việc kiểm tra sẽ ngày càng nhiều nếu như có một sự thay đổi mặc dù chỉ là nhỏ trong quá trình sản xuất vì sẽ đòi hỏi các thiết bị phải được thiết kế và kiểm chứng lại lần nữa trước khi áp dụng vào thực tiễn. Ngoài ra do xu hướng thay đổi kích thước CMOS cho nên tại mỗi thời điểm khác nhau của một công nghệ CMOS thì các thiết bị MBOST phải yêu cầu được thiết kế lại hoàn toàn cho phù hợp với các mô hình lỗi mới phát sinh tương ứng, đồng thời việc bảo trì và nâng cấp sẽ tốn khá nhiều chi phí phát sinh.

Ngày nay các hệ thống đa phần đã được mở rộng lên tới hàng trăm bộ nhớ bên trong, cho nên với việc kiểm tra từ các thiết bị truy cập các bộ nhớ từ bên ngoài vào là điều khó thực hiện và tính kinh tế không cao. Ngoài ra các bộ MBOST muốn truy cập với bộ nhớ bên trong hệ thống cần phải kết nối với thông qua đường truyền khá dài và phải yêu cầu hoạt động với đúng tốc độ truyền dẫn, tuy nhiều điều này là không dễ thực hiện khi các cổng vào ra của bộ nhớ phải chia sẻ cho nhiều chức năng khác nhau dẫn đến kết quả sau khi test sẽ không ổn định, và làm giảm chất lượng kiểm tra.

b. Kỹ thuật MBIST

Việc kiểm tra chất lượng bộ nhớ bằng phương pháp MBOST có hiệu suất thấp do tốn chi phí cao, yêu cầu thiết kế riêng cho từng chip. Do đó, theo xu hướng phát triển chung của các phương phát test (DFT) thì sẽ chuyển từ MBOST thành tự xây dựng mô hình test trên chính thiết bị nhớ (Memory Built In Self Test - MBIST).

MBIST là một thành phần bên trong chip nên nó có thể giảm giá thành cho sản phẩm, MBIST cũng không yêu cầu việc nâng cấp hay bảo hành nên đã giảm được chi phí cho hệ thống, đồng thời muốn MBIST hoạt động với tốc độ cao thì có thể tái sử dụng các tần số hoạt động của bộ nhớ.

Phần lõi (IP) của của kiến trúc MBIST có thể được tái sử dụng trên các bộ nhớ khác để tiết kiệm chi phí thiết kế, tính linh hoạt của kiến trúc MBIST khá cao khi nó có thể được sử dụng để kiểm tra trên nhiều cấp độ bộ nhớ khác nhau thông qua khối kiến trúc khối điều khiển của MBIST. MBIST giải quyết được tính phức tạp của hệ thống MBOST bằng cách cung cấp trực tiếp tín hiệu điều khiển cho quá trình kiểm tra trên bộ nhớ, sau đó thực thi và trả về kết quả sau khi kiểm tra, chính điều nay tiết kiệm được thời gian kiểm tra trên MBIST.

MBISR (Built In Self Repair) sẽ là một giải pháp nhằm tối ưu cho hệ thống nâng cao năng suất một cách tốt nhất cho hệ thống với khả năng tự động sửa lỗi tuy nhiên trước khi đó cần phải phát hiện vị trí lỗi trên hệ thống thông qua kiến trúc MBIST.

Tuy nhiên theo chiều ngược lại, MBIST lại yêu cầu thêm:

- Phần diện tích trên chip để dành riêng cho MBIST

- Một số chân IO để giao diện với MBIST
- Đòi hỏi người thiết kế phải có kinh nghiệm trong quá trình thiết kế phần cứng cho bộ nhớ

Thường đặc trưng của kiến trúc MBIST là tích hợp chung với bộ nhớ nên sẽ có thời gian trễ để đi qua bộ phân kênh tín hiệu trên bộ nhớ, với thời gian tầm 200ps, đây sẽ là một vấn đề nếu người thiết kế không xem nó như là một yêu cầu của kiến trúc kiểm tra sẽ dẫn tới quá trình đọc ghi của bộ nhớ phát sinh lỗi. Kiến trúc BIST sẽ được thiết kế với các thuật toán cố định cho nên sau khi thực hiện công đoạn sản xuất chip thì rất khó để chỉnh sửa nếu muốn sử dụng các mô hình test mới để phát hiện ra lỗi mới, hay cần thiết là đi cấu hình lại thì việc này sẽ tốn thêm thời gian cho việc kiểm tra. Các yêu tố về timing và routing sẽ ảnh hưởng tới bộ nhớ khi có kèm theo MBIST, tuy nhiên vấn đề này có thể giải quyết được nhờ thực hiện đi layer với từ 6-8 lớp.

1.2 Tình hình nghiên cứu trong và ngoài nước

Kiến trúc vi mạch MBIST là hướng tiếp cận mới trong việc kiểm tra và đảm bảo chất lượng bộ nhớ trong qúa trình sản xuất, tuy nhiên ở mức độ nghiên cứu trong nước thì hầu như chưa có bài báo hay công trình khoa học hoàn chỉnh nào được công bố chính thức. Các đề tài chỉ dừng lại ở cấp độ báo cáo chuyên đề, đồ án tốt nghiệp.

Trên thế giới, do những ưu điểm của kiến trúc MBIST mang lại nên nó đã được nghiên cứu sâu rộng trên khía cạnh khác nhau, theo xu hướng hiện nay thì các đề tài nghiên cứu tập trung phát triển các giải thuật mới hoặc chỉnh sửa các giải thuật hiện có hiện tại để đa dạng hóa giải thuật trong việc xử lý như bảng khảo sát 1.1.

[13] tập trung trong việc chỉnh sửa và ứng dụng thuật toán March C - để đi phát hiện ra các lỗi xuất hiện trên bộ nhớ của hệ thống nhúng. [14] tiếp cận các thuật toán March sau khi được chỉnh sửa nhỏ bên trong giải thuật để ứng dụng lên kiến trúc MBIST microcode. Đa phần các giải thuật kiểm tra liên quan tới giải thuật March đã phát hiện được các lỗi như bên dưới.

Bảng 1.1 - Thống kê về các nghiên cứu MBIST từ 2005-2013

Năm	**Hội nghị**	**Tên bài báo**	**Bộ nhớ kiểm tra**	**Giải thuật**
2006	IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems	Low Power SoC Memory BIST	RAM	- Modification March C
2005	System Theory, 2005. SSST '05. Proceedings of the Thirty-Seventh Southeastern Symposium on	A System for Automated Built-In Self-Test of Embedded Memory Cores in System-on-Chip	Embedded Single port and dual port RAM: synchronous and asynchronous	- March LR - March Y
2006	ISOCC 2006	A New BIST Architecture for Word Oriented Memory	RAM	- March C, - March AB - APNPSF test algorithm
2012	International Journal of Soft Computing and Engineering (IJSCE)	Design of Improved Built-In-Self-Test Algorithm (8n) for Single Port Memory	DRAM, SRAM, ROM	March Y
2012	Department of Computer Science and Engineering,"Gheorghe Asachi" Technical University of Iași, Romani	A Multibackground March Test for Static Neighborhood PatternSensitive Faults in Random-Access Memories	RAM	March 76N
2011	International Journal of Scientific & Engineering Research,	Comparative Simulation of MBIST using MarchTest Algorithms	RAM, ROM	March BDN, March Y, March SS, March C-, March AB, March RAW, March LR
2010	2nd International Conference on Signal Processing Systems (	MBIST design and implementation of a H.264/AVC video decoder chip	RAM and ROM	March17n
2008	2008 International SoC Design Conference	A Programmable Memory BIST for Embedded Memory	RAM, ROM	March X, C-, A, B, Zerro One

2011		A Novel Access Scheme for Online Test in RFID Memories	EEPROM	March C and March SS
2011	International Conference on Communication Technology and System Design 2011	Area Overhead and Power Analysis of March Algorithms for Memory BIST	RAM	March X, C-, A, B, Zerro One, LA, AB, Y
2012	International Journal of Electrical and Computer Engineering	Modified March C - Algorithm for Embedded Memory Testing	RAM	March C modification
2013	International Journal of Computer Science and Mobile Computing	High Speed FSM-based programmable Memory Built-In Self-Test (MBIST) Controller	RAM	March C+ algorithm
2009	Industrial Electronics & Applications, 2009. ISIEA 2009. IEEE Symposium on	An overview of microcode-based and FSM-based programmable memory built-in self test (MBIST) controller for coupling fault detection	RAM	March SS

Bảng 1.2 - Thống kê các dạng lỗi có thể phát hiện được bằng giải thuật March

Giải thuật	**Các dạng lỗi có thể phát hiện được**
MATS+	SAF
March Y	SAF, TF, ADF, some CFs, some linked TFs
MARCH C-	SAF, TFs, RDF, IRF, ADF, CFs.
March X	SAF, TF, ADF, some CFs
March LR	Also linked faults
March SR	SAF, TF, CFs, IRF,RDF, DRDF, SOF
March A	SAF, TF, ADF, some CFs, some linked CFs
March B	SAF, TF, SOF, IRF, RDF, ADF, some linked TFs
March LA	SAF, TF, ADF, CFs, some linked faults
March AB	dRDF, dIRF, dDRDF, dCFds, dCFrd, dCFdrd, dCFir, static linked faults

Mặc dù là các dạng lỗi xuất hiện ngày càng nhiều, tuy nhiên xét về mặt cơ bản thì các mô hình lỗi có nét tương đồng, nên phần lớn các hướng tiếp cận về mặt thuật toán là chỉ đơn thuần chỉnh sửa các thuật toán hiện có nhằm tăng tính hiệu quả

trong việc phát hiện lỗi, còn theo như bảng thống kê 1.1 các nghiên cứu gần đây để phát hiện ra giải thuật hoàn toàn mới thì hầu như chưa có.

Mà cũng theo một nghiên cứu trong [1]-[4] đã chỉ ra rằng sự thiếu sót của một số giải thuật March hays MATS+ là không bao hàm được hết tất cả các mô hình lỗi xảy ra trên bộ nhớ, đặc biệt là các lỗi liên quan tới sự ảnh hưởng của các cell nhớ xung quanh. Từ bảng khảo sát 1.3 độ bao phủ các thuật toán cho thấy để tăng độ bao phủ cho các thuật toán cần kết hợp nhiều các thuật toán lại với nhau.

Bên ngoài hướng tiếp cận về mặt thuật toán, kiến trúc MBIST còn mở rộng hay tối ưu hóa để tiết kiệm diện tích và công suất dành cho MBIST: [15] tối ưu khối tạo địa chỉ (address generator). [16] xây dựng lại kiến trúc cho các thanh ghi bên trong các khối của MBIST theo kiểu trực giao nhằm tiết kiếm dung lượng xử lý cho các khối bên trong kiến trúc MBIST. [17] bằng cách thay đổi cấu trúc hoạt động của bộ đếm của khối điều khiển trong kiến trúc MBIST, hay [18] tập trung tối ưu kiến trúc trên MBIST để giảm thiểu phần chiếm dụng diện tích trên bộ nhớ. Với hướng tiếp cận này, việc đa dạng hóa các hướng nghiên cứu để cải thiện kiến trúc MBIST mở ra nhiều cải tiến mới, nhưng nó cũng đòi hỏi một thách thức lớn trong việc tìm tòi hướng đi mới và những thách thức mới khi xu hướng công nghệ ngày càng phát triển mạnh.

Bảng 1.3 - Khảo sát độ bao phủ của các thuật toán [1] [4]

Giải thuật	**Các dạng lỗi**							
	SAF	**TF**	**CFin**	**CFid**	**CFdyn**	**NPSF**		
						A	**P**	**S**
MATS	X							
MATS++	X	X						
MARCH C-	X	X	X	X	X			
MARCH-A	X	X	X					
TLSNPSF	X					X		
TLAPNPSF	X	X				X	X	X
TDANPSF	X							X

Xét đến khía cạnh dung lượng bộ nhớ thì vấn đề về giới hạn dung lượng bộ nhớ rất ít được đề cập, do khi có yêu cầu về việc đa dạng hóa trong việc kiểm tra các dung lượng khác nhau của bộ nhớ thì đòi hỏi yêu cầu cao về việc kiểm soát địa chỉ xuất ra trong suốt quá trình kiểm tra. [7] đã đề xuất kiểm tra chỉ riêng đối với một cấu hình duy nhất 1Kx8, hay [6] các cấu hình 1Kx48, 4Kx16, 8Kx32 sẽ được kiểm tra bởi MBIST 8Kx48 tuy nhiên với một cấu hình kiểm tra duy nhất thì điều này dẫn đến việc lãng phí tài nguyên cho kiến trúc MBIST và làm chậm thời gian xử lý của các khối bên trong kiến trúc MBIST.

Dựa trên những trên khảo sát trên đây thì tính cấp thiết của đề tài được đặt ra là làm sao có thể phát hiện được các mã lỗi xuất hiện trên bộ nhớ và kiểm tra được trên các dung lượng bộ nhớ khác nhau. Với nhiệm vụ đó luận văn sẽ xây dựng một kiến trúc MBIST để kiểm tra SRAM có dung lượng thay đổi từ 1Kx8 cho tới 64Kx64, đồng thời sử dụng các thuật toán MarchA, MarchC- và TLAPNPSF để bao hàm tất cả các mô hình lỗi có thể xảy ra trên SRAM. Mặc dù theo bảng 1.2, thuật toán MarchA không cover được hết các mô hình lỗi, nhưng nó cũng được tích hợp vào trong đề tài như một hướng tiếp cận mới trong việc tích hợp nhiều thuật toán vào cùng một kiến trúc MBIST, và đối với một số chức năng kiểm tra đơn giản, thì MarchA cũng được sử dụng để tạo ra một kiến trúc MBIST không chiếm dụng nhiều diện tích và là cơ sở để thiết lập ra môi trường tích hợp sẵn có các thuật toán để tạo ra kiến trúc MBIST theo yêu cầu.

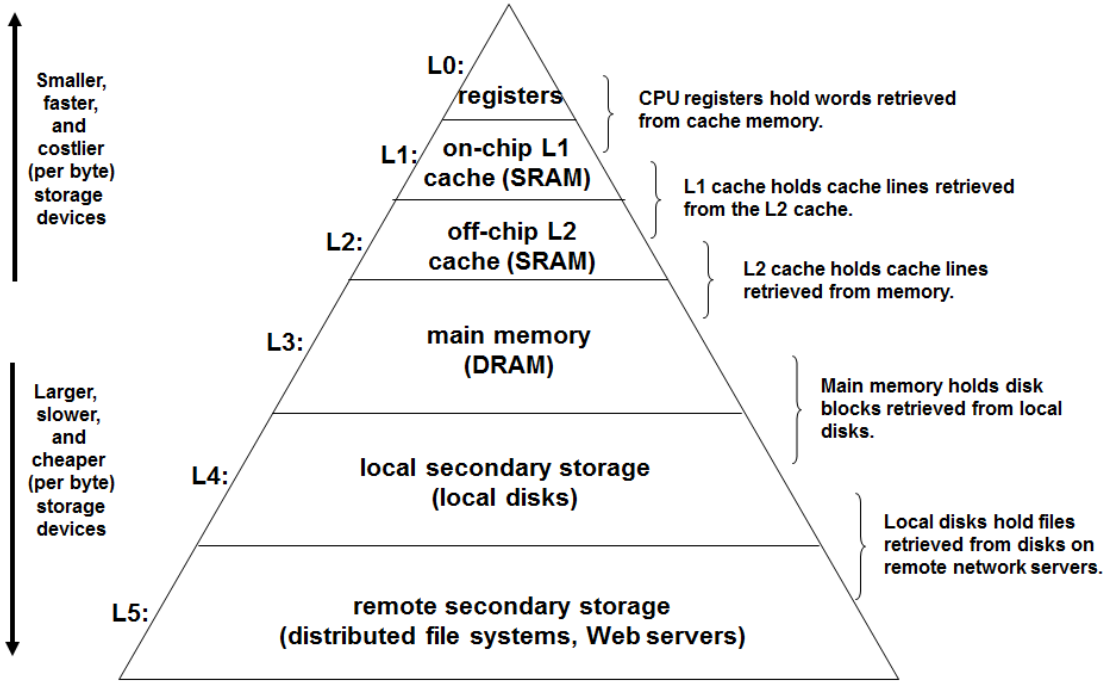
Chương 2: CƠ SỞ LÝ THUYẾT VÀ ĐỀ XUẤT KIẾN TRÚC MBIST

Giải thuật MarchC-, TLAPNPSF, MarchA sẽ giúp cho khả năng bao quát hết tất cả các mô hình lỗi. Việc ứng dụng các giải thuật này vào bên trong kiến trúc được thiết kế thông qua các mô hình trạng thái máy.

2.1 Cấu trúc bộ nhớ bán dẫn

2.1.1 Các loại bộ nhớ bán dẫn

Mô hình tháp như hình 2.1 cho thấy sự khác biệt trong các ứng dụng cũng như tốc độ của bộ nhớ nội



Hình 2.1 - Tốc độ tương ứng các bộ nhớ nội

- **Bộ nhớ truy cập ngẫu nhiên động (DRAM)**: Mật độ cao nhất, nhưng thời gian truy cập chậm khoảng 20ns. Các bit nhớ được lưu trữ như là trên một tụ điện duy nhất, nên bộ nhớ phải được làm mới (nạp lại điện tích trên tụ) thông thường mỗi 2, 4, hoặc 6ms, nếu thông tin trên bộ nhớ (giá trị điện tích trên tụ lưu trữ) bị mất.

- **Bộ nhớ truy cập ngẫu nhiên tĩnh (SRAM):** Tốc độ nhanh nhất, với một thời gian truy cập khoảng 2ns. Bit được lưu trữ trong chốt cross-coupled, và bộ nhớ không cần phải được làm mới.
- **Cache DRAM (CDRAM-SDRAM):** Kết hợp cả hai SRAM và DRAM trên cùng một chip, để đẩy nhanh chuyển giao giữa khối bộ nhớ cache nhanh SRAM và chậm DRAM.
- **Bộ nhớ chỉ đọc (ROM):** Nội dung mỗi bit được lập trình bởi sự hiện diện hay vắng mặt của một bóng bán dẫn vào thời điểm sản xuất, và không bị mất thông tin khi nguồn nuôi tắt đi.
- **Bộ nhớ lập trình và xóa được (EPROMs):** ROM có thể được lập trình trong lĩnh vực này. Toàn bộ nội dung được xóa bằng cách áp dụng ánh sáng cực tím, và sau đó các EPROM có thể được lập trình lại.
- **Bộ nhớ lập trình và xóa bằng điện (EEPROM):** ROM lập trình được lập trình trong nhiều lĩnh vực khác nhau, và từ trong chúng có thể được lựa chọn xóa bằng điện tử.

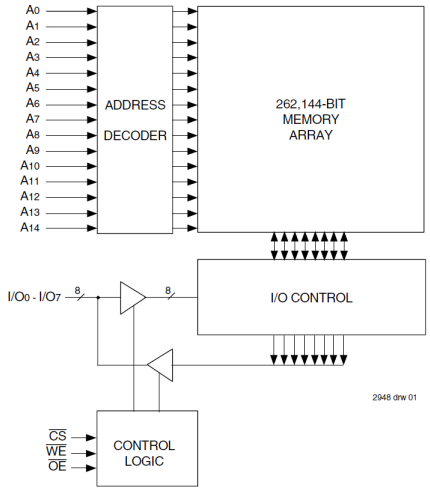
Các loại bộ nhớ khác nhau đảm nhận các vai trò cũng như đáp ứng các tốc độ khác nhau. Ngoài ra ở mỗi loại bộ nhớ sẽ được đảm nhận để phân chia theo số lượng cổng hay độ dài và chiều sâu của từ nhớ.

2.1.2 Cấu trúc cơ bản bên trong bộ nhớ

Kiến trúc cơ bản của một bộ nhớ gồm hai thành phần chính là khối giải mã và khối nhớ trung tâm. Khối giải mã bao gồm các mạch logic là chủ yếu nhằm giải mã các đường địa chỉ cũng như các đường dữ liệu ra hoặc vào đúng với các vị trí nhớ tương ứng cho từng cột và từng hàng. Tương ứng như trong hình 2.2 đó là các khối “giải mã địa chỉ và khối điều khiển kiểm soát việc đọc ghi

Khối nhớ trung tâm chính là tập hợp các cell nhớ. Các cell nhớ lưu trữ một bit nhớ tương ứng. Mặt khác các cell nhớ này được sắp xếp theo một thứ tự hàng/cột nhất định tương ứng với việc giải mã bởi các thành phần logic kể trên. Tuỳ thuộc vào từng loại bộ nhớ mà các kiến trúc cell nhớ sẽ khác nhau.

Đồng nghĩa với việc tăng kích thước bộ nhớ nội là sự khó khăn trong việc kiểm soát sự chính xác các cell nhớ bên trong bộ nhớ. Việc sai lệch giá trị bất cứ một cell nhớ nào bên trong bộ nhớ đều sẽ ảnh hưởng đến toàn bộ giá trị của phần dữ liệu đọc ra vì thông thường ở cấp độ người sử dụng thì mỗi lần truy xuất bộ nhớ tương ứng từng hàng hoặc từng cột chứ không đơn lẽ một cell nhớ nào. Lỗi truy xuất bộ nhớ thường là lỗi xảy ra do một cell nhớ nào đó gây ra. Lỗi này xảy ra là do quá trình thiết kế hoặc sản xuất nhưng chủ yếu là do sản xuất.



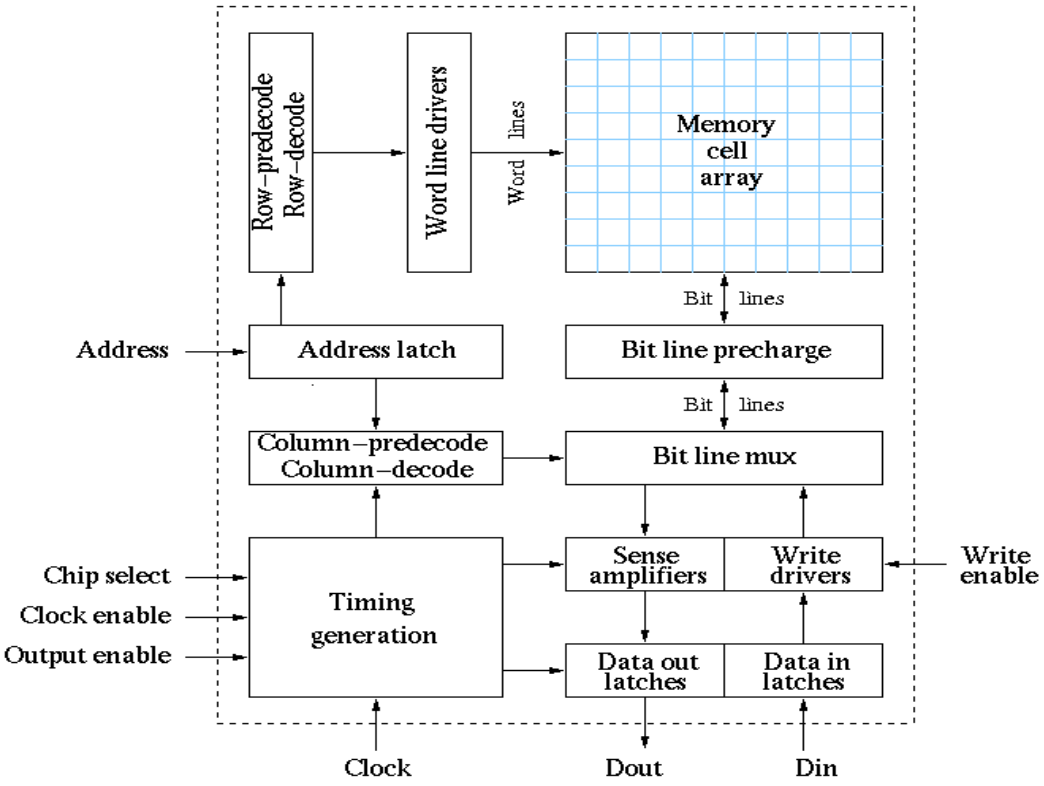
Hình 2.2 - Kiến trúc tổng quát bên trong bộ nhớ

2.2 Các lỗi thường xảy ra trên bộ nhớ SRAM

2.2.1 Cấu trúc bộ nhớ SRAM và các nguyên nhân gây ra lỗi

Hình 2.3 là cấu trúc của bộ nhớ SRAM, nên khi có lỗi xảy ra trên bộ nhớ thì các khối thường xuất hiện lỗi nhất là

- Khối giải mã địa chỉ
- Bộ điều khiển việc đọc và ghi
- Hoặc là mảng các ô nhớ bị lỗi.



Hình 2.3 - Cấu trúc bên trong của bộ nhớ SRAM

Lỗi bộ nhớ xảy ra khi hệ thống thực hiện hành vi là không chính xác hoặc bị gián đoạn, đó là những biểu hiện của lỗi trong bộ nhớ. Một lỗi là hiện nay trong hệ thống khi có sự khác biệt về đặc tính vật lý của tín hiệu trước và sau khi thực hiện việc so sánh.

Các mô hình lỗi xảy ra trên bộ nhớ có thể là vĩnh viễn (tồn tại vô thời hạn, có thể được mô hình với một mô hình lỗi) hoặc không vĩnh viễn (chỉ tồn tại một phần của thời gian, và xảy ra một cách ngẫu nhiên, không được xác định mô hình lỗi).

+ *Lỗi vĩnh viễn*: Có nghĩa là các lỗi xảy ra do phần cứng bị sai. Một ví dụ tiêu biểu là khi ghi vào một cell nhớ với giá trị là 1 nhưng khi đọc ra lại là 0. Những lỗi được gây ra bởi các cơ chế sau đây:

- Kết nối tệ về điện (thiếu hoặc thừa).
- Hỏng linh kiện (khiếm khuyết mặt nạ khi chế tọa IC hoặc silicon)
- Cháy dây dẫn trong chip.

- Kết nối bị ăn mòn.
- Lỗi logic trong quá trình thiết kế.

+ *Lỗi tạm thời*: xuất hiện trong một khoảng thời gian ngắn, và xảy ra một cách ngẫu nhiên. Nó không được xác định rõ bởi một mô hình lỗi. Nhưng các nhà thiết kế hệ thống bộ nhớ xử lý những vấn đề này bằng cách bao gồm dự phòng thông tin, hoặc mã sửa lỗi dư thừa, trong mỗi vị trí trên bộ nhớ. Vì vậy, nó là cách để phát hiện sự sai sót trên hệ thống bộ nhớ, và để tự động sửa chữa các lỗi không vĩnh viễn này. Lỗi không vĩnh viễn là do điều kiện môi trường, trong khi lỗi vĩnh viễn là nguyên nhân của điều kiện chủ quan đã biết do quá trình thiết kế hay sản xuất. Dưới đây là những nguyên nhân phổ biến của lỗi không vĩnh viễn do các điều kiện khách quan:

- Tia vũ trụ
- Tia Alpha
- Ô nhiễm không khí
- Độ ẩm (làm ngắn dây tạm thời)
- Nhiệt độ (gây ra lỗi trên bộ logic tạm thời)
- Áp suất (gây ngắn/hở mạch)
- Rung động (gây hở mạch tạm thời)
- Sự thay đổi từ nguồn cung cấp (gây ra lỗi điện áp)
- Nhiễu điện tử
- Ảnh hưởng từ tĩnh điện

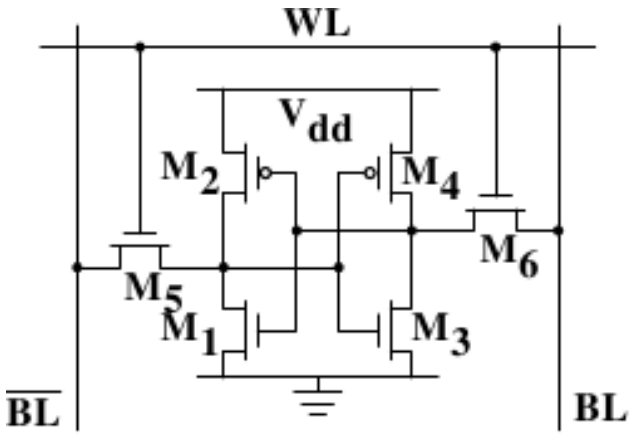
2.2.2 Các mô hình lỗi trên bộ nhớ SRAM

Hầu hết các mô hình lỗi liên quan trên SRAM sẽ được chi tiết hóa của từng dạng mô hình lỗi.

2.2.2.1 Stuck-at-faults (SAF)

Stuck-at faults xảy ra khi nội dung tại một ô nhớ hay các ô nhớ trên một hàng bị giữ một giá trị cố định 0 (SA0) hoặc 1 (SA1). Điều này có nghĩa là nội dung ô nhớ đó không thể thay đổi được.

Đây được xem là lỗi phổ biến nhất của quá trình sản xuất bộ nhớ. Nguyên có thể đến từ khâu thiết kế layout ban đầu. Việc đặt các cell cũng như cách đi dây trong mạch đang gặp vấn đề. Bên cạnh đó, sự sai sót trong quá trình sản xuất cũng mang đến hệ quả tương tự.

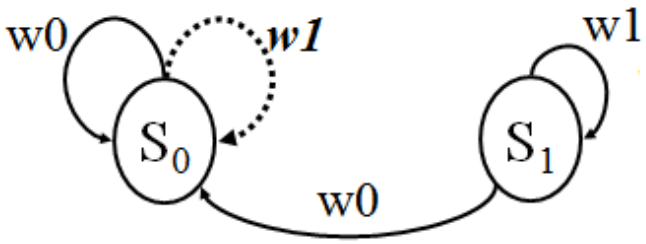


Hình 2.4 - Cell nhớ của SRAM

Cho ví dụ Cell nhớ Sram ở hình 2.4, Stuck-at fault xảy ra trên cell nhớ này là khi giá trị lấy ra từ BL và /BL luôn luôn là 1 hoặc là 0. Điều đó có nghĩa là hai tín hiệu này bị ngắn mạch lên nguồn hoặc ngắn mạch xuống đất. Nói cách khác là MOSFET M1 và MOSFET M3 không được xả dòng điện tích xuống đất hay MOSFET M2 và MOSFET M4 luôn bị nối tắt lên nguồn. Điều này khiến hai kiến trúc inverter giữ trạng thái nhớ không thể nào thay đổi được giá trị khi được yêu cầu đọc vào hay ghi vào.

2.2.2.2 Transistion faults (TF)

Transistion faults là một trường hợp đặc biệt của SA. Khi một ô nhớ có sự chuyển nội dung 0 → 1 hoặc 1 →0 trong quá trình ghi lên các cell nhớ. Giả sử có sự chuyển nội dung ô nhớ 0 → 1 thì nội dung ô nhớ được giữ cố định là 1 sau đó, hoặc có sự chuyển nội dung ô nhớ từ 1 →0 thì nội dung ô nhớ đó được giữ cố định là 0 sau đó.



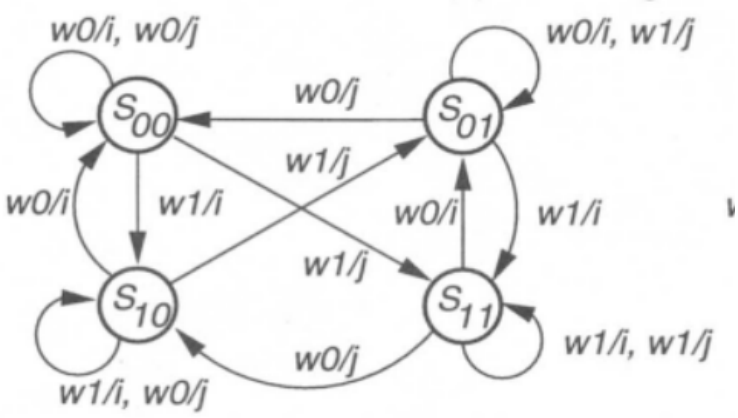
Hình 2.5 - Chuyển trạng thái của cell nhớ

2.2.2.3 Coupling faults (CF)

Coupling faults xảy ra khi một ô nhớ i nào đó có sự chuyển trạng thái 0 → 1 hoặc 1→ 0 dẫn đến giá trị ô nhớ j bị thay đổi theo. Lỗi này được chia làm ba lỗi nhỏ với mô tả chi tiết như sau:

a/ Inversion Coupling faults (CFin)

Inversion Coupling faults làm nội dung ô nhớ j bị đảo ngược khi có sự chuyển nội dung của ô nhớ i từ 0 → 1 hoặc từ 1 → 0.

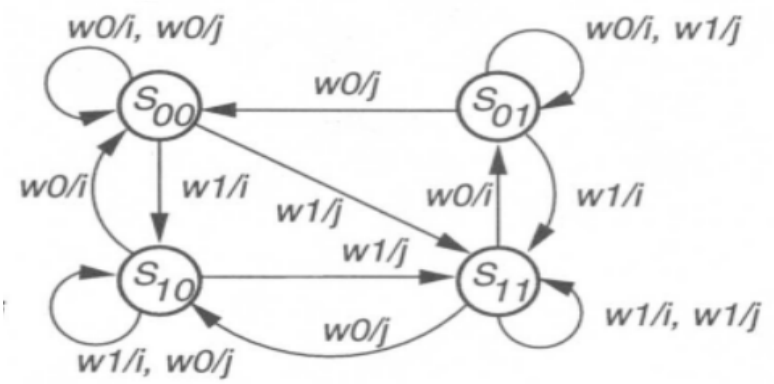


Hình 2.6 - Chuyển trạng thái của CFin

Giả sử việc ghi 1/0 lên các cell i/j được mô tả qua các trạng thái ở hình vẽ 2.6. Khi đó, sự chuyển trạng thìa từ S00 sang S11 trong quá trình ghi 1 vào cell j thì nội dung cell i bị thay đổi.

b/ Idempotent Coupling faults (CFid)

Idempotent Coupling faults làm nội dung ô nhớ j bị đặt ở giá trị cố định 0 hoặc 1 nếu có sự chuyển nội dung ô nhớ i từ 0 → 1 hoặc từ 1→ 0.

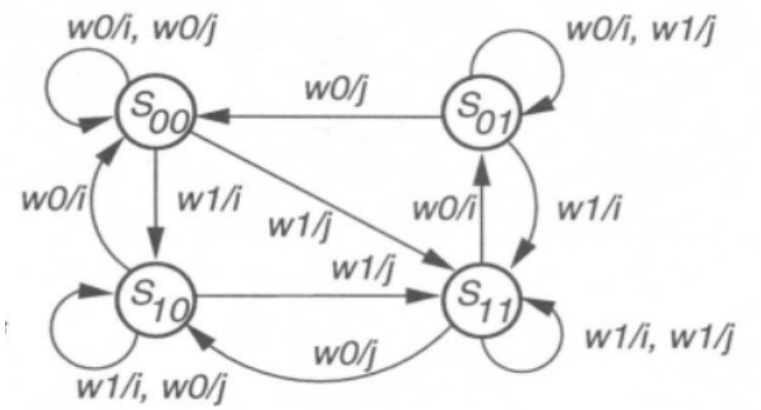


Hình 2.7 - Chuyển trạng thái của Cid

Như hình 2.7, giả sử có sự chuyển trạng thái từ S11 sang S10 của ô nhớ j thì nội dung ô nhớ i bị đặt cố định 1 hoặc 0.

c/ Dynamic Coupling faults (CFdyn)

Dynamic Coupling faults xảy ra giữa hai dòng khác nhau của ô nhớ. Hoạt động đọc hoặc ghi ở một ô nhớ trên dòng này làm thay đổi nội dung ô nhớ trên dòng khác. Và nội dung ô nhớ trên dòng thứ 2 bị thiết lập cố định không thay đổi được.



Hình 2.8 - Chuyển trạng thái của CFdyn

Ngoài ra có một số lỗi thường gặp trên bộ giải mã địa chỉ, thời gian truy cập

Lỗi ADF (Adress Decoder Fault)

Trong bộ nhớ nếu bộ truy xuất đường địa chỉ bị lỗi thì sẽ dẫn đến một số lỗi như bên dưới:

+ Với địa chỉ đã được đưa ra nhưng không có ô nhớ nào được truy cập

+ Có cùng một địa chỉ nhưng nhiều ô nhớ được truy cập cùng lúc tại một thời điểm

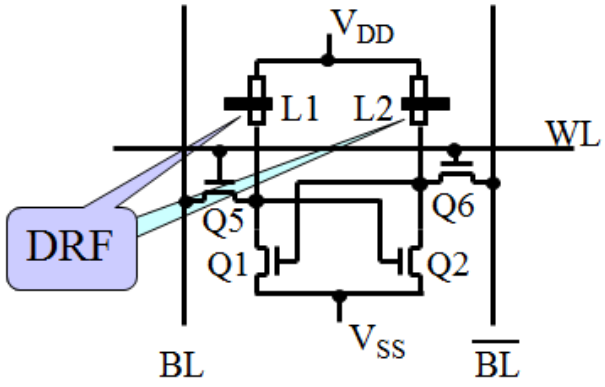
+ Một ô nhớ được truy cập bởi nhiều địa chỉ truy cập tới

+ Một địa chỉ nhưng có thể được truy cập tới với nhiều ô nhớ khác nhau

Lỗi DRF (Data Retention Fault)

Các ô nhớ của bộ nhớ vẫn hoạt động với giá trị đọc và ghi được nhưng thời gian duy trì cho mức logic đó trên ô nhớ không kéo dài được lâu tới giá trị của ô nhớ bị thay đổi. Ví dụ như ô nhớ n lưu giữ được giá trị logic 1 sau khoảng thời gian Tdelay thì thay đổi thành mức logic 0.

Việc thay đổi các mức logic bên trong các ô nhớ có các bộ chốt giá trị nhưng các bộ này hoạt động không bình thường dẫn tới các ô nhớ bị thay đổi sau khoảng thời gian bị trì hoãn.



Hình 2.9 - Mô hình lỗi DRF

Lỗi RDF (Read Destructive Fault)

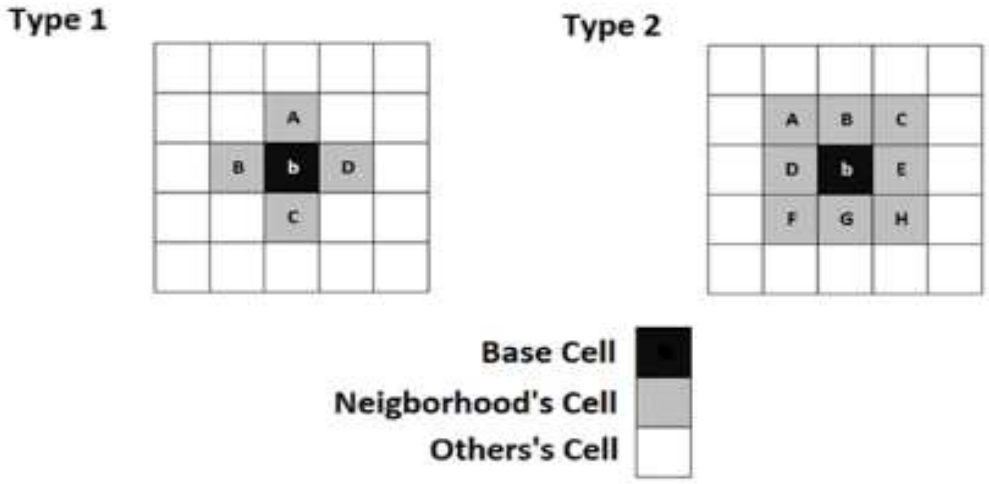
Các cell nhớ cần thời gian ổn định giữa thời gian đọc và ghi, nên khi các ô nhớ yêu cầu hoạt động đọc diễn ra nhưng do thời gian ổn định giữa hoạt động đọc và ghi dẫn tới giá trị trên cell nhớ bị thay đổi cho nên các giá trị sau khi đọc xong bị sai.

Lỗi IRF (Incorrect Read Fault)

Đây là lỗi mà bộ đọc thực hiện các hoạt động đọc bình thường trên bộ nhớ, và các ô nhớ lưu trữ các giá trị hoàn toàn đúng. Tuy nhiên, lỗi xảy ra khi các các yêu cầu đọc giá trị ô nhớ được thực hiện nhưng các ô nhớ lại trả về giá trị lại sai.

2.2.2.4 Neighborhood Pattern Sensitive faults (NPSF)

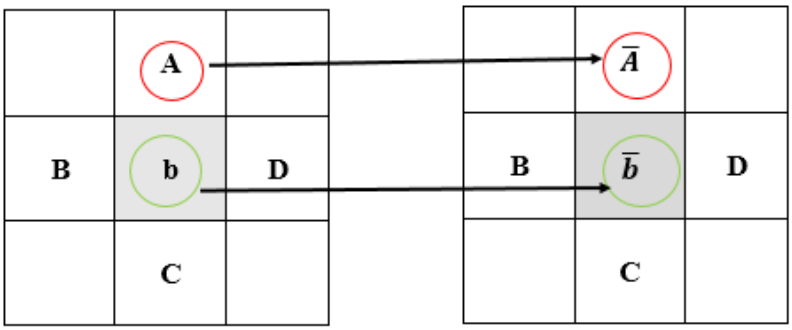
NPSF là lỗi xảy ra tại một ô nhớ bị ảnh hưởng bởi các ô nhớ kế cạnh. Có thể là bị thiết lập giá trị nội dung là 0 hoặc 1 hoặc là sự chuyển giá trị 0 → 1 hoặc 1 → 0. Gọi ô nhớ bị ảnh hưởng là *base* còn các ô nhớ xung quanh là *neighborhood* (hình 2.10) ta có chi tiết các kiểu lỗi này được trình bày ở các mục sau.



Hình 2.10- Các ô nhớ neighborhood có hai dạng

a/ Active NPSF (ANPSF)

Nội dụng ô nhớ base bị thay đổi khi một trong các ô nhớ neighborhood xung quanh có sự chuyển giá trị 0 → 1 hoặc 1 → 0.

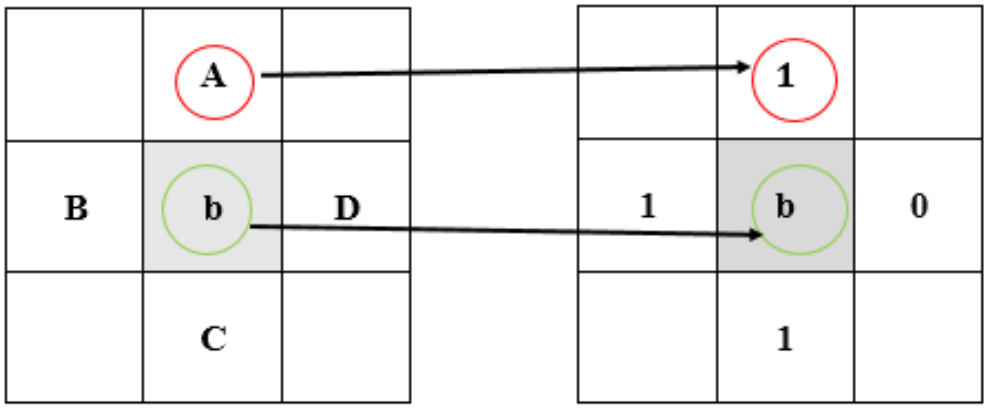


Hình 2.11- Chuyển trạng thái Active NPSF

Nội dung ô nhớ A chuyển từ 0 → 1 hoặc 1 → 0 làm nội dung của ô nhớ nền "b" bị thay đổi theo như trên hình 2.11.

b/ Passive NPSF (PNPSF)

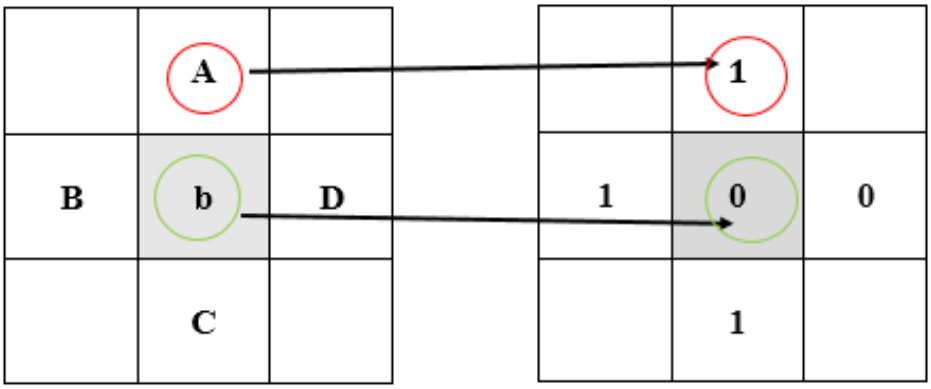
Nội dung ô nhớ base không thay đổi khi các ô nhớ neighborhood xung quanh ở những nội dung nhất định. Giả sử 4 neighborhood xung quanh có nội dung ô nhớ là 1111 thì nội dung ô nhớ base không thể bị thay đổi (bị giữ cố định).



Hình 2.12 - Chuyển trạng thái Passive NPSF

Như hình 2.12, giả sử nội dung 4 ô nhớ xung quanh ô nhớ nền ABCD = 1110, tiến hành ghi giá trị $\bar{b}$ vào ô nhớ base, nhưng không thể thực hiện được. Ô nhớ base vẫn giữ nguyên giá trị 1 khi được đọc ra sau đó.

c/ Static NPSF (SNPSF)



Hình 2.13 - Chuyển trạng thái Static NPSF

Nội dung ô nhớ base bị thiết lập ở một giá trị khi các ô nhớ neighborhood xung quanh chứa một nội dung nào đó. Giả sử 4 neighborhood xung quanh có nội dung ô nhớ là 1111 thì nội dung ô nhớ base bị thiết lập 0 hoặc 1. Lỗi này khác với ANPSF vì không cần ô nhớ neighborhood có một sự chuyển trạng thái nào hết.

Hình 2.13 cho thấy nội dung 4 ô nhớ xung quanh ô nhớ nền ABCD = 1110, tiến hành ghi giá bị $\bar{b}$ vào ô nhớ nên, nhưng nối dung bị đặt cố định 0 hoặc 1.

2.2.3 Thuật toán được sử dụng để đi phát hiện lỗi của khối MBIST

Bộ nhớ cần kiểm tra phải được thực hiện với một mô hình lỗi nhất định. MBIST kiểm tra các mảng bộ nhớ, kiểm tra các logic làm mới, phát hiện lỗi và logic điều chỉnh. Việc lựa chọn thuật toán phù hợp với yêu cầu thiết kế được đặt lên hàng đầu trong việc thiết kế MBIST. Số lượng lỗi có thể kiểm tra được, thời gian kiểm tra, tần số hoạt động của MBIST, số lượng bộ nhớ có thể kết nối để tiến hành kiểm tra là những vấn đề được quan tâm trong thiết kế MBIST.

Trước khi đi sâu vào phần định nghĩa các giải thuật MarhC- hay MarchA, ta cần làm rõ các khái niệm liên quan đến thuật toán March.

Giải thuật March là chuỗi liên tục các phần tử march (MEs), và mỗi phần tử sẽ thực hiện tuần tự trên các nội dung ô nhớ trước khi thực hiện tới các ô nhớ tiếp theo. Mỗi phần tử MEs sẽ cách nhau bởi kí tự “;”, và các thành phần bên trong mỗi phần tử MEs đó sẽ cách nhau bởi kí tự “,”. Mỗi phần tử MEs được đặt trong dấu “()”, toàn bộ các phần tử MEs sẽ đặt trong kí tự “{}”. Thứ tự thực hiện của các phần tử

được đặt bên trong giải thuật March sẽ được tăng hay giảm phụ thuộc vào các phần tử xác định địa chỉ của các giải thuật.

Để có thể làm quen và hiểu được các thuật toán, một số kí hiệu của thuật toán được giới thiệu chi tiết

- r: Một hoạt động đọc
- w: Một hoạt động ghi
- r0: Đọc 0 từ vị trí bộ nhớ
- r1: Đọc 1 từ vị trí bộ nhớ
- w0: Ghi một 0 đến vị trí nhớ
- w1: Ghi 1 đến vị trí nhớ
- ↑: Ghi 1 vào ô nhớ có nội dung 0 hoặc ô nhớ có sự chuyển trạng thái 0 →1
- ↓: Ghi 0 vào ô nhớ có nội dung 1 hoặc ô nhớ có sự chuyển trạng thái 1 →0
- ⇧: Tăng địa chỉ ô nhớ
- ⇩: Giảm đại chỉ ô nhớ
- ⇳: Tăng/Giảm địa chỉ ô nhớ

Mức độ bao phủ của từng giải thuật phụ thuộc vào các giải thuật March với nhau và cũng như là kiến trúc của bộ nhớ. Tính phức tạp của giải thuật March phụ thuộc và độ dài của giải thuật, nghĩa là số phần tử được thể hiện bên trong của từng giải thuật đó.

Ngoài ra khi một số giải thuật March được thực hiện trên nhiều port khác nhau thì sẽ thêm các chỉ số để nhận dạng các phần tử đó sẽ thực hiện trên những port khác nhau nào, cũng như là khi thực hiện trên các cell nhớ nào, và trình tự trước sau ra sao. Ví dụ như phần tử X (p/r,c) là phần tử X sẽ thực hiện các chức năng trên port p với các cell nhớ thuộc cột c và hàng r.

a. Giải thuật MarchC- [1]

(↕ (w0); ↑ (r0, w1); ↑(r1,w0);↓(r0,w1);↓(r1,w0); ↕(r0))

Là giải thuật được phát triển dựa trên giải thuật MarchC, nhưng giảm bớt đi một số phần tử bên trong. Có thể nói đây là giải thuật phát hiện được hầu hết các mô hình lỗi liên quan tới CFs, inversionCF, State CF, Dynamic CF. Đây là giải

thuật đơn giản nhất để phát hiện những mô hình lỗi liên quan tới chuỗi các địa chỉ bị lỗi AFs trong bộ giải mã địa chỉ. Đây là giải thuật được rút ngắn số lượng các phần tử thực hiện bên trong giải thuật MarchC.

b. Giải thuật MarchA [1]

(↕ (w0); ↑ (r0, w1, w0, w1); ↑(r1,w0,w1);↓(r1,w0,w1,w0);↓(r0,w1,w0))

Với mục đích nhằm đưa vào kiến trúc để thực hiện khả năng linh hoạt cho giải thuật nên giải thuật March A được thực hiện trên kiến trúc.

March A là giải thuật kiểm tra được hoàn thiện đã tinh giảm được một số phần tử bên trong để phát hiện các lỗi liên quan tới CFids, và nó cho thấy đây là giải thuật có số phần tử ít nhất để phát hiện ra các mô hình lỗi liên quan tới AFs, SAFs, TFs không có link với CFids, l ink CFids, March A có tổng tộng 15 các phần tử nhỏ bên trong giải thuật nên nó tốn mất 15.2N các giải thuật.

c. Giải thuật TLAPNPSF [21]

Bên trong các bộ nhớ sẽ có các cell nhớ trung tâm và các cell nhớ kế cận. Có hai giá trị có thể trong "ô nhớ nền-base cell" là 0 hoặc 1, và có k-1 cách lựa chọn các ô nhớ lân cận để phát hiện lỗi khi có sự chuyển trạng thái 0 → 1 (↑) hoặc 1 → 0 (↓) và 2^{k-1} khả năng cho các nội dung ô nhớ còn lại. Do vậy tổng số trường hợp là "**$2.(k-1).2.2^{k-2} = (k-1).2^k$**". Tổng số trường hợp cho việc phát hiện lỗi APNPSFs là **"$(k-1).2^k + 2^k = k.2^k$"**

Với số lượng lớn nội dung (các trường hợp) cần ghi vào bộ nhớ để kiểm tra, cần tìm cách để giảm nhưng vẫn đảm bảo chất lượng kiểm tra, đồng thời giảm đáng kể thời gian kiểm tra. Có nhiều phương pháp được đưa ra để thực hiện yêu cầu trên.

Phương pháp Halmiltonian tuần tự

Nội dung trong một k-nit Hamiltonian tuần tự khác nhau chỉ bởi 1 bit so với nội dung trước đó. Như vậy tối thiểu hóa số lượng ghi cần để tạo ra các nội dung. Mã Gray là một Hamiltonian tuần tự.

Bảng 2.1 - Các mẫu dữ liệu của Phương pháp Hamiltonian

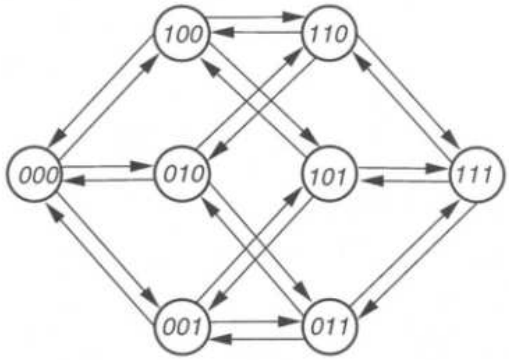
STT	DANP ABCD		STT	DANP ABCD		STT	DANP ABCD		STT	DANP ABCD	
0	0001	000↑	16	0100	0↑00	32	0010	00↑0	48	1000	10↓0
1	0011	00↑1	17	1100	↑100	33	0011	001↑	49	1100	1↑00
2	0111	0↑11	18	1101	110↑	34	1011	↑011	50	1110	11↑0
3	0101	01↓1	19	1111	11↑1	35	1111	1↑11	51	1111	111↑
4	0100	010↓	20	0111	↓111	36	1011	1↓11	52	1110	111↓
5	0110	01↑0	21	0110	011↓	37	0011	↓011	53	1100	11↓0
6	1110	↑110	22	0010	0↓10	38	0010	001↓	54	1000	1↓00
7	1010	1↓10	23	1010	↑010	39	0000	00↓0	55	1010	10↑0
8	0010	↓010	24	1110	1↑10	40	1000	↑000	56	1011	101↑
9	0110	0↑10	25	0110	↓110	41	1001	100↑	57	1001	10↓1
10	0111	011↑	26	0100	01↓0	42	1101	1↑01	58	0001	↓001
11	1111	↑111	27	0101	010↑	43	0101	↓101	59	0101	0↑01
12	1101	11↓1	28	0111	01↑1	44	0001	0↓01	60	1101	↑101
13	1100	110↓	29	0011	0↓11	45	1001	↑001	61	1001	1↓01
14	0100	↓100	30	0001	00↓1	46	1011	10↑1	62	1000	100↓
15	0000	0↓00	31	0000	000↓	47	1010	101↓	63	0000	↓000

Phương pháp giản đồ Euler

Giản đồ Euler có một nút cho mỗi mô hình k-bit của số 0 và số 1 và có một vòng cung giữa hai nút, nếu và chỉ nếu chúng khác nhau bởi đúng một bit. Khi hai nút được kết nối, chúng được kết nối bởi chỉ có hai vòng cung. Các giản đồ vòng cung tròn này tương ứng với ANPs, PNPs và APNPs của một k-bit ô nhớ kế cận.

Giản đồ Euler tuần tự đi qua mỗi vòng cung trong đồ thị đúng một lần, trong khi chuỗi Hamiltonian đi qua mỗi nút trong giản đồ chỉ đúng một lần. Việc sử dụng những trình tự để có được số lượng tối thiểu của nội dung ghi. Một chuỗi Hamiltonian cho phép áp dụng tất cả 2^k SNPs đến các ô nhớ kế cận với k giá trị ghi khởi động cho nội dung đầu tiên, tiếp theo là $2^k - 1$ lần ghi cho 2^k -1 nội dung còn lại.

Giản đồ Euler cho phép áp dụng cho tất cả "k.2^k" APNPs đến các ô nhớ kế cận với k lần ghi để khởi tạo nội dung các ô nhớ kế cận, và thêm một lần ghi cho mỗi "k.2^k" APNPS.



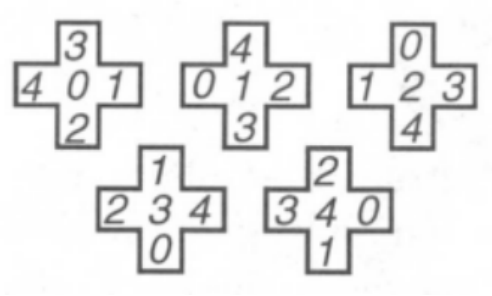
Hình 2.14 - Giản đồ Eulerian

Phương pháp Tiling

Phương pháp này sẽ quét hết các ô nhớ của bộ nhớ mà không chồng chéo các ô nhớ kế cận. Điều này được biết đến như một nhóm gọi là "tiling", và các thiết lập của tất cả các ô nhớ kế cận trong nhóm được gọi là ô nhớ "tiling". Mô tả với 5 ô nhớ được đánh số từ 0 đến 4 (có 4 ô nhớ kế cận một ô nhớ nền) được miêu tả.

Ô nhớ 2 luôn được gọi là ô nhớ nền. Tiling loại 2 gồm 9 ô nhớ (8 ô nhớ kế cận một ô nhớ nền và được đánh dấu luôn là số 4. Trong loại 1, chúng ta có n/5 ô nhớ nền. Tuy nhiên, trong khi chúng ta áp dụng các nội dung đánh giá đến n/5 ô nhớ nền, nó chỉ ra rằng chúng ta cũng áp dụng các mô hình phù hợp với bộ nhớ khi các ô nhớ là các ô nhớ nền 0 hoặc 1 hoặc 3 hoặc 4. Đây là cách giảm thiểu độ dài của nội dung kiểm tra được đặt là các ô nhớ kế cận dùng để kiểm tra ô nhớ nền từ n x 2^k xuống còn n/k x 2^k.

Dựa theo phương pháp này, mỗi ô nhớ đồng thời là ô nhớ nền vừa là ô nhớ kế cận cho các ô nhớ nền kế cạnh. Điều này cũng tương tự dành cho loại 2-ô nhớ kế cận. Số lượng sẽ được giảm rất nhiều khi tiến hành kiểm tra ANPSFS.



Hình 2.15 Ô nhớ nền 0, 1, 2, 3 và 4 loại 1 tiling neighborhood

Dựa trên các ô nhớ theo phương pháp tiling, các ô nhớ sẽ được phân chia thành 2 group khác nhau như hình 2.16.

A	b	**B**	b	**A**	b	**B**	b
b	**C**	b	**D**	b	**C**	b	**D**
B	b	**A**	b	**B**	b	**A**	b
b	**D**	b	**C**	b	**D**	b	**C**
A	b	**B**	b	**A**	b	**B**	b
b	**C1**	b	**D**	b	**C**	b	**D**
B	b	**A**	b	**B**	b	**A**	b
b	**D**	b	**C**	b	**D**	b	**C**

Nhóm - 1

b	**A**	b	**B**	b	**A**	b	**B**
C	b	**D**	b	**C**	b	**D**	b
b	**B**	b	**A**	b	**B**	b	**A**
D	b	**C**	b	**D**	b	**C**	b
b	**A**	b	**B**	b	**A**	b	**B**
C	b	**D**	b	**C**	b	**D**	b
b	**B**	b	**A**	b	**B**	b	**A**
D	b	**C**	b	**D**	b	**C**	b

Nhóm - 2

Hình 2.16 Tên ô nhớ trong phương pháp hai nhóm

Một ô nhớ đồng thời là ô nhớ nền vừa là ô nhớ kế cận của nhóm khác và ngược lại. Dựa vào tính đối ngẫu, các ô nhớ sẽ được chia thành hai nhóm, nhóm 1 và nhóm 2, như hình bàn cờ quân trắng là quân đen. Các ô nhớ nền của nhóm 1 và ô nhớ kế cận của nhóm 2 và ngược lại các ô nhớ nền của nhóm 2 là ô nhớ kế cận của nhóm 1. Mỗi nhóm có n/2 ô nhớ nền và n/2 ô nhớ kế cận được hình thành bởi 4 nhóm nhỏ A, B, C và D. Điều này chỉ thực thi cho loại 1 ở trên, bởi vì loại 2 có đến 9 ô nhớ kế cận, có cả hai ô nhớ ở giữa là 1, 3, 5 và 7 và các ô nhớ ở góc 0, 2, 6 và 8. Thật không may, tính hai mặt (một ô nhớ hoặc là ô nhớ cơ sở hoặc là ô nhớ kế cận) không được giữ ở đây.

Và phương pháp tiling với sự phân chia thành 2 nhóm khác nhau sẽ được sử dụng trong luận văn để đi chia phân vùng cho các ô nhớ và từ đó áp dụng các thuật toán kiểm tra các ô nhớ sau khi đọc ghi các giá trị liên tục lên các ô nhớ đó.

Trong luận văn phương pháp tiling với two groups được sử dụng để đi mô tả cách thức thực hiện của giải thuật TLAPNPSF.

Sử dụng phân chia thành 2 nhóm

A1	n	**B1**	n	**A1**	n	**B1**	n
n	**C1**	n	**D1**	n	**C1**	n	**D1**
B2	n	**A2**	n	**B2**	n	**A2**	n
n	**D2**	n	**C2**	n	**D2**	n	**C2**
A1	n	**B1**	n	**A1**	n	**B1**	n
n	**C1**	n	**D1**	n	**C1**	n	**D1**
B2	n	**A2**	n	**B2**	n	**A2**	n
n	**D2**	n	**C2**	n	**D2**	n	**C2**

Nhóm - 1

n	**A1**	n	**B1**	n	**A1**	n	**B1**
C1	n	**D1**	n	**C1**	n	**D1**	n
n	**B2**	n	**A2**	n	**B2**	n	**A2**
D2	n	**C2**	n	**D2**	n	**C2**	n
n	**A1**	n	**B1**	n	**A1**	n	**B1**
C1	n	**D1**	n	**C1**	n	**D1**	n
n	**B2**	n	**A2**	n	**B2**	n	**A2**
D2	n	**C2**	n	**D2**	n	**C2**	n

Nhóm - 2

n: ô nhớ neighberhood, A B C D là các ô nhớ nền.

Hình 2.17 Phân ô nhớ làm các nhóm

Ta nhận thấy các ô nhớ nền ở cùng vị trí bit ô nhớ là:

- Bit 7 và bit 3 của ô nhớ: G1_A1, G1_B2 và G2_C1, G2_D2.
- Bit 6 và bit 2 của ô nhớ: G1_C1, G1_D2 và G2_A1, G2_B2.
- Bit 5 và bit 1 của ô nhớ: G1_B1, G1_A2 và G2_D1, G2_C2.
- Bit 4 và bit 0 của ô nhớ: G1_D1, G1_C2 và G2_B1, G2_A2.

Mỗi ô nhớ nền đều bị ảnh hưởng bởi trạng thái 4 ô nhớ kế cận. Do đó ta có 24 = 16 trường hợp thay đổi của 4 ô nhớ kế cận.

	t	
x	**A**	z
	y	

Hình 2.18 Các ô nhớ kế cận của ô nhớ nền

Xét tại địa chỉ ô nhớ bất kỳ, ta ghi giá trị 0 vào A, x, y, z, t (A là ô nhớ nền; x, y, z, t là các ô nhớ kế cận; A, x, z cùng địa chỉ; t có địa chỉ ô nhớ thấp hơn địa chỉ ô nhớ của A 1 đơn vị; y có địa chỉ ô nhớ cao hơn địa chỉ ô nhớ của A 1 đơn vị). Ta lần lượt thay đổi giá trị x, y, z, t (có 16 trường hợp xảy ra). Mỗi lần thay đổi giá trị

x, y, z, t nếu giá trị ban đầu của A bằng 0 thì sau khi đọc giá trị của A, ta ghi giá trị 1 vào A và ngược lại.

Các trường hợp xảy ra của A khi ta thay đổi giá trị x, y, z, t

Bảng 2.2 - Các lỗi có thể xảy ra khi các ô nhớ kế cận thay đổi giá trị

Giá trị A ban đầu	Giá trị đọc A ban đầu	Các lỗi có thể xảy ra	Ghi giá trị A tiếp theo	Giá trị đọc A tiếp theo	Các lỗi có thể xảy ra
0	0	Passive Static Không có lỗi	1	0	Passive Static
	1	Actice Static		1	Không có lỗi
1	0	Actice Static	0	0	Không có lỗi
	1	Passive Static Không có lỗi		1	Passive Static

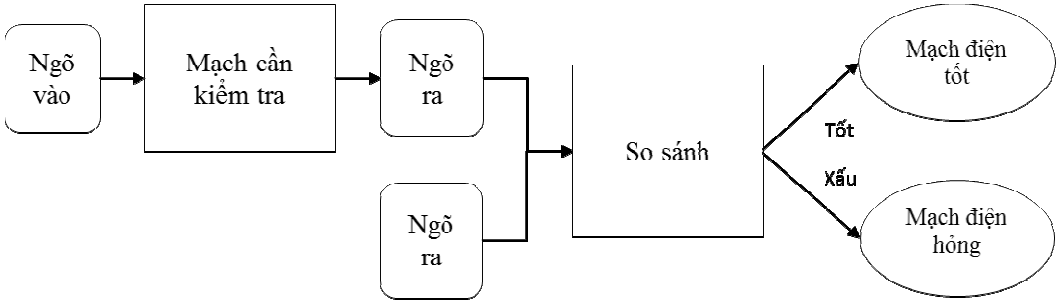
(Các lỗi có thể xảy ra trong trường hợp ghi giá trị A tiếp theo chỉ xét khi giá trị đọc A ban đầu giống với giá trị A ban đầu)

Chương 3: KIẾN TRÚC MBIST CÓ TÍNH LINH HOẠT CAO

Kiến trúc MBIST sẽ được phát triển dựa trên những giải thuật chính là MarchC- và TLAPNPSF, MarchA đồng thời xây dựng môi trường kiểm tra của những giải thuật đó. Sau đó tính linh hoạt của kiến trúc được thể hiện dung lượng bộ nhớ được kiểm tra và môi trường tự động tạo ra kiến trúc MBIST theo xu hướng chọn lựa của người dùng.

3.1 Kiến trúc cơ bản của MBIST

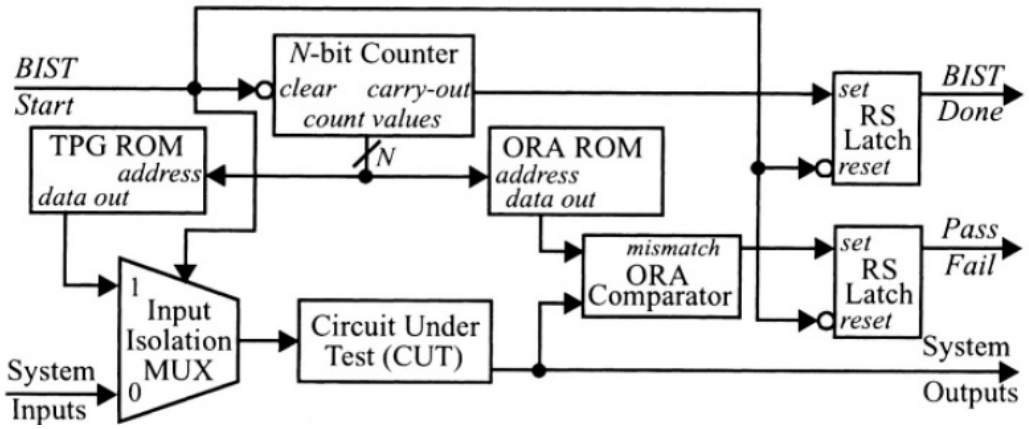
Để xây dựng được khối kiến trúc MBIST, thì lưu đồ ý tưởng thiết kế cơ bản để xây dựng cho giải thuật như hình 3.1.



Hình 3.1- Lưu đồ ý tưởng của một mạch kiểm tra cơ bản

Với kiến trúc MBIST, bộ các tín hiệu đầu vào sẽ được đưa tới mạch kiểm tra là bộ nhớ, sau đó tín hiệu tại đầu ra được so sánh với tín hiệu mong muốn thông qua một mạch so sánh. Dựa trên kết quả so sánh sẽ có thể biết được đâu là bộ nhớ tốt nếu kết quả phù hợp với kết quả đầu ra và đầu ra bộ nhớ không tốt trong trường hợp không phù hợp với nhau. Với các thiết kế MBIST, nó sẽ đảm nhận việc phát ra tín hiệu và so sánh các tín hiệu thu về từ bộ nhớ, xác định đâu là bộ nhớ tốt và đâu là bộ nhớ không tốt.

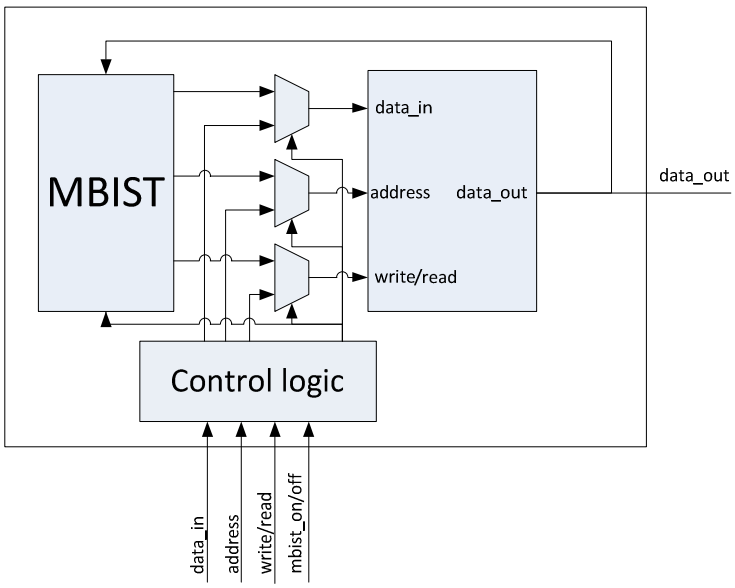
Với lưu đồ giải thuật cơ bản cho khối kiến trúc MBIST như hình 3.1, thì một mạch kiểm tra bộ nhớ - MBIST được xây dựng như hình 3.2 với các thiết bị nhớ bên trong ROM, RAM, D flip flop, mạch phân kênh và mạch so sánh để đảm bảo cho thiết bị nhớ cần kiểm tra được tính đúng đắn trên thiết bị.



Hình 3.2 - Mô tả mạch điện kiểm tra MBIST cơ bản

Để hiểu rõ được tại sao kiến trúc MBIST có thể làm việc, sơ đồ khối ở hình 3.3 trình bày cách thức hoạt động của MBIST, qua đó cho thấy khi tích cực trạng thái kiểm tra cho bộ nhớ nội ví dụ SRAM, thì bộ nhớ nội này sẽ không nhận dữ liệu từ các khối kiến trúc khác mà chỉ nhận dữ liệu từ khối kiểm tra MBIST.

Việc điều khiển được thực hiện thông qua bit xác lập trạng thái kiểm tra và tác dụng của bộ phân kênh từ bên ngoài.



Hình 3.3 - Kết nối cơ bản khối MBIST-MEMORY và các khối khác bên trong chip

Bất cứ một kiến trúc vi mạch phần cứng nào cũng bao gồm hai thành phần chính là:

- Khối điều khiển
- Khối luồng dữ liệu

Kiến trúc MBIST không nằm ngoài quy luật thiết kế phần cứng này, khối điều khiển mang tính chất quan trọng nhất đối với toàn bộ hệ thống. Đối với các hệ thống phức tạp hình thành nhiều tầng điều khiển chồng lấn lên nhau. Có nghĩa là trong một máy trạng thái điều khiển có từng trạng thái con. Tại mỗi trạng thái con tương ứng lại kích thích cho một máy trạng thái khác. Nói cách khác, các máy trạng thái đại diện cho các luồng điều khiển trong thiết kế kiến trúc vi mạch lồng vào nhau tạo một hệ thống điều khiển hoàn chỉnh và phức tạp tuỳ vào mức độ điều khiển và phạm vi ứng dụng của hệ thống.

Khối luồng dữ liệu làm nhiệm vụ thực thi các tác vụ chính của MBIST như sau:

- Tạo ra dữ liệu ghi vào bộ nhớ
- Tạo ra địa chỉ truyền đến bộ nhớ
- Tạo ra dữ liệu so sánh với dữ liệu đọc ra từ bộ nhớ
- Tạo các tín hiệu điều khiển đọc/ghi đối với bộ nhớ
- Kiểm tra giá trị đọc về
- Báo hệ thống lỗi
- Khối sửa lỗi nếu có

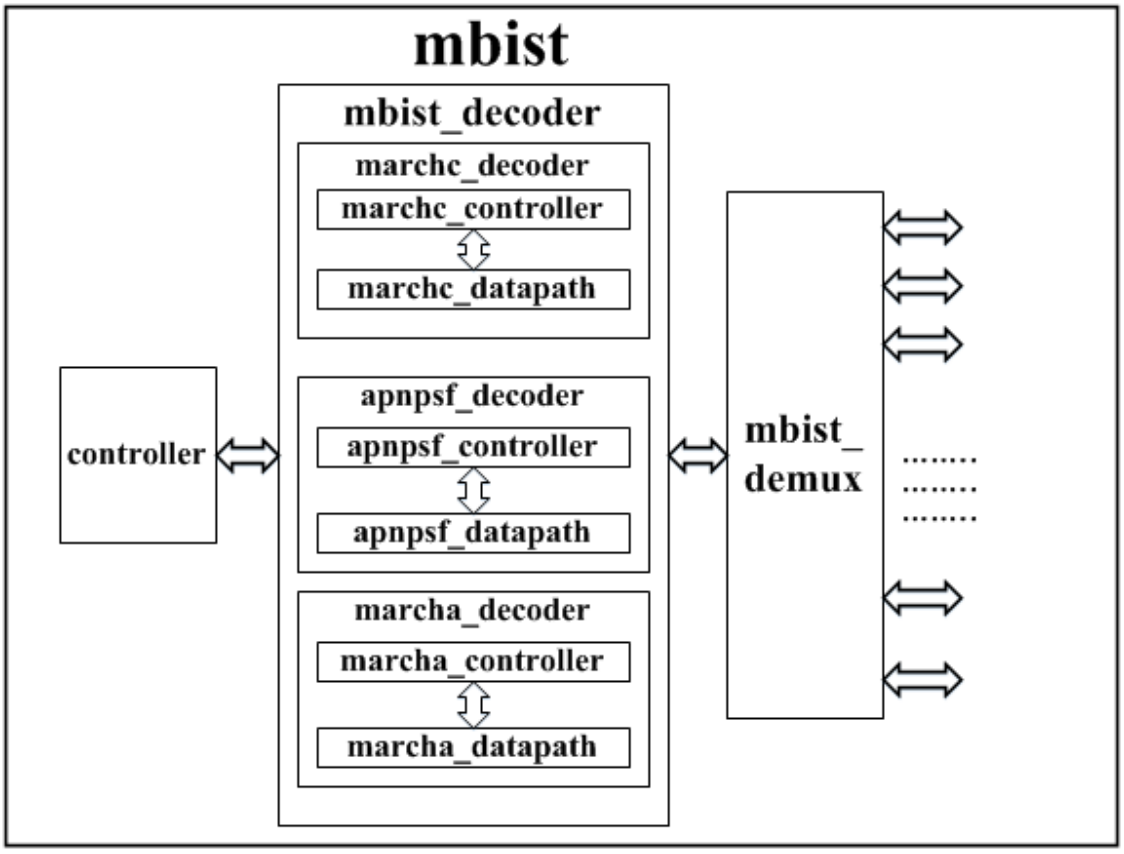
…….

Kiến trúc MBIST cơ bản bao gồm hai thành phần chính là điều khiển và luồng dữ liệu như đã đề cập. Việc tiếp cận các thuật toán sẽ được điều khiển bên trong khối điều khiển, khối dữ liệu sẽ đi phát dữ liệu theo các giải thuật đã quy định.

Dựa vào đặc điểm của từng giải thuật, khối luồng dữ liệu sẽ đảm nhận việc tạo các giá trị về địa chỉ, dữ liệu truyền đi và dữ liệu so sánh tương ứng, các thành phần bên trong của khối chọn lựa giải thuật sẽ bao gồm khối kiểm tra lỗi, khối giải mã địa được thiết kế để hỗ trợ hệ thống hoàn chỉnh.

3.2 Kiến trúc đề xuất MBIST

Mục tiêu đặt ra cho đề tài là làm sao có thể phát hiện được hết các mô hình lỗi thông thường hay xuất hiện trên bộ nhớ SRAM, cũng như là thực hiện chức năng mở rộng để kiểm tra với các bộ nhớ có dung lượng khác nhau và tổng số thiết bị nhớ được kiểm tra. Kiến trúc đề xuất như hình 3.4 sẽ giải quyết được các yêu cầu đặt ra.

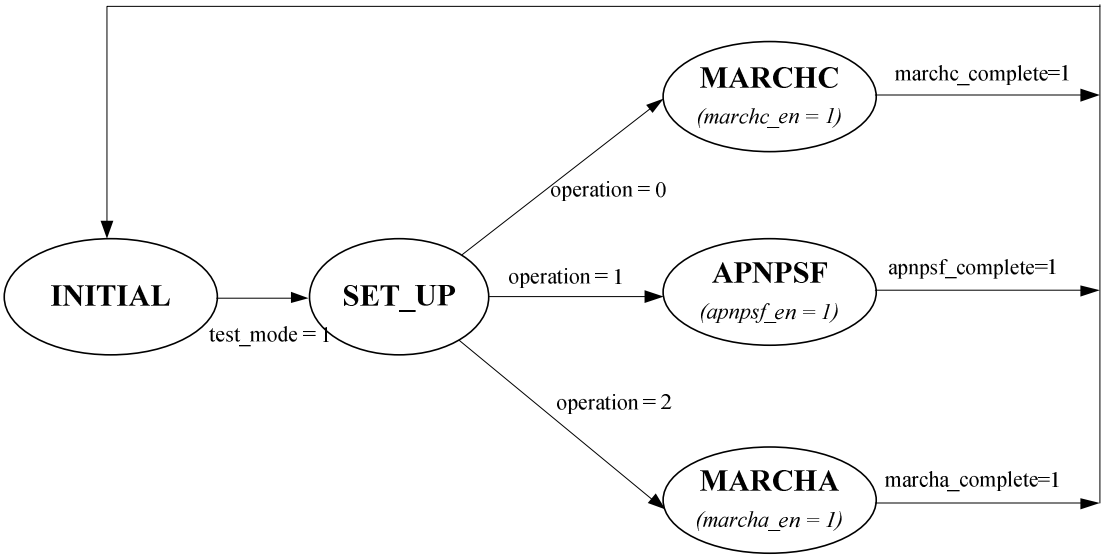


Hình 3.4 - Các khối thiết kế bên trong kiến trúc MBIST

Khối kiến trúc chính của MBIST được thiết kế với 3 khối chính và có nhiệm vụ như sau:

3.2.1 Khối controller

Các giải thuật bên trong MBIST sẽ được chọn lựa bởi khối controller



Hình 3.5 – Mô hình trạng thái máy cho khối controller

Bên trong sơ đồ trạng thái máy của khối điều khiển, thì việc lựa giải thuật của khối được thực hiện thông qua bit operation.

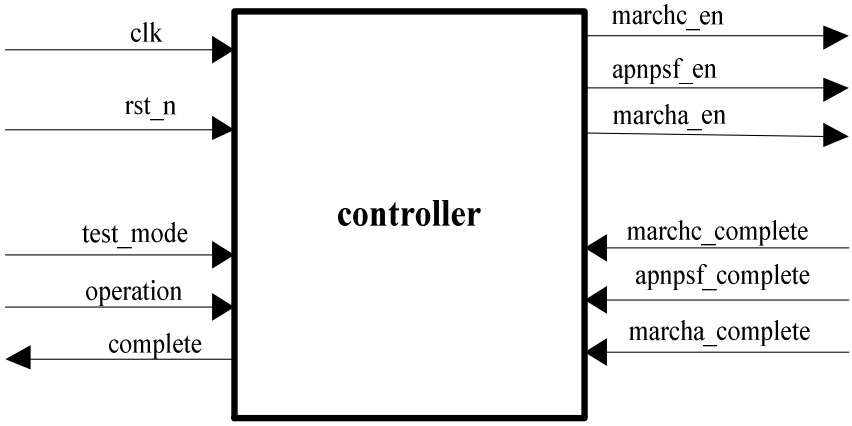
+ Nếu *opertation* =0, kích hoạt khối giải thuật MarchC thực hiện chức năng phát hiện lỗi xảy ra bên trong bộ nhớ.

+ Nếu *operation* =1, kích hoạt khối giải thuật TLAPNPSF thực hiện chức năng phát hiện lỗi xảy ra bên trong bộ nhớ.

+ Nếu *operation* =2, kích hoạt khối giải thuật MarchA thực hiện chức năng phát hiện lỗi xảy ra bên trong bộ nhớ.

Với phương pháp xây dựng khối controller bằng sơ đồ trạng thái máy, thì việc thêm nhiều thuật toán bên trong kiến trúc được thiết kế RTL bằng các trạng thái máy.

Sơ đồ in-out



Hình 3.6 - Sơ đồ in-out của khối controller với 3 giải thuật

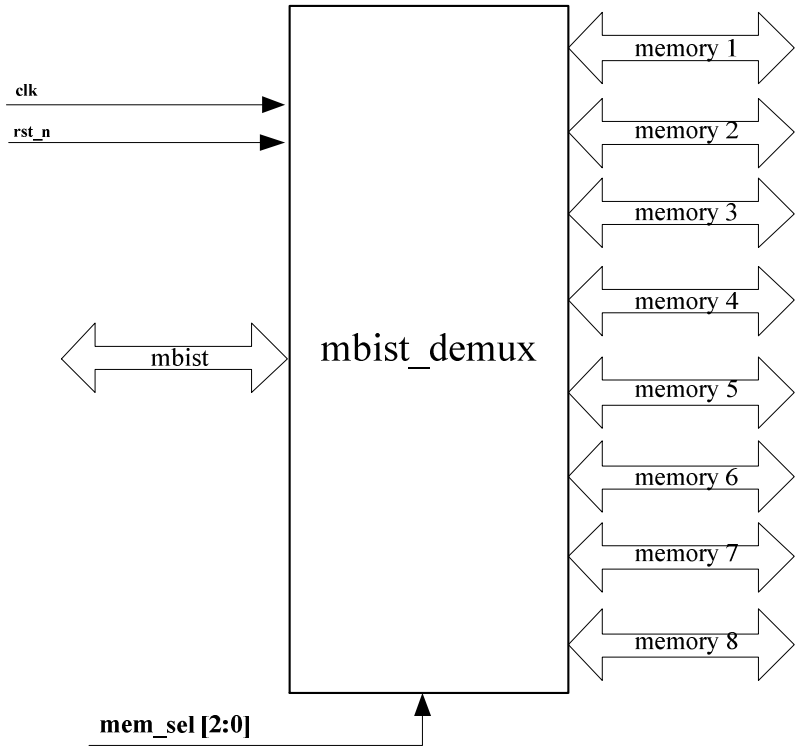
Bảng 3.1 - Định nghĩa các tín hiệu của khối controller với 3 giải thuật

STT	Loại	Tên tín hiệu	Băng thông	Mô tả
1	input	clk	1	Tín hiệu xung clock nhận từ hệ thống.
2	input	rst_n	1	Tín hiệu reset khi bắt đầu kiểm tra bộ nhớ, đặt giá trị ban đầu cho tín hiệu output.
3	input	test_mode	1	MBIST bắt đầu thực hiện khi tín hiệu bằng 1.
4	input	operation	2	Thực hiện giải thuật MARCH-C khi tín hiệu bằng 0. Thực hiện giải thuật APNPSF khi tín hiệu bằng 1. Thực hiện giải thuật MARCH-A khi tín hiệu bằng 2
5	input	marchc_complete	1	Tín hiệu bằng 1 khi thực hiện xong giải thuật MARCH C-.
6	input	apnpsf_complete	1	Tín hiệu bằng 1 khi thực hiện xong giải thuật TLAPNPSF.
7	input	marcha_complete	1	Tín hiệu bằng 1 khi thực hiện xong giải thuật MARCHA.
8	output	marchc_en	1	Tín hiệu bằng 1 khi khối controller đang ở trạng thái MARCHC-.
9	output	apnpsf_en	1	Tín hiệu bằng 1 khi khối controller đang ở trạng thái TLAPNPSF.
10	output	marcha_en	1	Tín hiệu bằng 1 khi khối controller đang ở trạng thái MARCH A.
11	output	complete	1	Tín hiệu bằng 1 khi thực hiện xong 1 trong 2 giải thuật.

3.2.2 Khối mbist_demux

Để mở rộng số lượng bộ nhớ được kết nối với MBIST, khối **mbist_demux** bên trong MBIST sẽ có thêm với nhiệm vụ chính là kết nối với các bộ nhớ ngoài, và tùy chọn bộ nhớ nào sẽ được kiểm tra và được điều khiển bởi người dùng thông qua chân **mem_sel** tương ứng với bộ nhớ đó. Số bit của **mem_sel** quy định số lượng bộ nhớ được kết nối tới MBIST.

Khối chức năng mbist_demux



Hình 3.7 - Giao diện khối mbist_demux trong MBIST kiểm tra 8 bộ nhớ

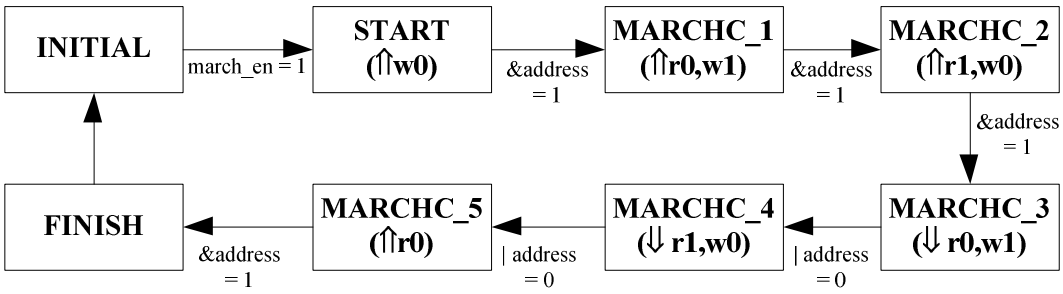
3.2.3 Khối mbist_decoder

Nhiệm vụ của kiến trúc MBIST là ghi và đọc tín hiệu, sau đó kiểm tra xem các ô nhớ có bị lỗi hay không với phương pháp kiểm tra được thực hiện tương ứng theo từng giải thuật, sẽ được đảm nhận bởi khối mbist_decoder. Ngoài ra, việc mở rộng và phát triển các giải thuật được tích hợp bên trong kiến trúc MBIST được xây dựng trên mbist_decoder.

Với mỗi giải thuật sẽ có tương ứng một khối decoder.

a. Khối marchc_decoder

Khối **marchc_decoder** thực hiện quy trình kiểm tra và phát hiện lỗi sai trên RAM dựa vào từng bước của thuật toán MarchC-, việc thực hiện quy trình kiểm tra được thực hiện thông qua sơ đồ trạng thái máy của giải thuật MarchC- theo mô tả như hình 3.8



Hình 3.8 - Mô hình trạng thái máy của thuật toán MarchC-

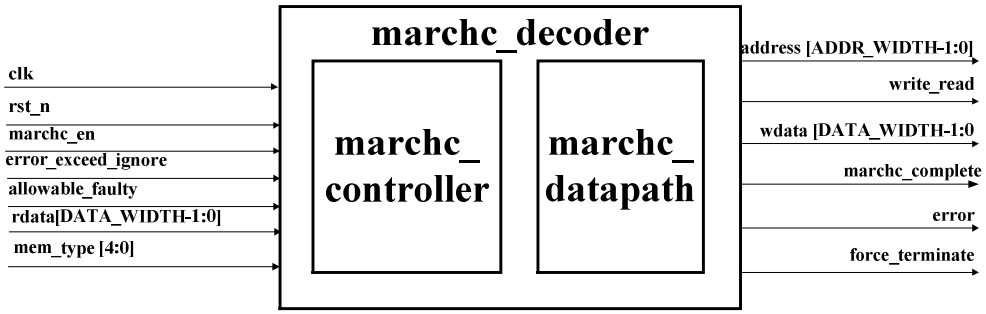
Bảng 3.2 - Các trạng thái MarchC-

Trạng thái	Ý nghĩa
INITIAL	Trạng thái ban đầu.
START	Trạng thái bắt đầu thực hiện giải thuật MarchC- Ghi giá trị 0 vào địa chỉ ô nhớ, từ địa chỉ thấp nhất đến địa chỉ cao nhất.
MARCHC_1	Đọc giá trị, so sánh, ghi giá trị 1. Thực hiện từ địa chỉ thấp nhất đến địa chỉ cao nhất.
MARCHC_2	Đọc giá trị, so sánh, ghi giá trị 0. Thực hiện từ địa chỉ thấp nhất đến địa chỉ cao nhất.
MARCHC_3	Đọc giá trị, so sánh, ghi giá trị 1. Thực hiện từ địa chỉ cao nhất đến địa chỉ thấp nhất.
MARCHC_4	Đọc giá trị, so sánh, ghi giá trị 0. Thực hiện từ địa chỉ cao nhất đến địa chỉ thấp nhất.
MARCHC_5	Đọc giá trị, so sánh. Thực hiện từ địa chỉ thấp nhất đến địa chỉ cao nhất.
FINISH	Giải thuật Giải thuật MarchC- đã thực hiện hoàn tất.

Bảng 3.3 - Sự chuyển trạng thái của MarchC-

Trạng thái hiện tại	Trạng thái kế tiếp	Lý do chuyển trạng thái
INITIAL	START	marchc_en = 1, bắt đầu thực hiện giải thuật MarchC-.
START	MARCHC_1	Trạng thái START thực hiện xong (Dựa trên giá trị địa chỉ cao nhất vì ghi 0 từ địa chỉ thấp nhất và tăng dần).
MARCHC_1	MARCHC_2	Trạng thái MARCHC_1 thực hiện xong. (Dựa trên giá trị địa chỉ cao nhất).
MARCHC_2	MARCHC_3	Trạng thái MARCHC_2 thực hiện xong. (Dựa trên giá trị địa chỉ cao nhất)
MARCHC_3	MARCHC_4	Trạng thái MARCHC_3 thực hiện xong. (Dựa trên giá trị địa chỉ thấp nhất)
MARCHC_4	MARCHC_5	Trạng thái MARCHC_4 thực hiện xong. (Dựa trên giá trị địa chỉ thấp nhất)
MARCHC_5	FINISH	Trạng thái MARCHC_5 thực hiện xong. (Dựa trên giá trị địa chỉ cao nhất)
FINISH	INITIAL	marchc_complete = 1, giải thuật MarchC- đã thực hiện hoàn tất.

Sơ đồ in-out



Hình 3.9 - Sơ đồ tín hiệu khối marchc_decoder

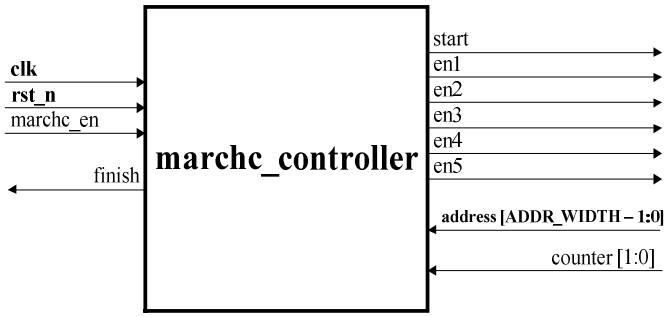
Bảng 3.4 - Định nghĩa tín hiệu trên khối marchc_decoder

STT	Loại	Tên tín hiệu	Băng thông	Mô tả
1	input	clk	1	Tín hiệu xung clock nhận từ hệ thống.
2	input	rst_n	1	Tín hiệu reset khi bắt đầu kiểm tra bộ nhớ, đặt giá trị ban đầu cho tín hiệu output.
3	input	marchc_en	1	Tín hiệu bằng 1 khi khối controller đang ở trạng thái

				kiểm tra bằng giải thuật MarchC-.
4	input	error_exceed_ignore	1	Tín hiệu bằng 0 cho phép tín hiệu force_terminate hoạt động nếu số lỗi kiểm tra vượt quá số lỗi cho phép.
5	input	allowable_faulty	ADDR_WIDTH	Số lỗi cho phép.
6	input	rdata	DATA_WIDTH	Giá trị dữ liệu đọc.
7	input	mem_type	5	Tín hiệu quy định số chân địa chỉ và dữ liệu được kiểm tra trên bộ nhớ
8	output	address	ADDR_WIDTH	Địa chỉ ô nhớ cần kiểm tra.
9	output	write_read	1	Được phép ghi khi tín hiệu bằng 1. Được phép đọc khi tín hiệu bằng 0.
10	output	wdata	DATA_WIDTH	Giá trị dữ liệu ghi.
11	output	error	1	Tín hiệu bằng 1 khi phát hiện lỗi tại địa chỉ ô nhớ bất kỳ.
12	output	force_terminate	1	Tín hiệu bằng 1 khi số lỗi kiểm tra lớn hơn số lỗi cho phép.
13	output	marchc_complete	1	Tín hiệu bằng 1 khi thực hiện xong giải thuật MarchC-.

Bên trong marchc_decoder sẽ có 2 khối thực hiện việc điều khiển và xuất luồng dữ liệu cho thuật toán MarchC- là **marchc_controller** và **marchc_datapath**

MarchC_controller

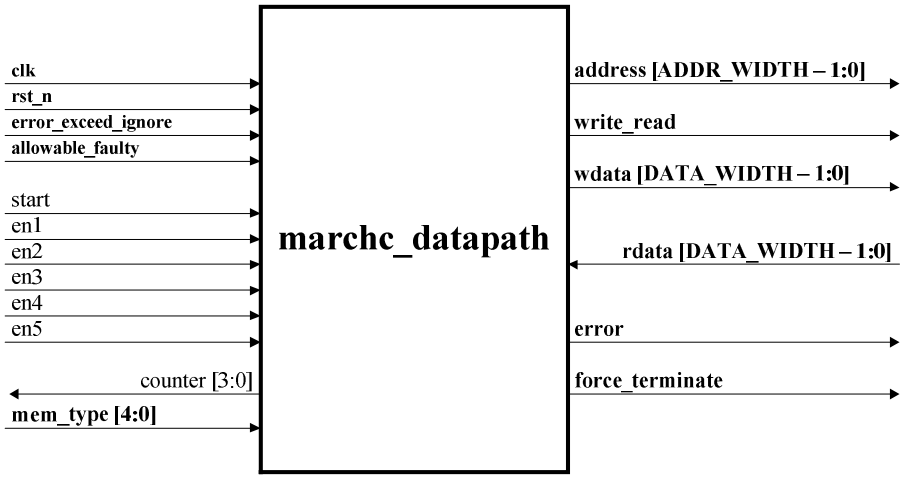


Hình 3.10 - Sơ đồ tín hiệu của khối controller

Bảng 3.5 - Định nghĩa tín hiệu của khối marchc_controller

STT	Loại	Tên tín hiệu	Băng thông	Mô tả
1	input	clk	1	Tín hiệu xung clock nhận từ hệ thống.
2	input	rst_n	1	Tín hiệu reset khi bắt đầu kiểm tra bộ nhớ, đặt giá trị ban đầu cho tín hiệu output.
3	input	marchc_en	1	Tín hiệu bằng 1 khi khối controller đang ở trạng thái MarchC-.
4	input	address	ADDR_WIDTH	Địa chỉ ô nhớ cần kiểm tra.
5	input	counter	4	Đếm từ 0 đến 4 ở bước 1 Đếm từ 0 đến 8 ở bước 2,3,4,5.
6	output	start	1	Tín hiệu bằng 1 khi khối marchc_controller đang ở trạng thái START.
7	output	en1	1	Tín hiệu bằng 1 khi khối marchc_controller đang ở trạng thái MARCHC_1.
8	output	en2	1	Tín hiệu bằng 1 khi khối marchc_controller đang ở trạng thái MARCHC_2.
9	output	en3	1	Tín hiệu bằng 1 khi khối marchc_controller đang ở trạng thái MARCHC_3.
10	output	en4	1	Tín hiệu bằng 1 khi khối marchc_controller đang ở trạng thái MARCHC_4.
11	output	en5	1	Tín hiệu bằng 1 khi khối marchc_controller đang ở trạng thái MARCHC_5.
12	output	finish	1	Tín hiệu bằng 1 khi khối marchc_controller đang ở trạng thái FINISH.

Khối marchc_datapath



Hình 3.11 - Sơ đồ tín hiệu khối marchc_datapath

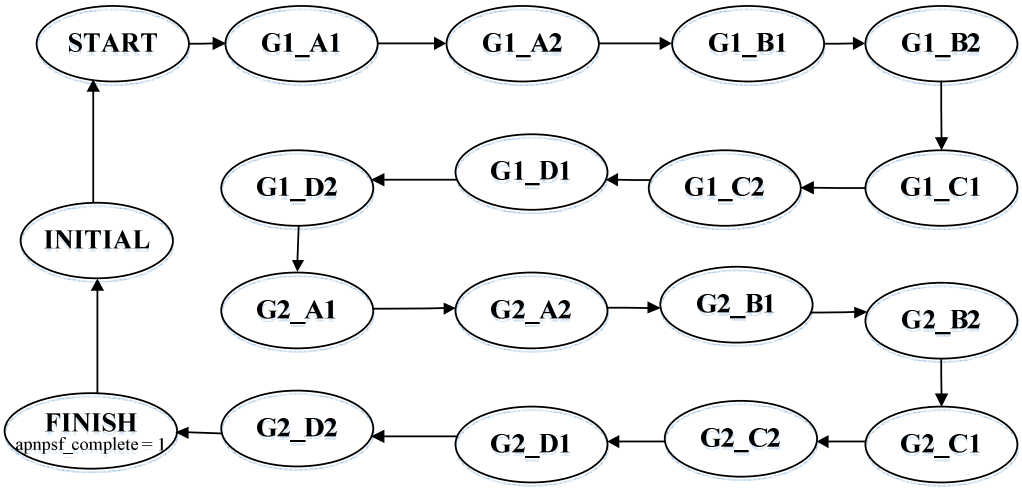
Bảng 3.6 - Định nghĩa các tín hiệu của khối marchc_datapath

STT	Loại	Tên tín hiệu	Băng thông	Mô tả
1	input	clk	1	Tín hiệu xung clock nhận từ hệ thống.
2	input	rst_n	1	Tín hiệu reset khi bắt đầu kiểm tra bộ nhớ, đặt giá trị ban đầu cho tín hiệu output.
3	input	error_exceed_ignore	1	Tín hiệu bằng 0 cho phép tín hiệu force_terminate hoạt động nếu số lỗi kiểm tra vượt quá số lỗi cho phép.
4	input	allowable_faulty	ADDR_WIDTH	Số lỗi cho phép.
5	input	start	1	Tín hiệu bằng 1 khi khối marchc_controller đang ở trạng thái START.
6	input	en1	1	Tín hiệu bằng 1 khi khối marchc_controller đang ở trạng thái MARCHC_1.
7	input	en2	1	Tín hiệu bằng 1 khi khối marchc_controller đang ở trạng thái MARCHC_2.
8	input	en3	1	Tín hiệu bằng 1 khi khối marchc_controller đang ở trạng thái MARCHC_3.
9	input	en4	1	Tín hiệu bằng 1 khi khối marchc_controller đang ở

				trạng thái MARCHC_4.
10	input	en5	1	Tín hiệu bằng 1 khi khối marchc_controller đang ở trạng thái MARCHC_5.
11	input	rdata	DATA_WIDTH	Giá trị dữ liệu đọc.
	input	mem_type	5	Tín hiệu quy định dung lượng bộ nhớ được kiểm tra
12	output	address	ADDR_WIDTH	Địa chỉ ô nhớ cần kiểm tra.
13	output	write_read	1	Được phép ghi khi tín hiệu bằng 1. Được phép đọc khi tín hiệu bằng 0.
14	output	wdata	DATA_WIDTH	Giá trị dữ liệu ghi.
15	output	error	1	Tín hiệu bằng 1 khi phát hiện lỗi tại địa chỉ ô nhớ bất kỳ.
16	output	force_terminate	1	Tín hiệu bằng 1 khi số lỗi kiểm tra lớn hơn số lỗi cho phép.
17	output	counter	4	Đếm từ 0 đến 4 ở bước 1 Đếm từ 0 đến 8 ở bước 2,3,4,5.

b. Khối apnpsf_decoder

Để thực hiện việc kiểm tra và phát hiện lỗi sai trên bộ nhớ SRAM bằng thuật toán TLAPNPSF, khối apnpsf_decoder được xây dựng. Sơ đồ trạng thái máy của thuật toán TLAPNPSF trên apnpsf decoder được mô tả như hình 3.12, mô hình trạng thái máy được xây dựng dựa trên việc mô hình hóa bộ nhớ bằng phương pháp 2 groups.



Hình 3.12 - Sơ đồ trạng thái máy của giải thuật TLAPNPSF

Bảng 3.7 - Các trạng thái của TLAPNPSF

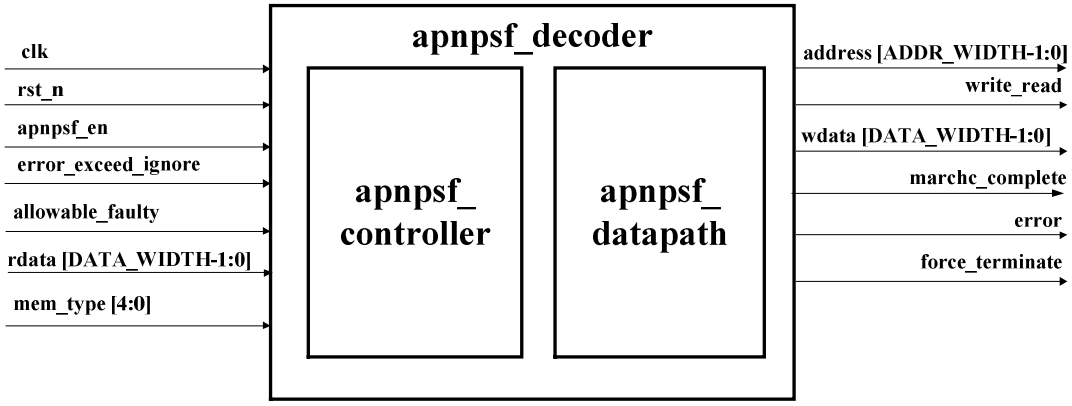
Trạng thái	Ý nghĩa
INITIAL	Trạng thái ban đầu.
START	Trạng thái bắt đầu thực hiện giải thuật APNPSF. Ghi giá trị 0 vào địa chỉ ô nhớ, từ địa chỉ thấp nhất đến địa chỉ cao nhất.
G1_A1	Thực hiện giải thuật APNPSF cho các ô nhớ thuộc Group 1, nhóm A1
G1_A2	Thực hiện giải thuật APNPSF cho các ô nhớ thuộc Group 1, nhóm A2
G1_B1	Thực hiện giải thuật APNPSF cho các ô nhớ thuộc Group 1, nhóm B1
G1_B2	Thực hiện giải thuật APNPSF cho các ô nhớ thuộc Group 1, nhóm B2
G1_C1	Thực hiện giải thuật APNPSF cho các ô nhớ thuộc Group 1, nhóm C1
G1_C2	Thực hiện giải thuật APNPSF cho các ô nhớ thuộc Group 1, nhóm C2
G1_D1	Thực hiện giải thuật APNPSF cho các ô nhớ thuộc Group 1, nhóm D1
G1_D2	Thực hiện giải thuật APNPSF cho các ô nhớ thuộc Group 1, nhóm D2
G2_A1	Thực hiện giải thuật APNPSF cho các ô nhớ thuộc Group 2, nhóm A1
G2_A2	Thực hiện giải thuật APNPSF cho các ô nhớ thuộc Group 2, nhóm A2
G2_B1	Thực hiện giải thuật APNPSF cho các ô nhớ thuộc Group 2, nhóm B1

G2_B2	Thực hiện giải thuật APNPSF cho các ô nhớ thuộc Group 2, nhóm B2
G2_C1	Thực hiện giải thuật APNPSF cho các ô nhớ thuộc Group 2, nhóm C1
G2_C2	Thực hiện giải thuật APNPSF cho các ô nhớ thuộc Group 2, nhóm C2
G2_D1	Thực hiện giải thuật APNPSF cho các ô nhớ thuộc Group 2, nhóm D1
G2_D2	Thực hiện giải thuật APNPSF cho các ô nhớ thuộc Group 2, nhóm D2
FINISH	Giải thuật APNPSF thực hiện hoàn tất.

Bảng 3.8 - Sự chuyển trạng thái của TLAPNPSF

Trạng thái hiện tại	**Trạng thái kế tiếp**	**Lý do chuyển trạng thái**
INITIAL	START	apnpsf_en = 1, bắt đầu thực hiện giải thuật TLAPNPSF.
START	G1_A1	START thực hiện xong.
G1_A1	G1_A2	G1_A1 thực hiện xong.
G1_A2	G1_B1	G1_A2 thực hiện xong.
G1_B1	G1_B2	G1_B1 thực hiện xong.
G1_B2	G1_C1	G1_B2 thực hiện xong.
G1_C1	G1_C2	G1_C1 thực hiện xong.
G1_C2	G1_D1	G1_C2 thực hiện xong.
G1_D1	G1_D2	G1_D1 thực hiện xong.
G1_D2	G2_A1	G1_D2 thực hiện xong.
G2_A1	G2_A2	G2_A1 thực hiện xong.
G2_A2	G2_B1	G2_A2 thực hiện xong.
G2_B1	G2_B2	G2_B1 thực hiện xong.
G2_B2	G2_C1	G2_B2 thực hiện xong.
G2_C1	G2_C2	G2_C1 thực hiện xong.
G2_C2	G2_D1	G2_C2 thực hiện xong.
G2_D1	G2_D2	G2_D1 thực hiện xong.
G2_D2	FINISH	G2_D2 thực hiện xong.
FINISH	INITIAL	apnpsf_complete = 1, giải thuật TLAPNPSF đã thực hiện hoàn tất.

Sơ đồ khối in-out của khối apnpsf_decoder



Hình 3.13 - Sơ đồ tín hiệu của khối apnpsf_decoder

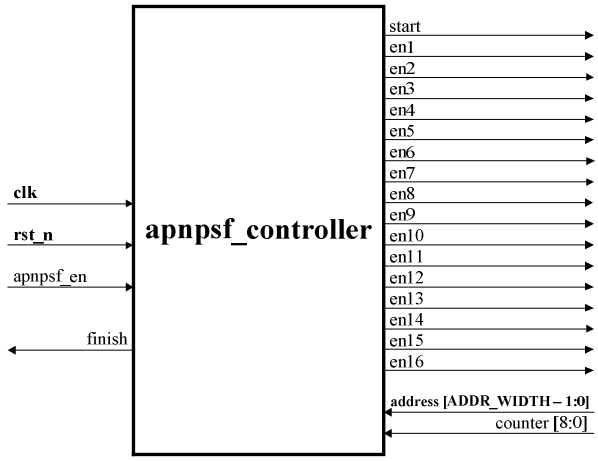
Bảng 3.9 - Định nghĩa các tín hiệu của apnpsf_decoder

STT	Loại	Tên tín hiệu	Băng thông	Mô tả
1	input	clk	1	Tín hiệu xung clock nhận từ hệ thống.
2	input	rst_n	1	Tín hiệu reset khi bắt đầu kiểm tra bộ nhớ, đặt giá trị ban đầu cho tín hiệu output.
3	input	apnpsf_en	1	Tín hiệu bằng 1 khi khối controller đang ở trạng thái TLAPNPSF.
4	input	error_exceed_ignore	1	Tín hiệu bằng 0 cho phép tín hiệu force_terminate hoạt động nếu số lỗi kiểm tra vượt quá số lỗi cho phép.
5	input	allowable_faulty	ADDR_WIDTH	Số lỗi cho phép.
6	input	rdata	DATA_WIDTH	Giá trị dữ liệu đọc.
	input	mem_type	5	Tín hiệu quy định dung lượng bộ nhớ được kiểm tra.
7	output	address	ADDR_WIDTH	Địa chỉ ô nhớ cần kiểm tra.
8	output	write_read	1	Được phép ghi khi tín hiệu bằng 1. Được phép đọc khi tín hiệu bằng 0.
9	output	wdata	DATA_WIDTH	Giá trị dữ liệu ghi.
10	output	error	1	Tín hiệu bằng 1 khi phát hiện lỗi tại địa chỉ ô nhớ bất kỳ.

11	output	force_terminate	1	Tín hiệu bằng 1 khi số lỗi kiểm tra lớn hơn số lỗi cho phép.
12	output	apnpsf_complete	1	Tín hiệu bằng 1 khi thực hiện xong giải thuật TLAPNPSF.

Bên trong khối apnpsf_decoder sẽ có 2 khối chức năng thực hiện việc điều khiển và xuất dữ liệu cho thuật TLAPNPSF.

Khối apnpsf_controller



Hình 3.14 - Sơ đồ khối apnpsf_controller

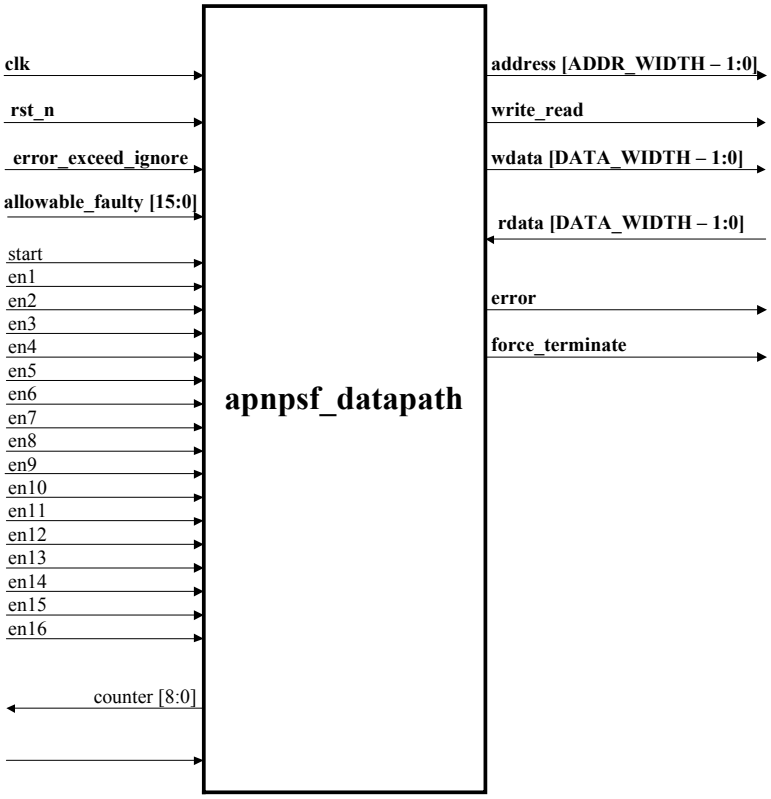
Bảng 3.10 - Định nghĩa tín hiệu sơ đồ khối apnpsf_controller

STT	Loại	Tên tín hiệu	Băng thông	Mô tả
1	input	clk	1	Tín hiệu xung clock nhận từ hệ thống.
2	input	rst_n	1	Tín hiệu reset khi bắt đầu kiểm tra bộ nhớ, đặt giá trị ban đầu cho tín hiệu output.
3	input	apnpsf_en	1	Tín hiệu bằng 1 khi khối controller đang ở trạng thái TLAPNPSF.
4	input	address	ADDR_WIDTH	Địa chỉ ô nhớ cần kiểm tra.
5	input	counter	9	Đếm từ 9’h000 đến 9’h19E ở mỗi bước, ngoại trừ bước 0 không đếm.
6	output	start	1	Tín hiệu bằng 1 khi khối apnpsf_controller đang ở trạng thái START.
7	output	en1	1	Tín hiệu bằng 1 khi khối

				apnpsf_controller đang ở trạng thái G1_A1.
8	output	en2	1	Tín hiệu bằng 1 khi khối apnpsf_controller đang ở trạng thái G1_A2.
9	output	en3	1	Tín hiệu bằng 1 khi khối apnpsf_controller đang ở trạng thái G1_B1.
10	output	en4	1	Tín hiệu bằng 1 khi khối apnpsf_controller đang ở trạng thái G1_B2.
11	output	en5	1	Tín hiệu bằng 1 khi khối apnpsf_controller đang ở trạng thái G1_C1.
12	output	en6	1	Tín hiệu bằng 1 khi khối apnpsf_controller đang ở trạng thái G1_C2.
13	output	en7	1	Tín hiệu bằng 1 khi khối apnpsf_controller đang ở trạng thái G1_D1.
14	output	en8	1	Tín hiệu bằng 1 khi khối apnpsf_controller đang ở trạng thái G1_D2.
15	output	en9	1	Tín hiệu bằng 1 khi khối apnpsf_controller đang ở trạng thái G2_A1.
16	output	en10	1	Tín hiệu bằng 1 khi khối apnpsf_controller đang ở trạng thái G2_A2.
17	output	en11	1	Tín hiệu bằng 1 khi khối apnpsf_controller đang ở trạng thái G2_B1.
18	output	en12	1	Tín hiệu bằng 1 khi khối apnpsf_controller đang ở trạng thái G2_B2.
19	output	en13	1	Tín hiệu bằng 1 khi khối apnpsf_controller đang ở trạng thái G2_C1.
20	output	en14	1	Tín hiệu bằng 1 khi khối apnpsf_controller đang ở trạng thái G2_C2.
21	output	en15	1	Tín hiệu bằng 1 khi khối apnpsf_controller đang ở trạng thái G2_D1.

22	output	en16	1	Tín hiệu bằng 1 khi khối apnpsf_controller đang ở trạng thái G2_D2.
23	output	finish	1	Tín hiệu bằng 1 khi khối apnpsf_controller đang ở trạng thái FINISH.

Khối apnpsf_datapath



Hình 3.15 - Sơ đồ tín hiệu khối apnpsf_datapath

Bảng 3.11 - Định nghĩa các tín hiệu của sơ đồ khối apnpsf_datapath

STT	Loại	Tên tín hiệu	Băng thông	Mô tả
1	input	clk	1	Tín hiệu xung clock nhận từ hệ thống.
2	input	rst_n	1	Tín hiệu reset khi bắt đầu kiểm tra bộ nhớ, đặt giá trị ban đầu cho tín hiệu output.
3	input	error_exceed_ignore	1	Tín hiệu bằng 0 cho phép tín hiệu force_terminate hoạt động nếu số lỗi kiểm

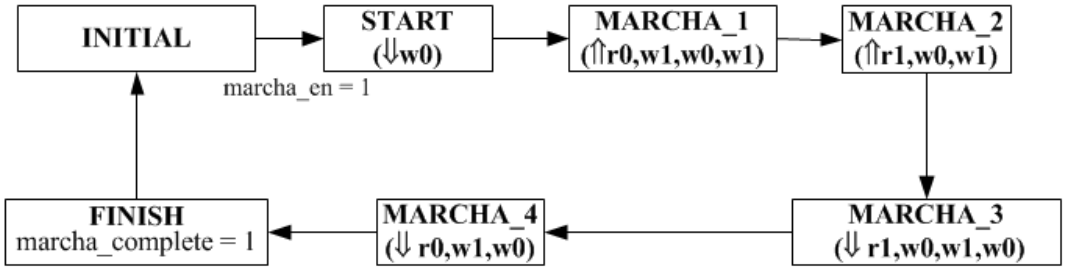
				tra vượt quá số lỗi cho phép.
4	input	allowable_faulty	ADDR_WIDTH	Số lỗi cho phép.
5	input	start	1	Tín hiệu bằng 1 khi khối apnpsf_controller đang ở trạng thái START.
6	input	en1	1	Tín hiệu bằng 1 khi khối apnpsf_controller đang ở trạng thái G1_A1.
7	input	en2	1	Tín hiệu bằng 1 khi khối apnpsf_controller đang ở trạng thái G1_A2.
8	input	en3	1	Tín hiệu bằng 1 khi khối apnpsf_controller đang ở trạng thái G1_B1.
9	input	en4	1	Tín hiệu bằng 1 khi khối apnpsf_controller đang ở trạng thái G1_B2.
10	input	en5	1	Tín hiệu bằng 1 khi khối apnpsf_controller đang ở trạng thái G1_C1.
11	input	en6	1	Tín hiệu bằng 1 khi khối apnpsf_controller đang ở trạng thái G1_C2.
12	input	en7	1	Tín hiệu bằng 1 khi khối apnpsf_controller đang ở trạng thái G1_D1.
13	input	en8	1	Tín hiệu bằng 1 khi khối apnpsf_controller đang ở trạng thái G1_D2.
14	input	en9	1	Tín hiệu bằng 1 khi khối apnpsf_controller đang ở trạng thái G2_A1.
15	input	en10	1	Tín hiệu bằng 1 khi khối apnpsf_controller đang ở trạng thái G2_A2.
16	input	en11	1	Tín hiệu bằng 1 khi khối apnpsf_controller đang ở trạng thái G2_B1.
17	input	en12	1	Tín hiệu bằng 1 khi khối apnpsf_controller đang ở trạng thái G2_B2.
18	input	en13	1	Tín hiệu bằng 1 khi khối apnpsf_controller đang ở

				trạng thái G2_C1.
19	input	en14	1	Tín hiệu bằng 1 khi khối apnpsf_controller đang ở trạng thái G2_C2.
20	input	en15	1	Tín hiệu bằng 1 khi khối apnpsf_controller đang ở trạng thái G2_D1.
21	input	en16	1	Tín hiệu bằng 1 khi khối apnpsf_controller đang ở trạng thái G2_D2.
22	input	rdata	DATA_WIDTH	Giá trị dữ liệu đọc.
	input	mem_type	5	Tín hiệu quy định dung lượng bộ nhớ được kiểm tra.
23	output	address	ADDR_WIDTH	Địa chỉ ô nhớ cần kiểm tra.
24	output	write_read	1	Được phép ghi khi tín hiệu bằng 1. Được phép đọc khi tín hiệu bằng 0.
25	output	wdata	DATA_WIDTH	Giá trị dữ liệu ghi.
26	output	error	1	Tín hiệu bằng 1 khi phát hiện lỗi tại địa chỉ ô nhớ bất kỳ.
27	output	force_terminate	1	Tín hiệu bằng 1 khi số lỗi kiểm tra lớn hơn số lỗi cho phép.
28	output	counter	9	Đếm từ 9’h000 đến 9’h19E ở mỗi bước, ngoại trừ bước 0 không đếm.

c. Khối marcha_decoder

Ngoài 2 thuật toán MarchC- và TLAPNPSF đã được tích hợp lên kiến trúc MBIST, thì giải thuật MarchA cũng sử dụng trên kiến trúc MBIST nhằm nâng cao tính đa dạng cho các giải thuật được kiểm tra và cũng như kiểm chứng được tính linh hoạt của kiến trúc MBIST khi xét trên khía cạnh số thuật toán được tích hợp.

Khối marcha_decoder sẽ kiểm tra và phát hiện lỗi trên SRAM bằng thuật toán MarchA và cũng giống như hai thuật toán đã được thực hiện, thì quá trình xây dựng kiến trúc cũng được thông qua mô hình trạng thái máy như hình 3.16



Hình 3.16 - Mô hình trạng thái máy của thuật toán MarchA

Bảng 3.12 - Các trạng thái của MarchA

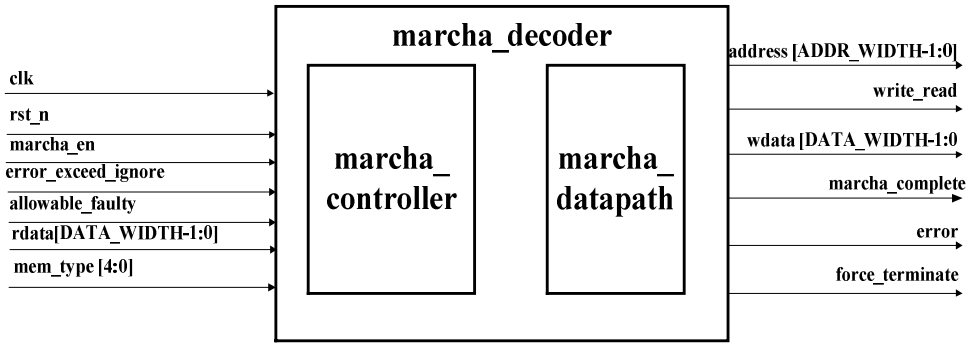
Trạng thái	Ý nghĩa
INITIAL	Trạng thái ban đầu.
START	Trạng thái bắt đầu thực hiện giải thuật MarchA Ghi giá trị 0 vào địa chỉ ô nhớ, từ địa chỉ thấp nhất đến địa chỉ cao nhất.
MARCHA_1	Đọc giá trị, so sánh giá trị 0, sau đó ghi giá trị 1, 0, 1. Thực hiện từ địa chỉ thấp nhất đến địa chỉ cao nhất.
MARCHA_2	Đọc giá trị, so sánh, ghi giá trị 1, sau đó ghi giá trị sau đó ghi giá trị 0, 1 Thực hiện từ địa chỉ thấp nhất đến địa chỉ cao nhất.
MARCHA_3	Đọc giá trị, so sánh, ghi giá trị 1, sau đó ghi giá trị sau đó ghi giá trị 0,1,0 Thực hiện từ địa chỉ cao nhất đến địa chỉ thấp nhất.
MARCHA_4	Đọc giá trị, so sánh, ghi giá trị 0, sau đó ghi giá trị sau đó ghi giá trị 1, 0 Thực hiện từ địa chỉ cao nhất đến địa chỉ thấp nhất.
FINISH	Giải thuật March-A đã thực hiện hoàn tất.

Bảng 3.13 - Sự chuyển trạng thái của MARCHA

Trạng thái hiện tại	Trạng thái kế tiếp	Lý do chuyển trạng thái
INITIAL	START	marchc_en = 1, bắt đầu thực hiện giải thuật March-A.
START	MARCHA_1	Trạng thái START thực hiện xong (Dựa trên giá trị địa chỉ cao nhất vì ghi 0 từ địa chỉ thấp nhất và tăng dần).
MARCHA_1	MARCHA_2	Trạng thái MARCHA_1 thực hiện xong. (Dựa trên giá trị địa chỉ cao nhất).
MARCHA_2	MARCHA_3	Trạng thái MARCHA_2 thực hiện xong. (Dựa trên giá trị địa chỉ cao nhất)

MARCHA_3	MARCHA_4	Trạng thái MARCHA_3 thực hiện xong. (Dựa trên giá trị địa chỉ thấp nhất)
MARCHA_4	FINISH	Trạng thái MARCHA_4 thực hiện xong. (Dựa trên giá trị địa chỉ cao nhất)
FINISH	INITIAL	marcha_complete = 1, giải thuật March-A đã thực hiện hoàn tất.

Sơ đồ khối in-out của **marcha_decoder**



Hình 3.17- Sơ đồ tín hiệu khối marcha_decoder

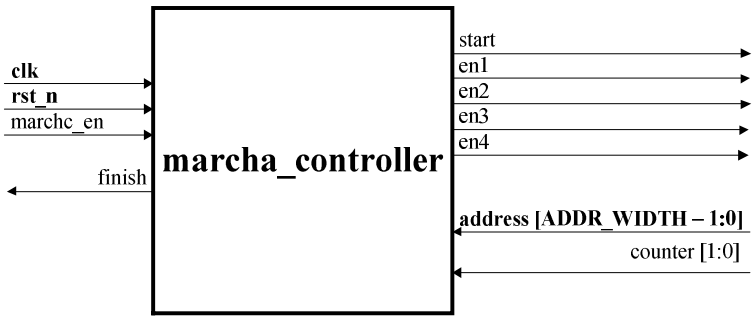
Bảng 3.14 - Sơ đồ tín hiệu khối marcha_decoder

STT	Loại	Tên tín hiệu	Băng thông	Mô tả
1	input	clk	1	Tín hiệu xung clock nhận từ hệ thống.
2	input	rst_n	1	Tín hiệu reset khi bắt đầu kiểm tra bộ nhớ, đặt giá trị ban đầu cho tín hiệu output.
3	input	marcha_en	1	Tín hiệu bằng 1 khi khối controller đang ở trạng thái MarchA.
4	input	error_exceed_ignore	1	Tín hiệu bằng 0 cho phép tín hiệu force_terminate hoạt động nếu số lỗi kiểm tra vượt quá số lỗi cho phép.
5	input	allowable_faulty	ADDR_WIDTH	Số lỗi cho phép.
6	input	rdata	DATA_WIDTH	Giá trị dữ liệu đọc.
7	input	mem_type	5	Tín hiệu quy định dung lượng bộ nhớ được kiểm tra
7	output	address	ADDR_WIDTH	Địa chỉ ô nhớ cần kiểm tra.
8	output	write_read	1	Được phép ghi khi tín hiệu

				bằng 1. Được phép đọc khi tín hiệu bằng 0.
9	output	wdata	DATA_WIDTH	Giá trị dữ liệu ghi.
10	output	error	1	Tín hiệu bằng 1 khi phát hiện lỗi tại địa chỉ ô nhớ bất kỳ.
11	output	force_terminate	1	Tín hiệu bằng 1 khi số lỗi kiểm tra lớn hơn số lỗi cho phép.
12	output	marcha_complete	1	Tín hiệu bằng 1 khi thực hiện xong giải thuật MarchA.

Bên trong khối marcha_decoder sẽ có 2 khối thực hiện quá trình điều khiểm việc kiểm tra và xuất dữ liệu cho bộ nhớ.

Khối **marcha_controller**



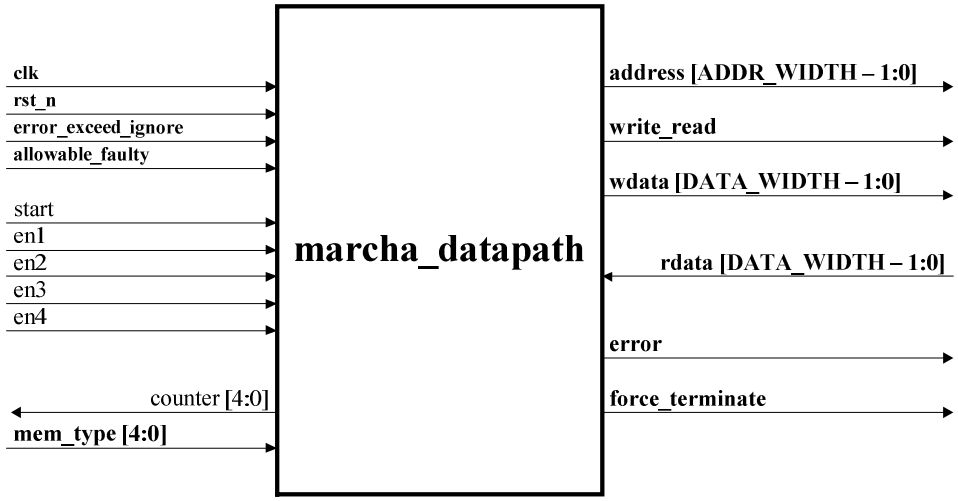
Hình 3.18 - Sơ đồ tín hiệu khối marcha_controller

Bảng 3.15 - Định nghĩa tín hiệu của khối marcha_controller

STT	Loại	Tên tín hiệu	Băng thông	Mô tả
1	input	clk	1	Tín hiệu xung clock nhận từ hệ thống.
2	input	rst_n	1	Tín hiệu reset khi bắt đầu kiểm tra bộ nhớ, đặt giá trị ban đầu cho tín hiệu output.
3	input	marcha_en	1	Tín hiệu bằng 1 khi khối controller đang ở trạng thái MARCHA.
4	input	address	ADDR_WIDTH	Địa chỉ ô nhớ cần kiểm tra.
5	input	counter	5	Đếm từ 0 đến 4 ở bước 1. Đếm từ 0 đến 16 ở bước 2,3,4.
6	output	start	1	Tín hiệu bằng 1 khi khối

				marcha_controller đang ở trạng thái START.
7	output	en1	1	Tín hiệu bằng 1 khi khối marcha_controller đang ở trạng thái MARCHA_1.
8	output	en2	1	Tín hiệu bằng 1 khi khối marcha_controller đang ở trạng thái MARCHA_2.
9	output	en3	1	Tín hiệu bằng 1 khi khối marcha_controller đang ở trạng thái MARCHA_3.
10	output	en4	1	Tín hiệu bằng 1 khi khối marcha_controller đang ở trạng thái MARCHA_4.
12	output	finish	1	Tín hiệu bằng 1 khi khối marcha_controller đang ở trạng thái FINISH.

Khối marcha_datapath



Hình 3.19 - Sơ đồ khối của các chân tín hiệu của marcha_datapath

Bảng 3.16 - Định nghĩa các tín hiệu của khối marcha_datapath

STT	Loại	Tên tín hiệu	Băng thông	Mô tả
1	input	clk	1	Tín hiệu xung clock nhận từ hệ thống.
2	input	rst_n	1	Tín hiệu reset khi bắt đầu kiểm tra bộ nhớ, đặt giá trị ban đầu cho tín hiệu output.

3	input	error_exceed_ignore	1	Tín hiệu bằng 0 cho phép tín hiệu force_terminate hoạt động nếu số lỗi kiểm tra vượt quá số lỗi cho phép.
4	input	allowable_faulty	ADDR_WIDTH	Số lỗi cho phép.
5	input	start	1	Tín hiệu bằng 1 khi khối marcha_controller đang ở trạng thái START.
6	input	en1	1	Tín hiệu bằng 1 khi khối marcha_controller đang ở trạng thái MARCHA_1.
7	input	en2	1	Tín hiệu bằng 1 khi khối marcha_controller đang ở trạng thái MARCHA_2.
8	input	en3	1	Tín hiệu bằng 1 khi khối marcha_controller đang ở trạng thái MARCHA_3.
9	input	en4	1	Tín hiệu bằng 1 khi khối marcha_controller đang ở trạng thái MARCHA_4.
11	input	rdata	DATA_WIDTH	Giá trị dữ liệu đọc.
	input	mem_type	5	Tín hiệu quy định dung lượng bộ nhớ được kiểm tra
12	output	address	ADDR_WIDTH	Địa chỉ ô nhớ cần kiểm tra.
13	output	write_read	1	Được phép ghi khi tín hiệu bằng 1. Được phép đọc khi tín hiệu bằng 0.
14	output	wdata	DATA_WIDTH	Giá trị dữ liệu ghi.
15	output	error	1	Tín hiệu bằng 1 khi phát hiện lỗi tại địa chỉ ô nhớ bất kỳ.
16	output	force_terminate	1	Tín hiệu bằng 1 khi số lỗi kiểm tra lớn hơn số lỗi cho phép.
17	output	counter	5	Đếm từ 0 đến 4 ở bước 1 Đếm từ 0 đến 2 ở bước 2,3,4.

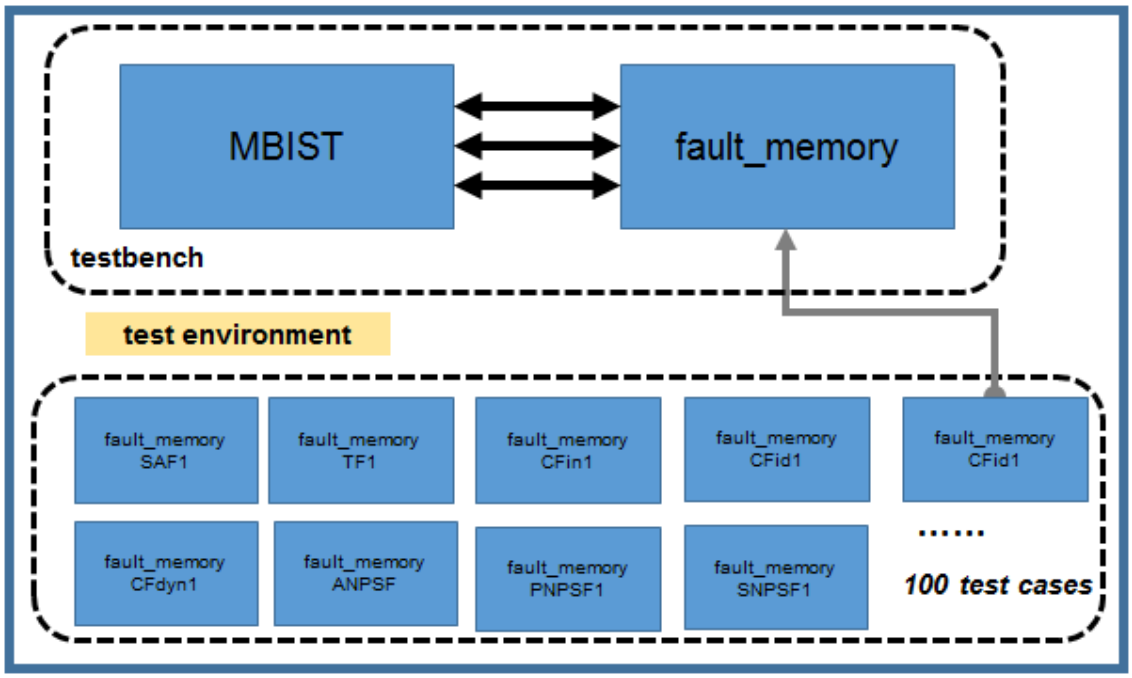
3.3 Môi trường kiểm tra độ chính xác của MBIST

Việc kiểm chứng được sự hiệu quả của giải thuật đã được tích hợp lên kiến trúc MBIST là một bước trong quy trình thiết kế ASIC, để thực hiện việc đó môi trường giả lập các lỗi xảy ra trên bộ nhớ được thiết kế theo các mô hình lỗi đã được định nghĩa trong mục 2.2.2. Môi trường giả lập sẽ chứa bộ nhớ bị lỗi và kiến trúc MBIST. Mỗi bộ nhớ sẽ đại diện cho một mô hình lỗi xảy ra trên một cell nhớ tại một hay một số vùng địa chỉ, kiến trúc MBIST sử dụng các giải thuật mà người dùng đưa vào để đi phát hiện lỗi xem có đúng là nằm trên vùng địa chỉ đó hay không.

Nhằm nâng cao độ tin cậy cho kiến trúc với các giải thuật đã có, 100 trường hợp lỗi tương ứng 100 bộ nhớ lỗi được giả lập và đưa vào môi trường kiểm tra để kiểm chứng.

Trong môi trường sản xuất công nghiệp, việc kiểm tra sẽ tốn nhiều thời gian nếu thực hiện việc kiểm tra trên 100 test case một cách lần lượt, nên để tiết kiệm thời gian và nâng cao năng suất cho việc kiểm tra, một môi trường kiểm tra tự động được xây dựng sử dụng ngôn ngữ Perl và Bash cell. Trên môi trường đó các nhà kiểm tra chỉ cần chọn lựa giải thuật nào được sử dụng, bộ nhớ có dung lượng bao nhiêu và sẽ kiểm tra trên cổng nào. Sau đó thì môi trường sẽ tự động thực hiện công việc kiểm tra.

Hình 3.20 mô tả sơ đồ giao diện giữa kiến trúc MBIST với các bộ nhớ trong môi trường kiểm tra, và các bộ nhớ lỗi sẽ đưa vào bên trong môi trường kiểm tra theo sự chọn lựa của người kiểm tra.



Hình 3.20 - Môi trường test bench kiểm tra sự hiệu quả của kiến trúc MBIST

3.4 Môi trường tạo tính linh hoạt cho MBIST

Yêu cầu đặt ra cho đề tài là xây dựng tính linh hoạt cho MBIST và để thực hiện điều này thì tính hoạt của kiến trúc MBIST được thể hiện thông qua:

+ Giải thuật nào sẽ được sử dụng để tích hợp lên chip

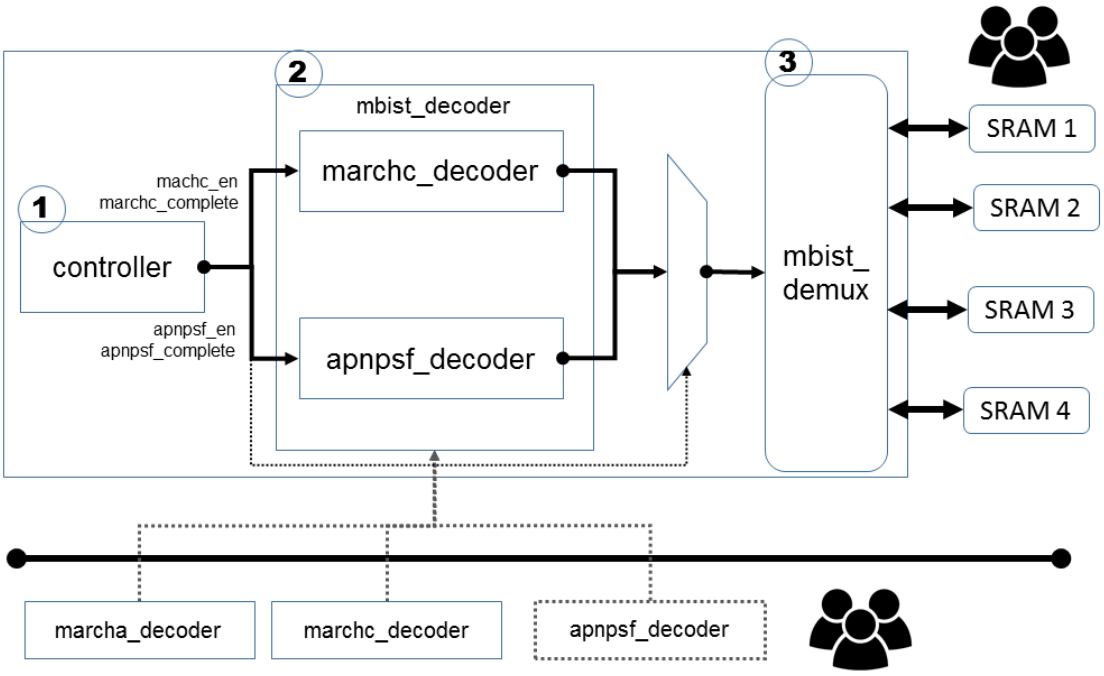
+ Quy định số lượng bộ nhớ được kết nối trên đầu ra

+ Dung lượng bộ nhớ được kiểm tra

Tính linh hoạt về dung lượng bộ nhớ sẽ được kiểm tra đã lồng ghép vào bên trong kiến trúc nên đề giải quyết những vấn đề còn lại sẽ dựa trên môi trường được sử dụng.

Tính linh hoạt của đề tài sẽ đáp ứng được sự đa dạng trong môi trường sản xuất, với mục đích là tạo ra sự thuận tiện cao nhất và nâng cao năng suất, thì môi trường giao tiếp trực tiếp với người dùng đã được xây dựng sử dụng ngôn ngữ Perl và Bash cell. Trên môi trường tự tạo ra kiến trúc MBIST, với theo từng yêu cầu của nhà sản xuất là quyết định chọn lựa giải thuật nào sẽ được sử dụng và số lượng bộ

nhớ, sau đó môi trường sẽ tự động tạo ra kiến trúc MBIST tương ứng. Mô hình của môi trường được mô tả như hình 3.21.



Hình 3.21 - Môi trường tạo ra kiến trúc MBIST

Tương ứng với sự lựa chọn từ người dùng thì các khối: Điều khiển – controller; Giải mã – Decoder; Phân /Ghép kênh – Demux/Mux sẽ tự động thay đổi cập nhật các dựa trên các thông số nhập từ nhà sản xuất hay người sử dụng.

Chương 4 : KẾT QUẢ THỰC HIỆN

Thiết kế RTL của kiến trúc MBIST sẽ được tạo ra bởi sự lựa chọn của người sản xuất. Các giải thuật được kiểm chứng ở môi trường kiểm và đồng thời kiến trúc MBIST với 3 giải thuật MarchC-, TLAPNPF, MarchA tương ứng sẽ được phân tích xuống lớp vật lí dưới dạng cổng để thấy được đặc tả kỹ thuật của kiến trúc MBIST.

4.1 Kết quả verification

Bộ công cụ VCS được của công ty Synopsys được sử dụng để đi mô tả quá trình mô phỏng. Sơ đồ cấu trúc cây thư mục như bên dưới được mô tả kiến trúc bên trong của môi trường giả lập lỗi.

Bảng 4.1- Sơ đồ cây thư mục của môi trường kiểm chứng giải thuật

```
|-- 01_script
|   |-- 01_Testbench
|   |   |-- gen_testbench.pl
|   |   `-- input
|   |       `-- testbench_skl.v
|   `-- 02_Fault_Mem
|       |-- gen_fault_mem.pl
|       `-- input
|           |-- fault_mem_0101.v
|           |-- fault_mem_0102.v
|           |-- fault_mem_072E.v
|           `-- fault_mem_072F.v
|-- 02_design
|   |-- apnpsf_controller.v
|   |-- apnpsf_datapath.v
|   |-- apnpsf_decoder.v
|   |-- controller.v
|   |-- marcha_controller.v
|   |-- marcha_datapath.v
|   |-- marcha_decoder.v
|   |-- marchc_controller.v
|   |-- marchc_datapath.v
|   |-- marchc_decoder.v
|   |-- mbist_decoder.v
|   |-- mbist_mux_demux.v
|   `-- mbist.v
```

```
|-- 03_simulate
|   |-- address.log
|   |-- collect_report.pl
|   |-- compile_list.txt
|   |-- config.txt
|   |-- DVEfiles
|   |-- memtype_defination.txt
|   |-- run
|   |-- run_all_testcase.pl
|   `-- run_one_testcase.sh |    |-- run_all_test_case.pl
```

Bảng 4.2 - Chức năng các thư mục trong cấu trúc cây thư mục

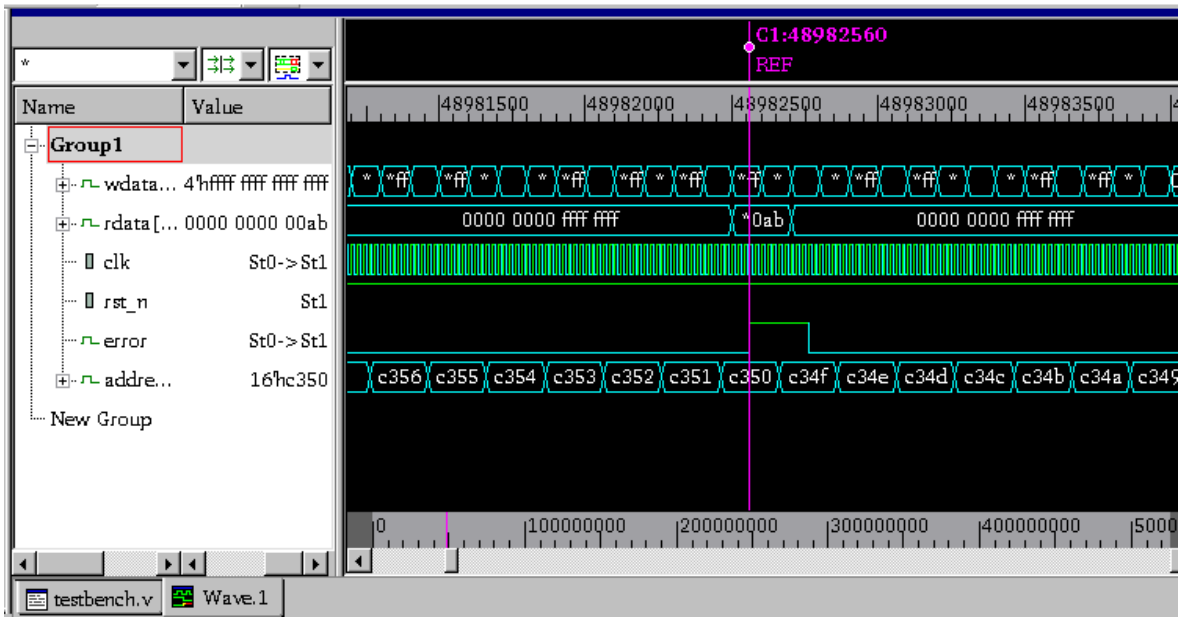
STT	Tên	Đặc tả
1	*script*	Thư mục chứa file để tạo ra testbench và bộ nhớ lỗi tương ứng cho từng trường hợp test cho MBIST
2	*design*	Thư mục chứa các file thiết kế RTLcủa kiến trúc bằng ngôn ngữ verilog
3	*simulate*	Thư mục chứa các file chạy mô phỏng cho kiến trúc MBIST theo từng trường hợp một của bộ nhớ lỗi.

- Trường hợp sử dụng MarhC- để đi phát hiện ra mode lỗi tại địa chỉ tại địa chỉ 50000 (0C350H) của bộ nhớ 64Kx32.

Bảng 4.3 - Kết quả chạy phân tích trên giải thuật MarchC-

```
VCD+ Writer D-2010.06-SP1 Copyright (c) 1991-2010 by Synopsys Inc.

 == MARCH C ARE RUNNING ==
MEMORY IS FAILED AT ADDRESS = : 050000
MEMORY IS FAILED AT ADDRESS = : 050000
MEMORY IS FAILED AT ADDRESS = : 050000
MEMORY IS FAILED AT ADDRESS = : 050000

$finish called from file "testbench.v", line 52.

$finish at simulation time          700001022

        V C S  S i m u l a t i o n  R e p o r t

Time: 700001022
CPU Time:     19.750 seconds;       Data structure size:   0.5Mb
Sun Nov 27 16:14:03 2016
CPU time: .915 seconds to compile + .144 seconds to elab + .801 seconds to link +

76.255 seconds in simulation
```

Kết quả dạng sóng tại vị trí có lỗi xảy ra trên bộ nhớ



Hình 4.1 - Dạng sóng tín hiệu tại vị trí lỗi C350H (5000D)

- Trường hợp sử dụng thuật toán MarhA để đi phát hiện ra mode lỗi tại địa chỉ tại địa chỉ 100 (64H) của bộ nhớ 1Kx64.

Bảng 4.4 - Kết quả chạy phân tích trên giải thuật MarchA

VCD+ Writer D-2010.06-SP1 Copyright (c) 1991-2010 by Synopsys Inc.

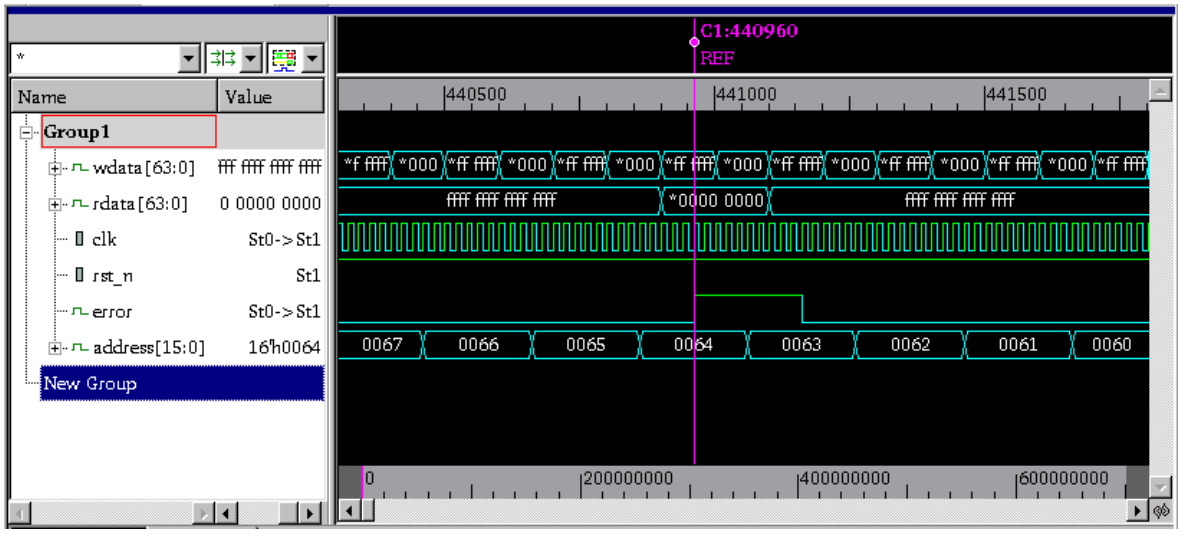
== MARCH A ARE RUNNING ==

MEMORY IS FAILED AT ADDRESS = : 0100
MEMORY IS FAILED AT ADDRESS = : 0100
$finish called from file "testbench.v", line 52.

$finish at simulation time 700001022
VCS Simulation Report

Time: 700001022
CPU Time: 44.030 seconds; Data structure size: 0.0Mb
Sun Nov 27 16:16:57 2016
CPU time: .618 seconds to compile + .072 seconds to elab + .516 seconds to link + 91.525 seconds in simulation

Kết quả dạng sóng tại vị trí có lỗi xảy ra trên bộ nhớ



Hình 4.2 - Dạng sóng tín hiệu tại vị trí lỗi 64H (100D)

- Trường hợp sử dụng TLAPNPSF để đi phát hiện ra mode lỗi tại địa chỉ tại địa chỉ 2016 (07E0H) của bộ nhớ 2Kx16

Bảng 4.5 - Kết quả chạy phân tích trên giải thuật TLAPNPSF

VCD+ Writer D-2010.06-SP1 Copyright (c) 1991-2010 by Synopsys Inc.

== APNPSF ARE RUNNING ==
MEMORY IS FAILED AT ADDRESS = : 02016
MEMORY IS FAILED AT ADDRESS = : 02016
MEMORY IS FAILED AT ADDRESS = : 02016
MEMORY IS FAILED AT ADDRESS = : 02016
MEMORY IS FAILED AT ADDRESS = : 02016
MEMORY IS FAILED AT ADDRESS = : 02016
MEMORY IS FAILED AT ADDRESS = : 02016
MEMORY IS FAILED AT ADDRESS = : 02016

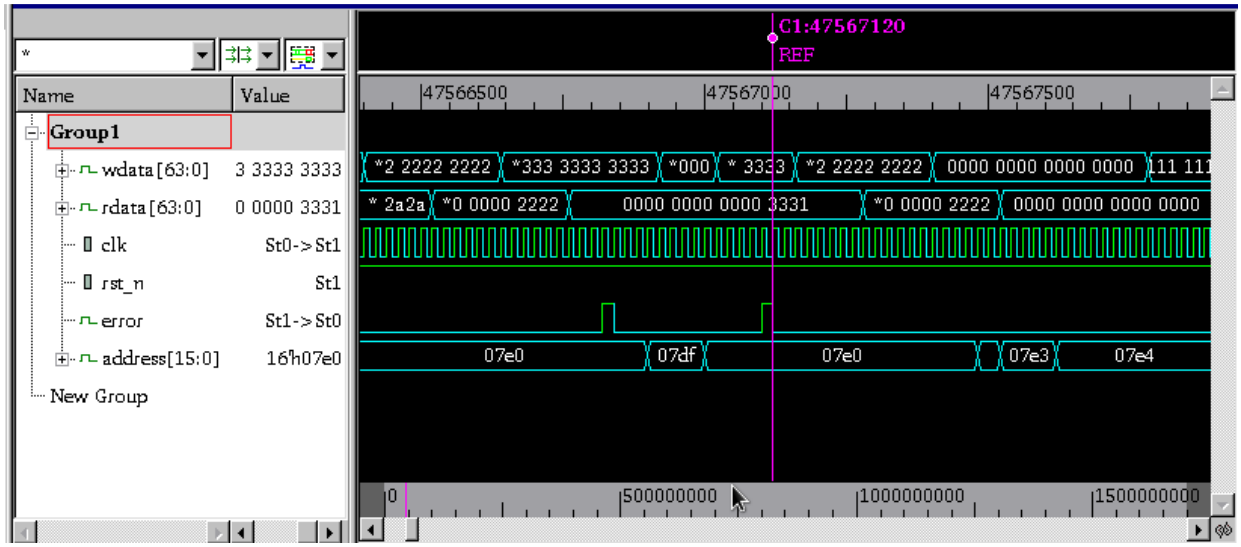
$finish called from file "testbench.v", line 52.
$finish at simulation time 1698701022

VCS Simulation Report

Time: 1698701022
CPU Time: 27.140 seconds; Data structure size: 0.0Mb
Sun Nov 27 16:23:18 2016
CPU time: .890 seconds to compile + .095 seconds to elab + .700 seconds to link +

158.332 seconds in simulation

Kết quả dạng sóng tại vị trí có lỗi xảy ra trên bộ nhớ



Hình 4.3 - Dạng sóng tín hiệu tại vị trí lỗi 7E0H (2016D)

Tính linh hoạt trong việc tạo ra kiến trúc MBIST với các thông số được người sử dụng hay nhà sản xuất nhập vào được thể hiện như theo sơ đồ cây như bảng 4.6

Bảng 4.6 - Sơ đồ cây thư mục của môi trường tự tạo ra kiến trúc MBIST

```
|-- 01_controller
|   |-- controller_skl.v
|   |-- controller_skl_v2.v
|   |-- controller.v
|   `-- gen_controller.pl
|-- 02_decoder
|   |-- gen_mbist_decoder.pl
|   |-- mbist_decoder_skl.v
|   |-- mbist_decoder_skl_v2.v
|   `-- mbist_decoder.v
|-- 03_demux
|   |-- gen_mbist_demux.pl
|   |-- mbist_demux_skl.v
|   |-- mbist_memory_number.txt
|   |-- mbist_mux_demux_original.v
|   |-- mbist_mux_demux.v
|   |-- memtype_skl.v
|   |-- run_memtype_2.pl
|   |-- run_memtype.pl
|   `-- run_memtype.sh
|-- 04_mbist
|   |-- gen_mbist.pl
```

```
|   |-- mbist_skl.v
|   |-- mbist_skl_v2.v
|   |-- mbist_skl_v3.v
|   `-- mbist.v
|-- 05_marchc
|   |-- marchc_controller.v
|   |-- marchc_datapath.v
|   `-- marchc_decoder.v
|-- 06_apnpsf
|   |-- apnpsf_controller.v
|   |-- apnpsf_datapath.v
|   `-- apnpsf_decoder.v
|-- 07_marcha
|   |-- marcha_controller.v
|   |-- marcha_datapath.v
|   `-- marcha_decoder.v
|-- address_sort.log
|-- mbist_algorithm.txt
|-- run_algorithm.pl
|-- run_algorithm.sh
```

Bảng 4.7 - Chức năng của các thư mục bên trong môi trường tự tạo kiến trúcMBIST

STT	Tên	Đặc tả
1	controller	Thư mục chứa file để tạo ra khối controller trong MBIST
2	decoder	Thư mục chứa file để tạo ra khối decoder trong MBIST
3	demux	Thư mục chứa file để tạo ra khối demux trong MBIST và số lượng bộ nhớ sẽ được kiểm tra
4	marchc	Thư mục chứa file liên quan tới thiết kế của giải thuật MarchC
5	apnpsf	Thư mục chứa file liên quan tới thiết kế của giải thuật TLAPNPSF
6	marcha	Thư mục chứa file liên quan tới thiết kế của giải thuật MarchA
7	mbist_algorithm	Nhập các giải thuật sẽ được tích hợp vào trong kiến trúc MBIST
8	run_algorithm.pl	File perl để thực hiện tạo ra các khối kiến trúc bên trong MBIST
9	run_algorithm.sh	File bash cell để thực hiện tạo ra các khối kiến trúc bên trong MBIST

Để tạo ra kiến trúc MBIST theo giải thuật mong muốn, người dùng chỉ cần input những giải thuật nào sẽ được chạy vào file text *run_algorithm.txt*

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

Để có thể tổng hợp từ cấp độ RTL xuống lớp cổng, một số công cụ cần thiết được sử dụng để thực hiện việc phân tích. Sau đây tóm tắt chi tiết các công cụ cũng như nền tảng tổng hợp cho khối kiến trúc.

Bảng 4.8 - Các thông số bên trong môi trường thiết kế

Hệ điều hành	Linux
Ngôn ngữ tạo ra môi trường	Bash Shell, Perl, Tcl
Ngôn ngữ thiết kế	Verilog
Thư viện sử dụng	TSMC 90nm
Công cụ	Design compiler
Thiết kế	MBIST
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

Với mục đích nhằm phân tích rõ đặc tính tối ưu của thiết kế và đặc tính kỹ thuật của hệ thống sẽ như thế nào nếu như được phân tích trong môi trường có tích hợp 3 giải thuật và kiểm tra trên bộ nhớ 64Kx64. Các công việc bên dưới thực hiện việc phân tích từ thiết kế RTL thành các cổng vật lý.

Thư viện chọn được thiết kế là thư viện TSMC 90nm, cấu trúc cơ bản của môi trường tổng hợp ra kiến trúc ở lớp cổng vật lý được thể hiện như sơ đồ cây bảng 4.9. Trong đó file vr.v chứa các mã code Verilog của kiến trúc MBIST.

Bảng 4.9 - Sơ đồ cây thư mục môi trường tổng hợp kiến trúc vật lý

|-- *formality.log*
|-- *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
```

Bảng 4.10 - Đặc tả cây thư mục của môi trường tổng hợp

STT	**Tên**	**Đặc tả**
1	reports	Thư mục chứa các file tường trình chứa các đặc tả của kiến trúc
2	result	Thư mục chứa các file kết quả
3	rm_dc_script	Thư mục chứa chứa các công cụ thực thi viêt bởi ngôn ngữ TCL
4	rm_setup	Thư mục chứa các file thiết lập ràng buộc
5	rtl	Thư mục chứa thiết kế ở cấp độ RTL
6	run_dc	File để thực thi tổng hợp xuống lớp cổng
7	run_fm	File thực thi kiểm tra thiết kế ở RTL và lớp cổng
8	run_fm_gate	File thực thi kiểm tra thiết kế ở lớp cổng trước và sau đi dây
9	wire_load	File chứa các thông số ước lượng thời gian trì hoãn các dây dẫn

Các ràng buộc được thể hiện ở các giá trị sau

Bảng 4.11 - Các ràng buộc trong quá trình tổng hợp kiến trúc MBIST

```
create_clock  -name clk [get_ports clk] -period 10 -waveform {0 5}
set_clock_uncertainty 0.1 clk
set_clock_latency 0.1 clk
set_clock_transition 0.1 clk
set_dont_touch_network clk
#dont touch
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 clk 0.1 [all_inputs]
set_output_delay  -clock clk 0.1 [all_outputs]
```

Các file tường thuật thể hiện các kết quả đọc diện tích của thiết kế từ file tổng hợp.

Bảng 4.12 - Phân tích kết quả diện tích của kiến trúc ở lớp vật lý

```
****************************************
Report : area
Design : vr
Version: C-2009.06
Date   : Sat Nov 19 16:04:26 2016
****************************************

Library(s) Used:

   saed90nm_typ_ht (File:
/home/quangtruong/synopsys/LIB/SAED90_EDK/SAED_EDK90nm_REF/reference
s/ChipTop/models/saed90nm_typ_ht_pg.db)

Number of ports:          1194
Number of nets:           1345
Number of cells:             3
Number of references:        3

Combinational area:      29933.818179
Noncombinational area:   46938.934685
Net Interconnect area:   undefined  (No wire load specified)

Total cell area:         76872.752863
Total area:              undefined
```

File tường thuật về việc phân tích công suất của kiến trúc MBIST ở lớp vật lý

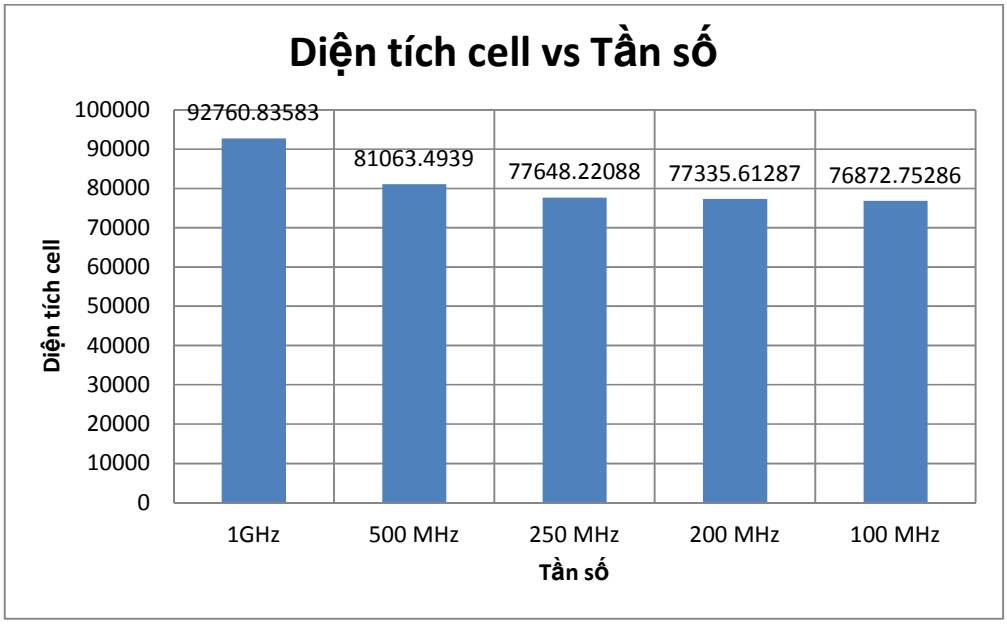
Bảng 4.13 - Phân tích công suất của kiến trúc MBIST ở lớp vật lý

```
****************************************
Report : power
       -analysis_effort low
Design : vr
Version: C-2009.06
Date   : Sat Nov 19 16:04:28 2016
****************************************
Library(s) Used:

   saed90nm_typ_ht (File:
/home/quangtruong/synopsys/LIB/SAED90_EDK/SAED_EDK90nm_REF/reference
s/ChipTop/models/saed90nm_typ_ht_pg.db)

Operating Conditions: TYPICAL   Library: saed90nm_typ_ht
Wire Load Model Mode: top

Global Operating Voltage = 1.2
Power-specific unit information :
   Voltage Units = 1V
   Capacitance Units = 1.000000ff
   Time Units = 1ns
   Dynamic Power Units = 1uW    (derived from V,C,T units)
   Leakage Power Units = 1pW
  Cell Internal Power  =  13.6313 mW  (100%)
  Net Switching Power  =  44.2603 uW    (0%)
                  ---------
Total Dynamic Power    =  13.6756 mW  (100%)
Cell Leakage Power     = 998.9882 uW
```

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

Bảng 4.14 - Mối quan hệ giữa tần số và tổng diện tích các cell nhớ của thiết kế

No	Tần số	Tổng diện tích của các cell
1	1GHz	92760.835829
2	500 MHz	81063.493901
3	250 MHz	77648.220884
4	200 MHz	77335.612871
5	100 MHz	76872.752863



Hình 4.4 - Mối quan hệ giữa tần số và diện tích cell nhớ

Phân tích các kết quả nghiên cứu với kiến trúc MBIST đề xuất với các nghiên cứu khác được thể hiện bảng so sánh 4.15:

+ Với cùng công nghệ 130nm, nhưng thiết kế lại tiết kiệm được thời gian so với các kiến trúc khác bằng cách tần số được tăng lên nhanh hơn 500 Mhz

+ Mặc dù cùng thuật toán MarchC – Nhưng thiết kế lại có diện tích dành cho MBIST thấp hơn so với nghiên cứu [12].

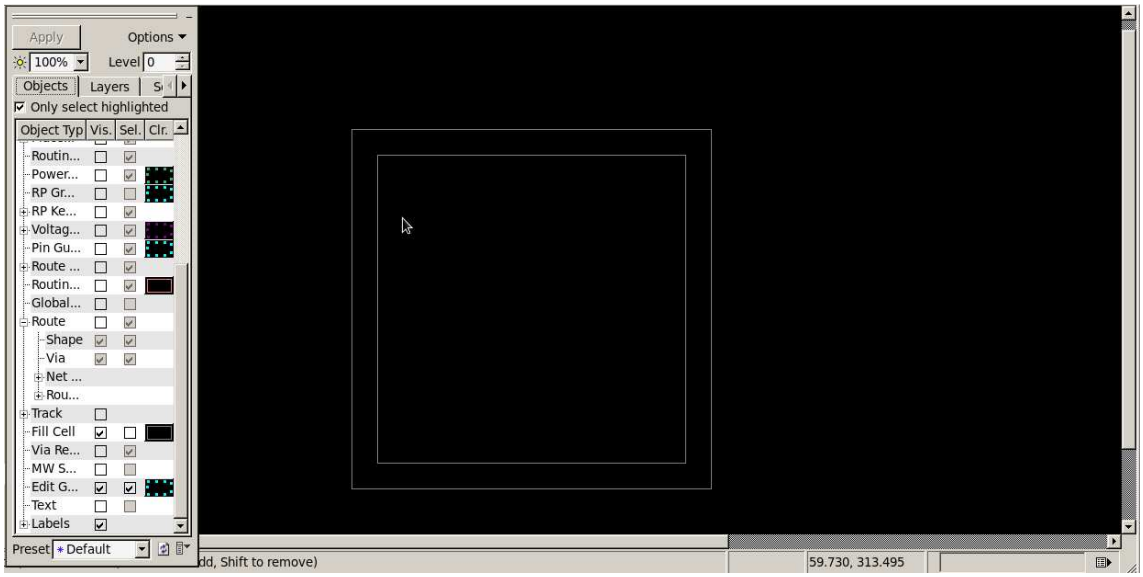
+ Ngoài ra kiến trúc đề xuất đã mở rộng được dung lượng bộ nhớ so với các nghiên cứu khác.

Bảng 4.15 - So sánh kết quả kiến trúc đề xuất

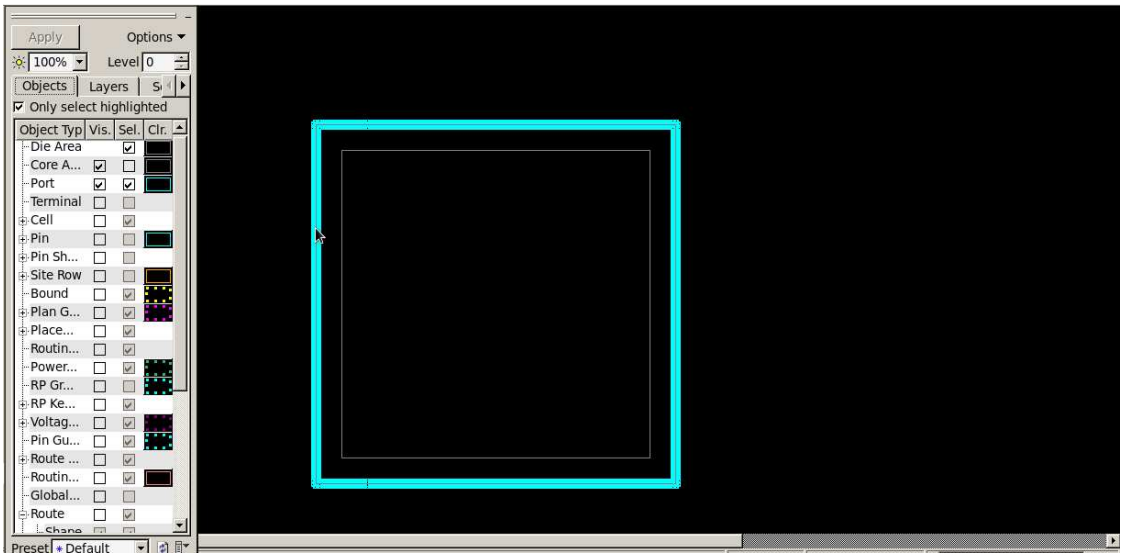
Authors	Technology	Algorithm	Memory capacity	Area	Freqency
[5]	130 nm	March C+, March SS, March Y, Walking, Galloping	16.3Kx16	7942 gates	400 MHz
[7]	130 nm	March C+, March C–, Checkerboard	1Kx8	1582 gates	500 MHz
[10]	130 nm	March C+	16Kx16	6427 gates	300 MHz
[11]	90 nm	March C–, MATS+	8Kx32	3400 gates	500 MHz
[12]	250 nm	March C–	16M x8	27469 gates	344 MHz
Proposed MBIST	**130 nm**	**MARCH C–, TLAPNPSF**	**1Kx8 to 64Kx64**	**11207 gates**	**500 MHz**
Proposed MBIST	**90nm**	**MARCH C–, TLAPNPSF, MARCH A,**	**1Kx8 to 64Kx64**	**12009 gates**	**500MHz**

4.3 Kết quả Place and Route

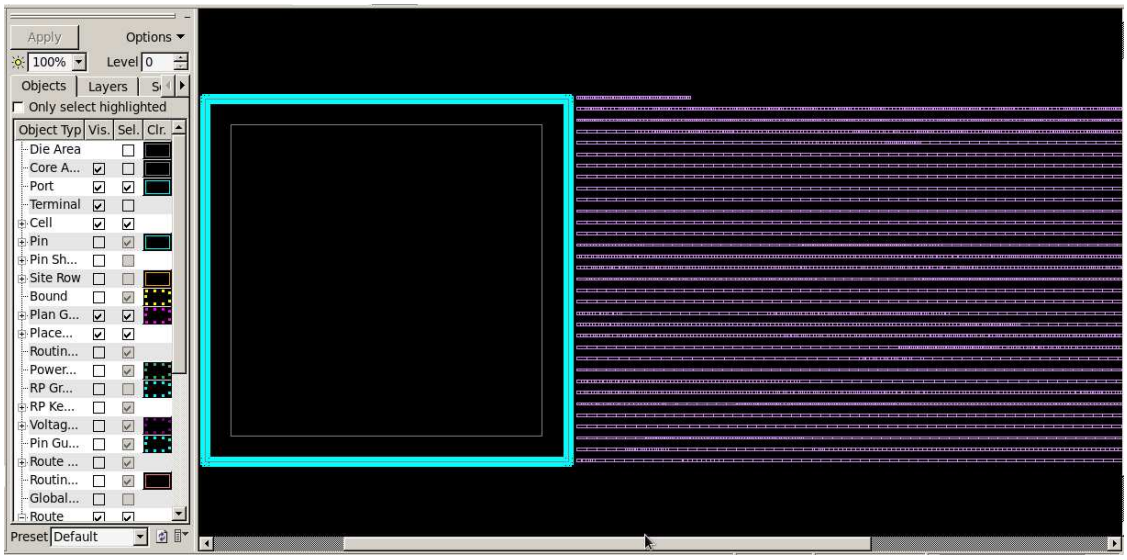
Thiết kế ở mức cổng sau khi được đảm bảo chính xác bởi các kiểm tra cần thiết, toàn bộ thiết kế được thực hiện đặt và đi dây nhằm tạo thiết kế vật lý cuối cùng cho sản xuất. Các kết quả bên dưới thể hiện của việc đặt và đi dây.



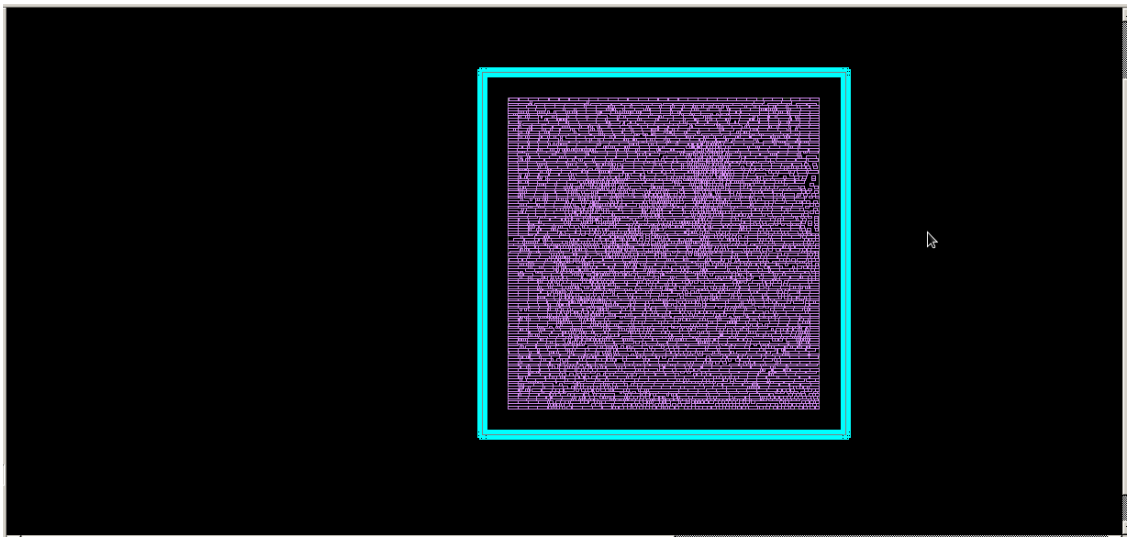
Hình 4.5 Tạo core cho chip



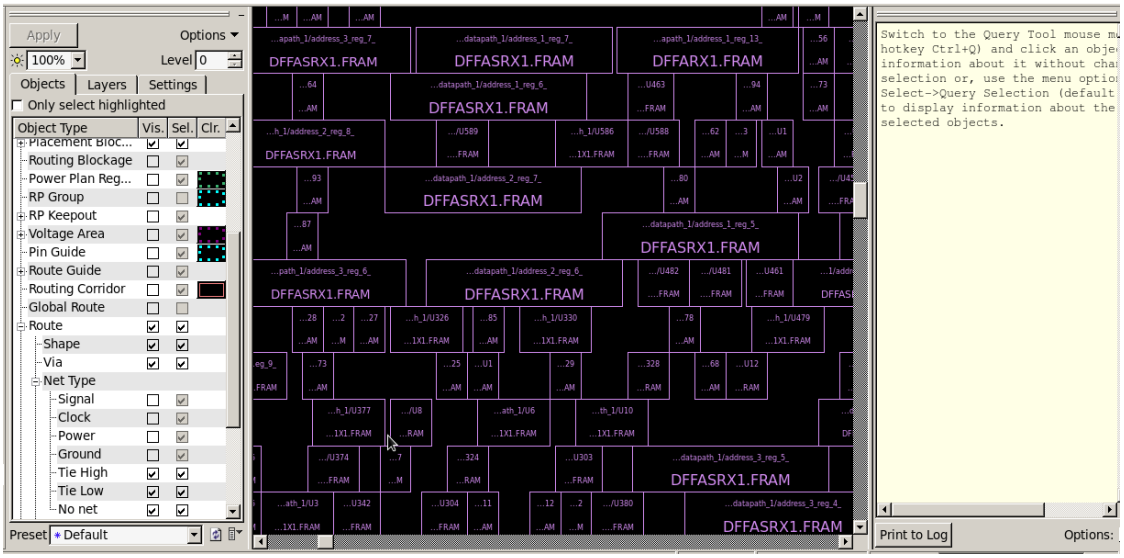
Hình 4.6 – Tạo các chân kết nối (port)



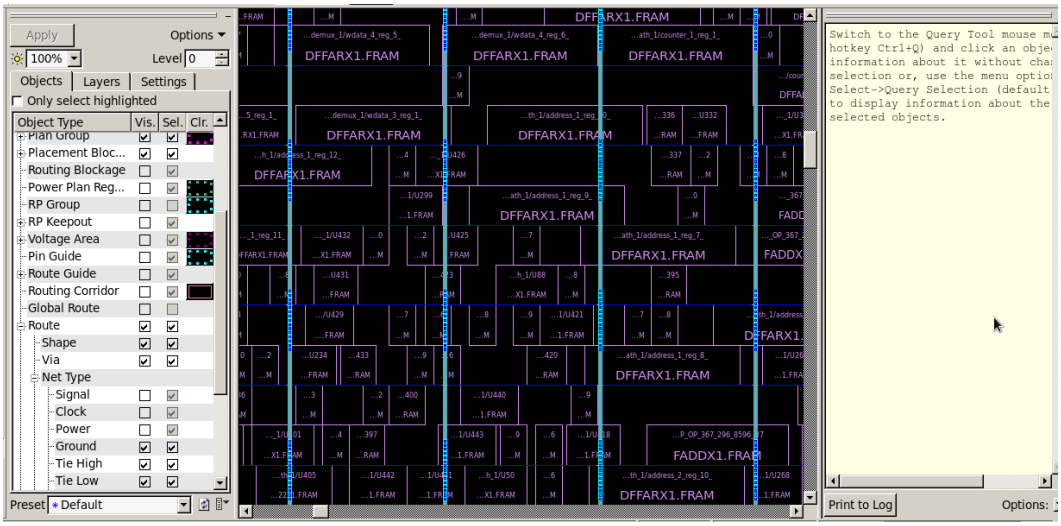
Hình 4.7 – Các cổng trước khi đặt trên core và kết nối với port



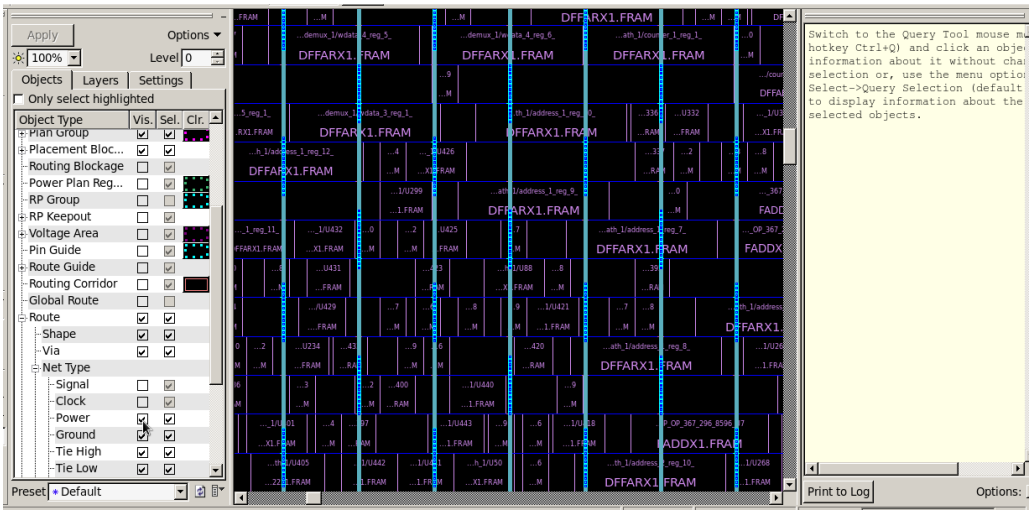
Hình 4.8 - Kết quả sau khi đặt (place) các cổng thiết kế vào core và terminal



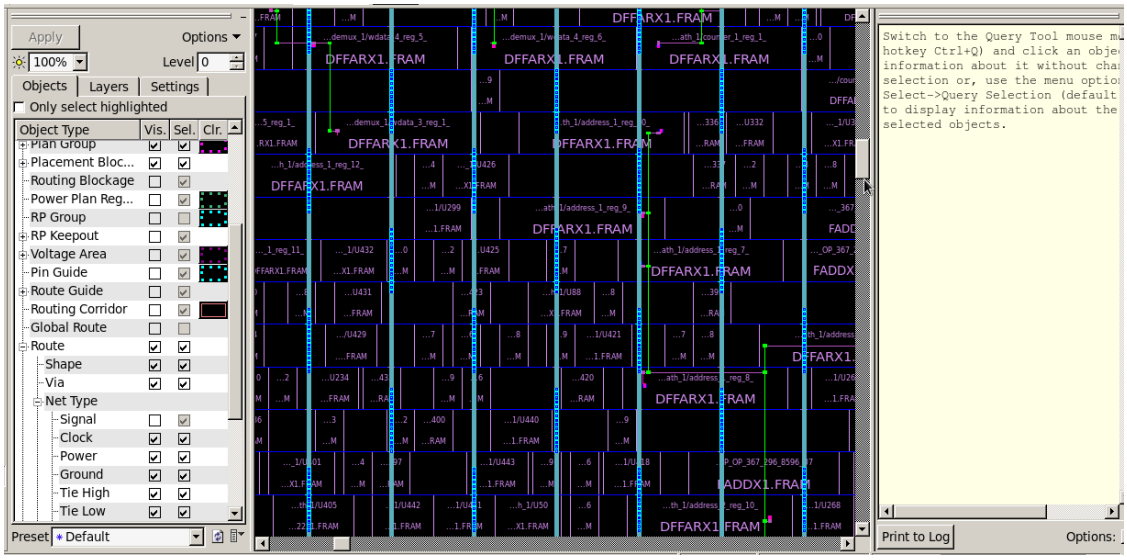
Hình 4.9 - Chi tiết sau khi đặt vào bên trong các terminal



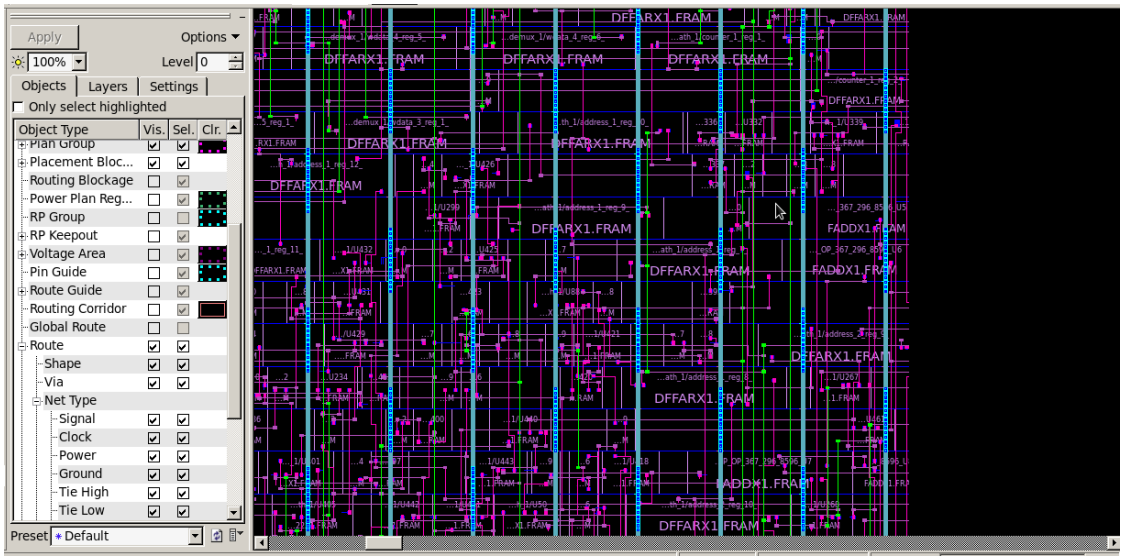
Hình 4.10 - Chi tiết của thiết kế sau khi đi dây ground



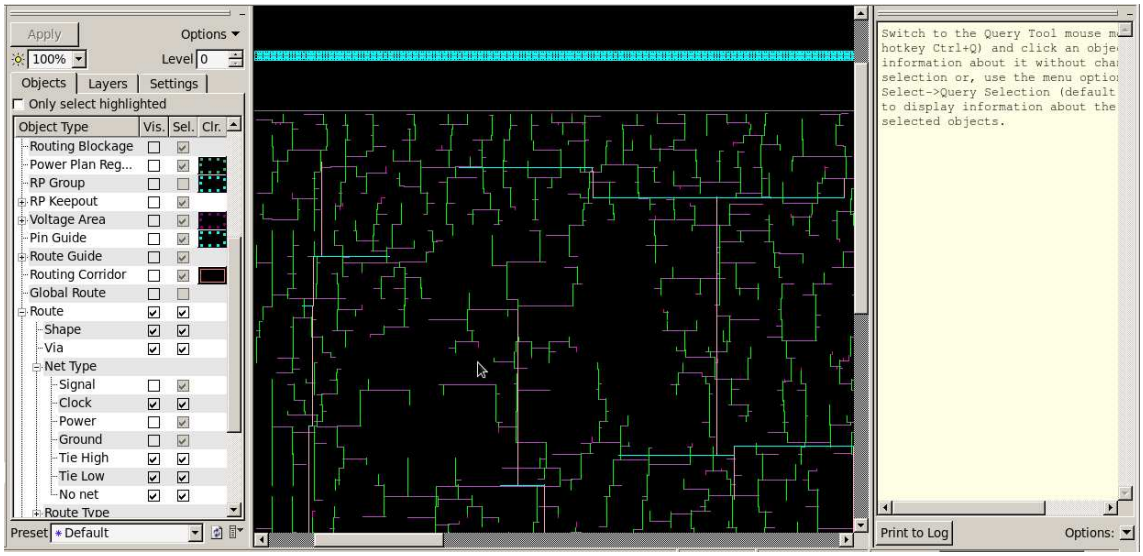
Hình 4.11 - Chi tiết sau khi đặt thêm dây power



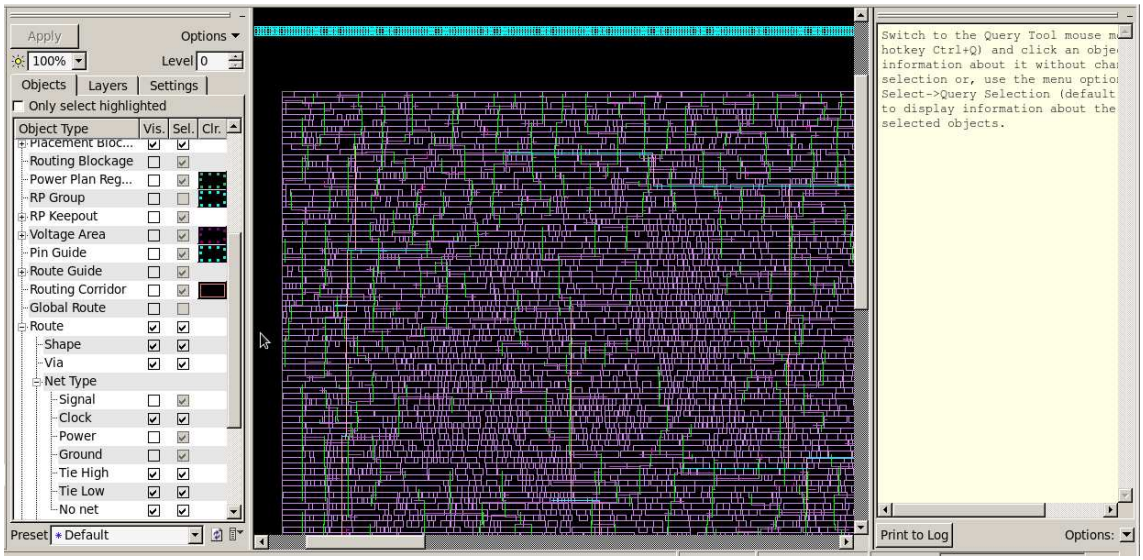
Hình 4.12 - Chi tiết kết quả sau khi đi thêm xung clock



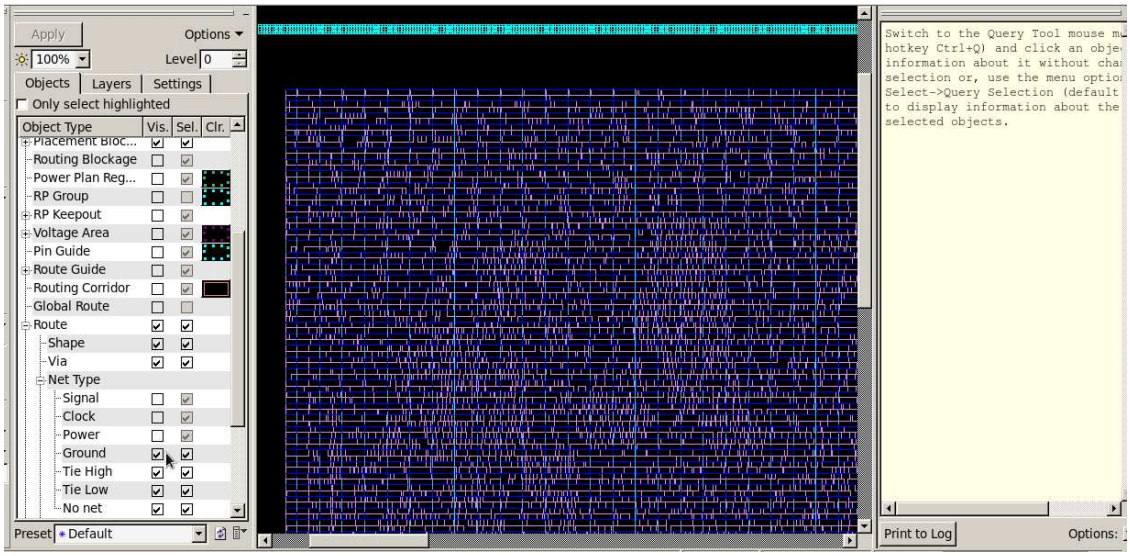
Hình 4.13 - Kết quả sau khi đi thêm các dây tín hiệu



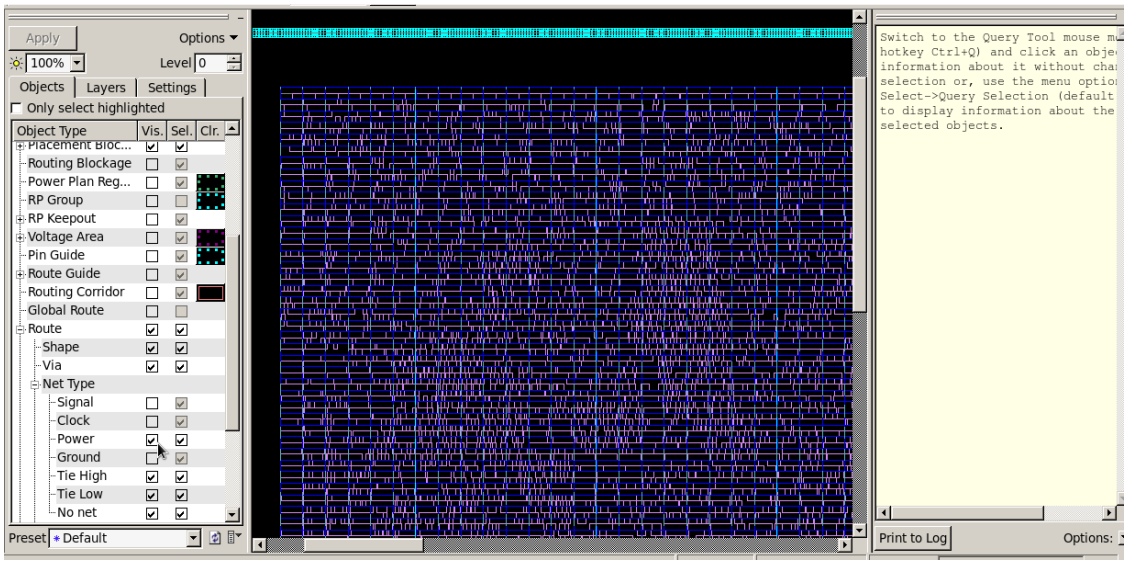
Hình 4.14 - Kết quả sau khi tiến hành chỉ đi dây clock



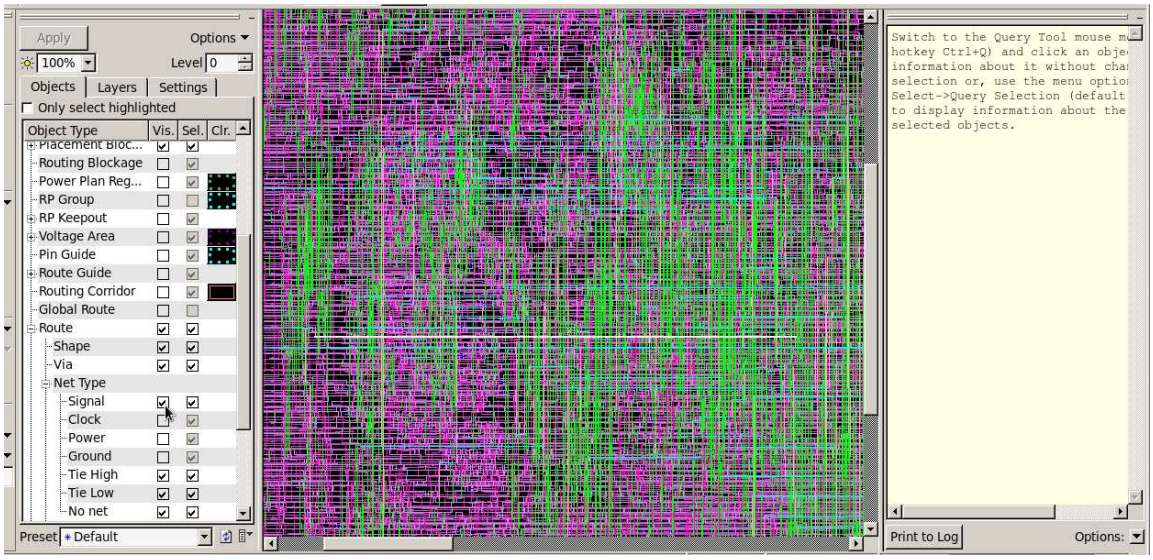
Hình 4.15 - Kết quả sau khi đi dây clock



Hình 4.16 - Kết qua sau khi đi dây ground



Hình 4.17- Kết quả sau khi đi dây power



Hình 4.18 - Kết quả đặt và đi dây toàn bộ chip (chi tiết)

Chương 5: Ý NGHĨA KHOA HỌC VÀ HƯỚNG PHÁT TRIỂN

5.1 Ý nghĩa khoa học

Luận văn đã đóng góp cho lĩnh vực kiểm tra bộ nhớ đặc biệt là kiểm tra chất lượng bộ nhớ SRAM và đồng thời nâng cao được tính linh hoạt của khối kiến trúc MBIST.

Tính đóng góp của luận văn được thể hiện thông qua những điểm như bên dưới:

- Các mô hình lỗi hay phát sinh trên bộ nhớ SRAM đã được liệt kê trong luận văn. Đồng thời tìm hiểu các giải thuật để đi phát hiện các mô hình lỗi đó.
- Kiến trúc MBIST đã được xây dựng một cách hoàn chỉnh với các giải thuật đã tìm hiểu để bao quát hết tất cả các mô hình lỗi hay xảy ra trên SRAM dựa trên các trạn thái máy đã được xác định từ trước. Toàn bộ các giải thuật được xây dựng dựa trên các mô hình trạng thái máy tương ứng.
- Xây dựng môi trường kiểm tra tính hiệu quả của các giải thuật tích hợp lên kiến trúc MBIST. Đây là cơ sở để phát triển thành một môi trường hoàn chỉnh có thể kiểm tra hết tất cả các giải thuật được tích hợp vào kiến trúc ở cấp độ RTL.
- Tính linh hoạt cho kiến trúc kiểm tra SRAM của luận văn được thể hiện qua việc kiểm tra được dung lượng bộ nhớ từ 1K x 8 cho tới 64K x 64. Đồng thời đa dạng hóa số lượng bộ nhớ được kết nối tới MBIST để kiểm tra.
- Với mục tiêu hướng đến tính ứng dụng thực tiễn cao cho đề tài với môi trường công nghiệp, môi trường tự động tạo ra kiến trúc MBIST, ở đó người dùng tự nhập các thông số các giải thuật cần sử dụng, cũng như là số lượng mong muốn các bộ nhớ được kết nối tới MBIST, kiến trúc MBIST với cấp độ RTL sẽ được thiết kế tương ứng.

5.2 Hướng phát triển của đề tài

- Xây dựng thêm nhiều mô hình trạng thái máy cho các giải thuật đang sử dụng để từ đó làm cơ sơ xây dựng kiến trúc MBIST với tính đa dạng các giải thuật ngày càng được phong phú và đáp ứng với yêu cầu kiểm tra của kiến trúc. Đồng thời có thể mở rộng dung lượng bộ nhớ được kiểm tra trên MBIST

- Nâng cấp môi trường kiểm tra tính hiệu quả của kiến trúc MBIST bằng cách tích hợp thêm các mô hình lỗi mới hơn nữa.
- Xây dựng tính linh hoạt cho kiến trúc MBIST ở cấp độ cổng, mà ở đó các nhà sản xuất sẽ tiết kiệm thời gian cho việc sản xuất khi không cần thiết thiết kế lại kiến trúc MBIST ở cấp độ RTL.

CÔNG TRÌNH CÔNG BỐ

1. “An Effective Architecture of Memory Built-In Self-Test for Wide Range of SRAM” in International Conference on Advanced Computing and Applications - ACOMP, Can Tho, 2016.

TÀI LIỆU THAM KHẢO

[1] B. Singh *et al*. "Area overhead and power analysis of March algorithms for memory BIST," in *International Conference on Communication Technology and System Design*, 2012, vol. 30, p. 930-936.

[2] G. Harutyunyan *et al*. "A new method for March test algorithm generation and its application for fault detection in RAMs," in *IEEE Transction of Computer-Aided Design of Integrated Circuits and System*, 2012, vol. 31, pp. 941-949.

[3] Y. J. Huang *et al*. "A low cost built in self test scheme for an array of memories," in *15th IEEE European Test Symposium*, 2010, pp. 75-80.

[4] M. Riedel and J. Rajski. "Fault coverage analysis of RAM," in *13th IEEE VLSI Test Symposium*, 1995pp. 227-234,.

[5] Y. Park *et al*. "A flexible progarmable memory BIST for embedded single port memory and dual port memory," in *Electronic and Telecomunications Research Institute Journal*, vol. 35, pp. 808-818, October 2013.

[6] Q. L. Rao *et al*. "A memory built in self test architecture for memories different in size," in *IEEE Circuit and System International Conference on Testing and Diagnosis*, pp.1-4, 2009.

[7] C. F. Lin and Y. J. Chang. "An area-efficient design for programble memory built in self test," in *IEEE International Symposium on VLSI Design Automatic and Test*, pp. 17-20, 2008.

[8] C. Huzum and P. Cascaval. "All multibackground March test for all static sample Neighborhood Patterns Sensitivity Fault in RAMs," in *15th International Conference*, pp. 01-06, 2010.

[9] Y.-J. Huang and J.-F. Li. "Testing ternary content addressable memories with active neighborhood pattern sensitive faults," in *IET Computers Digital Techniques*, pp. 246-255, 2007.

[10] Y. Park *et al*. "An efficient BIST architecture for embedded RAMs," in *International Conference On Electronic Information and Communication (ICEIC)*, vol. E92D, pp. 2508-2511, 2009.

[11] A. J. Van de Goor *et al*. "Generic, orthogonal and low cost March element based memory BIST," in *International Test Conference*, pp. 1-10, 2011.

[12] Y. Park *et al*. "An efficient BIST architecture for embedded dual port memories," in *IEEE International Solid –State Circuits Conference*, pp. 157 – 160, 2007.

[13] M. Parvathi *et al*. "Modified March C - Algorithm for Embedded Memory Testing," in *International Journal of Electrical and Computer Engineering*, 2012.

[14] D. Youn *et al*. "A Microcode-based Memory BIST Implementing Modified March Algorithm," in Test Symposium, 2001. Proceedings. 10th Asian, 2012

[15] A.J. Van de Goor *et al*. "Optimizing Memory BIST Address Generator Implementations," in *Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*, 2011 6th International Conference, 2011

[16] A.J Van de Goor and S.Hamdioui. "MBIST architecture framework based on orthogonal constructs," in *Design and Test Workshop (IDT)*, 2010 5th International, 2010

[17] N.Q.M.Noor *et al*. "Low Area FSM-Based Memory BIST for Synchronous SRAM," in *5th International Colloquium on Signal Processing & Its Applications (CSPA)*, 2009.

[18] C.F.Lin and Y.J. Chang. "An Area-Efficient Design for Programmable Memory Built-In Self Test," in VLSI Design, Automation and Test, 2008. VLSI-DAT 2008. IEEE International Symposium on, 2008.

[19] Charles E. Stround. *A Designer's Guide to Built - In Self – Test*. Dordrecht: Kluwer Academic Publishers, 2002, pp 15-16.

[20] H.Kukner, "Generic and Orthogonal March Element based Memory BIST Engine", Ms Thesis, Delft University of Technology, Nertherland, 2011.

[21] A.D. Agrawal , Topic "Memory NPSF and Parametric Test.", Auburn University, Auburn, Alabama, American, 2005.