

**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:
Mã số:

Kỹ thuật điện tử
60520203

TÓM TẮT LUẬN VĂN THẠC SĨ

TP. HỒ CHÍ MINH, tháng 12 năm 2016

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 mode lỗi thường hay xảy ra trên SRAM và các giải thuật có khả năng phát hiện ra các mode lỗi tương ứng. Kiến trúc MBIST mới sẽ dựa trên các giải thuật đó với khả năng bao phủ hết tất cả các mode 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 tạo 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 tiễn của đề tài

Tính linh hoạt cao của kiến trúc mở ra hướng đi mới trong việc kiểm tra các mode 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 kiểm 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 trúc 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.

Chương I

TỔNG QUAN

1.1 Đặt vấn đề

1.1.1 Tầm quan trọng của việc kiểm tra bộ nhớ

Theo định luật Moore's, kích thước CMOS trong ngành 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.

Ngoài ra, để 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

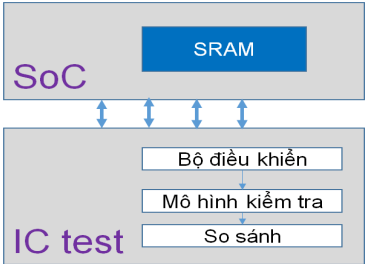
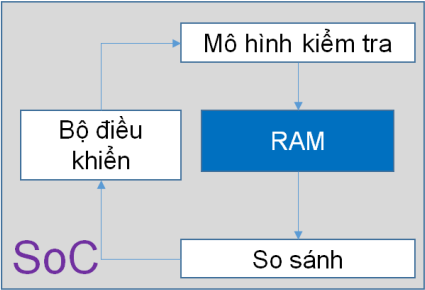
a. Mô hình MBOST

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.

b. Mô hình 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 pháp 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).

Bảng 1.1 So sánh 2 kỹ thuật MBIST và MBOST

Kỹ thuật	MBOST (Memory Built in Self Test)	MBIST (Memory Built in Self Test)
		
Ưu điểm	<ul style="list-style-type: none"> - Tính rủi ro thấp - Thời gian thiết kế cho SRAM nhanh. 	<ul style="list-style-type: none"> - Tính ổn định cao - Tốc độ kiểm tra nhanh, không cần thiết bị bên ngoài - Tiết kiệm chi phí và thời gian cho nghiên cứu, sản xuất
Nhược điểm	<ul style="list-style-type: none"> - Tiêu tốn nhiều thời gian cho việc kiểm tra. - Tính linh hoạt thấp. - Hệ thống kết nối phức tạp. 	<ul style="list-style-type: none"> - Mang tính rủi ro cao trong việc thiết kế và sản xuất - Tốn nhiều diện tích cho hệ thống và thời gian cho việc thiết kế.

1.2 Tình hình nghiên cứu trong nước

Kiến trúc vi mạch MBIST là hướng tiếp cận mới trong việc kiểm tra và đảm bảo chất lượng của bộ nhớ trong quá trình sản xuất, tuy nhiên ở mức độ nghiên cứu cấp độ trong nước thì hầu như chưa có bài báo hay công trình khoa học hoàn chỉnh nào được công bố chính thức. Hoặc nếu có chỉ dừng lại ở mức nghiên cứu các giải thuật hoạt động như thế nào và mô phỏng cách thức hoạt động ở mức độ luận văn tốt nghiệp đại học hoặc các chuyên đề.

1.3 Tình hình nghiên cứu ngoài nước

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.

Theo thống kê về khảo sát các nghiên cứu giải thuật thì đa phần tập trung và các giải thuật March, hay chỉnh sửa các giải thuật để đạt được những hiệu suất về mặt kỹ thuật. [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 1.2 - 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

Theo khảo sát 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 mode 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 thuật toán, kiến trúc MBIST còn 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 kiệm dung lượng xử lý MBIST. [17] thay đổi cấu trúc hoạt động của bộ đếm 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 dung lượng bộ nhớ rất ít được đề cập, do đòi hỏi 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.

Chương II:**ĐỀ XUẤT KIẾN TRÚC MBIST****2.1 Tóm tắt 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
- \uparrow : Ghi 1 vào ô nhớ có nội dung 0 hoặc ô nhớ có sự chuyển trạng thái $0 \rightarrow 1$
- \downarrow : Ghi 0 vào ô nhớ có nội dung 1 hoặc ô nhớ có sự chuyển trạng thái $1 \rightarrow 0$
- \uparrow : Tăng địa chỉ ô nhớ
- \downarrow : Giảm địa chỉ ô nhớ
- \updownarrow : Tăng/Giảm địa chỉ ô nhớ

2.1.1 Thuật toán MarchC-

$(\updownarrow(w0); \up(r0, w1); \up(r1, w0); \downarrow(r0, w1); \downarrow(r1, w0); \up(r0))$

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 mode lỗi liên quan tới CFs, inversionCF, State CF, Dynamic CF. Đây là giải thuật đơn giản nhất để phát hiện những mode 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.

2.1.2 Thuật toán MarchA

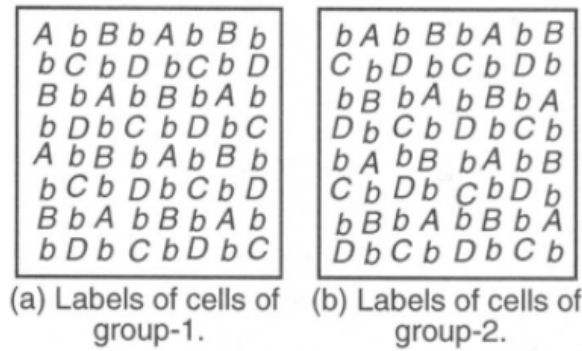
$(\updownarrow(w0); \up(r0, w1, w0, w1); \up(r1, w0, w1); \downarrow(r1, w0, w1, w0); \downarrow(r0, w1, w0))$

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 mode lỗi liên quan tới AFs, SAFs, TFs không có link với CFids, link CFids, March A có tổng tổng 15 các phần tử nhỏ bên trong giải thuật nên nó tốn mất $15.2N$ các giải thuật.

2.1.3 Thuật toán TLAPNPSF

Dựa trên các ô nhớ theo phương pháp tiling, các ô nhớ sẽ được phân chia thành 2 group khác nhau như hình 2.1.



Hình 2.1 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.

Trong kiến trúc được đề xuất phương pháp two groups được sử dụng để đi mô tả cách thức thực hiện của giải thuật TLAPNPSF.

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.

Sử dụng phương pháp two-group

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

Group - 1

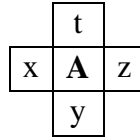
n	A1	n	B1	n	A1	n	B1
C1	n	D1	n	C1	n	D1	n
n	B2	n	A2	n	B2	n	A2
D2	n	C2	n	D2	n	C2	n
n	A1	n	B1	n	A1	n	B1
C1	n	D1	n	C1	n	D1	n
n	B2	n	A2	n	B2	n	A2
D2	n	C2	n	D2	n	C2	n

Group - 2

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

Hình 2.2 Phân ô nhớ làm các nhóm

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.

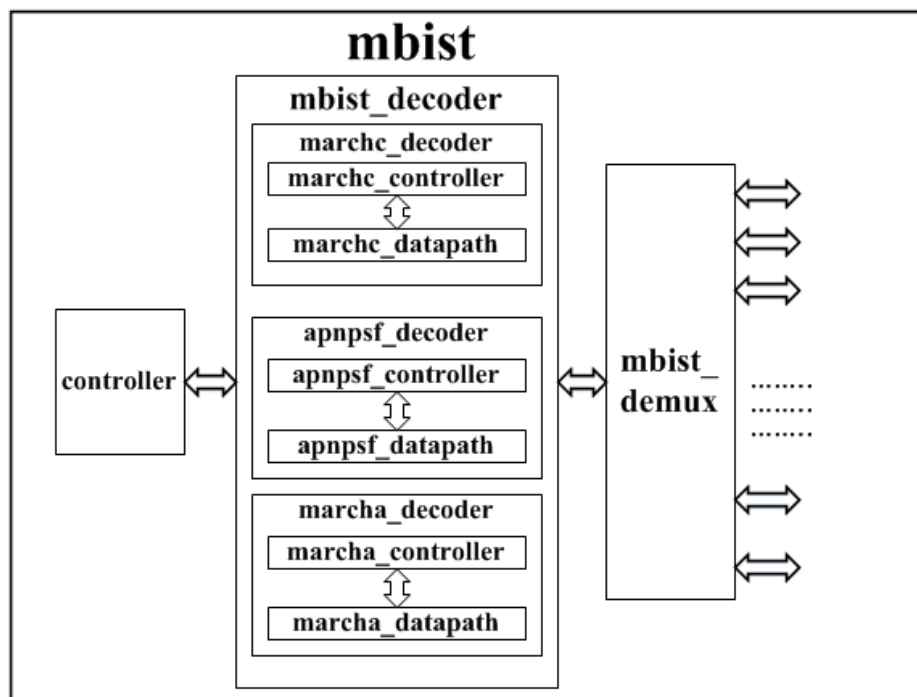


Hình 2.3 Các ô nhớ kế cận của ô nhớ nền

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.

2.2 Kiến trúc tổng quan với các giải thuật đề cập

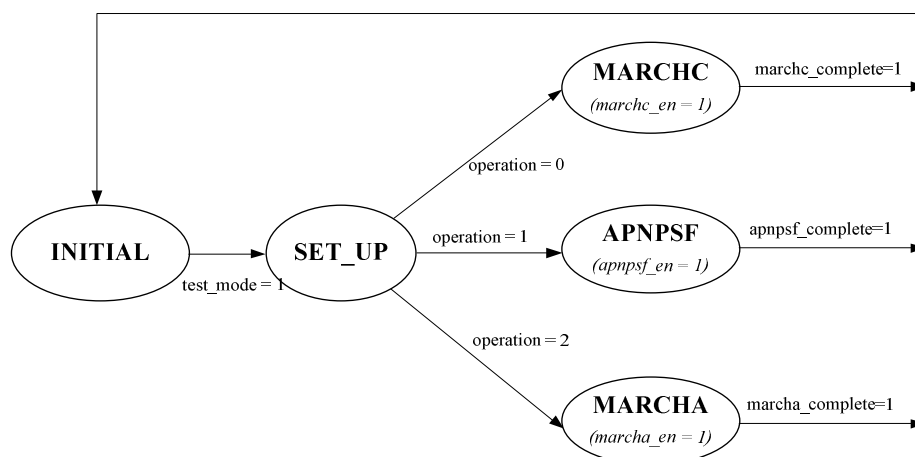
- Sơ đồ các khối giải thuật bên trong kiến trúc MBIST với 3 giải thuật MarchC-, MarchA, TLAPNPSF được sử dụng.



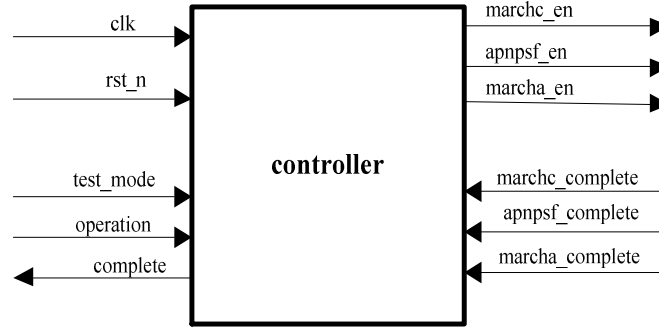
Hình 2.4 Sơ đồ khối tổng quan kiến trúc khi sử dụng cả 3 giải thuật

Khối controller

Khối kiến trúc controller có nhiệm vụ chọn lựa giải thuật nào được sử dụng theo sơ đồ trạng thái máy như bên dưới.



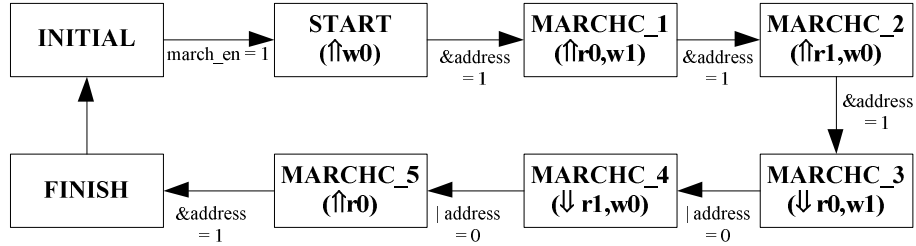
Hình 2.5 Sơ đồ trạng thái máy của khối controller



Hình 2.6 Sơ đồ in-out khối controller

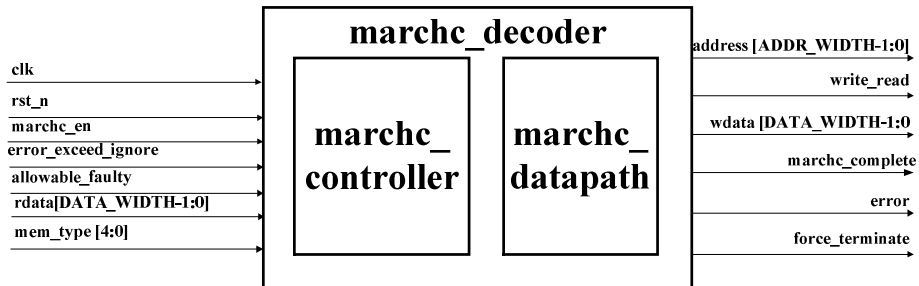
Khối marchc_decoder

Thực hiện việc điều khiển và kiểm soát việc sửa lỗi trên bộ nhớ bằng thuật toán MarchC theo mô hình trạng thái máy.



Hình 2.7 Sơ đồ trạng thái máy thuật toán MarchC-

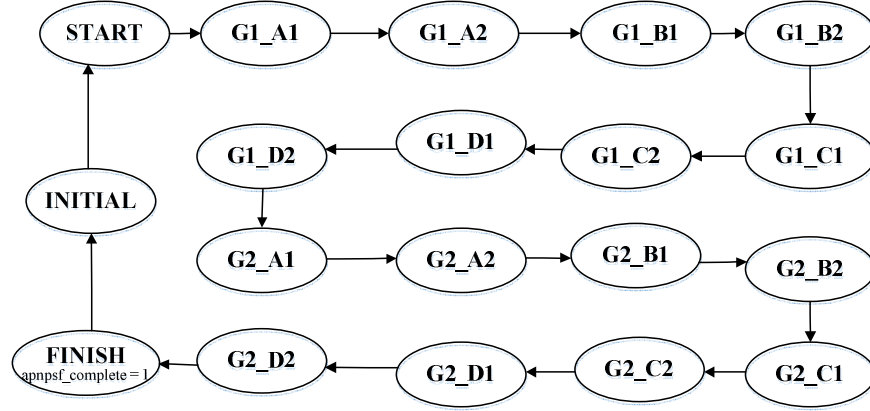
Sơ đồ in out



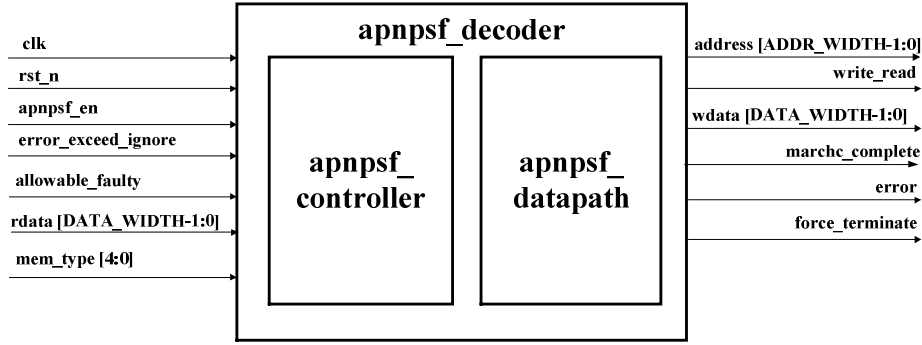
Hình 2.8 Sơ đồ in-out khối marchc_decoder

Khối apnpsf_decoder

Thực hiện việc điều khiển và kiểm soát việc sửa lỗi trên bộ nhớ bằng thuật toán TLAPNPSF theo mô hình trạng thái máy.



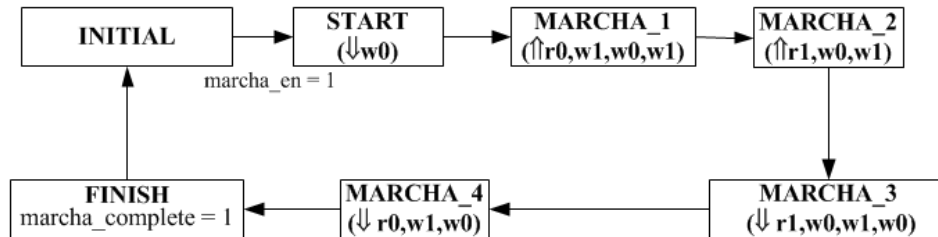
Hình 2.9 Sơ đồ trạng thái máy thuật toán TLAPNPSF



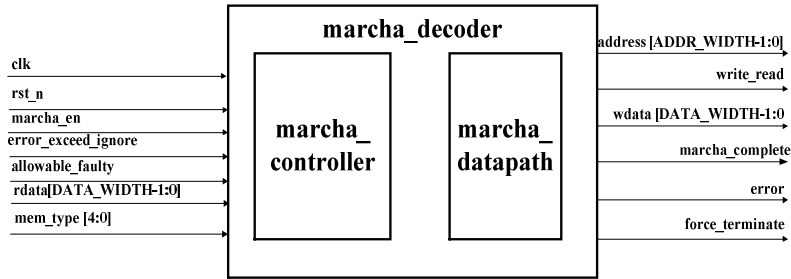
Hình 2.10 Sơ đồ inout khối apnpsf_decoder

Khối marchc_decoder

Thực hiện việc điều khiển và kiểm soát việc sửa lỗi trên bộ nhớ bằng thuật toán MarchA theo mô hình trạng thái máy.



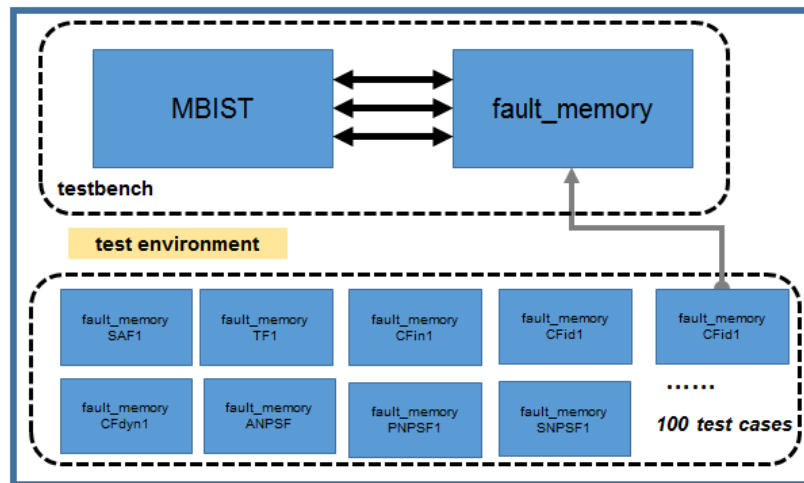
Hình 2.11 Sơ đồ trạng thái máy thuật toán MarchC-



Hình 2.12 Sơ đồ trạng thái máy thuật toán MarchC-

2.3 Môi trường kiểm tra

Theo quy trình thiết kế ASIC thì ở cấp độ RTL cần kiểm tra tính đúng đắn của các giải thuật bên trong kiến trúc MBIST, môi trường giả lập được xây dựng chứa bộ nhớ bị lỗi và kiến trúc MBIST. 100 test case tương ứng với 100 trường hợp lỗi trên bộ nhớ giả lập được đưa vào bên trong môi trường kiểm tra để nâng cao độ tin cậy cho kiến trúc khi được áp dụng vào thực tế. Đồng thời ngôn ngữ Perl và Bash cell đã được sử dụng xây dựng nên môi trường.

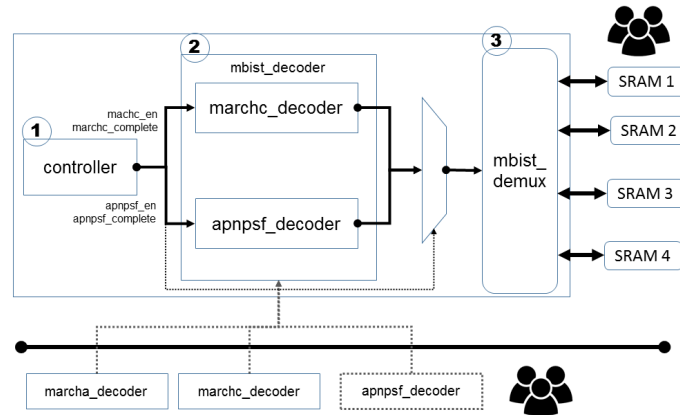


Hình 2.13 Môi trường kiểm chứng kiến trúc MBIST

2.4 Môi trường tự tạo kiến trúc 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 thông qua dung lượng bộ nhớ, số lượng bộ nhớ được kiểm tra (đã thực hiện bên trong giải thuật), và môi trường tự tạo kiến trúc MBIST.

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 bên dưới**..



Hình 2.14 Môi trường tự động tạo ra kiến trúc MBIST

Chương 3:**KẾT QUẢ ĐẠT ĐƯỢC****3.1 Kết quả chạy trên môi trường test**

- 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 3.1 - 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.

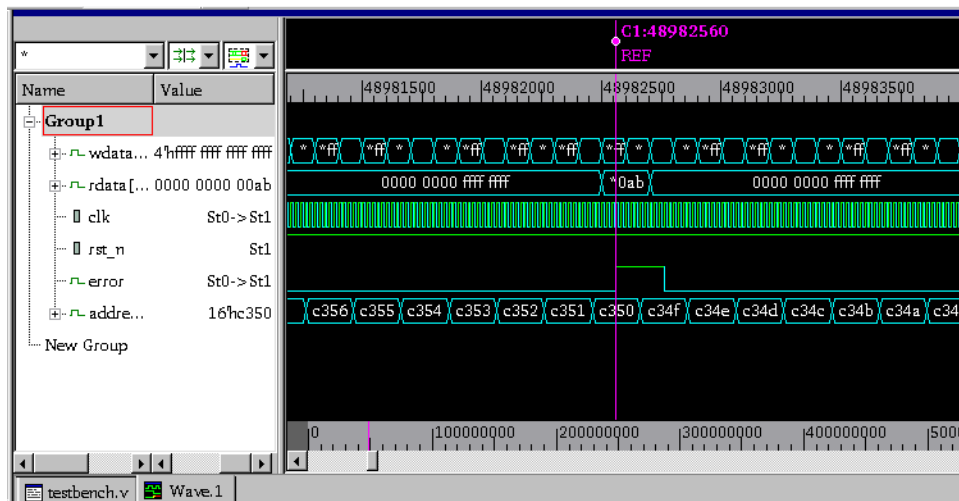
$finish at simulation time      700001022

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 3.1 - 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 3.2 - Kết quả chạy phân tích trên giải thuật MarchA

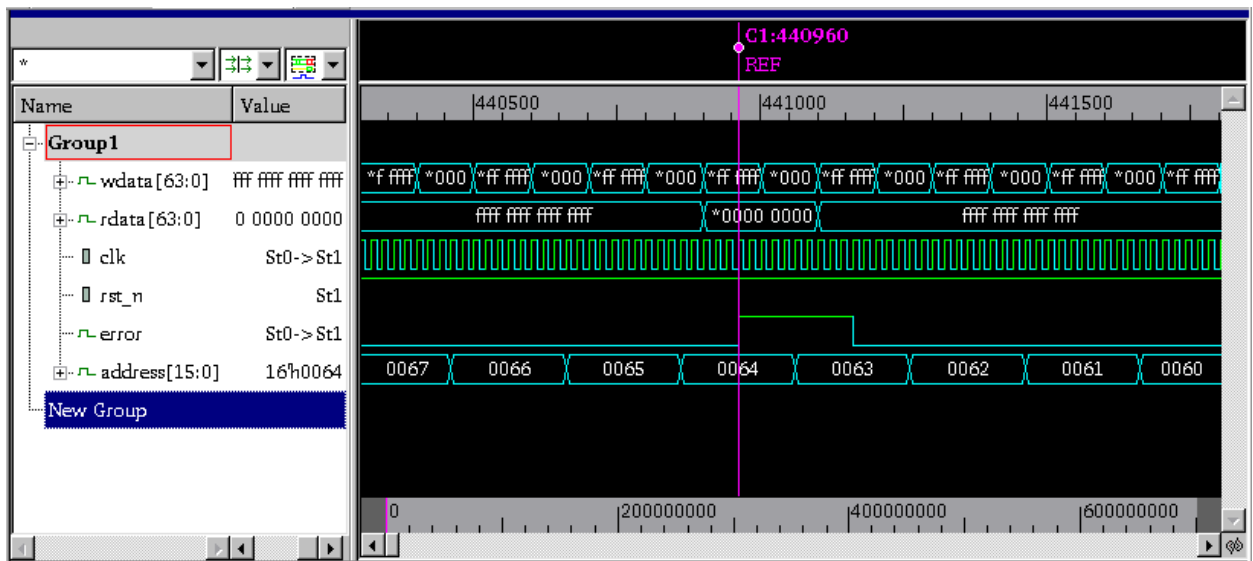
```
VCD+ Writer D-2010.06-SP1 Copyright (c) 1991-2010 by Synopsys Inc.

== 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 3.2 - Dạng sóng tín hiệu tại vị trí lỗi 64H (100D)

Bảng 3.4 - Môi trường tạo ra kiến trúc MBIST

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	<ul style="list-style-type: none"> + Trì hoãn ngõ vào + Trì hoãn ngõ ra + Tần số xung clock + Sử dụng bộ nhớ nội + Reset bất đồng bộ + Trì hoãn clock

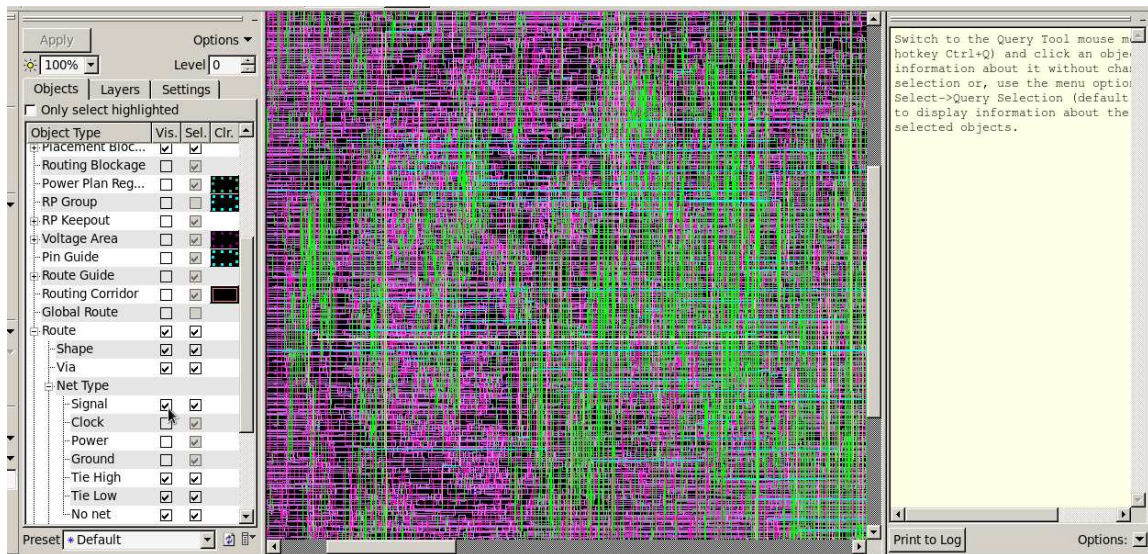
Phân tích các kết quả nghiên cứu với kiến trúc MBIST đề xuất đã được thực hiện theo bảng so sánh 3.5 :

- + 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 3.5 - So sánh kết quả tổng hợp lớp công của thiết kế với các nghiên cứu khác

Authors	Technology	Algorithm	Memory capacity	Area	Frequency
[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

3.3 Kết quả bước place and route



Hình 3.4 - Kết quả đặt và đi dây toàn bộ chip (chi tiết)

Chương 4: Ý NGHĨA KHOA HỌC VÀ HƯỚNG PHÁT TRIỂN

4.1 Ý nghĩa khoa học

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

- Tìm hiểu các mode lỗi và nguyên nhân phát sinh trên bộ nhớ SRAM Đồng thời tìm hiểu các 3 giải thuật để đi phát hiện các mô hình lỗi hay phát sinh trên SRAM.
- 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 trạng 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.
- Một môi trường kiểm tra tính hiệu quả của các giải thuật tích hợp lên chip được đưa ra, ở đó các kiến trúc MBIST có thể test được các mode lỗi thường hay gặp bên trong hệ thống.
- 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.
- Môi trường tự động tạo ra kiến trúc MBIST bằng ngôn ngữ PERL, Bash cell đã được đưa ra. Với môi trường 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 kiến trúc MBIST với cấp độ RTL sẽ được thiết kế tương ứng.

4.2 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 sở 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 mode 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.