ROBERTO CANGELLAR COSSI JUNIOR

Engenheiro Eletricista, Escola Politécnica da Universidade de São Paulo, 1988

APLICAÇÕES E EXTENSÃO DA TÉCNICA DE *BOUNDARY-SCAN* AO TESTE DE MÓDULOS *MULTICHIP*

Dissertação apresentada à
ESCOLA POLITÉCNICA DA
UNIVERSIDADE DE SÃO PAULO
para obtenção do título de
"Mestre em Engenharia Elétrica"

Orientador:

Dr. José Roberto de Almeida Amazonas

São Paulo - 1996

UNIVERSIDADE DE SÃO PAULO
ESCOLA POLITÉCNICA

TERMO DE JULGAMENTO
DE
DEFESA DE DISSERTAÇÃO DE MESTRADO

Aos 03 dias do mês de junho de 1996, às 10:00 horas, no Departamento de Engenharia Eletrônica da Escola Politécnica da Universidade de São Paulo, presente a Comissão Julgadora, integrada pelos Senhores Professores Doutores José Roberto de Almeida Amazonas, orientador do candidato, Marius Strum e Mário Lúcio Cortês iniciou-se a Defesa de Dissertação de Mestrado do Senhor **ROBERTO CANGELLAR COSSI JÚNIOR.**

Título da Dissertação: "APLICAÇÕES E EXTENSÃO DA TÉCNICA DE BOUNDARY-SCAN AO TESTE DE MÓDULOS MULTICHIP".

Concluída a argüição, procedeu-se ao julgamento na forma regulamentar, tendo a Comissão Julgadora Atribuído ao candidato as seguintes notas:

Prof. Dr. José Roberto de Almeida Amazonas.....(10,0)..(dez)
Prof. Dr. Marius Strum.....(10,0)..(dez)
Prof. Dr. Mário Lúcio Cortês.....(9,0)..(nove)

Para constar, é lavrado o presente termo, que vai assinado pela Comissão Julgadora e pela Secretária de Pós-Graduação.

São Paulo, 03 de junho de 1996.

Presidente: _____

_____ _____

Secretária Mara Fátima de Jesus Luz Sanches _____

Observações: _____

_____

Homologado pela C.P.G. em reunião realizada à 17/06/1996.

PEE

Agradecimentos

Ao Dr. José Roberto Amazonas, pela orientação e inúmeras sugestões que enriqueceram este trabalho.

A todas as pessoas da Quickchip e da Itaucom que apoiaram a realização deste trabalho.

À Quickchip, pelo fornecimento do ambiente de projeto onde foi realizada a parte prática.

Ao colega Tadashi Edward Iko, pela colaboração durante a execução dos *layouts* das células desenvolvidas.

Resumo

É apresentado o conceito de módulo *multichip* e suas características. As implicações dessas características em termos de complexidade para a realização de testes dos módulos são analisadas.

É feita uma introdução ao padrão IEEE 1149.1 para uma arquitetura de teste através de *boundary-scan*. A partir deste padrão é desenvolvida uma aplicação voltada para o teste de módulos *multichip*. Esta aplicação envolve a implementação das instruções obrigatórias e de duas opcionais, incluindo a de auto-teste. Além disso são propostas duas novas instruções voltadas para aplicação em módulos *multichip*.

É descrita a teoria utilizada para a implementação do auto-teste e como ela pode ser aplicada.

Os circuitos projetados para a execução dos testes são descritos. Também é introduzido um circuito exemplo, utilizado para verificação dos resultados.

Os resultados finais são apresentados em termos de funcionalidade e tamanho dos circuitos projetados expresso em número de *gates* equivalentes e área.

Abstract

The concept of multichip modules and its characteristics are presented. The consequences of such characteristics in terms of the modules testing complexity are analyzed.

The standard IEEE 1149.1 for test using a boundary-scan architecture is introduced. An application for multichip modules testing is developed based on this standard. This application includes the implementation of the mandatory instructions and two optional, including self-test. Besides those, two additional instructions aiming multichip modules are proposed.

The theory used for the self-test implementation and how it can be applied is descripted.

The circuits designed for the tests execution are depicted. A sample circuit used for results verification is introduced.

The final results are presented in terms of functionality and the designed circuits size, expressed in equivalent gates as well as area.

Índice

Índice de Figuras

Índice de Tabelas

Capítulo 1

Introdução

1. OBJETIVOS

Este trabalho tem como objetivo analisar as características específicas dos módulos *multichip* e como essas características podem influenciar o seu teste. A partir dessa análise é desenvolvida uma aplicação do padrão de teste IEEE 1149.1 [IEEE90] e é proposta um extensão deste padrão voltada para o teste destes módulos. Com isso espera-se facilitar os procedimentos de criação e execução do teste, bem como permitir que seja mais abrangente do que seria caso as técnicas descritas neste trabalho não sejam adotadas.

Organização do Trabalho

Este trabalho consta de 5 capítulos e um anexo, com os conteúdos descritos a seguir.

Capítulo 1:
Apresentação do trabalho e dos módulos *multichip*.

Capítulo 2:
Descrição da tecnologia empregada nos módulos *multichip*. Vantagens e desvantagens de seu uso. Como é feito o projeto e os problemas enfrentados para a realização do seu teste.

Capítulo 3:
Abordagens de teste e sua aplicação aos módulos *multichip*. Apresentação do padrão IEEE 1149.1 para *boundary-scan* e sugestão de novas instruções voltadas para o teste dos módulos *multichip*.

Capítulo 4:
Descrição dos circuitos projetados segundo a linha descrita no capítulo 3.

Capítulo 5:
Apresentação de resultados e sugestões para trabalhos futuros.

Anexo:
Descrições funcionais e esquemáticos dos circuitos projetados e do circuito utilizado como exemplo de aplicação.

2. HISTÓRICO

Com o constante aumento da escala de integração dos circuitos eletrônicos, o encapsulamento destes tem representado um desafio crescente para comportar adequadamente os avanços verificados em termos de desempenho e redução de tamanho dos circuitos. As conseqüências desses avanços vieram igualmente representar problemas adicionais, como um maior número de pinos e maior dissipação térmica. Para que não representasse um entrave ao progresso da tecnologia eletrônica, a tecnologia de encapsulamento buscou soluções para os novos desafios que surgiram. Em meio a esse processo é que surge a tecnologia de módulos *multichip*, utilizada comercialmente pela primeira vez pela IBM há mais de 10 anos atrás, e que com o tempo se firmou como a melhor e por vezes a única solução para o encapsulamento de sistemas de alta integração e desempenho.

A necessidade da utilização de módulos *multichip* surge quando é impossível extrair toda a funcionalidade desejada a partir de um único *chip* ou mesmo de integração a nível de *wafer*, tanto pelo baixo aproveitamento de processo como por limitações de desempenho. Os módulos *multichip* permitem desta forma obter uma escala de integração que só seria possível vários anos mais tarde, com o progresso tecnológico.

Inicialmente esta tecnologia foi aplicada em computadores de grande porte, onde sua utilização era fundamental e economicamente viável. Atualmente este é um dos segmentos de maior crescimento na área de encapsulamento, devendo seu uso se expandir largamente para estações de trabalho, computadores pessoais e finalmente para a eletrônica de consumo em geral.

Espera-se no futuro o uso cada vez maior de módulos integrando *chips* padrões para o desempenho de várias funções, como por exemplo, memórias. Isso representará mais um salto em termos de redução de custos pelo aumento de escala e diminuição significativa dos tempos de projeto.

Muito do desenvolvimento futuro nas áreas de informática e telecomunicações será devido à esta forma de encapsulamento, principalmente quando os tempos de ciclo caírem abaixo de 5ns, como começa a ser comum em arquiteturas RISC, onde atrasos de

1,5ns provocados pelas interconexões em uma placa de circuito impresso, somados aos atrasos devidos a vários encapsulamentos separados se tornam proibitivamente altos.

3. DEFINIÇÃO DE MÓDULOS *MULTICHIP*

Os módulos *multichip* - usualmente referidos como MCM, de "*Multichip Modules*" - são dois ou mais *chips*, não encapsulados ou pré-encapsulados, montados em um módulo responsável pela interconexão, alimentação, proteção e resfriamento destes. É interessante notar que em muitas das aplicações são utilizados *chips* já encapsulados, mais fáceis de manusear e de montar e em disponibilidade bem maior no mercado, embora não apresentem tantas vantagens quantos os *chips* não encapsulados em termos de tamanho, número de entradas e saídas, facilidade de retrabalho e menores resistência, indutância e capacitância parasitárias. Ainda assim, o desempenho e a densidade de interconexões possíveis através desta tecnologia é consideravelmente maior do que o que poderia ser obtido da utilização de placas de circuito impresso tradicionais para a realização do sistema.

4. APLICAÇÕES

O maior campo de aplicação para os módulos *multichip* são sem dúvida os computadores, sendo que este segmento representa um consumo estimado em 1995 de 80% do total produzido. Sendo este um mercado bastante grande e ainda em forte expansão, a difusão do uso de MCMs neste segmento representa um considerável mercado para esta tecnologia. Apenas o mercado de supercomputadores e *mainframes*, onde os MCMs são utilizados por todos os grandes fabricantes, atingiu vendas na faixa de US$ 30 bilhões em 1994. Sendo, porém este um mercado de pouco crescimento, o aumento do uso desta tecnologia ficará por conta da sua incorporação crescente em estações de trabalho e microcomputadores de alto desempenho, um mercado que em 1995 deverá ter alcançado a marca de US$ 100 bilhões. Entretanto, por ser um mercado altamente competitivo, nem todas as tecnologias utilizadas na produção de módulos *multichip* são viáveis atualmente para este tipo de aplicação. O melhor desempenho a nível de sistema, a redução do espaço utilizado e a maior confiabilidade e o menor consumo [Fran93] obtidos em comparação com outras tecnologias são os principais fatores de introdução desta solução neste campo, já que há uma perfeita relação entre as vantagens que podem ser obtidas e os desafios encontrados.

De uma forma geral, considera-se que sistemas que operem em uma freqüência maior do que 50MHz podem se beneficiar bastante com esta tecnologia. Sistemas assim deverão representar 18% do total de sistemas digitais CMOS em meados desta década.

Tabela 1.1: Vantagens do uso de módulo *multichip* para algumas aplicações e sua participação dentro do mercado total

Aplicação	**Vantagens buscadas**	**Participação projetada**
Telecomunicações:	Desempenho em altas freqüências	8%
Área militar:	Menor espaço e peso	7%
Eletrônica de consumo:	Menor espaço e peso	2%
Automotivo:	Menor espaço	2%
Outros:		1%

Considerando todos esses mercados, estima-se que o mercado total para os módulos *multichip* chegue a US$ 10 bilhões até o ano 2000, havendo mesmo previsões chegando ao valor de US$ 20 bilhões [Bald92].

Capítulo 2

Tecnologia dos Módulos *Multichip*

1. TECNOLOGIAS UTILIZADAS

Os módulos *multichip* tiveram sua base e inspiração nos circuitos híbridos, que, admitindo uma definição mais abrangente de MCMs, poderiam mesmo ser considerados como tais. A semelhança entre um e outro está na montagem e no encapsulamento. Porém os MCMs agregam a isto muito da tecnologia dos circuitos integrados, pois para atingir seus objetivos de alta densidade é necessária a utilização de técnicas que permitam conexões de dimensões micrométricas. Isso posto, os MCMs se assemelham bastante também a uma placa de circuito impresso no que diz respeito à sua organização como diversos circuitos integrados interconectados através de diversas camadas de trilhas de material condutivo, com a significativa diferença que a densidade de ligações que pode ser obtida em duas camadas de um MCM exigiria até 14 camadas em um circuito impresso no estado da arte [Garr92].

Tipos Básicos

Existem atualmente 3 tipos básicos de tecnologia para a fabricação de MCMs:

MCM-C: Formado por filme espesso cerâmico impresso e sinterizado.

MCM-L: Formado com tecnologia de placas de circuito impresso laminado de alta densidade.

MCM-D: Formado com filme fino depositado, sendo esse filme composto de polímeros de metais.

Destes tipos básicos derivam variações e combinações entre essas tecnologias, como por exemplo WSI ("*Wafer Scale Integration*") híbrido [Habi93], baseada em MCM-D e as tecnologias MCM-CD e MCM-DC, combinando filme espesso cerâmico e filme fino depositado, somando vantagens das duas tecnologias [Tumm92]. A tabela 2.1 mostra as principais características desses três tipos de tecnologia.

Tabela 2.1: Características dos três tipos de módulos *multichip* [Tumm92]

Característica	MCM-C	MCM-L	MCM-D
Máxima densidade de interconexão (cm/cm^2)	800	300	250-750
Mínima largura de linha (μm)	75-100	60-100	8-25
Espaçamento entre linhas (μm)	50-450	625-2250	25-75
Tamanho máximo do substrato (mm)	245	700	50-225
Constante Dielétrica	5-5,9	3,7-4,5	3,5
Disposição dos pinos de saída - distância em mm	Matriz 1 a 2,54	Matriz 2,54	Periféricos 0,63
Número máximo de camadas de interconexão	63	46	8
Espaço entre vias (μm)	125-450	1250	25-75
Diâmetro das vias (μm)	50-100	150-500	8-25

A utilização de cada um dos tipos deve ser decidida considerando fatores como densidade de interconexão e volume de produção, além das características do projeto em questão.

Conexão ao Substrato

Para os nossos objetivos neste trabalho é ainda mais importante a forma pela qual os circuitos integrados são conectados ao substrato.

Isso determinará as condições em que podem ser feitos testes e as possibilidades de reparos do módulo. As possibilidades de conexão ao substrato são:

Flip-Chip: O circuito integrado é posicionado com a face voltada para o substrato. À cada abertura de contato no substrato corresponde uma saliência de solda no *chip*. Após a montagem, a conexão elétrica é feita por fusão da solda, em componentes individuais ou no módulo todo. A tensão superficial da solda faz com que as peças alcancem o alinhamento final.

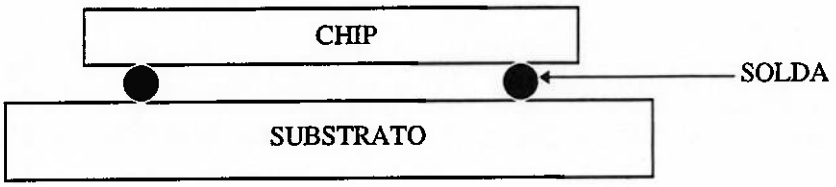


Figura 2.1: Montagem *Flip-Chip*

TAB: Nesta técnica cada circuito tem seus terminais que são soldados ao substrato. Dependendo de como são os terminais, o *chip* pode ficar com sua face voltada para cima ou para baixo (*flip*-TAB), o que terá implicações para o método de remoção do calor a ser empregado. Os circuitos integrados montados desta forma são dos mais fáceis de serem pré-testados, por apresentarem terminais que podem ser acessados por soquetes especialmente projetados para isso.

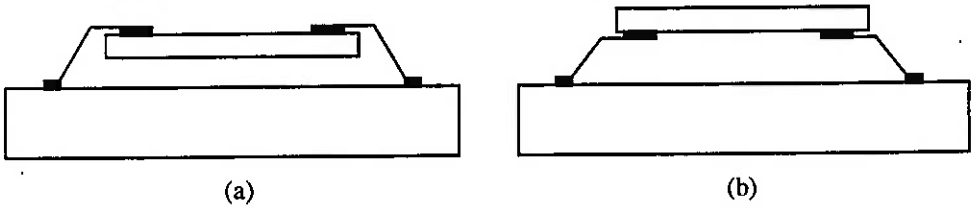


Figura 2.2: a) Montagem TAB
b) Montagem FLIP-TAB

Wire-Bond: Este tipo de montagem implica no posicionamento do circuito sobre o substrato e na utilização de fios soldados entre as aberturas de contato de *chip* e substrato. É o tipo de conexão mais empregado, sendo usado em cerca de 95% dos casos. Proporciona bom rendimento do processo de produção.

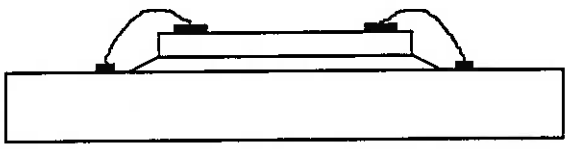


Figura 2.3: Montagem por *wire-bond*

ACAF: Esta é a sigla para "*Anisotropic Conductive Adhesive Film*". Esta técnica utiliza filmes adesivos contendo partículas condutivas

posicionadas em uma direção perpendicular ao plano do filme. Este filme é aplicado sobre o substrato e sobre o filme são posicionados os *chips*. O conjunto é submetido então a altas temperaturas sob pressão, e então resfriado. Neste processo o filme diminui de volume fazendo então que haja contato por pressão entre componente e substrato [Hagg92].

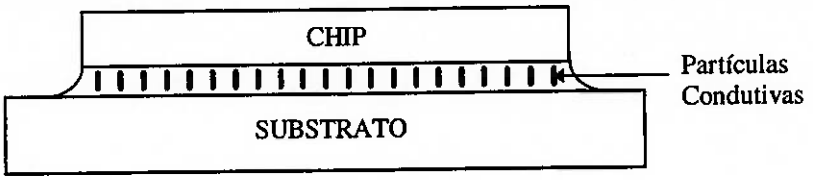


Figura 2.4: Montagem com ACAF

Revestimento. Nesta modalidade os circuitos são colocados em cavidades no substrato e as interconexões em filme fino são depositadas sobre os componentes. As conexões aos contatos dos circuitos são formadas no próprio processo de fabricação do filme fino [Daum93].

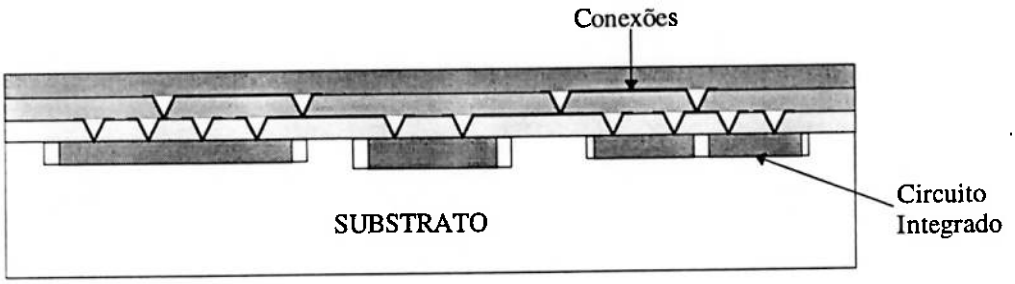


Figura 2.5: Montagem por revestimento

A tecnologia a ser utilizada deve por princípio permitir a conexão de centenas de entradas e saídas para cada *chip*, uma vez que, por se tratar o módulo *multichip* de uma solução para alta integração, estes *chips* são em sua grande maioria de um grau de integração compatível, sendo comuns circuitos com 300 conexões. Todos os tipos citados atendem a essa necessidade, porém o crescente número de conexões exigirá avanços tecnológicos nesta área.

Das formas de montagem apresentadas, *Flip-Chip*, ACAF e Revestimento são as que apresentam as maiores vantagens do ponto de vista da correspondência aos objetivos do uso de módulos *multichip*. Esses métodos permitem minimizar as distâncias entre os *chips*; por não consumirem os espaços ao lado dos circuitos para a realização das conexões permitem maiores densidades de entradas e saídas, bem como seu maior número em cada um dos *chips*. Apresentam também menores atrasos parasitários e

melhor capacidade de adaptação às características de linhas de transmissão. Como desvantagem temos a menor disponibilidade de circuitos para este tipo de interconexão no mercado, sendo portanto mais adequados quando se pretende projetar os circuitos necessários.

Podemos ainda contar com a possibilidade de usar duas ou mais técnicas em um mesmo módulo, de acordo com as características e disponibilidade de cada um dos circuitos a serem utilizados.

Substrato

Há uma série de requisitos a que o substrato utilizado na confecção do módulo *multichip* deve atender, em maior ou menor grau, conforme a aplicação a que se destina. De uma forma geral, as características de um bom substrato são:

- Possibilidade de pequenas distâncias entre *chips*
- Alta densidade de interconexão
- Baixa constante dielétrica
- Alto número de entradas e saídas do módulo
- Expansão térmica compatível com os *chips*
- Alta condutividade térmica

A minimização do tempo de propagação dos sinais está relacionada diretamente com a constante dielétrica do substrato e a distância mínima entre *chips*. Esse tempo é expresso pela equação (2.1):

$$T = \frac{l\sqrt{\varepsilon_s}}{c} \qquad (2.1)$$

onde: T: tempo de propagação
l: comprimento da ligação
ε_S: constante dielétrica do substrato
c: velocidade da luz

Portanto, o fator que realmente importa, que é o tempo de propagação do sinal, está atrelado a dois fatores que por vezes caminham em direções opostas, ou seja, muitas vezes um substrato de menor constante dielétrica exige maiores comprimentos de interconexão devido a maiores larguras e espaçamento mínimos associados a esse substrato.

Com o aumento de número de *chips* no módulo e do número de portas lógicas por *chip* aumentam também as exigências de alta densidade de interconexão. Essa densidade não está ligada somente à menor largura de trilha e menor espaçamento entre trilhas, mas também ao número de camadas possível do ponto de vista econômico e tecnológico. Com o aumento das densidades exigidas, tecnologias de filme fino depositado vão adquirindo maior importância, já que permitem densidades de interconexão de 4 a 12 vezes maiores que a tecnologia de filme espesso em substrato cerâmico. O desenvolvimento de supercondutores [Glas93] deve permitir no futuro menores larguras de interconexão e conseqüentemente maiores densidades.

A compatibilidade do coeficiente expansão térmica do substrato e dos *chips* adquire importância quando são utilizadas tecnologias de montagem dos *chips* onde a conexão das entradas e saídas é feita diretamente entre *chip* e substrato, como no caso de *Flip-Chip*, ACAF e Revestimento. Nesses casos, após um certo número de ciclos térmicos poderá haver rompimento de conexões por fadiga do material.

Conforme a montagem, também é importante a condutividade térmica do substrato. Se o *chip* é montado sobre o substrato como é o caso do *Wire-Bond* e do TAB, a dissipação térmica se dá principalmente através desse. Para sistemas de alta integração e portanto grande dissipação, alternativas como substrato de diamante [Glas93] estão sendo viabilizadas para uso comercial.

A escolha de um ou outro substrato não se limita porém à otimização de todos os parâmetros mencionados; são levados em conta muitos outros fatores como a maturidade da tecnologia, confiabilidade, custos, requisitos específicos do projeto em questão e a combinação do desempenho do módulo com o do sistema ao qual se integra [Tumm92].

2. VANTAGENS DO USO DE MÓDULOS *MULTICHIP*

Diversas são as vantagens advindas do uso de módulos *multichip* em um sistema:

- Maior eficiência em área
- Miniaturização e redução de peso
- Maior desempenho em alta freqüência
- Maior confiabilidade
- Menores custos

Como já foi mencionado, os módulos *multichip* permitem a colocação dos *chips* a uma distância menor um do outro, dispensam encapsulamento individual de cada um deles, e oferecem maior densidade de interconexão. É de se esperar portanto a maior eficiência em área e a miniaturização, com a conseqüente redução de peso do sistema.

A eficiência em termos de área pode ser expressa como a relação entre a área ativa e a área total ocupada pelo sistema, comumente dada em porcentagem. A figura 2.6 [Mess87] ilustra a faixa de eficiência em área para diversas tecnologias, relacionada também à eficiência com a largura das linhas de conexão empregadas nos diversos processos.

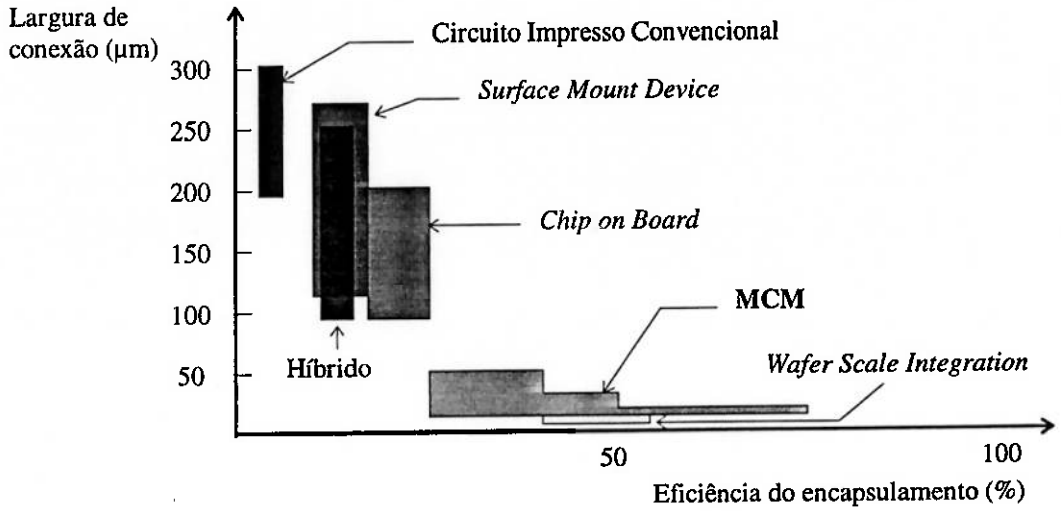


Figura 2.6: Eficiência em área de diversos tipos de encapsulamento [Mess87].

O melhor desempenho dos módulos *multichip* é uma vantagem que não surgiu como conseqüência, ou seja, é o que se buscava quando esta alternativa foi desenvolvida para atender requisitos da arquitetura dos computadores de grande porte. Com o avanço tecnológico dos semicondutores e a drástica queda nos tempos de ciclo de processamento, atrasos típicos de propagação encontrados em uma placa de circuito impresso foram tornando-se impraticáveis, o que primeiro se fez sentir nos sistemas maiores e de desempenho de ponta, e mais recentemente nas estações de trabalho e microcomputadores. A figura 2.7 [Hagg92] ilustra o comprimento máximo de conexão, limitando-se o atraso devido a este fator a 25% do ciclo de relógio, para alguns valores de constante dielétrica do substrato. Esse atraso exclui a ação de resistências e capacitâncias parasitárias, que também são menores no caso dos módulos *multichip*.

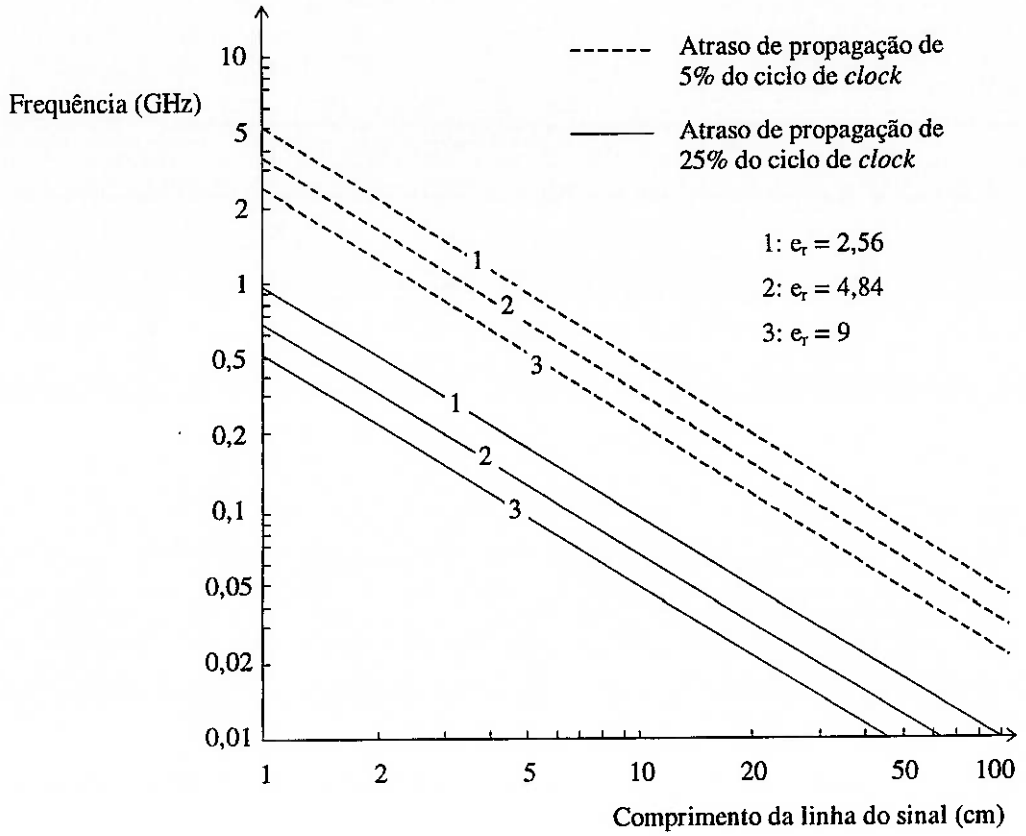


Figura 2.7: Limitação do tamanho da conexão em função da freqüência de operação para um atraso igual a 5% e 25% do ciclo de *clock* [Hagg92].

A outra solução para este problema específico é a reunião de vários *chips* em um único, o que realmente é feito em muitos casos, no limite em que o rendimento de processo permita a viabilidade desta opção. A solução mais genérica ainda é o módulo *multichip*, que permite a colocação dos circuitos muito próximos um do outro, tornando os comprimentos de ligação e as capacitâncias associadas compatíveis com as freqüências de relógio na faixa de GHz, previstas para um futuro próximo.

Outro efeito da redução do comprimento das ligações e da eliminação de um nível de encapsulamento é o aumento da confiabilidade do sistema. Isso foi fundamental para a viabilização dos módulos *multichip*, já que são de difícil reparo e têm um valor bastante superior ao dos componentes que o compõe individualmente, os quais poderiam ser substituídos separadamente caso não estivessem integrados em um mesmo sistema. Se a taxa de falhas se mantivesse a mesma, os custos de reposição de peças ou de reparos, no caso dos módulos de maior valor, poderiam se tornar proibitivos.

A maior confiabilidade não surge imediatamente das características dos módulos *multichip* já citadas, estas características apenas permitem que a confiabilidade seja maior, devido ao número de conexões necessárias muito menor e ao menor comprimento das ligações. Essas vantagens poderiam não se traduzir em ganho de confiabilidade em função da tecnologia empregada ser mais sensível e potencialmente mais sujeita a falhas. Para assegurar que realmente se tenham tais ganhos são

necessários cuidados que incluem um projeto voltado para este fim e um controle cuidadoso das etapas de fabricação e de teste.

Em termos de custo, pelos mesmos motivos já expostos, espera-se que haja uma vantagem a favor dos módulos *multichip* quando seu uso se justifique, ou seja, quando o número de interconexões for elevado. O gráfico mostrado na figura 2.8 [Knas84] ilustra comparativamente os custos por interconexão para algumas alternativas de encapsulamento.

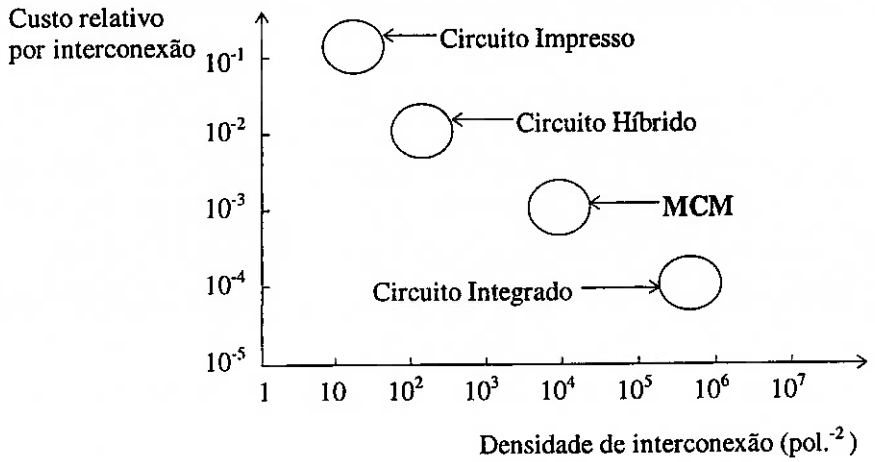


Figura 2.8: Custo por interconexão para várias tecnologias de encapsulamento [Knas84].

Outros custos que devem ser considerados são os de processo, montagem, teste e relativos ao rendimento do processo. O maior impacto desses fatores se dá sobre a escolha do tipo de tecnologia *multichip* a ser empregada.

Além dessas vantagens deve-se mencionar a flexibilidade proporcionada pela possibilidade de combinar diversas tecnologias, como por exemplo componentes analógicos e digitais ou circuitos de silício e arseneto de gálio em um mesmo módulo.

3. DESVANTAGENS DO USO DE MÓDULOS *MULTICHIP*

Como qualquer outra tecnologia os módulos *multichip* não representam a solução perfeita para todos os sistemas eletrônicos. Existem desvantagens que, se não forem plenamente compensadas com o aproveitamento dos pontos fortes já apresentados, tornam seu uso desaconselhável. Os pontos fracos a serem considerados são:

- Custo
- Baixa disponibilidade de componentes prontos
- Alta concentração de dissipação de potência
- Retrabalho e manutenção
- Teste

A questão do custo, como já foi colocado, deve ser analisada a cada caso. O processo para sua obtenção é sem dúvida mais custoso que o processo de fabricação de uma placa de circuito impresso. A possível vantagem a favor do módulos *multichip* surge quando a redução em tamanho ou o nível de tecnologia necessário na adoção da placa de circuito impresso sejam tais que considerado o sistema haja redução de custo. Atualmente isso não se dá com grande freqüência, assim, na maioria dos casos o custo de desenvolvimento e fabricação é um empecilho para a adoção dos módulos *multichip*. Somando à questão do custo temos a dificuldade de encontrar-se o componente desejado no mercado. Isso torna o desenvolvimento além de mais custoso mais lento e arriscado, necessitando de um volume de produção previsto alto o suficiente para justificar este procedimento.

Existem diversos fatores que contribuem para tornar o projeto destes módulos uma tarefa delicada. Há a questão da distribuição adequada dos sinais e da alimentação, o problema do particionamento do sistema em componentes, a preocupação com a eliminação de potência dissipada e os cuidados devidos para garantir a testabilidade do módulo [Scha92]. Distribuição de sinais e alimentação, e particionamento são problemas comuns a qualquer tecnologia que seja adotada, sem grandes diferenças. Quanto à dissipação térmica e à testabilidade existem agravantes inerentes a essa classe de componentes, devido à alta concentração de circuitos integrados e o pouco acesso à possíveis pontos de teste, nos *chips* e nas interconexões. Portanto esforços especiais são empreendidos nestas áreas, sendo, como já foi mencionado, exatamente a questão da testabilidade a que é abordada neste trabalho.

Finalmente, mesmo identificado um circuito defeituoso entre os que compõem o módulo, resta ainda o problema de repará-lo trocando este circuito, o que em muitos casos compensa a despeito das dificuldades, dado o alto custo de alguns desses módulos. É consideravelmente mais difícil a substituição de uma parte com defeito neste caso do que em uma placa de circuito impresso, mas é um problema que pode-se considerar como resolvido na maior parte das técnicas utilizadas na fabricação. Mesmo a conexão por revestimento permite que todas as camadas de ligações sejam retiradas, um circuito seja substituído e as interconexões novamente refeitas. No entanto, qualquer que seja a tecnologia, a utilização dos módulos *multichip* implica em custos de manutenção mais altos, que, para que seja viável, devem ser compensados através de outras vantagens que podem ser obtidas através de seu uso.

4. O PROJETO DE MÓDULOS *MULTICHIP*

Para esta classe de sistemas, um projeto envolve considerações específicas, o que inclui alguns aspectos que demandam cuidados maiores em seu tratamento, seja por suas características inerentes, seja por causa do alto desempenho que se procura extrair deles.

O próprio desenvolvimento dos módulos *multichip* está subordinado ao desenvolvimento de técnicas de projeto e de ferramentas de automação adequadas que implementem e permitam a aplicação destas técnicas. Schaper [Scha92] apresenta um amplo leque de considerações sobre esse assunto.

O projeto de um módulo *multichip* é equivalente ao projeto de todo um sistema complexo envolvendo componentes avulsos. Partindo desta comparação, e lembrando que o projeto de tais sistemas usualmente envolve especialistas e ferramentas dedicadas a cada fase do projeto e que os módulos *multichip* têm peculiaridades ausentes nas placas de circuito impresso (dimensões menores, freqüências maiores, maior número de camadas de roteamento, maior densidade, etc.) chega-se na importância da existência de uma metodologia de projetos que englobe e integre adequadamente todas as necessidades.

De igual importância é a existência de ferramentas de *software* que provejam o suporte à implantação das metodologias que sejam propostas. Somente o grande avanço dos sistemas de automação de projetos eletrônicos ocorrido nestes últimos anos proporcionou condições para que esta necessidade seja pelo menos parcialmente atendida. Os recursos disponíveis atualmente, não só em termos de *software* como também de *hardware*, permitem a síntese, simulação, particionamento, execução do projeto físico e verificação de regras de projeto -- físicas e elétricas -- de sistemas de grande complexidade. Ao mesmo tempo em que aumenta a capacidade dessas ferramentas, são desenvolvidas outras voltadas especificamente ao projeto de módulos *multichip* [LaPo93] [Vemu93] [Stae93] [Cho93] [Cade92].

A seguir são enfocados os pontos mais relacionados com o escopo deste trabalho. Existem vários trabalhos publicados que proporcionam uma abordagem completa do assunto.

Síntese Automática

Dadas as pressões de projetos cada vez mais complexos e tempos de desenvolvimento progressivamente mais curtos, um número crescente de projetistas está se tornando adepto da síntese automática de circuitos. É seu desejo trabalhar com níveis cada vez mais altos de descrição, em um maior grau de abstração. Considerando os módulos *multichip*, o ideal seria pensar no sistema completo e assim descrevê-lo em alto nível, sem a preocupação com arquiteturas específicas, particionamento e reutilização de

partes do projeto. A partir de uma tal descrição, mesmo genérica e incompleta, um sistema de síntese sofisticado seria apto a reunir blocos ou mesmo circuitos integrados capazes de realizar as funções desejadas, de acordo com restrições de tamanho, custo e freqüência de operação. Eventualmente poderiam ser apresentadas figuras de mérito que auxiliassem a escolher entre as alternativas viáveis, que podem também incluir alternativas de diferentes tecnologias para implementação dos *chips*. Então, uma vez descrito completamente o sistema, as partes seriam reunidas com a síntese da lógica necessária para isso.

Os sistemas de síntese lógica atuais contribuem de forma bastante eficiente para o desenvolvimento de projetos complexos. No caminho para o ambiente de síntese descrito estão se tornando disponíveis ferramentas focadas em arquiteturas específicas, como DSP e controle de mémoria, por exemplo.

Particionamento

O início do problema do particionamento está na inviabilidade de reunirmos todo o circuito em uma mesma pastilha ou lâmina. As razões para isso são os altos custos decorrentes do baixo rendimento de processo que isso acarretaria, os tempos elevados de propagação de sinais por grandes extensões de conexões no semicondutor, e as dificuldades para realizar o teste de circuitos muito grandes.

Isso determina a divisão do projeto em diversas pastilhas. Muitas vezes o tamanho de cada uma destas pastilhas é reduzido procurando-se um maior rendimento na sua fabricação, porém trazendo conseqüências como o maior número de entradas e saídas total e o aumento do número de interconexões. Em termos de testabilidade, se por um lado circuitos menores são mais facilmente testáveis, o módulo resultante terá o problema de teste agravado.

Se a procura de soluções de compromisso neste caso já é por si só uma questão complexa, o desafio se amplia ao considerarmos a arquitetura do projeto. Além da decisão de o quanto dividir, é preciso determinar que parcela do circuito completo colocar em cada *chip*. Tudo isso é ainda familiar aos projetistas habituados a montar sistemas com ASICs sobre placas de circuito impresso. Procura-se agrupar blocos críticos ao mesmo tempo em que se procura reduzir o número total de pinos dos ASICs, bem como o tamanho das ligações críticas na montagem final. O uso da técnica de projeto por *pipeline* ajuda a transferir ligações para a placa de circuito impresso já que tal técnica produz muitos sinais não críticos em termos de tempo de propagação. A abordagem, se está se pensando em apenas um módulo *multichip*, é muito parecida. Apenas é necessário lembrar que o custo, em termos de desempenho e área, de transferir o sinal para fora de cada *chip* é bem menor, pois a corrente extraída de cada pino é reduzida, o que incentiva o uso de *chips* menores. Tecnologias de montagem como "*flip-chip*", que permitem a colocação de pinos em toda a área do circuito diminuem por sua vez restrições ao número de entradas e saídas desejável.

Quando se aborda os módulos *multichip*, é acrescentada uma nova dimensão à situação descrita. Fatores como tamanho máximo do módulo, número de pinos e dissipação

térmica, além de dificuldades práticas para o teste impedem a integração de circuitos muito grandes em um só módulo. Agora devem ser feitas decisões sobre o que colocar em cada circuito, em cada módulo e algumas vezes, em cada placa que comporá o sistema total. A otimização global só é possível considerando todas as interações entre esses níveis e suas diferentes características particulares. Questões relativamente simples, como atribuição de pinos, podem se tornar complexas, se for desejado otimizar a conexão das entradas e saídas das pastilhas internas ao módulo com os pinos deste, e os pinos dos módulos entre si. O número de opções se torna tão gigantesco que análises exaustivas tornam-se impossíveis e a intuição do projetista tem que suprir as deficiências das ferramentas automáticas atuais.

Restrições e Regras de Projeto

Uma vez definido o circuito e sua partição, o próximo passo é garantir que seu funcionamento seja como especificado. A simples realização de simulações por si só não é mais suficiente para garantir isso dada a grande influência da implementação física. Todas as restrições que se espera que sejam respeitadas devem ser passadas à ferramenta de posicionamento e roteamento, que por sua vez deve observar todas as regras de projeto em todos os níveis. Além dos limites físicos, limites de tempo de propagação, interferência entre sinais, reflexão e perdas ôhmicas são traduzidos em termos de capacitâncias, resistências e indutâncias e linhas de transmissão.. Esses parâmetros são enfim relacionados com larguras e comprimentos de linhas de ligação, separação entre linhas, espessuras de dielétrico e condutor e constantes dielétricas.

Teste

O teste de um módulo *multichip* envolve a verificação da satisfação das especificações do projeto, em termos tanto de funcionalidade como desempenho. Como em outros sistemas, não se trata apenas de identificar falhas no processo de fabricação do componente, mas também detectar falhas devido a limitações das ferramentas de projeto em termos de simulação ou mesmo falhas decorrentes do uso.

A identificação de falhas de fabricação que conduzam a erros funcionais não é um problema trivial quando se trata de um grande circuito; a geração de vetores de teste que exercitem todos os possíveis pontos de problema do circuito exigem uma combinação de boas ferramentas de geração automática de vetores de teste, simulação de falhas e experiência e talento do projetista. Por sua vez, a localização de erros de projeto (existência de corridas críticas, violações de tempos de *setup* e *hold*, geração de pulsos espúrios, sobrecarga de saídas, existência de *crosstalk* e reflexões, efeitos de capacitância e resistência de trilhas e planos de alimentação, etc.), quando estes escapam das simulações funcionais e vêm a se manifestar durante o uso, muitas vezes de forma esporádica, é ainda mais complexa, pois exige a realização do teste na freqüência de operação do circuito, um profundo conhecimento do projeto e o acesso a pontos internos que permitam o rastreamento do problema. Dificuldades semelhantes

são encontradas para o isolamento de falhas devidas ao uso, tarefa importante na realização de manutenção ou quando se procura a causa de uma vida útil do circuito menor que a esperada.

Além da alta complexidade comum aos módulos *multichip*, o que representa um desafio à especificação dos testes a serem realizados, o difícil acesso a pontos interiores do componente aliado às pequenas dimensões tornam difícil ou mesmo impossível a sua realização.

Os meios de acesso a pontos internos do módulo *multichip* são através de pontas de prova, fixas ou móveis, e de feixe de elétrons ou plasma. Esses meios permitem verificar, antes da montagem das pastilhas, a integridade das trilhas e medir parâmetros como a capacitância, indutância e resistência do substrato e das trilhas. Após a montagem dos componentes sobre o substrato, esse acesso que normalmente é complicado fica ainda mais prejudicado, comprometendo o teste das ligações. Além disso, durante o processo de montagem, componentes podem ter sido danificados ou podem ter tido sua posição trocada. A utilização de pontas de prova nesse caso pode não ser possível ou pode ser potencialmente danosa. Todo o acesso ao módulo ficaria então restrito aos seus pinos. O problema que isso representa pode ser sentido lembrando que um módulo *multichip* é semelhante a uma placa de circuito impresso, e imaginando como seria testar toda uma placa a partir de seus terminais apenas.

Outra questão referente a esse assunto é a qualidade dos componentes usados na montagem. É muito mais fácil realizar o teste de um componente encapsulado do que um ainda por montar. Isso faz com que freqüentemente os componentes utilizados não tenham a mesma confiabilidade, pela menor quantidade de testes realizados, que aqueles usados em placas de circuito impresso.

5. ASPECTOS DA TESTABILIDADE DOS MÓDULOS *MULTICHIP*

Buscar soluções eficientes para o problema da testabilidade implica em primeiro lugar a identificação de quais são as conseqüências das deficiências nos testes realizados, e a causa das dificuldades na sua aplicação.

Duas são as principais preocupações que demandam a realização de testes dentro do processo produtivo, que são a quantidade de retrabalho necessária e o nível de defeito das peças (proporção de peças com defeito aprovadas e enviadas ao mercado).

A porcentagem de peças que necessitam retrabalho será maior quanto mais se adiarem os testes possíveis de serem realizados ao longo da produção. A redução desta porcentagem requer que todos os circuitos integrados utilizados sejam pré-testados e que as conexões no substrato sejam testadas, quando possível, antes da montagem dos componentes. Boa parte da responsabilidade recai sobre os fornecedores de circuitos

integrados, pois nem sempre é viável para o fabricante de módulos *multichip* testar os *chips* não encapsulados, seja por razões econômicas, seja por não ter o conhecimento suficiente sobre a arquitetura e o projeto dos componentes empregados.

O rendimento da fabricação dos módulos *multichip*, ou seja, a porcentagem de componentes que não têm necessidade de serem retrabalhados ou ainda descartados, é o produto do nível de aproveitamento de todos os componentes que o integram, do substrato e das interconexões realizadas. Como geralmente há grande quantidade de componentes e certamente de conexões, cada um deles deve apresentar um alto aproveitamento para não levar a um péssimo resultado na fabricação. Como exemplo, considerando substrato e conexões perfeitas, considerando apenas os componentes e um módulo com 50 deles, se cada um tiver um rendimento de 90%, apenas 0,5% dos módulos fabricados não teriam defeitos. Subindo o aproveitamento para 99%, ainda assim teríamos 40% dos módulos fabricados com defeito. Um resultado aceitável só é atingido quando 99.9% dos componentes utilizados não têm falhas, neste caso apenas 5% dos módulos necessitariam retrabalho ou seriam descartados.

Além da implicação do rendimento da produção nos custos há também influência deste parâmetro no nível de defeitos das peças aprovadas, juntamente com o nível de cobertura de falhas (porcentagem de falhas testadas e identificadas dentro do total do conjunto de falhas possíveis). Essa dependência é expressa por (2.2) [Brgl89].

$$ND = 1 - Y^{(1 - CF)} \quad (2.2)$$

Onde:

ND: Nível de defeito
Y: Rendimento da fabricação do módulo *multichip*
CF: Cobertura de falhas

Voltando para o nosso exemplo e considerando o melhor caso apresentado (95% de aproveitamento) e uma cobertura de falhas de 90%, teríamos um nível de defeito de 0,5%, bastante alto, principalmente considerando o alto custo dos módulos. Um nível bom seria obtido se a cobertura de falhas fosse também de 99,9%, resultando em 0,005% de peças falhas entre as entregues (1 a cada 20.000).

Podemos notar portanto que bons resultados dependem tanto de um processo produtivo bem cuidado como de uma boa cobertura de falhas.

Soluções Para o Problema da Testabilidade

Muitos componentes de *chip* único são atualmente complexos suficiente para que um abordagem do tipo "aplicar todas as combinações possíveis de entrada em todos os estados possíveis e conferir as saídas" seja impraticável. Por exemplo, um processador popular como o 80386, se consideradas suas entradas e registradores internos, apresentaria um número de combinações possíveis da ordem de 10^{275}, o que quer dizer

que mesmo que um teste tivesse sido iniciado nos primórdios do universo, a parte já testada até hoje seria totalmente insignificante.

Testar completamente um componente, no entanto, não significa ter que passar por todas as combinações possíveis mencionadas acima. Boa parte dessas combinações são impossíveis durante o funcionamento e a maior parte repete condições de teste já aplicadas. Para termos uma idéia de como esse processo pode ser otimizado, o auto-teste do processador 80386, abrangendo cerca de metade da lógica do componente, é executado a 40MHz em menos de 14ms [Inte91] [Gels86].

O primeiro método para a otimização da geração dos vetores de teste foi a geração manual, feita geralmente pelos próprios projetistas, ou por pessoas que tivessem profundo conhecimento do sistema. Com o surgimento da simulação de falhas, pode-se fazer uma avaliação da cobertura de falhas proporcionada pelos vetores determinados.

Outra alternativa para a obtenção dos vetores de teste é o ATPG - *Automatic Test Pattern Generation*. Ferramentas para a geração automática de vetores de teste podem prover um conjunto próximo do ideal para a realização do teste, porém sua aplicação ainda está restrita a circuitos não tão complexos ou que incorporem estruturas que melhorem sua testabilidade.

Com o aumento da complexidade, os tempos de geração dos vetores foram se tornando muito extensos, seja manual ou automaticamente, e então foram surgindo técnicas visando a redução da complexidade dos circuitos através de seu particionamento em circuitos menores. Isso não quer dizer necessariamente dividir o sistema em um maior número de componentes, e sim realizar o projeto de tal forma que se tenha acesso para controlar e observar partes determinadas da lógica, tornando-as o máximo possível independentes entre si. Um circuito que tivesse 20 entradas e 20 registradores, e que permitisse uma divisão em quatro circuitos com 10 entradas cada para efeitos de teste, teria a complexidade de um teste exaustivo reduzida por um fator de mais de 200 milhões (de 2^{40} para 4 X 2^{10}). Formas para a obtenção de maior controlabilidade, observabilidade e de particionamento incluem a adição de pinos dedicados para esse fim, a adoção de técnicas de *scan* e a inclusão de auto-teste interno (BIST - *Built In Self Test*).

Uma importante iniciativa conduzida nesta direção foi a definição do padrão IEEE 1149.1, provendo normas para a implementação de acesso de teste e *boundary-scan*, que permite isolamento e acesso a todos o pinos dos circuitos montados em uma placa que estiverem em conformidade com este padrão.

A associação de auto-teste e *boundary-scan* não apenas soma as vantagens dessas duas técnicas, mas as potencializa mutuamente. Através dos pinos de teste podem ser inseridas as seqüências que isolam um circuito do restante do sistema e dão início ao procedimento de teste (o que está também definido no padrão IEEE 1149.1). O auto-teste pode então ser realizado na velocidade de operação do circuito, o que é impossível utilizando-se apenas técnicas de *scan*, dispensando a necessidade de um conhecimento interno da lógica, já que os vetores são gerados internamente. Adicionalmente, o teste pode ser realizado tanto no circuito antes da montagem, como depois de inserido em

eventuais subsistemas, até a montagem no sistema final. O teste das conexões pode ser realizado através do *boundary-scan.*

A perfeita aplicação dos métodos mencionados exige a compatibilidade de todos os componentes do sistema que devam ser testados. Componentes "de prateleira" que não atendam a este requisito e que venham a ser utilizados com os outros não inviabilizam o teste de todos os outros, e podem inclusive ter seu próprio teste facilitado pela inserção de vetores de teste através dos componentes dentro do padrão utilizado. A não conformidade de um ou poucos circuitos dentro de um grande sistema pode mesmo não ter grande impacto, principalmente se estivermos trabalhando com placas de circuito impresso, caso em que podem ser adicionadas pontas de prova que permitam acessos a pontos internos. Quando trabalhamos com módulos *multichip*, por outro lado, esse acesso não é tão fácil, e temos que encarar a dificuldade de trabalhar com uma grande maioria de circuitos "de prateleira" que ainda não dispõe de qualquer facilidade para teste.

Capítulo 3

Abordagens para Teste

1. O PADRÃO IEEE PARA *BOUNDARY-SCAN*

O padrão IEEE 1149.1-1990, chamado *"Standard Test Access Port and Boundary-Scan Architecture"*, define diretrizes para a implementação de uma lógica de teste padronizada com os seguintes objetivos [IEEE90]:

- Testar as interconexões entre circuitos integrados montados sobre um substrato, por exemplo uma placa de circuito impresso, tanto o seu estado (curtos e abertos) como a sua correção.
- Testar o funcionamento do próprio circuito integrado.
- Testar a interação entre os componentes do sistema e o correto funcionamento do mesmo.
- Observar e alterar a funcionalidade do circuito durante a operação normal do sistema.

Características da Lógica de Teste Proposta

Um sistema projetado de acordo com este padrão permite que vetores de teste sejam inseridos por meio de uma linha de registradores construída através dos pinos de entrada e de saída dos componentes e que da mesma forma sejam extraídos os resultados da operação com esses vetores de teste para análise. Dessa forma, vetores de

teste usados para o teste do circuito após sua fabricação podem ser novamente utilizados para verificar a continuidade do funcionamento depois da montagem no substrato e também periodicamente durante a operação, conforme a necessidade.

Quando presente em vários componentes do sistema permite que estímulos colocados em pinos de saída sejam lidos em pinos de entrada de outros circuitos, possibilitando assim o teste das interconexões. Ao mesmo tempo permite a isolação de cada circuito para que sejam realizados testes livres da interferência de estímulos externos.

Além da implementação básica que proporciona as características descritas acima, a existência desta estrutura permite ainda que outras facilidades sejam acrescentadas, como por exemplo auto-teste, particionamento do circuito e linhas internas de *scan*, como as que são exigidas por uma metodologia de projeto para testabilidade.

Descrição

Todo o acesso à lógica de teste proposta é feita por uma estrutura chamada de "*Test Access Port*" ou "TAP". Esta inclui pinos de *clock* para teste (TCK - *Test Clock*), seleção de modo de teste (TMS - *Test Mode Select*), dado de entrada para a linha de *scan* (TDI - *Test Data Input*), dado de saída da linha de *scan* (TDO - *Test Data Output*) e, opcionalmente, inicialização assíncrona do TAP (TRST - *Test Reset*). Todos esses pinos são obrigatoriamente de uso exclusivo para o TAP, não podendo ser compartilhados com outros sinais do circuito, o que, caso ocorresse, comprometeria as características de testabilidade pretendidas. Compartilhar, por exemplo, o mesmo sinal de *clock* para teste e operação eliminaria a possibilidade de funcionamento normal do circuito durante a realização dos testes. Alternar o função de alguns pinos durante o teste faria com que toda a lógica, interna no caso de pinos de entrada e externa para pinos de saída, dependente desses pinos tenha sua testabilidade comprometida ou até impossibilitada.

Há diversas maneiras de conectar os circuitos assim providos, sendo a que menos exige conexões do sistema com o exterior apresentada na figura 3.1.

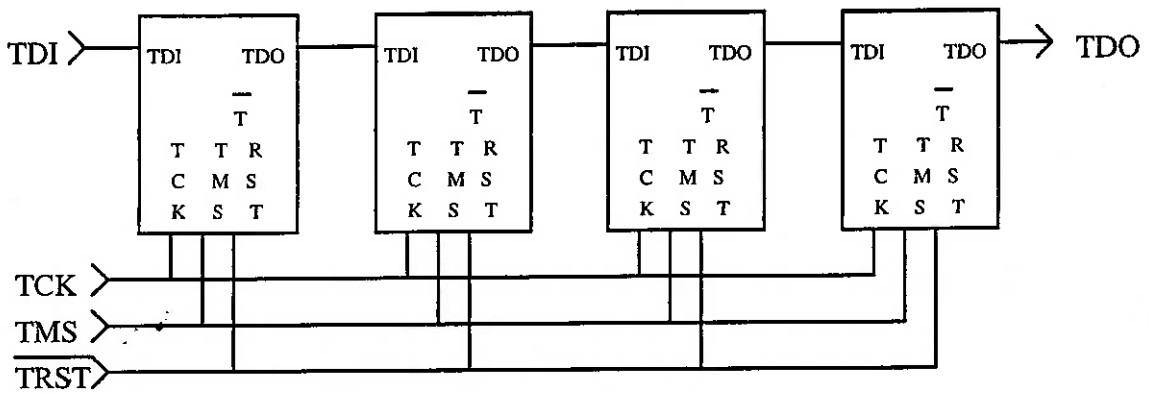


Figura 3.1: Conexão dos circuitos com um *port* de teste.

Tal tipo de conexão, entretanto, exige um tempo maior para a inserção de vetores de teste e sua extração, uma vez que o tamanho total da linha de registradores é igual à soma do número de pinos em teste de todos os circuitos. Na medida da existência de mais conexões disponíveis a um testador externo, é possível a ligação das linhas de teste em paralelo até que, no melhor caso o tamanho do vetor a ser inserido seja o da maior linha de registradores presente entre os circuitos.

A realização do teste a partir destes sinais pode ser a partir de então realizada por um equipamento dedicado, por circuitos adicionados ao sistema ou ainda através de uma ligação a um computador rodando um programa adequado.

Componentes do Sistema de Teste

Para a adesão ao padrão citado é necessário que a lógica de teste inclua, além do TAP, os seguintes componentes:

- Controlador do TAP
- Registrador de instruções
- Registradores de dados

Os registradores mencionados são conectados paralelamente entre TDI e TDO e sua seleção é realizada pelo controlador. Este por sua vez interpreta os sinais recebidos em TMS nas bordas de subida de TCK para realizar esta seleção, bem como a determinação do estado em que se encontra o circuito de teste.

A estrutura do controlador do TAP é montada em cima de uma máquina de estados definida na figura 3.2. Esta máquina possui 16 estados, sendo implementada com a ajuda de um registrador de 4 *bits*. A relação dos estados com os valores deste registrador pode ser definida pelo projetista, embora seja apresentada uma sugestão em [IEEE90], aproveitada neste trabalho e apresentada no capítulo 4.

Pode-se notar que uma das características desta máquina de estado é que quando o pino de TMS é deixado no estado lógico "1" a máquina de estados voltará para o estado de *reset* após no máximo cinco ciclos de TCK. Por definição, TMS em aberto deve ser interpretado como o estado "1". Nesse estado, toda lógica de teste está desabilitada e o funcionamento do restante do circuito não é de nenhuma forma alterado.

Os outros estados permitem a captura de um estado inicial, a introdução de dados e a leitura da linha de registradores (*shift*) e a atualização de registradores internos com os valores introduzidos por TDI. A operação dos registradores de dado e de instrução é idêntica, apenas diferindo estes pela sua forma de utilização.

Para exemplificar o operação desta máquina de estados, vamos supor duas funções sendo realizadas: a introdução de um vetor de teste e a leitura da ação deste vetor na lógica interna.

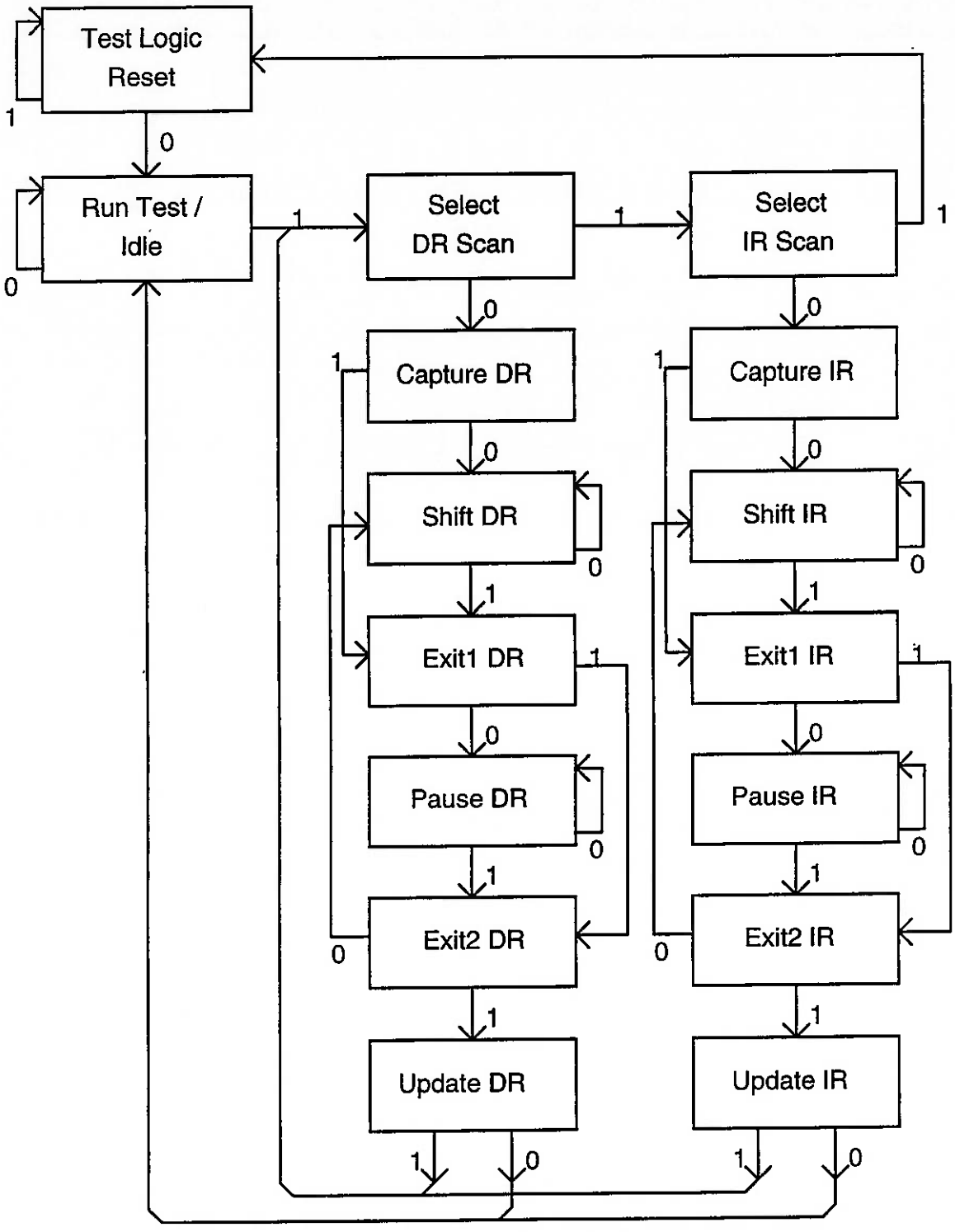


Figura 3.2: Diagrama de estados do controlador do TAP.

Para realizar a primeira das funções, é necessário que o registrador de *boundary-scan* esteja selecionado. Isso é feito carregando-se a instrução INTEST (a ser definida

adiante) no registrador de instruções. Neste ponto, no estado <Shift DR>, o vetor de teste é inserido na linha de registradores. O estado <Pause DR> pode ser usado como estado intermediário, caso haja necessidade de parar momentaneamente o teste para, por exemplo, o testador ler mais dados de um disco. No estado <Update DR> o vetor de teste é efetivamente aplicado às entradas do circuito. Haveria então uma passagem por <Run Test/Idle> ou voltar-se-ia imediatamente para <Select DR Scan>. Estando o circuito definido internamente, os resultados seriam carregados nas células ligadas aos pinos de saída em <Capture DR>. Finalmente, enquanto um novo vetor é inserido em <Shift DR>, os resultados são também extraídos, e tem início um novo ciclo.

Instruções

O padrão IEEE 1149.1 define algumas instruções que devem necessariamente estar disponíveis e outras opcionais. As obrigatórias são:

- BYPASS
- SAMPLE/PRELOAD
- EXTEST

E as opcionais, que se implementadas deverão estar de acordo com o padrão:

- INTEST
- RUNBIST
- IDCODE
- USERCODE

O motivo básico para a existência deste padrão é a garantia de que, ao se projetar um componente que siga suas regras, tenha-se a certeza de que toda a sua funcionalidade para teste poderá ser utilizada sem adaptações ou lógica adicional quando este é ligado a outros componentes que sigam o mesmo padrão. Como será notado adiante através da descrição das instruções citadas, as instruções opcionais, quando não implementadas não afetam em nada o teste de outros componentes, enquanto que a falta de instruções obrigatórias causam prejuízo ao teste do sistema.

Além destas instruções, públicas, ou seja, devem estar disponíveis ao usuário final e adequadamente documentadas, podem ser implementadas quaisquer outras instruções desejadas, também públicas ou, caso seja do interesse do fabricante, privadas. Neste último caso só há a obrigatoriedade de informar ao usuário instruções que podem potencialmente trazer dano ao componente se selecionadas.

Segue-se uma breve descrição das instruções citadas:

BYPASS

A lógica de teste definida neste padrão deve conter o chamado "*bypass register*", que é selecionado por esta instrução. Sua função é criar um caminho curto alternativo ao registrador de *boundary-scan* quando não é necessário que um ou mais componentes do sistema participem do teste em determinado momento. A utilidade desta instrução pode ser facilmente entendida quando se adota uma estrutura de ligação como a da figura 3.1. Neste caso, se em determinado teste só há a participação, digamos, do terceiro componente, economiza-se um tempo considerável não carregando os registradores de *boundary-scan* dos outros componentes.

Quando a instrução IDCODE não for oferecida, o código para a instrução BYPASS deve ser carregado no registrador de instrução no estado de controle <Test Logic Reset>.

SAMPLE/PRELOAD

Esta instrução tem duas utilidades básicas, que são a amostragem do estado dos pinos do circuito em um determinado instante durante a operação normal do sistema, e a pré-carga de um vetor de teste inicial na saída do registrador de teste como preparação para a execução de uma outra instrução, para que haja desde o início dados conhecidos nos registradores.

A amostragem dos sinais nos pinos durante a operação permite que se verifique o estado dos circuitos sem a necessidade do contato físico de uma ponta de prova, o que é especialmente útil no caso de um módulo *multichip*.

EXTEST

Esta instrução tem como objetivo o teste da lógica externa e das conexões entre os circuitos. Ao ser selecionada, no estado de controle <Update IR>, o conteúdo do registrador de *boundary-scan* (normalmente programado inicialmente através da instrução SAMPLE/PRELOAD) é transferido para os pinos de saída dos circuitos que estiverem executando esta instrução. A partir de então, no estado <Capture DR>, são lidos nos pinos de entrada dos diversos circuitos o resultado proveniente de uma lógica externa ou simplesmente o estado de pinos de saída conectados a essas entradas. O resultado é lido deslocando-se os dados em <Shift DR>, enquanto um novo vetor é introduzido, dando prosseguimento ao teste.

Circuitos do sistema que não disponham de registradores de *boundary-scan* podem ser testados com o uso desta instrução, desde que suas entradas possam ser controladas por outros circuitos que disponham desta facilidade.

INTEST

O uso desta instrução permite o teste estático da lógica interna ao circuito, sem interferências externas, mesmo quando montado em um sistema.

Como na instrução EXTEST, usualmente é carregado um vetor de teste através da instrução SAMPLE/PRELOAD, garantindo uma condição inicial conhecida e estados nos pinos de saída não prejudiciais ao sistema (como dois pinos *tri-state* conectados entre si ativos ao mesmo tempo). O resultado deste primeiro vetor é então capturado e transferido para fora para ser conferido. Da mesma forma que em EXTEST, um novo vetor é carregado enquanto o resultado do vetor anterior é lido, podendo assim ser aplicado logo a seguir.

Por ser um teste estático, dado o intervalo longo entre a aplicação de dois vetores, o circuito assim testado não deve apresentar limitações quanto à freqüência mínima de operação, o que ocorre, por exemplo, em uma memória dinâmica.

Um ponto importante a ser considerado durante a execução desta instrução é a geração de *clock* para o circuito. A borda ativa do *clock* deve ocorrer em um instante em que o vetor de entrada tenha tido tempo de se propagar através da lógica e esteja estável. Algumas alternativas para isso são:

- Controlar a entrada de *clock* de forma que durante o estado <Run Test/Idle> seja aplicado um pulso de *clock* proveniente da fonte externa original.
- Derivar um pulso de *clock* a partir de TCK durante o estado <Run Test/Idle>.
- Controlar o sinal de *clock* através do *boundary-scan*. Neste caso, um mesmo vetor é repetido para os dois estados do *clock*.

Mesmo em caso de componentes em que as duas bordas de *clock* sejam utilizadas em circuitos seqüenciais as formas de controle apresentadas acima podem ser utilizadas, as duas primeiras com adaptações, e a última através da geração de vetores adequados. Neste caso, deve-se notar que existem sinais de entrada que não podem mudar simultaneamente com determinada borda de *clock*. Assim, em vez de mudar as entradas, por exemplo, em todas as bordas de descida, seria necessário inserir um novo vetor com o mesmo valor para o sinal de *clock*, repetir o vetor alterando apenas o estado do sinal de *clock*, e adotar o mesmo procedimento para a geração da outra borda. Um exemplo é apresentado na figura 3.3. O impacto que trazem tais exigências é um motivo para que se evite este tipo de estilo de projeto.

Sinais	A	B	C	D	E	*Clock*
vetor 1	0	1	0	1	1	0
vetor 2	**1**	**0**	0	1	1	0
vetor 3	1	0	0	1	1	**1**
vetor 4	1	0	**1**	**0**	**0**	1
vetor 5	1	0	1	0	0	**0**

Figura 3.3: Vetores de teste incluindo o sinal de *clock*. A lógica acionada pelos sinais A e B é registrada pela borda de subida e a lógica de C, D e E pela borda de descida. As mudanças de estado entre vetores são assinaladas em negrito.

RUNBIST

Para componentes em que é implementado um ou mais testes internos (BIST - *Built-in Self Test*), esta instrução permite o seu acionamento por meio do *port* de teste.

Esse tipo de teste tem a grande vantagem de não exigir o carregamento de um grande número de vetores de teste através da entrada serial e de poder ser realizado à freqüência de operação normal do circuito.

A sistemática para seu acionamento é parecida com a da instrução INTEST. Um vetor inicial é inserido previamente pela instrução SAMPLE/PRELOAD, garantindo estados conhecidos nas saídas, que devem permanecer estáveis durante todo o auto-teste, e opcionalmente, uma semente ou um estado inicial nas células de entrada do registrador de *boundary-scan*. Essas próprias células podem, a critério do projetista, gerar os padrões de entrada, assim como as células de saída podem ser utilizadas para a compactação do resultado em uma assinatura.

O teste propriamente dito ocorre durante o estado <Run Test/Idle>, e sua duração deve ser especificada pelo fabricante. Essa instrução específica visa ser executada simultaneamente por todos os componentes integrantes do sistema que a suportem, embora possam haver instruções adicionais para auto-testes particulares, portanto ela é definida de forma que não haja limitações impostas por um dos circuitos aos outros. Isso implica em que, após o término do teste, o resultado se mantenha indefinidamente estável até sua leitura, de forma que componentes com durações de teste diferentes possam conviver em um mesmo sistema.

O sinal de *clock* para a realização do auto-teste pode ter duas origens: tanto pode ser derivado do sinal TCK durante o estado <Run Test/Idle>, como pode ser obtido diretamente do pino de entrada de *clock* do componente.

Ao término do teste em todos os componentes onde ele esteja sendo executado, o resultado é carregado no registrador conectado entre os pinos TDI e TDO no estado <Capture DR> para leitura.

IDCODE

Embora não seja obrigatória, quando esta instrução estiver disponível ela deve ser carregada no registrador de instrução no estado de controle <Test Logic Reset>. Quando selecionada, ela permite a extração do conteúdo de um registrador de identificação através de TDO. Isso permite a verificação dos componentes presentes em um sistema, quando puder haver variações nesses componentes entre diferentes versões de um mesmo tipo de sistema. Por ser carregada sempre em <Test Logic Reset>, esta verificação pode ser feita mesmo que não se saiba *a priori* o número exato de componentes na placa e conseqüentemente a sua ordem de ligação em uma linha de *scan*. Feito o levantamento dos componentes, o testador pode então programar o teste adequado para o modelo de sistema a ele conectado.

USERCODE

Esta instrução complementa a anterior no caso de componentes programáveis pelo usuário, e é obrigatória quando IDCODE é oferecida e a programação do componente não pode ser determinada de outra forma pela lógica de teste.

Seu funcionamento é idêntico ao da IDCODE, mudando apenas o registrador que é conectado para o acesso por meio de TDO.

2. ESTRATÉGIAS PARA TESTE DE MÓDULOS *MULTICHIP*

Como visto no capítulo 1, o problema da realização de testes é agravado quando o componente é um módulo *multichip*. Por esse motivo, este trabalho se propõe a reunir algumas técnicas de teste e implementá-las com base no padrão IEEE 1149.1.

Nesta aplicação, além da implementação básica, dois tipos de auto-teste serão abordados: o teste das interconexões entre os circuitos e o teste da lógica interna aos circuitos.

Para o teste das interconexões, o esquema proposto baseia-se em um trabalho de Hassan, Agarwal, Nadeau-Dostie e Rajski [Hass92], consistindo na determinação de padrões de teste e da compactação dos resultados obtidos.

Para o teste da lógica interna, será apresentada uma implementação de um *Cellular Automata Register* (CAR) programável e sua forma de programação para a realização de uma série de auto-testes.

Teste das Interconexões do Módulo *Multichip*

Para efeitos de teste, pode-se considerar um módulo *multichip* como sendo semelhante a uma placa de circuito impresso que não permite acesso direto aos pinos dos componentes ou a trilhas de roteamento. Além desse fator de dificuldade para a realização do teste, podemos esperar que o número de interconexões existentes em um módulo *multichip* seja equivalente ao de uma placa de circuito impresso bastante complexa. Por outro lado, o tipo de teste a ser realizado para a verificação da funcionalidade é bastante simples por não envolver nenhuma lógica entre a entrada e a saída, simplesmente a última deve reproduzir sempre a primeira. Essa particularidade permite que se utilize esquemas simples para a consecução de um auto-teste. O esquema escolhido para este trabalho apresenta esta característica de simplicidade aliada à vantagem de ser baseado em uma arquitetura de *boundary-scan*, ou seja, é adaptável à arquitetura pretendida.

Será feita adiante uma exposição do trabalho de Hassan, Agarwal, Nadeau-Dostie e Rajski [Hass92] e a seguir será apresentado o circuito desenvolvido neste trabalho para a sua implementação no contexto de módulos *multichip*.

Modelo de Falhas

Como vimos anteriormente, a interconexão entre os circuitos integrados que compõem um módulo *multichip* pode ser realizada de diferentes maneiras, conforme a tecnologia escolhida. Por esse motivo, um modelo baseado apenas em falhas do tipo *stuck-at* pode não ser o mais indicado em todos os casos. Assim, além desse modelo também serão consideradas as falhas de curto e aberto [Abra90].

As falhas tipo *stuck-at* são as mais empregadas na análise de problemas em circuitos integrados e podem ser descritas como mantendo toda uma rede de conexões em um valor fixo (1 ou 0) não importando o valor colocado no(s) pino(s) de saída conectados a esta rede.

Uma falha de curto circuito envolve no mínimo duas redes e consiste em tornar essas redes equivalentes a uma só. Nessa condição haverá no mínimo duas saídas ativas ao mesmo tempo, e, caso estas saídas estejam em estados lógicos diferentes, haverá um conflito. Dependendo da tecnologia empregada nessas saídas em conflito, uma delas prevalecerá. Embora possa ser previsto em cada caso qual será o resultado de tal conflito, em um universo de vários casos possíveis não podemos particularizar um determinado tipo de comportamento. De qualquer forma, na presença deste tipo de falha e na ocorrência de um conflito, um conjunto de entradas receberá um valor diferente do esperado, possibilitando a detecção da falha. Para efeito de análise do esquema de teste a ser apresentado, iremos considerar que em uma determinada rede pode haver uma falha do tipo curto-E ou curto-OU, querendo significar que o sinal que prevalecerá na rede será respectivamente o resultado de uma operação lógica E ou de uma operação lógica OU entre as saídas das redes envolvidas.

A falha de circuito aberto tem um tratamento mais complicado que as já descritas por não afetar toda a rede de conexão em que se manifesta. Por exemplo, se o ponto aberto for próximo a uma entrada, somente esta sofrerá os efeitos desta falha, e mesmo assim será difícil saber que espécie de comportamento esta entrada apresentaria. Entretanto pode-se esperar que em tal condição, entradas que não estejam sendo ativadas por nenhuma saída devido a uma falha deste tipo apresentem sempre o mesmo valor lógico (de forma semelhante a uma falha *stuck-at*), o que possibilita a detecção pela análise de todas as entradas de uma rede durante uma transição (de nível lógico 0 para 1 ou vice-versa). Caso haja múltiplas saídas ligadas à rede, a transição deverá ocorrer em todas elas, estando ativa uma por vez. A falha é descoberta quando esta transição não é acompanhada em todas as entradas (figura 3.4).

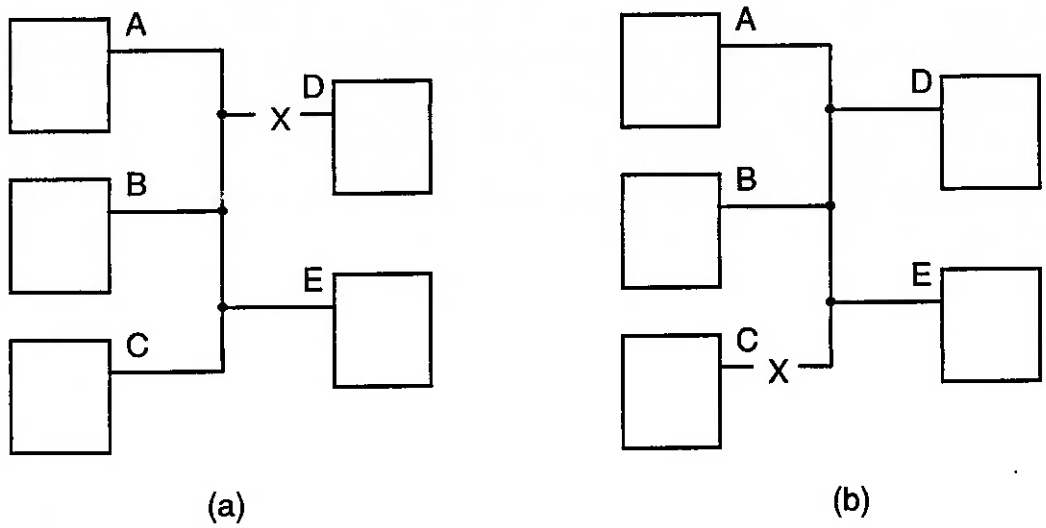


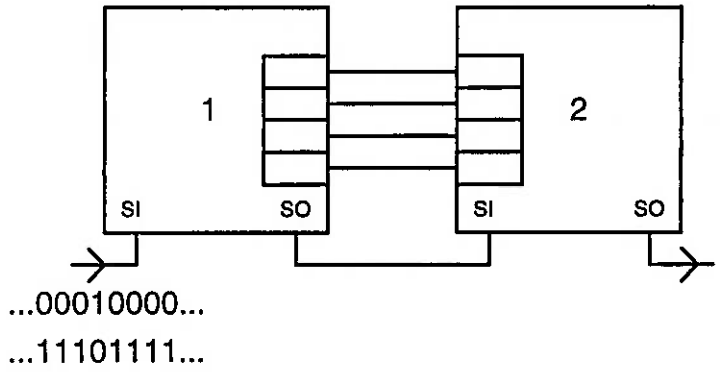
Figura 3.4: Dois exemplos de rede com falha de circuito aberto. a) Uma transição em qualquer saída pode detectar o circuito aberto antes da entrada D. b) Apenas a transição da saída C permite detectar a falha existente logo após essa saída.

O esquema de teste proposto permite que falhas sejam detectadas em uma rede de conexão independentemente da condição das outras redes, falhas ou não. A única limitação se verifica quanto à possibilidade de mais de um tipo de falha estar presente em uma mesma rede.

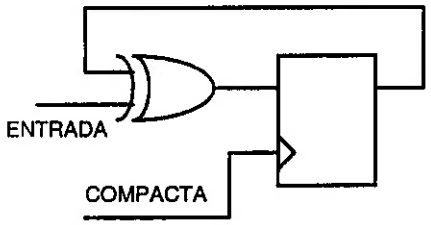
Vetores Utilizados

O teste de curtos e *stuck-at*'s será realizado através da aplicação de uma seqüência *walking*-"1" e uma seqüência *walking*-"0". Para isso inicialmente é introduzido nos registradores de *boundary-scan* um vetor composto apenas de zeros terminando com um único *bit* em um para a seqüência *walking*-"1" e inversamente um vetor composto de uns terminado por um único zero no caso da outra seqüência (figura 3.5a). Tendo como base este vetor inicial, cada novo vetor da seqüência é obtido através de um deslocamento do registrador de *boundary-scan* de forma que este *bit* diferente passe a

ocupar as sucessivas posições. O resultado será compactado em cada célula de entrada através de um OU-exclusivo, que é adicionado à célula padrão descrita em [IEEE90] (figura 3.5b). A seguir é demonstrada a suficiência deste esquema para detectar curtos e *stuck-at*'s.



(a)



(b)

Figura 3.5: a) As seqüências *walking*-"0" e *walking*-"1" passam pelas células de saída do circuito 1 e excitam as entradas do circuito 2. b) Compactador de resultado presente nas células de entrada.

Primeiramente, vamos lembrar que no caso de uma rede de interconexões alimentada por duas ou mais saídas (tipo *tri-state*[*]) apenas uma deverá estar ativa. Nessa condição, dada uma seqüência *walking*-"1", cada uma das entradas será alimentada por uma única saída, e estará, estando perfeito o circuito, no valor lógico "1" apenas durante um dos ciclos de teste dentro desta seqüência. Isso implicará no uso de uma célula de controle de *tri-state* especial, que permita a carga de um valor inicial que não seja alterado posteriormente pelos deslocamentos de valores no registrador de *boundary-scan*. Em caso de falha na rede associada a uma célula de entrada, poderemos observar os seguintes resultados, conforme a falha:

- curto-E: Quando uma das redes em curto estiver em "1", a outra estará em "0" e o resultado global será "0". Portanto, somente serão lidos "0" nesta entrada.

[*] Saídas de tipo coletor aberto ou dreno aberto ligadas em comum são interpretadas por este esquema de teste estando em curto, o que em última análise resume a sua situação. Quando esta situação está presente, as falhas reais podem passar despercebidas e portanto este método não se aplica nestes casos.

- curto-OU: Quando uma das redes em curto estiver em "1", a outra estará em "0" e o resultado global será "1". Cada entrada conectada à estas redes em curto receberá mais vezes (tantas quantas forem as redes em curto) o valor "1" em vez de apenas uma.

- *stuck-at* 1: A leitura da entrada é "1" independentemente do valor da saída associada, portanto a seqüência obtida será inteiramente composta por "1"s.

- *stuck-at* 0: Inversamente à falha anterior, teremos uma seqüência inteiramente composta por "0"s na entrada analisada.

Esta seqüência é suficiente para detectar as falhas de curto e *stuck-at*, desde que as células de entrada sejam constantemente monitoradas. Como isto não é possível em um circuito de teste como o proposto, estamos recorrendo a um esquema de compactação de forma a obter um resultado final que será então lido através do registrador de *boundary-scan*. Esta compactação é bastante simples e se resume a realizar a operação lógica OU-exclusivo entre todos os *bits* recebidos em cada entrada ao longo da aplicação das seqüências propostas (figura 3.5b). Com isso, ao final de cada seqüência teremos como resultado em uma determinada célula de entrada o valor "1" se esta tiver recebido um número ímpar de níveis lógicos "1" e valor "0" caso este número seja par. Considerando que o número total de células de *boundary-scan* seja *R*, o resultado da compactação será como apresentado na tabela 3.1 (o operador "mod", utilizado nesta tabela, é a operação módulo, ou seja o resto da divisão, no caso, de *R* por 2).

Tabela 3.1: Resultados compactados para uma seqüência *walking*-"1"

Condição da rede	**Resultado compactado**
Sem falha	1
curto-E	0
curto-OU entre n° par de redes	0
curto-OU entre n° ímpar de redes	1
stuck-at 1	*R* mod 2
stuck-at 0	0

Se garantirmos que R é par, através de uma eventual adição de uma célula extra caso tenhamos um número ímpar originariamente, teremos apenas em um caso de falha um resultado igual ao obtido para um circuito sem falhas.

Vamos agora verificar quais seriam os resultados no caso da aplicação de uma seqüência *walking*-"0".

Aplicando uma análise semelhante à realizada para a seqüência *walking*-"1", teremos o seguinte comportamento em cada uma das entradas conforme o tipo de falha na rede a que ela está conectada:

- curto-E: Haverá um número de valores "0" equivalente ao número de redes em curto.
- curto-OU: Serão recebidos apenas valores "1".
- *stuck-at* 1: Serão recebidos apenas valores "1".
- *stuck-at* 0: Serão recebidos apenas valores "0".

O resultado da compactação será como apresentado na tabela 3.2, agora já considerando *R* par.

Tabela 3.2: Resultados compactados para uma seqüência *walking*-"0"

Condição da rede	**Resultado compactado**
Sem falha	1
curto-E entre n° par de redes	0
curto-E entre n° ímpar de redes	1
curto-OU	0
stuck-at 1	0
stuck-at 0	0

Como no caso da outra seqüência, apenas um tipo de falha produz o mesmo resultado que o circuito perfeito, porém neste caso é um tipo diferente de falha. Combinando portanto os dois resultados poderemos detectar a existência de qualquer falha de curto ou *stuck-at* já que somente um circuito sem defeito apresentará o resultado compactado "1" para a aplicação das duas seqüências.

A leitura do registrador de *boundary-scan* contendo os resultados compactados permite não só a detecção como também o diagnóstico da falha e de sua posição. Contudo, se o objetivo for apenas a determinação da existência da falha, pode ser feita uma compactação do resultado de todas as células de entrada através da contagem de "1"s.

O número total de valores "1" contido no registrador de *boundary-scan* após a aplicação das seqüências *walking*-"0" e *walking*-"1" é a soma do número de "1"s contidos nas células de entrada, de saída e de controle. Em um circuito sem falhas todas as células de entrada deverão conter o valor "1" após a aplicação das duas seqüências. O número de saídas e controles que contém "1" depende da última célula do registrador de *boundary-scan*. Se esta célula for de saída ou de controle conterá "1" para uma seqüência *walking*-"1" e "0" para uma seqüência *walking*-"0", se for de entrada, ela conterá um resultado compactado. Sendo *S* o número total de saídas e controles, o número de saídas *N* que contém "1" em cada caso será como indicado nas equações 3.1 e 3.2. Nestas equações, o primeiro termo representa o número de valores "1" nas células de saída e de controle após uma seqüência *walking*-"1" e o segundo termo após uma seqüência *walking*-"0". Em ambos os casos notamos que a soma dá o mesmo número, *N*.

A última célula é de entrada:

$$N = 0 + S \qquad (3.1)$$

A última célula é de saída ou controle:

$$N = 1 + (S\text{-}1) \qquad (3.2)$$

A contagem final para um circuito perfeito será portanto a soma de N mais duas vezes o número de entradas, já que todas devem conter "1" após a aplicação de cada uma das seqüências. Qualquer falha reduzirá este total, podendo portanto ser detectada.

Detecção de circuitos abertos

Como visto nos modelos de falhas, a detecção de abertos envolve a aplicação de uma transição em uma célula de saída e o acompanhamento dos seus efeitos nas células de entrada. No caso de redes com uma só saída, a aplicação das duas seqüências descritas no item sobre modelos de falhas atende a este critério e portanto é suficiente para determinar a existência deste tipo de falha. No entanto, uma das limitações ao teste proposto é a de que quando uma rede é alimentada por duas ou mais saídas apenas uma esteja ativa. Isso impede que pontos abertos isolando uma saída que não esteja ativa sejam detectados. A solução para este caso é repetir o teste tantas vezes quanto for o número máximo de saídas interconectadas dentro do MCM, ativando em cada uma dessas vezes uma saída diferente.

Teste dos Circuitos do Módulo *Multichip*

Embora um módulo *multichip* possa conter uma quantidade de lógica muitas vezes maior do que a possível em um único circuito integrado, há um fator que contribui para facilitar o seu teste, que é a partição natural imposta pela divisão em circuitos em seu interior. Sendo assim, pode-se considerar o seu teste como sendo uma generalização do problema de teste de circuitos integrados.

Por outro lado, mesmo estando pré-particionado, o maior número de circuitos em teste ao mesmo tempo exige uma capacidade maior do testador utilizado. É fundamental portanto a utilização de técnicas de teste eficientes. O uso de auto-teste para módulos *multichip* é uma opção bastante atrativa, pois permite que todos os circuitos estejam em teste simultaneamente e que dessa forma o tempo total de teste seja pouco maior que o tempo gasto pelo auto-teste do componente que demore mais para realizá-lo.

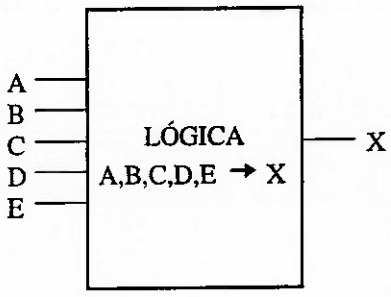
Diversas formas de implementação de auto-teste propostas na literatura se baseiam no teste pseudo-exaustivo. A vantagem deste método se apresenta quando as saídas de um circuito dependem de um número pequeno, proporcionalmente ao número total, de entradas. Em termos de número de vetores totais necessários para a realização do auto-

teste (e conseqüentemente do tempo consumido) o teste pseudo-exaustivo, dependendo de como é implementado apresenta bons resultados comparados a outras alternativas, podendo ser a melhor entre todas conforme o circuito [Wang85]. Sob outro aspecto, se considerarmos a implementação de um circuito de auto-teste, aparecerá uma outra grande vantagem, que é a possibilidade de uma implementação sistemática dos testes pseudo-exaustivos. Além disso, este tipo de circuito não necessita que se carregue um grande número de vetores iniciais, ou "sementes", para que atinjam a cobertura desejada.

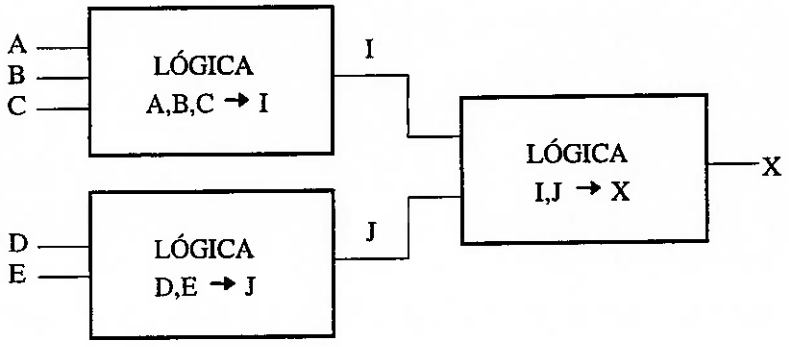
Teste Pseudo-Exaustivo

O teste pseudo-exaustivo consiste em aplicar todas as combinações possíveis em todos os conjuntos de entradas que influenciam cada uma das saídas. Em outras palavras, para cada uma das saídas se verifica as entradas das quais ela depende e se garante a aplicação de todas as combinações a este grupo de entradas. Há diversas formas para identificar esse grupo de entradas, variando em complexidade e eficiência, de forma que não abordaremos este problema neste trabalho [Amaz88].

Há casos em que o número de entradas com influência sobre uma das saídas ainda é grande o suficiente para tornar o teste mais longo do que o desejável, lembrando que a duração do teste cresce exponencialmente com o tamanho do conjunto de entradas. Quando isto acontece, é possível a adoção de uma segmentação do circuito. Isto é feito escolhendo pontos internos adequados do circuito dentro da estrutura lógica associada a esta saída de tal forma que se tornam eles mesmos saídas referentes ao grupo original de entradas, e entradas referentes à saída original. Com esta segmentação em grupos menores, o tamanho do teste se reduzirá bastante. Isto está exemplificado na figura 3.6. A técnica de segmentação é extensivamente discutida em [Amaz88].



(a)

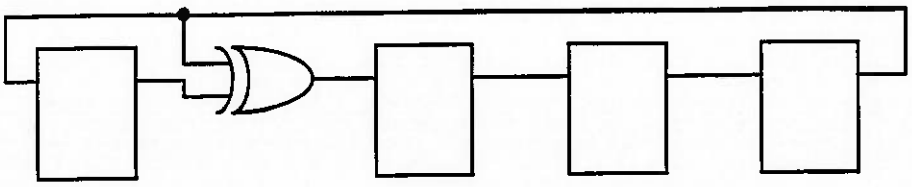


(b)

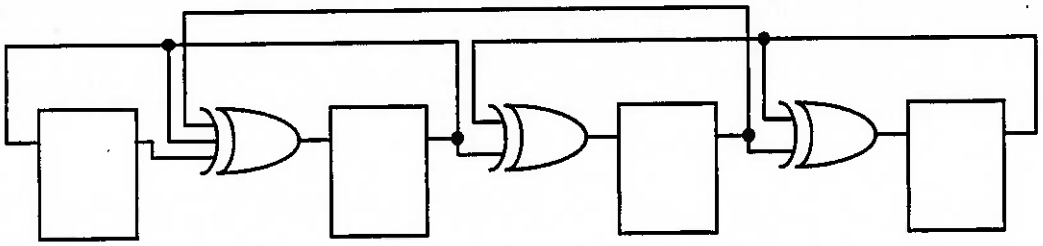
Figura 3.6: a) Para o teste exaustivo desta lógica seriam necessários *a priori* 2^5, ou seja, 32 vetores de teste. b) Neste segundo caso, apenas 8 (2^3) vetores de teste seriam necessários, desde que os pontos I e J pertençam à linha de *scan*.

Os circuitos geradores de testes pseudo-exaustivos se baseiam comumente em dois tipos de estruturas, denominadas *Linear Feedback Shift Register* (LFSR) e *Cellular Automata Register* (CAR). O uso dessas estruturas é devido a certas propriedades que elas apresentam e que permitem a geração de seqüências pseudo-aleatórias baseadas em códigos lineares binários. Esses códigos [Lin83] têm origem em um polinômio gerador, que podem ser implementados através do LFSR e do CAR. Na figura 3.7 são apresentados exemplos de duas implementações de um mesmo polinômio gerador, $x^4 + x + 1$.

Podemos notar que o CAR apresentou uma implementação mais complexa do que a do LFSR, porém todas as suas ligações são feitas com os registradores vizinhos, simplificando o roteamento. Em termos de freqüência de operação, o CAR tem a vantagem de ter um OU-exclusivo de no máximo três entradas realimentando um registrador, seja qual for a complexidade do polinômio implementado. O LFSR tem apenas uma porta OU-exclusivo para realimentação, com um número de entradas que pode ser bastante alto para polinômios complexos, provocando maiores atrasos de propagação e conseqüentemente menor freqüência de operação.



(a)



(b)

Figura 3.7: a) exemplo de um LFSR b) exemplo de um CAR

O polinômio que os circuitos da figura 3.7 implementam, chamado de polinômio característico é um polinômio primitivo, e portanto é possível usá-lo para gerar uma seqüência de tamanho 2^n - 1, onde n é o grau do polinômio, neste caso, 4 [Lin83]. No exemplo, se tivermos um valor inicial {1000} nos registradores da figura 3.7, poderemos gerar as seqüências apresentadas na tabela 3.3.

Tabela 3.3: Seqüências geradas pelos circuitos da figura 3.7 a partir de um valor inicial {1000}.

LFSR	CAR
1 0 0 0	1 0 0 0
0 1 0 0	0 1 0 0
0 0 1 0	1 1 1 0
0 0 0 1	1 1 1 1
1 1 0 0	1 1 0 0
0 1 1 0	1 0 1 0
0 0 1 1	0 0 0 1
1 1 0 1	0 0 1 1
1 0 1 0	0 1 1 0
0 1 0 1	1 0 1 1
1 1 1 0	0 0 1 0
0 1 1 1	0 1 0 1
1 1 1 1	1 1 0 1
1 0 1 1	1 0 0 1
1 0 0 1	0 1 1 1

Podemos notar que embora ambos os circuitos tenham o mesmo polinômio característico, eles geram seqüências diferentes, mas com o mesmo tamanho. É demonstrado em [Hort89] que o CAR apresenta melhores resultados em termos de aleatoriedade da seqüência gerada.

A determinação do polinômio característico nos dois casos apresentados é feito da mesma forma, que é através da matriz de transição de estado dos circuitos. Essa matriz (chamada de matriz característica) é construída de tal forma que o produto dela pelo vetor contido nos registradores será o próximo valor da seqüência. As operações de soma e multiplicação são aquelas definidas em [Lin83], sendo que a multiplicação opera como a multiplicação algébrica e a soma corresponde à operação OU-exclusivo.

Podemos verificar que as matrizes características do LFSR e do CAR apresentados são respectivamente:

$$M_{LFSR}= \begin{bmatrix} 0001 \\ 1001 \\ 0100 \\ 0010 \end{bmatrix} \qquad \text{e} \qquad M_{CAR}= \begin{bmatrix} 0100 \\ 1110 \\ 0101 \\ 0011 \end{bmatrix}$$

A partir dessas matrizes, é possível obter o polinômio característico através do cálculo do seguinte determinante:

$$p(x) = \det (xI - M) \quad (3.3)$$

onde $p(x)$ é o polinômio característico, I é a matriz identidade e M é a matriz característica.

Conforme apresentado, há um polinômio característico que corresponde a cada circuito, porém é possível existir mais de um circuito que correspondente a um dado polinômio.

Definida uma estrutura para o LFSR (há mais de uma possibilidade) [Lin83], dado o polinômio característico, é possível obter o circuito correspondente.

Seja um polinômio genérico definido por:

$$p(x) = x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_1x + a_0 \quad (3.4)$$

Considerando a estrutura da figura 3.7a, onde temos a saída do último elemento registrador realimentado através de um OU-exclusivo à entrada dos elementos anteriores, a existência ou não da realimentação em um determinado elemento *i* fica definida pela seguinte regra:

Se $a_i = 1$ há realimentação
Se $a_i = 0$ não há realimentação

Sendo o elemento que corresponde à i=0 o primeiro da linha de deslocamento, ou seja o que está à esquerda na figura 3.7a. Pode-se demonstrar que este esquema leva à

construção de um LFSR correspondente ao polinômio desejado pelo cálculo do polinômio característico para um circuito genérico construído segundo este tipo de estrutura. Para tal circuito, a matriz de transição é:

$$M = \left[\begin{array}{cccc:cc:c} 0 & 0 & 0 & 0 & 0 & 0 & a_0 \\ 1 & 0 & 0 & 0 & 0 & 0 & a_1 \\ 0 & 1 & 0 & 0 & 0 & 0 & a_2 \\ 0 & 0 & 1 & 0 & 0 & 0 & a_3 \\ 0 & 0 & 0 & 1 & 0 & 0 & a_4 \\ \hdashline 0 & 0 & 0 & 0 & 1 & 0 & a_{n-2} \\ 0 & 0 & 0 & 0 & 0 & 1 & a_{n-1} \end{array}\right]$$

Para esta matriz, o polinômio característico, dado por det(xI-M), será como a equação 3.4.

Embora seja simples, este esquema apresenta alguns problemas quando o tamanho do LFSR começa a crescer. Como a realimentação parte de um único elemento, ela fica sujeita à uma carga elevada não só pelo grande número de portas conectadas como também pela capacitância parasitária devida ao roteamento. Devido a esta carga, será necessário o uso de *buffers*, o que aumenta o atraso de propagação e diminui a freqüência máxima de operação. Além disso, o roteamento em si deste sinal é complicado e pode significar aumento de área do circuito integrado.

O *Cellular Automata Register*

O circuito apresentado na figura 3.7b é chamado de *Cellular Automata* unidimensional, pois o próximo estado de um elemento é determinado pelo seu estado atual e por dois vizinhos, no caso os dois mais próximos na linha de deslocamento. Uma característica marcante desta configuração é que as realimentações são sempre locais e limitadas, exigindo pouco esforço de roteamento e pequena capacidade de corrente dos registradores e portas lógicas. Nessas condições pode-se realizar um projeto baseado em algumas células pré-definidas e também obter uma maior velocidade de operação do teste.

Em relação ao LFSR, a grande desvantagem do CAR é não possibilitar sua construção imediata a partir de um determinado polinômio. Entretanto são apresentados em [Serr90] e [SerS90] algoritmos que permitem a obtenção de um CAR a partir de um dado polinômio primitivo. Em [Serr89] é apresentada uma tabela com resultados já calculados.

É comum a descrição de um CAR através da regra que define o próximo estado de um elemento do registrador. O nome das possíveis regras para um CAR unidimensional deriva da tabela da verdade para o próximo estado. Sendo o próximo estado uma função do vizinho anterior, do próprio elemento e do vizinho seguinte, temos 2^3=8 combinações de entrada possíveis. Para cada uma dessas combinações, o próximo

estado pode assumir 1 ou 0, ou seja, teremos 2^8=256 possíveis funções ou regras. O nome da regra é então inspirado no número formado pela seqüência de 1's e 0's correspondentes às saídas para cada uma das combinações de entrada. Por exemplo, a regra 150 é formada da seguinte forma:

Entradas:	111	110	101	100	011	010	001	000
Próximo estado:	1	0	0	1	0	1	1	0

O número binário resultante dos valores para o próximo estado é portanto 10010110, que corresponde à 150 em decimal. A implementação desta função é feita através de um OU-exclusivo entre as saídas do próprio elemento e dos seus dois vizinhos.

Pode ser demonstrado [Serr90] que somente um CAR construído com as regras 90 e 150 leva a um polinômio característico irredutível. Portanto, iremos nos concentrar apenas nestas duas regras. Nessas condições, teremos uma matriz de transição de estado tridiagonal, já que as funções que implementam essas regras são:

regra 90: $s_n(t+1)=s_{n-1}(t)\oplus s_{n+1}(t)$ (3.5)

regra 150: $s_n(t+1)=s_{n-1}(t)\oplus s_n(t)\oplus s_{n+1}(t)$ (3.6)

onde $s_n(t)$ representa o estado do n-ésimo elemento no instante t.

A matriz de transferência do circuito da figura 3.7b, que obedece esta limitação, é um exemplo de matriz nessas condições.

Esta limitação, longe de prejudicar a implementação de tal circuito, é muito conveniente por levar a uma estrutura bastante regular. Como podemos notar pelas equações 3.5 e 3.6, a diferença entre as regras 90 e 150 é apenas a realimentação do próprio elemento. Se quisermos, pois, construir um CAR que permita a geração de uma seqüência pseudo-aleatória de um comprimento qualquer, basta construir células que implementem a regra 150, porém com a realimentação do próprio elemento programável. Dessa forma, se eliminarmos essa realimentação, obteremos a regra 90. De posse do conjunto de regras necessárias para a geração da seqüência pseudo-aleatória desejada, fica fácil a programação do CAR de acordo com o requerido. Tal célula básica teria a forma apresentada na figura 3.8.

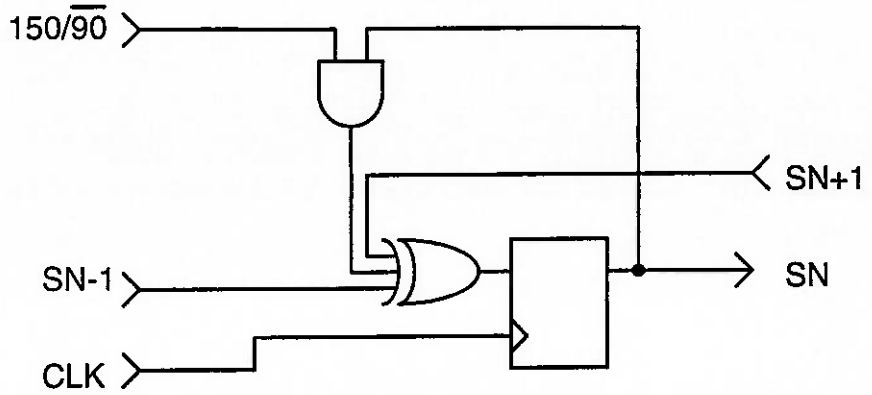


Figura 3.8: Célula para a implementação de um CAR com programação entre as regras 90 e 150.

3. IMPLEMENTAÇÃO LÓGICA DE CÉLULAS DE *SCAN* PARA O AUTO-TESTE

Com base na estratégia de teste apresentada, é possível acrescentar lógica a uma célula de *scan* de forma que ela permita realizar as funções necessárias para os tipos de testes propostos.

Quanto ao auto-teste da lógica, este já é previsto pelo padrão IEEE 1149.1, sendo que este trabalho apenas sugere uma forma de implementá-lo segundo as regras deste padrão. O teste das interconexões, ao contrário, não é previsto, pelo menos do ponto de vista de cada componente que compõe um MCM, e que é considerado como tendo o seu próprio *boundary-scan*. É interessante notar que, considerando o MCM como um único componente, o teste das interconexões é simplesmente uma etapa de um auto-teste, e que este mesmo princípio não poderia ser aplicado para um circuito integrado isolado. Isto porque, ainda que em um circuito haja uma partição bem definida e que tenha sido construída um linha de *scan* entre essas partições, não há uma ligação entre um registrador de saída de um bloco e um registrador de entrada para outro bloco, pois tais registradores são um mesmo registrador intermediário entre duas porções de lógica, como ilustrado na figura 3.9. O teste das interconexões internas a um circuito integrado está contido no auto-teste da lógica, que se for abrangente é capaz de detectar todas as falhas de circuito aberto, curtos e *stuck-at*. Portanto, o que será proposto se aplica especificamente a um MCM e pode ser interpretado apenas como mais uma etapa de auto-teste, seguindo portanto, as regras a que este está sujeito. Por outro lado, este conceito não se aplica também a uma placa de circuito impresso, já que esta, diferentemente de um MCM não dispõe ela própria de um *boundary-scan*. Neste último caso, porém, não seria difícil acrescentar esta funcionalidade através de um circuito integrado extra que controlasse todos os registradores de *boundary-scan* dos demais.

Para isso seria necessário apenas que todos os circuitos integrados dispusessem da funcionalidade para teste de interconexões a ser descrita adiante, tal como esta funcionalidade também é necessária para todos os circuitos que componham um MCM.

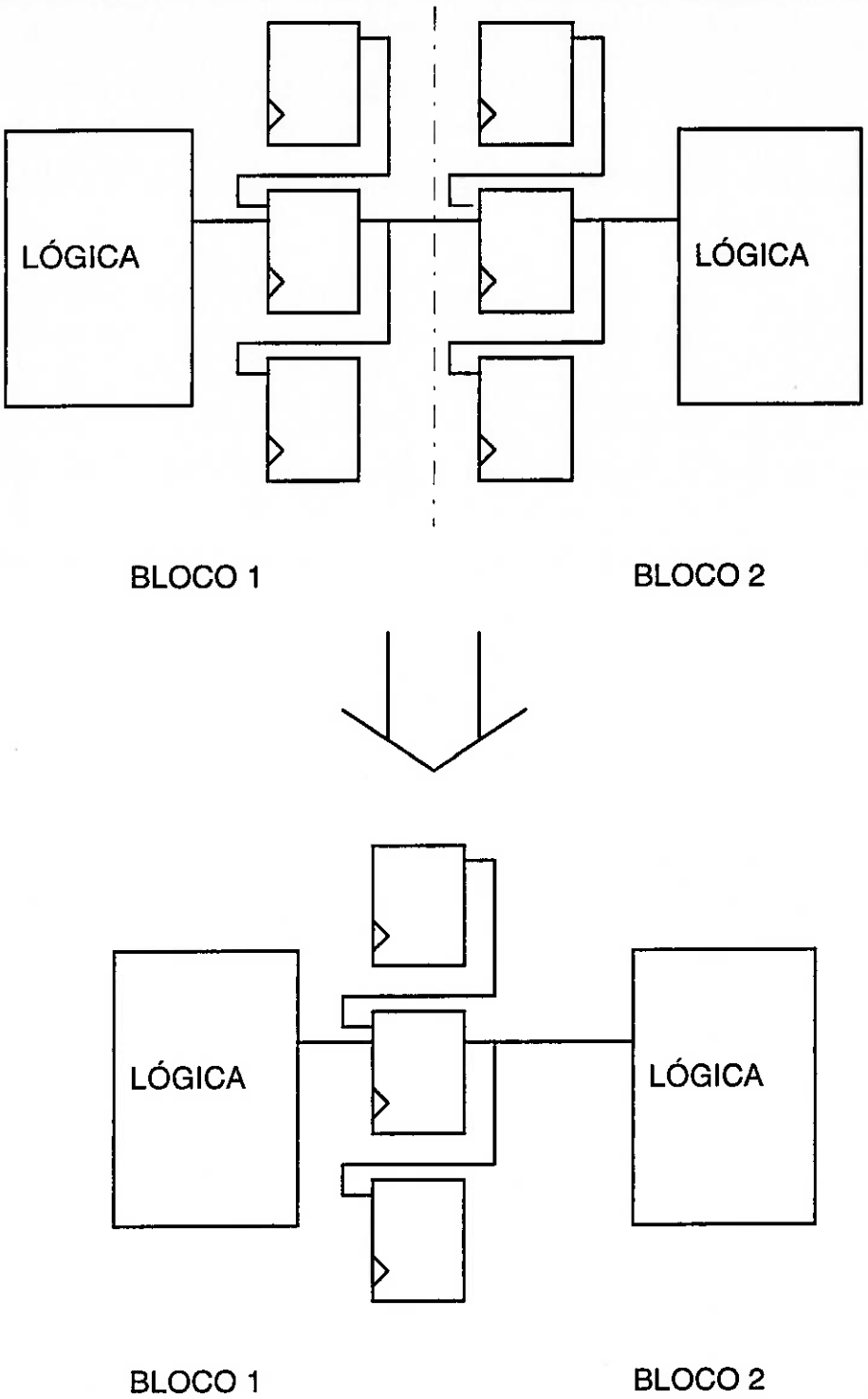


Figura 3.9: A separação entre blocos encontrada entre dois circuitos integrados com *boundary-scan* torna-se mais simples quando estes dois blocos são integrados em um mesmo circuito, tornando desnecessário o teste de interconexões.

Lógica Para o Teste das Interconexões

Como foi colocado no item 2 deste capítulo, o teste das interconexões é feito com seqüências *walking*-"0" e *walking*-"1" e compactação dos resultados. As seqüências podem ser geradas tanto por um controlador externo como por um circuito dedicado a esta tarefa. Este circuito pode ser um único circuito externo comum a todos os circuitos agrupados no MCM ou integrado em cada um deles. Essas possibilidades serão analisadas adiante. Já a compactação deve ser feita idealmente pelas células de entrada do *boundary-scan*, já que são elas que recebem o resultado imediato dos vetores de teste da interconexão. A funcionalidade a ser adicionada se resume à capacidade de executar a operação OU-exclusivo de forma cumulativa, o que pode ser feito por uma porta lógica que execute tal operação associada a um registrador que vá acumulando o resultado. Um circuito simples para a execução desta operação é apresentado na figura 3.10, integrado a uma célula de *boundary-scan* de entrada. Nesta célula o sinal ClockDR tem a função de comandar a captura do valor do pino de entrada no estado <Capture DR>, quando o controle 1 deverá estar selecionando este valor e a função de comandar o deslocamento do valor de ShiftIn para ShiftOut, com o controle 1 fazendo a seleção adequada. O sinal UpdateDR faz a operação de atualização do estado <Update DR> e o controle 2 torna a célula "transparente", por exemplo, durante uma instrução BYPASS.

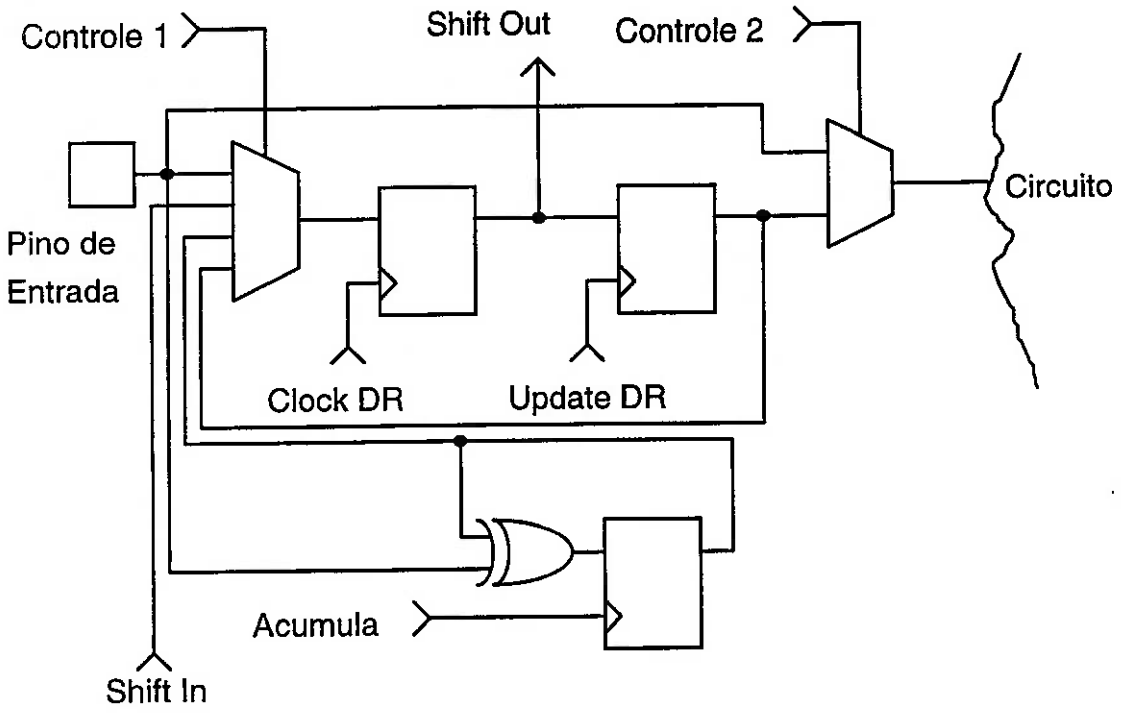


Figura 3.10: Célula de entrada para *boundary-scan* com lógica adicional para teste de interconexões [Hass92].

Em relação à célula de *scan* apresentada como exemplo em [IEEE90], percebemos apenas duas diferenças, que são o compactador de resultado que aparece na parte inferior da figura 3.10 na forma de uma porta OU-exclusivo e de um registrador e um

multiplex com quatro entradas em vez de duas. Uma dessas entradas extras tem a finalidade de permitir a leitura do resultado do teste através da linha de *scan*, e a outra permite que, dada a passagem pelo estado <UpdateDR>, necessário após um <Shift DR> para aplicar o vetor nas células de saída, a seqüência não seja destruída durante o estado <Capture DR>. Uma outra opção seria alterar o sinal Clock DR da figura 3.10 de forma que ele não agisse durante o teste das interconexões no estado <Capture DR>.

A célula de saída básica de um *boundary-scan* requer ainda menos alterações, já que a ela cabe apenas aplicar os vetores. Porém, como a célula de entrada, ela está sujeita a provocar a destruição da seqüência em uso durante o estado <Capture DR> e portanto deve contar com a realimentação extra apresentada na figura 3.11 na forma de mais uma entrada no *multiplex* da esquerda ou com um sinal Clock DR projetado de forma a não permitir a destruição da seqüência aplicada.

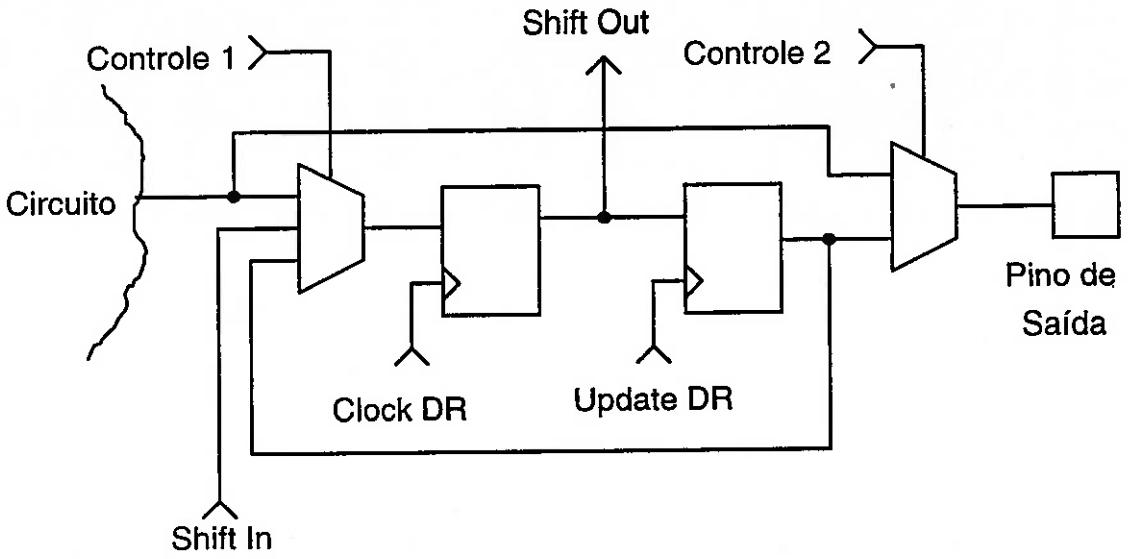


Figura 3.11: Célula de saída para boundary-scan com lógica adicional para teste de interconexões [Hass92].

Resta ainda o caso das células de controle. Como foi visto no item 2, elas devem manter seu estado durante toda a aplicação da seqüência. Esse comportamento pode ser conseguido adicionando um registrador extra para conservar este valor desejado e realizando a carga deste registrador através de uma instrução especial para este fim, a ser executada logo após a carga da seqüência de controle das células de E/S e antes do início do teste propriamente dito. Neste momento, o sinal de controle desejado, que é fornecido através do deslocamento da seqüência dentro do *boundary-scan*, é carregado no registrador extra para este fim e durante o teste é aplicado através do *multiplex* mostrado à direita na figura 3.12, controlado adequadamente. Adicionalmente pode-se notar que também neste caso é necessária a realimentação do registrador controlado pelo sinal Update DR de forma a manter intacta a seqüência de teste.

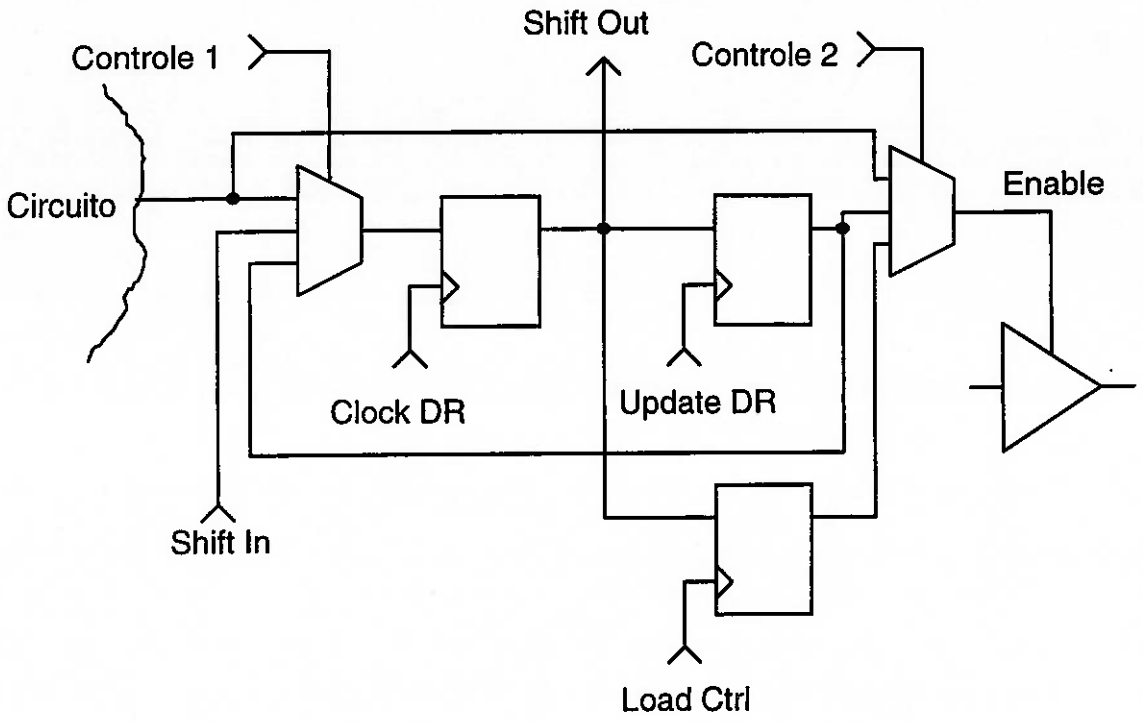


Figura 3.12: Célula de controle de *tri-state* para *boundary-scan* com lógica adicional para teste de interconexões.

As células apresentadas nas figuras 3.10, 3.11 e 3.12 não são a única solução para implementar o teste das interconexões, porém ilustram adequadamente o conceito do teste proposto implementado através de um *boundary-scan*.

Lógica Para o Teste Interno dos Circuitos

Na figura 3.8 foi apresentada uma célula básica para a construção de um CAR empregando as regras 90 e 150. A integração de um CAR para a realização do teste interno dos circuitos consiste na adaptação da estrutura desta célula à estrutura das células de entrada do *boundary-scan*. Através desta nova célula seria então possível aplicar um conjunto exaustivo de vetores (a menos do vetor de "0"s) a qualquer conjunto de entradas. Tal célula é apresentada nas figuras A.3 e A.4.

A realização do teste exige que inicialmente sejam definidos os grupos de entradas em que serão realizados os testes pseudo-exaustivos, o que é feito através da partição do circuito. Para cada um desses conjuntos de entrada é determinada a programação do CAR em termos da seqüência de regras 90 e 150 que deve ser utilizada. Essa seqüência pode ser obtida diretamente através da tabela apresentada em [Serr89] ou calculada com o auxílio dos algoritmos apresentados em [Serr90] e [SerS90]. Com esses dados é definido um vetor de programação do CAR para cada conjunto de entradas, obtido por:

$$P[2i\text{-}1] = \begin{cases} 0 & \text{se a regra a ser aplicada para o } \textit{bit } i \text{ é a 90.} \\ 1 & \text{se a regra a ser aplicada para o } \textit{bit } i \text{ é a 150.} \end{cases} \qquad (3.7)$$

$$P[2i] = \begin{cases} 0 & \text{se a entrada } i \text{ não faz parte do conjunto} \\ 1 & \text{se a entrada } i \text{ faz parte do conjunto} \end{cases} \quad (3.8)$$

onde P é um vetor de tamanho igual ao dobro do número de entradas do circuito e i varia de 1 até o número de entradas do circuito.

Os vetores assim obtidos deverão ser armazenados em uma memória, que pode ser interna ao MCM, ligada a um circuito responsável por sua leitura e introdução dos vetores na linha de *scan* ou interna a cada um dos circuitos a serem testados. Como a aplicação deste teste exige que cada circuito onde ele será utilizado tenha sido projetado para isto, é possível incluir a especificação de uma memória interna a cada circuito para conter os vetores necessários, tornando-o independente de estar ou não montado em um MCM e portanto esta será a linha adotada.

Controlador para o auto-teste

Como foi descrito, o auto-teste envolve duas etapas, o teste de interconexão e o teste da lógica dos circuitos. Se estivéssemos tratando de uma placa de circuito impresso, tudo o que teríamos a fazer era introduzir as seqüências para o teste das interconexões e depois comandar uma auto-teste em todos os circuitos simultaneamente. Porém, como se trata de um MCM, que pode ser considerado como um único componente, temos a alternativa de implementar um circuito que comande esse auto-teste. Este circuito pode tanto ser um a mais dentro do módulo como pode estar integrado a um dos circuitos já existentes. Se adotada a última opção deve-se ter em mente que esta é apenas uma forma de otimizar o sistema, devendo este módulo ficar totalmente isolado do restante do circuito, assim como o circuito responsável pela implementação do *boundary-scan*. Esse controlador será descrito e implementado no capítulo 4.

4. EXTENSÕES PARA O PADRÃO IEEE 1149.1

Existem basicamente duas opções para a implementação de um *boundary-scan* em um MCM. Uma delas considera o módulo como um só componente, portanto o *boundary-scan* deve percorrer todas as entradas e saídas do módulo e pode não incluir as entradas e saídas utilizadas apenas para as interconexões sem se ligar a nenhum pino externo. Tal implementação está plenamente de acordo com o padrão IEEE 1149.1. A outra opção considera o módulo como um subsistema e portanto cada entrada e cada saída de todos os circuitos deve ser incluída na linha de *scan*. Este caso em nada diferiria de uma placa de circuito impresso que estaria, por exemplo, sendo ligada a um sistema através de um conector, e neste trabalho nossa intenção é ir além deste enfoque. Nossa abordagem será portanto a de considerar o MCM como um componente e interagir com ele como tal. Assim, a implementação do *boundary-scan* se restringirá apenas aos pinos de interface com o exterior do módulo. Contudo, dada a complexidade verificada em muitos MCMs,

é desejável também um tipo de acesso que considere cada um dos componentes internos como um componente do sistema total, e portanto permita tratá-los dessa forma. Já que é prevista uma linha de *scan* percorrendo todas as entradas e saídas dos circuitos (necessária para o auto-teste) esta linha de *scan* já está disponível e será apresentada uma extensão ao padrão IEEE 1149.1 que possibilite o tratamento do MCM tanto como um conjunto de componentes ou como um componente único. A forma de implementação é objeto do próximo capítulo.

Componente X Subsistema

Após a inicialização do MCM ou do circuito de teste, todo o módulo deverá ser visto como um único componente. Nesta situação, todas as funções definidas em [IEEE90] estão disponíveis em relação aos pinos do módulo, bem como opcionalmente a função de auto-teste.

Através da programação de uma instrução definida como EXPLODE, todo o módulo passa a ser visto, do ponto de vista de teste, como a composição de todos os componentes que o compõem. A volta ao estado anterior implica no uso de outra instrução, SYSTEST.

De acordo com o definido em [IEEE90], as instruções EXPLODE e SYSTEST não satisfazem às especificações do item 7.1, no mínimo quanto à regra (c), e possivelmente também a regra (a)*, já que durante a vigência destas instruções, mais de um registrador de dados pode estar conectado entre TDI e TDO, e a função desses registradores pode variar conforme as instruções subsequentes que venham a ser programadas. Por outro lado, em [IEEE90] não há nenhuma previsão do uso de uma instrução que tenha por função modificar o comportamento da lógica de teste, ou seja, a forma com que as outras instruções atuam. Se considerarmos esse par de instruções colocando a lógica de teste em uma situação como a definida pela instrução BYPASS, e considerando que após o uso de qualquer das instruções temos um MCM que responde ao mundo exterior, ou como subsistema ou como componente, de forma plenamente compatível com o definido em [IEEE90], podemos dizer que não há conflito com as especificações citadas acima.

Programação de SYSTEST e EXPLODE

O registrador de instrução do MCM é na realidade a soma do registrador de instrução do *TEST ACCESS PORT* (TAP) do módulo e dos registradores de instrução dos TAPs de todos os componentes nele incluídos. Quando uma das instruções, SYSTEST ou EXPLODE, é programada, o efeito será de uma alteração da configuração dos componentes e do TAP do MCM. Essa configuração atuará até que a outra instrução deste par seja carregada. Todas essas mudanças ficam praticamente transparentes a um controlador externo de teste, cabendo ressaltar apenas que haverá uma alteração substancial no tamanho da cadeia de registradores de *boundary-scan*. Considerando

* Essas regras definem que cada instrução habilite apenas um registrador de dados entre os pinos TDI e TDO e defina completamente o conjunto de registradores em operação que podem interferir no teste.

apenas o módulo, em um caso o registrador de *boundary-scan* corresponde aos pinos de comunicação com o circuito exterior, e no outro caso, ao encadeamento de todos os pinos dos componentes internos.

Capítulo 4

Implementação do *Boundary-Scan*

A implementação do *boundary-scan* para um MCM será feita na forma de um exemplo que não é de forma alguma a única maneira de traduzir na prática os resultados deste trabalho. A implementação sugerida será baseada nas células de *scan* apresentadas no capítulo 3, complementando-se com unidades de controle tanto para o módulo como um todo como para os circuitos que o compõem. Além disso será apresentada a forma pela qual se pode ligar as células de *scan* entre si e com a unidade de controle.

O projeto dos módulos de controle será feito através de sua descrição funcional em linguagem Verilog e posterior síntese.

Composição do MCM Exemplo

O circuito utilizado nesta exemplificação é formado por quatro componentes que formam o módulo propriamente dito, acrescido de um controlador de teste, denominado controle-mestre. Este controlador poderia estar, na prática, incorporado a um dos componentes do módulo, conforme a possibilidade e conveniência. Na figura A.1 temos o circuito utilizado, sendo que o controlador pode incluir também a memória que é mostrada em separado.

Instruções

Neste exemplo são implementadas as instruções BYPASS, SAMPLE/PRELOAD, EXTEST, INTEST, RUNBIST, SYSTEST e EXPLODE. Das instruções definidas em [IEEE90], as opcionais IDCODE e USERCODE não foram implementadas, uma vez que sua implementação é muito parecida com a da instrução BYPASS, mudando apenas o registrador envolvido e o seu valor inicial. Cabe notar que essas últimas duas instruções, se implementadas, devem seguir a distinção entre componentes individuais e

sistema, tendo o MCM seus próprios códigos de identificação, independentes daqueles dos componentes internos.

O registrador de instrução é formado pela composição dos registradores de instrução dos circuitos internos e do registrador de instrução do controle-mestre. Os códigos das instruções derivam deste encadeamento. Os códigos de cada componente, do controle-mestre e os resultantes de sua junção são apresentados na tabela 4.1.

Tabela 4.1: Código das instruções

	Componentes	Controle-mestre	MCM
BYPASS	1111	111	1111111111111111
SAMPLE/PRELOAD	0010	*	0010001000100010000
EXTEST	0000	*	0000000000000000000
INTEST	0001	*	0001000100010001000
RUNBIST - COMPON.	0011	*	0011001100110011000
RUNBIST - SISTEMA	†	011	0011001100110011011
SYSTEST	0100	100	0100010001000100100
EXPLODE	0101	101	0101010101010101101

Na descrição dos módulos de controle serão feitas considerações sobre o modo de atuação dessas instruções quando em teste de sistema ou de componentes individuais.

Módulo de Controle de Teste do Sistema (Controle-Mestre)

O módulo "controle-mestre" é o responsável por toda a comunicação dos circuitos de teste contidos no MCM com o exterior. É através dele que os registradores "virtuais" formados por encadeamento são construídos e gerenciados. Apesar disso, se retirada a parte responsável pelo comando do auto-teste do sistema, trata-se de um circuito muito simples.

Quando o circuito é inicializado através do sinal TRSTB, o controle entra no estado BYPASS e modo de teste de sistema (SYSTEST). Nessa condição, um registrador de deslocamento de um estágio está conectado entre os pinos TDI e TDO.

Estando todos os componentes do módulo no estado de teste de sistema, o uso das instruções SAMPLE/PRELOAD, EXTEST e INTEST conectará aos pinos TDI e TDO o

* É indiferente o código usado nesta situação, desde que não seja o código de nenhuma outra função definida. Na composição final foi usado 000 para atender o código obrigatório para EXTEST, mas poderia ser também 001, 010 ou 110.

† Nesta implementação é indiferente o código utilizado. Na composição foi colocado 0011, uma vez que este é o estado final do registrador de instruções após o término do auto-teste. Sendo este o valor inicial, ele não estará alterado no final do auto-teste.

registrador de *scan* dos componentes do MCM, neste momento formado pelas células conectadas aos pinos do módulo.

Quando o modo de teste de sistema é comutado para teste dos componentes, a principal alteração neste módulo é que também a instrução BYPASS provoca a conexão da linha de *scan* dos componentes aos pinos TDI e TDO, agora formada pelo conjunto dos registradores de *bypass*. Além disso, o auto-teste do sistema fica desativado.

Durante a execução do auto-teste do sistema, que só ocorre quando SYSTEST foi ativado, ou após a inicialização do MCM, o controle-mestre comanda inicialmente um instrução EXPLODE para todos os componentes internos para possibilitar o teste das interconexões. Este teste é realizado como descrito no item 2 deste capítulo. Após este teste, é comandada a instrução SYSTEST e finalmente a instrução RUNBIST. O controlador aguarda então o término do auto-teste dos componentes e disponibiliza em um registrador o resultado do teste de conexões (que, para este exemplo, em condição de bom funcionamento deve ser igual a 108 ou 6C em hexadecimal*) e o encadeamento dos resultados dos auto-testes dos componentes internos.

Há ainda uma memória associada a este módulo. Ela contém a programação das células de controle de pinos de saída *tri-state* ou bidirecionais. Como visto no capítulo 3, esses pinos devem ser controlados de forma que dentro de um grupo de saídas ligadas entre si, somente uma esteja ativa. Assim, o número de repetições das seqüências "*walking*-"0" e *walking*-"1" deve ser igual ao número máximo de saídas diferentes ligadas a uma mesma rede, e a esse número corresponde a quantidade de posições de memória necessária. O tamanho da palavra deve ser igual ao número de células de controle existentes no MCM. É importante não confundir o número de células de controle com o número de pinos *tri-state* ou bidirecionais, já que uma única célula de controle pode comandar um *bus* inteiro. Caso essa especificação de memória resulte em um formato incômodo (por exemplo 5 X 34) e não disponível na tecnologia empregada, podemos utilizar o mesmo recurso que foi empregado para implementar a memória de programação do CAR, dentro dos módulos de controle de teste dos componentes (descritos a seguir). Neste caso, foram utilizadas palavras de 8 *bits* que eram então concatenadas à medida em que iam sendo deslocadas para o registrador de programação do CAR.

Módulos de Controle de Teste dos Componentes

Este módulo é baseado no controle básico descrito em [IEEE90]. Às instruções definidas neste padrão acrescentam-se SYSTEST, EXPLODE, CONTEST e SUMSHIFT/LOAD. As duas primeiras já tiveram sua função explicada para o módulo controle-mestre. As outras duas tornaram-se necessárias para a implementação do auto-teste das conexões. As funções dessas instruções são detalhadas a seguir.

---

* A forma de definir qual é o resultado correto é apresentada no ítem "Teste das Interconexões do Módulo Multichip", no capítulo 3. No circuito exemplo, as seqüências "*walking*-"0" e *walking*-"1" são aplicadas duas vezes, então o resultado será 2 x (n° de saídas e controles + 2 x n° de entradas).

SYSTEST: Coloca o componente em modo teste de sistema. Na prática, apenas as células ligadas a pinos do MCM ficam sendo parte do *boundary-scan*. As outras células passam a ficar transparentes, sendo desativado seu circuito de teste. No caso de componentes que não estejam conectados a pinos do MCM, o pino TDI passa a se conectar diretamente a TDO.

EXPLODE: Define o modo de teste dos componentes individuais. A lógica de teste se comporta de forma a permitir o isolamento do componente, o qual obedece agora individualmente as especificações do padrão IEEE 1149.1.

CONTEST: Destina-se à aplicação das seqüências *walking*-"0" e *walking*-"1". Durante esta instrução, quando o controlador se encontra no estado <Shift DR>, a cada ciclo do *clock* de teste a seqüência desloca-se, o novo valor é atualizado nas saídas e o resultado é compactado nas células de entrada.

SUMSHIFT/LOAD: Semelhante à instrução SHIFT/PRELOAD, com a diferença que no estado <Capture DR> o valor compactado através da porta OU-exclusivo contida em cada célula de entrada é carregado no registrador de *boundary-scan* para ser lido no estado <Shift DR>. Enquanto este valor é lido, uma nova programação é inserida, fornecendo os valores inicias das células de saída e a programação das células de controle.

Foram criados 4 desses módulos de controle, um para cada um dos componentes do MCM, já que, a título de exemplo, cada um deles foi definido com certas particularidades. Assim, o primeiro componente é o único que contém *flip-flops* internos integrando o CAR usado no auto-teste, o segundo não tem ligação nenhuma com os pinos do MCM, e o quarto não tem nenhuma célula de controle, como é o caso dos componentes 1 e 3. O componente 1, portanto, tem o módulo de controle completo, sendo que dos outros módulos foi retirada a lógica desnecessária, como será descrito. Para efeito de padronização, o módulo de controle completo pode ser usado em todos os componentes, se desejado.

Sinais de Controle para as Células de Entrada, Saída e Controle de *Tri-state*

O módulo de controle (figura A.11) gera os seguintes sinais para o controle das células de *scan*:

clksum: Através de uma borda de subida comanda o armazenamento do resultado compactado do teste de interconexão. Corresponde ao sinal COMPACTA da figura 3.5b. É ligado à todas as células de entrada que participam do teste de interconexão, ou seja, aquelas internas ao MCM.

clockDRin: Produz bordas de subida durante os estados <Shift DR> e, durante as instruções EXTEST, SAMPLE/PRELOAD e SUMSHIFT/LOAD, <Capture DR> (figura 3.2). É ligado somente às células de entrada, pois idealmente [IEEE90] somente estas devem fazer a captura do registrador de dados durante a instrução EXTEST.

clockDRout: Semelhante à clockDRin, com a diferença que é ligado às células de saída do componente, e portanto somente comanda a captura de dados durante as instruções INTEST, SAMPLE/PRELOAD e SUMSHIFT/LOAD [IEEE90].

hab_updateDR_inpad: Quando ativo permite a atualização, na borda de descida de TCK do estado <Update DR>, da saída das células de entrada ligadas à linha de *scan*. Seguindo recomendação do padrão IEEE 1149.1, atua somente nas instruções INTEST, SAMPLE/PRELOAD e SUMSHIFT/LOAD, ficando inativo durante a instrução EXTEST.

hab_updateDR_outpadb: Difere de hab_updateDR_inpad por estar ativo em EXTEST em vez de INTEST. Além disso, também fica ativo no estado <Shift DR> durante a instrução CONTEST, permitindo que o valor deslocado pela borda de subida de TCK seja atualizado na borda de descida seguinte. Desta forma, a aplicação das seqüências *walking*-"0" e *walking*-"*1*" pode ser feita apenas no estado <Shift DR>, sem necessidade de passar por <Update DR>, o que torna o teste muito mais rápido.

hab_updateDR_outextb: Tem a princípio as mesmas funções que hab_updateDR_outpadb, com a diferença que controla as células de saída ligadas a pinos do MCM, e que portanto não participam do teste de interconexões. Por estarem ligadas ao exterior, estas células não podem ter seu valor de saída alterado durante a execução da instrução RUNBIST [IEEE90] e assim este sinal não fica ativo nunca durante as instruções SUMSHIFT/LOAD e CONTEST (acionadas pelo controle-mestre durante um auto-teste).

inicio_car: Quando ativo libera o controlador de auto-teste do componente, que comanda a programação e ativa o CAR. Permanece ativo tantos ciclos de TCK quanto os previstos para a duração do auto-teste.

loadsum: Em combinação com clockDRin comanda a carga do resultado compactado do teste de interconexões no registrador de deslocamento para possibilitar sua leitura.

scanbypassb: Faz com que o sinal de entrada das células (de entrada, saída e controle de *tri-state*) passe direto para a saída, deixando estas células "transparentes". É conectado às células que tem comunicação com o exterior, que são as conectadas aos pinos do MCM. Fica ativo durante as instruções BYPASS e SAMPLE/PRELOAD.

scanbypassintb: Como scanbypassb, porém conectado às células que se comunicam com outros circuitos internos ao MCM. Está ativo durante as instruções de BYPASS e SAMPLE/PRELOAD, e durante o teste de sistema. Neste último caso, é desativado quando o controle-mestre comanda um auto-teste do componente.

shiftDR: Se ativo durante uma borda de subida de ClockDRin ou ClockDRout comanda o deslocamento do dado contido no registrador de *scan*. Quando nem ShiftDR nem

loadsum estão ativos durante estas bordas é feita uma operação de captura da entrada da célula.

loadctrl: Durante o teste das conexões, mais especificamente durante a instrução SUMSHIFT/LOAD, permite a carga do valor de controle de *tri-state* durante o estado <Update DR>. Este valor se manterá constante durante a execução de CONTEST pela ação do sinal wkingbypass.

wkingbypass: Fica ativo durante toda a instrução CONTEST e é conectado às células de controle de *tri-state* com o objetivo de preservar seu valor durante o deslocamento do registrador de *scan*.

resetsumb: É encarregado de inicializar o registrador do resultado compactado do teste de interconexão após o valor resultante da aplicação de uma seqüência de *walking*-"0" ou *walking*-"*1*" ter sido lido no estado <Shift DR> durante a instrução SUMSHIFT/LOAD. Para isso, ele é ativado nesta mesma instrução, no estado <Update DR>.

Configuração do registrador de *scan*

O registrador de deslocamento conectado entre os pinos TDI e TDO pode variar conforme a instrução e o estado do controlador. Em um primeiro caso, quando em modo de teste de sistema, apenas as células que são conectadas aos pinos do MCM fazem parte deste registrador. Quando em modo de teste dos componentes, todas as células correspondentes a pinos do componente são conectados entre TDI e TDO. Finalmente, em qualquer um dos modos, durante a instrução BIST, além de todas as células correspondentes aos pinos, os registradores que são parte do CAR também integram a linha de *scan*. Isso pode ser facilmente visualizado na figura 4.1.

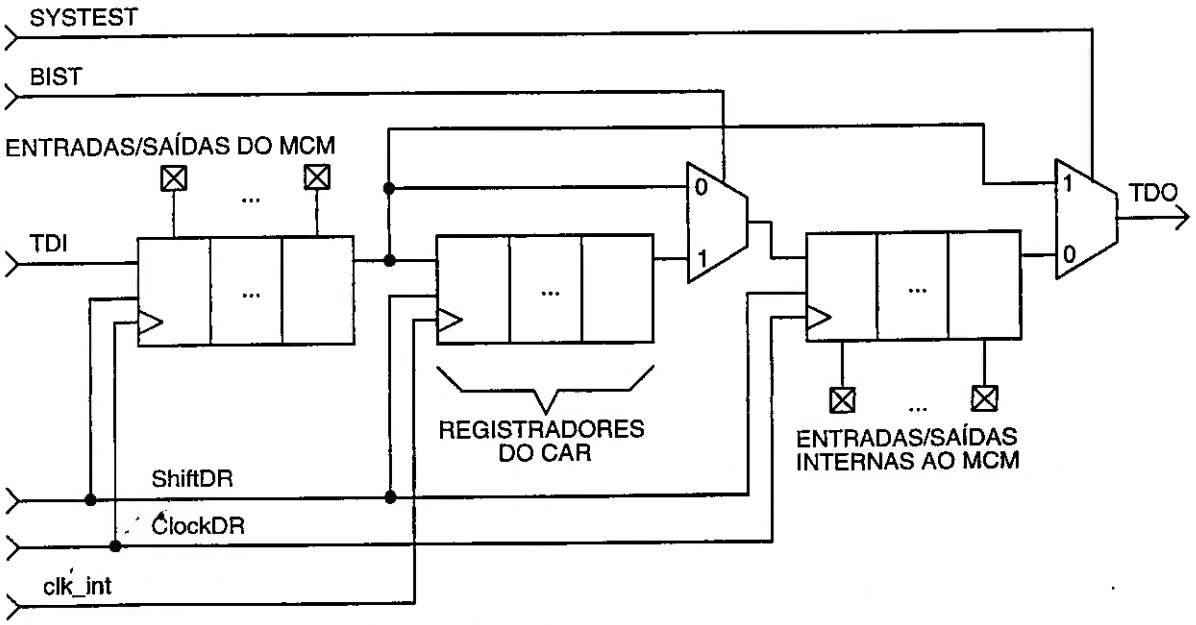


Figura 4.1: Montagem do registrador de *scan*.

Interrupção do Auto-Teste

Se antes do término do auto-teste o controlador for retirado do estado <Run Test/Idle> o auto-teste será interrompido. Nessa condição, o estado dos módulos de controle dos componentes internos, controlado durante o auto-teste pelo controle-mestre, perderá o sincronismo com o estado deste último. Para corrigir esta situação é necessário manter TMS no valor lógico "1" por um mínimo de 5 ciclos de TCK, garantindo assim que tanto os estados dos controladores dos componentes quanto o do controle-mestre seja <Test Logic Reset>. Este procedimento não é necessário se for dado tempo suficiente para o término do auto-teste.

Módulo de Controle do Auto-Teste do Componente

O módulo de controle do componente, quando recebe uma instrução BIST comanda um início de auto-teste que na prática é realizado por um outro módulo. Este módulo é encarregado de ler a programação dos CARs de uma memória, introduzir esta programação no CAR, aguardar o tempo suficiente para a geração dos vetores correspondentes a essa programação, e introduzir a programação seguinte até completar o teste. Se o teste for interrompido, ele será retomado desde o começo, pois os registradores podem ter sido alterados durante a interrupção.

A memória anexa a este controlador contém toda a programação necessária ao CAR, em um formato bastante conveniente para a implementação através de uma macro-célula de RAM, disponível com ampla variedade de tamanhos de memória e de número de *bits* por palavra em diversas *foundries*, tais como ES2 [ES2a95] e AMI [AMI94]. O tamanho de palavra utilizado no exemplo foi 8 *bits*, o qual pode ser facilmente alterado se for conveniente para a construção da memória. O módulo lê estas palavras e as serializa, introduzindo o resultado em um registrador de deslocamento responsável pela programação do CAR. Este registrador é formado pela junção dos registradores de programação das células de entrada da linha de *scan*, e será visto com detalhe na descrição destas células de entrada.

O resultado do auto-teste fica em um registrador responsável pela compactação da resposta da lógica aos vetores aplicados, o qual não é abordado neste trabalho. Esse registrador deve ser conectado entre TDI e TDO durante a instrução BIST e pode ser lido durante o estado <Shift DR>.

Células de Entrada, Saída e Controle de *Tri-state* da Linha de *Scan*

Estas células foram projetadas de forma a combinar a funcionalidade do teste de interconexões e do auto-teste através de um CAR. Seu projeto é baseado na teoria apresentada no capítulo 3.

Células de Entrada

Foram projetadas duas células de entrada, sendo que de uma delas foi retirado o compactador do teste de interconexão, substituído por uma lógica que simula o resultado correto do teste. Esta última célula deve ser usada nos pinos que se conectam aos pinos de entrada do MCM e portanto não participam do teste de interconexão. Esta alteração pode ser vista na figura A.4.

A figura A.3 representa a célula com funcionalidade completa. Na parte superior desta figura podemos ver os dois registradores de programação do CAR conectados em série e controlados pelos sinais car_prog_in e clk_car_prog, gerados pelo módulo de controle do auto-teste. No registrador da esquerda é programada a regra utilizada na célula, 90 ou 150. No outro registrador é programada a participação desta célula no CAR. Quando a célula está inativa, as realimentações das células anterior e posterior são propagadas para as células seguintes. O sinal car ativa a geração de vetores, e a cada borda de subida de updateDR (durante a geração de vetores controlado pelo módulo de controle do auto-teste) um novo vetor é gerado. A propagação para as células seguinte e anterior é feita através dos sinais fwdout, fwdin, bckout e bckin, respectivamente saída e entrada para propagação para a célula posterior e saída e entrada para propagação para a célula anterior. Quando o estágio do CAR correspondente a esta célula está inativo, fwdout é conectado a fwdin e bckout a bckin, em vez de serem conectados à saída da célula.

A funcionalidade correspondente ao teste de interconexões é controlada pelos sinais clksum, resetsumb e loadsum. Uma borda de subida em clksum armazena no registrador de compactação o resultado da operação OU-exclusivo entre seu próprio conteúdo e o valor do sinal in, correspondente à operação de compactação. O sinal resetsumb inicializa este registrador e loadsum habilita a transferência de seu conteúdo para o registrador de deslocamento.

O restante do controle é exercido pelos seguintes sinais:

scbypassb: responsável por tornar a célula "transparente", conectando out à in

shiftDR: seleciona shiftin para ser carregado por clkDR durante o estado <Shift DR>

clkDR: faz o deslocamento durante o estado <Shift DR> (shiftDR ativo) e a captura em <Shift DR> (da saída do registrador de compactação quando loadsum está ativo durante a instrução SUMSHIFT/LOAD ou da entrada in em caso contrário).

updateDR: Atualização do registrador de saída da célula no estado <Update DR>.

A construção do módulo de teste dos componentes e da célula de entrada permitiu que durante o teste de interconexões o deslocamento das seqüências *walking*-"0" e *walking*-"1" e a compactação do resultado fossem feitas em um mesmo ciclo de TCK, durante o estado <Shift DR>. Isto representa uma melhoria em relação ao esquema proposto em [Hass92], que prevê a passagem pelos estados <Shift DR>, <Exit1 DR>, <Update DR>, <Select DR Scan> e <Capture DR> para completar o ciclo. Da forma como foi implementado, pode-se chegar a um tempo de teste de interconexões quase cinco vezes menor, já que a maior parte das operações é feita em um ciclo, em vez de cinco.

Célula de Saída

É uma simples célula de *boundary-scan*. A funcionalidade de seus sinais é correspondente à dos sinais de mesmo nome da célula de entrada. É importante notar que em relação à célula apresentada no item 3 do capítulo 3, a lógica adicional, que fazia a realimentação da registrador de saída foi retirada. Isso foi possível devido ao esquema de teste de interconexão adotado, que não exige a passagem pelo estado <Capture DR> durante as seqüências *walking*-"0" e *walking*-"1". Neste estado o valor do registrador de deslocamento poderia ser alterado sem esta realimentação. Pelo mesmo motivo esta realimentação não existe nas células de entrada e de controle. É apresentada na figura A.5.

Célula de Controle de *Tri-state*

Em relação à célula de saída, tem a mais um *latch* responsável por manter o valor de saída constante durante o deslocamento das seqüências *walking*-"0" e *walking*-"1". Sua carga é feita pelo sinal loadctrl e seu valor é transmitido à saída pelo sinal wkingbypass. A figura A.6 mostra o circuito utilizado.

Capítulo 5

Resultados

Os circuitos descritos no capítulo 4 foram simulados, as descrições funcionais foram sintetizadas e foi feito o *layout* do resultado da síntese. Foram utilizadas no processo as ferramentas de projeto da Cadence e a biblioteca utilizada para simulação, síntese e *layout* foi a ECPD07, de largura de canal 0,7µm, da *foundry* ES2 [ES2a95].

Por se tratarem em parte de estruturas novas, não há referências de vetores de simulação para verificar o funcionamento correto dos circuitos. Procurou-se exercitar todas as funções desempenhadas pelo circuito em condições variadas em relação às entradas fornecidas para estas funções.

Todas as instruções previstas foram testadas, tanto em modo de teste de sistema quanto em modo de teste dos componentes. Foram verificadas as seqüências geradas pelos CAR e o efeito das seqüências *walking*-"0" e *walking*-"1" nas interconexões e nos comparadores. Para verificação da geração de vetores aleatórios pelo CAR foram utilizadas os conjuntos de regras 90-150-90-150 e 150-90-150-150-150-90, que geraram seqüências de 15 e 63 vetores respectivamente.

Para verificar a eficiência do teste de interconexões, foram introduzidas diversas falhas (a figura A.2 apresenta um exemplo) nas conexões do MCM exemplo. Todas as falhas simuladas geraram uma resposta adequada na compactação final do resultado, no caso um valor menor do que 108, como apresentado no capítulo 4.

O módulo de controle-mestre resultou em um circuito integrado de 16 pinos, apresentado na figura A.11. Novamente, lembramos que outra opção teria sido integrá-lo a um dos componentes do MCM, caso fosse conveniente. As dimensões do circuito, excluídos os pinos são 743,2µm X 788,3µm, correspondendo a uma área de 0,5858mm^2. Incluindo os pinos as dimensões são 1.736,1µm X 1.656,3µm e a área 2,875mm^2. Foram usadas 465 células, ou 4.598 transistores equivalentes (cerca de 1.150 *gates* equivalentes).

O módulo de controle de teste dos componentes completo (o do circuito 1 - figura A.7) resultou em uma célula de dimensões 324,3µm X 396,1µm, ou seja, 0,1284mm^2 de área. É composto de 118 células que se traduzem em 1.105 transistores equivalentes (cerca de

276 *gates* equivalentes). Uma célula semelhante da ES2 (CORE2R) [ES2b95] tem 1.267 transistores equivalentes, sendo portanto um pouco maior. Para efeito de comparação foi feito o seu *layout* (na biblioteca da ES2 ela é apresentada como uma macro-célula e pode ser usada em qualquer tecnologia) resultando em uma área de 0,1377mm^2. Esta célula da ES2 não implementa as instruções SYSTEST, EXPLODE, SUMSHIFT/LOAD e CONTEST, definidas neste trabalho, mas por outro lado disponibiliza conexão através de *multiplexes* e *demultiplexes* a 8 registradores definíveis pelo usuário, justificando um tamanho ligeiramente maior. Com este resultado verificamos que a célula produzida neste trabalho se assemelha bastante a uma em uso comercial em termos de eficiência de implementação.

O módulo de controle do auto-teste dos componentes resultou em um bloco com 162 células ou 1846 transistores equivalentes (cerca de 461 *gates* equivalentes). O *layout* deste bloco ficou com dimensões de 436µm X 447µm, ou seja, uma área de 0,1948mm^2.

Trabalhos Futuros

A linha de trabalho seguida dá margem a diversas extensões. Entre estas destacamos algumas.

Flexibilização das células ligadas a pinos do MCM

O exemplo apresentado considerava que os pinos dos componentes que se conectavam a pinos do MCM estavam bem determinados, ou porque se destinavam a uma única aplicação ou porque teriam sido projetados para serem usados de forma que esta seria a única possibilidade de conexão. Contudo é muito provável que existam componentes que podem ser empregados em diversos tipos de MCM e que nem sempre os mesmos pinos sejam conectados ao exterior. Isso pode ser resolvido adotando-se uma estrutura de programação nas células de entrada, saída e controle que as tornem de uso geral. A instrução EXPLODE poderia selecionar o registrador responsável por esta programação, a qual seria então introduzida serialmente.

Auto-Teste do MCM

Pode ser investigada, em uma determinada aplicação, a conveniência de fazer um auto-teste simplificado, envolvendo apenas as células de entrada do MCM. Este objetivo poderia suprir parcialmente o teste interno de componentes que não disponham de auto-teste próprio e o teste das conexões envolvendo componentes que não implementem as instruções sugeridas neste trabalho.

Implementação da Compactação dos Resultados do Auto-Teste

O circuito de auto-teste dos componentes deve ser completado com base em algum dos esquemas de compactação de resultados existentes. Além disso, no caso de um componente específico deve ser feito o particionamento, o que será usado para determinar a programação do CAR.

Conclusão

Foi feita uma aplicação de técnicas de testabilidade ao teste de MCMs, implementadas de acordo com o padrão IEEE 1149.1. Os circuitos simples e viáveis comercialmente mostraram que estes esquemas são adequados ao teste do MCM, principalmente por sua característica de subdividir os testes necessários. O esquema de teste de interconexões se revela ainda mais adequado quando aplicado a um MCM do que quando aplicado a uma placa de circuito impresso, já que é maior a probabilidade de que sejam projetados circuitos dedicados a um MCM que contenham a funcionalidade necessária. Além disso, um circuito gerador do teste pode ser facilmente incluído, simplificando a geração de programas de teste.

Também no sentido de simplificar os testes e facilitar o acesso interno, foi proposto um esquema que transforma o MCM no conjunto de seus componentes que então podem ser acessados como se tivessem sido montados diretamente na placa de circuito impresso.

Com os resultados obtidos espera-se ter apresentado soluções para alguns dos problemas de testabilidade dos MCMs, apresentados no primeiro capítulo.

Anexo

MCM Exemplo

Neste anexo são apresentadas as descrições funcionais dos módulos de controle, os circuitos esquemáticos da composição do MCM e o *layout* da célula de controle-mestre. Os blocos 1,2,3 e 4 que aparecem dentro dos componentes, ligados entre as entradas e as saídas representam blocos de lógica genérica e por isso não são destacados.

Descrição funcional do módulo "controle-mestre"

```
// Verilog HDL for "bscan", "contr_mestre" "_behavioral"

`timescale 1ns/100ps

module contr_mestre
(tdi,tms,tclk,tdo,trstb,tms_int,tdi_int,tdo_int,conta_teste,mem_prog);


parameter     EXIT2_DR=0,       EXIT1_DR=1,       SHIFT_DR=2,
              PAUSE_DR=3,       SELECT_IR_SCAN=4,
              UPDATE_DR=5,      CAPTURE_DR=6,     SELECT_DR_SCAN=7,
              EXIT2_IR=8, EXIT1_IR=9, SHIFT_IR='hA,
              PAUSE_IR='hB,     RUN_TEST_IDLE='hC,
              UPDATE_IR='hD,    CAPTURE_IR='hE,
              TEST_LOGIC_RESET='hF;


parameter     BYPASS=7,  BIST=3,
              SYS_TEST=4,       EXPLODE=5;


parameter     TAM_CT_DADO_INSTR=6,
              INSTR_SYSTEM=16'b0100010001000100,
              TAM_SCAN_INSTR=16,
              INSTR_EXPLODE=16'b0101010101010101,
              INSTR_CON_TEST=16'b0111011101110111,
              INSTR_BIST=16'b0011001100110011,
              INSTR_SUMSHIFT_LOAD=16'b1000100010001000,
              FILL=24'b0,       TAM_SHIFT=34,     TAM_DADO_CTRL=36,
              PRIM_BIT_UM=36'h000000001,
        PRIM_BIT_ZERO=36'hFFFFFFFFE,
              N_TESTE_CON=2,          TUDO_UM=36'hFFFFFFFFF,
        TAM_COMPACTA=7,
              TAM_CONT_BIST=8,        N_BIST_INT=148;


input tdi,tms,tclk,trstb,tdo_int;
input [1:0]mem_prog;

output tdi_int,tms_int,tdo;
output [1:0] conta_teste;

reg [1:0] conta_teste;
reg [2:0] estado_dd_instr,ireg,iregl;
reg [3:0] estado;
reg [4:0] estado_bist;
```

```
reg [TAM_DADO_CTRL-1:0] dado_instr;
reg [TAM_CT_DADO_INSTR-1:0] ct_dado_instr;
reg [TAM_COMPACTA-1:0] compacta,bistreg;
reg [TAM_CONT_BIST-1:0] cont_bist;


reg
shiftDR,captureDR,shiftIR,bypassreg,systest,carrega_instr,carrega_dado,bit_fim,fim_dd
_instr,fim_bist_interno;


wire
        bypass_sys=((iregl==BYPASS)||(iregl==SYS_TEST)||(iregl==EXPLODE))&&sy
stest,
        bist=(iregl==BIST),
        tdi_test=dado_instr[0],
        tdi_int=(bist&&~shiftIR)? tdi_test : tdi,
        tdo=(shiftDR||shiftIR)?
(~shiftIR&&((bist&&bistreg[0])||(bypass_sys&&bypassreg)||(~bypass_sys&&~bist&&t
do_int)))||(ireg[0]&&shiftIR) : 1'bz,
//Quando em shiftIR, o registrador deste modulo esta' no final da linha
        hab_tms_int=~((estado_bist==0)||(estado_bist==14)||(estado_bist==15)),
        tms_int=(hab_tms_int)?
(estado_dd_instr==1||estado_dd_instr==2||estado_dd_instr==6||estado_dd_instr==7) :
tms;

//Reset do modulo
always @(trstb)
if (trstb==0)
begin
        assign estado='hF;
        assign iregl='h7;
        assign shiftDR=0;
        assign captureDR=0;
        assign systest=1;
end
else
begin
        deassign estado;
        deassign iregl;
        deassign shiftDR;
        deassign captureDR;
        deassign systest;
end


//Reset do BIST
always @(bist&&(estado==RUN_TEST_IDLE))
if (~(bist&&(estado==RUN_TEST_IDLE)))
```

```
        begin
        assign estado_bist='hF;
        assign estado_dd_instr=0;
        assign carrega_instr=0;
        assign carrega_dado=0;
        end
else
        begin
        deassign estado_bist;
        deassign estado_dd_instr;
        deassign carrega_instr;
        deassign carrega_dado;
        end


//Maquina de estado de controle do modulo
always @(posedge tclk)
#1 case(estado)
        //Exit2-DR
        EXIT2_DR:if (tms) estado=UPDATE_DR;
                else estado=SHIFT_DR;
        //Exit1-DR
        EXIT1_DR:if (tms) estado=UPDATE_DR;
                else estado=PAUSE_DR;
        //Shift-DR
        SHIFT_DR:if (tms) estado=EXIT1_DR;
        //Pause-DR
        PAUSE_DR:if (tms) estado=EXIT2_DR;
        //Select-IR-Scan
        SELECT_IR_SCAN: if (tms) estado=TEST_LOGIC_RESET;
                        else estado=CAPTURE_IR;
        //Updade-DR
        UPDATE_DR:if (tms) estado=SELECT_DR_SCAN;
                else estado=RUN_TEST_IDLE;
        //Capture-DR
        CAPTURE_DR:if (tms) estado=EXIT1_DR;
                else estado=SHIFT_DR;
        //Select-DR-Scan
        SELECT_DR_SCAN:if (tms) estado=SELECT_IR_SCAN;
                else estado=CAPTURE_DR;
        //Exit2-IR
        EXIT2_IR:if (tms) estado=UPDATE_IR;
                else estado=SHIFT_IR;
        //Exit1-IR
        EXIT1_IR:if (tms) estado=UPDATE_IR;
                else estado=PAUSE_IR;
        //Shift-IR
        SHIFT_IR:if (tms) estado=EXIT1_IR;
        //Pause-IR
```

```
        PAUSE_IR:if (tms) estado=EXIT2_IR;
        //Run-Test/Idle
        RUN_TEST_IDLE:if (tms) estado=SELECT_DR_SCAN;
        //Update-IR
        UPDATE_IR:if (tms) estado=SELECT_DR_SCAN;
                else estado=RUN_TEST_IDLE;
        //Capture-IR
        CAPTURE_IR:if (tms) estado=EXIT1_IR;
                else estado=SHIFT_IR;
        //Test-Logic-Reset
        TEST_LOGIC_RESET:if (~tms) estado=RUN_TEST_IDLE;
endcase

always @(negedge tclk)
begin
if (estado==SHIFT_DR)
        //Seleciona Shift-DR
        shiftDR=1;
        else
        shiftDR=0;
if (estado==CAPTURE_DR)
        //Seleciona Capture-DR
        captureDR=1;
        else
        captureDR=0;
if (estado==SHIFT_IR)
        //Seleciona Shift-IR
        shiftIR=1;
        else
        shiftIR=0;
if (estado==UPDATE_IR)
        //No estado Update-IR a saida paralela do registrador de instrucao
        //recebe o conteudo do registrador de deslocamento
        iregl=ireg;
if (estado==TEST_LOGIC_RESET)
        begin
        //a saida paralela do registrador de instrucao recebe a instrucao BYPASS
        iregl='h7;
        end
end


always @(posedge tclk)
begin
fork
if (estado==CAPTURE_IR)
        //No estado Capture-IR o registrador de deslocamento deve carregar
        //01 nos dois bits menos significativos
        #1 ireg=3'b001;
```

```
if (shiftIR)
        //No estado Shift-IR o registrador desloca uma posicao e recebe tdi no
        //bit menos significativo
        #1 ireg={tdo_int,ireg[2:1]};
if (shiftDR&&bypass_sys)
        #1 bypassreg=tdi;
if (shiftDR&&bist)
        #1 bistreg={tdo_int,bistreg[TAM_COMPACTA-1:1]};
if (iregl==SYS_TEST)
        #1 systest=1;
if (iregl==EXPLODE)
        #1 systest=0;
if (captureDR&&bist)
        #1 bistreg=compacta;
case(estado_bist)
0:      begin
        if (bist) compacta=0;
        cont_bist=0;
        fim_bist_interno=0;
        end
7,11:if (estado_dd_instr==5||estado_dd_instr==6)
        compacta=compacta+tdo_int;
13:     begin
        cont_bist=cont_bist+1;
        #1 if (cont_bist==N_BIST_INT) fim_bist_interno=1;
        end
default:compacta=compacta;
endcase
#1 fim_dd_instr=(estado_dd_instr==7);
join
end

//A proxima maquina de estados e' responsavel pela realizacao do auto-teste
//Primeiro e' feito o teste de interconexao e depois o auto-teste dos componentes
always @(negedge tclk)
case(estado_bist)
0:      begin
        conta_teste=0;
        #1 carrega_instr=1;
        estado_bist=1;
        end
1:      begin
        if(fim_dd_instr)
                begin
                estado_bist=2;
                end
        end
2:      begin
        #1 carrega_instr=0;
```

```
        carrega_dado=1;
        if(fim_dd_instr)
                begin
                estado_bist=3;
                end
        end
3:      begin
        #1 carrega_dado=0;
        carrega_instr=1;
        if(fim_dd_instr)
                begin
                estado_bist=4;
                end
        end
4:      begin
        #1 carrega_instr=0;
        carrega_dado=1;
        if(fim_dd_instr)
                begin
                bit_fim=0;
                estado_bist=5;
                end
        end
5:      begin
        #1 carrega_dado=0;
        carrega_instr=1;
        if(fim_dd_instr)
                begin
                estado_bist=6;
                end
        end
6:      begin
        #1 carrega_instr=0;
        carrega_dado=1;
        if(fim_dd_instr)
                begin
                dado_instr={5'h1F,mem_prog[1],16'hFFFF,mem_prog[0],13'h1FFF};
                bit_fim=1;
                estado_bist=7;
                end
        end
7:      begin
        #1 carrega_dado=0;
        carrega_instr=1;
        if(fim_dd_instr)
                begin
                estado_bist=8;
                end
        end
```

```
8:      begin
        #1 carrega_instr=0;
        carrega_dado=1;
        if(fim_dd_instr)
                begin
                estado_bist=9;
                end
        end
9:      begin
        #1 carrega_dado=0;
        carrega_instr=1;
        if(fim_dd_instr)
                begin
                conta_teste=conta_teste+1;
                estado_bist=10;
                end
        end
10:     begin
        #1 carrega_instr=0;
        carrega_dado=1;
        if(fim_dd_instr)
           begin
bit_fim=0;
           estado_bist=11;
                end
        end
11:     begin
        #1 carrega_dado=0;
        carrega_instr=1;
        if(fim_dd_instr)
                begin
                if(conta_teste<N_TESTE_CON)
                        begin
                        estado_bist=4;
                        end
                else begin
                        estado_bist=12;
                        end
                end
        end
12:     begin
        if(fim_dd_instr)
                begin
                estado_bist=13;
                end
        end
13:     begin
        #1 carrega_instr=0;
        if (fim_bist_interno) estado_bist=14;
```

```
        end
14:     estado_bist=14;
15:     if(bist && estado==RUN_TEST_IDLE && systest) estado_bist=0;
        else
        begin
        #1 carrega_instr=0;
        carrega_dado=0;
        end
endcase


// Quando hab_tms_int=1: tms_int =
//(estado_dd_instr==1||estado_dd_instr==2||estado_dd_instr==6||estado_dd_instr==7)
// tms_int controla a mudanca de estados dos componentes internos durante o auto-teste
// Esta maquina de estados e' responsavel pelo controle dos estados internos dos
//componentes

always @(negedge tclk)
case(estado_dd_instr)
0:      begin
        #1 if (carrega_instr) estado_dd_instr=1;
        if (carrega_dado) estado_dd_instr=2;
        end
//durante estado 1 tms_int leva estado dos componentes internos para
//Select-DR-Scan
1:      #1 estado_dd_instr=2;
//tms_int leva estado para Select-IR-Scan
2:      #1 estado_dd_instr=3;
//tms_int leva estado para Capture-IR
3:      begin
        #1 estado_dd_instr=4;
        case (estado_bist)
        1:      begin
            dado_instr={FILL,INSTR_EXPLODE};
            ct_dado_instr=TAM_SCAN_INSTR;
            end
    2:      begin
            dado_instr={FILL,INSTR_SUMSHIFT_LOAD};
            ct_dado_instr=TAM_SCAN_INSTR;
            end
    3,11:   begin
            dado_instr={5'h0,mem_prog[1],16'h0,mem_prog[0],13'h0};
            ct_dado_instr=TAM_DADO_CTRL;
            end
    4:      begin
            dado_instr={FILL,INSTR_CON_TEST};
            ct_dado_instr=TAM_SCAN_INSTR;
            end
    5:      begin
```

```
            dado_instr=PRIM_BIT_UM;
                ct_dado_instr=TAM_SHIFT;
            end
        6:   begin
            dado_instr={FILL,INSTR_SUMSHIFT_LOAD};
            ct_dado_instr=TAM_SCAN_INSTR;
            end
        7:   begin
            dado_instr={5'h1F,mem_prog[1],16'hFFFF,mem_prog[0],13'h1FFF};
            ct_dado_instr=TAM_DADO_CTRL;
            end
        8:   begin
            dado_instr={FILL,INSTR_CON_TEST};
            ct_dado_instr=TAM_SCAN_INSTR;
            end
        9:   begin
            dado_instr=PRIM_BIT_ZERO;
            ct_dado_instr=TAM_SHIFT;
            end
        10:    begin
            dado_instr={FILL,INSTR_SUMSHIFT_LOAD};
            ct_dado_instr=TAM_SCAN_INSTR;
            end
          12:    begin
            if(conta_teste<N_TESTE_CON)
                        dado_instr={FILL,INSTR_CON_TEST};
                            else dado_instr={FILL,INSTR_SYSTEM};
            ct_dado_instr=TAM_SCAN_INSTR;
            end
        13:    begin
            dado_instr={FILL,INSTR_BIST};
            ct_dado_instr=TAM_SCAN_INSTR;
            end
        default:begin
            dado_instr=dado_instr;
            ct_dado_instr=ct_dado_instr;
            end
          endcase
          end
//tms_int leva estado para Shift-IR
4:      #1 estado_dd_instr=5;
5:      begin
        #1 dado_instr={bit_fim,dado_instr[TAM_DADO_CTRL-1:1]};
        if(ct_dado_instr==2) estado_dd_instr=6;
        #1 ct_dado_instr=ct_dado_instr-1;
        end
//tms_int leva estado para Exit1-IR
6:      #1 estado_dd_instr=7;
//tms_int leva estado para Update-IR
```

```
7:      begin
        fork
        if (carrega_instr) #1 estado_dd_instr=1;
        if (carrega_dado) #1 estado_dd_instr=2;
        if (~(carrega_dado||carrega_instr)) #1 estado_dd_instr=0;
        join
        end
//tms_int leva estado para Run-Test-Idle
endcase


endmodule
```

Descrição funcional do módulo de controle de teste do componente 1

```
// Verilog HDL for "bscan", "bscan_ctrl" "_behavioral"

`timescale 1ns/100ps

module bscan_ctrl
(tdi,tms,tclk,tdo,trstb,scanout,scanout0,scanout1,inicio_car,inicia_car,clockDRin,clock
DRout,shiftDR,hab_updateDR_inpad,hab_updateDR_outpadb,hab_updateDR_outextb,s
canbypassb,scanbypassintb,scanin,scanin0,loadsum,clksum,loadctrl,wkingbypass,resets
umb);

parameter     EXIT2_DR=0,    EXIT1_DR=1,    SHIFT_DR=2,
          PAUSE_DR=3,    SELECT_IR_SCAN=4,
          UPDATE_DR=5,   CAPTURE_DR=6,  SELECT_DR_SCAN=7,
          EXIT2_IR=8,    EXIT1_IR=9,    SHIFT_IR='hA,
          PAUSE_IR='hB,  RUN_TEST_IDLE='hC,
          UPDATE_IR='hD, CAPTURE_IR='hE, TEST_LOGIC_RESET='hF;

parameter     BYPASS=15, EXTEST=0,  INTEST=1,
              SAMPLE_PRELOAD=2,    SUMSHIFT_LOAD=8,      BIST=3,
              SYS_TEST=4,        EXPLODE=5,        CON_TEST=7;

input tdi,tms,tclk,trstb,scanout,scanout0,scanout1,inicia_car;

output
tdo,inicio_car,clockDRin,clockDRout,shiftDR,hab_updateDR_inpad,hab_updateDR_ou
tpadb,hab_updateDR_outextb,scanbypassb,scanbypassintb,scanin,scanin0,loadsum,clks
um,loadctrl,wkingbypass,resetsumb;

reg [3:0] ireg,iregl;
reg [3:0] estado;

reg
captureDR,bypassreg,bistreg,shiftDR_int,shiftIR,inicia_car2,systest,shiftDR_int2,shift
DR_int3;

wire    bypass=(iregl==BYPASS||iregl==SYS_TEST||iregl==EXPLODE),
        bist=(iregl==BIST),
        intest=(iregl==INTEST),
        extest=(iregl==EXTEST),
        sample_preload=(iregl==SAMPLE_PRELOAD),
        con_test=(iregl==CON_TEST),
        sumshift_load=(iregl==SUMSHIFT_LOAD),
        shiftDR=shiftDR_int&&(~bypass),
```

```
        clockDRin=(shiftDR||(captureDR&&(extest||sample_preload||sumshift_load)))&
&tclk,
        clockDRout=(shiftDR||(captureDR&&(intest||sample_preload)))&&tclk,
        clksum=shiftDR_int3&&con_test&&tclk,
        loadsum=captureDR&&sumshift_load,
        wkingbypass=con_test,
        resetsumb=~(sumshift_load&&(estado==UPDATE_DR)),
        scanin0=(bist)? scanout1 : scanout0,
        scanin=tdi||inicia_car2,
        hab_updateDR_inpad=(intest||sample_preload||sumshift_load)&&(estado==UPD
ATE_DR),
        hab_updateDR_outpadb=~(((extest||sample_preload||sumshift_load)&&(estado=
=UPDATE_DR))||(shiftDR_int2&&con_test)),
        hab_updateDR_outextb=~((extest||sample_preload)&&(estado==UPDATE_DR))
,
        loadctrl=(sample_preload||sumshift_load)&&(estado==UPDATE_DR),
        scanbypassb=~(bypass||sample_preload),
        scanbypassintb=(scanbypassb&&~systest)||bist,
        pre_tdo=(systest)? scanout0 : scanout,
        tdo=(shiftDR_int||shiftIR)?
(~shiftIR&&((bist&&bistreg)||(bypass&&bypassreg)||(~bypass&&~bist&&pre_tdo)))||(i
reg[0]&&shiftIR) : 1'bz,
        inicio_car=bist&&(estado==RUN_TEST_IDLE);

//Reset do modulo
always @(trstb)
if (trstb==0)
begin
        assign estado='hF;
        assign iregl='hF;
        assign shiftDR_int=0;
        assign shiftIR=0;
        assign captureDR=0;
        assign systest=1;
        assign bistreg=1;
end
else
begin
        deassign estado;
        deassign iregl;
        deassign shiftDR_int;
        deassign shiftIR;
        deassign captureDR;
        deassign systest;
        deassign bistreg;
end

//Maquina de estado de controle do modulo
```

```
always @(posedge tclk)
#1 case(estado)
      //Exit2-DR
      EXIT2_DR:if (tms) estado=UPDATE_DR;
            else estado=SHIFT_DR;
      //Exit1-DR
      EXIT1_DR:if (tms) estado=UPDATE_DR;
            else estado=PAUSE_DR;
      //Shift-DR
      SHIFT_DR:if (tms) estado=EXIT1_DR;
      //Pause-DR
      PAUSE_DR:if (tms) estado=EXIT2_DR;
      //Select-IR-Scan
      SELECT_IR_SCAN: if (tms) estado=TEST_LOGIC_RESET;
                  else estado=CAPTURE_IR;
      //Updade-DR
      UPDATE_DR:if (tms) estado=SELECT_DR_SCAN;
            else estado=RUN_TEST_IDLE;
      //Capture-DR
      CAPTURE_DR:if (tms) estado=EXIT1_DR;
            else estado=SHIFT_DR;
      //Select-DR-Scan
      SELECT_DR_SCAN:if (tms) estado=SELECT_IR_SCAN;
            else estado=CAPTURE_DR;
      //Exit2-IR
      EXIT2_IR:if (tms) estado=UPDATE_IR;
            else estado=SHIFT_IR;
      //Exit1-IR
      EXIT1_IR:if (tms) estado=UPDATE_IR;
            else estado=PAUSE_IR;
      //Shift-IR
      SHIFT_IR:if (tms) estado=EXIT1_IR;
      //Pause-IR
      PAUSE_IR:if (tms) estado=EXIT2_IR;
      //Run-Test/Idle
      RUN_TEST_IDLE:if (tms) estado=SELECT_DR_SCAN;
      //Update-IR
      UPDATE_IR:if (tms) estado=SELECT_DR_SCAN;
            else estado=RUN_TEST_IDLE;
      //Capture-IR
      CAPTURE_IR:if (tms) estado=EXIT1_IR;
            else estado=SHIFT_IR;
      //Test-Logic-Reset
      TEST_LOGIC_RESET:if (~tms) estado=RUN_TEST_IDLE;
endcase

always @(posedge tclk)
begin
```

```
fork
if (estado==CAPTURE_IR)
        #1 ireg=4'b0001;
if (estado==SHIFT_IR)
        #1 ireg={tdi,ireg[3:1]};
if (shiftDR_int&&bypass)
        #1 bypassreg=tdi;
if (shiftDR_int&&bist)
        #1 bistreg=tdi;
if (iregl==SYS_TEST)
        #1 systest=1;
if (iregl==EXPLODE)
        #1 systest=0;
#1 shiftDR_int2=shiftDR_int;
join
end

always @(negedge tclk)
begin
#1 inicia_car2=inicia_car;
shiftDR_int3=shiftDR_int2;
if ((estado==SHIFT_DR)||inicia_car)
        //Seleciona Shift-DR
        shiftDR_int=1;
        else
        shiftDR_int=0;
if (estado==CAPTURE_DR)
        //Seleciona Capture-DR
        captureDR=1;
        else
        captureDR=0;
if (estado==SHIFT_IR)
        //Seleciona Shift-IR
        shiftIR=1;
        else
        shiftIR=0;
if (estado==UPDATE_IR)
        //No estado Update-IR a saida paralela do registrador de instrucao
        //recebe o conteudo do registrador de deslocamento
        iregl=ireg;
if (estado==TEST_LOGIC_RESET)
        begin
        //a saida paralela do registrador de instrucao recebe a instrucao BYPASS
        iregl='hF;
        end
end

endmodule
```

Descrição funcional do módulo de controle de teste do componente 2

```
// Verilog HDL for "bscan", "bscan_ctrl1" "_behavioral"

`timescale 1ns/100ps

module bscan_ctrl1
(tdi,tms,tclk,tdo,trstb,scanout,inicio_car,inicia_car,clockDRin,clockDRout,shiftDR,hab
_updateDR_inpad,hab_updateDR_outpadb,scanbypassb,scanbypassintb,scanin,loadsum,
clksum,resetsumb);

parameter     EXIT2_DR=0,    EXIT1_DR=1,    SHIFT_DR=2,
          PAUSE_DR=3,    SELECT_IR_SCAN=4,
          UPDATE_DR=5,   CAPTURE_DR=6,  SELECT_DR_SCAN=7,
          EXIT2_IR=8,    EXIT1_IR=9,    SHIFT_IR='hA,
          PAUSE_IR='hB,  RUN_TEST_IDLE='hC,
          UPDATE_IR='hD, CAPTURE_IR='hE, TEST_LOGIC_RESET='hF;

parameter     BYPASS=15, EXTEST=0,  INTEST=1,
              SAMPLE_PRELOAD=2,    SUMSHIFT_LOAD=8,      BIST=3,
              SYS_TEST=4,        EXPLODE=5,        CON_TEST=7;

input tdi,tms,tclk,trstb,scanout,inicia_car;

output
tdo,inicio_car,clockDRin,clockDRout,shiftDR,hab_updateDR_inpad,hab_updateDR_ou
tpadb,scanbypassb,scanbypassintb,scanin,loadsum,clksum,resetsumb;

reg [3:0] ireg,iregl;
reg [3:0] estado;

reg
captureDR,bypassreg,bistreg,shiftDR_int,shiftIR,inicia_car2,systest,shiftDR_int2,shift
DR_int3;

wire    bypass=(iregl==BYPASS||iregl==SYS_TEST||iregl==EXPLODE),
        bist=(iregl==BIST),
        intest=(iregl==INTEST),
        extest=(iregl==EXTEST),
        sample_preload=(iregl==SAMPLE_PRELOAD),
        con_test=(iregl==CON_TEST),
        sumshift_load=(iregl==SUMSHIFT_LOAD),
        shiftDR=shiftDR_int&&(~bypass),
        clockDRin=(shiftDR||(captureDR&&(extest||sample_preload||sumshift_load)))&
&tclk,
```

```
        clockDRout=(shiftDR||(captureDR&&(intest||sample_preload)))&&tclk,
        clksum=shiftDR_int3&&con_test&&tclk,
        loadsum=captureDR&&sumshift_load,
        resetsumb=~(sumshift_load&&(estado==UPDATE_DR)),
        scanin=tdi||inicia_car2,
        hab_updateDR_inpad=(intest||sample_preload||sumshift_load)&&(estado==UPD
ATE_DR),
        hab_updateDR_outpadb=~(((extest||sample_preload||sumshift_load)&&(estado=
=UPDATE_DR))||(shiftDR_int2&&con_test)),
        scanbypassb=~(bypass||sample_preload),
        scanbypassintb=(scanbypassb&&~systest)||bist,
        pre_tdo=(systest)? tdi : scanout,
        tdo=(shiftDR_int||shiftIR)?
(~shiftIR&&((bist&&bistreg)||(bypass&&bypassreg)||(~bypass&&~bist&&pre_tdo)))||(i
reg[0]&&shiftIR) : 1'bz,
        inicio_car=bist&&(estado==RUN_TEST_IDLE);


//Reset do modulo
always @(trstb)
if (trstb==0)
begin
        assign estado='hF;
        assign iregl='hF;
        assign shiftDR_int=0;
        assign shiftIR=0;
        assign captureDR=0;
        assign systest=1;
        assign bistreg=1;
end
else
begin
        deassign estado;
        deassign iregl;
        deassign shiftDR_int;
        deassign shiftIR;
        deassign captureDR;
        deassign systest;
        deassign bistreg;
end


//Maquina de estado de controle do modulo
always @(posedge tclk)
#1 case(estado)
      //Exit2-DR
      EXIT2_DR:if (tms) estado=UPDATE_DR;
            else estado=SHIFT_DR;
```

```
        //Exit1-DR
        EXIT1_DR:if (tms) estado=UPDATE_DR;
              else estado=PAUSE_DR;
        //Shift-DR
        SHIFT_DR:if (tms) estado=EXIT1_DR;
        //Pause-DR
        PAUSE_DR:if (tms) estado=EXIT2_DR;
        //Select-IR-Scan
        SELECT_IR_SCAN: if (tms) estado=TEST_LOGIC_RESET;
                    else estado=CAPTURE_IR;
        //Updade-DR
        UPDATE_DR:if (tms) estado=SELECT_DR_SCAN;
              else estado=RUN_TEST_IDLE;
        //Capture-DR
        CAPTURE_DR:if (tms) estado=EXIT1_DR;
              else estado=SHIFT_DR;
        //Select-DR-Scan
        SELECT_DR_SCAN:if (tms) estado=SELECT_IR_SCAN;
              else estado=CAPTURE_DR;
        //Exit2-IR
        EXIT2_IR:if (tms) estado=UPDATE_IR;
              else estado=SHIFT_IR;
        //Exit1-IR
        EXIT1_IR:if (tms) estado=UPDATE_IR;
              else estado=PAUSE_IR;
        //Shift-IR
        SHIFT_IR:if (tms) estado=EXIT1_IR;
        //Pause-IR
        PAUSE_IR:if (tms) estado=EXIT2_IR;
        //Run-Test/Idle
        RUN_TEST_IDLE:if (tms) estado=SELECT_DR_SCAN;
        //Update-IR
        UPDATE_IR:if (tms) estado=SELECT_DR_SCAN;
              else estado=RUN_TEST_IDLE;
        //Capture-IR
        CAPTURE_IR:if (tms) estado=EXIT1_IR;
              else estado=SHIFT_IR;
        //Test-Logic-Reset
        TEST_LOGIC_RESET:if (~tms) estado=RUN_TEST_IDLE;
endcase

always @(posedge tclk)
begin
fork
if (estado==CAPTURE_IR)
        #1 ireg=4'b0001;
if (estado==SHIFT_IR)
        #1 ireg={tdi,ireg[3:1]};
```

```
if (shiftDR_int&&bypass)
        #1 bypassreg=tdi;
if (shiftDR_int&&bist)
        #1 bistreg=tdi;
if (iregl==SYS_TEST)
        #1 systest=1;
if (iregl==EXPLODE)
        #1 systest=0;
#1 shiftDR_int2=shiftDR_int;
join
end


always @(negedge tclk)
begin
#1 inicia_car2=inicia_car;
shiftDR_int3=shiftDR_int2;
shiftDR_int=((estado==SHIFT_DR)||inicia_car);
        //Seleciona Shift-DR
captureDR=(estado==CAPTURE_DR);
        //Seleciona Capture-DR
shiftIR=(estado==SHIFT_IR);
        //Seleciona Shift-IR
if (estado==UPDATE_IR)
        //No estado Update-IR a saida paralela do registrador de instrucao
        //recebe o conteudo do registrador de deslocamento
        iregl=ireg;
if (estado==TEST_LOGIC_RESET)
        begin
        //a saida paralela do registrador de instrucao recebe a instrucao BYPASS
        iregl='hF;
        end
end


endmodule
```

Descrição funcional do módulo de controle de teste do componente 3

```
// Verilog HDL for "bscan", "bscan_ctrl2" "_behavioral"

`timescale 1ns/100ps

module bscan_ctrl2
(tdi,tms,tclk,tdo,trstb,scanout,scanout0,inicio_car,inicia_car,clockDRin,clockDRout,shif
tDR,hab_updateDR_inpad,hab_updateDR_outpadb,hab_updateDR_outextb,scanbypassb
,scanbypassintb,scanin,scanin0,loadsum,clksum,loadctrl,wkingbypass,resetsumb);

parameter     EXIT2_DR=0,    EXIT1_DR=1,    SHIFT_DR=2,
          PAUSE_DR=3,    SELECT_IR_SCAN=4,
          UPDATE_DR=5,   CAPTURE_DR=6,  SELECT_DR_SCAN=7,
          EXIT2_IR=8,    EXIT1_IR=9,    SHIFT_IR='hA,
          PAUSE_IR='hB,  RUN_TEST_IDLE='hC,
          UPDATE_IR='hD, CAPTURE_IR='hE, TEST_LOGIC_RESET='hF;

parameter     BYPASS=15, EXTEST=0,  INTEST=1,
              SAMPLE_PRELOAD=2,   SUMSHIFT_LOAD=8,      BIST=3,
              SYS_TEST=4,    EXPLODE=5,    CON_TEST=7;

input tdi,tms,tclk,trstb,scanout,scanout0,inicia_car;

output
tdo,inicio_car,clockDRin,clockDRout,shiftDR,hab_updateDR_inpad,hab_updateDR_ou
tpadb,hab_updateDR_outextb,scanbypassb,scanbypassintb,scanin,scanin0,loadsum,clks
um,loadctrl,wkingbypass,resetsumb;

reg [3:0] ireg,iregl;
reg [3:0] estado;

reg
captureDR,bypassreg,bistreg,shiftDR_int,shiftIR,inicia_car2,systest,shiftDR_int2,shift
DR_int3;

wire    bypass=(iregl==BYPASS||iregl==SYS_TEST||iregl==EXPLODE),
        bist=(iregl==BIST),
        intest=(iregl==INTEST),
        extest=(iregl==EXTEST),
        sample_preload=(iregl==SAMPLE_PRELOAD),
        con_test=(iregl==CON_TEST),
        sumshift_load=(iregl==SUMSHIFT_LOAD),
        shiftDR=shiftDR_int&&(~bypass),
        clockDRin=(shiftDR||(captureDR&&(extest||sample_preload||sumshift_load)))&
&tclk,
```

```
        clockDRout=(shiftDR||(captureDR&&(intest||sample_preload)))&&tclk,
        clksum=shiftDR_int3&&con_test&&tclk,
        loadsum=captureDR&&sumshift_load,
        wkingbypass=con_test,
        resetsumb=~(sumshift_load&&(estado==UPDATE_DR)),
        scanin0=scanout0,
        scanin=tdi||inicia_car2,
        hab_updateDR_inpad=(intest||sample_preload||sumshift_load)&&(estado==UPD
ATE_DR),
        hab_updateDR_outpadb=~(((extest||sample_preload||sumshift_load)&&(estado=
=UPDATE_DR))||(shiftDR_int2&&con_test)),
        hab_updateDR_outextb=~((extest||sample_preload)&&(estado==UPDATE_DR))
,
        loadctrl=(sample_preload||sumshift_load)&&(estado==UPDATE_DR),
        scanbypassb=~(bypass||sample_preload),
        scanbypassintb=(scanbypassb&&~systest)||bist,
        pre_tdo=(systest)? scanout0 : scanout,
        tdo=(shiftDR_int||shiftIR)?
(~shiftIR&&((bist&&bistreg)||(bypass&&bypassreg)||(~bypass&&~bist&&pre_tdo)))||(i
reg[0]&&shiftIR) : 1'bz,
        inicio_car=bist&&(estado==RUN_TEST_IDLE);


//Reset do modulo
always @(trstb)
if (trstb==0)
begin
        assign estado='hF;
        assign iregl='hF;
        assign shiftDR_int=0;
        assign shiftIR=0;
        assign captureDR=0;
        assign systest=1;
        assign bistreg=1;
end
else
begin
        deassign estado;
        deassign iregl;
        deassign shiftDR_int;
        deassign shiftIR;
        deassign captureDR;
        deassign systest;
        deassign bistreg;
end


//Maquina de estado de controle do modulo
```

```
always @(posedge tclk)
#1 case(estado)
      //Exit2-DR
      EXIT2_DR:if (tms) estado=UPDATE_DR;
            else estado=SHIFT_DR;
      //Exit1-DR
      EXIT1_DR:if (tms) estado=UPDATE_DR;
            else estado=PAUSE_DR;
      //Shift-DR
      SHIFT_DR:if (tms) estado=EXIT1_DR;
      //Pause-DR
      PAUSE_DR:if (tms) estado=EXIT2_DR;
      //Select-IR-Scan
      SELECT_IR_SCAN: if (tms) estado=TEST_LOGIC_RESET;
                  else estado=CAPTURE_IR;
      //Updade-DR
      UPDATE_DR:if (tms) estado=SELECT_DR_SCAN;
            else estado=RUN_TEST_IDLE;
      //Capture-DR
      CAPTURE_DR:if (tms) estado=EXIT1_DR;
            else estado=SHIFT_DR;
      //Select-DR-Scan
      SELECT_DR_SCAN:if (tms) estado=SELECT_IR_SCAN;
            else estado=CAPTURE_DR;
      //Exit2-IR
      EXIT2_IR:if (tms) estado=UPDATE_IR;
            else estado=SHIFT_IR;
      //Exit1-IR
      EXIT1_IR:if (tms) estado=UPDATE_IR;
            else estado=PAUSE_IR;
      //Shift-IR
      SHIFT_IR:if (tms) estado=EXIT1_IR;
      //Pause-IR
      PAUSE_IR:if (tms) estado=EXIT2_IR;
      //Run-Test/Idle
      RUN_TEST_IDLE:if (tms) estado=SELECT_DR_SCAN;
      //Update-IR
      UPDATE_IR:if (tms) estado=SELECT_DR_SCAN;
            else estado=RUN_TEST_IDLE;
      //Capture-IR
      CAPTURE_IR:if (tms) estado=EXIT1_IR;
            else estado=SHIFT_IR;
      //Test-Logic-Reset
      TEST_LOGIC_RESET:if (~tms) estado=RUN_TEST_IDLE;
endcase

always @(posedge tclk)
begin
```

```
fork
if (estado==CAPTURE_IR)
        #1 ireg=4'b0001;
if (estado==SHIFT_IR)
        #1 ireg={tdi,ireg[3:1]};
if (shiftDR_int&&bypass)
        #1 bypassreg=tdi;
if (shiftDR_int&&bist)
        #1 bistreg=tdi;
if (iregl==SYS_TEST)
        #1 systest=1;
if (iregl==EXPLODE)
        #1 systest=0;
#1 shiftDR_int2=shiftDR_int;
join
end


always @(negedge tclk)
begin
#1 inicia_car2=inicia_car;
shiftDR_int3=shiftDR_int2;
shiftDR_int=((estado==SHIFT_DR)||inicia_car);
        //Seleciona Shift-DR
captureDR=(estado==CAPTURE_DR);
        //Seleciona Capture-DR
shiftIR=(estado==SHIFT_IR);
        //Seleciona Shift-IR
if (estado==UPDATE_IR)
        //No estado Update-IR a saida paralela do registrador de instrucao
        //recebe o conteudo do registrador de deslocamento
        iregl=ireg;
if (estado==TEST_LOGIC_RESET)
        begin
        //a saida paralela do registrador de instrucao recebe a instrucao BYPASS
        iregl='hF;
        end
end


endmodule
```

Descrição funcional do módulo de controle de teste do componente 4

```
// Verilog HDL for "bscan", "bscan_ctrl3" "_behavioral"

`timescale 1ns/100ps

module bscan_ctrl3
(tdi,tms,tclk,tdo,trstb,scanout,scanout0,inicio_car,inicia_car,clockDRin,clockDRout,shif
tDR,hab_updateDR_inpad,hab_updateDR_outpadb,hab_updateDR_outextb,scanbypassb
,scanbypassintb,scanin,scanin0,loadsum,clksum,resetsumb);

parameter     EXIT2_DR=0,    EXIT1_DR=1,    SHIFT_DR=2,
          PAUSE_DR=3,    SELECT_IR_SCAN=4,
          UPDATE_DR=5,   CAPTURE_DR=6,  SELECT_DR_SCAN=7,
          EXIT2_IR=8,    EXIT1_IR=9,    SHIFT_IR='hA,
          PAUSE_IR='hB,  RUN_TEST_IDLE='hC,
          UPDATE_IR='hD, CAPTURE_IR='hE, TEST_LOGIC_RESET='hF;

parameter     BYPASS=15, EXTEST=0,  INTEST=1,
              SAMPLE_PRELOAD=2,    SUMSHIFT_LOAD=8,      BIST=3,
              SYS_TEST=4,        EXPLODE=5,        CON_TEST=7;

input tdi,tms,tclk,trstb,scanout,scanout0,inicia_car;

output
tdo,inicio_car,clockDRin,clockDRout,shiftDR,hab_updateDR_inpad,hab_updateDR_ou
tpadb,hab_updateDR_outextb,scanbypassb,scanbypassintb,scanin,scanin0,loadsum,clks
um,resetsumb;

reg [3:0] ireg,iregl;
reg [3:0] estado;

reg
captureDR,bypassreg,bistreg,shiftDR_int,shiftIR,inicia_car2,systest,shiftDR_int2,shift
DR_int3;

wire    bypass=(iregl==BYPASS||iregl==SYS_TEST||iregl==EXPLODE),
        bist=(iregl==BIST),
        intest=(iregl==INTEST),
        extest=(iregl==EXTEST),
        sample_preload=(iregl==SAMPLE_PRELOAD),
        con_test=(iregl==CON_TEST),
        sumshift_load=(iregl==SUMSHIFT_LOAD),
        shiftDR=shiftDR_int&&(~bypass),
        clockDRin=(shiftDR||(captureDR&&(extest||sample_preload||sumshift_load)))&
&tclk,
```

```
        clockDRout=(shiftDR||(captureDR&&(intest||sample_preload)))&&tclk,
        clksum=shiftDR_int3&&con_test&&tclk,
        loadsum=captureDR&&sumshift_load,
        resetsumb=~(sumshift_load&&(estado==UPDATE_DR)),
        scanin0=scanout0,
        scanin=tdi||inicia_car2,
        hab_updateDR_inpad=(intest||sample_preload||sumshift_load)&&(estado==UPD
ATE_DR),
        hab_updateDR_outpadb=~(((extest||sample_preload||sumshift_load)&&(estado=
=UPDATE_DR))||(shiftDR_int2&&con_test)),
        hab_updateDR_outextb=~((extest||sample_preload)&&(estado==UPDATE_DR))
,
        scanbypassb=~(bypass||sample_preload),
        scanbypassintb=(scanbypassb&&~systest)||bist,
        pre_tdo=(systest)? scanout0 : scanout,
        tdo=(shiftDR_int||shiftIR)?
(~shiftIR&&((bist&&bistreg)||(bypass&&bypassreg)||(~bypass&&~bist&&pre_tdo)))||(i
reg[0]&&shiftIR) : 1'bz,
        inicio_car=bist&&(estado==RUN_TEST_IDLE);


//Reset do modulo
always @(trstb)
if (trstb==0)
begin
        assign estado='hF;
        assign iregl='hF;
        assign shiftDR_int=0;
        assign shiftIR=0;
        assign captureDR=0;
        assign systest=1;
        assign bistreg=1;
end
else
begin
        deassign estado;
        deassign iregl;
        deassign shiftDR_int;
        deassign shiftIR;
        deassign captureDR;
        deassign systest;
        deassign bistreg;
end


//Maquina de estado de controle do modulo
always @(posedge tclk)
#1 case(estado)
```

```
        //Exit2-DR
        EXIT2_DR:if (tms) estado=UPDATE_DR;
            else estado=SHIFT_DR;
        //Exit1-DR
        EXIT1_DR:if (tms) estado=UPDATE_DR;
            else estado=PAUSE_DR;
        //Shift-DR
        SHIFT_DR:if (tms) estado=EXIT1_DR;
        //Pause-DR
        PAUSE_DR:if (tms) estado=EXIT2_DR;
        //Select-IR-Scan
        SELECT_IR_SCAN: if (tms) estado=TEST_LOGIC_RESET;
                else estado=CAPTURE_IR;
        //Updade-DR
        UPDATE_DR:if (tms) estado=SELECT_DR_SCAN;
            else estado=RUN_TEST_IDLE;
        //Capture-DR
        CAPTURE_DR:if (tms) estado=EXIT1_DR;
            else estado=SHIFT_DR;
        //Select-DR-Scan
        SELECT_DR_SCAN:if (tms) estado=SELECT_IR_SCAN;
            else estado=CAPTURE_DR;
        //Exit2-IR
        EXIT2_IR:if (tms) estado=UPDATE_IR;
            else estado=SHIFT_IR;
        //Exit1-IR
        EXIT1_IR:if (tms) estado=UPDATE_IR;
            else estado=PAUSE_IR;
        //Shift-IR
        SHIFT_IR:if (tms) estado=EXIT1_IR;
        //Pause-IR
        PAUSE_IR:if (tms) estado=EXIT2_IR;
        //Run-Test/Idle
        RUN_TEST_IDLE:if (tms) estado=SELECT_DR_SCAN;
        //Update-IR
        UPDATE_IR:if (tms) estado=SELECT_DR_SCAN;
            else estado=RUN_TEST_IDLE;
        //Capture-IR
        CAPTURE_IR:if (tms) estado=EXIT1_IR;
            else estado=SHIFT_IR;
        //Test-Logic-Reset
        TEST_LOGIC_RESET:if (~tms) estado=RUN_TEST_IDLE;
endcase

always @(posedge tclk)
begin
fork
if (estado==CAPTURE_IR)
```

```
        #1 ireg=4'b0001;
if (estado==SHIFT_IR)
        #1 ireg={tdi,ireg[3:1]};
if (shiftDR_int&&bypass)
        #1 bypassreg=tdi;
if (shiftDR_int&&bist)
        #1 bistreg=tdi;
if (iregl==SYS_TEST)
        #1 systest=1;
if (iregl==EXPLODE)
        #1 systest=0;
#1 shiftDR_int2=shiftDR_int;
join
end


always @(negedge tclk)
begin
#1 inicia_car2=inicia_car;
shiftDR_int3=shiftDR_int2;
shiftDR_int=((estado==SHIFT_DR)||inicia_car);
        //Seleciona Shift-DR
captureDR=(estado==CAPTURE_DR);
        //Seleciona Capture-DR
shiftIR=(estado==SHIFT_IR);
        //Seleciona Shift-IR
if (estado==UPDATE_IR)
        //No estado Update-IR a saida paralela do registrador de instrucao
        //recebe o conteudo do registrador de deslocamento
        iregl=ireg;
if (estado==TEST_LOGIC_RESET)
        begin
        //a saida paralela do registrador de instrucao recebe a instrucao BYPASS
        iregl='hF;
        end
end


endmodule
```

Descrição funcional do módulo de controle de auto-teste dos componentes internos

```
// Verilog HDL for "bscan", "bist_ctrl" "_behavioral"

`timescale 1ns/100ps

module bist_ctrl
(clk,inicio,mem_car_in,init_car,address,clk_car_prog,car_prog_data,hab_car,hab_updat
eDR);

parameter TAM_CONT=8, TAM_PROGREG=16, TAM_CONT_PROG=5,
TAM_ADDRESS=2, TAM_N_TESTE=2, TAM_CONTATESTE=8, TAM_BS=12,
N_TOTAL=2;

input clk,inicio;
input [7:0] mem_car_in;

output init_car,clk_car_prog,car_prog_data,hab_car,hab_updateDR;
output [TAM_ADDRESS-1:0] address;

reg [7:0] shiftreg;
reg [2:0] estado;
reg [TAM_CONT_PROG-1:0] cont_prog;
//conta a tamanho do registro de programacao do CAR
reg [TAM_CONT-1:0] cont;
//conta a inicializacao do b.s. para o teste e depois a posicao de memoria
//para carregar a programacao de cada um dos testes
reg [TAM_N_TESTE-1:0] n_teste;
//numero de teste (diferentes programacoes) a serem realizados
reg [TAM_CONTATESTE-1:0] contateste;
//conta o teste para cada uma das programacoes
wire [TAM_ADDRESS-1:0] address;

wire   reset_ctrlb=~(estado==0||estado==2),
       //inicializa cont e contateste
       init_car=(estado==1),
       //comanda o inicio do car atraves do controle do b.s.
     carrega=(estado==2||estado==3),
       //vai ser combinado em hab para habilitar a contagem da programacao do CAR
     hab_car=(estado==4),
       //saida para colocar as celulas de entrada em modo CAR
       hab_updateDR=(estado==2||estado==4),
       //habilitando updateDR tem inicio o teste
       reset_cont_prog=hab_updateDR;
       //inicializa cont_prog antes da primeira programacao e depois toda
       //vez que esta' executando o teste
```

```
wire    load=(cont[2:0]==0),
        //carrega um byte da memoria
        shiftin = shiftreg[7] || init_car,
        //so' e' usado se for carregar dados na linha de scan
        hab = carrega || init_car,
        //habilita a contagem em cont
        fim_init_car = (cont == TAM_BS),
        //init_car poe todas as celulas do b.s. no estado logico "1"
        fim_carrega = (cont_prog == TAM_PROGREG);
        //sinaliza o final da carga da programacao do CAR

wire    clk_car_prog = ~clk && carrega,
        //clock para os registradores de programacao do CAR
        car_prog_data = shiftreg[7];

assign address=cont[TAM_ADDRESS+2:3];
//O endereco corresponde ao contador menos os tres bits menos significativos

// Quando inicio vai para "1" libera o circuito
// Deve ser mantido em "1" ate' o final do teste

always @(inicio)
if (inicio==0)
begin
        assign estado=0;
        assign n_teste=0;
end
else
begin
        deassign estado;
        deassign n_teste;
end


always @(reset_ctrlb)
if (reset_ctrlb==0)
begin
        assign contateste=1;
        assign cont=0;
end
else
begin
        deassign contateste;
        deassign cont;
end
```

```
always @(reset_cont_prog)
if (reset_cont_prog) assign cont_prog=0;
                else deassign cont_prog;


always @(negedge clk)
begin
        if (hab) cont=cont+1;
        cont_prog=cont_prog+1;
end


always @(posedge clk)
begin
if (load) #1 shiftreg=mem_car_in;
        else #1 shiftreg=shiftreg<<1;
end

always @(posedge clk)
case(estado)
        0:begin
                estado=1;
        end
// No estado 1 o sinal init_car deve ser aproveitado para habilitar a carga
// serial do b.s. e colocar a entrada serial no estado "1"
        1:begin
                if (fim_init_car) estado=2;
        end
// Nos estado 2 e 3 a programacao do CAR e' carregada, atraves dos sinais
// clk_car_prog e car_prog_data
        2:begin
                estado=3;
        end
        3:begin
                if (cont[0] && shiftreg[7]) contateste = contateste<<1;
                //Para cada celula habilitada, dobra o tamanho do teste para
                //esta determinada programacao
                if (fim_carrega)
                        begin
                        estado=4;
                        n_teste=n_teste+1;
                        end
        end
        4:begin
                if (contateste==2)
                begin
                if (n_teste==N_TOTAL)
                                        estado=7;
```

```
                    else estado=3;
            end
            #1 contateste=contateste-1;
        end
        //no estado 7 fica estavel
        7:estado=7;
        default:estado=7;
endcase

endmodule
```

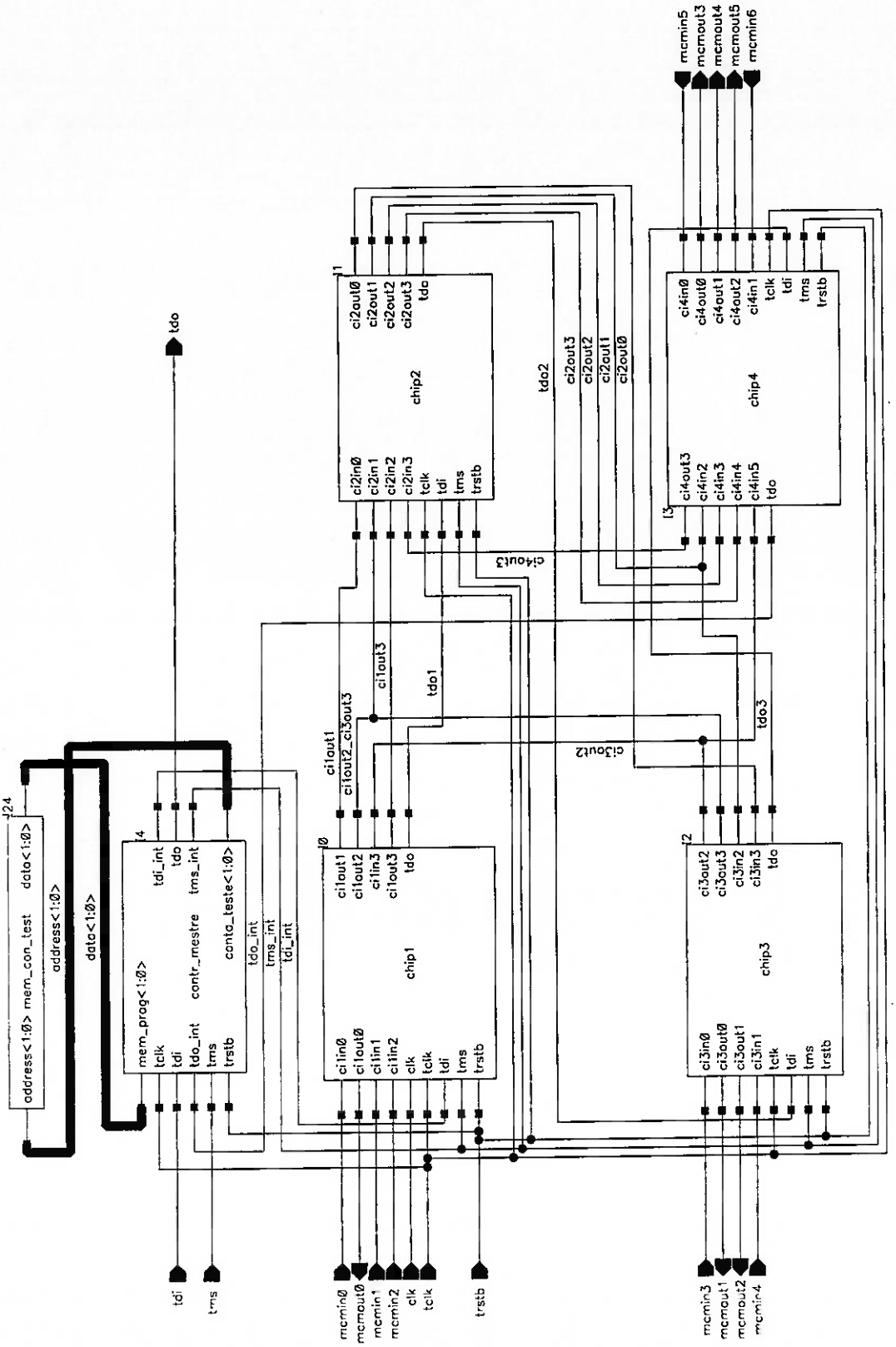


FIGURA A.1: Diagrama do MCM usado como exemplo

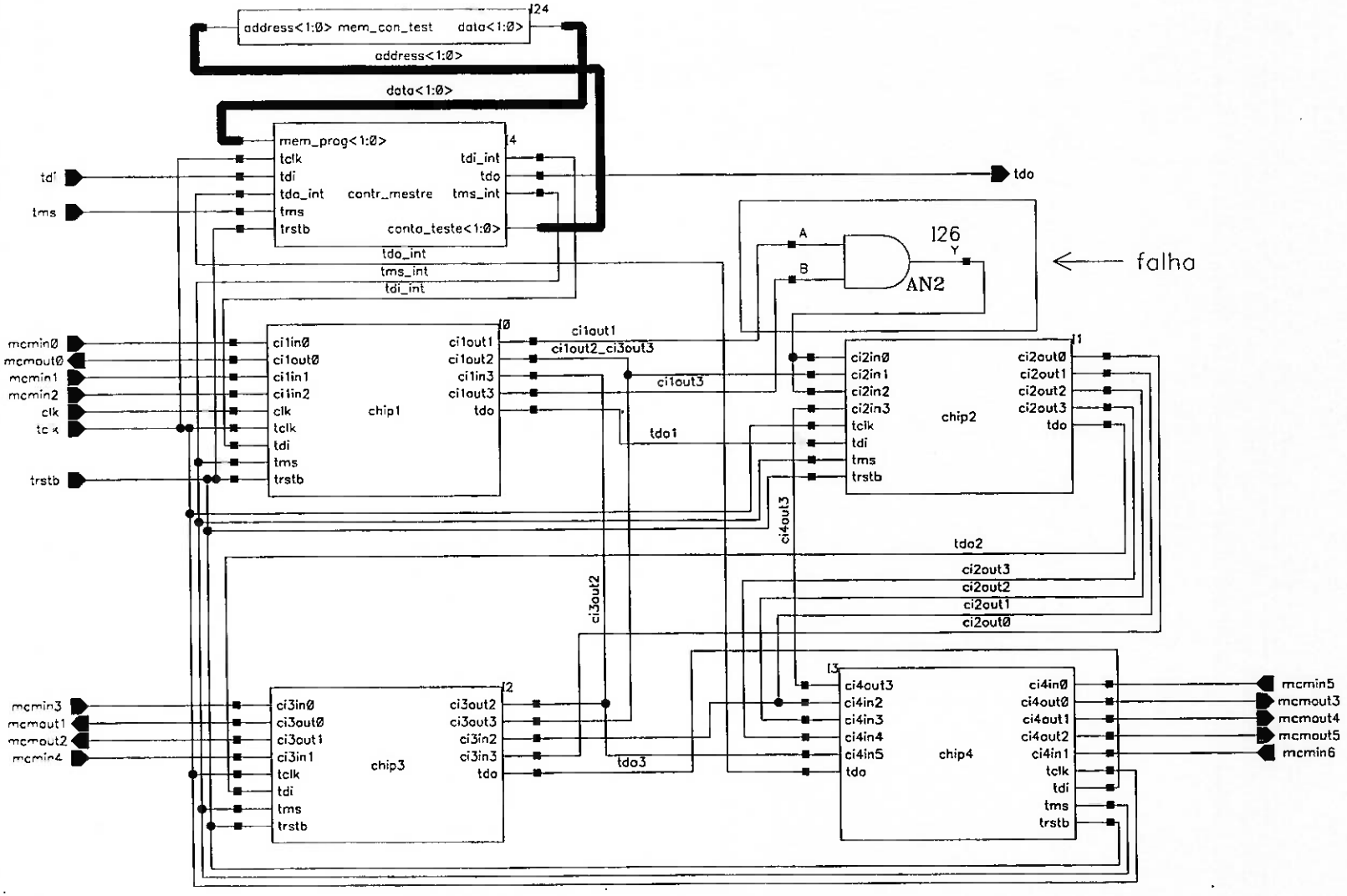


FIGURA A.2: MCM com curto tipo "E" entre duas trilhas

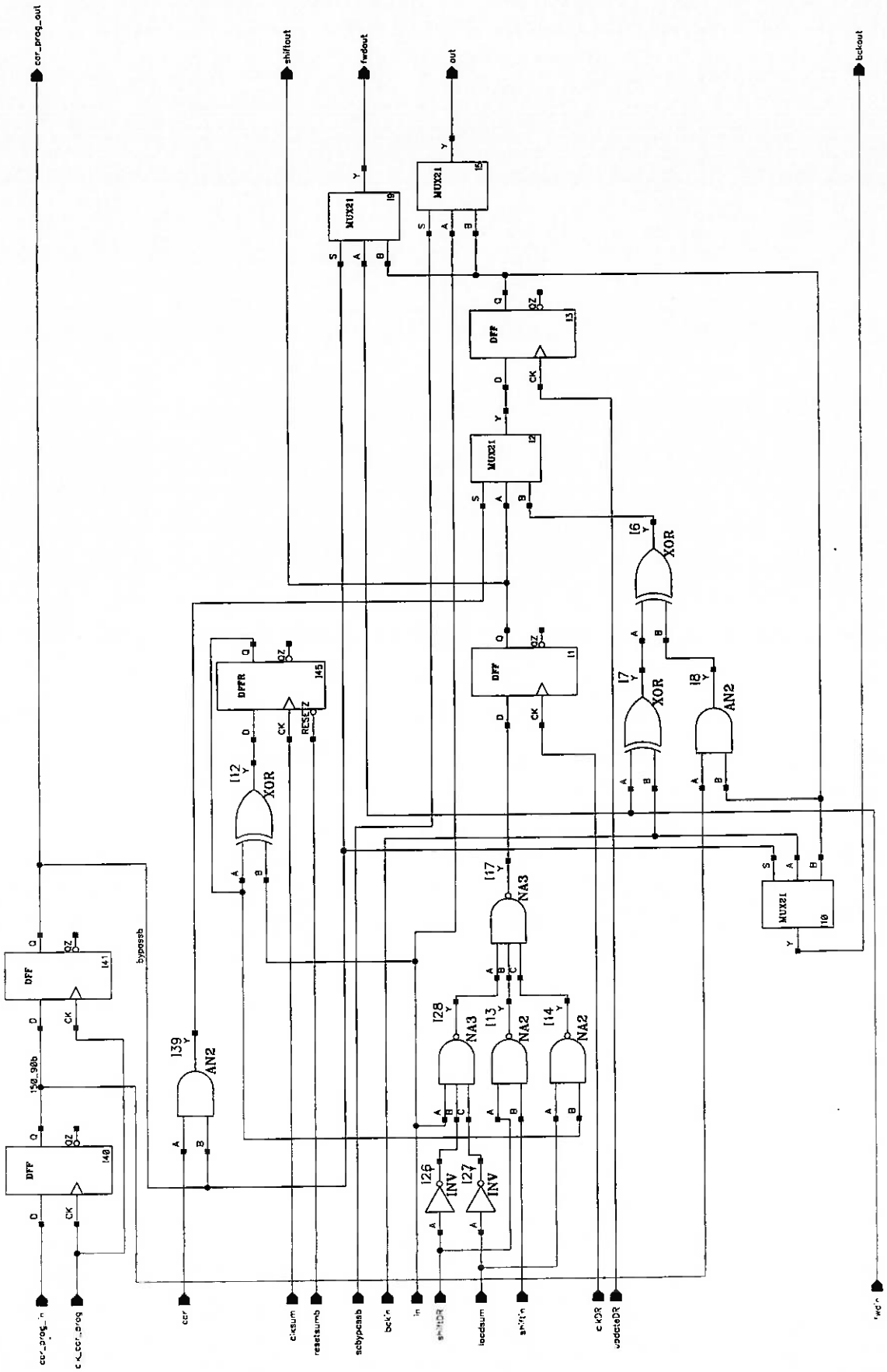


FIGURA A.3: Pino de entrada tipo Ø

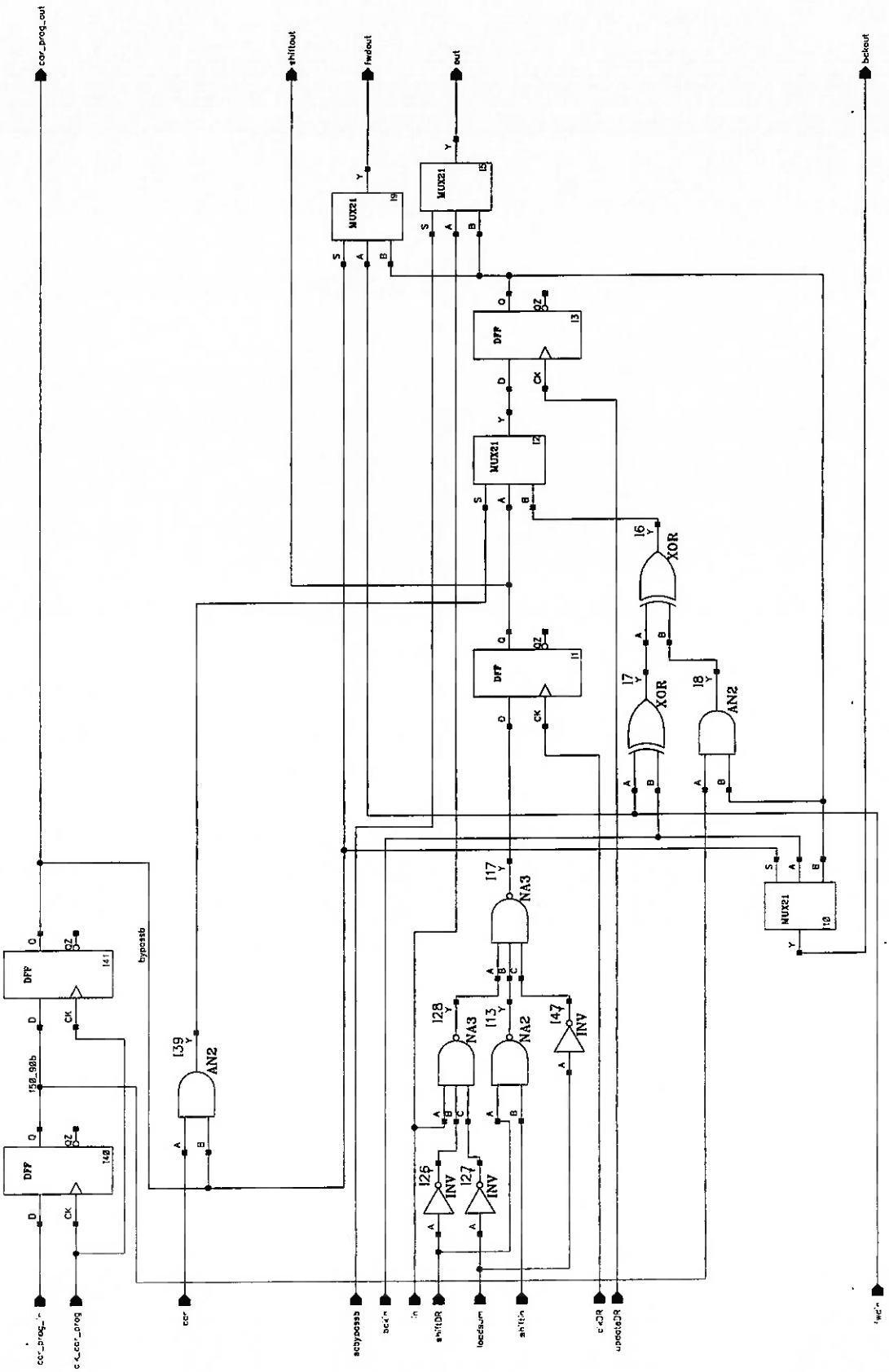


FIGURA A.4: Pino de entrada tipo 1

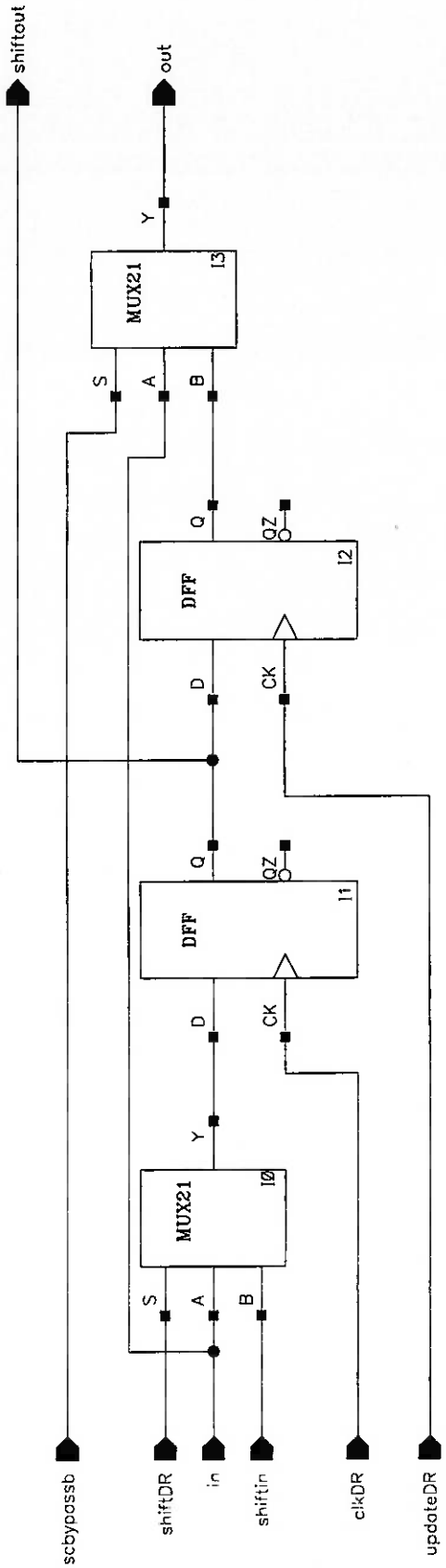


FIGURA A.5: Pino de saida

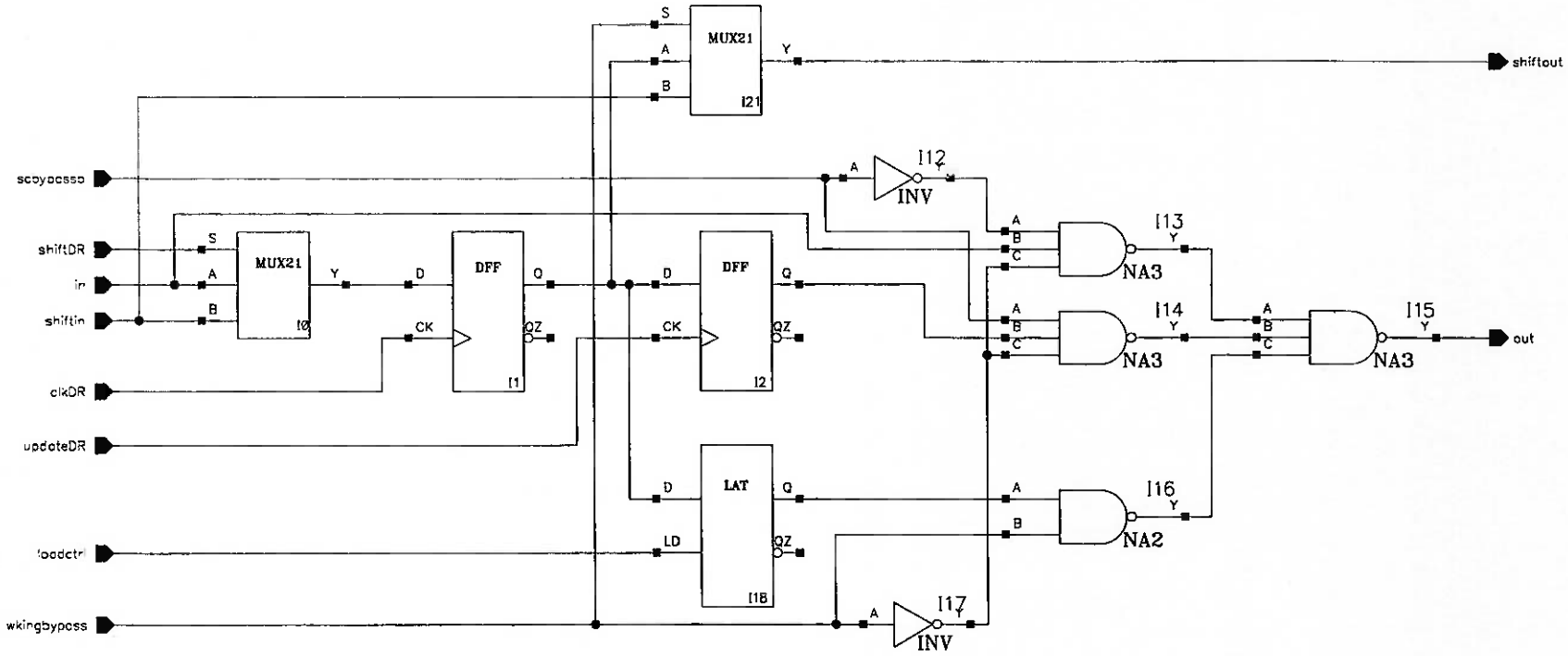


FIGURA A.6: Pino de controle de "tristate"

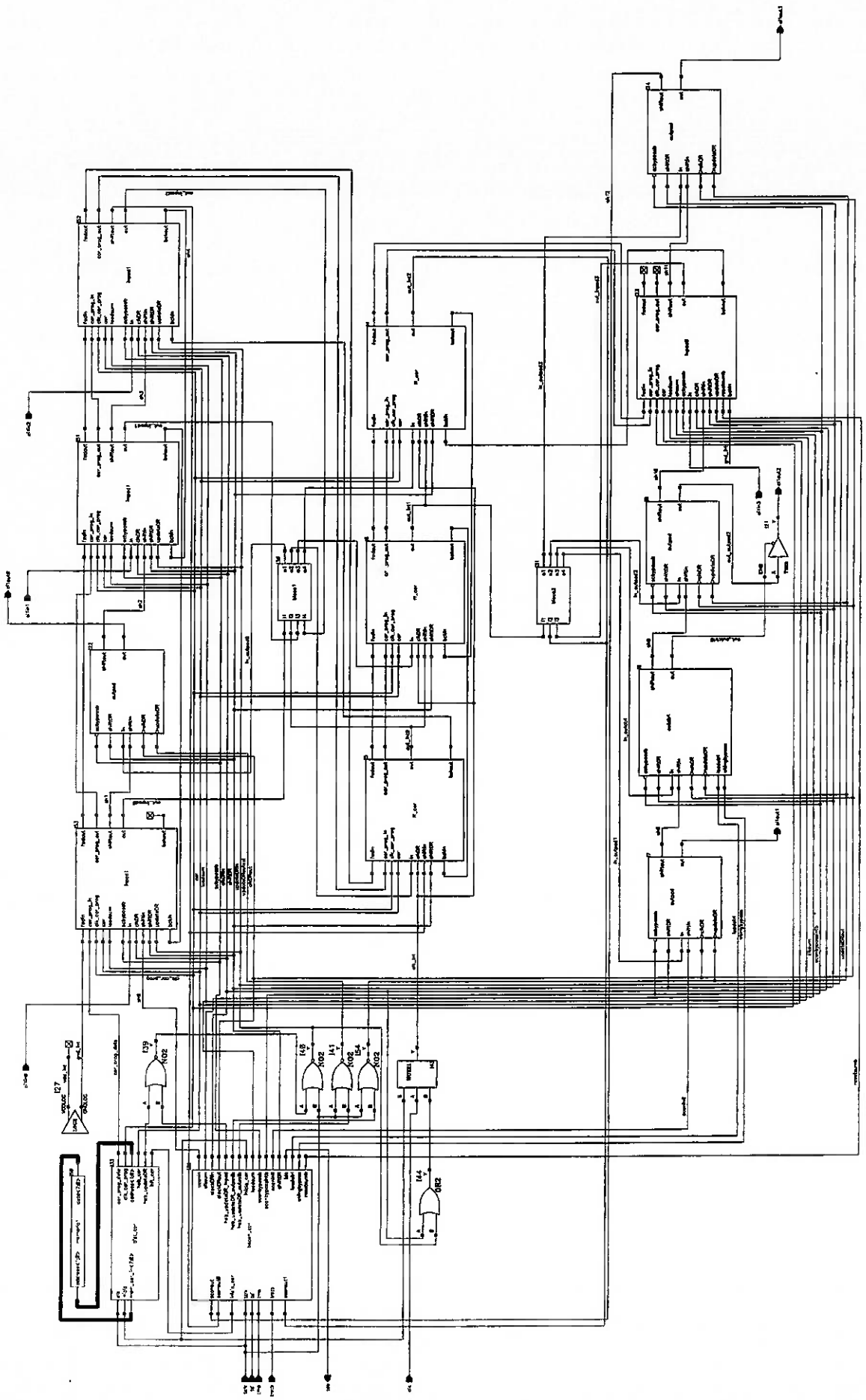


FIGURA A.7: Circuito Integrado N.1

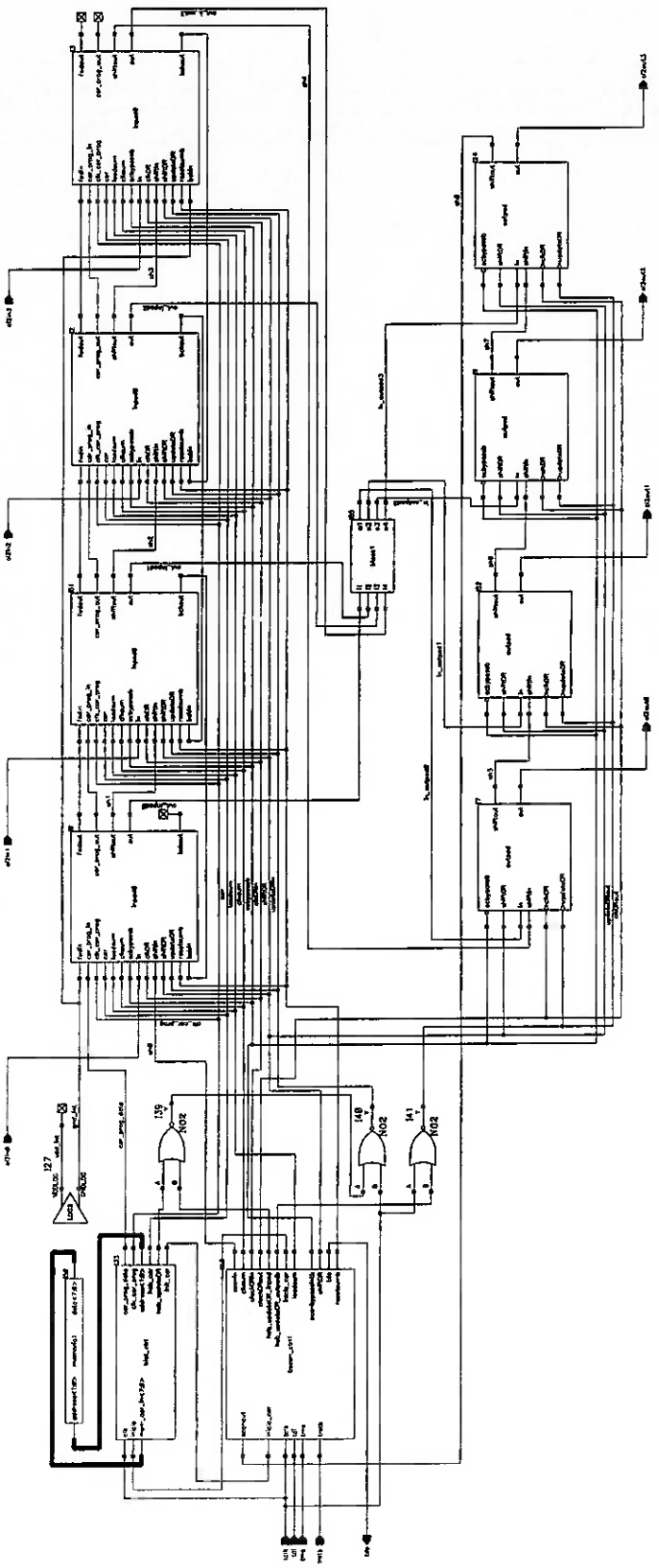


FIGURA A.8: Circuito Integrado N.2

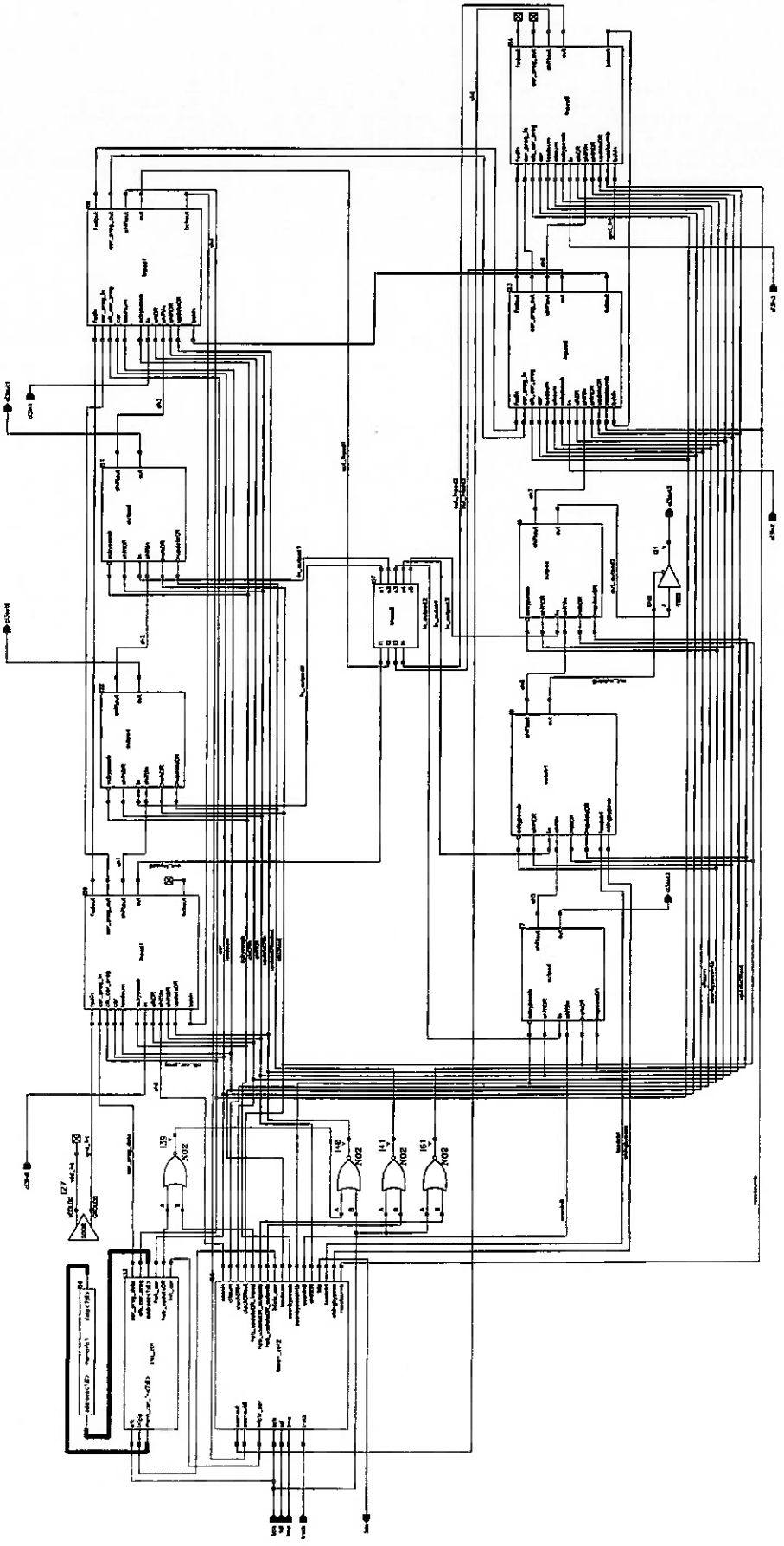


FIGURA A.9: Circuito Integrado N.3

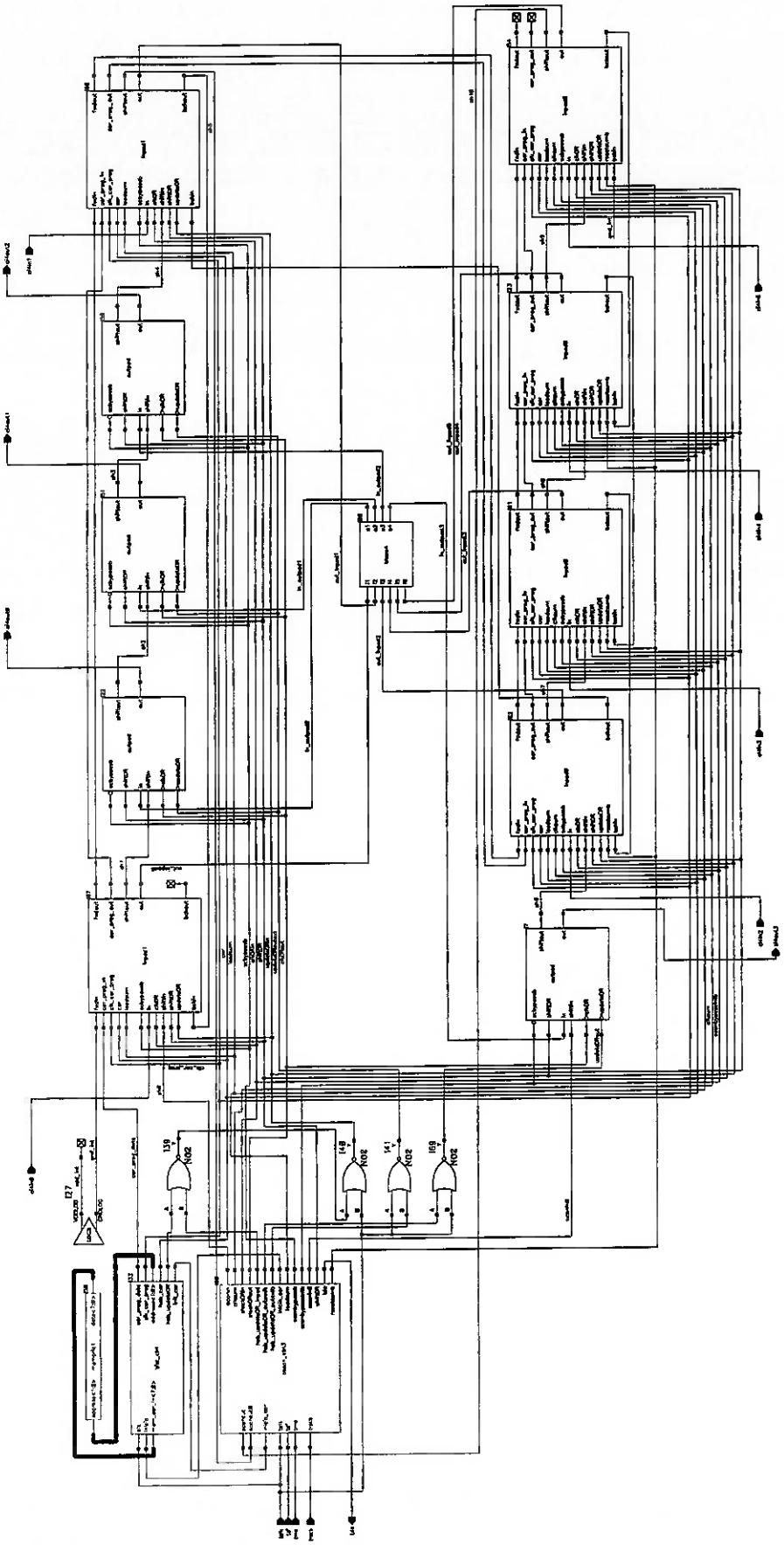


FIGURA A.10: Circuito Integrado N.4

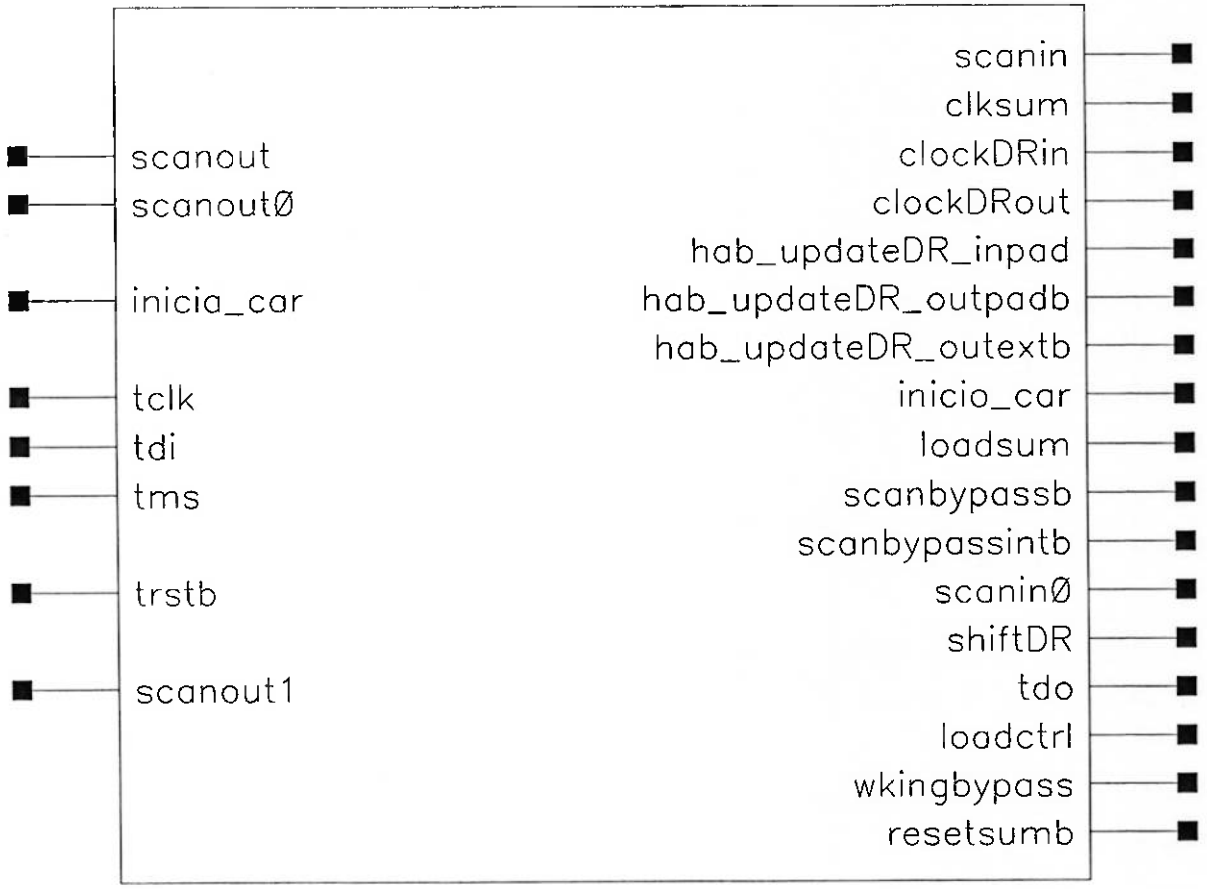


FIGURA A.11: Modulo de controle de teste do circuito integrado N.1

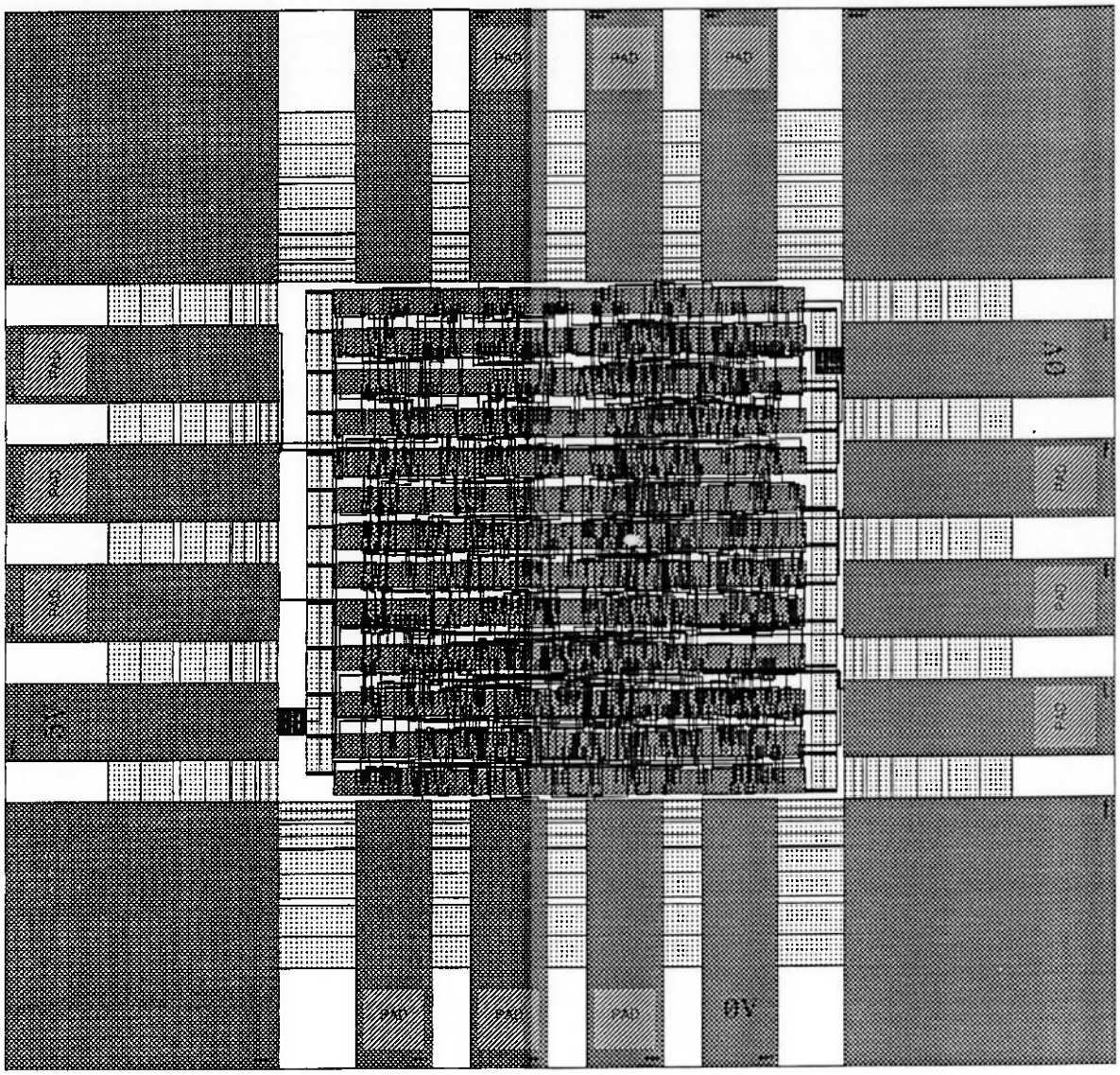


Figura A.12: Layout do circuito integrado "controle-mestre"

Bibliografia

[Abra90] Miron Abramovici, Melvin Breuer, Arthur Friedman, "Digital Systems Testing and Testable Design", IEEE Press, 1990

[Amaz88] José Roberto de A. Amazonas, "Estudo, Implementação e Aplicação de Técnicas Pseudo-Exaustivas ao Teste de Circuitos Integrados Digitais", Tese de Doutoramento apresentada à Escola Politécnica da USP, 1988

[AMI94] "0.8 micron CMOS Standard Cell Data Book", American Microsystems Inc., 1994

[bald92] John W. Balde, "Crisis in Technology: The Questionable U.S. Ability to Manufacture Thin-film Multichip Modules", Proceedings of the IEEE, Vol.80, No.12, Dezembro 1992, p.1995

[Brgl89] Franc Brglez, "Topics in Silicon Compilation - Synthesis for Testability", Course ECE692S, MCNC, 1989

[Cade92] A Vision Form Multichip Module Design in the Nineties, Cadence Design Systems, 1992

[Cho93] Jun-Dong Cho et al., "High Performance MCM Routing", IEEE Design & Test of Computers, Vol.10, No.4, Dezembro 1993, p.27

[Daum93] Wolfgang Daum, William E, Burdick Jr. e Raymond A. Fillion, "Overlay High-Density Interconnect: A Chips-First Multichip Module Technology", Computer, Vol.26, No.4, Abril 1993, p.23

[ES2a95] "ES2 ECPD07 Library Databook", European Silicon Structures, 1995

[ES2b95] "ES2 Process-Independent Library Databook", European Silicon Structures, 1995

[Fran93] Paul D. Franzon e Robert J. Evans, "A Multichip Module Design Process for Notebook Computers", Computer, Vol.26, No.4, Abril 1993, p.41

[Garr92] Philip Garrou, "Polymer Dieletrics for Multichip Module Packaging", Proceedings of the IEEE, Vol.80, No.12, Dezembro 1992, p.1942

[Gels86] Patrick P. Gelsinger, "Built In Self Test of the 80386", Intel Corporation, 1986

[Glas93] Lance A. Glasser, "A Road Map to ARPA Involvement in Electronic Packaging", Computer, Vol.26, No.4, Abril 1993, p.82

[Habi93] Claus M. Habiger e R. Mike Lea, "Hybrid-WSI: A Massively Parallel Computing Technology?", Computer, Vol.26, No.4, Abril 1993, p.50

[Hagg92] John K. Hagge e Russell J. Wagner, "High-Yield Assembly of Multichip Modules Through Known-Good IC's and Effective Test Strategies", Proceedings of the IEEE, Vol.80, No.12, Dezembro 1992, p.1965

[Hass92] Abu S. M. Hassan et al., "BIST of PCB Interconnects Using Boundary-Scan Architecture, IEEE Transactions on Computer-Aided Design, Vol.11, No.10, Outubro 1992, p.1278

[Hort89] P. D. Hortensius, H. C. Card e R. D. McLeod, "Parallel Random Number Generation for VLSI Using Cellular Automata", IEEE Transactions on Computer, Vol.38, Outubro 1989, No.10, p.1466

[IEEE90] IEEE Standard Test Access Port and Boundary-Scan Architecture, IEEE Std 1149.1-1990, 1990

[Inte91] 80386 Data Sheet, Intel Corporation, 1991

[Knas84] W. H. Knasenberger e L. Schaper, "Interconnection Costs of Various Substrates -- The Myth of Cheap Wire", ", IEEE Transactions on Components, Hybrids, Manufacturing Technology, Vol.7, No.3, Setembro 1984, p.261

[LaPo93] David P. LaPotin, Toufie R. Mazzawy e Marlin L. White, "Early Package Analysis: Considerations and Case Study", Computer, Vol.26, No.4, Abril 1993, p.30

[Lin83] Shu Lin e Daniel J. Costell, Jr. , "Error Control Coding: Fundamentals and Applications", Prentice-Hall, 1983

[Mess87] G. Messner, "Cost-Density Analysis of Interconnections", IEEE Transactions on Components, Hybrids, Manufacturing Technology, Vol.10, No.2, Junho 1987, p.143

[Scha92] Leonard W. Schaper, "Design of Multichip Modules", Proceedings of the IEEE, Vol.80, No.12, Dezembro 1992, p.1955

[Serr89] Micaela Serra e Terry Slater, "Tables of Linear Hybrid 90/150 Cellular Automata", Technical Report DCS-105-IR, University of Victoria, Dept. of Computer Science, Victoria, Canadá, Janeiro 1989

[Serr90] Micaela Serra et al., "The Analysis of One-Dimensional Linear Cellular Automata and Their Aliasing Properties", IEEE Transactions on Computer-Aided Design, Vol.9, No.7, Julho 1990, p.767

[SerS90] Micaela Serra e Terry Slater, "A Lanczos Algorithm in a Finite Field and Its Application", JCMCC, University of Victoria, Victoria, Canadá, No.7, 1990, p.11

[Stae93] David Staepelaere et al., "Surf: A Rubber-Band Routing System for Multichip Modules", IEEE Design & Test of Computers, Vol.10, No.4, Dezembro 1993, p.18

[Tumm92] Rao R. Tummala, "Multichip Packaging - A Tutorial", Proceedings of the IEEE, Vol.80, No.12, Dezembro 1992, p.1924

[Vemu93] Ranga Vemuri et al., "An Integrated Multicomponent Synthesis Environment for MCMs", Computer, Vol.26, No.4, Abril 1993, p.62

[Wang85] Laung-Terng Wang e Edward J. McCluskey, "Condensed Linear Feedback Shift Register (LFSR) Testing -- A Pseudo-Exhaustive Test Technique", CRC Technical Report No.85-24, Center for Realiable Computing, Stanford University, Dezembro 1985