UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL - UFRGS
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

Inserção de Testabilidade em um Núcleo Pré-projetado de um Microcontrolador 8051 Fonte Compatível

por

Eduardo Santos Back

Trabalho de Conclusão submetido à avaliação,
como requisito parcial para a obtenção do grau de Mestre
em Informática

Prof. Dr. Marcelo Soares Lubaszewski
Orientador

Porto Alegre, abril de 2002

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Back, Eduardo Santos
Inserção de Testabilidade em um Núcleo Pré-projetado do Microcontrolador 8051 Fonte Compatível / por Eduardo Santos Back. – Porto Alegre: PPGC da UFRGS, 2002.
178 f.:il.

Trabalho de Conclusão (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR – RS, 2001.
Orientador: Lubaszewski, Marcelo Soares

1. Teste. 2. Boundary-Scan 3. IEEE1149.1 4. JTAG 5. Microcontrolador 8051. I. Lubaszewski, Marcelo Soares. II. Título.

Agradecimentos

Inicio com um agradecimento especial a meus pais, que sempre me deram apoio nos mais diversos momentos. À minha esposa, incansável auxiliadora e incentivadora, especialmente nas horas mais complicadas e por não me deixar desistir. Espero eu mesmo agir da mesma forma quando chegar a hora.

Ao meu orientador, pela paciência e dedicação. Sempre foi alguém que me elevou o ânimo, mostrando que as coisas podem ser muito mais fáceis do que parecem em um primeiro momento.

Finalmente à Érika Cota, minha guru nos assuntos ligados a VHDL e ao ambiente aqui utilizado, por todo o apoio e gentileza em me ajudar, mesmo fisicamente distante.

E aos demais que de alguma forma ajudaram, saibam que, apesar de não haver mais nomes aqui citados, foram muito importantes para a conclusão deste trabalho.

Eduardo Santos Back

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

No intuito de validar seus projetos de sistemas integrados, o Grupo de Microeletrônica da UFRGS tem investido na inserção de estruturas de teste nos núcleos de hardware que tem desenvolvido. Um exemplo de tal tipo de sistema é a "caneta tradutora", especificada e parcialmente desenvolvida por Denis Franco. Esta caneta se utiliza de um microcontrolador 8051 descrito em VHDL, o qual ainda carece de estruturas dedicadas com funções orientadas à testabilidade.

Este trabalho exemplifica a integração de teste em um circuito eletrônico pré-projetado. Neste caso específico, foi utilizado o microcontrolador 8051 fonte compatível que será inserido no contexto da caneta tradutora. O método utilizado apoiou-se na norma IEEE1149.1, destinada a definir uma infra-estrutura baseada na técnica do *boundary scan* para o teste de placas de circuito impresso.

São apresentadas características de testabilidade desenvolvidas para o microcontrolador, utilizando-se a técnica do *boundary scan* em sua periferia e a técnica do *scan path* em seu núcleo. A inserção destas características de teste facilita a depuração e testes em nível de sistema, imaginando-se o sistema como algo maior, fazendo parte do sistema da caneta tradutora como um todo.

São elaborados exemplos de testes, demonstrando a funcionalidade do circuito de teste inserido neste núcleo e a possibilidade de detecção de falhas em pontos distintos do sistema.

Finalmente, avalia-se o custo associado à integração desta infra-estrutura de teste, tanto em termos de acréscimo de área em silício, quanto em termos de degradação de desempenho do sistema.

Palavras-chave: Teste, Boundary-Scan, IEEE1149.1, JTAG, Scan Path, Microcontrolador 8051.

TITLE: "INSERTION OF TESTABILITY IN A PRE-PROJECTED MICROCONTROLLER 8051 CORE SOURCE COMPATIBLE"

Abstract

Aiming to validate its integrated system designs, the UFRGS Microelectronics Group has been making an effort to insert test structures into the hardware cores that it has been developping. An example of this kind of system is the "translating pen", specified and partially developped by Denis Franco. This pen uses a 8051 microcontroller described in VHDL, which still lacks of dedicated structures for testability purposes.

This work illustrates the integration of testing into a pre-designed integrated circuit. In this particular case, the 8051 compatible microcontroller was used that will be inserted into the context of the translating pen. The method used is based on the IEEE1149.1 standard, devoted to define a boundary scan infrastructure for testing printed circuit boards.

We present the testability features integrated into the microcontroller that use the boundary scan and the internal scan path techniques. The insertion of these test capabilities ease debugging and system level tests, specially when considering it as a part of the translating pen system.

We present test examples, showing the functionality of the core test circuitry and its capability of detecting faults in different internal points.

Finally, we evaluate the costs associated to the integration of this test infrastructure, both in terms of silicum area overhead and performance degradation.

Keywords: Test, Boundary-Scan, IEEE1149.1, JTAG, Scan Path, 8051 Microcontroller.

1 Introdução

A cada dia, tornam-se mais importantes as características de testabilidade dos circuitos eletrônicos produzidos em série. Com a crescente miniaturização dos componentes e placas, a tarefa de testar torna-se cada vez mais complexa, originando o problema da testabilidade frente à miniaturização.

A complexidade dos sistemas em um único chip (SOC - *System on a chip*) e a tecnologia submicrônica (VDSM - *Very deep submicron*) norteiam o desenvolvimento de semicondutores, fazendo com que os métodos de teste se tornem rapidamente inadequados e com alto custo. Atualmente, a demanda do mercado é por equipamentos eletrônicos cada vez mais baratos, menores e mais rápidos, com certeza também de melhor qualidade. Para garantir este pontos básicos, há a necessidade de melhores testes e procedimentos de diagnose.

Para ajudar a resolver estes problemas, normas tem sido criadas a fim de definir padrões de integração de capacidades de testabilidade em placas e em componentes eletrônicos. Desta forma, facilita-se e até mesmo, em casos mais graves, permite-se que possam ser feitos testes que antes eram impossíveis, tamanha a miniaturização dos componentes e placas dos sistemas eletrônicos atuais.

Os semicondutores são considerados como o início, o ponto de partida para os produtos modernos, mesmo os tradicionalmente não-eletrônicos. Cada vez mais baratos, mais e mais funcionalidades são acrescidas, aumentando o seu grau de complexidade e, conseqüentemente, as necessidades de testes mais elaborados. Sendo assim, pode-se assumir que o teste também ajuda no desenvolvimento da indústria eletrônica.

Problemas que antes eram solucionados com a simples técnica de acesso baseada em “camas de pregos” (*Bed-of-Nails*), atualmente necessitam de técnicas industriais mais avançadas, que permitam a rápida identificação de falhas o mais cedo possível. Estas falhas normalmente se concentram nas interconexões dos componentes, soldas ruins por exemplo, e não nos componentes, uma vez que já devem ter sido previamente testados antes de serem montados nas placas. Ou seja, componentes que individualmente funcionam e que, após serem integrados a outros em uma placa de circuito impresso, não apresentam a funcionalidade desejada, podem simplesmente estar apresentando sintomas relacionados à má interconexão entre as diversas partes.

Partindo da hipótese de que há a funcionalidade do teste na placa e em seus circuitos, há ainda a necessidade de que possam ser feitos os devidos testes com o auxílio de computadores. Neste contexto as normas IEEE se inserem, padronizando os

circuitos com funcionalidades mínimas de teste, bem como a forma de acessar estes componentes a partir de um equipamento testador externo.

Neste contexto, o microcontrolador 8051 torna-se objeto desta dissertação, onde o núcleo do processador é estudado e modificado. Exemplifica-se a inserção de circuitos auxiliares, compatibilizando o microcontrolador 8051 de acordo com a norma IEEE1149.1[IEE90], visto que o microcontrolador 8051 não possui originalmente em sua organização estruturas dedicadas com funções orientadas à testabilidade.

No capítulo 2, são apresentadas considerações a respeito de teste e projeto de circuitos digitais orientados ao teste. Também é apresentado o contexto onde este trabalho está inserido.

No capítulo 3, tem-se uma explicação da norma IEEE1149.1, base para o tipo de teste inserido no circuito do trabalho. São apresentados os registradores disponíveis pela norma, o controlador TAP e os tipos de testes possíveis.

O capítulo 4 apresenta as especificações e detalhes inerentes a esta implementação, decisões que foram tomadas e os porquês destas. Apresenta também o diagrama em blocos do 8051 com e sem teste, detalhes da cadeia de varredura interna e as características da célula básica BS, incluindo as instruções implementadas.

No capítulo 5, tem-se alguns experimentos práticos sobre a implementação desenvolvida, partindo de um comparativo entre o 8051 original e com teste, além de diversos outros circuitos que fazem parte do 8051 completo com teste. Após isto, há simulações do teste de memória e diversos exemplos de teste, incluindo o teste dos registradores que fazem parte da cadeia de varredura interna.

Finalmente, no capítulo 6, são apresentadas as conclusões deste trabalho.

2 Teste e projeto visando o teste de sistemas digitais

2.1 Testando circuitos integrados: tendências e soluções

Em substituição parcial às placas de circuito impresso, o projeto de sistemas eletrônicos vem assumindo uma nova forma, que consiste na integração de diversos sub-projetos em um projeto maior de um único chip, implementado portanto sobre um mesmo substrato de silício. São os chamados projetos baseados em núcleos de hardware, ou *cores*. Os sub-projetos são como circuitos independentes, projetados absolutamente em separado, mas passíveis de comunicação com outros núcleos de hardware. O reaproveitamento de núcleos é fator interessante, mas alguns problemas estão relacionados a esta organização, entre eles a dificuldade de comunicação entre os *cores* (núcleos, módulos) de fornecedores distintos e a necessidade de testes dos núcleos quando integrados em silício.

Os núcleos de hardware atualmente são disponibilizados pelos mais diversos fabricantes, facilitando a sua utilização no que se refere ao custo, que tende a ser reduzido com o passar do tempo. Tarefas executadas por diversos circuitos integrados agora podem ser agregadas em um único chip, porém isto agrega maior complexidade em uma única pastilha, que pode conter milhões de transistores.

Os testes em circuitos integrados iniciam com as provas nos wafers de circuitos, que serão divididos, formando diversos chips. Os circuitos integrados (CIs) são testados a cada etapa de sua vida útil: após a produção dos primeiros protótipos, após a fabricação em série, após a montagem em placa de circuito impresso e durante sua utilização, por razões de manutenção preventiva ou corretiva. Como a tecnologia segue evoluindo, continua a confrontar a tendência de: maior complexidade, maior desempenho e maior densidade. Para permitir estes três aspectos, mudanças fundamentais são esperadas na forma de realização de CIs, como o projeto, encapsulamento, processo de elaboração do silício, processo de montagem em placa, etc. Estas mudanças têm impacto direto nos métodos de teste, ferramentas e equipamentos adotados.

A dificuldade de acesso aos transistores internos é um problema bastante sério nos testes de chips. Há uma disparidade entre o crescimento interno do relógio destes chips e a capacidade de entrada e saída, fazendo com que testes à velocidade real do chip tornem-se muito difíceis ou praticamente impossíveis. A capacidade de banda de I/O tem um maior impacto sobre os métodos de teste de chips. Como os equipamentos

de teste são de tecnologias sempre anteriores às dos chips que estão sendo desenvolvidos, é muito mais difícil garantir os seus testes de forma adequada.

Os equipamentos de teste, além de caros, estão baseados em tecnologias anteriores àquelas utilizadas nos circuitos a serem testados. É aqui que surge o conceito de teste no próprio chip. É o projeto visando a testabilidade (*design for testability*). Significa que, conforme as necessidades de teste, componentes específicos são colocados dentro do chip, como parte integrante de seus circuitos. Uma destas técnicas é o *scan path*. Consiste em ter-se um caminho serial dentro do chip, de forma que possam ser testadas as suas funcionalidades por meio da introdução de dados específicos em determinados pinos. Por outro lado, quanto mais pinos, maior o custo.

Projetos de *system on a chip* (SOC), ou sistema em um único chip, podem implicar no reuso de blocos funcionais pré-projetados, também chamados componentes virtuais, IP (*Intellectual Properties*, ou propriedade intelectual). Estes núcleos de funções podem ser adquiridos com funções pré-definidas de teste e serem reutilizados em diferentes circuitos. Uma linguagem de teste foi definida pelo IEEE, a CTL (*Core Test Language*) [KAP99], a fim de facilitar e simplificar a integração de núcleos (*cores*). Os núcleos mais usados hoje são de memórias. Estas memórias têm alguma redundância e, em caso de falha em testes, são reconfiguradas de forma a continuarem com sua capacidade original.

Com o contínuo incremento da velocidade dos CIs, defeitos relacionados à velocidade são cada vez mais importantes. Um teste externo nestes casos não é eficiente, além de caro. É o caso já citado de que os testadores estão sempre atrasados em relação aos seus testados. Enquanto a precisão de um testador externo aumenta da ordem de 12% ao ano, a velocidade interna dos chips aumenta 30% no mesmo período [ZOR99]. Desta forma, ainda mais importante é o teste embutido, interno ao chip.

O teste externo será utilizado juntamente com o interno. Mas cada vez mais partes que eram testadas externamente estão passando a ter seus testes implementados internamente ao chip, aumentando a confiabilidade e reduzindo custos em relação ao teste externo.

2.2 Os diferentes métodos de teste e de projeto visando o teste

Partindo do início do projeto, qualquer circuito passa por várias etapas de verificação, quais sejam: fase de depuração do projeto, teste de produção e teste de manutenção. Nestas etapas é possível identificar e isolar os dispositivos que contém falhas ou ainda substituir partes com problemas. Estes são os testes *off-line*, ou seja, quando o circuito em teste não está em funcionamento. Outro tipo de teste é o *on-line*,

quando o dispositivo está em funcionamento e, neste caso, os próprios estímulos funcionais, devidamente codificados segundo códigos detectores e/ou corretores de erros, são utilizados para a aplicação do teste e para a verificação da validade das operações realizadas.

No contexto de testes *off-line* e com o intuito de verificar-se o correto funcionamento do circuito na velocidade nominal de operação, temos o teste funcional. Este, de fato, confronta o comportamento observado com a especificação do sistema e, por isso, é capaz de identificar problemas de desempenho. Por outro lado, tem-se o teste estrutural, onde procura-se confrontar a implementação física ao esquemático originalmente previsto para o circuito. Este, tentando abstrair ao máximo a funcionalidade do circuito, verifica a presença ou ausência de problemas neste (*fault-based*).

Mas para ter-se testes eficientes são necessários modelos de falhas realistas. Grande parte dos defeitos são relacionados ao substrato de silício onde as estruturas foram fabricadas, como por exemplo impurezas no material utilizado para produzir-se os wafers. Em outros casos, os defeitos ocorrem durante alguma etapa do processo de fabricação, como o deslocamento das máscaras ou a presença de partículas de poeira na sala onde os circuitos estão sendo produzidos. Neste caso, têm-se as chamadas falhas múltiplas, onde um defeito afeta várias partes do circuito. Outros problemas podem acontecer durante a vida útil do circuito, ocasionados por fatores térmicos ou fenômenos mecânicos, e neste caso têm-se falhas simples que afetam, em geral, dispositivos individuais. Pode-se estabelecer ainda uma diferenciação entre os tipos de falhas no que diz respeito a permanência de resultados errôneos: falhas transitórias, que ocorrem devido a fenômenos aleatórios e de curta duração, como interferências eletromagnéticas; falhas intermitentes, que tem um comportamento lógico determinístico, fornecendo sempre o mesmo tipo de erro, mas que não se manifestam sempre; e falhas permanentes, que uma vez instaladas permanecem, como interconexões abertas ou em curto-circuito.

Alguns exemplos de modelos de falhas utilizados em circuitos digitais são: entradas e saídas de portas lógicas sempre em nível ‘1’ ou ‘0’ (*stuck-at*), transistores sempre conduzindo (*stuck-on*) ou que nunca conduzem (*stuck-open*) e atrasos em portas na subida ou descida do sinal (*slow-to-rise* ou *slow-to-fall*) representam falhas de dispositivos. Curto-circuitos entre conexões (*bridging*), fios desconectados ou ainda atrasos no caminho crítico do circuito representam falhas de interconexão.

O processo de elaboração de uma bateria de testes a ser aplicada a um determinado circuito precisa de ferramentas computacionais que auxiliem a contornar a complexidade da obtenção de estímulos adequados de teste. A mais elementar das ferramentas é a simulação de falhas. Esta consiste na simulação do comportamento do circuito injetando-se na sua descrição falhas que pertençam ao modelo adotado, e na comparação dos resultados obtidos na saída do circuito com e sem falhas. O objetivo é verificar se estas falhas são detectadas a partir de determinados estímulos de entrada.

Uma ferramenta mais elaborada seria capaz de gerar, de maneira automática, os vetores de entrada necessários ao teste. O problema consiste em encontrar um conjunto de estímulos de entrada que seja suficiente para garantir a máxima cobertura de falhas, ou seja, para garantir a detecção do maior número possível de falhas que fazem parte do modelo adotado. Se o objetivo é diagnóstico, torna-se necessária a distinção entre possíveis falhas que apresentem o mesmo comportamento na saída do circuito, as ditas falhas equivalentes.

Para a geração automática de vetores de teste podem ser utilizadas as abordagens exaustiva, pseudo-aleatória ou determinística. Na primeira, considera-se a geração e aplicação de todos os vetores de teste possíveis. Desta forma, tem-se a maior cobertura possível, porém esta abordagem só deve ser aplicada a circuitos muito pequenos. Na segunda abordagem, geradores especiais produzem vetores pseudo-aleatórios a partir de uma semente, um vetor de inicialização. Este método gera uma boa cobertura de falhas em um tempo razoável de teste, porém para circuitos onde poucos vetores gerariam seqüências suficientes, as seqüências de testes obtidas de maneira pseudo-aleatória podem ser muito longas. Na abordagem determinística, há dois métodos: algébrico e topológico. No método algébrico, vetores de teste são computados a partir de expressões booleanas que descrevem o funcionamento do circuito em teste. No método topológico, os vetores de teste são derivados a partir da estrutura do circuito.

Ainda que se utilize uma ferramenta que gere estímulos de teste, algumas falhas podem ser difíceis de detectar, aumentando o tempo e o custo do teste para se atingir uma melhor cobertura de falhas. Sendo assim, o re-projeto de partes do circuito pode ser vantajoso, melhorando a acessibilidade a partes internas deste, o chamado projeto visando a testabilidade, ou DFT (*Design for Testability*). Aqui se enquadra a técnica do *scan path*, já mencionada na seção anterior.

Devido à crescente miniaturização, pode-se considerar outra possibilidade de projeto visando o teste, que é a concepção de circuitos auto-testáveis, ou com BIST (*Built In Self Test*). A idéia básica aqui consiste em transferir módulos originalmente implementados no testador para dentro do próprio circuito a testar. Neste caso, blocos geradores de estímulos e analisadores das respostas de teste são integrados no circuito. Os métodos de BIST têm, é claro, um custo associado, devido ao aumento da área do circuito e, quanto mais transistores ele tiver, maior é a probabilidade de falhas ocorrerem neste circuito. Ou seja, deve ser pesado o custo deste incremento de unidades funcionais frente à provável diminuição de peças com defeito saindo de uma linha de produção. Além do mais, apesar da diminuição dos custos de teste, a integração de estruturas BIST em muitos casos ocasiona a inserção de atrasos no circuito, que devem ser verificados, analisados e, na medida do possível, eliminados.

2.3 O contexto deste trabalho

Um núcleo de hardware do microcontrolador 8051 vem sendo utilizado em diversos trabalhos. Em 1996, a primeira descrição VHDL deste microprocessador foi realizada e utilizada como estudo de caso em [CAR96]. Após, esta descrição inicial tem sido modificada para trabalhos distintos tais como a sua prototipação em FPGA [SIL97], a implementação de uma versão auto-testável em FPGA [COT99a], implementação de uma versão protegida em relação a falhas de radiação [LIM2000] e, finalmente, como parte integrante de um protótipo do sistema integrado apresentado em [FRA2000].

Motivados pela evolução do projeto de sistemas eletrônicos e de seu teste, um conjunto de pesquisadores do Grupo de Microeletrônica vem utilizando o sistema descrito em [FRA2000] como exemplo para o desenvolvimento de técnicas de teste de sistemas integrados baseados em núcleos. O sistema em questão é um protótipo de uma "caneta tradutora", que pode ser visto na figura 2.1.

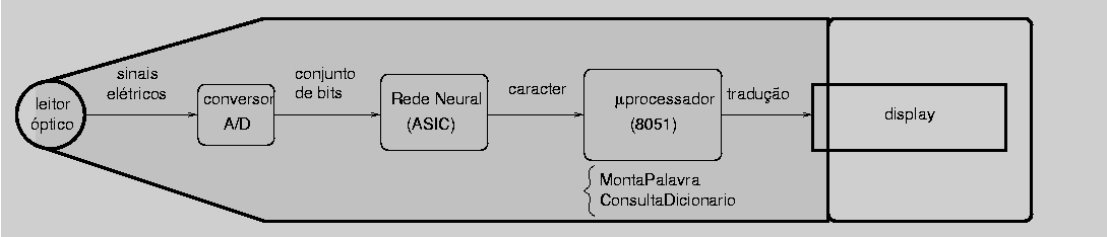

FIGURA 2.1 - Protótipo de caneta tradutora

Este sistema é composto de um dispositivo ótico que "lê" um conjunto de caracteres (um *string*), um conversor A/D que transforma estes sinais analógicos em um conjunto de dados digitais, uma rede neural que identifica cada caracter do conjunto e, finalmente, um microprocessador que monta o *string* e realiza uma pesquisa em um dicionário previamente armazenado, realizando a tradução, a apresentação da palavra lida e sua correspondente em Português.

O sistema como um todo é formado por diversos sub-sistemas ou núcleos. Devido à sua heterogeneidade (partes analógicas, conversor, ASIC, etc.), este exemplo permite um estudo bastante realístico dos sistemas integrados atuais. Uma estratégia de teste do sistema como um todo está sendo estudada para este exemplo. Esta estratégia deve considerar não só o teste do sistema completo, mas também o teste de cada núcleo de hardware em particular [ZOR98].

A principal característica da maior parte das implementações do 8051 descritas acima é a total compatibilidade com o microprocessador original da Intel, ou seja, as instruções são implementadas de forma a serem executadas no mesmo número de

ciclos do microprocessador original e qualquer código que execute em um processador original pode ser executado na descrição VHDL disponível. A única exceção está em [FRA2000], cuja descrição é compatível em termos de software, mas não em tempo de execução de instruções. Isto se deve à necessidade de se utilizar uma versão otimizada do microprocessador no sistema alvo daquele trabalho.

A versão utilizada conta com as seguintes características:

- Núcleo VHDL [FRA2000] em FPGA;
- Compatível em software mas não em tempo de execução;
- 8 estados; modelo RISC => 1 ciclo de máquina/instrução;
- RAM 256 x 8; ROM 256 x 8; Microcódigo 256 x 20;
- 4 portas de entrada;
- 3 portas de saída.

Uma vez que o tipo de teste implementado em cada núcleo influencia de maneira decisiva o teste do sistema completo, várias técnicas precisam ser tentadas. Porém, é comum ter-se o teste de microprocessadores baseado em cadeias de varredura e/ou *boundary scan* (*scan path* de periferia). Daí a importância de se ter uma versão do 8051 com esta técnica de teste. Com esta implementação, pretende-se verificar como será afetado o teste do sistema como um todo. Além disso, o microprocessador pode ser considerado como um sistema integrado em si, pois apresenta blocos distintos como RAM, ULA e FSM, entre outros, cujas estratégias de teste são diversas.

Este trabalho exemplifica portanto a colocação de teste no microcontrolador 8051 fonte compatível. O mesmo será utilizado no núcleo da caneta tradutora. O objetivo é ter-se um exemplo de um sistema completo formando um SOC (System on a chip) com uma infra-estrutura baseada na técnica do *boundary scan* para o teste de placas de circuito impresso, facilitando a depuração e testes em nível de sistema.

3 A norma IEEE std 1149.1

A partir da década de 70, os testes de placas de circuitos eletrônicos eram baseados na técnica de Bed-of-Nails (cama de pregos). Esta técnica consiste em utilizar-se uma estrutura com diversos pinos, as chamadas ponteiras ou simplesmente pregos. Sobre esta estrutura é colocada uma placa a ser testada, existindo na placa predisposição para fazer contato com os pinos. Nos pinos são colocados vários sinais lógicos, de forma a testar os componentes da placa, bem como as interconexões entre as diversas partes.

O teste pode ser realizado em 2 etapas distintas: power-off e power-on. No primeiro caso, a integridade dos contatos entre os pinos dos componentes e partes da placa é testada, sendo identificadas falhas de circuito aberto e curto-circuito, a partir da medição da impedância. No caso do teste de power-on, aplicam-se sinais lógicos sobre um componente, em parte da placa ou na sua totalidade, verificando-se a resposta. Esta técnica depende de um acesso físico às partes sob teste, dificultado pelo fato de, com o passar do tempo, utilizar-se placas com mais de uma face e ainda com trilhas internas, além da constante miniaturização dos componentes.

Na década de 80, um grupo de engenheiros de teste de companhias européias se uniu, formando o JETAG (Joint European Test Action Group), a fim de buscar soluções para o problema da testabilidade de circuitos eletrônicos. Este grupo sugeriu a adoção de uma técnica utilizando um caminho serial de registradores de deslocamento entre os componentes do circuito, recebendo o nome de Boundary-Scan. Tempos mais tarde, o grupo se uniu a outros engenheiros da América do Norte, dando origem a um novo grupo, o JTAG (Joint Test Action Group). E esta idéia acabou sendo padronizada internacionalmente, tornando-se a norma IEEE 1149.1[IEE90].

Algumas vantagens da utilização desta norma podem ser verificadas:

- verificação de falhas estruturais durante a fabricação;
- redução nos pontos de teste na placa;
- laioutes mais simples, com redução do custo do teste;
- redução de tempo de teste;
- interface padrão;
- redução no time-to-market.

A norma IEEE 1149.1 não define detalhes da arquitetura de teste, detendo-se na sua funcionalidade. A infra-estrutura de teste proposta consiste em uma porta de acesso (TAP – *Test Access Port*), uma lógica de controle, que inclui a decodificação de instruções de teste, e um conjunto de registradores, dentre os quais o registrador de

deslocamento de periferia composto por células de memorização e varredura conhecidas como células *boundary scan*. A eficiência de um teste BS depende, evidentemente, da disponibilidade de circuitos integrados que suportem esta tecnologia. Nos últimos anos, muitos têm sido os fabricantes a disponibilizar CIs com esta tecnologia, apesar de ainda raras as placas 100% compatíveis com a norma IEEE1149.1.

3.1 A Arquitetura Boundary-Scan

Na arquitetura Boundary-Scan, a cada pino de entrada e de saída de sinais é acrescido um registrador de 1 bit (flip-flop), conforme mostra a figura 3.1. Os registradores conectados a pinos de entrada de sinal são chamadas de células de entrada. Os registradores conectados a pinos de saída de sinal são chamados de células de saída.

No modo de operação normal, os dados de entrada são passados diretamente para a saída, ou seja, as células não interferem no funcionamento do CI. No modo update, os valores de entrada (pinos de entrada) são colocados na lógica central do CI e o conteúdo das células de saída é passado para os pinos de saída. O modo de captura faz com que os dados de entrada sejam colocados nas células de entrada, sendo os valores capturados no próximo relógio. Simultaneamente, os dados que se encontravam na lógica central são colocados nos registradores de saída.

No modo de deslocamento, o scan-out de um dos flip-flops é passado ao scan-in do próximo, ou seja, através do caminho serial. Nota-se que as operações de captura e deslocamento não interferem com o transcurso dos dados, ou seja, pode-se capturar os dados durante o funcionamento normal do CI.

Os sinais podem ser inseridos através de um pino adicional no circuito integrado, chamado TDI (Test Data In), o início do caminho BS. Ao final do caminho, há também o pino TDO (Test Data Out), por onde os dados são retirados do chip.

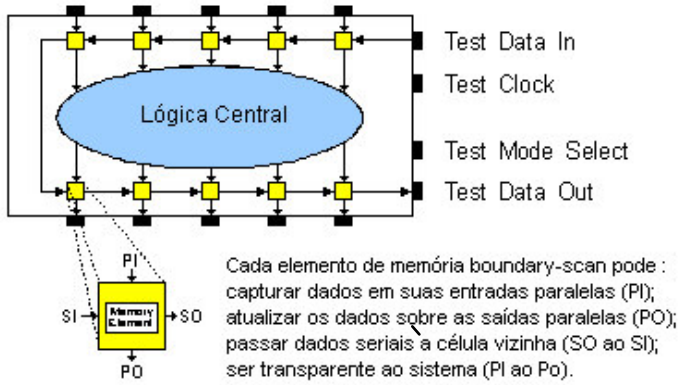

FIGURA 3.1 - Princípio da arquitetura Boundary-Scan

Observando-se o caso de uma placa de circuito impresso, cada pino TDO de um componente é conectado ao pino TDI de seu componente vizinho, formando o caminho serial chamado de chain (corrente) ou canal (figura 3.2).

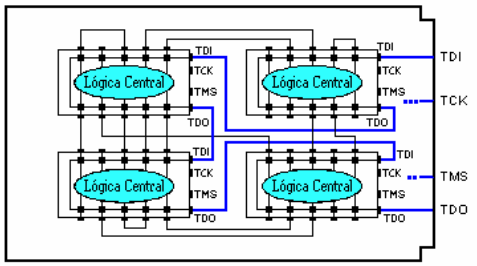

FIGURA 3.2 - O caminho serial Boundary-Scan

Em função do caminho ser serial, é necessário haver um relógio dedicado (test clock) nas células BS, resultando no pino TCK do componente. Podem ser realizados testes em componentes individuais do scan path global, a partir de sinais colocados na entrada da célula BS através do pino TDI.

Muitos tipos de teste podem e devem ser executados sobre uma placa de circuito impresso, antes dela ser aprovada e deixar a fábrica. Para isto, há mais um pino adicional ao circuito integrado: o pino TMS (Test Mode Select). O pino TMS ativa um circuito adicional dedicado ao controle de testes. A norma IEEE 1149.1 permite ainda um pino opcional de TRST (Test Reset).

O uso das células BS não deve interferir no funcionamento da lógica do CI, e o caminho de teste (scan path) é independente da função do componente.

A figura 3.3 mostra um exemplo de uma célula básica de Boundary-Scan. A norma IEEE 1149.1 não define a arquitetura, mas apenas a sua funcionalidade. O circuito é constituído de 2 flip-flops tipo D e 2 multiplexadores (MUX). Há 4 modos de operação da célula BS: normal, update (atualização), capture (captura) e deslocamento (serial shift).

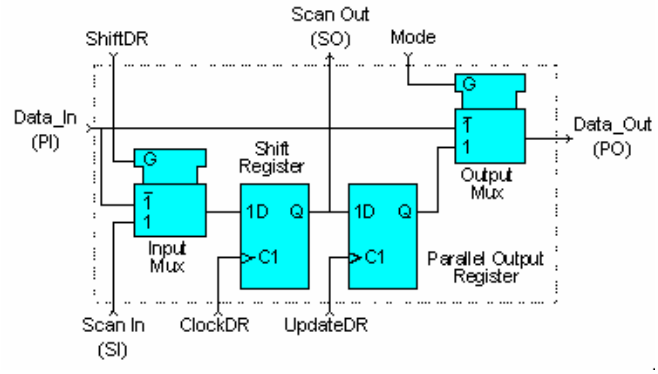

FIGURA 3.3 - Célula básica Boundary-Scan

De fato, o interesse nos testes de placas de circuito impresso não está em testar a funcionalidade dos componentes individualmente, uma vez que já foram feitos durante o processo de fabricação, e sim em proporcionar testes entre as interconexões dos circuitos. As causas de defeitos mais comuns em circuitos são descarga eletrostática, choque mecânico ou choque térmico, afetando ou a periferia do dispositivo ou pontos de solda ou as interconexões entre componentes. Bastante incomum é encontrar problemas na lógica central do componente, sem haver problemas na sua periferia. Justamente entre as células BS fica a lógica do componente.

3.2 Tipos de Testes

Há três tipos de testes possíveis. O teste da lógica central de um componente é chamado de Intest, ou teste interno. Consiste na aplicação de vetores de teste para checar o funcionamento interno do componente. Há também o teste externo, ou Extest, o teste de interconexão dos componentes. Esta é a principal aplicação do BS, buscando falhas de circuitos abertos, curto-circuitos e demais danos na periferia do componente. No terceiro tipo de teste, o sample test, os multiplexadores da figura 3.3 são programados de forma a deixar os sinais passarem diretamente da entrada para a saída da célula BS, ao mesmo tempo em que são capturados nos flip-flops.

3.3 A Arquitetura de um chip IEEE 1149.1

Após 5 anos de discussão, o JTAG finalmente propôs a norma IEEE1149.1 [IEE90], apresentada no circuito da figura 3.4.

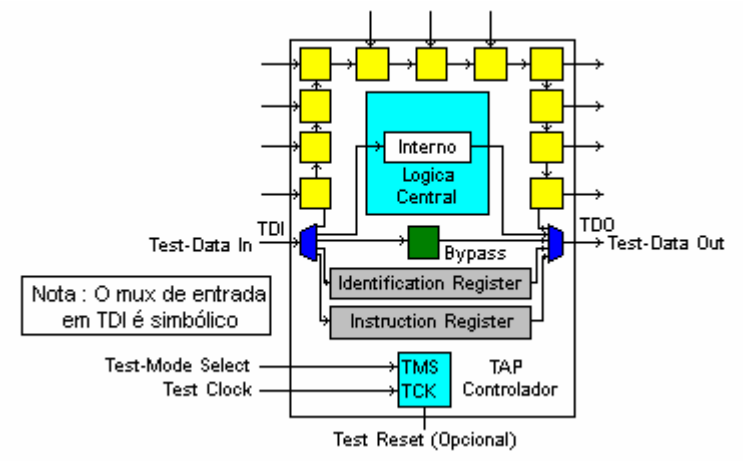

FIGURA 3.4 - Um chip com a arquitetura IEEE 1149.1

Os elementos principais são os cinco pinos de teste, que formam o TAP (*Test Access Port*), dedicados às seguintes funções:

- TDI (Test Data Input): entrada serial para as instruções de teste e dados;
- TDO (Test Data Output): saída serial para os dados e instruções. É um pino tri-state controlado por um sinal de Enable;
- TCK (Test Clock): sinal de relógio, independente do relógio do sistema;
- TMS (Test Mode Select): este sinal é encarregado de controlar os estados do controlador TAP;
- TRST (Test Reset): um pino opcional para reset assíncrono.

Todos os registradores são de deslocamento, cujos conteúdos podem ser modificados através de TDI e TDO e a ativação correta de TMS. Os sinais de TDI e TMS são amostrados na borda de subida do relógio, enquanto que o sinal alcança TDO na borda de descida. Cada um dos sinais TDI, TMS e TRST tem um resistor encarregado de assegurar o sinal lógico "1", mesmo que por algum motivo o pino

esteja desconectado. Isto faz com que o controlador seja conduzido a um estado seguro, quando TMS é mantido em "1" por pelo menos 5 ciclos de relógio.

Os registradores são de dados ou de instruções, de acordo com o seu conteúdo. Os pinos TDI e TDO são usados para buscar instruções ou dados, de acordo com a seleção do sinal TMS. Na placa, o pino TDO é ligado ao pino TDI do próximo circuito, sendo que todos os pinos TCK das células são interconectados, o mesmo ocorrendo com os pinos de TMS. Ou seja, todos os circuitos da placa são controlados paralelamente.

3.3.1 O Registrador de Instruções (IR)

O registrador de instruções é encarregado de selecionar um registrador de dados de teste e, como conseqüência, o tipo de teste. Antes de buscar uma nova instrução de teste, o IR será carregado com o status do teste anterior. Este status será então enviado através de TDO, ao mesmo tempo em que a instrução seguinte é carregada por TDI.

Este registrador tem pelo menos 2 estágios (2 bits), para codificar as 3 instruções obrigatórias: Bypass, Sample/preload e Extest. Cada um destes estágios têm capacidade de deslocamento e carga paralela, além de um latch de saída. No modo de captura (capture), os 2 bits menos significativos devem capturar o valor "01". Esta seqüência permite que algumas falhas no caminho BS possam ser diagnosticadas, pela simples observação de TDO.

3.3.2 O Registrador Bypass

Este registrador possui apenas um estágio (1 bit) e sua função é apenas deslocar os dados através da célula BS, facilitando o particionamento em um teste de interconexão e o diagnóstico (figura 3.4). Uma vez o componente com Bypass ativado, permite-se aos dados passaram diretamente através de si. Este registrador é ativado pela instrução Bypass, e no estado de Capture DR, recebe o valor "0".

3.3.3 O Registrador de Identificação

Este registrador (Ident) é opcional, sendo ativado pela instrução Idcode. É um registrador de 32 bits que realiza operações de captura e deslocamento (figura 3.5). No caso de não estar implementado, é substituído pelo registrador de bypass. Da mesma forma que o registrador de instruções, o seu bit menos significativo tem valor "1". Desta forma, é diferenciado do registrador de bypass, que tem sempre o valor "0" neste bit.

Estrutura de Ident:

Bit 0: sempre "1".

Bits 1 a 11: Identificam o fabricante, de acordo com o código JEDEC.

Bits 12 a 27: Formato livre.

Bits 28 a 31: especificação da versão do dispositivo.

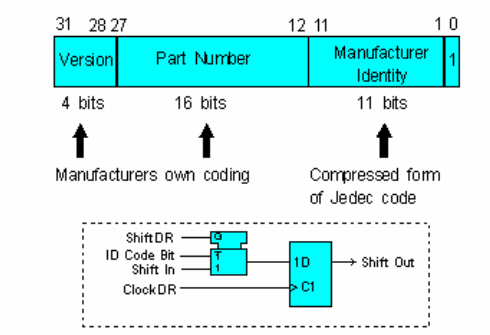

FIGURA 3.5 - Dispositivo de identificação do código da estrutura

Quanto a função deste registrador, considere-se o seguinte exemplo. Um equipamento está com problemas. A suspeita do problema é um defeito de hardware e está em uma placa em particular. Existem muitas variações de placas e o prestador de serviço precisa identificar o tipo de placa e as versões dos componentes. Sabe-se que existem componentes de boundary-scan na placa e a localização dos pinos TDI, TDO, TMS, TCK, Power e Ground. O procedimento seguinte [CAS2000] identifica os componentes de boundary-scan na placa e se eles tem ou não tem registradores de Ident.

1) Alimenta-se a placa e introduz-se a seqüência de valores em TMS para entrar nos modos de operações Select IR_Scan. Através do padrão, a instrução carregada no estado de hold de todo componente boundary-scan deve ser Idcode, se o componente contém um registrador de Ident, ou Bypass se o componente não contém um registrador de Ident. Isto é designado pelo padrão.

2) Capturam-se os valores padrões (Capture_DR) selecionados por Bypass ou pelo registrador de Ident.

3) Deslocam-se os valores capturados (ShiftDR) para a saída TDO primária. Um primeiro “0” identifica um componente sem um registrador de Ident. E um primeiro “1” identifica um componente com um registrador de Ident e neste caso, os próximos 31 bits são de interesse.

Em situações onde não é conhecido quantos componentes da placa são compatíveis com o padrão IEEE std. 1149.1, o processo pode ser terminado inserindo-se uma seqüência ilegal pelo TDI primário e esperando por esta seqüência no TDO primário. Neste caso, a seqüência pode ser sete consecutivos "1s" nos bits 1-7 do campo de identificação do fabricante. O sistema JEDEC de codificação evita esta seqüência. É habitual acrescentar um adicional "0" nesta seqüência, para ter certeza que o TDI primário não está em *stuck-at-1*.

3.3.4 O Registrador Boundary Scan

O registrador Boundary Scan consiste de uma corrente de várias células BS, cada uma conectada a um pino de entrada ou de saída do dispositivo. Como regra, cada célula BS deve permitir teste externo e teste de amostragem (*sample test*). O teste interno é opcional. O registrador boundary-scan é selecionado pelas instruções de Extest, Sample/Preload e Intest. Devem ser colocadas células boundary-scan em todos os pinos de entrada e saída do componente, com a exceção dos pinos de alimentação e terra. É importante salientar que não deve haver nenhum circuito entre o pino e a célula boundary-scan com a exceção de drivers amplificadores ou outras formas de circuito analógico.

Este registrador forma o caminho de varredura da periferia, daí o nome boundary scan. Podem ser realizados três tipos de testes sobre uma placa de circuito impresso [MAU90]: o teste externo, a partir do controle e observação das interconexões de placas e componentes; o teste interno, controlando-se entradas e observando as saídas de sinais; e finalmente o teste de amostragem, que baseia-se na observação das entradas e saídas dos circuitos integrados.

3.3.5 Registradores Definidos pelo Usuário (*user-defined*):

Os registradores definidos pelo usuário são específicos e permitem o acesso a recursos internos em determinados circuitos integrados. Estes recursos podem ser registradores usados para auto-teste ou ainda caminhos internos de teste.

Pensando nestes registradores, a norma IEEE 1149.1 propõe a instrução RUNBIST (Run Built In Self Test). Da mesma forma que na instrução Intest, o CI deve estar isolado do resto da placa para que esta instrução seja executada, evitando possíveis danos.

3.3.6 O Controlador TAP

O controlador TAP é na verdade uma máquina de estados, conforme a figura 3.6.

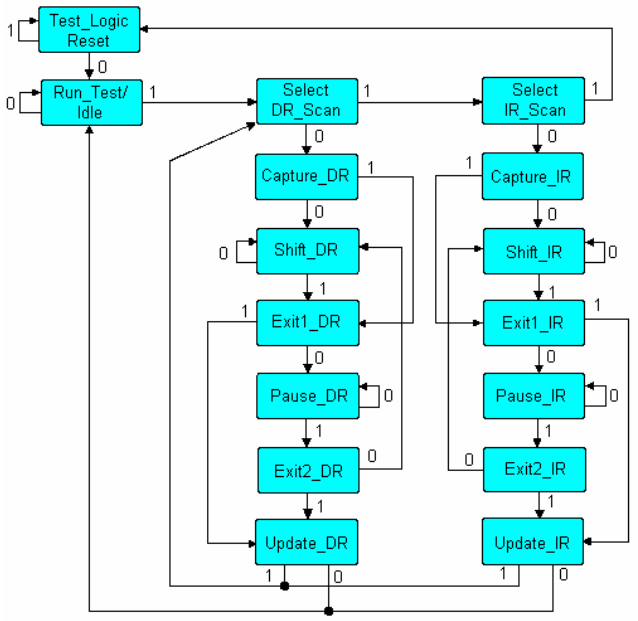

FIGURA 3.6 Diagrama de estados do controlador TAP

Os principais estados são quatro:

Test Logic Reset: Neste estado, o circuito é inicializado e o controlador TAP é programado de forma a não interferir no funcionamento do CI. Não importa em que estado o controlador estiver, em até 5 ciclos de relógio, se TMS for mantido em "1", chegará a este estado. Se algum problema de conexão ocorrer, um resistor se encarrega de manter o sinal em "1". Por definição, o registrador selecionado neste estado é o registrador de identificação, se ele existir.

Run Test/Idle: Este estado tem 2 funcionalidades: mantém a infra-estrutura parada (durante algum intervalo de relógio) ou inicia a execução do teste. É neste estado que os registradores definidos pelo usuário são ativados e utilizados.

Select DR Scan: Aqui seleciona-se qual o tipo de registrador que será usado. Se o valor de TMS for "0", o registrador de teste de dados é selecionado (DR).

Select IR Scan: Neste estado, se TMS tiver valor "0", o registrador de instruções (IR) será colocado em modo de captura. Se TMS for mantido em "1", o controlador retornará ao estado inicial de Test Logic Reset.

Há ainda outros 12 estados, na verdade com 6 funções, multiplicadas por 2 em função dos 2 registradores DR (dados) e IR (instruções). Segue abaixo explicação de cada um dos estados. Deve-se observar que cada um dos estados existe para cada registrador X, representados em 6 para fins de simplificação:

Capture XR: Neste estado, o registrador DR ou IR é carregado com os sinais presentes no circuito.

Shift XR: Aqui ocorre o deslocamento entre TDI e TDO.

Exit1 XR: Neste estado, ocorre a decisão entre continuar o teste ou retornar.

Pause XR: Este estado permite que o teste seja parado por alguns momentos. É interessante quando há necessidade de sincronização com um testador.

Exit2 XR: Outro estado onde há a decisão de continuar ou não uma operação de teste.

Update XR: Neste estado, o registrador X (DR ou IR) é carregado com o conteúdo que foi deslocado para dentro do circuito em Shift XR. Se o registrador é IR, uma nova instrução de teste será ativada, caso contrário, um novo vetor de teste será carregado em DR.

3.4 As Instruções

Há três instruções obrigatórias: Bypass, Extest e Sample/Preload. Além destas, há mais as opcionais Intest, Idcode, Runbist e, a partir de 1993, na norma 1149.1a, as instruções Clamp e Highz.

Bypass: A instrução Bypass deve ser codificada somente com 1s. Uma vez executada, faz com que o registrador de bypass fique no caminho entre TDI e TDO. O caminho selecionado por esta instrução pode ser visto na figura 3.7.

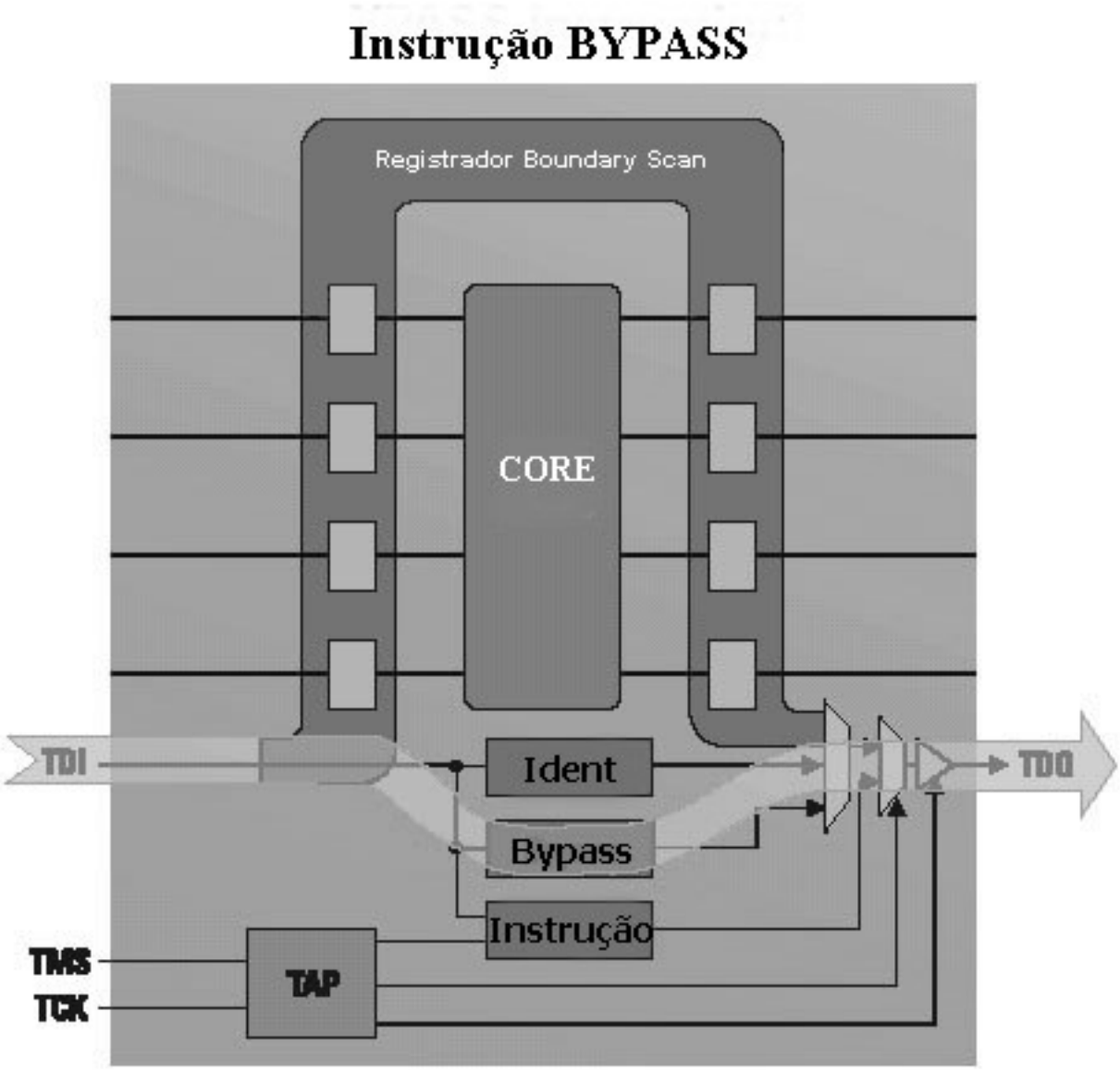

FIGURA 3.7 - Funcionamento da instrução Bypass

Extest: Permite a aplicação, através do registrador BS de saída, de uma seqüência de teste para as interconexões em nível de placa. As respostas são checadas a partir dos registradores BS de entrada do circuito seguinte, conforme pode ser visto na figura 3.8.

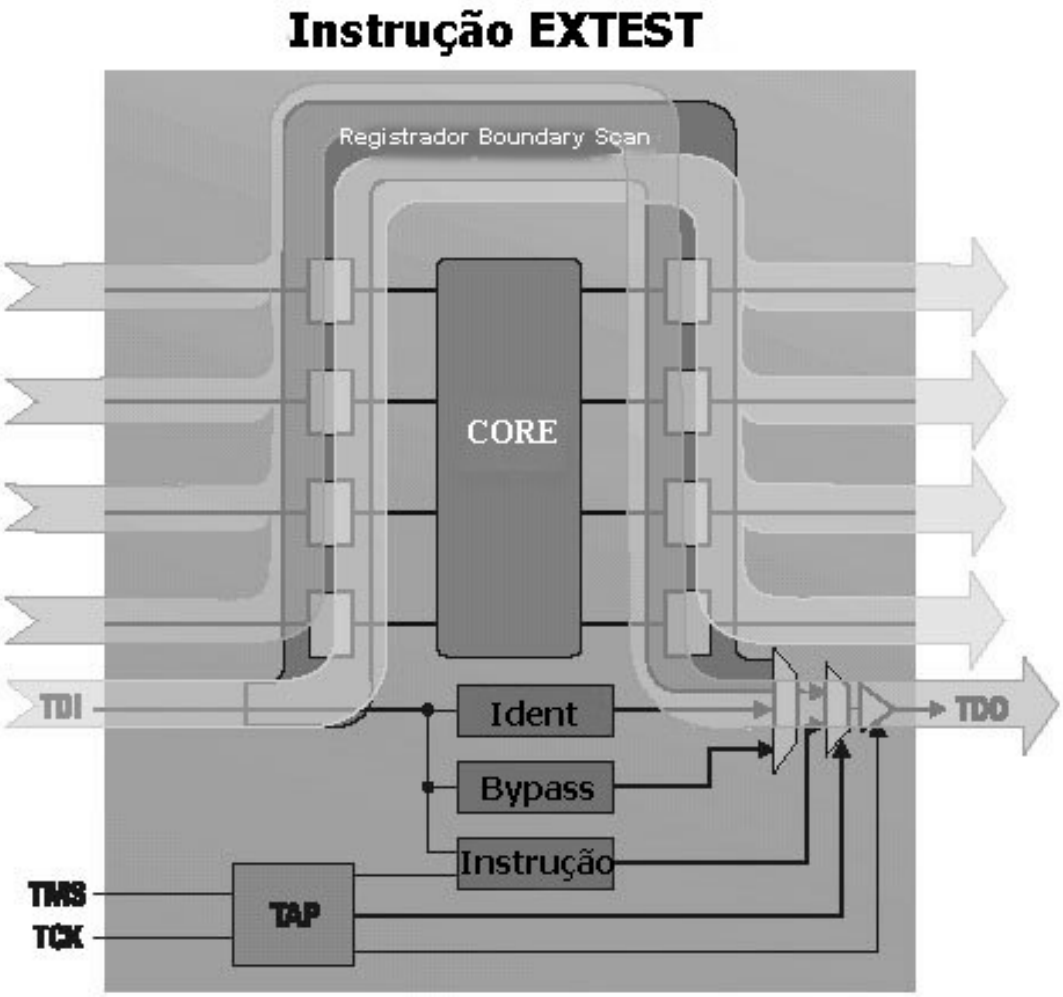

FIGURA 3.8 - Funcionamento da instrução Extest

Sample/Preload: Esta instrução permite a amostragem do estado da placa (Sample) e a pré-carga (preload) do registrador BS com um valor conhecido (figura 3.9). Torna-se interessante para a realização de um teste funcional. Um estado seguro – sinais nos pinos de modo a não danificar o CI em teste durante o funcionamento dos demais circuitos da placa - pode ser colocado no registrador antes de se realizar qualquer tipo de teste.

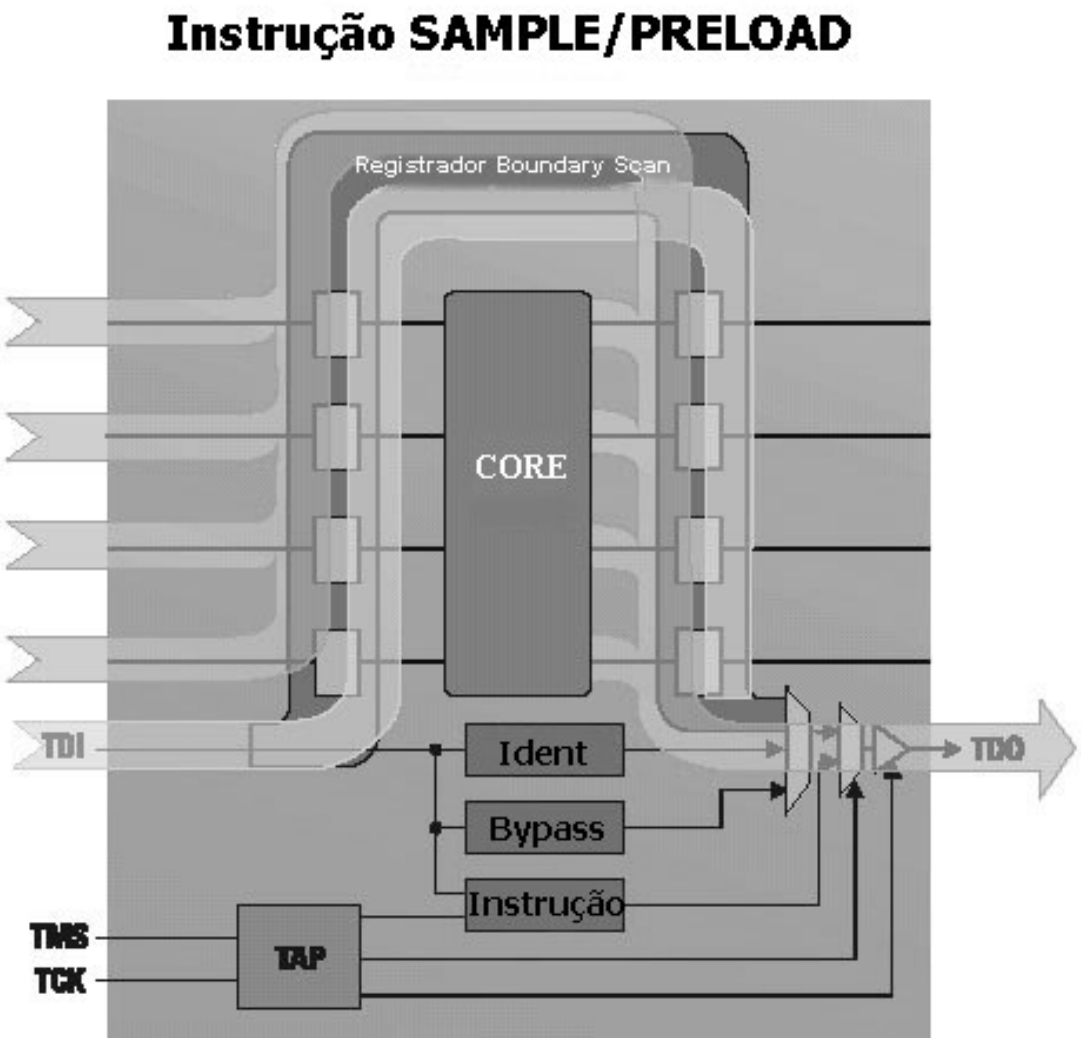

FIGURA 3.9 - Funcionamento da instrução Sample/preload

Intest: Esta instrução permite a aplicação através do registrador BS de entrada de uma seqüência de teste para a checagem do *núcleo* dos circuitos. As respostas saem através do registrador BS de saída e podem então ser analisadas. Para a execução desta instrução, o CI deve estar isolado do resto do circuito, ou ainda, a instrução não pode afetar tensões adjacentes ao CI.

Idcode: Seleciona o registrador de identificação entre TDI e TDO. Se não houver registrador de identificação no chip, esta instrução então é decodificada como se fosse a instrução Bypass.

Runbist: Ao ser executada, pode retornar valores a algum registrador user-defined.

Clamp: Esta instrução leva valores pré-fixados à saída do componente, colocando o registrador bypass entre TDI e TDO.

Highz: Semelhante à instrução Clamp, mas coloca a saída do componente em alta impedância.

Os códigos para todas as instruções não têm valor definido, a exceção da instrução Bypass. Também pode haver mais de um código para a mesma instrução. Códigos não definidos são interpretados como Bypass. Além disso, o projetista pode criar instruções privadas, ou seja, cuja função não está previamente disponível, e que o usuário não pode carregar.

3.5 O Teste BS

A eficiência de um teste BS depende da disponibilidade de circuitos integrados que suportem 100% esta tecnologia. Nos últimos anos, muitos têm sido os fabricantes a disponibilizar CIs com esta tecnologia, apesar de ainda raros os chips 100% compatíveis com a IEEE1149.1.

De acordo com [TUL89], uma seqüência capaz de verificar todas as partes do hardware em uma placa com suporte a BS é composta de inicialização e teste de infra-estrutura, teste de interconexões e teste dos circuitos integrados, detalhados abaixo.

1) Inicialização e teste de infra-estrutura:

1.1) Colocar o controlador TAP no estado de Test Logic Reset (5 ciclos de TCK com TMS em "1");

1.2) Testar o caminho BS através do registrador de instruções (a seqüência esperada é do tipo "X...X01X...X01X...X...X01);

1.3) Carregar os registradores de saída BS com um estado seguro para evitar danos devido a conflitos de barramento (instrução preload);

1.4) Verificar a montagem dos circuitos na placa (instrução Idcode);

1.5) Verificar a operação dos registradores de dados (instrução Bypass e outras).

2) Testes de interconexão:

2.1) Aplicar a todas as interconexões uma seqüência de teste para a detecção de falhas (através da instrução Extest);

2.2) Em caso de detecção de falha, aplicar à conexão suspeita uma seqüência de teste para diagnóstico de falha (instruções Bypass e Extest).

3) Testes de CIs:

3.1) Iniciar o auto-teste nos CIs que têm esta função (instruções Runbist e Bypass);

3.2) Verificar a lógica do *núcleo* dos demais CIs (instruções Intest e Bypass).

[JON91] propõe um procedimento de teste que cobre uma grande gama de problemas, envolvendo a fabricação da placa, registradores, caminhos TDI-TDO e linhas de alimentação. Constitui-se nos seguintes passos:

1) Reiniciar a máquina de estados, usando a técnica de colocar TMS em "1" durante 5 ciclos;

2) Verificar a saída dos registradores de instruções de forma complementar (....11110000....);

3) Verificar a saída do registrador de identificação;

4) Testar a função TRST.

A ordem de execução não pode ser alterada, uma vez que cada etapa seguinte não terá valor no teste se a anterior não tiver sido executada com sucesso.

3.6 Exemplos de testes

A figura 3.10 mostra quatro interconexões entre 2 circuitos. Ambos os componentes seguem a norma IEEE1149.1, sendo que o chip da esquerda envia dados ao da direita. Assumindo que há um curto-circuito entre a conexão 1 e a conexão 2, e a conexão 4 está aberta, como pode-se detectar estas falhas?

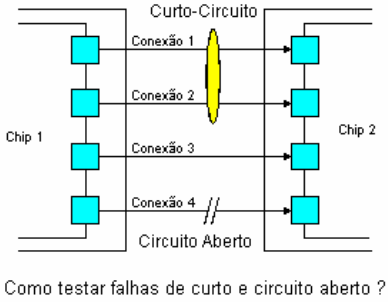

FIGURA 3.10 Exemplo de teste das interconexões

Supõe-se que o curto-circuito assume um comportamento AND. Apenas haverá sinal "1" no chip 2 (saída) quando as 2 entradas (conexões 1 e 2 do chip 1) estiverem com nível lógico "1". Descobre-se este problema a partir da aplicação de

valores inversos do chip 1 para o chip 2. Ou seja, ou aplica-se "0" na conexão 1 e "1" na conexão 2, ou o inverso. Neste caso, o chip 2 receberá sempre o valor "0", permitindo a identificação da falha.

Da mesma forma, o circuito aberto pode ser detectado a partir da falha conhecida como colagem em "zero" (*stuck-at-0*), ou seja, coloca-se o valor "1" no chip 1 na conexão 4 e o valor "0" permanece no chip 2. Similarmente poderia ocorrer que o valor "0" fosse colocado na saída do chip 1 e o valor "1" permanecesse no chip 2, caracterizando a falha do tipo colagem em "um" (*stuck-at-1*); ou ainda um curto do tipo OR. Neste caso, se uma das entradas tiver valor "1", a saída também terá valor "1", da mesma forma que em uma porta lógica deste tipo.

Mas como diagnosticar de forma estruturada falhas do tipo *stuck-at*? Em [Kau74] esta pergunta foi respondida. Considere-se o circuito da figura 3.11.

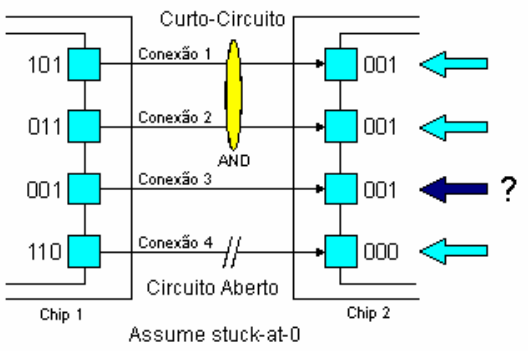

FIGURA 3.11 - Detectando a falha

São mostrados 3 testes consecutivos aplicados às conexões de 1 a 4. O primeiro é o vetor 1110, o segundo 0101 e o terceiro 1001. O valor 101 é aplicado através da conexão 1 - chamado aqui de código horizontal, o valor 011 é passado à conexão 2 e assim por diante. O autor mostra que os códigos horizontais serem únicos para cada uma das conexões é condição suficiente para descobrir um par de conexões em curto-circuito. Sendo assim, o número total de bits em cada código (número de testes) é expressado por $\log_2(N) + 1$, onde N é o número de conexões.

Cada código de estímulo horizontal no circuito anterior é diferente. Observa-se que a resposta nas linhas 1 e 2 estão erradas, por causa do curto-circuito entre as duas conexões. Mas por que utilizar um código de 3 bits, quando apenas 2 cobririam as 4 conexões? Simplesmente porque não estariam cobertas as situações de falhas do tipo *stuck-at*. Os vetores de mesmo valor em todos os bits (000... ou 111...), ou tudo-1 e tudo-0, não podem ser utilizados.

A partir então da comparação dos valores de entrada no chip 1, e de saída, no chip 2, pode-se concluir que:

1) há um curto-circuito entre as conexões 1 e 2;

2) a falha é do tipo AND;

3) a conexão 4 está aberta.

Porém, esta informação pode não estar totalmente correta. A linha de conexão 3 foi testada com o código 001, o mesmo das saídas defeituosas 1 e 2. Ou seja, o curto-circuito poderia envolver as linhas 1, 2 e 3. Este problema é chamado de síndrome de aliasing [JAR89]. A adição de um novo teste pode reduzir a ambigüidade. É o caso da aplicação do código 0011, conforme a figura 3.12. Em suma, o curto-circuito entre as conexões 1 e 2 é separado da conexão 3.

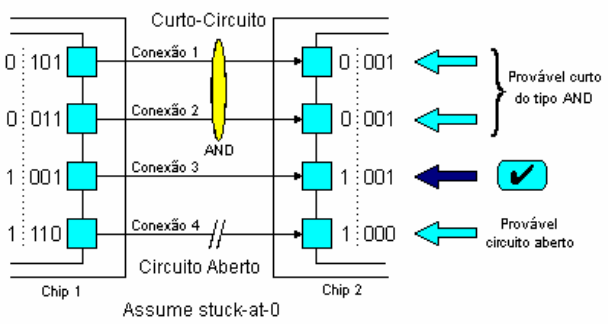

FIGURA 3.12 - A localização da falha

Outro problema relacionado ao diagnóstico de falhas é chamado de síndrome de confounding [JAR89]. Pode ser exemplificada com o mesmo circuito analisado anteriormente, porém com 2 curto-circuitos: um entre as conexões 1 e 2 e outro entre as conexões 3 e 4. Dependendo dos vetores de teste, no caso de falhas do tipo AND, pode-se diagnosticar erroneamente que há um curto-circuito interligando todas as 4 conexões, o que não seria verdadeiro.

3.7 BIST

Funções de BIST (Built In Self Test), ou auto-teste integrado, aprimoram as funções de testabilidade de circuitos dotados de células boundary-scan. Simplificam a detecção de falhas e o diagnóstico, formando na associação BIST-BS uma ótima solução para testabilidade. Como vantagens, pode-se esperar uma diminuição do

tempo necessário ao teste, além da simplificação dos elementos testadores. Também observa-se que os custos advindos do aumento da superfície de silício necessária à sua implementação podem ser minimizados, na medida em que a lógica do componente é aproveitada no teste. O uso de BIST na produção de circuitos integrados em larga escala é muito interessante, pois tem-se uma significativa redução no custo de produção associado ao teste, como por exemplo a redução do tempo de aplicação do teste e, portanto, do tempo de utilização de um testador externo, e ainda a simplificação do hardware e do software do testador, reduzindo, assim, seu custo total [COT99b].

4 Incluindo Testabilidade no Microcontrolador 8051

4.1 Especificações e desenvolvimento do teste

A implementação de testabilidade no 8051, como em qualquer outro CI, não é tarefa fácil. Primeiro porque tem de partir-se de uma estrutura já pronta, no caso o 8051 de autoria de Denis Franco [FRA2000]. Isto é, entender como funciona, tudo o que ocorre dentro do circuito, a tecnologia utilizada, forma de acesso à memória e demais aspectos físicos e lógicos da implementação. Em segundo lugar, porque é necessário entender-se profundamente o funcionamento do teste: o que é, a que se propõe, qual a melhor estratégia e como implementar. Logo a seguir, a dificuldade é entender e, porque não, interpretar a norma IEEE1149.1, englobando as funções obrigatórias, deixando a cargo do projetista a forma de elaboração do circuito e, por isso mesmo, dando maior flexibilidade durante o projeto.

O 8051 utilizado é um tanto pequeno comparado a outros processadores comerciais de funções semelhantes, porém mais complexas, com pipelines e outros circuitos que aceleram o processamento das instruções. No entanto, este circuito, sendo mais simples, facilita a compreensão da elaboração do teste e, assim, contribui para o entendimento deste e de suas implicações.

O circuito soc_8051 tem as seguintes características:

- Núcleo VHDL [FRA2000] em FPGA
- Compatível em software mas não em tempo de execução
- Máquina com 8 estados seguindo um modelo RISC de 1 ciclo de máquina/instrução
- RAM 256 x 8 bits; ROM 256 x 8 bits; memória para microcódigo de 256 x 20 bits
- 4 portas de entrada
- 3 portas de saída

Conforme já mencionado, a idéia principal consiste em prover acessos eletrônicos ao 8051 que permitam testar o interior do núcleo e as suas interconexões, quando este estiver integrado em um sistema maior. A infra-estrutura boundary scan provê acesso à interface do núcleo e pode ser utilizada para o teste de interconexões entre núcleos. O interior do núcleo pode se tornar acessível a partir do boundary scan,

se uma cadeia de varredura interna (*scan path*) for implementada. Esta cadeia pode vir a tirar proveito dos registradores já existentes e pode necessitar que novos registradores sejam incluídos no núcleo. Esta cadeia interna deve servir ao teste dos diversos blocos internos ao microcontrolador como seus multiplexadores, operadores lógico-aritméticos, e os próprios registradores, dentre outros.

Partindo-se então do código VHDL original sem teste, optou-se por incluir descrições do TAP (*Test Access Port*) e da máquina de controle deste, com seus 16 estados, além do registrador de Instrução, registrador Bypass e sinais de controle para os demais registradores que se fizessem necessários ao circuito.

Na seqüência, identificou-se registradores internos que poderiam ser transformados em registradores de varredura de forma a se realizar um teste serial dos blocos internos. Constatou-se que 8 registradores extra seriam necessários para se obter a acessibilidade adequada para a aplicação de testes que pudessem conduzir a uma boa cobertura de falhas internas. Estes registradores extras devem ser transparentes durante o funcionamento normal do circuito, ou seja, devem funcionar sem interferir nos sinais originais. Desta forma, não provocando atrasos. De acordo com os sinais de controle que vêm do TAP, estes registradores podem funcionar tanto como registradores normais (atraso de 1 pulso de relógio para atualização do valor), como sinais combinacionais ou fios, sem nenhum atraso para atualização de valor. Assim, dois modelos de células foram utilizadas para todos os registradores de varredura necessários.

O 8051 original foi então alterado de forma a integrar as características necessárias à norma IEEE1149.1, quais sejam:

1) A inserção de um controlador e a porta TAP, uma interface padrão com pinos que permitem a troca de informações com o ambiente externo ao núcleo (*cor*e) do chip, composta por cinco pinos (figura 4.1):

- TDI (*Test Data Input*): entrada serial para instruções e dados de teste;
- TDO (*Test Data Output*): saída serial para dados e instruções. É um pino de alta impedância controlado por um sinal de *enable*;
- TCK (*Test Clock*): sinal de relógio, independente do relógio do sistema;
- TRST (*Test Reset*): sinal de reset assíncrono;
- TMS (*Test Mode Select*): este sinal é encarregado de controlar os estados do controlador TAP.

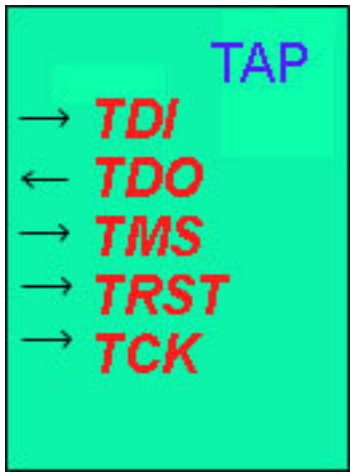

FIGURA 4.1 - Pinos do TAP

2) Um registrador de instrução que suporta quatro tipos de teste:

- *o teste da lógica central do componente*, ou teste interno, que consistirá na aplicação de vetores de teste, através de cadeias internas de varredura, para verificar o funcionamento dos vários módulos internos ao processador, tais como ULA, RAM, ROM e parte de controle (instrução ScanInt);
- *o teste externo*, que consiste na verificação das interconexões dos componentes. Esta é a principal aplicação do BS, buscando falhas de circuitos abertos, curto-circuitos e demais danos na periferia do componente (instrução Extest);
- *teste de amostragem*, no qual as células BS são programadas de forma a deixar os sinais passarem diretamente do pino ao núcleo do circuito e vice-versa, ao mesmo tempo em que são capturados na própria célula (instrução Sample/Preload);
- *teste da memória RAM*, através do algoritmo March [LAL97], com ativação pela instrução RunBIST. As demais memórias, de microinstrução e de programa, têm um bit de paridade acrescido à sua área de dados.

3) Um conjunto de registradores de dados de teste que inclui:

- os registradores tidos como obrigatórios na norma IEEE 1149.1, dentre eles o registrador *boundary scan*;

- uma cadeia de varredura interna ao processador, que implementa a técnica do *scan path*, e que é composta por registradores internos modificados e registradores acrescidos com o propósito único do teste.

Todos os registradores são de deslocamento, cujos conteúdos podem ser modificados através de TDI e TDO conforme ativação do TMS, todos sincronizados por TCK.

Uma vez definido o circuito-alvo desta implementação e a utilização do padrão IEEE1149.1, a primeira etapa era definir o que exatamente seria realizado dentro do circuito e de que forma, quais as estruturas utilizadas e o que efetivamente seria implementado. O primeiro aspecto importante é que se teria uma máquina de estados controlando o funcionamento do circuito responsável pelo teste, os registradores e sinalizações: o controlador TAP. Como foi visto no capítulo anterior, o TAP tem 16 estados e, de acordo com o estado, o circuito como um todo estará em modo de teste ou funcionando normalmente.

A norma define que há três instruções obrigatórias. São elas: bypass, sample/preload e extest. Além disso, há a obrigatoriedade de um registrador de instruções, responsável por armazenar a instrução que está sendo executada no momento pelo controlador TAP; do registrador bypass, registrador de um bit responsável por encurtar caminho entre um chip que esteja dentro de uma cadeia boundary-scan maior em uma placa de circuito impresso; e do registrador boundary-scan, responsável por interconectar o núcleo do chip, com toda a sua funcionalidade, com o meio externo.

Além das instruções obrigatórias, mais duas instruções foram agregadas. São elas: scanint e runbist, criadas para ativar a cadeia de varredura interna e uma cadeia de controle para a função BIST interna ao chip.

Em resumo, a implementação das seguintes instruções faz parte da especificação do teste:

- Teste da lógica interna do componente: *ScanInt*
- Teste externo *(Boundary Scan): Extest*
- Teste de amostragem: *Sample/Preload*
- Caminho para o próximo chip: *Bypass*
- Teste da memória RAM (BIST): *RunBIST*

A figura 4.2 mostra o núcleo do microcontrolador 8051 fonte compatível com exemplos de células boundary-scan ligando o núcleo aos pinos de entrada ou saída. Também pode-se observar o bloco controlador TAP e o registrador de instruções.

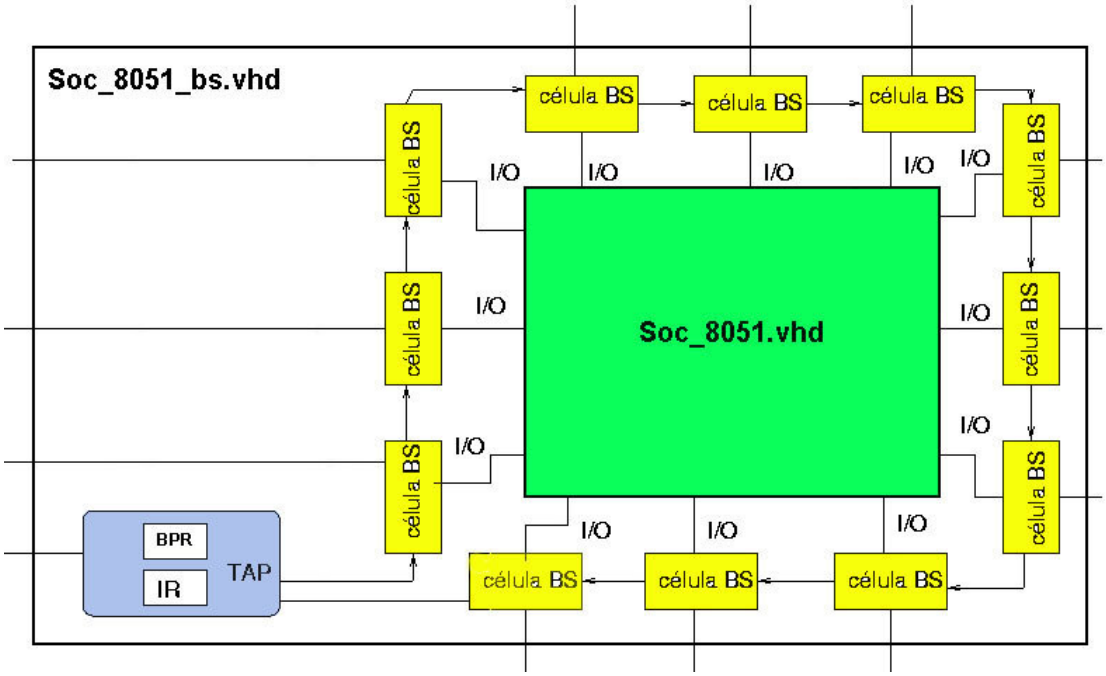

FIGURA 4.2 - Diagrama-exemplo de blocos do 8051_bs

As células que compõem a cadeia boundary-scan são de fato registradores que podem em um determinado instante armazenar um bit a partir de uma entrada (Data_In) e, em outro instante, e de acordo com sinais de controle aplicados sobre pontos da célula, levar este bit armazenado até a saída do registrador boundary-scan (Data_Out). De acordo com o pino onde a célula estiver conectada, se entrada ou saída, ela irá fornecer este dado ao meio externo ou ao núcleo do chip. O diagrama básico da célula boundary-scan pode ser observado na figura 4.3.

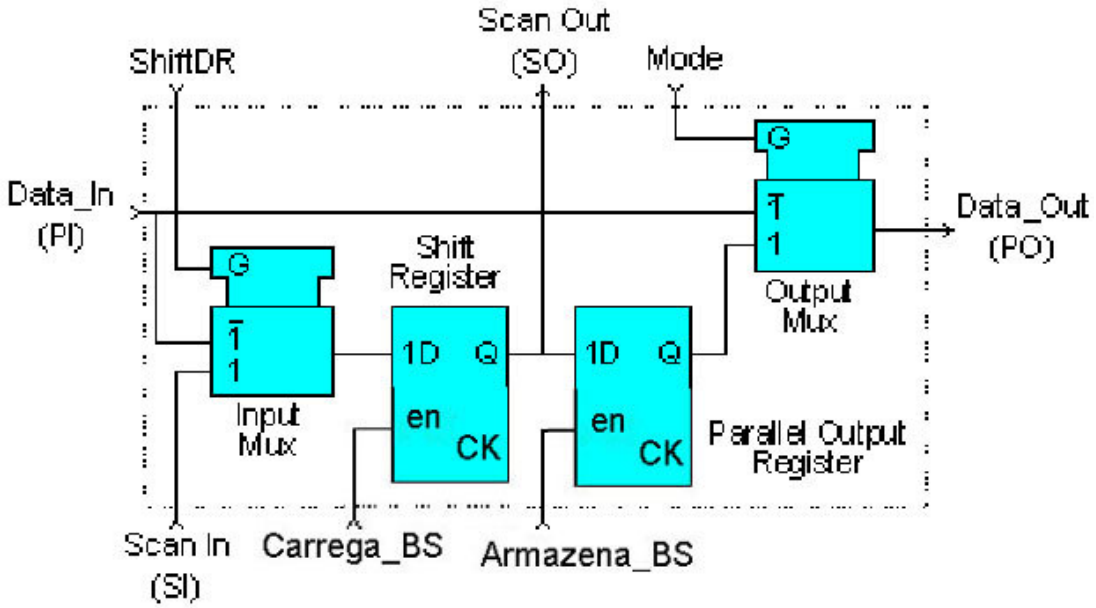

FIGURA 4.3 - A célula básica bs_cell

Sendo assim, da união de várias células BS tem-se um registrador de deslocamento, formando a cadeia boundary-scan. Neste caso, como o microcontrolador 8051 tem 48 pinos, alguns de entrada, outros de saída de sinal, tem-se 48 células básicas boundary-scan. Somam-se a estes pinos os pinos do TAP: TDI, TDO, TRST, TMS e TCK, além de pinos de alimentação, clock (relógio) e reset, os quais não participam com informações para o teste.

Até este ponto tem-se o TAP, com o registrador de instruções e o registrador bypass, e a cadeia boundary-scan visando a inserção de testabilidade. Porém, foi definido que internamente ao núcleo do circuito haveria uma cadeia de teste, a chamada cadeia interna (*scan path*).

O microcontrolador possui em sua arquitetura interna alguns registradores necessários ao seu funcionamento, como por exemplo o registrador PC (Program Counter), registrador de instruções do 8051 (IR), o registrador de estados, entre outros, conforme pode-se observar na figura 4.4.

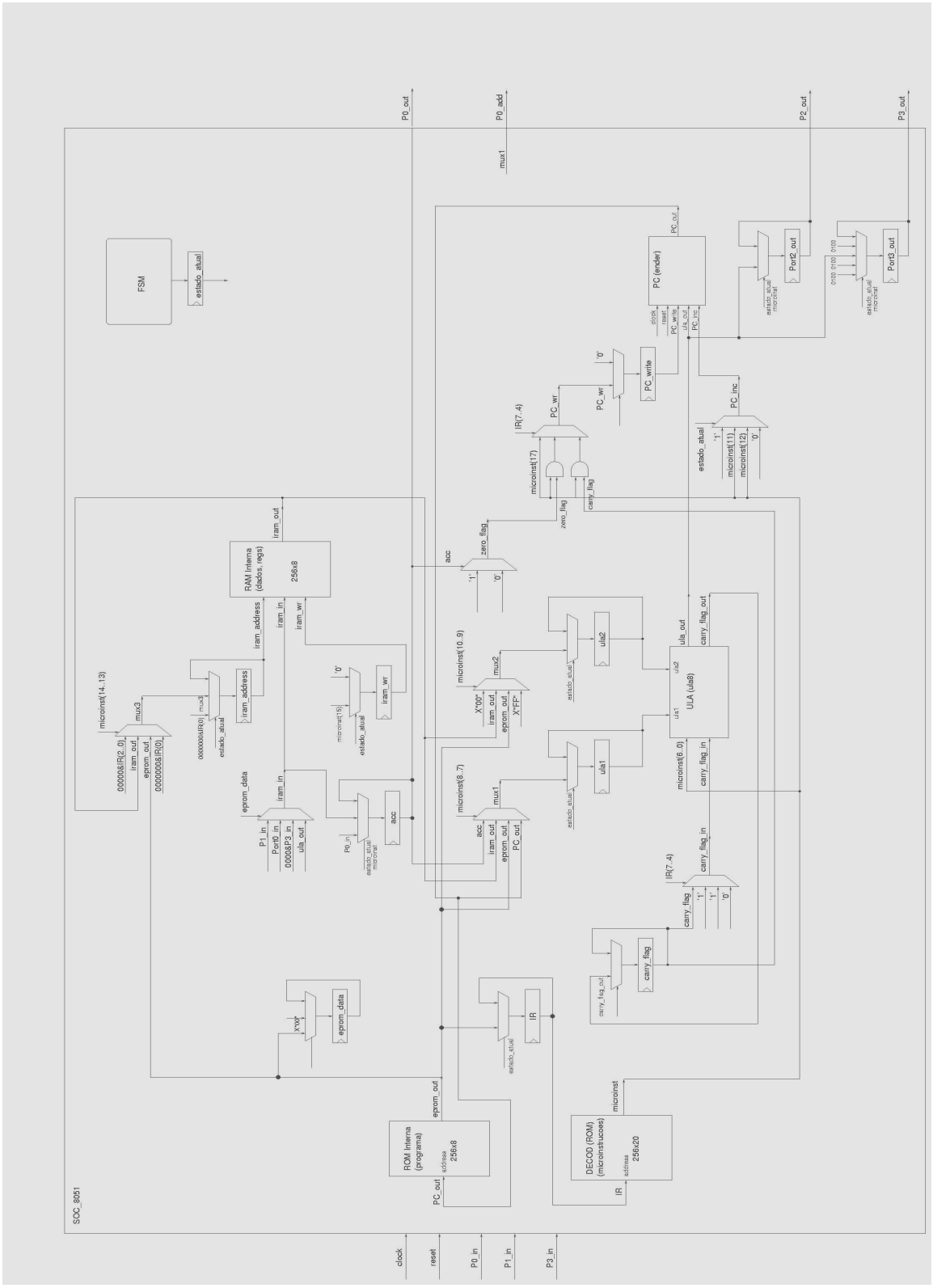

FIGURA 4.4 - Diagrama do 8051 original

Pois serão estes registradores – PC, IR, registrador de estados - alguns dos quais formarão a cadeia interna de varredura, necessária para que se possa testar o funcionamento interno do núcleo do circuito e dos próprios registradores. A escolha de que registradores e que ordem irão assumir na cadeia de varredura interna influencia a eficiência do teste. Os registradores devem ser escolhidos devido à sua importância ao chip e levando-se em conta que mais módulos do microcontrolador são acessados por estes mesmos registradores.

Os registradores mais usados são colocados no início da cadeia, levando a uma diminuição no tempo de teste (menos deslocamentos no total). Os registradores mais usados para receber as respostas de teste são colocados ao final da cadeia. Este ponto é definido de acordo com o circuito. Por exemplo, ligados à ULA estão ULA1 e ULA2. Então o mais coerente é carregar sinais antes nestes registradores, para depois buscá-los no registrador subseqüente à ULA.

Colocar os registradores mais usados mais próximos do início da cadeia de varredura faz com que se tenha testes com um menor número de ciclos de relógio. Isto ocorre porque, como pode ser visto a seguir, alguns registradores dependem de outros para o seu funcionamento. É o caso do registrador de estados. Dependendo do valor inserido neste, vários sinais serão disparados, definindo o comportamento de diversas outras partes do circuito naquele momento. Portanto, é importante organizar-se o teste de forma a colocar todos os dados nos registradores a serem testados e nos demais que realizam funções que influenciam outros registradores, de maneira a realizar-se o trabalho em paralelo, maximizando o desempenho pela redução do tempo necessário ao teste.

Sendo assim, a instrução ScanInt, ativando a cadeia única ligando TDI, registradores internos e TDO, ficou definida com a seguinte ordem:

Caminho: TDI => estado_atual(3 bits) => PC_out(8) => IR(8) => acc (8) => aux_mux2 (8) => aux2_mux2 (8) => ula1 (8) => ula2 (8) => aux_eprom_data (8) => eprom_data (8) => aux_iram_wr(1) => iram_wr(1) => carry_flag(1) => pc_write(1) => aux_iram_address (8) => aux1_mux3 (8) => aux2_mux3 (8) => iram_out (8) => iram_address (8) => port2_out (8) => TDO, totalizando 127 bits.

Desta forma, são necessários 128 ciclos de relógio para que um dado na entrada da cadeia fique disponível na saída desta, lembrando que a cadeia forma um grande registrador de deslocamento que permite carga paralela dos dados de e para o restante do *núcleo*, controlada a partir de estados do controlador TAP.

Escolheu-se uma única cadeia, pois rotacionar 127 bits em um registrador único é rápido e não implica em muito trabalho computacional, comparando-se com o tempo de execução do BIST da memória que está em torno de 3ms, como será visto na seqüência. Uma idéia é dar-se o início no teste da RAM e, durante este tempo, proceder à execução de outros testes.

Mas algumas questões podem ser levantadas. Por exemplo, qual o impacto no tempo e na qualidade do teste ao mudar-se o número de registradores *scan* internos, a seqüência dos registradores na cadeia e o número de cadeias?

O tempo de teste pode ser reduzido, pois com mais cadeias ter-se-á menos bits para rotacionar na cadeia. Dependendo do teste, menos tempo poderá ser necessário para cada teste. Quanto à qualidade do teste, pode-se incluir alguns sinais em determinados pontos de uma cadeia interna e outros sinais em outros, ocasionando maior controle e outras possibilidades de teste, mas não testes simultâneos, pois apenas uma instrução estará ativa no tap em cada momento. O que pode ser feito é implementar uma instrução que, por exemplo, acesse os sinais de duas partes do circuito em paralelo. Há que se verificar como trabalhar estes sinais, talvez interligando duas cadeias através de TDI e TDO.

E qual o impacto das mesmas mudanças no TAP e na cadeia boundary-scan?

Uma maior complexidade do TAP, devido à inserção de mais estados. Isto porque haverá mais sinais saindo do TAP, já que cada cadeia tem sinais de acionamento próprios. Quanto ao BS, não há implicações. A cadeia BS definida realiza os testes necessários e não há nenhuma necessidade de alterações neste ponto.

A figura 4.5 apresenta detalhes importantes a respeito do circuito e como ficaram os registradores e interconexões destes a partir da inserção da cadeia de varredura interna.

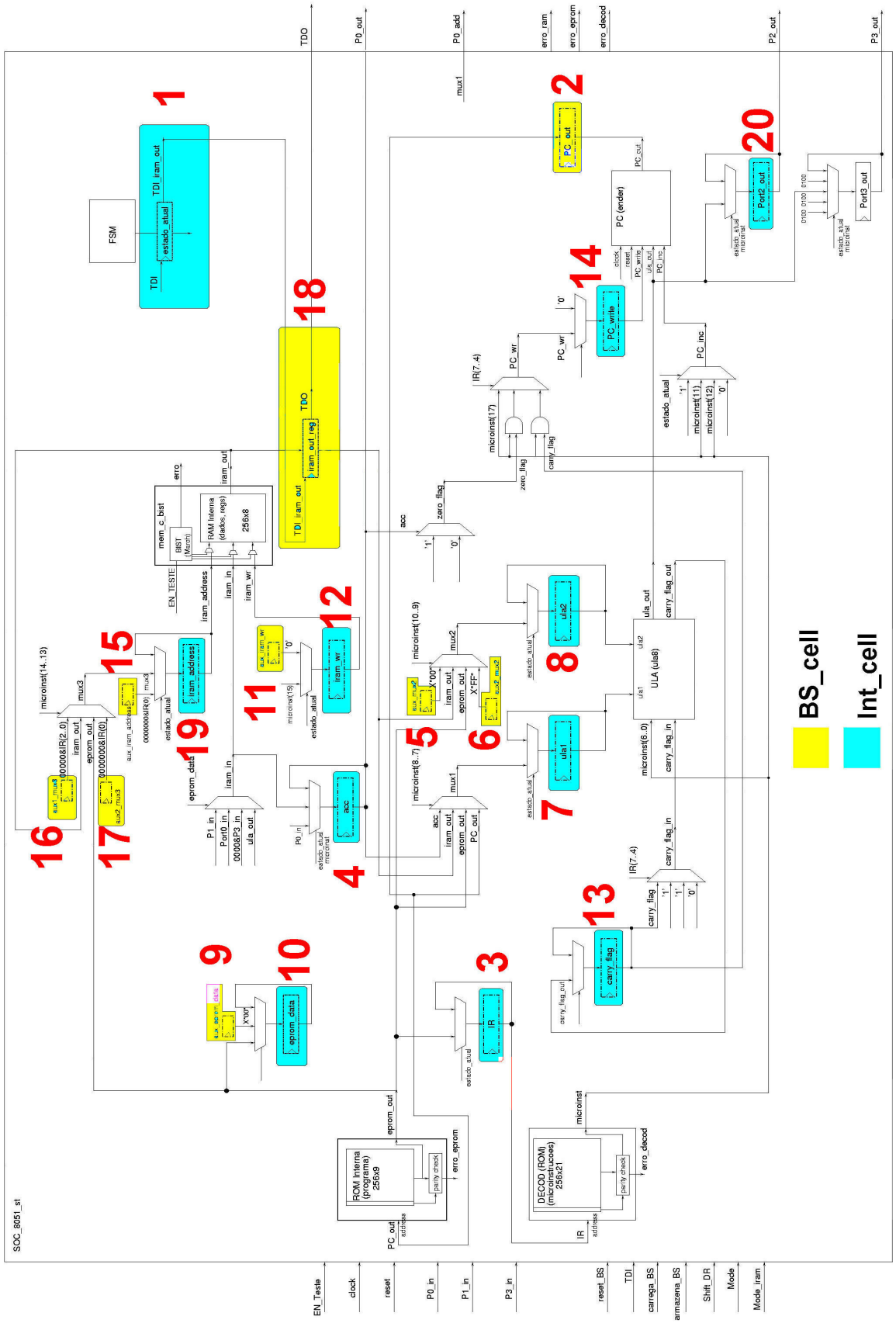

FIGURA 4.5 - Diagrama do 8051 completo, com teste

A numeração corresponde à ordem dos registradores na cadeia de varredura interna. Os registradores hachurados em tom mais claro correspondem a registradores que não existiam e foram ali inseridos de forma a facilitar e manter/receber sinais no seu caminho, em virtude do teste. Os registradores hachurados mais escuros são formados por células do tipo int_cell, mostrada em maior detalhe na figura 4.6.

Primeiramente fez-se necessário alterar os registradores pré-existentes no núcleo do microcontrolador, de forma que não fossem inseridos outros ciclos de relógio no sistema. Usando-se o mesmo tipo de registradores da cadeia boundary-scan, haveria o problema de inserção de ciclos, gerando atrasos.

A partir de então alterou-se estes registradores para que eles reagissem conforme os sinais do TAP e continuassem a realizar as suas tarefas originais normalmente, sem mudanças na temporização em relação ao circuito original. Esta célula, que agrupada com outras de forma a ter a quantidade de bits do registrador original, substitui o registrador original da descrição e pode ser vista na figura 4.6.

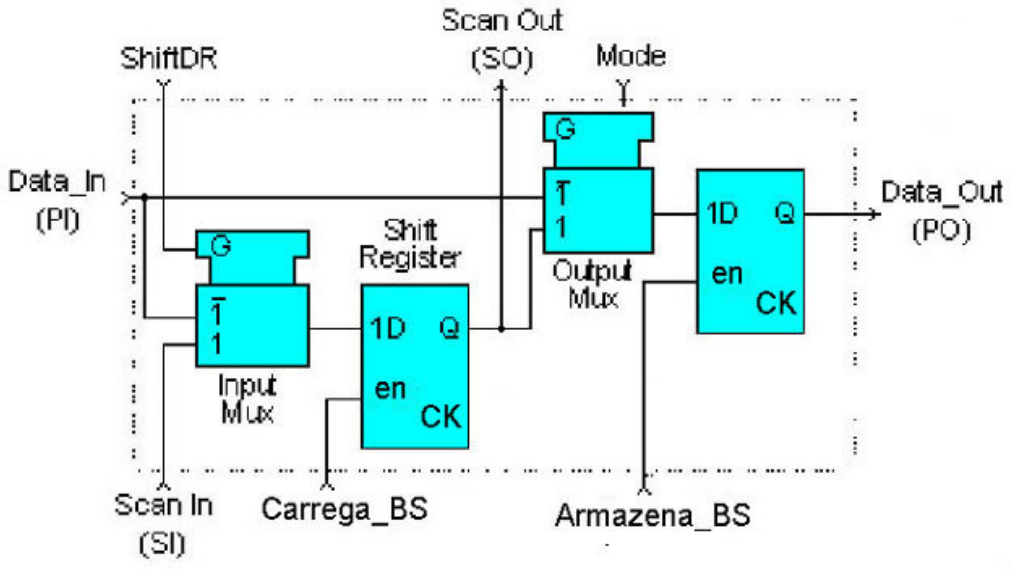

FIGURA 4.6 - A célula básica int_cell

Como se achou necessário a colocação de células de teste em alguns pontos onde ainda não havia registradores, colocou-se células BS convencionais, mantendo-se o funcionamento normal do circuito e, no caso de uma situação de teste, aí sim o comportamento sendo alterado e respondendo à TAP.

A diferença entre as duas células básicas consiste então em:

- Célula BS:
  - Não necessariamente está interferindo no funcionamento do circuito, deixando os sinais passarem através dela por um caminho que não seja através do segundo flip-flop.
- Célula interna:
  - Substitui um registrador e um dos flip-flops está sempre interferindo no funcionamento do circuito, ou seja, armazenando um bit de acordo com o relógio.

Uma vez TAP implementado e testado, com as instruções definidas e com a cadeia interna e registradores funcionando de acordo com os sinais do TAP, pode-se unir o circuito ao 8051, formando o 8051 com teste.

A figura 4.7 mostra o microcontrolador 8051 com teste em blocos.

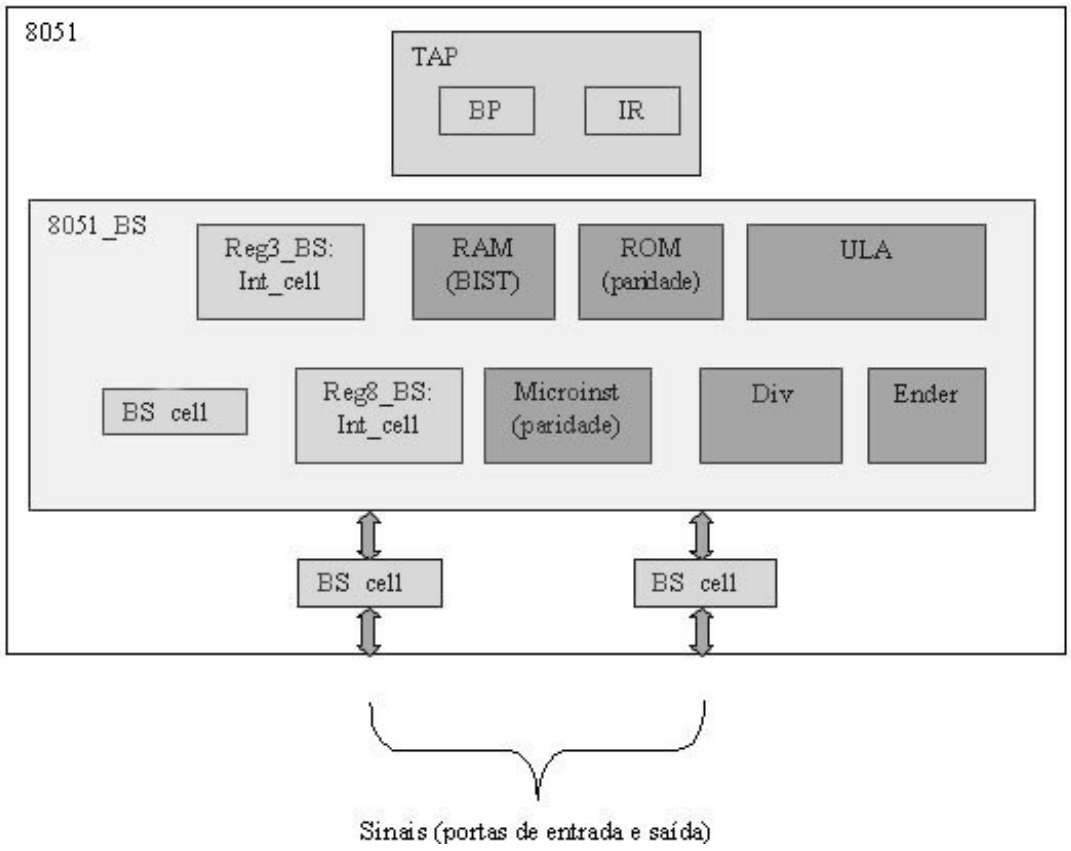

FIGURA 4.7 - Diagrama de blocos do microcontrolador 8051 com teste

Tem-se o TAP, com os registradores Bypass (BP) e de Instrução (IR), as células BS_cell, formando a cadeia boundary-scan, e as células do tipo int_cell, agrupadas de forma a substituir os registradores originais sem teste. A memória RAM utiliza *BIST*, as memórias de microinstrução e ROM têm paridade e tem-se ainda a

ULA, Div, para dividir o sinal de relógio em cinco vezes, e ainda o módulo Ender, responsável pela geração de endereços.

Células int_cell são agregadas de acordo com o número de bits do registrador original, tipicamente registradores de um bit, três bits ou de 8 bits, tais como carry_flag, estado_atual e registrador de instruções do 8051, respectivamente.

Além das cadeias interna e boundary-scan previamente mencionadas, há ainda uma terceira cadeia de varredura, ativada pela instrução RunBIST. Esta instrução ativa uma cadeia com dois registradores, e é responsável por levar sinais de ativação de teste à memória RAM, ativando o BIST, para que em um momento seguinte recuperem-se os dados resultantes do teste. Este sinal irá indicar se o teste resultou em sucesso ou falha desta memória RAM.

Na verdade, esta cadeia foi elaborada para fins didáticos, uma vez que seria muito simples ter-se apenas um pino de entrada ativando o teste e um pino de saída responsável por indicar sucesso ou erro no teste da memória. Mas claro que o 8051 deveria permanecer em um modo de espera, em um looping, a fim de não gravar nem ler nada da memória para não influenciar e não ser influenciado pelo teste em seu funcionamento.

O registrador estado_atual da máquina de estados do 8051 foi transformado em um registrador scan de 3 bits feito com o agrupamento de 3 células do tipo BS_cell. Para isto, modificou-se a descrição da máquina de estados, pois necessita-se ter controle sobre a codificação dos estados. Este registrador deve ser carregado com Data_in com 1 pulso de relógio, quando em modo normal. Em modo teste (de acordo com as variáveis de controle dos flip-flops e mux da bs_cell) ele funciona da forma que se definiu.

A seguir, inseriu-se um registrador Scan de 8 bits na saída da RAM. Este registrador não existia anteriormente, ou seja, o dado da RAM estava disponível tão logo o endereço estivesse estabilizado. A inclusão de um registrador real neste ponto implicaria a mudança da máquina de estados e da temporização das instruções. Então, quando o processador funciona normalmente, este registrador deve ser transparente (Mode = 0), ou seja, deve deixar passar Data_in sem atrasos.

Para a memória RAM, o algoritmo March utilizado na versão auto-testável do 8051 original [COT99a] foi reutilizado sem modificações. Este algoritmo funciona da seguinte forma:

- escreve-se 0 em todos os bits da memória;
- percorre-se o primeiro bit de cada palavra, verifica-se se é 0 e escreve-se 1;
- percorre-se o segundo bit de cada palavra e faz-se o mesmo, verificando-se se é 0 e escrevendo-se 1;
- ao chegar-se ao último bit de cada palavra, faz-se o processo inverso, verificando-se se cada bit é 1 e após escrevendo-se 0.

Em caso de algum erro na leitura esperada de cada bit, o processo é parado e a flag (sinal) de erro é ativada. Para este caso específico, são necessários 3,008337ms, ou seja, 29.208 ciclos de relógio (considerando-se um relógio de 103ns), para realizar o BIST da RAM.

Como o relógio ficou em 103ns, 127 x 103ns é o tempo para inserir um vetor de teste na cadeia, no pior caso. Este também é o tempo necessário para ler um resultado de teste no fim da cadeia. Significa que o tempo de teste total seria: 2*no.vetores*(127 x 103). Para um número reduzido de vetores, o tempo ficou menor que o tempo necessário para o teste da RAM, levando a imaginar em se disparar o BIST e neste tempo se utilizar da cadeia interna para outros testes, desde que não relacionados à RAM.

Ao mesmo tempo, tem-se pinos indicando erro de paridade saindo do chip. Estes sinais são ativados pelas memórias EPROM – de programa - e Decod – de microprograma - e não foram colocados na cadeia. Visto que a sinalização de um erro de paridade é instantânea, não poderia ser feita uma leitura apenas em um determinado momento onde o chip estivesse em modo teste e com a instrução e cadeia responsáveis ativados. Neste caso foi decidido desta forma, mas há circuitos de memorização que permitem que se faça a leitura da paridade ao final do teste, como pode ser visto com maior detalhamento em [LUB92].

A cadeia *boundary scan* pode ser utilizada para um teste de interconexão, conforme pôde ser visto na seção 3.6. Neste caso, colocam-se vetores de teste nos pinos de entrada e/ou saída de forma a alimentar ou retirar sinais de outros componentes adjacentes, supondo que o microntrolador faça parte de uma placa maior com mais componentes testáveis ou não. Se forem testáveis, tanto melhor, pois ter-se-á uma melhor flexibilidade e possibilidade de testes.

A versão completa em VHDL do 8051 com teste, incluindo o BIST, a cadeia de varredura interna, a cadeia boundary-scan e a paridade nas memórias de programa e de microprograma pode ser observada no Apêndice I. O circuito raiz é chamado de *soc_8051_bs.vhd*.

5 Experimentos Práticos

Neste capítulo são apresentados alguns testes aplicados ao microcontrolador modificado e descrito no capítulo anterior, principalmente testes relacionados à cadeia interna de varredura. A idéia é mostrar algumas possibilidades, visto que não é objetivo desta dissertação esgotar o teste deste microcontrolador, mas elaborar um chip que permita o teste de acordo com a norma IEEE1149.1. Portanto, apenas alguns testes serão apresentados, a fim de demonstrar que o circuito funciona para a atividade a que se destina.

Foi utilizada a ferramenta MaxPlus II versão 9.23 nos experimentos (www.altera.com), tanto para o projeto como para os testes e simulações.

Em um primeiro momento foram feitos experimentos com o circuito do microcontrolador em partes. São apresentados resultados de síntese e simulações de circuitos descritos em VHDL com as seguintes características:

- 8051 original;
- TAP sem os sinais para a cadeia de varredura interna;
- TAP com os sinais para a cadeia de varredura interna;
- Memória RAM com BIST;
- 8051 apenas com RAM com BIST;
- 8051 somente com a cadeia interna de varredura;
- 8051_bs sem a cadeia interna e nem BIST, apenas a cadeia BS e o TAP;
- 8051 completo mas sem BIST de RAM;
- 8051 completo;
- 8051 completo, mas com a cadeia interna com os registradores dispostos em ordem diferente.

Várias tabelas apresentam o tipo de circuito, o componente FPGA utilizado, a quantidade de pinos de entrada e saída, a memória e a quantidade de células lógicas utilizadas. Além disso, pode-se visualizar dados sobre o desempenho do circuito, como o período de relógio e, portanto, a freqüência máxima de operação.

5.1 Simulações e comparativos

5.1.1 8051 original:

Na tabela 5.1 pode-se observar o desempenho do microcontrolador 8051 original. O número de células utilizadas é relativamente baixo e o desempenho é da ordem de 11,73MHz. O chip-alvo selecionado foi o da família Flex 10k20. Isto porque utilizou-se a mesma família para alguns dos testes subseqüentes. Procurou-se manter, quando possível, a mesma família, principalmente para a verificação de desempenho e verificação de área ocupada, visto que, dependendo do tipo de chip, obtém-se inclusive desempenhos diferentes.

TABELA 5.1 - 8051 original, sem teste ou alterações

Componente	Dispositivo	Memória utilizada	% Memória utilizada	Células usadas	% Células usadas
soc_8051	EPF10K20TC144-3	9216	75	282	24
Desempenho					
Período	85,2 ns				
Freqüência	11,73 MHz				

5.1.2 TAP sem os sinais para a cadeia de varredura interna:

O circuito do TAP foi sintetizado e testado em separado, com a finalidade de testar bloco a bloco todos os componentes necessários ao circuito. Na realidade, utilizou-se a técnica de dividir para conquistar (*divide-to-conquerer*), onde o circuito é particionado em blocos e em seguida se trabalha no nível de cada bloco em separado, visando-se assim diminuir a complexidade e facilitar os testes do código VHDL. O TAP em separado suportou um relógio de 27,17MHz e ocupou muito pouco do chip da família Flex 10k10, conforme demonstra a tabela 5.2.

TABELA 5.2 - TAP completa, porém sem os sinais para a cadeia interna

Componente	Dispositivo	Memória utilizada	% Memória utilizada	Células usadas	% Células usadas
TAP	EPF10K10LC84-3	0	0	87	15
Desempenho					
Período	36,8 ns				
Freqüência	27,17 MHz				

Esta síntese não tem os sinais de controle para as células participantes da cadeia de varredura interna, visto que são testadas sínteses com e sem esta cadeia. Os resultados obtidos para o TAP com os sinais de controle da cadeia de varredura interna podem ser vistos na tabela 5.3.

5.1.3 TAP com os sinais para a cadeia de varredura interna:

TABELA 5.3 - TAP completa, com todos os sinais necessários

Componente	**Dispositivo**	**Memória utilizada**	**% Memória utilizada**	**Células usadas**	**% Células usadas**
TAP	EPF10K10LC84-3	0	0	116	20
Desempenho					
Período	37,2 ns				
Freqüência	26,88 MHz				

Esta síntese do TAP inclui os sinais necessários ao circuito do microcontrolador com a cadeia de varredura interna. O relógio ficou em 26,88MHz, ou seja, pouco menor que a síntese do TAP sem os sinais. Isto pode ser explicado pelo aumento da complexidade da parte de controle do próprio TAP, o que é confirmado pelo aumento da área utilizada no chip-alvo desta síntese.

5.1.4 Memória RAM com BIST:

Os resultados da síntese obtidos apenas para a memória, acrescida de BIST baseado no algoritmo March, aparecem na tabela 5.4. Na verdade, o desempenho obtido se refere ao algoritmo em si, suas partes operativa e de controle, e ficou limitado em 9,72MHz.

TABELA 5.4 - Apenas a memória RAM, com o circuito completo para o BIST

Componente	Dispositivo	Memória utilizada	% Memória utilizada	Células usadas	% Células usadas
mem_c_bist	EPF10K10LC84-3	2048	33	502	87
Desempenho					
Período	102,8 ns				
Freqüência	9,72 MHz				

5.1.5 8051 apenas com RAM com BIST:

A partir da tabela 5.5, tem-se os testes dos vários códigos VHDL desenvolvidos que incluem o microcontrolador. Neste caso particular, foi utilizado o 8051 original. Este circuito apresenta paridade na memória ROM e na memória de microinstruções. O BIST com algoritmo March é implementado apenas na RAM, conforme explicado anteriormente. Percebe-se que a área utilizada do chip FPGA aumentou bastante, de 282 para 849 células. Isto se deve ao circuito BIST, que é bastante oneroso em termos de células lógicas. Convém salientar que esta memória é bastante pequena, de 256 x 8 bits, ou seja, 2Kbits. Se a memória fosse muito maior, o circuito March seria praticamente o mesmo, justificando muito mais a sua implementação, frente à área ocupada.

TABELA 5.5 - O microcontrolador 8051 original apenas com BIST

Componente	Dispositivo	Memória utilizada	% Memória utilizada	Células usadas	% Células usadas
soc_8051_st	EPF10K20TC144-3	9728	79	849	73
Desempenho					
Período	109,6 ns				
Freqüência	9,12 MHz				

5.1.6 8051 somente com a cadeia interna de varredura:

Para fins comparativos, a síntese descrita na tabela 5.6 foi elaborada a partir do microcontrolador apenas com a cadeia de varredura interna, com os registradores modificados. Pode-se observar que o acréscimo de células frente ao microcontrolador original foi da ordem de 3,5 vezes, provavelmente pelas novas linhas de conexão que são necessárias, além da substituição das estruturas originais pelas células tipo int_cell e bs_cell. Este tamanho inicialmente assusta, mas deve ser levado em conta que o circuito em questão é muito pequeno. Segundo informações de funcionários de empresas ligadas ao setor de microeletrônica, a inclusão de teste deste tipo se justifica

em circuitos maiores, onde a relação do aumento do número de células utilizado frente ao circuito original sem teste é muito menor. Quanto à freqüência de relógio, a redução foi significativa, ficando da ordem de 2,86MHz.

TABELA 5.6 - O microcontrolador 8051 apenas com a cadeia interna de varredura

Componente	Dispositivo	Memória utilizada	% Memória utilizada	Células usadas	% Células usadas
soc_8051	EPF10K20TC144-3	9216	75	988	85
Desempenho					
Período	349,6 ns				
Freqüência	2,86 MHz				

5.1.7 8051_bs sem a cadeia interna de varredura e nem BIST, apenas a cadeia BS e o TAP:

O circuito cujo resumo da síntese aparece na tabela 5.7 não tem a cadeia interna de varredura, mas está provido da cadeia boundary-scan conectando o *núcleo* do chip aos pinos de entrada e de saída, além do TAP para controle do BS. O aumento de área é menor do que o acréscimo obtido com a cadeia interna, bem como o relógio é maior. Neste caso são dois relógios: tck, para o TAP e clock, para o restante do circuito. Os valores de desempenho não diferem muito dos encontrados para o 8051 original.

TABELA 5.7 - O microcontrolador 8051 completo, mas sem a cadeia interna de varredura

Componente	Dispositivo	Memória utilizada	% Memória utilizada	Células usadas	% Células usadas
soc_8051_bs	EPF10K20TC144-3	9216	75	602	52
Desempenho					
Período	86,2 ns				
Freqüência	11,60 MHz				
Período (tck)	45,4 ns				
Freqüência(tck)	22,02 MHz				

5.1.8 8051 completo mas sem BIST:

O 8051 a que se referem os resultados da tabela 5.8 não tem BIST mas tem todos os demais componentes necessários ao teste, incluindo a cadeia BS, a cadeia interna e o TAP. O número de células é o maior encontrado até o momento, como não poderia deixar de ser, já que o número de estruturas envolvidas é também maior. Como era previsível, o desempenho diminuiu, de 11,60MHz e 22,02MHz para 9,14MHz e 15,87MHz, para clock e tck respectivamente.

TABELA 5.8 - O microcontrolador 8051 completo sem BIST

Componente	Dispositivo	Memória utilizada	% Memória utilizada	Células usadas	% Células usadas
soc_8051_bs	EPF10K30RC208-3	9216	75	1321	76
Desempenho					
Período	109,3 ns				
Freqüência	9,14 MHz				
Período (tck)	63 ns				
Freqüência(tck)	15,87 MHz				

5.1.9 8051 completo:

Finalmente, tem-se na síntese documentada pela tabela 5.9 o circuito completo com teste. A freqüência de 11,73MHz caiu para 8,33MHz (considerando o menor relógio como o maior possível para a implementação) e a quantidade de células, que era de 282 subiu para 1838. São acréscimos bastante expressivos, mas que se justificam pela quantidade de código inserida no contexto do chip e o aumento de funcionalidade. Há de se pensar se justifica um incremento desta ordem para se realizar o teste, até porque pode-se relacionar que quanto mais código, ou circuito,

maior a área onde pode-se ter erros. Porém, como foi mencionado anteriormente, no caso de um chip pequeno e em se tratando de uma síntese para um chip FPGA com o MaxPlus, estes valores são aceitáveis. Isto porque parte-se da hipótese de que o roteamento do sintetizador em questão utiliza muitas células para realizar as interconexões que, no caso de uma cadeia interna de teste, são muitas, formando o caminho serial, a cadeia interna de varredura.

Com relação a como é realizada a síntese pelo algoritmo do MaxPlus, não se conseguiu informações, mesmo através de contato direto com o fabricante do simulador (*support@altera.com*).

TABELA 5.9 - O microcontrolador 8051 completo com teste

Componente	Dispositivo	Memória utilizada	% Memória utilizada	Células usadas	% Células usadas
soc_8051_bs	EPF10K30RC208-3	9728	19	1838	79
Desempenho					
Período	120 ns				
Freqüência	8,33 MHz				
Período (tck)	61,5 ns				
Freqüência(tck)	16,26 MHz				

Uma hipótese levantada a partir de então foi de que, de acordo com o roteamento utilizado pelo algoritmo do MaxPlus, este poderia gerar um circuito de forma diferente a partir da mudança da ordem da cadeia interna de varredura. Os resultados podem ser verificados na tabela 5.10.

5.1.10 8051 completo, mas com a cadeia interna com os registradores dispostos em ordem diferente:

O circuito que aqui se trata é o mesmo que o completo simulado anteriormente, mas com uma ordem diferente na cadeia interna. A escolha da cadeia foi feita de forma empírica e aleatória. O resultado pode ser observado: obteve-se o mesmo número de células. Houve apenas uma pequena alteração nos valores de clock e tck, onde relógio aumentou de 8,33MHz para 8,53MHz e tck diminuiu de 16,26MHz para 14,34MHz. Ou seja, o valor a ser utilizado tanto para clock como para tck será de 8,53MHz para o teste.

TABELA 5.10 - O microcontrolador 8051 completo com teste, mas com uma disposição diferente dos registradores da cadeia interna

Componente	Dispositivo	Memória utilizada	% Memória utilizada	Células usadas	% Células usadas
soc_8051_bs	EPF10K20TC144-3	9728	19	1838	79

Desempenho	
Período	117,1 ns
Freqüência	8,53 MHz
Período (tck)	69,7 ns
Freqüência(tck)	14,34 MHz

Tentou-se mudar a ordem dos registradores e analisar-se a partir da comparação quais as alterações perceptíveis. As diferenças foram muito pequenas. Porém, o que seria fator importante – o número de células utilizado – não houve alteração alguma. Apenas pôde-se notar ligeira alteração na velocidade máxima possível do relógio (clock) do sistema.

Em um segundo momento, utilizou-se um componente de uma família FPGA com maior quantidade de células, mas isto também não levou a variação alguma na quantidade de células. Imaginou-se que porventura o ambiente tendo mais espaço possível para rotear as interconexões das diversas partes do circuito, poderia resultar em uma otimização da quantidade de células, ou mesmo um aumento na quantidade utilizada, com alguma mudança com relação ao relógio. Mesmo a mudança de parâmetros da síntese não levou a alterações no resultado final. Ou seja, os ajustes possíveis à ferramenta se mostraram inócuos.

A tabela 5.11 resume os resultados dos testes com componentes FPGA.

TABELA 5.11 - Quadro-resumo dos testes com componentes FPGA

Função	Dispositivo	Memória utilizada	Células utilizadas	Total células	Freqüência(MHz)	% Células usadas	% Acréscimo células
soc_8051 original	EPF10K20TC144-3	9216	282	9498	11,73	24	100
soc_8051 original	EPF10K30ATC144-1	9216	282	9498	19,41	16	100
8051 com teste interno, sem BS	EPF10K20TC144-3	9216	979	10195	10,98	84	107,3
8051 com teste interno completo sem BIST	EPF10K30ATC144-1	9216	1321	10537	10,25 (Clock)	76	110,93
					16,39 (TCK)		
8051 completo	EPF10K40RC208-3	9728	1839	11567	8,88 (Clock)	79	121,78

Observa-se que o acréscimo no número de células lógicas usadas, tomando por base o circuito original frente ao circuito completo com BIST, foi de 6,52 vezes. Este acréscimo não considera as células de memória do FPGA utilizadas na síntese do circuito.

Um raciocínio interessante pode ser retirado deste comparativo. Tem-se 48 pinos de entrada/saída na cadeia BS. Cada flip-flop utiliza uma célula lógica e cada MUX também utiliza uma única célula lógica. Estes dados foram retirados de experiências práticas. Como cada célula BS tem dois flip-flops e dois muxes, tem-se 4 células lógicas. Portanto, 48 x 4 = 192 células lógicas. O TAP utiliza 116 células lógicas. Totalizando, tem-se: 282 células (circuito original) + 116 células (TAP) + 48*4 (192 células da cadeia BS) = 590 células. Na prática obteve-se 602 células. Um valor aproximado que pode ser considerado correto, tendo em vista que também são usadas células para interconexão das unidades lógicas funcionais.

Continuando na mesma linha de raciocínio para o circuito completo, sem o BIST na memória RAM, tem-se 127 bits na cadeia interna, ou seja, são 127 células do tipo int_cell, resultando 127*4 = 508 células. Somando-se então o circuito com a cadeia BS, tem-se 590 + 508 = 1098 células. Na prática obteve-se 1321 células.

Agora, considerando o circuito sintetizado em sua globabilidade, ou seja, incluindo na comparação as células de memória do FPGA utilizadas, os percentuais de acréscimo tornam-se bastante distintos. Para efeitos de comparação, consideramos que uma célula de memória apresenta área equivalente à de uma célula lógica. Nestas circunstâncias, a área total de cada implementação é dada pela soma das colunas 3 (memória utilizada) e 4 (células utilizadas) da tabela 5.11. Os números obtidos passam a ser aqueles apresentados abaixo da tabela 5.11. Assim sendo, o 8051 com todas as funcionalidades de teste apresenta um acréscimo de 21,78% na quantidade de

estruturas necessárias ao teste. Isto posto, pode-se considerar que um aumento desta ordem seria observado no caso de uma implementação ASIC deste circuito.

Finalmente, observando-se os dados acima, conclui-se que o acréscimo na quantidade de células não é tão significativo em termos percentuais. Valores muito semelhantes são verificados em outras implementações de circuitos com teste quando comparados ao mesmo circuito original sem teste.

5.1.11 Testes com as ferramentas Mentor: ModelSim e LeonardoSpectrum:

Afim de ter-se um comparativo em relação à ferramenta MaxPlus, optou-se por sintetizar o mesmo código fonte através das ferramentas Mentor ModelSim PE 5.5e e LeonardoSpectrum 2001.1d (www.mentor.com). A primeira permite a modelagem do código, a compilação deste e a verificação de erros. Além disso, é possível a simulação do funcionamento do circuito. A segunda ferramenta, LeonardoSpectrum, permite a síntese do circuito, em FPGA ou ASIC, otimizações e análises de temporização.

Após vários testes, pôde-se verificar alguns aspectos interessantes a respeito da síntese com estas ferramentas. Como era de se esperar, o código VHDL original, criado e compilado no MaxPlus, não executa sem problemas tanto no LeonardoSpectrum quanto no ModelSim.

O MaxPlus parece ser o sistema mais básico e simples, que aceita interconexões entre sinais de tipos diferentes, como conexões entre variáveis do tipo std_logic e unsigned, e outros tipos. Nas ferramentas ModelSim e LeonardoSpectrum isto não é possível. Os tipos devem ser os mesmos, caso contrário não se consegue compilar o código. E a sintaxe, bem como a forma de construir as estruturas, são diferentes, forçando o programador a ser muito mais criterioso quanto à elaboração do código.

Quanto ao Leonardo, é sem dúvida o mais complicado de ser utilizado. Quaisquer detalhes e recebe-se *warnings* e outras mensagens de erros mais graves que impossibilitam a compilação. Resumindo, o MaxPlus é a ferramenta mais simples e fácil de ser utilizada, mas também a que oferece menor quantidade de recursos.

Ao final, foi necessário, antes de mais nada, modificar o circuito original e o de teste, de forma a poder realizar a compilação e a simulação no ModelSim. De posse do circuito, agora compilado, passou-se para o LeonardoSpectrum. Novos problemas foram encontrados, principalmente em alguns pontos do circuito onde há processos com alguns sinais que devem ser realimentados, tais como em flip-flops. Resolvidos estes pontos, compilou e sintetizou para FPGA. Pôde-se verificar que a diferença entre o circuito com e sem teste permaneceu grande. Foram então realizadas outras sínteses, para componentes-alvo diferentes. Alguns aspectos mudaram, como velocidade e quantidade de células lógicas usadas, mas percebe-se que o circuito completo ficou

com tamanho próximo de dez vezes maior que o original em todas as sínteses realizadas.

O mesmo circuito foi compilado para um chip ASIC. Acreditava-se que, em um comparativo entre a síntese de componentes ASIC, a diferença entre o circuito com e sem teste diminuiria substancialmente. Na verdade, observou-se que as diferenças entre o circuito com e sem teste, na mesma tecnologia, permaneceram muito próximas em termos percentuais, como pode ser visto na tabela 5.12.

TABELA 5.12 - Quadro-resumo dos testes com componentes ASIC

Função	Library	Área (um2)	Clock (MHz)
soc_8051 original	AMS 0,35um - csx_HRDLIB	29156	306,6
8051 completo	AMS 0,35um - csx_HRDLIB	261534	138
		Gates	
soc_8051 original	Sample scl05u	1380	501,6
8051 completo	Sample scl05u	11179	138,2

O crescimento de área utilizando-se a biblioteca AMS, como pode ser observado na tabela 5.12, foi de 8,97 vezes. Com relação ao número de *Gates*, no caso da compilação com a biblioteca Sample, obteve-se um aumento de 8,1 vezes.

5.2 O teste fazendo uso da cadeia interna de varredura:

Um teste possível, que utiliza a cadeia interna de varredura, é o teste sobre o registrador de instruções do 8051. Para isto, em primeiro lugar, é necessário carregar-se a instrução ScanInt no TAP. Esta instrução vai selecionar a cadeia interna entre os pinos TDI e TDO. Partindo-se então de um estado inicial de reset (figura 3.6 em estado de test_logic_reset), são necessários 2 pulsos de TCK com a entrada de TMS em "1" e TDI em "0", logo após TDI="1", TMS="0" e outros dois pulsos de TCK. Desta forma é colocada a instrução Scanint no registrador de instrução do controlador TAP. A partir de então, anda-se nos diversos estados do controlador de forma a seguir-se a coluna correspondente à manipulação do registrador selecionado pela instrução. Neste caso, o registrador scan interno, conforme pode ser visto na figura 5.1.

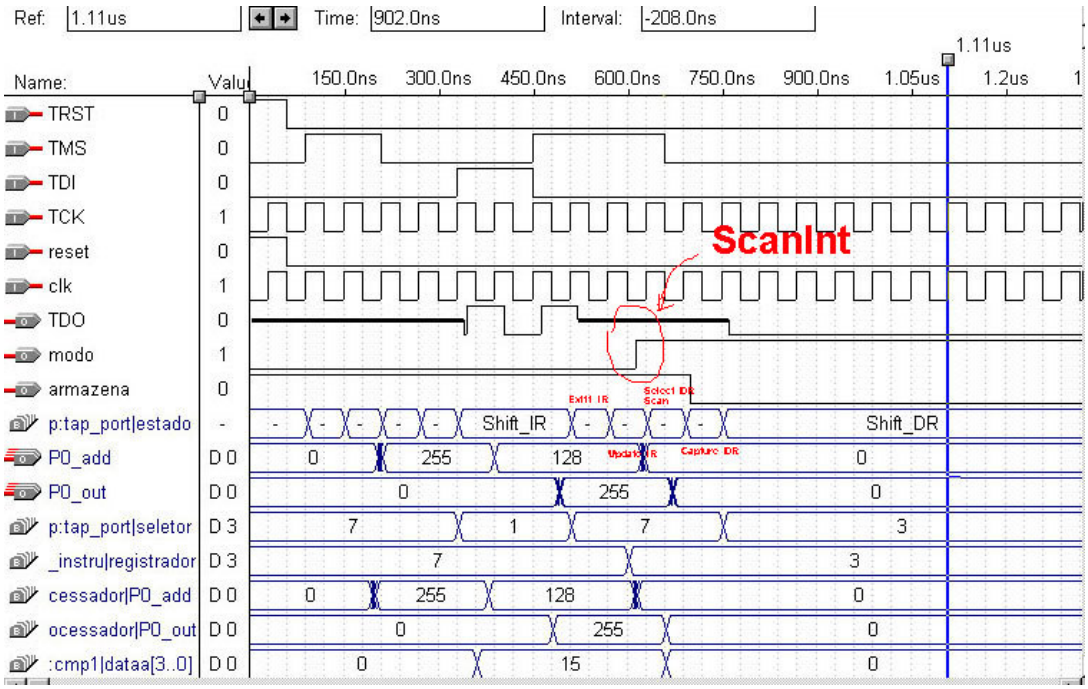

FIGURA 5.1 - Simulação da instrução Scanint

Quando o TAP estiver com o seu registrador de estados no estado de Shift_DR, inserem-se os valores na cadeia, de acordo com o teste necessário. Por exemplo, para testar-se o registrador de instruções do 8051, envia-se primeiramente FF para este registrador, com o trem de pulsos "1111111100000000000" através de TDI. Estes dados passarão na cadeia interna por estado_atual, pc_out até chegarem ao registrador destino. Ao final de 19 pulsos de tck (3 para estado_atual + 8 para PC_out + 8 para o próprio registrador de instruções), teremos registrador de instruções="FF", estado_atual="00" e PC_out="00". Passa-se ao estado de Update_DR e estes dados que estão na cadeia passarão efetivamente aos registradores em um pulso de TCK. Como o registrador estado_atual do 8051 tem valor diferente de est1 (‘001’), então IR – o registrador de instruções do controlador TAP - recebe o sinal IR_BS, ou seja, a entrada do registrador é realimentada pelo próprio valor de IR. Isto pode ser observado no esquemático da figura 4.5.

Com isto, pode-se novamente ir com o TAP ao estado de Shift_DR e capturar os dados da entrada do registrador para o flip-flop da cadeia e deslocar estes dados até a saída. Como a cadeia tem 127 bits, ter-se-á 127 - 3 (estado_atual) - 8 (PC_Out) = 116 ciclos até obter-se o conteúdo de IR através de TDO. Os dados iniciam em 116-

8=108 ciclos e terminam em 116 ciclos. Estes resultados podem ser vistos na figura 5.2.

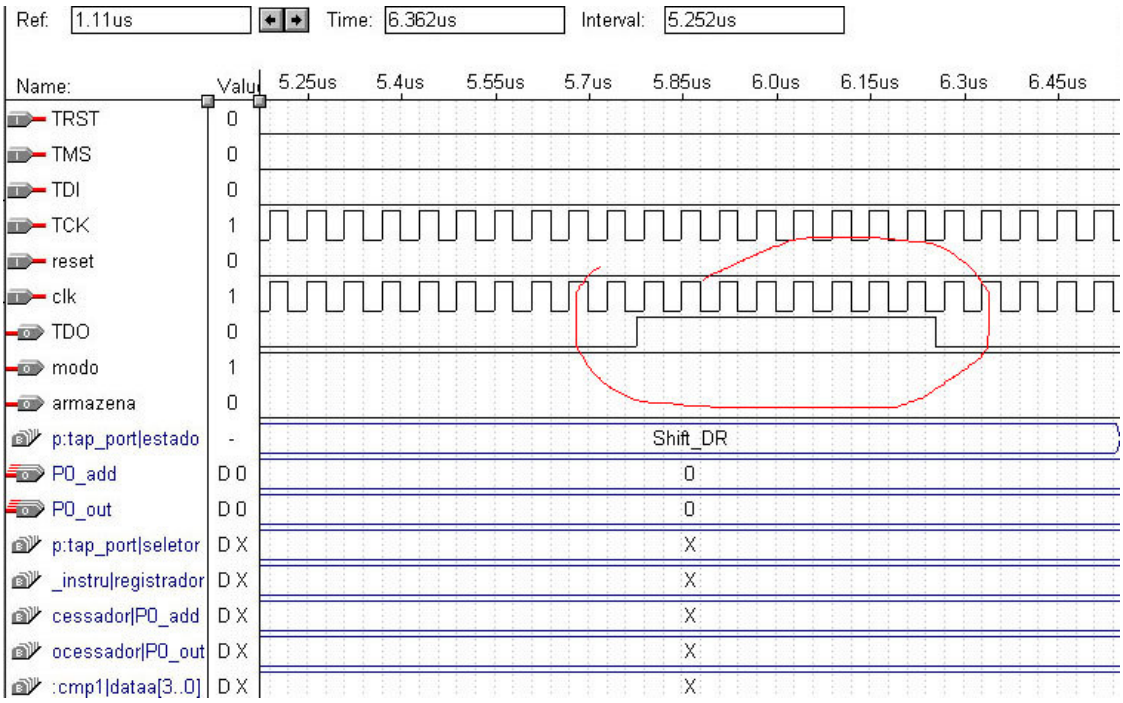

FIGURA 5.2 - Simulação dos dados de IR saindo via TDO

Feito isto, procede-se o mesmo teste para o dado 00 neste registrador. Com isto, podemos testar falhas de *stuck-at* (1 ou 0) no registrador de instruções do 8051.

Estes testes podem ser feitos para todos os registradores. Alguns podem ser testados simultaneamente, dependendo do tamanho do registrador e a ordem na cadeia de varredura. Inicialmente pode-se assumir que registradores de mesmo tamanho de palavra podem ser testados ao mesmo tempo, mas isto vai depender de como cada multiplexador que realimenta os registradores se portará de acordo com o registrador estado_atual.

5.3 O teste de memória baseado no algoritmo March

A figura 5.3 apresenta a simulação do funcionamento do algoritmo March. A execução leva pouco mais de 3ms e é ativada através do TAP com a instrução

RunBIST. Nota-se que após pouco mais de 3ms o sinal de fim fica ativo, indicando o término da execução do algoritmo de teste.

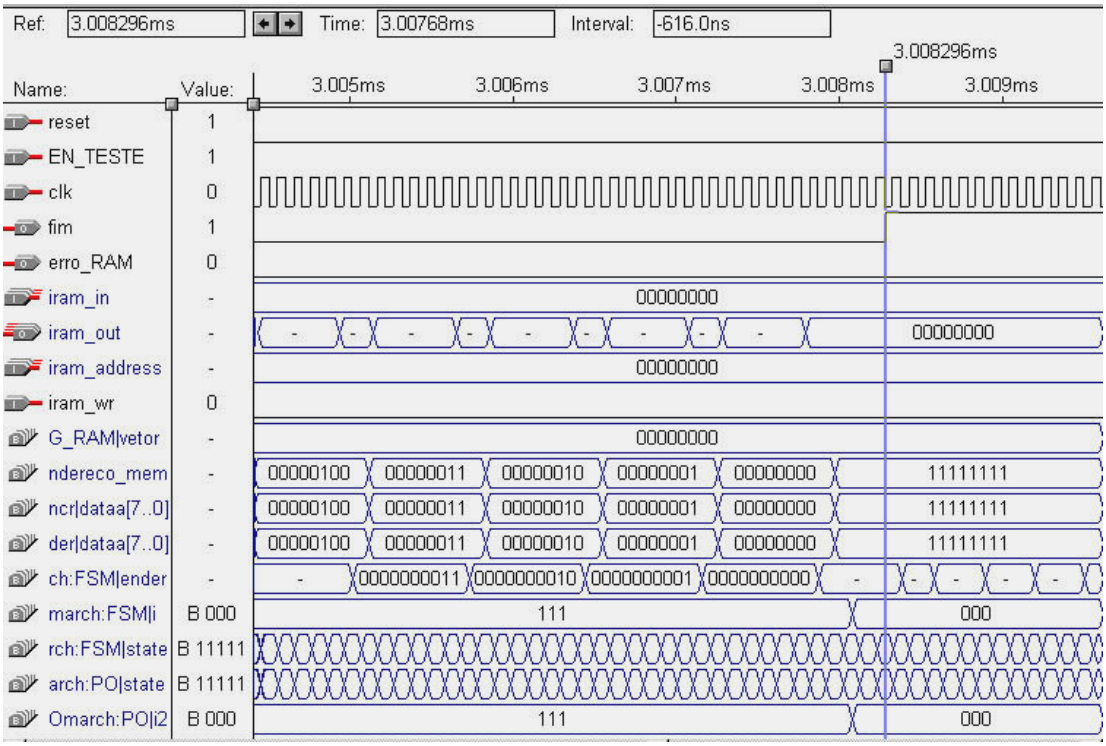

FIGURA 5.3 - Simulação de execução de um teste BIST na memória RAM

5.4 Geração automática e aplicação dos testes

Existem ferramentas de geração automática de testes, como nos sistemas Mentor e Synopsis, que procuram pelos registradores existentes e definem testes para os módulos conectados a estes registradores, de forma que estes possam não somente levar vetores de teste a um módulo específico, mas também receber respostas de outro ou do mesmo módulo. A definição de quais vetores de teste aplicar pode ser feita de diversas formas: determinística (ou estrutural), aleatória, pseudo-aleatória, funcional, etc. A forma pseudo-aleatória é a mais comum mas, em geral, é necessário se fazer uma análise de testabilidade e simulação de falhas no circuito para verificação da cobertura de falhas e da qualidade dos vetores gerados. O problema com essas ferramentas é que como devem funcionar para diversos circuitos, a maioria delas nem sempre gera a melhor solução e nem sempre dá opções para o projetista, ou seja, geram uma solução baseada na informação estrutural do circuito apenas. Esta é uma solução rápida, boa para circuitos grandes e complexos, mas nem sempre ótima. Apesar disso, neste trabalho supõe-se que os vetores a serem aplicados aos módulos

internos do 8051 através da cadeia de varredura implementada serão gerados por tais ferramentas.

Do ponto de vista da aplicação do teste e portanto da definição de clock_BS (relógio do circuito do TAP que será aplicado aos registradores internos ao chip que, por sua vez, fazem parte da cadeia de varredura), algumas considerações são necessárias. O relógio original, que alimentava todos os registradores do microcontrolador, foi substituído por clock_BS. Isto porque clock_BS precisa ser controlado a partir do TAP, para que ela tenha controle sobre o que está sendo realizado dentro do chip. Clock_BS também é o sinal de controle do próprio TAP.

Portanto, durante o funcionamento normal do microcontrolador, clock e clock_BS devem necessariamente ter o mesmo valor, de forma que o sincronismo dentro e fora do chip seja preservado. Durante o funcionamento normal, deve-se manter o sinal de TMS em nível alto (‘1’), afim de que a máquina de estados do TAP permaneça em Test_Logic_Reset, ou seja, sem influir no funcionamento do circuito. No caso de se querer realizar um teste, por exemplo de amostragem, durante o funcionamento do chip, basta trabalhar a sinalização em TMS de forma a levar o controlador TAP a um estado de carga de instrução e de carga de sinais do chip ou para o chip.

Convém lembrar que a instrução Sample/Preload não influencia no funcionamento do chip e pode-se amostrar dados a partir de pinos de entrada e/ou saída. Outro dado importante é que a descrição inicial sem teste foi alterada pois recebia um relógio único e o dividia por 5 para uso nas unidades funcionais. Esta parte foi removida, colocando o relógio direto ao processador em todos os pontos. Esta divisão por 5 é realizada devido ao circuito original da caneta, onde o relógio é 5 vezes maior que o suportável pelo microcontrolador original. O relógio é mantido sincronizado com clock_BS apenas durante o funcionamento normal. Durante o teste, o sinal clock é mantido parado.

Uma pergunta pode surgir. Se o sinal clock_BS tem valor diferente do sinal clock e os registradores internos recebem clock_BS na descrição com teste mesmo durante o funcionamento normal, qual o valor que deve ser utilizado?

O valor de clock_BS e clock deve ser o menor entre os dois. Ou seja, se tem-se 10MHz em clock_BS e 9MHz em clock, utiliza-se 9MHz nos dois sinais.

5.5 Testes registrador a registrador

Pode-se realizar diversos testes com o microcontrolador a partir da implementação da infra-estrutura descrita no capítulo anterior. O mais básico é a sinalização de erro de paridade nas memórias ROM e de microinstrução, além da execução do BIST na memória RAM, esta ativada a partir da cadeia selecionada pela instrução RunBIST no controlador TAP. Para a verificação de paridade, foi inserido

um bit a mais em cada palavra constante nas memórias onde o teste em funcionamento é realizado.

Outra modificação necessária foi feita na máquina de controle. Como o registrador estado_atual é utilizado em diversos testes, é necessário conhecer a codificação dos estados para que os caminhos corretos sejam selecionados. Assim, foi feita uma codificação manual dos estados na máquina de controle.

Quanto aos registradores e blocos combinacionais internos ao microcontrolador, os vetores devem ser capazes de detectar o maior número de falhas possível. O ideal é que os vetores de teste sejam gerados por algum TPG para scan, de forma a aumentar a cobertura e diminuir o número total de vetores de teste. Não havia ferramentas deste tipo disponíveis, então foram definidos vetores que detectassem falhas *stuck-at* apenas nos circuitos acessíveis pelos registradores de varredura.

A idéia básica é excitar caminhos nas redondezas dos registradores envolvidos nos testes de forma que eles sejam carregados com valores vindos de locais onde as falhas podem ocorrer.

Alguns exemplos de testes podem ser idealizados, a partir desta estrutura:

- Teste do registrador de instruções do 8051: verifica o multiplexador (mux) na entrada do registrador e o próprio registrador;
- Teste dos registradores ULA1 e ULA2: verifica os quatro multiplexadores associados aos registradores ULA1 e ULA2 e os próprios registradores;
- Teste eprom_data: verifica o mux na entrada do registrador eprom_data e o próprio registrador;
- Teste de iram_wr: verifica o mux de entrada e registrador iram_wr;
- Teste de PC_write: verifica os multiplexadores e o registrador PC_write;
- Teste de iram_address: verifica o mux e o registrador iram_address;
- Teste da ULA: verifica a Unidade Lógica e Aritmética.

5.5.1 Os testes

Os testes são divididos em diferentes fases, de forma a se verificar falhas de *stuck-at-0* e *stuck-at-1*. São informados os valores que deverão constar em cada registrador, tanto para controle como para observação. Todos os sinais são inseridos via TAP, com a instrução de ativação da cadeia interna (ScanInt).

Um ponto importante também é testar-se primeiramente a estrutura scan (seção 3.5), a fim de ter-se a certeza de que a estrutura está sem problemas e que então não irá influir no teste, indicando erros em partes corretas do circuito.

Alguns pontos da memória foram alterados a fim de alimentar com dados certos registradores que estão em teste. Foram modificadas três posições da memória DECOD e duas posições da memória de programa para que os requisitos pudessem ser atendidos. As modificações são:

ROM interna (memória de programa):

Posição "FE"H = "00"H (mantido o valor original = "00"H)

Posição "FF"H = "FF"H (valor original = "00"H)

ROM DECOD (microinstruções):

Posição "AA"H = "004280"H (valor original = "000000"H)

Posição "AB"H = "002500"H (valor original = "000000"H)

Posição "AC"H = "00EF80"H (valor original = "000000"H)

Estas modificações têm sentido prático quando da aplicação dos testes. No caso da memória ROM, é necessário testar e comparar os registradores em teste com os valores “00”H e “FF”H, para detectar-se falhas de *stuck-at-1* e *stuck-at-0.* No caso da memória ROM DECOD, os valores inseridos nas posições descritas fazem com que os sinais de controle correpondentes aos sinais de alguns circuitos de MUX sejam ativados, selecionando determinados caminhos nos registradores ULA1 e ULA2.

É importante salientar que os casos estudados a seguir são apenas um exemplo basetante concreto de como seria um teste simples para múltiplas possibilidades de falhas nos registradores, multiplexadores e interconexões. Sinalizações de carga, rotações e controles do TAP estão sendo subentendidos na análise para fins de simplificação.

5.5.1.1 Teste de IR (2 fases)

Explicando-se este teste, primeiramente insere-se via cadeia scan o valor “00” no registrador IR do 8051. O registrador estado_atual deve receber qualquer valor que seja diferente de estado1(diferente de est_1 = “001”). Feito isto, após um pulso de relógio, verifica-se se IR continua com o valor “00”. Se isto se confirmar, o registrador está sem falhas para a primeira fase de seu teste. Repete-se o processo para o valor “FF” da mesma forma. E assim sucessivamente para todos os testes que seguem.

Fase I:

IR = "00"H

estado_atual <> est1 (diferente de estado 1, qualquer estado)

observar se IR = "00"H (após 1 pulso de tck) , o que indica que o multiplexador realimentou o registrador como se esperava.

Fase II:

IR = "FF"H

estado_atual <> est1 (diferente de estado 1, qualquer estado, onde IR<=IR)

observar se IR = "FF"H (após 1 pulso de tck), o que indica que o multiplexador realimentou o registrador como se esperava.

Em quaisquer dos casos, se algum bit for diferente do esperado, teremos uma situação de *stuck-at*, ou seja, falha no registrador, no multiplexador de entrada ou nas suas conexões.

5.5.1.2 Teste ULA1 e ULA2 (10 fases)

Fase I:

estado_atual = est4 -- estado 4, onde ula_1 <= mux_1 e ula_2 <= mux_2

acc = "00"H

auxmux2 = "00"H

IR = "00"H (microinst(10..7) = "0000"B, selecionando auxmux2 em mux2. Isto porque os bits 10 a 9 da microinstrução controlam mux1 e 8 a 7 controlam mux2.)

observar se ULA1 = ULA2 = "00"H (após 1 pulso de tck)

Fase II:

estado_atual = est4 -- estado 4, onde ula_1 <= mux_1 e ula_2 <= mux_2

acc = "FF"H

auxmux2 = "FF"H

IR = "00"H (microinst(10..7) = "0000"B , selecionando auxmux2 em mux2)

observar se ULA1 = ULA2 = "FF"H (após 1 pulso de tck)

Fase III:

estado_atual = est4 -- estado 4, onde ula_1 <= mux_1 e ula_2 <= mux_2

iram_out = "00"H

IR = "AA"H (microinst(10..7) = "0101"B, selecionando iram_out direto para a saída de ambos os muxes de entrada de ULA1 e ULA2.)

observar se ULA1 = ULA2 = "00"H (após 1 pulso de tck)

Fase IV:

estado_atual = est4 -- estado 4, onde ula_1 <= mux_1 e ula_2 <= mux_2

iram_out = "FF"H

IR = "AA"H (microinst(10..7) = "0101"B, selecionando iram_out direto para a saída de ambos os muxes de entrada de ULA1 e ULA2)

observar se ULA1 = ULA2 = "FF"H (após 1 pulso de tck)

Fase V:

estado_atual = est4 -- estado 4, onde ula_1 <= mux_1 e ula_2 <= mux_2

PC_out = "FE"H (eprom_out = "00"H)

IR = "AB"H (microinst(10..7) = "1010"B)

observar se ULA1 = ULA2 = "00"H (após 1 pulso de tck)

Fase VI:

estado_atual = est4 -- estado 4, onde ula_1 <= mux_1 e ula_2 <= mux_2

PC_out = "FF"H (eprom_out = "FF"H)

IR = "AB"H (microinst(10..7) = "1010"B)

observar se ULA1 = ULA2 = "FF"H

Fase VII:

estado_atual = est4 -- estado 4, onde ula_1 <= mux_1 e ula_2 <= mux_2

PC_out = "00"H

aux2mux2 = "00"H

IR = "AC"H (microinst(10..7) = "1111"B)

observar se ULA1 = ULA2 = "00"H

Fase VIII:

estado_atual = est4 -- estado 4, onde ula_1 <= mux_1 e ula_2 <= mux_2

PC_out = "FF"H

aux2mux2 = "FF"H

IR = "AC"H (microinst(10..7) = "1111"B)

observar se ULA1 = ULA2 = "FF"H

Fase IX:

estado_atual <> est4 (diferente de estado 4, qualquer estado)

ULA1 = ULA2 = "00"H

observar se ULA1 = ULA2 = "00"H (após 1 ciclo de tck)

Fase X:

estado_atual <> est4 (diferente de estado 4, qualquer estado)

ULA1 = ULA2 = "FF"H

observar se ULA1 = ULA2 = "FF"H (após 1 ciclo de tck)

5.5.1.3 Teste eprom_data (6 fases)

Fase I:

estado_atual = est4

PC_out = "FE"H (eprom_out = "00"H)

IR = "AC"H (microinst(11) = '1', ou seja, eprom_data <= eprom_out)

observar se eprom_data = "00"H

Fase II:

estado_atual = est4

PC_out = "FF"H (eprom_out = "FF"H)

IR = "AC"H (microinst(11) = '1', ou seja, eprom_data <= eprom_out)

observar se eprom_data = "FF"H

Fase III:

estado_atual = est4

aux_eprom_data = "00"H

IR = "00"H (microinst(11) = '0', ou seja, eprom_data <= aux_eprom_data)

observar se eprom_data = "00"H

Fase IV:

estado_atual = est4

aux_eprom_data = "FF"H

IR = "00"H (microinst(11) = '0', ou seja, eprom_data <= aux_eprom_data)

observar se eprom_data = "FF"H

Fase V:

estado_atual <> est4

eprom_data = "00"H

observar se eprom_data = "00"H (após 1 ciclo de tck)

Fase VI:

estado_atual <> est4

eprom_data = "FF"H

observar se eprom_data = "FF"H (após 1 ciclo de tck)

5.5.1.4 Teste iram_wr (4 fases)

Fase I:

estado_atual = est6 (onde iram_wr recebe microinst(15).)

IR = "00"H (microinst(15) = '0')

observar se iram_wr = '0' (após 1 ciclo de tck)

Fase II:

estado_atual = est6 (onde iram_wr recebe microinst(15).)

IR = "AC"H (microinst(15) = '1')

observar se iram_wr = '1' (após 1 ciclo de tck)

Fase III:

estado_atual <> est6 (diferente de estado 6, qualquer estado, onde iram_wr <= aux_iram_wr)

aux_iram_wr = '0'

observar se iram_wr = '0' (após 1 ciclo de tck)

Fase IV:

estado_atual <> est6 (diferente de estado 6, qualquer estado, onde iram_wr <= aux_iram_wr)

aux_iram_wr = '1'

observar se iram_wr = '1' (após 1 ciclo de tck)

5.5.1.5 Teste PC_write (7 fases)

Fase I:

estado_atual = est5 (PC_write <= PC_wr)

IR = "81"H (microinst(17) = '0' e IR(7..4) = '1000')

acc <> "00"H (qualquer valor diferente de zero)

OBS: PC_wr <= microinst(17) when "1000";

PC_wr <= (microinst(17) and zero_flag) when "0110";

PC_wr <= (microinst(17) and carry_flag_BS) when "0100";

PC_wr <= '0' when others.

observar se PC_write = '0' (após 1 ciclo de tck)

Fase II:

estado_atual = est5 (PC_write <= PC_wr)

IR = "61"H (microinst(17) = '0' e IR(7..4) = '0110')

acc <> "00"H (qualquer valor diferente de zero)

observar se PC_write = '0' (após 1 ciclo de tck)

Fase III:

estado_atual = est5 (PC_write <= PC_wr)

IR = "41"H (microinst(17) = '0' e IR(7..4) = '0100')

carry_flag = '0'

observar se PC_write = '0' (após 1 ciclo de tck)

Fase IV:

estado_atual = est5 (PC_write <= PC_wr)

IR = "80"H (microinst(17) = '1' e IR(7..4) = '1000')

acc = "00"H

observar se PC_write = '1' (após 1 ciclo de tck)

Fase V:

estado_atual = est5 (PC_write <= PC_wr)

IR = "60"H (microinst(17) = '1' e IR(7..4) = '0110')

acc = "00"H

observar se PC_write = '1' (após 1 ciclo de tck)

Fase VI:

estado_atual = est5 (PC_write <= PC_wr)

IR = "40"H (microinst(17) = '1' e IR(7..4) = '0100')

carry_flag = '1'

observar se PC_write = '1' (após 1 ciclo de tck)

Fase VII:

estado_atual <> est5 (PC_write <= ‘0’)

observar se PC_write = '0' (após 1 ciclo de tck)

5.5.1.6 Teste iram_address (12 fases)

Fase I:

estado_atual = est2 (onde iram_address <= aux_Iram_address)

aux_iram_address = "00"H

observar se iram_address = "00"H (após 1 ciclo de tck)

Fase II:

estado_atual = est2 (onde iram_address <= aux_Iram_address)

aux_iram_address = "FF"H

observar se iram_address = "FF"H (após 1 ciclo de tck)

Fase III:

estado_atual = est3 (onde iram_address <= mux_3)

IR = "00"H (microinst(14..13) = '00', onde mux3 <= aux1_mux3)

aux1_mux3 = "00"H

observar se iram_address = "00"H (após 1 ciclo de tck)

Fase IV:

estado_atual = est3 (onde iram_address <= mux_3)

IR = "00"H (microinst(14..13) = '00', onde mux3 <= aux1_mux3)

aux1_mux3 = "FF"H

observar se iram_address = "FF"H (após 1 ciclo de tck)

Fase V:

estado_atual = est3 (onde iram_address <= mux_3)

IR = "AB"H (microinst(14..13) = '01', onde mux3 <= iram_out)

iram_out = "00"H

observar se iram_address = "00"H (após 1 ciclo de tck)

Fase VI:

estado_atual = est3 (onde iram_address <= mux_3)

IR = "AB"H (microinst(14..13) = '01', onde mux3 <= iram_out)

iram_out = "FF"H

observar se iram_address = "FF"H (após 1 ciclo de tck)

Fase VII:

estado_atual = est3 (onde iram_address <= mux_3)

PC_out = "FE"H (eprom_out = "00"H)

IR = "AA"H (microinst(14..13) = '10', onde mux3 <= eprom_out)

observar se iram_address = "00"H (após 1 ciclo de tck)

Fase VIII:

estado_atual = est3 (onde iram_address <= mux_3)

PC_out = "FF"H (eprom_out = "FF"H)

IR = "AA"H (microinst(14..13) = '10', onde mux3 <= eprom_out)

observar se iram_address = "FF"H (após 1 ciclo de tck)

Fase IX:

estado_atual = est3 (onde iram_address <= mux_3)

IR = "AC"H (microinst(14..13) = '11', onde mux3 <= aux2_mux3)

aux2_mux3 = "00"H

observar se iram_address = "00"H (após 1 ciclo de tck)

Fase X:

estado_atual = est3 (onde iram_address <= mux_3)

IR = "AC"H (microinst(14..13) = '11', onde mux3 <= aux2_mux3)

aux2_mux3 = "FF"H

observar se iram_address = "FF"H (após 1 ciclo de tck)

Fase XI:

estado_atual <> est2 e <> est3 (qualquer outro, onde iram_address <= iram_address)

iram_address = "00"H

observar se iram_address = "00"H (após 1 ciclo de tck)

Fase XII:

estado_atual <> est2 e <> est3 (qualquer outro, onde iram_address <= iram_address)

iram_address = "FF"H

observar se iram_address = "FF"H (após 1 ciclo de tck)

5.5.1.7 Teste da ULA

Neste teste, os registradores ULA1 e ULA2 devem ser carregados com vetores de teste capazes de detectar falhas de *stuck-at* na estrutura da ULA. Os resultados podem ser retirados pela cadeia de varredura interna a partir do registrador Port2_out.

Como a ULA foi implementada como um circuito puramente combinacional, uma alternativa é descrever este circuito como portas lógicas na ferramenta ÁGATA [CAR97] e executar o algoritmo D que foi implementado nesta ferramenta. Esta implementação, porém, gera um vetor para cada falha possível, mas não realiza nenhum tipo de otimização (como falhas equivalentes, atribuição de *don't cares*, etc.). Então compila-se manualmente os vetores gerados para tirar as redundâncias e diminuir o número de padrões, reduzindo e otimizando o tempo necessário ao teste.

6 Conclusões

Este trabalho exemplificou a colocação de teste em um circuito eletrônico pré-projetado, com a criação de uma infra-estrutura baseada na técnica do *boundary scan* para o teste de placas de circuito impresso. Foram discutidos o teste e o projeto orientados ao teste de circuitos digitais e suas implicações em relação a área utilizada e ao desempenho.

Também foi verificada a fundo a norma IEEE1149.1, base para o tipo de teste inserido no circuito do trabalho, o controlador TAP e os tipos de testes possíveis, a criação de instruções e a utilização das instruções obrigatórias e das personalizadas.

Os detalhes desta implementação, as decisões que foram tomadas e o porquê destas foram vistos no capítulo 4, além do diagrama em blocos do 8051 com e sem teste, os detalhes da cadeia de varredura interna e as características da célula básica BS.

No capítulo 5 foram vistos alguns experimentos práticos sobre a implementação desenvolvida, comparando-se o 8051 original e o com teste, além de simulações do teste de memória.

Finalmente, foram elaborados exemplos de testes, demonstrando a funcionalidade do circuito de teste inserido neste núcleo e a possibilidade de detecção de falhas em pontos distintos do sistema.

O objetivo de inserção de estruturas de teste em um circuito eletrônico pré-projetado foi atingido. Nesta dissertação, a infra-estrutura baseada na técnica do *boundary scan,* de acordo com a norma IEEE 1149.1, funcionou perfeitamente, resultando em um circuito testável e em um código VHDL reaproveitável, principalmente no tocante às estruturas do TAP.

A elaboração de exemplos, demonstrando a funcionalidade do circuito de teste inserido neste núcleo, e a possibilidade de detecção de falhas se mostrou satisfatória, dando ampla possibilidade de teste ao chip em questão depois de inserido em um sistema integrado do tipo SOC.

A inclusão do teste no 8051 se mostrou tarefa bastante complicada, pela necessidade do entendimento bastante completo das estruturas e do funcionamento do microcontrolador 8051, bem como da norma de teste em questão. Outro ponto importante foi a escolha dos registradores da cadeia interna de varredura. A fase de depuração foi bastante complexa, o que levou a atrasos em relação às datas iniciais programadas para o fechamento das etapas. A depuração foi bastante penosa, porém de final bastante gratificante devido ao resultado de sucesso atingido.

Quanto à quantidade de células utilizadas na síntese da descrição com teste em relação à descrição sem teste, o aumento observado de células para a implementação do microcontrolador foi significativo. Isto se deve à quantidade de estruturas agregadas ao circuito original e aceitável para este tipo de implementação. O aumento observado de células para a implementação do microcontrolador foi de 21,78% (computando-se circuitos de memória), valor aceitável e justificado. A comparação de ferramentas e versões de circuitos com e sem teste foi aspecto importante, confirmando a independência de ferramenta em relação a implementação do circuito. Isto se comprova pela variação de acréscimo de área/gates muito próxima nas sínteses realizadas.

Foram feitos diversos contatos com o suporte técnico do fabricante (support@altera.com), mas de concreto não foi possível verificar-se nada de diferente do que já havia sido testado e comprovado. O fato é que não conseguiu-se, sob hipótese alguma, controlar a síntese no ambiente MaxPlus, da Altera, de modo a realizar-se alguma variação no sistema em termos de consumo de células em uma estrutura FPGA. Contatos com profissionais da área apontaram também para o fato de que o circuito como um todo, por ser muito pequeno, acaba crescendo em muito seu tamanho ao receber estruturas adicionais.

Uma possível continuação deste trabalho aponta para uma implementação prática. A inclusão e validação dos testes deste microcontrolador 8051 no contexto da caneta tradutora ou ainda a introdução das estruturas aqui desenvolvidas como parte de uma biblioteca. O teste prático pode ser realizado através de um equipamento de teste que atende à norma, o testador Boundary-Scan JTAG PM3705 [JTA95], disponível nos laboratórios da UFRGS. Além disso, a inserção de testes com a norma P1500 no mesmo circuito pode ser realizada, visando o teste em unidades SOC.

Anexo

Ula8.vhd

```
--------------------------------------------------------------------------------
-- Dissertação de Mestrado 99/2                                               --
-- Microcontrolador 8051                                                      --
-- Versão para System-on-Chip                                                 --
-- Circuito da unidade lógica e aritmética de 8 bits                          --
-- Operações: Soma, Subtração, AND, XOR, OR e Rotação à esquerda e direita    --
-- Por Denis Franco em janeiro de 2000 (19.01.00)                             --
--------------------------------------------------------------------------------

entity ula8 is
  port(
       ula_A         : in bit_vector(7 downto 0);  -- Primeiro operando
       ula_B         : in bit_vector(7 downto 0);  -- Segundo operando
       ula_operation  : in bit_vector(6 downto 0);  -- Seleção de operação
       carry_in       : in bit;                  -- Entrada do sinal de carry
       ula_result     : out bit_vector(7 downto 0); -- Resultado da operação
       carry_out      : out bit                  -- Saída do sinal de carry
      );
end ula8;

architecture A_ula8 of ula8 is

signal carry_aux   : bit_vector(7 downto 0); -- sinal de carry interno
signal A           : bit_vector(7 downto 0); -- entrada 1 da ula
signal B           : bit_vector(7 downto 0); -- entrada 2 da ula
signal add_op      : bit_vector(7 downto 0); -- saída da soma (somador tipo ripple-carry)
signal and_op      : bit_vector(7 downto 0); -- saída da operação AND
signal or_op       : bit_vector(7 downto 0); -- saída da operação OR
signal xor_op      : bit_vector(7 downto 0); -- saída da operação XOR
signal rot_op      : bit_vector(7 downto 0); -- saída da operação de rotação

begin

B(0)        <= ula_operation(1) xor ula_B(0); -- complementação da entrada 2 para subtração
and_op(0)   <= A(0) and B(0);
xor_op(0)   <= A(0) xor B(0);
or_op(0)    <= A(0) or b(0);
add_op(0)   <= (xor_op(0)) xor carry_in;
carry_aux(0) <= and_op(0) or (xor_op(0) and carry_in);

B(1)        <= ula_operation(1) xor ula_B(1);
and_op(1)   <= A(1) and B(1);
xor_op(1)   <= A(1) xor B(1);
```

```
or_op(1)     <= A(1) or b(1);
add_op(1)    <= (xor_op(1)) xor carry_aux(0);
carry_aux(1) <= and_op(1) or (xor_op(1) and carry_aux(0));

B(2)         <= ula_operation(1) xor ula_B(2);
and_op(2)    <= A(2) and B(2);
xor_op(2)    <= A(2) xor B(2);
or_op(2)     <= A(2) or b(2);
add_op(2)    <= (xor_op(2)) xor carry_aux(1);
carry_aux(2) <= and_op(2) or (xor_op(2) and carry_aux(1));

B(3)         <= ula_operation(1) xor ula_B(3);
and_op(3)    <= A(3) and B(3);
xor_op(3)    <= A(3) xor B(3);
or_op(3)     <= A(3) or b(3);
add_op(3)    <= (xor_op(3)) xor carry_aux(2);
carry_aux(3) <= and_op(3) or (xor_op(3) and carry_aux(2));

B(4)         <= ula_operation(1) xor ula_B(4);
and_op(4)    <= A(4) and B(4);
xor_op(4)    <= A(4) xor B(4);
or_op(4)     <= A(4) or b(4);
add_op(4)    <= (xor_op(4)) xor carry_aux(3);
carry_aux(4) <= and_op(4) or (xor_op(4) and carry_aux(3));

B(5)         <= ula_operation(1) xor ula_B(5);
and_op(5)    <= A(5) and B(5);
xor_op(5)    <= A(5) xor B(5);
or_op(5)     <= A(5) or b(5);
add_op(5)    <= (xor_op(5)) xor carry_aux(4);
carry_aux(5) <= and_op(5) or (xor_op(5) and carry_aux(4));

B(6)         <= ula_operation(1) xor ula_B(6);
and_op(6)    <= A(6) and B(6);
xor_op(6)    <= A(6) xor B(6);
or_op(6)     <= A(6) or b(6);
add_op(6)    <= (xor_op(6)) xor carry_aux(5);
carry_aux(6) <= and_op(6) or (xor_op(6) and carry_aux(5));

B(7)         <= ula_operation(1) xor ula_B(7);
and_op(7)    <= A(7) and B(7);
xor_op(7)    <= A(7) xor B(7);
or_op(7)     <= A(7) or b(7);
add_op(7)    <= (xor_op(7)) xor carry_aux(6);
carry_aux(7) <= and_op(7) or (xor_op(7) and carry_aux(6));

-- Circuito de deslocamento a direita/esquerda
rot_op(0) <= (A(7) and ula_operation(6)) or (A(1) and ula_operation(5));
rot_op(1) <= (A(0) and ula_operation(6)) or (A(2) and ula_operation(5));
rot_op(2) <= (A(1) and ula_operation(6)) or (A(3) and ula_operation(5));
rot_op(3) <= (A(2) and ula_operation(6)) or (A(4) and ula_operation(5));
rot_op(4) <= (A(3) and ula_operation(6)) or (A(5) and ula_operation(5));
rot_op(5) <= (A(4) and ula_operation(6)) or (A(6) and ula_operation(5));
rot_op(6) <= (A(5) and ula_operation(6)) or (A(7) and ula_operation(5));
rot_op(7) <= (A(6) and ula_operation(6)) or (A(0) and ula_operation(5));
```

```
-- Multiplexador de saída
  with ula_operation select
    ula_result <=  add_op when "0000001",   -- seleciona Soma
                   add_op when "0000010",   -- seleciona Subtração
                   and_op when "0000100",   -- seleciona AND
                   or_op  when "0001000",   -- seleciona OR
                   xor_op when "0010000",   -- seleciona XOR
                   rot_op when "0100000",   -- seleciona Rotação à direita
                   rot_op when "1000000",   -- seleciona Rotação à esquerda
                   A      when others;      -- entrada 1 da ula

A <= ula_A;
carry_out <= carry_aux(7);

end A_ula8;
```

Iram.mif

```
-- Iram.mif
WIDTH = 8;
DEPTH = 256;
ADDRESS_RADIX = HEX;
DATA_RADIX = HEX;
CONTENT BEGIN
            0           :           00;
            1           :           00;
            2           :           00;
            3           :           00;
            4           :           00;
            5           :           00;
            6           :           00;
            7           :           00;
            8           :           00;
            9           :           00;
            a           :           00;
            b           :           00;
            c           :           00;
            d           :           00;
            e           :           00;
            f           :           00;
            10          :           00;
            11          :           00;
            12          :           00;
            13          :           00;
            14          :           00;
            15          :           00;
            16          :           00;
            17          :           00;
            18          :           00;
```

```
19      :       00;
1a      :       00;
1b      :       00;
1c      :       00;
1d      :       00;
1e      :       00;
1f      :       00;
20      :       00;
21      :       00;
22      :       00;
23      :       00;
24      :       00;
25      :       00;
26      :       00;
27      :       00;
28      :       00;
29      :       00;
2a      :       00;
2b      :       00;
2c      :       00;
2d      :       00;
2e      :       00;
2f      :       00;
30      :       44;
31      :       41;
32      :       54;
33      :       41;
34      :       3f;
35      :       00;
36      :       00;
37      :       00;
38      :       00;
39      :       00;
3a      :       00;
3b      :       00;
3c      :       00;
3d      :       00;
3e      :       00;
3f      :       00;
40      :       00;
41      :       00;
42      :       00;
43      :       00;
44      :       00;
45      :       00;
46      :       00;
47      :       00;
48      :       00;
49      :       00;
4a      :       00;
4b      :       00;
4c      :       00;
4d      :       00;
4e      :       00;
4f      :       00;
```

```
50      :       00;
51      :       00;
52      :       00;
53      :       00;
54      :       00;
55      :       00;
56      :       00;
57      :       00;
58      :       00;
59      :       00;
5a      :       00;
5b      :       00;
5c      :       00;
5d      :       00;
5e      :       00;
5f      :       00;
60      :       00;
61      :       00;
62      :       00;
63      :       00;
64      :       00;
65      :       00;
66      :       00;
67      :       00;
68      :       00;
69      :       00;
6a      :       00;
6b      :       00;
6c      :       00;
6d      :       00;
6e      :       00;
6f      :       00;
70      :       00;
71      :       00;
72      :       00;
73      :       00;
74      :       00;
75      :       00;
76      :       00;
77      :       00;
78      :       00;
79      :       00;
7a      :       00;
7b      :       00;
7c      :       00;
7d      :       00;
7e      :       00;
7f      :       00;
80      :       00;
81      :       00;
82      :       00;
83      :       00;
84      :       00;
85      :       00;
86      :       00;
```

```
87      :       00;
88      :       00;
89      :       00;
8a      :       00;
8b      :       00;
8c      :       00;
8d      :       00;
8e      :       00;
8f      :       00;
90      :       00;
91      :       00;
92      :       00;
93      :       00;
94      :       00;
95      :       00;
96      :       00;
97      :       00;
98      :       00;
99      :       00;
9a      :       00;
9b      :       00;
9c      :       00;
9d      :       00;
9e      :       00;
9f      :       00;
a0      :       00;
a1      :       00;
a2      :       00;
a3      :       00;
a4      :       00;
a5      :       00;
a6      :       00;
a7      :       00;
a8      :       00;
a9      :       00;
aa      :       00;
ab      :       00;
ac      :       00;
ad      :       00;
ae      :       00;
af      :       00;
b0      :       00;
b1      :       00;
b2      :       00;
b3      :       00;
b4      :       00;
b5      :       00;
b6      :       00;
b7      :       00;
b8      :       00;
b9      :       00;
ba      :       00;
bb      :       00;
bc      :       00;
bd      :       00;
```

```
be      :       00;
bf      :       00;
c0      :       00;
c1      :       00;
c2      :       00;
c3      :       00;
c4      :       00;
c5      :       00;
c6      :       00;
c7      :       00;
c8      :       00;
c9      :       00;
ca      :       00;
cb      :       00;
cc      :       00;
cd      :       00;
ce      :       00;
cf      :       00;
d0      :       00;
d1      :       00;
d2      :       00;
d3      :       00;
d4      :       00;
d5      :       00;
d6      :       00;
d7      :       00;
d8      :       00;
d9      :       00;
da      :       00;
db      :       00;
dc      :       00;
dd      :       00;
de      :       00;
df      :       00;
e0      :       00;
e1      :       00;
e2      :       00;
e3      :       00;
e4      :       00;
e5      :       00;
e6      :       00;
e7      :       00;
e8      :       00;
e9      :       00;
ea      :       00;
eb      :       00;
ec      :       00;
ed      :       00;
ee      :       00;
ef      :       00;
f0      :       00;
f1      :       00;
f2      :       00;
f3      :       00;
f4      :       00;
```

```
        f5      :       00;
        f6      :       00;
        f7      :       00;
        f8      :       00;
        f9      :       00;
        fa      :       00;
        fb      :       00;
        fc      :       00;
        fd      :       00;
        fe      :       00;
        ff      :       00;
END;


-- MAX+plus II - generated Memory Initialization File

-- Copyright (C) 1988-1999 Altera Corporation
-- Any megafunction design, and related net list (encrypted or decrypted),
-- support information, device programming or simulation file, and any other
-- associated documentation or information provided by Altera or a partner
-- under Altera's Megafunction Partnership Program may be used only to
-- program PLD devices (but not masked PLD devices) from Altera.  Any other
-- use of such megafunction design, net list, support information, device
-- programming or simulation file, or any other related documentation or
-- information is prohibited for any other purpose, including, but not
-- limited to modification, reverse engineering, de-compiling, or use with
-- any other silicon devices, unless such use is explicitly licensed under
-- a separate agreement with Altera or a megafunction partner.  Title to
-- the intellectual property, including patents, copyrights, trademarks,
-- trade secrets, or maskworks, embodied in any such megafunction design,
-- net list, support information, device programming or simulation file, or
-- any other related documentation or information provided by Altera or a
-- megafunction partner, remains with Altera, the megafunction partner, or
-- their respective licensors.  No other licenses, including any licenses
-- needed under any third party's intellectual property, are provided herein.
```

EPROM2.mif

```
-- EPROM2.mif
WIDTH = 9;
DEPTH = 256;
ADDRESS_RADIX = HEX;
DATA_RADIX = HEX;
CONTENT BEGIN
        0       :       1E5;
        1       :       090;
        2       :       0F5;
        3       :       00A;
        4       :       1E5;
        5       :       090;
        6       :       0F5;
```

```
7   :   10B;
8   :   0E4;
9   :   0F5;
a   :   090;
b   :   0F5;
c   :   180;
d   :   1E5;
e   :   10B;
f   :   125;
10  :   00A;
11  :   0F5;
12  :   00C;
13  :   1E5;
14  :   00A;
15  :   014;
16  :   060;
17  :   006;
18  :   0F5;
19  :   00A;
1a  :   1E5;
1b  :   00C;
1c  :   180;
1d  :   1F1;
1e  :   1E5;
1f  :   00C;
20  :   0F5;
21  :   0A0;
22  :   000;
23  :   000;
24  :   000;
25  :   000;
26  :   000;
27  :   000;
28  :   000;
29  :   000;
2a  :   000;
2b  :   000;
2c  :   000;
2d  :   000;
2e  :   000;
2f  :   000;
30  :   000;
31  :   000;
32  :   000;
33  :   000;
34  :   000;
35  :   000;
36  :   000;
37  :   000;
38  :   000;
39  :   000;
3a  :   000;
3b  :   000;
3c  :   000;
3d  :   000;
```

```
3e      :       000;
3f      :       000;
40      :       000;
41      :       000;
42      :       000;
43      :       000;
44      :       000;
45      :       000;
46      :       000;
47      :       000;
48      :       000;
49      :       000;
4a      :       000;
4b      :       000;
4c      :       000;
4d      :       000;
4e      :       000;
4f      :       000;
50      :       000;
51      :       000;
52      :       000;
53      :       000;
54      :       000;
55      :       000;
56      :       000;
57      :       000;
58      :       000;
59      :       000;
5a      :       000;
5b      :       000;
5c      :       000;
5d      :       000;
5e      :       000;
5f      :       000;
60      :       000;
61      :       000;
62      :       000;
63      :       000;
64      :       000;
65      :       000;
66      :       000;
67      :       000;
68      :       000;
69      :       000;
6a      :       000;
6b      :       000;
6c      :       000;
6d      :       000;
6e      :       000;
6f      :       000;
70      :       000;
71      :       000;
72      :       000;
73      :       000;
74      :       000;
```

```
75      :       000;
76      :       000;
77      :       000;
78      :       000;
79      :       000;
7a      :       000;
7b      :       000;
7c      :       000;
7d      :       000;
7e      :       000;
7f      :       000;
80      :       000;
81      :       000;
82      :       000;
83      :       000;
84      :       000;
85      :       000;
86      :       000;
87      :       000;
88      :       000;
89      :       000;
8a      :       000;
8b      :       000;
8c      :       000;
8d      :       000;
8e      :       000;
8f      :       000;
90      :       000;
91      :       000;
92      :       000;
93      :       000;
94      :       000;
95      :       000;
96      :       000;
97      :       000;
98      :       000;
99      :       000;
9a      :       000;
9b      :       000;
9c      :       000;
9d      :       000;
9e      :       000;
9f      :       000;
a0      :       000;
a1      :       000;
a2      :       000;
a3      :       000;
a4      :       000;
a5      :       000;
a6      :       000;
a7      :       000;
a8      :       000;
a9      :       000;
aa      :       000;
ab      :       000;
```

```
ac      :       000;
ad      :       000;
ae      :       000;
af      :       000;
b0      :       000;
b1      :       000;
b2      :       000;
b3      :       000;
b4      :       000;
b5      :       000;
b6      :       000;
b7      :       000;
b8      :       000;
b9      :       000;
ba      :       000;
bb      :       000;
bc      :       000;
bd      :       000;
be      :       000;
bf      :       000;
c0      :       000;
c1      :       000;
c2      :       000;
c3      :       000;
c4      :       000;
c5      :       000;
c6      :       000;
c7      :       000;
c8      :       000;
c9      :       000;
ca      :       000;
cb      :       000;
cc      :       000;
cd      :       000;
ce      :       000;
cf      :       000;
d0      :       000;
d1      :       000;
d2      :       000;
d3      :       000;
d4      :       000;
d5      :       000;
d6      :       000;
d7      :       000;
d8      :       000;
d9      :       000;
da      :       000;
db      :       000;
dc      :       000;
dd      :       000;
de      :       000;
df      :       000;
e0      :       000;
e1      :       000;
e2      :       000;
```

```
        e3      :       000;
        e4      :       000;
        e5      :       000;
        e6      :       000;
        e7      :       000;
        e8      :       000;
        e9      :       000;
        ea      :       000;
        eb      :       000;
        ec      :       000;
        ed      :       000;
        ee      :       000;
        ef      :       000;
        f0      :       000;
        f1      :       000;
        f2      :       000;
        f3      :       000;
        f4      :       000;
        f5      :       000;
        f6      :       000;
        f7      :       000;
        f8      :       000;
        f9      :       000;
        fa      :       000;
        fb      :       000;
        fc      :       000;
        fd      :       000;
        fe      :       000;
        ff      :       000;
END;
```

Decod20_2.mif

```
-- Decod20_2.mif
WIDTH = 21;
DEPTH = 256;
ADDRESS_RADIX = HEX;
DATA_RADIX = HEX;
CONTENT BEGIN
        0       :       000000;
        1       :       000000;
        2       :       000000;
        3       :       010020;
        4       :       010001;
        5       :       10c881;
        6       :       00a081;
        7       :       00a081;
        8       :       108081;
        9       :       108081;
        a       :       108081;
```

```
b       :       108081;
c       :       108081;
d       :       108081;
e       :       108081;
f       :       108081;
10      :       000000;
11      :       000000;
12      :       000000;
13      :       000000;
14      :       010601;
15      :       10ce81;
16      :       00a681;
17      :       00a681;
18      :       108681;
19      :       108681;
1a      :       108681;
1b      :       108681;
1c      :       108681;
1d      :       108681;
1e      :       108681;
1f      :       108681;
20      :       000000;
21      :       000000;
22      :       000000;
23      :       100040;
24      :       100c01;
25      :       114a01;
26      :       012201;
27      :       012201;
28      :       110201;
29      :       110201;
2a      :       110201;
2b      :       110201;
2c      :       110201;
2d      :       11201;
2e      :       110201;
2f      :       110201;
30      :       000000;
31      :       000000;
32      :       000000;
33      :       000000;
34      :       010c01;
35      :       114a01;
36      :       012201;
37      :       012201;
38      :       110201;
39      :       110201;
3a      :       110201;
3b      :       110201;
3c      :       110201;
3d      :       110201;
3e      :       110201;
3f      :       110201;
40      :       020d81;
41      :       000000;
```

```
42      :       10ca08;
43      :       000000;
44      :       010c08;
45      :       114a08;
46      :       012208;
47      :       012208;
48      :       110208;
49      :       110208;
4a      :       110208;
4b      :       110208;
4c      :       110208;
4d      :       110208;
4e      :       110208;
4f      :       110208;
50      :       000000;
51      :       000000;
52      :       10ca04;
53      :       000000;
54      :       010c04;
55      :       114a04;
56      :       012204;
57      :       012204;
58      :       110204;
59      :       110204;
5a      :       110204;
5b      :       110204;
5c      :       110204;
5d      :       110204;
5e      :       110204;
5f      :       110204;
60      :       020d81;
61      :       000000;
62      :       10ca10;
63      :       000000;
64      :       010c10;
65      :       114a10;
66      :       012210;
67      :       012210;
68      :       110210;
69      :       110210;
6a      :       110210;
6b      :       110210;
6c      :       110210;
6d      :       110210;
6e      :       110210;
6f      :       110210;
70      :       000000;
71      :       000000;
72      :       000000;
73      :       000000;
74      :       110900;
75      :       000000;
76      :       00a900;
77      :       00a900;
78      :       108900;
```

```
79      :       108900;
7a      :       108900;
7b      :       108900;
7c      :       108900;
7d      :       108900;
7e      :       108900;
7f      :       108900;
80      :       020d81;
81      :       000000;
82      :       000000;
83      :       000000;
84      :       000000;
85      :       000000;
86      :       000000;
87      :       000000;
88      :       000000;
89      :       000000;
8a      :       000000;
8b      :       000000;
8c      :       000000;
8d      :       000000;
8e      :       000000;
8f      :       000000;
90      :       000000;
91      :       000000;
92      :       000000;
93      :       000000;
94      :       010c02;
95      :       114a02;
96      :       012202;
97      :       012202;
98      :       110202;
99      :       110202;
9a      :       110202;
9b      :       110202;
9c      :       110202;
9d      :       110202;
9e      :       110202;
9f      :       110202;
a0      :       000000;
a1      :       000000;
a2      :       000000;
a3      :       000000;
a4      :       000000;
a5      :       000000;
a6      :       000000;
a7      :       000000;
a8      :       000000;
a9      :       000000;
aa      :       000000;
ab      :       000000;
ac      :       000000;
ad      :       000000;
ae      :       000000;
af      :       000000;
```

```
b0      :       000000;
b1      :       000000;
b2      :       000000;
b3      :       000000;
b4      :       000000;
b5      :       000000;
b6      :       000000;
b7      :       000000;
b8      :       000000;
b9      :       000000;
ba      :       000000;
bb      :       000000;
bc      :       000000;
bd      :       000000;
be      :       000000;
bf      :       000000;
c0      :       000000;
c1      :       000000;
c2      :       000000;
c3      :       000000;
c4      :       000000;
c5      :       000000;
c6      :       000000;
c7      :       000000;
c8      :       000000;
c9      :       000000;
ca      :       000000;
cb      :       000000;
cc      :       000000;
cd      :       000000;
ce      :       000000;
cf      :       000000;
d0      :       000000;
d1      :       000000;
d2      :       000000;
d3      :       000000;
d4      :       000000;
d5      :       000000;
d6      :       000000;
d7      :       000000;
d8      :       000000;
d9      :       000000;
da      :       000000;
db      :       000000;
dc      :       000000;
dd      :       000000;
de      :       000000;
df      :       000000;
e0      :       000000;
e1      :       000000;
e2      :       046080;
e3      :       04c100;
e4      :       010004;
e5      :       014880;
e6      :       112080;
```

```
            e7          :           112080;
            e8          :           010080;
            e9          :           010080;
            ea          :           010080;
            eb          :           010080;
            ec          :           010080;
            ed          :           010080;
            ee          :           010080;
            ef          :           010080;
            f0          :           000000;
            f1          :           000000;
            f2          :           086080;
            f3          :           086080;
            f4          :           010610;
            f5          :           10c800;
            f6          :           00a000;
            f7          :           00a000;
            f8          :           108000;
            f9          :           108000;
            fa          :           108000;
            fb          :           108000;
            fc          :           108000;
            fd          :           108000;
            fe          :           108000;
            ff          :           108000;
END;
```

IR.vhd

```
--------------------------------------------------------------------------------
-- Dissertação de Mestrado 2001/02                                             --
-- Registrador de instruções para inserir no TAP                                                    --
-- Por Eduardo Santos Back em novembro de 2000
                                                --
-- Registrador de instrução do controlador TAP
--

entity ir is
  port(
       CLK       : in bit;                -- Sinal de sincronismo externo
       SET       : in bit;                -- Sinal de set nos flip-flops
       Entrada   : in bit;                 -- Porta de entrada de dados no registrador
       Saida     : out bit;                -- Porta de saída de dados do registrador
                   habilita  : in bit;                                          -- Entrada para habilitar a carga
do registrador
       carrega   : in bit;                                                  -- Entrada para deixar ff's em "101"
       seleciona : in bit;                                                  -- Entrada para selecionar ff's entre TDI
e TDO
       registrador : out bit_vector (2 downto 0) -- Mapeamento do registrador RI 3 bits
      );
```

```
 end ir;

architecture A_ir of ir is

Signal q1                       : bit;
Signal q2       : bit;
Signal q3       : bit;
Signal q11                      : bit;
Signal q12                      : bit;
Signal q13                      : bit;

begin
-- flip-flops funcionando como registrador de deslocamento
process (CLK, SET)
   begin
      if SET = '1' then q1 <='1'; -- Inicializa ff's
                                                q2 <='1';
                                          q3 <='1';
            elsif (CLK'EVENT and CLK='1') then
                        if seleciona = '1' then -- Seleciona ff's entre TDI e TDO
                        q1 <= Entrada;
                              q2 <= q1;
                              q3 <= q2;
                        elsif carrega ='1' then -- Faz a carga de "101" nos ff's,
-- condição inicial de IR de acordo com a norma ao entrar no estado Shift_IR
                              q1 <='1';
                           q2 <='0';
                           q3 <='1';
                        end if;
                  end if;
end process;
process (clk,set)
         begin
                  if SET = '1' then q11 <='1';-- Inicializa ff's
                              q12 <='1';
                                    q13 <='1';
                  elsif (CLK'EVENT and CLK='0') then
                        if habilita = '1' then -- Carrega IR a partir dos ff's
                              q11 <= q1;
                              q12 <= q2;
                              q13 <= q3;
                        end if;
                  end if;
end process;
saida <= q3;
registrador(2) <= q11;
registrador(1) <= q12;
registrador(0) <= q13;
end A_ir;
```

TAP.vhd

```
-- Dissertação de Mestrado 2001/02                                          --
-- TAP para inserir no Microcontrolador 8051 fonte compatível               --
-- Versão para System-on-Chip 8051                                          --
-- Por Eduardo Santos Back em dezembro de 2000, revisado em abril de 2001   --

LIBRARY ieee;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_1164.all;
entity TAP is
  port(
      TCK      : in std_logic;              -- Sinal de sincronismo externo
      TRST     : in std_logic;              -- Sinal de reset assíncrono (opcional)
      TDI      : in std_logic;              -- Porta de entrada de dados de teste
      TDO      : out std_logic;             -- Porta de saída de dados de teste
      TMS      : in std_logic;              -- Porta de entrada de seleção do teste
      --Sinais de saída para as células BS
      Shift_BS  : out std_logic;                                -- Saída para Shift_BS

      Mode_BS     : out std_logic;                              -- Saída para Mode
      Clock_BS  : out std_logic;                                -- Saída para Clock das células BS
                  carrega_BS        : out std_logic;                    -- Habilita o primeiro flip-flop
                  armazena_BS       : out std_logic;                    -- Habilita o segundo flip-flop
                  reset_test_cells  : out std_logic;   -- Coloca as saídas dos flip-flops em zero
                  TDI_BS            : out std_logic;                    -- TDI para células BS (OUT)
                  TDO_BS                  : in std_logic;                     -- TDO das células BS
(IN)
      --Sinais de saída para as células da cadeia de teste interna
      Shift_int  : out std_logic;                               -- Saída para Shift_int

      Mode_int      : out std_logic;                    -- Saída para Mode
          carrega_int         : out std_logic;                    -- Habilita o primeiro flip-flop
                  armazena_int      : out std_logic;              -- Habilita o segundo flip-flop
                  TDI_int           : out std_logic;                    -- TDI para células internas
(OUT)
                  TDO_int                 : in std_logic;                     -- TDO das células
internas (IN)
      --Sinais de saída para a cadeia de teste BIST das memórias
      Shift_BIST  : out std_logic;                    -- Saída para Shift_BIST

      Mode_BIST  : out std_logic;                     -- Saída para Mode
                  carrega_BIST      : out std_logic;              -- Habilita o primeiro flip-flop
                  armazena_BIST     : out std_logic;              -- Habilita o segundo flip-flop
                  TDI_BIST          : out std_logic;                    -- TDI para células BIST
(OUT)
                  TDO_BIST                : in std_logic            -- TDO das células BIST (IN)
     );

end TAP;

architecture A_TAP of TAP is

-------------------------------- Criando os estados --------------------------------
type estados is
(Test_Logic_Reset,Run_Test_Idle,Select_DR_Scan,Capture_DR,Shift_DR,Exit1_DR,Pause_DR,Exit2
_DR,Update_DR,Select_IR_Scan,Capture_IR, Shift_IR, Exit1_IR,Pause_IR,Exit2_IR,Update_IR);
```

```
signal estado : estados;

---------------------------------- Os registradores ----------------------------------

------------Registrador de instrução----------------
component ir
        port (
                clk                             : in std_logic;         -- Sinal de
sincronismo externo
                set                             : in std_logic;         -- Sinal de set
nos flip-flops
                entrada                 : in std_logic;         -- Porta de entrada de
dados no registrador
                saida                   : out std_logic;    -- Porta de saída de dados do
registrador
                habilita        : in std_logic;         -- Entrada para habilitar a carga
do registrador
                carrega                 : in std_logic;         -- Entrada para deixar
ff's em "101"
                seleciona               : in std_logic;         -- Entrada para
selecionar ff's entre TDI e TDO
                registrador             : out std_logic_vector(2 downto 0)-- Mapeamento
do registrador RI 3 bits
                );
end component;

------------Registrador Bypass----------------------
signal BPR: std_logic;
signal habilita_bp: std_logic;

------------------------ Sinal para mux de entrada e de saída --------------
signal seletor: std_logic_vector(2 downto 0);
-- Sinais do componente IR
signal clk: std_logic;
signal set: std_logic;
signal entrada: std_logic;
signal habilita_ir: std_logic;
signal saida: std_logic;
signal carrega_ir: std_logic;
signal seleciona_ir: std_logic;
signal registrador_ir: std_logic_vector (2 downto 0);

begin
reg_instru: ir
        port map(
                tck,
                set,
                entrada,
                saida,
                habilita_ir,
                carrega_ir,
                seleciona_ir,
                registrador_ir
                );
---------------------------------- Máquina de estados ----------------------------------
```

```
process (TCK, TRST)
   begin
      if TRST = '1' then estado <= Test_Logic_Reset;              -- reset assíncrono
                                                          --- Inicialização dos sinais de controle
                        reset_test_cells <= '1';
                        carrega_bs <='0';
                        carrega_int <='0';
                        armazena_bs <='0';
                                    armazena_BIST <= '0';
                        armazena_int <= '1'; -- Para deixar um ciclo de tck
                        carrega_ir <= '0';
                        habilita_ir <= '0';
                                    seleciona_ir <= '0';
                                    set         <='1';
                                    carrega_BIST <= '0';
                        seletor <= "111"; -- Seleciona nada
                        habilita_bp <= '0';
                        Shift_BS <= '0';
                        Shift_int <= '0';
                        Shift_BIST <= '0';
                  elsif (TCK'EVENT and TCK='1') then
                        reset_test_cells <='0';
                        case estado is
                              when Test_Logic_Reset =>
                                                          --- Inicialização dos sinais de controle
                        carrega_bs <='0';
                        carrega_int <='0';
                        armazena_bs <='0';
                                    armazena_BIST <= '0';
                        armazena_int <= '1'; -- para deixar os registradores internos funcionando
sempre
                        carrega_ir <= '0';
                        habilita_ir <= '0';
                                    seleciona_ir <= '0';
                                    carrega_BIST <= '0';
                        seletor <= "111"; -- Seleciona nada
                        habilita_bp <= '0';
                        Shift_BS <= '0';
                        Shift_int <= '0';
                        Shift_BIST <= '0';
                                    set         <=         '1';
                                    if TMS='1' then estado <= Test_Logic_Reset;
                                    else
                                          estado <= Run_Test_Idle;
                                          set <='0';
                                    end if;
                              when Run_Test_Idle =>
                                    if TMS='0'then estado <= Run_Test_Idle;
                                    else estado <= Select_DR_Scan;
                                    end if;
                              when Select_DR_Scan =>
                                    if TMS='0'then
                                          estado <= Capture_DR;
                                          if registrador_ir < "010" then
```

```
                                                carrega_BS <='1'; -- Instruções que usam
o caminho BS
                                        end if;
                                        if registrador_ir = "011" then
                                                carrega_int <='1'; -- Instrução ScanInt
                                                armazena_int <='0';
                                        end if;
                                        if registrador_ir = "010" then

                                                carrega_BIST <='1'; -- Instrução
RunBIST

                                        end if;
                                else estado <= Select_IR_Scan;

                                end if;
                        when Capture_DR =>

                                if TMS='0'then estado <= Shift_DR;
                                        case registrador_ir is

                                                when "000"  => -- Extest
                                                        Shift_BS <= '1';

                                                when "001"  => -- Sample/Preload
                                                        Shift_BS <= '1';

                                                when "010"  => -- RunBIST
                                                        Shift_BIST <= '1';

                                                when "011"  => -- ScanInt
                                                        Shift_int <= '1';

                                                when others => -- Bypass => > "011"
                                        end case;
                                else estado <= Exit1_DR;
                                end if;
--- Tabela de Instruções:    000 -> Extest

---                                                     001 -> Sample/Preload

---                                                     010 -> RunBIST

---                                                     011 -> ScanInt

---                                                     100 -> Bypass

---                                                     101 -> Bypass

---                                                     110 -> Bypass

---                                                     111 -> Bypass

------------------Verificação da instrução-------------------------
                                case registrador_ir is
```

```
                                        when "000"  => -- Extest
                                                Carrega_BS <= '1';
                                                Seletor <="000";
                                        when "001"  => -- Sample/Preload
                                                Carrega_BS <= '1';
                                                Seletor <="000";
                                        when "010"  => -- RunBIST
                                                carrega_BIST <= '1';
                                                seletor <= "100";
                                        when "011"  => -- ScanInt
                                                carrega_int <= '1';
                                                armazena_int <='0';
                                                seletor <= "011";
                                        when others => -- Bypass => > "011"
                                                habilita_bp <='1'; -- Habilita o
funcionamento do registrador bypass
                                                seletor <= "010"; -- Seleciona o
registrador bypass
                                end case;
                        when Shift_DR =>
                                if TMS='0'then estado <= Shift_DR;
                                else estado <= Exit1_DR;
                                        habilita_bp <='0'; -- Desabilita BPR entre TDI e
TDO
                                        seletor <= "111";
                                        Shift_BS <= '0';
                                        Shift_int <= '0';
                                        Shift_BIST <= '0';
                                        carrega_bs <= '0'; -- Desativa o clock das células
BS
                                        carrega_bist <= '0';
                                        carrega_int <= '0';
                                end if;
--                              Capturou em extest a entrada da célula BS
--                              TDI e TDO ficam entre as células BS
                        when Exit1_DR =>
                                if TMS='0'then estado <= Pause_DR;
                                else estado <= Update_DR;
                                        if registrador_ir < "010"  then
                                                armazena_BS <='1'; -- Instruções
diferentes de RunBISTn
                                        end if;
                                        if registrador_ir ="010" then
                                                armazena_BIST <='1';
                                        end if;
                                        if registrador_ir ="011" then
                                                armazena_int <='1';-- Verificar 1 ciclo
apenas de TCK
                                        end if;
                                end if;
                        when Pause_DR =>
                                if TMS='0'then estado <= Pause_DR;
                                else estado <= Exit2_DR;
                                end if;
                        when Exit2_DR =>
```

```
                                        if TMS='0'then estado <= Shift_DR;
                                        else estado <= Update_DR;
                                                if registrador_ir < "010"  then
                                                        armazena_BS <='1'; -- Instruções
diferentes de RunBISTn
                                                end if;
                                                if registrador_ir ="010" then
                                                        armazena_BIST <='1';
                                                end if;
                                                if registrador_ir ="011" then
                                                        armazena_int <='1';--??
                                                end if;
                                        end if;
                                when Update_DR =>
                                        armazena_BS <= '0'; -- Desabilita o TCK do segundo flip-
flop das células BS
                                        armazena_BIST <= '0'; -- Desabilita o TCK do segundo
flip-flop das células BIST
                                        if registrador_ir="011" then armazena_int <= '0'; --
Verificar para apenas um ciclo de tck
                                        end if;
                                        if TMS='0'then estado <= Run_Test_Idle;
                                        else estado <= Select_DR_Scan;
                                        end if;
----------------------Estados de manipulação de IR----------------------
                                when Select_IR_Scan =>
                                        if TMS='0'then estado <= Capture_IR;
                                                        carrega_ir <= '1'; -- IR recebe o valor 101

                                        else
                                                estado <= Test_Logic_Reset;
                                                set <= '1';
                                        end if;
                                when Capture_IR =>
                                        carrega_ir <= '0'; -- IR recebe 101 e o sinal pode ser
desligado
                                        if TMS='0'then estado <= Shift_IR;
                                                                        seletor <= "001"; --
Seleciona IR em TDO
                                                                        seleciona_ir <= '1'; --
Seleciona IR entre TDI e TDO
                                        else estado <= Exit1_IR;
                                        end if;
                                when Shift_IR =>
                                        if TMS='0'then estado <= Shift_IR;
                                        else estado <= Exit1_IR;
                                 seletor <="111";
                                                 seleciona_ir <= '0'; -- IR pode ser desligado de
TDI e TDO
                                        end if;
                                when Exit1_IR =>
                                        if TMS='0'then estado <= Pause_IR;
                                        else estado <= Update_IR;
                                        end if;
```

```
                                        habilita_ir <='1';  -- Faz a carga de IR a partir dos ff's

                                    when Pause_IR =>
                                        if TMS='0'then estado <= Pause_IR;
                                        else estado <= Exit2_IR;
                                        end if;
                                    when Exit2_IR =>
                                        if TMS='0'then estado <= Shift_IR;
                                        else estado <= Update_IR;
                                        end if;
                                        habilita_ir <='1';
                                    when Update_IR =>
                                        if TMS='0'then estado <= Run_Test_Idle;
                                        else estado <= Select_DR_Scan;
                                        end if;
                                        habilita_ir <='0'; -- IR já foi carregado, o sinal pode ser
desligado
                                        if registrador_ir /= "011" then armazena_int <='1';
                                        else armazena_int <='0';
                                        end if;
                            end case;
                    end if;
            end process;
-- Processo do registrador de bypass
process (TCK, TRST)
            begin
                    if TRST = '1' then
                    bpr <='0';
               elsif (TCK'EVENT and TCK='1' ) then
                    if (habilita_bp = '1' and seletor = "010") then bpr <= TDI;
                    elsif estado = Exit2_DR or estado = Exit1_DR then BPR <= '0';
                            end if;
                    end if;
end process;

-- Seleção MUX de entrada de TDI para as células BS
with seletor select -- Seleção da cadeia BS
            TDI_BS <= TDI when "000",
                            '0'     when others;
with seletor select -- Seleção da cadeia interna
            TDI_int <= TDI when "011",
                            '0' when others;
with seletor select -- Seleção da cadeia BIST
            TDI_BIST <= TDI when "100",
                            '0' when others;
with seletor select
            entrada <= TDI when "001",
                                    '1' when others;
with registrador_ir select
            Mode_BS <= '1' when "000",   -- Extest
                                    '0' when others;
with registrador_ir select
            Mode_BIST <= '1' when "010",   --BIST
                                    '0' when others;
with registrador_ir select
```

```
        Mode_Int <= '1' when "011",   -- ScanInt
                                      '0' when others;
-- Seleção MUX de saída para TDO
with seletor select
        TDO <=  TDO_BS when "000", -- Cadeia BS
                        saida when "001", -- Seleção de IR
                        BPR when "010",   -- Seleção de BPR
                        TDO_int when "011", -- Cadeia interna
                        TDO_BIST when "100", -- Cadeia BIST
                        'Z' when others;
--      Mapeamento do clock das células BS para TCK
    Clock_BS    <= TCK;
end A_TAP;
```

Ender.vhd

```
-------------------------------------------------------------------------------------
-- Dissertação de Mestrado 99/2                                                    --
-- Microcontrolador 8051                                                        --
-- Versão para System-on-Chip                                                   --
-- Circuito do contador de programa (PC)                                           --
-- Por Denis Franco em janeiro de 2000 (19.01.00)                                  --
-------------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_arith.all;

library ieee;
use ieee.std_logic_1164.all;

entity ender is
 port (
       clock     : in bit;                      -- Sinal de sincronismo
       reset     : in bit;                      -- Sinal de inicialização
       hab       : in bit;                      -- Sinal de escrita de endereço
       ender_in  : in unsigned(7 downto 0);          -- Endereço de entrada
       inc       : in bit;                      -- Sinal de incremento do endereço
       ender_out : out std_logic_vector(7 downto 0)  -- Endereço gerado
     );

end ender;

architecture A_ender of ender is

-- Endereço dos elementos da matriz de entrada
  signal end1 : integer range 0 to 255;  -- Valor de saída do contador
  signal end2 : integer range 0 to 255;  -- Valor de entrada do contador

begin

-- Processo de controle do endereço
```

```
 process(clock)
  begin
    if (clock = '0' and not clock'stable) then
      if reset = '1' then
        end1 <= 0;
      elsif hab = '1' then
        end1 <= end2;
      elsif inc ='1' then
        end1 <= end1 + 1;
      end if;
    end if;
 end process;

-- Conversão de dados do tipo inteiro para std_logic_vector
 ender_out <= conv_std_logic_vector (end1,8);
-- Conversão de dados do tipo não-sinalizado para inteiro
 end2      <= conv_integer (ender_in);

end A_ender;
```

POMarch.vhd

```
-- MODULO: POMARCH.VHD

-- CPGCC - CMP117
-- Trabalho Final: Teste implementacao de auto-teste da RAM.
-- Autora: Erika Fernandes Cota & Margrit Reni Krug
-- Cronograma: 03/12/98 - Primeira descricao
--                        98 - Primeira versao funcionando.
-- Objetivo: Esta descricao pretende implementar o algoritmo MARCH para geracao e
-- aplicacao de vetores de teste para Memoria RAM.

-- Descricao Comportamental.

LIBRARY ieee;
        USE ieee.std_logic_arith.all;
        USE ieee.std_logic_1164.all;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

ENTITY  POmarch IS
PORT
(
        clk                     : IN    BIT;      -- relogio
        state           : IN    BIT_VECTOR (4 DOWNTO 0);  -- estado atual do controle
        temp_end        : IN    BIT_VECTOR (7 DOWNTO 0); -- saida do somador que
incrementa endereco de memoria
        dado            : IN    BIT_VECTOR (7 DOWNTO 0); -- palavra de memoria
        pronto          : OUT   BIT;    -- sinal de resultado em vetor
        erro            : OUT   BIT;    -- sinal de existencia de erro na memoria
```

```
        hab_mem_wr      : OUT   BIT;   -- habilita escrita na memoria
        hab_mem_rd      : OUT   BIT; -- habilita leitura da memoria
        soma_sub        : OUT   STD_LOGIC; -- indica se e' soma (1) ou subtracao (0).
        i2                      : OUT    INTEGER RANGE 0 TO 7; -- bit sendo testado na
memoria
        ender2          : OUT   INTEGER range -1 TO 256;  -- contador de palavras de memoria
        vetor           : OUT   BIT_VECTOR (7 DOWNTO 0);   -- vetor de teste ;
        endereco        : OUT   BIT_VECTOR (7 DOWNTO 0)    -- registrador de endereco de
memoria
);
END POmarch;

ARCHITECTURE comportamento OF POmarch IS

 -- declaracao de componentes
 -- Definicao de tipos

CONSTANT max_mem : INTEGER := 255;   -- ultimo endereco de memoria
CONSTANT min_mem : INTEGER := 0;     -- primeiro endereco de memoria

-- Definicao de variaveis internas

SIGNAL  temp_end2       : BIT_VECTOR (7 DOWNTO 0);
SIGNAL          temp_dado : BIT_VECTOR (7 DOWNTO 0);
SIGNAL          ender : INTEGER range -1 TO 256;
SIGNAL          i : INTEGER RANGE 0 TO 7;
SIGNAL          en_soma : BIT;
 -- sinais internos

BEGIN
  PROCESS (state,clk) -- Descricao Parte Operativa
   BEGIN
                IF (clk'EVENT AND clk = '0') THEN
                        CASE state is
                        WHEN B"00000" =>      -- inicializa sinais dado e endereco
                                ender <= 0;
                                vetor <= B"00000000";
                                temp_end2 <= B"00000000";
                                i <= 0;
                                soma_sub <= '1';
                        WHEN B"00001" =>     --- h??? habilitar escrita na memoria
                                hab_mem_wr <= '1';
                        WHEN B"00010" =>      -- incrementa endereco
                                ender <= ender + 1;
                                hab_mem_wr <= '0';
                        WHEN B"00011" =>
                                temp_end2 <= temp_end;
                        WHEN B"00101" =>
                                hab_mem_wr <= '1';
                        WHEN B"00110" =>
                                hab_mem_wr <= '0';
                        WHEN  B"00111" =>
                                ender <= 0;
                                temp_end2 <= B"00000000";
                                endereco <= B"00000000";
```

```
        WHEN B"01000" =>
                                hab_mem_rd <= '1';
                        WHEN  B"01001" =>
                                temp_dado <= dado;
                                hab_mem_rd <= '0';
                                ---- ????? habilitar leitura da memoria ---
                                IF (dado(i) /= '0') THEN
                                        erro <= '1';
                                END IF;
        WHEN B"01010" =>
                                temp_dado(i) <= '1';
                        WHEN  B"01011" =>
                                vetor <= temp_dado;
                        WHEN B"01100" =>
                                hab_mem_wr <= '1';          -- habilita escrita na memoria
                        WHEN B"01101" =>     -- desabilita escrita na memoria
                                hab_mem_wr <= '0';
                                ender <= ender + 1;          -- incrementa contador de endereco de
Mem.
                        WHEN B"01110" =>     -- testa se chegou ao fim da memoria
                                temp_end2 <= temp_end;-- incrementa endereco de memoria
                                endereco <= temp_end;
                        WHEN B"01111" =>
                                i <= i + 1;
                        WHEN B"10010" =>
                                hab_mem_wr <= '1';
                        WHEN B"10011" =>
                                hab_mem_wr <= '0';
                        WHEN B"10100" =>
                                i <= 0;
                                soma_sub <= '0'; -- subtracao com a biblioteca.
                                erro <= '0';
                        WHEN B"10101" =>
                                ender <= max_mem;
                                temp_end2 <= B"11111111";
                                endereco <= B"11111111";
                        WHEN  B"10110" =>
                                hab_mem_rd <= '1';
                        WHEN  B"10111" =>
                                temp_dado <= dado;
                                hab_mem_rd <= '0';
                                ---- ????? habilitar leitura da memoria ---
                                IF (dado(i) /= '1') THEN
                                        erro <= '1';
                                END IF;
        WHEN  B"11000" =>
                                temp_dado(i) <= '0';
                        WHEN  B"11001" =>
                                vetor <= temp_dado;
                        WHEN  B"11010" =>
                                hab_mem_wr <= '1';
                        WHEN  B"11011" =>
                                hab_mem_wr <= '0';
                                ender <= ender - 1;
                        WHEN B"11100" =>
```

```
                        temp_end2 <= temp_end;
                        endereco <= temp_end;
                WHEN  B"11101" =>
                        i <= i + 1;
                WHEN  B"11111" =>
                        pronto <= '1';
                WHEN others =>
                        pronto <= '0';

                END CASE;
        END IF;
    END PROCESS;   -- Fim descricao PO

    --endereco <= temp_end2;
    ender2 <= ender;
    i2 <= i;
END comportamento;
```

Mem_c_bist.vhd

```
-- Modulo mem_c_bist.vhd

--
-- Este modulo e' a memoria RAM acrescida da logica necessaria para seu teste (MARCH).
-- Supoe-se que uma memoria disponibilizada como nucleo de hardware ja venha com estruturas
-- de teste do tipo BIST autonomo.
--

LIBRARY ieee;
        USE ieee.std_logic_1164.all;
        USE ieee.std_logic_arith.all;
LIBRARY lpm;
USE lpm.lpm_components.ALL;

ENTITY mem_c_bist is
port (
                clk                     : in std_logic;
                reset           : in std_logic;
                EN_TESTE        : in std_logic;    -- sinal indicando modo teste. Inserido por Erika
& Margrit em 23/11/98.
                iram_in      : in std_logic_vector(7 downto 0); -- entrada da RAM
    iram_address  : in std_logic_vector(7 downto 0); -- Endere‡o da mem□ria de dados
    iram_wr      : in std_logic;                -- Sinal de escrita na mem□ria de dados
    iram_out     : out std_logic_vector(7 downto 0); -- Sa□da da mem□ria de dados
                erro_RAM        : out std_logic; -- sinal indicando falha na RAM
                fim                     : out std_logic  -- sinal indicando fim do teste da RAM
         );

end mem_c_bist;

architecture arch of mem_c_bist is
```

```
---------------------- Elemento de memória RAM do microcontrolador ----------------------
component lpm_ram_dq
  generic (
          LPM_WIDTH          : POSITIVE;
          LPM_WIDTHAD        : POSITIVE;
          LPM_FILE           : STRING;
          LPM_INDATA         : STRING;
          LPM_ADDRESS_CONTROL : STRING;
          LPM_OUTDATA        : STRING);
   port (
        data     : in std_logic_vector(LPM_WIDTH-1 downto 0);
        address  : in std_logic_vector(LPM_WIDTHAD-1 downto 0);
        we       : in std_logic;
        q        : out std_logic_vector(LPM_WIDTH-1 downto 0)
       );
end component;

-- BIST para teste da RAM. E&m, 14/12/98
COMPONENT march
PORT
(
          clk                       : IN      std_logic;             -- relogio
          reset               : IN      std_logic;
          EN_TESTE       : IN      std_logic;          -- habilita geracao de vetores. Recebe valor do
sinal de teste primario
          dado       : IN     std_logic_VECTOR (7 DOWNTO 0); --
          pronto              : OUT    std_logic;      -- sinal de resultado em vetor
          erro                : OUT    std_logic;          -- sinal de existencia de erro na memoria
          hab_mem_wr     : OUT    std_logic;   -- habilita escrita na memoria
          hab_mem_rd     : OUT    std_logic; -- habilita leitura da memoria
          vetor               : OUT    std_logic_VECTOR (7 DOWNTO 0);   -- vetor de teste ;
          endereco_mem: OUT       std_logic_VECTOR (7 DOWNTO 0)    -- endereco de memoria
);
END COMPONENT;

-- sinais para teste da RAM
SIGNAL          we, hab_mem_wr            : std_logic;   -- habilita escrita na memoria
SIGNAL          hab_mem_rd      : std_logic; -- habilita leitura da memoria
SIGNAL          entrada_ram, dram_in      : std_logic_VECTOR (7 DOWNTO 0);   -- vetor de teste ;
SIGNAL  saida_ram   : std_logic_vector(7 downto 0); -- Sa□da da mem□ria de dados
SIGNAL          addr_ram, endereco_ram: std_logic_VECTOR (7 DOWNTO 0);    -- endereco de
memoria
SIGNAL          pronto_ram        : std_logic;

begin

-- inicializacao sinais de teste

          TPG_RAM: march
                    port map (clk => clk,
                         reset => reset,
                              EN_TESTE     => EN_TESTE,
                              dado   => saida_ram,
                              pronto => fim,
```

```
                        erro     => erro_RAM,
                        hab_mem_wr => hab_mem_wr,
                        hab_mem_rd => hab_mem_rd,
                        vetor => entrada_ram,
                        endereco_mem => endereco_ram
                    );



-- Memória de dados e registradores
  RAM_interna: lpm_ram_dq

    generic map ( 8, 8, "iram.mif", "UNREGISTERED","UNREGISTERED","UNREGISTERED")

    port map (
                            dram_in,
        addr_ram,
        we,
        saida_ram
     );


        RAM_BLOCK: BLOCK
                BEGIN

                        with EN_TESTE select
                                dram_in <=  entrada_ram when '1',
                                                        iram_in when others;

                        with EN_TESTE select
                                addr_ram <= endereco_ram when '1',
                                                iram_address when others;

                        with EN_TESTE select
                                we <= hab_mem_wr when '1',

            iram_wr when others;

                end block;

 iram_out <= saida_ram;


end arch;
```

Decod_c_bist.vhd

```
--------------------------------------------------------------------------------
-- Proposta de Tese - Estudo de caso                                          --
-- Memorias ROM com teste de paridade embutido.
-- Por Erika Cota, novembro de 2000.
--------------------------------------------------------------------------------

library lpm;
use lpm.lpm_components.all;

entity decod_c_bist is
  port(
       endereco  : in bit_vector(7 downto 0);
       decod_out   : out bit_vector(19 downto 0);
                     erro_ROM  : out bit  -- sinal indicando falha na RAM
     );
end decod_c_bist;

architecture bist of decod_c_bist is

--------------------------------------------------------------------------------
------------------------------ Componentes externos à descrição ----------------------------
--------------------------------------------------------------------------------

CONSTANT word_width : INTEGER := 21;

----------------------- Elemento de memória ROM do microcontrolador ----------------------
component lpm_rom
  generic (
          LPM_WIDTH          : POSITIVE;
          LPM_WIDTHAD        : POSITIVE;
          LPM_FILE           : STRING;
          LPM_ADDRESS_CONTROL : STRING;
                        LPM_OUTDATA        : STRING);
  port (
        address  : in bit_vector(LPM_WIDTHAD-1 downto 0);
        q        : out bit_vector(LPM_WIDTH-1 downto 0)
       );
end component;

component check_parity_decod
  port(
       dado       : in bit_vector(word_width-1 downto 0);   -- sinal a ser testado
                    erro       : out bit                    -- sinal indicando falha de paridade no dado
     );
end component;

signal dado    : bit_vector(word_width-1 downto 0);

BEGIN

ROM_interna: lpm_rom
  generic map ( word_width, 8, "decod20_2.mif", "UNREGISTERED","UNREGISTERED")
```

```
  port map (
          endereco,
          dado
         );

verifica: check_parity_decod
  port map (
          dado,
                        erro_ROM
  );

decod_out <= dado(word_width-2 downto 0);
end bist;
```

FSMMarch.vhd

```
-- MODULO: FSMMARCH.VHD

-- CPGCC - CMP117
-- Trabalho Final: Teste implementacao de auto-teste da RAM.
-- Autora: Erika Fernandes Cota & Margrit Reni Krug
-- Cronograma: 03/12/98 - Primeira descricao
--                              98 - Primeira versao funcionando.
-- Objetivo: Esta descricao pretende implementar a maquina de estados do algoritmo
--          MARCH para geracao e aplicacao de vetores de teste para Memoria RAM.

-- Descricao Comportamental.

LIBRARY ieee;
          USE ieee.std_logic_arith.all;

ENTITY  FSMmarch IS

PORT
(
          clk                       : IN      BIT;       -- relogio
          reset             : IN      BIT;
          EN_TESTE      : IN      BIT;     -- habilita geracao de vetores. Recebe valor do sinal de
teste primario
          ender             : IN      INTEGER RANGE -1 TO 256;
          i                         : IN      INTEGER RANGE 0 TO 7;
          state             : OUT     BIT_VECTOR (4 DOWNTO 0)  -- estado atual para PO
);
END FSMmarch;

ARCHITECTURE comportamento OF FSMmarch IS

-- Definicao de tipos

TYPE STATE_TYPE IS (estado_0, estado_1, estado_2, estado_3, estado_4, estado_5, estado_6,
estado_7,
```

```
                                        estado_8, estado_9, estado_10, estado_11, estado_12,
estado_13, estado_14, estado_15,
                                        estado_16, estado_17, estado_18, estado_19, estado_20,
estado_21, estado_22, estado_23,
                                        estado_24, estado_25, estado_26, estado_27, estado_28,
estado_29,
                                        estado_30, estado_31 );

-- Definicao constantes
CONSTANT  fim_memoria: INTEGER := 256;
CONSTANT  fim_palavra: INTEGER := 7;

-- Definicao de variaveis internas

SIGNAL          prox_state          : STATE_TYPE;
signal  flag : bit;   -- para fazer o etste no ultimo bit de cada palavra.

BEGIN

   -- Descricao Parte Controle
        PROCESS (clk,reset)
        BEGIN                                           -- inicio do processo
                IF (reset = '0') THEN
                        prox_state <= estado_0;
                        state <= B"00000";
       flag <= '0';
                ELSIF (clk'EVENT AND clk = '1') THEN
                                CASE prox_state IS
                                        WHEN estado_0 =>                        -- inicializa
dado de memoria e endereco inicial
                                                IF (EN_TESTE = '1') THEN
                                                        prox_state <= estado_1;
                                                        state <= B"00001";
                                                ELSE
                                                        prox_state <= estado_0;
                                                        state <= B"00000";
                                                END IF;
                                        WHEN estado_1 =>                        -- escreve na
memoria
                                                prox_state <= estado_2;
                                                state <= B"00010";
                                        WHEN estado_2 =>                -- desabilita escrita na
memoria
                                                prox_state <= estado_3; -- ender = ender +1
                                                state <= B"00011";
                                        WHEN estado_3 =>                        -- incrementa
endereco memoria
                                                IF (ender = fim_memoria) THEN   -- testa se
chegou ao fim da memoria
                                                        prox_state <= estado_4;
                                                        state <= B"00100";
                                                ELSE
                                                        prox_state <= estado_1;
                                                        state <= B"00001";
                                                END IF;
```

```
                                        WHEN estado_4 =>    -- coloca falha na entrada da RAM
                                                prox_state <= estado_7;
                                                state <= B"00111";
                                        WHEN estado_5 =>        -- habilita escrita
                                                prox_state <= estado_6;
                                                state <= B"00110";
                                        WHEN estado_6 =>        -- desabilita escrita RAM
                                                prox_state <= estado_7;
                                                state <= B"00111";

                                        -- inicio do ciclo de leitura


                                        WHEN estado_7 =>                        -- zera o
endereco de memoria
                                                prox_state <= estado_8; -- nao e' necessario
habilitar leitura
                                                state <= B"01000";              -- a memoria
utilizada (libr.) tem leitura sempre habilitada

                -- endereco = temp_end2;
                                        WHEN estado_8 =>                -- habilita leitura da
memoria
                                                prox_state <= estado_9;
                                                state <= B"01001";
                                        WHEN estado_9 =>                --  le o dado da
memoria e testa se houve erro
                                                prox_state <= estado_10;
                                                state <= B"01010";
                                        WHEN estado_10 =>                       -- calcula novo
valor do bit de memoria
                                                prox_state <= estado_11;

                                                state <= B"01011";
                                        WHEN estado_11 =>               -- escreve novo valor do
bit no RDM
                                                prox_state <= estado_12;
                                                state <= B"01100";
                                        WHEN estado_12 =>               -- habilita escrita na
memoria
                                                prox_state <= estado_13;
                                                state <= B"01101";
                                        WHEN estado_13 =>                       -- desabilita
escrita na memoria
                                                prox_state <= estado_14;  -- ender = ender + 1
                                                state <= B"01110";
                                        WHEN estado_14 =>
                                                IF (ender = fim_memoria) THEN   -- testa se
chegou ao fim da memoria
                                                        prox_state <= estado_15;  -- incrementa
endereco de memoria real
                                                        state <= B"01111";
                                                ELSE
                                                        prox_state <= estado_8;
                                                        state <= B"01000";
```

```
                                        END IF;
                                    WHEN estado_15 =>                   -- incrementar o i.
                                        prox_state <= estado_16;
                                        state <= B"10000";
                                    WHEN estado_16 =>                   -- testa se i chegou ao
fim
                                   if (flag = '0') THEN
                                            prox_state <= estado_7;
                                            state <= B"00111";
                                      IF (i = fim_palavra) then
                        flag <= '1';
                     end if;
                                      ELSE
                                                prox_state <= estado_17;
                                                state <= B"10001";
                      flag <= '0';
                                      END IF;
                                    WHEN estado_17 =>
                                                prox_state <= estado_20;
                                                state <= B"10100";
                                    WHEN estado_18 =>                   -- habilita escrita na
memoria
                                        prox_state <= estado_19;
                                        state <= B"10011";
                                    WHEN estado_19 =>
                                        prox_state <= estado_20; -- desabilita escrita na
memoria
                                        state <= B"10100";
                                    WHEN estado_20 =>                   --  reinicializa o i e o
erro.
                                        prox_state <= estado_21;
                                        state <= B"10101";
                                    WHEN estado_21 =>                   -- endereco =
max_mem
                                        prox_state <= estado_22;
                                        state <= B"10110";
                                    WHEN estado_22 =>                   -- habilita leitura da
memoria
                                        prox_state <= estado_23;
                                        state <= B"10111";
                                    WHEN estado_23 =>                   -- le um bit de um
endereco e testa se houve erro
                                        prox_state <= estado_24;
                                        state <= B"11000";
                                    WHEN estado_24 =>       -- calcula valor inverso no bit
lido
                                        prox_state <= estado_25;
                                        state <= B"11001";
                                    WHEN estado_25 =>                   -- escreve valor novo no
RDM
                                        prox_state <= estado_26;
                                        state <= B"11010";
                                    WHEN estado_26 => -- habilita escrita na memoria
                                        prox_state <= estado_27;
                                        state <= B"11011";
```

```
                                        WHEN estado_27 =>               -- desabilita escrita na
memoria
                                                prox_state <= estado_28; -- decrementa endereco
                                                state <= B"11100";
                                        WHEN estado_28 =>               -- testa se chegou no
inicio da memoria
                                                IF (ender = -1) THEN    -- e decrementa
endereco real
                                                        prox_state <= estado_29;
                                                        state <= B"11101";
                                                ELSE
                                                        prox_state <= estado_22;
                                                        state <= B"10110";
                                                END IF;
                                        WHEN estado_29 =>               -- incrementar o i.
                                                prox_state <= estado_30;
                                                state <= B"11110";
                                        WHEN estado_30 =>               -- testa se i chegou ao
fim
                                          if (flag = '0') THEN
                                                  prox_state <= estado_21;
                                                  state <= B"10101";
                                                  IF (i = fim_palavra) THEN
                    flag <= '1';
                end if;
                                          ELSE
                                                        prox_state <= estado_31;
                                                        state <= B"11111";
                  flag <= '0';
                                                END IF;
                                        WHEN estado_31 =>               -- finaliza o teste
                                                prox_state <= estado_0;
                                                state <= B"00000";
                                        WHEN OTHERS =>
                                                prox_state <= estado_0;
                                                state <= B"00000";
                                END CASE;
                  END IF;
          END PROCESS;  -- Fim descricao PC
END comportamento;
```

EPROM_c_bist.vhd

```
--------------------------------------------------------------------------------
-- Proposta de Tese - Estudo de caso                                          --
-- Memorias ROM com teste de paridade embutido.
-- Por Erika Cota, novembro de 2000.
--------------------------------------------------------------------------------
```

```
library lpm;                    -- Biblioteca dos componentes pré-caracterizados da Altera
use lpm.lpm_components.all;

entity eprom_c_bist is
  port(
       endereco  : in bit_vector(7 downto 0);
       eprom_out   : out bit_vector(7 downto 0);
                     erro_ROM  : out bit                    -- sinal indicando falha na RAM
      );

end eprom_c_bist;

architecture bist of eprom_c_bist is

-------------------------------------------------------------------------------------
---------------------------- Componentes externos à descrição -------------------------
-------------------------------------------------------------------------------------

CONSTANT word_width : INTEGER := 9;


---------------------- Elemento de mem□ria ROM do microcontrolador ----------------------
component lpm_rom
  generic (
           LPM_WIDTH           : POSITIVE;
           LPM_WIDTHAD         : POSITIVE;
           LPM_FILE            : STRING;
           LPM_ADDRESS_CONTROL : STRING;
                       LPM_OUTDATA         : STRING);

  port (
        address  : in bit_vector(LPM_WIDTHAD-1 downto 0);
        q        : out bit_vector(LPM_WIDTH-1 downto 0)
       );
end component;

component check_parity
  port(
       dado       : in bit_vector(word_width-1 downto 0);   -- sinal a ser testado
                    erro       : out bit                    -- sinal indicando falha de paridade no dado
      );

end component;

signal dado    : bit_vector(word_width-1 downto 0);

BEGIN

ROM_interna: lpm_rom
  generic map ( word_width, 8, "eprom2.mif", "UNREGISTERED","UNREGISTERED")
  port map (
            endereco,
            dado
           );
```

```
verifica: check_parity
  port map (
            dado,
                        erro_ROM
  );

eprom_out <= dado(word_width-2 downto 0);

end bist;
```

Check_parity_decod.vhd

```
--------------------------------------------------------------------------------
-- Proposta de Tese - Estudo de caso                                          --
-- teste de memorias ROM: verificador de paridade.
-- Por Erika Cota, novembro de 2000.
--------------------------------------------------------------------------------

library lpm;
use lpm.lpm_components.all;

entity check_parity_decod is
  port(
        dado       : in bit_vector(20 downto 0);  -- sinal a ser testado
                     erro       : out bit                    -- sinal indicando falha de paridade no dado
      );
end check_parity_decod;

architecture verifica of check_parity_decod  is

--------------------------------------------------------------------------------
------------------------------ Componentes externos à descrição ----------------------------
--------------------------------------------------------------------------------

BEGIN
    erro <= dado(0) XOR dado(1) xor dado(2) xor dado(3) xor dado(4) xor dado(5) xor dado(6) xor
dado(7) xor dado(8)
xor dado(9) xor dado(10) XOR dado(11)  xor dado(12) xor dado(13) xor dado(14) xor dado(15) xor
dado(16) xor dado(17)
xor dado(18) xor dado(19) xor dado(20);
end verifica;
```

Check_parity.vhd

```
--------------------------------------------------------------------------------
```

```
-- Proposta de Tese - Estudo de caso                                        --
-- teste de memorias ROM: verificador de paridade.
-- Por Erika Cota, novembro de 2000.
-------------------------------------------------------------------------------

library lpm;                -- Biblioteca dos componentes pré-caracterizados da Altera
use lpm.lpm_components.all;

entity check_parity is
  port(
       dado       : in bit_vector(8 downto 0);   -- sinal a ser testado
                    erro       : out bit                    -- sinal indicando falha de paridade no dado
     );
end check_parity;

architecture verifica of check_parity  is

-------------------------------------------------------------------------------
---------------------------- Componentes externos à descrição ----------------------------
-------------------------------------------------------------------------------

begin
   erro <= dado(0) XOR dado(1)  xor dado(2) xor dado(3) xor dado(4) xor dado(5) xor dado(6) xor
dado(7) xor dado(8);
end verifica;
```

March.vhd

```
-- MODULO: MARCH.VHD

-- Proposta de Tese - Estudo de caso
-- Algoritmo MARCH utilizado para teste da RAM presente no soc_8051 da aplicacao CANETA
TRADUTORA
-- Erika Cota, 14/11/2000.
--
--
-- CPGCC - CMP117
-- Trabalho Final: Teste implementacao de auto-teste da RAM.
-- Autora: Erika Fernandes Cota & Margrit Reni Krug
-- Cronograma: 03/12/98 - Primeira descricao
--                                98 - Primeira versao funcionando.
-- Objetivo: Esta descricao pretende implementar o algoritmo MARCH para geracao e
-- aplicacao de vetores de teste para Memoria RAM.

-- Descricao Comportamental.

LIBRARY ieee;
          USE ieee.std_logic_arith.all;
          USE ieee.std_logic_1164.all;
```

```
LIBRARY lpm;
USE lpm.lpm_components.ALL;

ENTITY  march IS

PORT
(
        clk                     : IN    std_logic;          -- relogio
        reset           : IN    std_logic;
        EN_TESTE        : IN    std_logic;          -- habilita geracao de vetores. Recebe valor do
sinal de teste primario
        dado            : IN    std_logic_VECTOR (7 DOWNTO 0); -- palavra de memoria
        pronto          : OUT   std_logic;      -- sinal de fim de teste da RAM
        erro            : OUT   std_logic;          -- sinal de existencia de erro na memoria
        hab_mem_wr      : OUT   std_logic;   -- habilita escrita na memoria
        hab_mem_rd      : OUT   std_logic; -- habilita leitura da memoria
        vetor           : OUT   std_logic_VECTOR (7 DOWNTO 0);   -- vetor de teste ;
        endereco_mem: OUT       std_logic_VECTOR (7 DOWNTO 0)    -- endereco de memoria
);
END march;

ARCHITECTURE estrutura OF march IS

 -- declaracao de componentes

COMPONENT lpm_add_sub
  GENERIC (LPM_WIDTH: POSITIVE
                        );
  PORT (dataa, datab: IN std_logic_VECTOR(LPM_WIDTH-1 DOWNTO 0);
          add_sub: IN STD_LOGIC;
    result: OUT std_logic_VECTOR(LPM_WIDTH-1 DOWNTO 0)
                    );
END COMPONENT;

COMPONENT FSMmarch
PORT
(
        clk                     : IN    std_logic;          -- relogio
        reset           : IN    std_logic;
        EN_TESTE        : IN    std_logic;          -- habilita geracao de vetores. Recebe valor do
sinal de teste primario
        ender           : IN    INTEGER RANGE -1 TO 256;
        i                       : IN    INTEGER RANGE 0 TO 7;
        state           : OUT   std_logic_VECTOR (4 DOWNTO 0)  -- estado atual para PO
);
END COMPONENT;

COMPONENT  POmarch
PORT
(
        clk                     : IN    std_logic;          -- relogio
        state           : IN    std_logic_VECTOR (4 DOWNTO 0);  -- estado atual do controle
        temp_end        : IN    std_logic_VECTOR (7 DOWNTO 0);
        dado            : IN    std_logic_VECTOR (7 DOWNTO 0); -- palavra de memoria
        pronto          : OUT   std_logic;      -- sinal de fim de teste da RAM
```

```
        erro            : OUT   std_logic;          -- sinal de existencia de erro na memoria
        hab_mem_wr      : OUT   std_logic;   -- habilita escrita na memoria
        hab_mem_rd      : OUT   std_logic; -- habilita leitura da memoria
        soma_sub        : OUT   STD_LOGIC; -- indica se e' soma (1) ou subtracao (0).
        i2                      : OUT    INTEGER RANGE 0 TO 7;
        ender2          : OUT   INTEGER range -1 TO 256;
        vetor           : OUT   std_logic_VECTOR (7 DOWNTO 0);   -- vetor de teste ;
        endereco        : OUT   std_logic_VECTOR (7 DOWNTO 0)    -- endereco de memoria
);
END COMPONENT;

SIGNAL um: std_logic_vector(7 downto 0);
SIGNAL estado : std_logic_VECTOR (4 DOWNTO 0);        -- estado atual da FSM
SIGNAL end_mem: INTEGER RANGE -1 to 256;   -- controle enderecamento de memoria
SIGNAL opera1 : std_logic_VECTOR (7 DOWNTO 0);   -- endereco de memoria
SIGNAL soma_res: std_logic_VECTOR (7 DOWNTO 0);  -- resultado do somador
SIGNAL oper: STD_LOGIC;  -- indica soma (1) ou subtracao (0)
SIGNAL num_std_logic: INTEGER RANGE 0 TO 7; -- numero do std_logic na palavra de memoria

BEGIN

        um <= "00000001";--X"01";
        incr : lpm_add_sub
                generic map(8)
                port map(opera1, um, oper, soma_res);
        FSM: FSMmarch
                 port map (clk      => clk,
                                reset => reset,
                                EN_TESTE      => EN_TESTE,
                                ender => end_mem,
                        i => num_std_logic,
                                state => estado
                 );

        PO: POmarch
                port map(clk => clk,
                                state => estado,
                                temp_end => soma_res,
                                dado => dado,
                                pronto => pronto,
                                erro => erro,
                                hab_mem_wr      => hab_mem_wr,
                                hab_mem_rd      => hab_mem_rd,
                                soma_sub => oper,
                                i2          => num_std_logic,
                                ender2  => end_mem,
                                vetor => vetor,
                                endereco => opera1
                );
        endereco_mem <= opera1;
END estrutura;
```

BS_Cell.vhd

```
--------------------------------------------------------------------------------
-- Dissertação de Mestrado 2001/02                                             --
-- Célula básica para a cadeia Boundary Scan                                                                  --
-- Versão para System-on-Chip 8051                                             --
--                                                --
-- Por Eduardo Santos Back em dezembro de 2000                                                                --

 LIBRARY ieee;
 USE ieee.std_logic_1164.all;

entity bs_cell is
  port(
       Data_in      : in std_logic;          -- Pino de entrada de dados na célula BS
       Data_out     : out std_logic;          -- Pino de saída de dados da célula BS
       Scan_in      : in std_logic;          -- Pino de entrada da cadeia BS
       Scan_out     : out std_logic;          -- Pino de saída da cadeia BS
                    clock                : in std_logic;                -- Entrada de clock
                    Shift_BS             : in std_logic;                        -- Seletor do MUX de entrada da
célula BS, que seleciona Data_In ou Scan_In
                    Mode_BS                      : in std_logic;                         -- Seletor do MUX de
saída da célula BS; 0 -> Normal; 1 -> Em teste
                    carrega_BS           : in std_logic;                -- Habilita o primeiro flip-flop
                    armazena_BS          : in std_logic;                        -- Habilita o segundo flip-flop
                    reset_BS             : in std_logic                         -- Coloca as saídas dos flip-flops
em zero
      );
end bs_cell;

architecture A_bs_cell of bs_cell is
           signal Out_mux_1: std_logic;              -- Saída do Mux_1
           signal Out_ff1: std_logic;                                          -- Saída do flip-flop_1
           signal Out_ff2: std_logic;                                          -- Saída do flip-flop_2

begin
process (clock, reset_BS)
           begin
                    if reset_BS = '1' then -- reset assíncrono
                              Out_ff1 <='0';
                              Out_ff2 <='0';
                    elsif (clock='0' and not clock'stable) then
                              if carrega_BS='1' then -- ativação do primeiro flip-flop
                                        Scan_out <= Out_mux_1;
                                        Out_ff1 <= Out_mux_1;
                              end if;
                    end if;
end process;
process (clock, reset_BS)
           begin
                    if reset_BS = '1' then -- reset assíncrono
                              Out_ff1 <='0';
                              Out_ff2 <='0';
```

```
                elsif (clock'event and clock = '0' ) then
                        if armazena_BS = '1' then Out_ff2 <= Out_ff1; -- ativação do segundo flip-
flop
                        end if;
                end if;
end process;

-- Mux de entrada da célula BS
with Shift_BS select
        Out_mux_1 <= Data_in when '0',
                                Scan_in when others;
-- Mux de saída da célula BS
with Mode_BS select
        Data_out <= Data_in when '0',
                                Out_ff2 when others;
end A_bs_cell;
```

Int_Cell.vhd

```
--------------------------------------------------------------------------------
-- Dissertação de Mestrado 2001/02                                            --
-- Célula básica para a cadeia interna, compatível com a célula Boundary Scan --
-- Versão para System-on-Chip 8051                                            --
--                                                                            --
-- Por Eduardo Santos Back em dezembro de 2000                                --

LIBRARY ieee;
USE ieee.std_logic_1164.all;

entity int_cell is
  port(
      Data_in      : in std_logic;       -- Pino de entrada de dados na célula interna
      Data_out     : out std_logic;      -- Pino de saída de dados da célula interna
      Scan_in      : in std_logic;       -- Pino de entrada da cadeia interna
      Scan_out     : out std_logic;      -- Pino de saída da cadeia interna
                   Clock                : in std_logic;              -- Entrada de clock
                   Shift_BS             : in std_logic;              -- Seletor do MUX de entrada da célula
interna, que seleciona Data_In ou Scan_In
                   Mode                 : in std_logic;              -- Seletor do MUX de saída da célula
interna; 0 -> Normal; 1 -> Em teste
                   carrega_BS           : in std_logic;              -- Habilita o primeiro flip-flop
                   armazena_BS          : in std_logic;              -- Habilita o segundo flip-flop
                   reset_BS             : in std_logic               -- Coloca as saídas dos flip-flops em zero
     );

end int_cell;

architecture A_int_cell of int_cell is
        signal Out_mux_1: std_logic;          -- Saída do Mux_1
        signal Out_mux_2: std_logic;          -- Saída do Mux_2
        signal Out_ff1: std_logic;                              -- Saída do flip-flop_1
```

```
begin

process (Clock, reset_BS)
	begin
		if reset_BS = '1' then -- reset assíncrono
			Out_mux_1 <='0';
			Out_mux_2 <='0';
			Scan_out  <='0';
		elsif (clock='0' and not clock'stable) then
			if carrega_BS='1' then -- habilita primeiro flip-flop
				Scan_out <= Out_mux_1;
				Out_ff1 <= Out_mux_1;
			end if;
		end if;
end process;



process (Clock, reset_BS)
	begin
		if reset_BS = '1' then -- reset assíncrono
			Out_ff1 <='0';
			Data_out <='0';
		elsif (clock='0' and not clock'stable) then
			if armazena_BS = '1' then Data_out <= Out_mux_2; -- habilita segundo flip-
flop
			end if;
		end if;
end process;


-- Mux de entrada da célula interna

with Shift_BS select

	Out_mux_1 <= Data_in when '0',

				Scan_in when others;


--Mux de saída da célula interna

with Mode select

	 Out_mux_2 <= Data_in when '0',

	   Out_ff1 when others;

end A_int_cell;
```

Reg3_BS.vhd

```
--------------------------------------------------------------------------------
-- Dissertação de Mestrado 2001/02                                            --
-- Célula de 3 bits para a cadeia interna com célula BS para não interferir no --
-- funcionamento do chip, já que substitui um fio                             --
-- Versão para System-on-Chip 8051                                            --
-- Por Eduardo Santos Back em janeiro de 2001                                 --

entity reg3_bs is
  port(
       Data_in       : in bit_vector(2 downto 0);  -- Pino de entrada de dados no reg BS
       Data_out      : out bit_vector(2 downto 0); -- Pino de saída de dados no reg BS
       Scan_in       : in bit;             -- Pino de entrada da cadeia BS
       Scan_out      : out bit;            -- Pino de saída da cadeia BS
                     Clock_BS              : in bit;                    -- Entrada de clock
                     shift_bs    : in bit;                    -- Seletor do MUX de entrada da célula
BS, que seleciona Data_In ou Scan_In
                     Mode                  : in bit;                    -- Seletor do MUX de saída da
célula BS; 0 -> Normal; 1 -> Em teste
                     carrega_BS            : in bit;                    -- Habilita o primeiro flip-flop
                     armazena_BS           : in bit;                    -- Habilita o segundo flip-flop
                     reset_BS              : in bit                     -- Coloca as saídas dos flip-
flops em zero
       );
end reg3_bs;

architecture estrutura of reg3_bs is
component bs_cell
  port(
       Data_in       : in bit;             -- Pino de entrada de dados na célula BS
       Data_out      : out bit;            -- Pino de saída de dados da célula BS
       Scan_in       : in bit;             -- Pino de entrada da cadeia BS
       Scan_out      : out bit;            -- Pino de saída da cadeia BS
                     Clock_BS              : in bit;                    -- Entrada de clock
                     shift_bs    : in bit;                    -- Seletor do MUX de entrada da célula
BS, que seleciona Data_In ou Scan_In
                     Mode_BS                     : in bit;                          -- Seletor do MUX de
saída da célula BS; 0 -> Normal; 1 -> Em teste
                     carrega_BS            : in bit;                    -- Habilita o primeiro flip-flop
                     armazena_BS           : in bit;                    -- Habilita o segundo flip-flop
                     reset_BS              : in bit                     -- Coloca as saídas dos flip-
flops em zero
       );
end component;

signal scan_out_bit2: bit;
signal scan_out_bit1: bit;
signal scan_out_bit0: bit;

begin
  bit2: bs_cell
  port map(
```

```
        Data_in => Data_in(2),
        Data_out => Data_out(2),
        Scan_in => Scan_in,
        Scan_out => scan_out_bit2,
        Clock_BS => Clock_BS,
        shift_bs => shift_bs,
        Mode_BS => Mode,
        carrega_BS => carrega_BS,
        armazena_BS => armazena_BS,
        reset_BS => reset_BS
  );

  bit1: bs_cell
  port map(
        Data_in => Data_in(1),
        Data_out => Data_out(1),
        Scan_in => Scan_out_bit2,
        Scan_out => scan_out_bit1,
        Clock_BS => Clock_BS,
        shift_bs => shift_bs,
        Mode_BS => Mode,
        carrega_BS => carrega_BS,
        armazena_BS => armazena_BS,
        reset_BS => reset_BS
  );

  bit0: bs_cell
  port map(
        Data_in => Data_in(0),
        Data_out => Data_out(0),
        Scan_in => Scan_out_bit1,
        Scan_out => scan_out,
        Clock_BS => Clock_BS,
        shift_bs => shift_bs,
        Mode_BS => Mode,
        carrega_BS => carrega_BS,
        armazena_BS => armazena_BS,
        reset_BS => reset_BS
  );
end estrutura;
```

Reg3_int.vhd

```
--------------------------------------------------------------------------------
-- Dissertação de Mestrado 2001/02                                            --
-- Célula de 3 bits para a cadeia interna
                                --
-- Versão para System-on-Chip 8051                                            --
-- Por Eduardo Santos Back em janeiro de 2001                                 --
```

```
entity reg3_int is
  port(
       Data_in       : in bit_vector(2 downto 0);  -- Pino de entrada de dados no reg interno
       Data_out      : out bit_vector(2 downto 0); -- Pino de saída de dados no reg interno
       Scan_in       : in bit;              -- Pino de entrada da cadeia interna
       Scan_out      : out bit;             -- Pino de saída da cadeia interna
                     clock                  : in bit;                                   -- Entrada de clock
                     shift_bs    : in bit;                                 -- Seletor do MUX de entrada da célula
interna, que seleciona Data_In ou Scan_In
                     Mode                   : in bit;                                   -- Seletor do MUX de saída da
célula interna; 0 -> Normal; 1 -> Em teste
                     carrega_BS             : in bit;                                   -- Habilita o primeiro flip-flop
                     armazena_BS            : in bit;                                   -- Habilita o segundo flip-flop
                     reset_BS               : in bit                                    -- Coloca as saídas dos flip-
flops em zero
       );
end reg3_int;

architecture estrutura of reg3_int is
component int_cell
  port(
       Data_in       : in bit;              -- Pino de entrada de dados na célula interna
       Data_out      : out bit;             -- Pino de saída de dados da célula interna
       Scan_in       : in bit;              -- Pino de entrada da cadeia interna
       Scan_out      : out bit;             -- Pino de saída da cadeia interna
                     clock                  : in bit;                                   -- Entrada de clock
                     shift_bs    : in bit;                                 -- Seletor do MUX de entrada da célula
interna, que seleciona Data_In ou Scan_In
                     Mode                   : in bit;                                   -- Seletor do MUX de saída da
célula interna; 0 -> Normal; 1 -> Em teste
                     carrega_BS             : in bit;                                   -- Habilita o primeiro flip-flop
                     armazena_BS            : in bit;                                   -- Habilita o segundo flip-flop
                     reset_BS               : in bit                                    -- Coloca as saídas dos flip-
flops em zero
       );
end component;

signal scan_out_bit2: bit;
signal scan_out_bit1: bit;
signal scan_out_bit0: bit;

begin
  bit2: int_cell
  port map(
          Data_in => Data_in(2),
          Data_out => Data_out(2),
          Scan_in => Scan_in,
          Scan_out => scan_out_bit2,
          Clock,
          shift_bs => shift_bs,
          Mode,
          carrega_BS => carrega_BS,
          armazena_BS => armazena_BS,
          reset_BS => reset_BS
  );
```

```
  bit1: int_cell
  port map(
          Data_in => Data_in(1),
          Data_out => Data_out(1),
          Scan_in => Scan_out_bit2,
          Scan_out => scan_out_bit1,
          Clock,
          shift_bs => shift_bs,
          Mode,
          carrega_BS => carrega_BS,
          armazena_BS => armazena_BS,
          reset_BS => reset_BS
  );

  bit0: int_cell
  port map(
          Data_in => Data_in(0),
          Data_out => Data_out(0),
          Scan_in => Scan_out_bit1,
          Scan_out => scan_out,
          Clock,
          shift_bs => shift_bs,
          Mode,
          carrega_BS => carrega_BS,
          armazena_BS => armazena_BS,
          reset_BS => reset_BS
  );
end estrutura;
```

Reg8_BS.vhd

```
--------------------------------------------------------------------------------
-- Dissertação de Mestrado 2001/02                                            --
-- Célula de 8 bits para a cadeia interna com célula BS para não interferir no --
-- funcionamento do chip, já que substitui um fio                             --
-- Versão para System-on-Chip 8051                                            --
-- Por Eduardo Santos Back em fevereiro de 2001                               --

entity reg8_bs is
  port(
       Data_in       : in bit_vector(7 downto 0);  -- Pino de entrada de dados no reg BS
       Data_out      : out bit_vector(7 downto 0); -- Pino de saída de dados no reg BS
       Scan_in       : in bit;              -- Pino de entrada da cadeia BS
       Scan_out      : out bit;             -- Pino de saída da cadeia BS
                     Clock                  : in bit;                    -- Entrada de clock
                     shift_bs    : in bit;                   -- Seletor do MUX de entrada da célula
BS, que seleciona Data_In ou Scan_In
                     Mode                   : in bit;                    -- Seletor do MUX de saída da
célula BS; 0 -> Normal; 1 -> Em teste
                     carrega_BS             : in bit;                    -- Habilita o primeiro flip-flop
```

```
                armazena_BS     : in bit;                       -- Habilita o segundo flip-flop
                reset_BS        : in bit                        -- Coloca as saídas dos flip-
flops em zero
    );

end reg8_bs;

architecture estrutura of reg8_bs is

component bs_cell
  port(
    Data_in     : in bit;          -- Pino de entrada de dados na célula BS
    Data_out    : out bit;          -- Pino de saída de dados da célula BS
    Scan_in     : in bit;          -- Pino de entrada da cadeia BS
    Scan_out    : out bit;          -- Pino de saída da cadeia BS
                Clock     : in bit;                         -- Entrada de clock
                shift_bs  : in bit;                         -- Seletor do MUX de entrada da célula
BS, que seleciona Data_In ou Scan_In
                Mode_BS             : in bit;                       -- Seletor do MUX de
saída da célula BS; 0 -> Normal; 1 -> Em teste
                carrega_BS      : in bit;                       -- Habilita o primeiro flip-flop
                armazena_BS     : in bit;                       -- Habilita o segundo flip-flop
                reset_BS        : in bit                        -- Coloca as saídas dos flip-
flops em zero
    );
end component;

signal scan_out_bit7: bit;
signal scan_out_bit6: bit;
signal scan_out_bit5: bit;
signal scan_out_bit4: bit;
signal scan_out_bit3: bit;
signal scan_out_bit2: bit;
signal scan_out_bit1: bit;
signal scan_out_bit0: bit;

begin
  bit7: bs_cell
  port map(
        Data_in => Data_in(7),
        Data_out => Data_out(7),
        Scan_in => Scan_in,
        Scan_out => scan_out_bit7,
        clock => clock,
        shift_bs => shift_bs,
        Mode_BS => Mode,
        carrega_BS => carrega_BS,
        armazena_BS => armazena_BS,
        reset_BS => reset_BS
  );

  bit6: bs_cell
  port map(
        Data_in => Data_in(6),
        Data_out => Data_out(6),
```

```
        Scan_in => Scan_out_bit7,
        Scan_out => scan_out_bit6,
        clock => clock,
        shift_bs => shift_bs,
        Mode_BS => Mode,
        carrega_BS => carrega_BS,
        armazena_BS => armazena_BS,
        reset_BS => reset_BS
);

bit5: bs_cell
port map(
        Data_in => Data_in(5),
        Data_out => Data_out(5),
        Scan_in => Scan_out_bit6,
        Scan_out => scan_out_bit5,
        clock => clock,
        shift_bs => shift_bs,
        Mode_BS => Mode,
        carrega_BS => carrega_BS,
        armazena_BS => armazena_BS,
        reset_BS => reset_BS
);

bit4: bs_cell
port map(
        Data_in => Data_in(4),
        Data_out => Data_out(4),
        Scan_in => Scan_out_bit5,
        Scan_out => scan_out_bit4,
        clock => clock,
        shift_bs => shift_bs,
        Mode_BS => Mode,
        carrega_BS => carrega_BS,
        armazena_BS => armazena_BS,
        reset_BS => reset_BS
);


bit3: bs_cell
port map(
        Data_in => Data_in(3),
        Data_out => Data_out(3),
        Scan_in => Scan_out_bit4,
        Scan_out => scan_out_bit3,
        clock => clock,
        shift_bs => shift_bs,
        Mode_BS => Mode,
        carrega_BS => carrega_BS,
        armazena_BS => armazena_BS,
        reset_BS => reset_BS
);


bit2: bs_cell
```

```
  port map(
        Data_in => Data_in(2),
        Data_out => Data_out(2),
        Scan_in => Scan_out_bit3,
        Scan_out => scan_out_bit2,
        clock => clock,
        shift_bs => shift_bs,
        Mode_BS => Mode,
        carrega_BS => carrega_BS,
        armazena_BS => armazena_BS,
        reset_BS => reset_BS
  );



  bit1: bs_cell
  port map(
        Data_in => Data_in(1),
        Data_out => Data_out(1),
        Scan_in => Scan_out_bit2,
        Scan_out => scan_out_bit1,
        clock => clock,
        shift_bs => shift_bs,
        Mode_BS => Mode,
        carrega_BS => carrega_BS,
        armazena_BS => armazena_BS,
        reset_BS => reset_BS
  );




  bit0: bs_cell
  port map(
        Data_in => Data_in(0),
        Data_out => Data_out(0),
        Scan_in => Scan_out_bit1,
        Scan_out => scan_out,
        clock => clock,
        shift_bs => shift_bs,
        Mode_BS => Mode,
        carrega_BS => carrega_BS,
        armazena_BS => armazena_BS,
        reset_BS => reset_BS
  );
end estrutura;
```

Reg8_Int.vhd

```
--------------------------------------------------------------------------------
-- Dissertação de Mestrado 2001/02                                            --
-- Célula básica para a cadeia interna
--
-- Versão para System-on-Chip 8051                                            --
-- Por Eduardo Santos Back em março de 2001                                                --

entity reg8_int is
  port(
       Data_in       : in bit_vector(7 downto 0);  -- Pino de entrada de dados no reg interno
       Data_out      : out bit_vector(7 downto 0); -- Pino de saída de dados no reg interno
       Scan_in       : in bit;              -- Pino de entrada da cadeia interna
       Scan_out      : out bit;             -- Pino de saída da cadeia interna
                       clock                : in bit;                                  -- Entrada de clock
                       shift_bs    : in bit;                                 -- Seletor do MUX de entrada da célula
interna, que seleciona Data_In ou Scan_In
                       Mode                 : in bit;                                  -- Seletor do MUX de saída da
célula interna; 0 -> Normal; 1 -> Em teste
                       carrega_BS           : in bit;                                  -- Habilita o primeiro flip-flop
                       armazena_BS          : in bit;                                  -- Habilita o segundo flip-flop
                       reset_BS             : in bit                                   -- Coloca as saídas dos flip-
flops em zero
       );

end reg8_int;

architecture estrutura of reg8_int is

component int_cell
  port(
       Data_in       : in bit;              -- Pino de entrada de dados na célula interna
       Data_out      : out bit;             -- Pino de saída de dados da célula interna
       Scan_in       : in bit;              -- Pino de entrada da cadeia interna
       Scan_out      : out bit;             -- Pino de saída da cadeia interna
                       clock                : in bit;                                  -- Entrada de clock
                       shift_bs    : in bit;                                 -- Seletor do MUX de entrada da célula
interna, que seleciona Data_In ou Scan_In
                       Mode                 : in bit;                                  -- Seletor do MUX de saída da
célula interna; 0 -> Normal; 1 -> Em teste
                       carrega_BS           : in bit;                                  -- Habilita o primeiro flip-flop
                       armazena_BS          : in bit;                                  -- Habilita o segundo flip-flop
                       reset_BS             : in bit                                   -- Coloca as saídas dos flip-
flops em zero
       );
end component;

signal scan_out_bit7: bit;
signal scan_out_bit6: bit;
signal scan_out_bit5: bit;
signal scan_out_bit4: bit;
signal scan_out_bit3: bit;
```

```
signal scan_out_bit2: bit;
signal scan_out_bit1: bit;
signal scan_out_bit0: bit;

begin
  bit7: int_cell
  port map(
        Data_in => Data_in(7),
        Data_out => Data_out(7),
        Scan_in => Scan_in,
        Scan_out => scan_out_bit7,
        Clock,
        shift_bs => shift_bs,
        Mode,
        carrega_BS => carrega_BS,
        armazena_BS => armazena_BS,
        reset_BS => reset_BS
  );

  bit6: int_cell
  port map(
        Data_in => Data_in(6),
        Data_out => Data_out(6),
        Scan_in => Scan_out_bit7,
        Scan_out => scan_out_bit6,
        Clock,
        shift_bs => shift_bs,
        Mode,
        carrega_BS => carrega_BS,
        armazena_BS => armazena_BS,
        reset_BS => reset_BS
  );

  bit5: int_cell
  port map(
        Data_in => Data_in(5),
        Data_out => Data_out(5),
        Scan_in => Scan_out_bit6,
        Scan_out => scan_out_bit5,
        Clock,
        shift_bs => shift_bs,
        Mode,
        carrega_BS => carrega_BS,
        armazena_BS => armazena_BS,
        reset_BS => reset_BS
  );

  bit4: int_cell
  port map(
        Data_in => Data_in(4),
        Data_out => Data_out(4),
        Scan_in => Scan_out_bit5,
        Scan_out => scan_out_bit4,
        Clock,
        shift_bs => shift_bs,
```

```
        Mode,
        carrega_BS => carrega_BS,
        armazena_BS => armazena_BS,
        reset_BS => reset_BS
);

bit3: int_cell
port map(
        Data_in => Data_in(3),
        Data_out => Data_out(3),
        Scan_in => Scan_out_bit4,
        Scan_out => scan_out_bit3,
        Clock,
        shift_bs => shift_bs,
        Mode,
        carrega_BS => carrega_BS,
        armazena_BS => armazena_BS,
        reset_BS => reset_BS
);

bit2: int_cell
port map(
        Data_in => Data_in(2),
        Data_out => Data_out(2),
        Scan_in => Scan_out_bit3,
        Scan_out => scan_out_bit2,
        Clock,
        shift_bs => shift_bs,
        Mode,
        carrega_BS => carrega_BS,
        armazena_BS => armazena_BS,
        reset_BS => reset_BS
);


bit1: int_cell
port map(
        Data_in => Data_in(1),
        Data_out => Data_out(1),
        Scan_in => Scan_out_bit2,
        Scan_out => scan_out_bit1,
        Clock,
        shift_bs => shift_bs,
        Mode,
        carrega_BS => carrega_BS,
        armazena_BS => armazena_BS,
        reset_BS => reset_BS
);


bit0: int_cell
port map(
        Data_in => Data_in(0),
        Data_out => Data_out(0),
        Scan_in => Scan_out_bit1,
```

```
        Scan_out => scan_out,
        Clock,
        shift_bs => shift_bs,
        Mode,
        carrega_BS => carrega_BS,
        armazena_BS => armazena_BS,
        reset_BS => reset_BS
  );
end estrutura;
```

soc_8051.vhd

```
-------------------------------------------------------------------------------
-- Dissertação de Mestrado 2001/02                                           --
-- Microcontrolador 8051 com teste IEEE1149.1                                --
-- Versão para System-on-Chip                                                --
-- Por Eduardo Santos Back em maio de 2001                                   --
-------------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.all;
entity soc_8051 is
  port(
      clock     : in std_logic;                    -- Sinal de sincronismo externo
      reset     : in std_logic;                    -- Sinal de inicialização
      P0_in     : in std_logic_vector(7 downto 0);   -- Porta de entrada 0 (dado)
      P0_add    : out std_logic_vector(7 downto 0);  -- Porta de saída 0 (endereço)
      P0_out    : out std_logic_vector(7 downto 0);  -- Porta de saída 0 (dado)
      P1_in     : in std_logic_vector(7 downto 0);   -- Porta de entrada 1
      P2_out    : out std_logic_vector(7 downto 0);  -- Porta de saída 2
      P3_out    : out std_logic_vector(3 downto 0);  -- Porta de saída 3
      P3_in     : in std_logic_vector(3 downto 0);   -- Porta de entrada 3
      -- Sinais para a cadeia interna do soc_8051
      TDI                       : in std_logic;
      TDO                       : out std_logic;
      reset_bs    : in std_logic;
      carrega_bs  : in std_logic;
      armazena_bs  : in std_logic;
      mode                      : in std_logic;
      shift_bs      : in std_logic;
                    -- Sinais para a cadeia BIST
      Shift_BIST  : in std_logic;                  -- Saída para Shift_DR

      Mode_BIST  : in std_logic;     -- Saída para Mode
                   carrega_BIST  : in std_logic;          -- Habilita o primeiro flip-flop
                   armazena_BIST : in std_logic;        -- Habilita o segundo flip-flop
                   TDI_BIST          : in std_logic;     -- TDI para células BS (OUT)
                   TDO_BIST          : out std_logic;   -- TDO das células BS (IN)
      erro_eprom    : out std_logic;   -- indicacao de erro na memoria de programa
      erro_decod    : out std_logic;   -- indicacao de erro na memoria de decodificação
```

```
                    erro_RAM_ext  : out std_logic    -- sinal indicando falha na RAM
  );

end soc_8051;

architecture A_soc_8051 of soc_8051 is

-------------------------------------------------------------------------------
----------------------------- Componentes externos à descrição --------------------------
-------------------------------------------------------------------------------

----------------------- Elemento de memória RAM do microcontrolador ----------------------

----------------------- Nucleo com BIST interno ----------------------

COMPONENT mem_c_bist
port (
                    clk                         : in std_logic;
                    reset               : in std_logic;
                    EN_TESTE            : in std_logic;    -- sinal indicando modo teste. Inserido por Erika
& Margrit em 23/11/98.
                    iram_in       : in std_logic_vector(7 downto 0); -- entrada da RAM
      iram_address  : in std_logic_vector(7 downto 0); -- Endereço da memória de dados
      iram_wr       : in std_logic;                 -- Sinal de escrita na memória de dados
      iram_out      : out std_logic_vector(7 downto 0); -- Saída da memória de dados
                    erro_RAM            : out std_logic; -- sinal indicando falha na RAM
                    fim                         : out std_logic  -- sinal indicando fim do teste da RAM
          );

end component;

----------------------- Elemento de memória ROM do microcontrolador com teste embutido ----------
component eprom_c_bist
  port(
      endereco  : in std_logic_vector(7 downto 0);
      eprom_out   : out std_logic_vector(7 downto 0);
                    erro_ROM  : out std_logic                   -- sinal indicando falha na RAM
    );

end component;

component decod_c_bist
  port(
      endereco  : in std_logic_vector(7 downto 0);
      decod_out   : out std_logic_vector(19 downto 0);
                    erro_ROM  : out std_logic                   -- sinal indicando falha na RAM
    );

end component;
---------------------------- Circuito do contador de programa PC --------------------------
component ender
 port (
      clock     : in std_logic;                    -- Sinal de sincronismo
      reset     : in std_logic;                   -- Sinal de inicialização
      hab       : in std_logic;                    -- Sinal de escrita de endereço
```

```
      ender_in  : in std_logic_vector(7 downto 0);        -- Endereço a ser escrito
      inc       : in std_logic;                          -- Sinal de incremento do endereço
      ender_out : out std_logic_vector(7 downto 0)       -- Endereço gerado
    );
end component;

------------------------- Circuito da unidade lógica e aritmética -----------------------
component ula8
  port (
       ula_A         : in std_logic_vector(7 downto 0);
       ula_B         : in std_logic_vector(7 downto 0);
       ula_operation : in std_logic_vector(6 downto 0);
       carry_in      : in std_logic;
       ula_result    : out std_logic_vector(7 downto 0);
       carry_out     : out std_logic
      );
end component;

---------------------------- Registrador BS de 3 std_logics -------------------------------------
component reg3_int
  port(
      Data_in       : in std_logic_vector(2 downto 0);  -- Pino de entrada de dados no reg BS
      Data_out      : out std_logic_vector(2 downto 0); -- Pino de saída de dados no reg BS
      Scan_in       : in std_logic;             -- Pino de entrada da cadeia BS
      Scan_out      : out std_logic;            -- Pino de saída da cadeia BS
                      clock       : in std_logic;                                  -- Entrada de clock
                      shift_bs    : in std_logic;                          -- Seletor do MUX de entrada da célula
BS, que seleciona Data_In ou Scan_In
                      Mode                : in std_logic;                          -- Seletor do MUX de saída da
célula BS; 0 -> Normal; 1 -> Em teste
                      carrega_BS          : in std_logic;                          -- Habilita o primeiro flip-flop
                      armazena_BS         : in std_logic;                          -- Habilita o segundo flip-flop
                      reset_BS            : in std_logic                           -- Coloca as saídas dos flip-
flops em zero
      );
end component;
---------------------------- Registrador Int de 8 std_logics -------------------------------------
component reg8_int
  port(
      Data_in       : in std_logic_vector(7 downto 0);  -- Pino de entrada de dados no reg BS
      Data_out      : out std_logic_vector(7 downto 0); -- Pino de saída de dados no reg BS
      Scan_in       : in std_logic;             -- Pino de entrada da cadeia BS
      Scan_out      : out std_logic;            -- Pino de saída da cadeia BS
                      clock       : in std_logic;                                  -- Entrada de clock
                      shift_bs    : in std_logic;                          -- Seletor do MUX de entrada da célula
BS, que seleciona Data_In ou Scan_In
                      Mode                : in std_logic;                          -- Seletor do MUX de saída da
célula BS; 0 -> Normal; 1 -> Em teste
                      carrega_BS          : in std_logic;                          -- Habilita o primeiro flip-flop
                      armazena_BS         : in std_logic;                          -- Habilita o segundo flip-flop
                      reset_BS            : in std_logic                           -- Coloca as saídas dos flip-
flops em zero
      );

end component;
```

```
---------------------------- Registrador BS de 8 std_logics -------------------------------------
component reg8_bs
  port(
       Data_in       : in std_logic_vector(7 downto 0);  -- Pino de entrada de dados no reg BS
       Data_out      : out std_logic_vector(7 downto 0); -- Pino de saída de dados no reg BS
       Scan_in       : in std_logic;           -- Pino de entrada da cadeia BS
       Scan_out      : out std_logic;          -- Pino de saída da cadeia BS
                     clock       : in std_logic;                                  -- Entrada de clock
                     shift_bs    : in std_logic;                          -- Seletor do MUX de entrada da célula
BS, que seleciona Data_In ou Scan_In
                     Mode                : in std_logic;                  -- Seletor do MUX de saída da
célula BS; 0 -> Normal; 1 -> Em teste
                     carrega_BS          : in std_logic;                  -- Habilita o primeiro flip-flop
                     armazena_BS         : in std_logic;                  -- Habilita o segundo flip-flop
                     reset_BS            : in std_logic                   -- Coloca as saídas dos flip-
flops em zero
       );

end component;

------------------------- Célula BS básica -----------------------
component bs_cell
  port(
       Data_in       : in std_logic;  -- Pino de entrada de dados no reg BS
       Data_out      : out std_logic; -- Pino de saída de dados no reg BS
       Scan_in       : in std_logic;  -- Pino de entrada da cadeia BS
       Scan_out      : out std_logic; -- Pino de saída da cadeia BS
                     clock       : in std_logic;       -- Entrada de clock
                     shift_bs    : in std_logic;  -- Seletor do MUX de entrada da célula BS, que seleciona
Data_In ou Scan_In
                     Mode_BS                 : in std_logic;  -- Seletor do MUX de saída da célula BS;
0 -> Normal; 1 -> Em teste
                     carrega_BS          : in std_logic;  -- Habilita o primeiro flip-flop
                     armazena_BS         : in std_logic;  -- Habilita o segundo flip-flop
                     reset_BS            : in std_logic   -- Coloca as saídas dos flip-flops em zero
       );
end component;


------------------------- Célula interna básica ----------------
component int_cell
  port(
       Data_in       : in std_logic;  -- Pino de entrada de dados no reg BS
       Data_out      : out std_logic; -- Pino de saída de dados no reg BS
       Scan_in       : in std_logic;  -- Pino de entrada da cadeia BS
       Scan_out      : out std_logic; -- Pino de saída da cadeia BS
                     clock       : in std_logic;       -- Entrada de clock
                     shift_bs    : in std_logic;  -- Seletor do MUX de entrada da célula BS, que seleciona
Data_In ou Scan_In
                     Mode                : in std_logic;  -- Seletor do MUX de saída da célula BS; 0 ->
Normal; 1 -> Em teste
                     carrega_BS          : in std_logic;  -- Habilita o primeiro flip-flop
                     armazena_BS         : in std_logic;  -- Habilita o segundo flip-flop
                     reset_BS            : in std_logic   -- Coloca as saídas dos flip-flops em zero
```

```
      );
end component;
------------------- Declaração dos sinais internos do microcontrolador -------------------

-- Sinais da memória RAM interna
signal iram_in        : std_logic_vector(7 downto 0); -- Entrada da memória de dados
signal iram_address   : std_logic_vector(7 downto 0); -- Endereço da memória de dados
signal iram_wr        : std_logic;                    -- Sinal de escrita na memória de dados
signal iram_out       : std_logic_vector(7 downto 0); -- Saída da memória de dados
signal fim_ram        : std_logic;                                    -- fim do teste da ram
signal en_teste       : std_logic;
signal TDI_Reg_err_bist   : std_logic;
signal erro_RAM                 : std_logic;

-- Sinais para o teste interno
signal iram_out_reg     : std_logic_vector(7 downto 0); -- Saída da memória de dados
signal N_reset                  : std_logic;
signal TDI_PC_BS                : std_logic;
signal PC_Out_BS                : std_logic_vector(7 downto 0);
signal IR_BS                    : std_logic_vector(7 downto 0);
signal TDI_IR_BS                : std_logic;
signal TDI_ACC_BS               : std_logic;
signal ACC_BS                   : std_logic_vector(7 downto 0);
signal TDI_ula1                 : std_logic;
signal TDI_ula2                 : std_logic;
signal TDI_carry_scan      : std_logic;
signal ula1_BS                  : std_logic_vector(7 downto 0);
signal ula2_BS                  : std_logic_vector(7 downto 0);
signal carry_flag_BS        : std_logic;
signal eprom_data_BS    : std_logic_vector(7 downto 0); -- Saída de operando imediato da ROM
signal iram_wr_BS               : std_logic;
signal iram_address_BS    : std_logic_vector(7 downto 0);
signal pc_write_BS              : std_logic;
signal Port2_out_BS     : std_logic_vector(7 downto 0); -- Porta de saída 2
signal TDI_eprom_data_scan      : std_logic;
signal TDI_iram_wr_scan : std_logic;
signal TDI_pc_write_scan: std_logic;
signal TDI_iram_out_scan        : std_logic;
signal TDI_iram_address_scan    : std_logic;
signal TDI_port2_scan      : std_logic;
signal estado_atual_BS     : std_logic_vector(2 downto 0);
signal aux_mux2_BS              : std_logic_vector(7 downto 0);
signal aux2_mux2_BS             : std_logic_vector(7 downto 0);
signal TDI_aux_mux2             : std_logic;
signal TDI_aux2_mux2     : std_logic;
signal uns                                : std_logic_vector(7 downto 0);
signal zeros                    : std_logic_vector(7 downto 0);
signal TDI_aux_eprom_data       : std_logic;
signal Aux_Eprom_data_BS        : std_logic_vector(7 downto 0);
signal Aux_iram_wr_BS   : std_logic;
signal zero                               : std_logic;
signal TDI_aux_iram_wr_scan     : std_logic;
signal TDI_aux1_mux3_scan       : std_logic;
signal TDI_aux2_mux3_scan       : std_logic;
signal aux_Iram_address_scan_data       : std_logic_vector(7 downto 0);
```

```
signal aux_Iram_addess_scan_dt_out          : std_logic_vector(7 downto 0);
signal aux1_mux3_scan_data         : std_logic_vector(7 downto 0);
signal aux1_mux3_scan_data_out   : std_logic_vector(7 downto 0);
signal aux2_mux3_scan_data         : std_logic_vector(7 downto 0);
signal aux2_mux3_scan_data_out   : std_logic_vector(7 downto 0);
signal TDI_aux_iram_address                 : std_logic;

-- Sinais da memória ROM interna
signal eprom_out      : std_logic_vector(7 downto 0); -- Saída da memória de programa
signal eprom_address  : std_logic_vector(7 downto 0); -- Endereço da memória de programa

-- Sinais da memória de decodificação
signal microinst      : std_logic_vector(19 downto 0); -- Sinais de controle das instruções

-- Sinais do contador de programa (PC)
signal PC_write       : std_logic;                  -- Sinal de escrita no PC
signal PC_inc         : std_logic;                  -- Sinal de incremento do PC
signal PC_out         : std_logic_vector(7 downto 0); -- Saída do contador de programa

-- Sinais da unidade lógica e aritmética
signal ula_1          : std_logic_vector(7 downto 0); -- Primeiro operando
signal ula_2          : std_logic_vector(7 downto 0); -- Segundo operando
signal ula_op         : std_logic_vector(6 downto 0); -- Controle da operação realizada
signal carry_flag_in  : std_logic;                  -- Mux do sinal carry da ula
signal ula_out        : std_logic_vector(7 downto 0); -- Resultado da operação
signal carry_flag_out : std_logic;                  -- Saída do sinal carry da ula
signal um                                   : std_logic;

-- Sinais gerais
signal divider        : std_logic_vector(2 downto 0); -- saída do divisor de freqüência
signal acc            : std_logic_vector(7 downto 0); -- Acumulador
signal mux_1          : std_logic_vector(7 downto 0); -- Mux para o 1º operando da ula
signal mux_2          : std_logic_vector(7 downto 0); -- Mux para o 2º operando da ula
signal mux_3          : std_logic_vector(7 downto 0); -- Mux para endereçamento da RAM
signal IR             : std_logic_vector(7 downto 0); -- Registrador de instrução
signal eprom_data     : std_logic_vector(7 downto 0); -- Saída de operando imediato da ROM
signal Port2_out      : std_logic_vector(7 downto 0); -- Porta de saída 2
signal Port3_out      : std_logic_vector(3 downto 0); -- Porta de saída 3
signal carry_flag     : std_logic;                  -- Sinalização de carry
signal zero_flag      : std_logic;                  -- Sinalização de zero no acumulador
signal PC_wr          : std_logic;                  -- Mux do sinal de escrita do PC
CONSTANT est0 : std_logic_vector(2 downto 0) := "000";
CONSTANT est1 : std_logic_vector(2 downto 0) := "001";
CONSTANT est2 : std_logic_vector(2 downto 0) := "010";
CONSTANT est3 : std_logic_vector(2 downto 0) := "011";
CONSTANT est4 : std_logic_vector(2 downto 0) := "100";
CONSTANT est5 : std_logic_vector(2 downto 0) := "101";
CONSTANT est6 : std_logic_vector(2 downto 0) := "110";
CONSTANT est7 : std_logic_vector(2 downto 0) := "111";
signal estado_atual     : std_logic_vector(2 downto 0);
signal proximo_estado   : std_logic_vector(2 downto 0);

begin

--------------------------------------------------------------------------------
```

```
---------------------------------- Máquina de estados ----------------------------------
-------------------------------------------------------------------------------------
-- Armazenamento da instrução corrente
 process(estado_atual)
  begin
    case estado_atual is
      when est1 =>
                              um <= '1';
                              aux_Iram_address_scan_data        <= "0000000" & IR_BS(0);
                              aux1_mux3_scan_data     <= "00000" & IR_BS(2 downto 0);
                              aux2_mux3_scan_data     <= "0000000" & IR_BS(0);
       IR <= eprom_out;
      when others  =>
                              um <= '1';
                              aux1_mux3_scan_data     <= "00000" & IR_BS(2 downto 0);
                              aux2_mux3_scan_data     <= "0000000" & IR_BS(0);
       IR <= IR_BS;
    end case;
 end process;

-- Sempre que a entrada reset é 1 no estado 0 a máquina não avança para os próximos estados
 process(estado_atual,reset)
  begin
   case estado_atual is
    when est0 =>
     if (reset = '1') then -- Inicializações de sinais
      proximo_estado <= est0;
                              uns                     <= "11111111";
                              zeros     <= "00000000";
                              zero      <= '0';
            else
      proximo_estado <= est1;
     end if;
    when est1 =>  proximo_estado <= est2;
    when est2 =>  proximo_estado <= est3;
    when est3 =>  proximo_estado <= est4;
    when est4 =>  proximo_estado <= est5;
    when est5 =>  proximo_estado <= est6;
    when est6 =>  proximo_estado <= est7;
    when others =>
     if (reset = '1') then
      proximo_estado <= est0;
     else
      proximo_estado <= est1;
     end if;
   end case;
 end process;

-------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------
-- Memória de dados e registradores

-- Cadeia BIST para a memória RAM
```

```
-- 1 Registrador de enable
Reg_en_BIST: bs_cell
            port map(
        Data_in                 =>      um,
        Data_out    =>      en_teste,
        Scan_in                 =>      TDI_BIST,
        Scan_out    =>      TDI_Reg_err_bist,
                    clock,
                    shift_bs => shift_bist,
                    Mode_bs                     => mode_bist,
                    carrega_BS      => carrega_bist,
                    armazena_BS     => armazena_bist,
                    reset_BS
      );

-- 2 Registrador de armazenamento de erro
Reg_err_BIST: bs_cell
            port map(
        Data_in                 =>      erro_ram,
        Data_out    =>      erro_ram_ext,
        Scan_in                 =>      TDI_Reg_err_bist,
        Scan_out    =>      TDO_BIST,
                    clock,
                    shift_bs => shift_bist,
                    Mode_bs                     => mode_bist,
                    carrega_BS      => carrega_bist,
                    armazena_BS     => armazena_bist,
                    reset_BS
      );

N_reset <= NOT(reset);

-- memória de dados e registradores: Erika, 14/11/00
testable_ram: mem_c_bist
  port map (
                    clk                             => clock,
                    reset                   => N_reset,
                    EN_TESTE        => EN_TESTE,       -- sinal indicando modo teste. Inserido por
Erika & Margrit em 23/11/98.
                    iram_in     => iram_in,         -- entrada da RAM
        iram_address  => iram_address_BS,  -- Endereço da memória de dados
        iram_wr     => iram_wr_BS,      -- Sinal de escrita na memória de dados
        iram_out    => iram_out,        -- Saída da memória de dados
                    erro_RAM        => erro_RAM,        -- sinal indicando falha na RAM
                    fim                             => fim_ram          -- sinal indicando fim do teste da RAM
  );


-- memória de programa
ROM_interna: eprom_c_bist
   port map (
            PC_out_BS,
            eprom_out,
            erro_eprom
   );
```

```
-- memória de microprograma
Decod: decod_c_bist
  port map (
          IR_BS,
          microinst,
          erro_decod
  );

-- Contador de programa
PC: ender
 port map(
          clock,
          reset,
          PC_write_BS,
          ula_out,
          PC_inc,
          PC_out
        );

-- 1 Registrador de estado_atual

estado_fsm: reg3_int
 port map (
       Data_in     => proximo_estado,
       Data_out    => estado_atual,
       Scan_in     => TDI,
       Scan_out    => TDI_PC_BS,
                      clock,                    -- => clock,
                      Shift_bs,                 --  => Shift_DR,
                      Mode,
                      carrega_BS,       --=> carrega_BS,
                      armazena_BS,      --=> --armazena_BS,
                      reset_BS                  --=> reset_BS
 );

-- 2 Registrador PC_Out
PC_BS: reg8_bs
           port map(
       Data_in                 =>        PC_out,
       Data_out     =>         PC_out_bs,
       Scan_in                 =>        TDI_PC_BS,
       Scan_out     =>         TDI_IR_BS,
                      clock,
                      shift_bs,
                      Mode,
                      carrega_BS,
                      armazena_BS,
                      reset_BS
      );

-- 3 Registrador IR
IR_scan: reg8_int
           port map(
       Data_in                 =>        IR,
```

```
        Data_out    =>      IR_BS,
        Scan_in             =>      TDI_IR_BS,
        Scan_out    =>      TDI_ACC_BS,
                    clock,
                    shift_bs,
                    Mode,
                    carrega_BS,
                    armazena_BS,
                    reset_BS
    );

-- 4 Registrador Acc
ACC_Scan: reg8_int
            port map(
        Data_in             =>      ACC,
        Data_out    =>      ACC_BS,
        Scan_in             =>      TDI_ACC_BS,
        Scan_out    =>      TDI_aux_mux2,
                    clock,
                    shift_bs,
                    Mode,
                    carrega_BS,
                    armazena_BS,
                    reset_BS
    );

-- 5 Registrador Aux_MUX2
aux_mux2_scan: reg8_bs
            port map(
        Data_in             =>      zeros,
        Data_out    =>      aux_mux2_BS,
        Scan_in             =>      TDI_aux_mux2,
        Scan_out    =>      TDI_aux2_mux2,
                    clock,
                    shift_bs,
                    Mode,
                    carrega_BS,
                    armazena_BS,
                    reset_BS
    );

-- 6 Registrador Aux2_MUX2
aux2_mux2_scan: reg8_bs
            port map(
        Data_in             =>      uns,
        Data_out    =>      aux2_mux2_BS,
        Scan_in             =>      TDI_aux2_mux2,
        Scan_out    =>      TDI_ula1,
                    clock,
                    shift_bs,
                    Mode,
                    carrega_BS,
                    armazena_BS,
                    reset_BS
    );
```

```
-- 7 Registrador ULA1
ULA1_Scan: reg8_int
          port map(
      Data_in               =>      Ula_1,
      Data_out    =>        Ula1_BS,
      Scan_in               =>      TDI_ula1,
      Scan_out    =>        TDI_ula2,
                  clock,
                  shift_bs,
                  Mode,
                  carrega_BS,
                  armazena_BS,
                  reset_BS
     );

-- 8 Registrador ULA2
ULA2_Scan: reg8_int
          port map(
      Data_in               =>      Ula_2,
      Data_out    =>        Ula2_BS,
      Scan_in               =>      TDI_ula2,
      Scan_out    =>        TDI_aux_eprom_data,
                  clock,
                  shift_bs,
                  Mode,
                  carrega_BS,
                  armazena_BS,
                  reset_BS
     );

-- 9 Registrador Aux_Eprom_DATA
Aux_Eprom_Data_scan: reg8_bs
          port map(
      Data_in               =>      zeros,
      Data_out    =>        Aux_Eprom_data_BS,
      Scan_in               =>      TDI_aux_eprom_data,
      Scan_out    =>        TDI_eprom_data_scan,
                  clock,
                  shift_bs,
                  Mode,
                  carrega_BS,
                  armazena_BS,
                  reset_BS
     );

-- 10 Registrador Eprom_DATA
Eprom_DATA_scan: reg8_int
 port map (
      Data_in     => eprom_data,
      Data_out    => eprom_data_BS,
      Scan_in     => TDI_eprom_data_scan,
      Scan_out    => TDI_iram_wr_scan,
                  clock,              -- => clock,
                  Shift_bs,           -- => Shift_DR,
```

```
                    Mode,               -- => Mode_iram,
                    carrega_BS,         -- => carrega_BS,
                    armazena_BS,        -- => armazena_BS,
                    reset_BS                    -- => reset_BS
 );

-- 11 Registrador Aux_IRAM_WR
Aux_IRAM_WR_Scan: bs_cell
 port map (
       Data_in      => zero,
       Data_out     => Aux_iram_wr_BS,
       Scan_in      => TDI_iram_wr_scan,
       Scan_out     => TDI_aux_iram_wr_scan,
                    clock,      -- => clock,
                    Shift_bs,   -- => Shift_DR,
                    Mode,       -- => Mode_iram,
                    carrega_BS,  -- => carrega_BS,
                    armazena_BS, -- => armazena_BS,
                    reset_BS             -- => reset_BS
 );

-- 12 Registrador IRAM_WR
iram_wr_scan: int_cell
 port map (
       Data_in      => iram_wr,
       Data_out     => iram_wr_BS,
       Scan_in      => TDI_aux_iram_wr_scan,
       Scan_out     => TDI_carry_scan,
                    clock,              -- => clock,
                    Shift_bs,           -- => Shift_DR,
                    Mode,               -- => Mode_iram,
                    carrega_BS,         -- => carrega_BS,
                    armazena_BS,        -- => --armazena_BS,
                    reset_BS                    -- => reset_BS
 );


-- 13 Registrador Carry_Flag
Carry_Scan: int_cell
  port map(
       Data_in              =>      Carry_flag,
       Data_out     =>      Carry_flag_BS,
       Scan_in              =>      TDI_carry_scan,
       Scan_out     =>      TDI_pc_write_scan,
                    clock,
                    Shift_BS,
                    Mode,
                    carrega_BS,
                    armazena_BS,
                    reset_BS
      );

-- 14 Registrador PC_Write
PC_Write_Scan: int_cell
 port map (
```

```
        Data_in      => PC_write,
        Data_out     => PC_write_BS,
        Scan_in      => TDI_pc_write_scan,
        Scan_out     => TDI_aux_iram_address,
                        clock,    -- =>       clock,
                        Shift_bs,    -- =>  Shift_DR,
                        Mode,         --=> Mode_iram,
                        carrega_BS,   --=> carrega_BS,
                        armazena_BS,--    => --armazena_BS,
                        reset_BS --    => reset_BS
 );

-- 15 Registrador Aux_IRAM_Address
Aux_IRAM_Address_Scan: reg8_bs
 port map (
        Data_in      => aux_Iram_address_scan_data,
        Data_out     => aux_Iram_addess_scan_dt_out,
        Scan_in      => TDI_aux_iram_address,
        Scan_out     => TDI_aux1_mux3_scan,
                        clock,    -- =>       clock,
                        Shift_bs,    -- =>  Shift_DR,
                        Mode,         --=> Mode_iram,
                        carrega_BS,   --=> carrega_BS,
                        armazena_BS,--    => --armazena_BS,
                        reset_BS --    => reset_BS
 );

-- 16 Registrador Aux1_MUX3
Aux1_MUX3_Scan: reg8_bs
 port map (
        Data_in      => aux1_mux3_scan_data,
        Data_out     => aux1_mux3_scan_data_out,
        Scan_in      => TDI_aux1_mux3_scan,
        Scan_out     => TDI_aux2_mux3_scan,
                        clock,    -- =>       clock,
                        Shift_bs,    -- =>  Shift_DR,
                        Mode,         --=> Mode_iram,
                        carrega_BS,   --=> carrega_BS,
                        armazena_BS,--    => --armazena_BS,
                        reset_BS --    => reset_BS
 );

-- 17 Registrador Aux2_MUX3
Aux2_MUX3_Scan: reg8_bs
 port map (
        Data_in      => aux2_mux3_scan_data,
        Data_out     => aux2_mux3_scan_data_out,
        Scan_in      => TDI_aux2_mux3_scan,
        Scan_out     => TDI_iram_out_scan,
                        clock,    -- =>       clock,
                        Shift_bs,    -- =>  Shift_DR,
                        Mode,         --=> Mode_iram,
                        carrega_BS,   --=> carrega_BS,
                        armazena_BS,--    => --armazena_BS,
                        reset_BS --    => reset_BS
```

```
 );

-- 18 Registrador IRAM_Out_Reg
iram_out_scan: reg8_bs
 port map (
       Data_in      => iram_out,
       Data_out     => iram_out_reg,
       Scan_in      => TDI_iram_out_scan,
       Scan_out     => TDI_iram_address_scan,
                       clock,    -- =>       clock,
                       Shift_bs,    -- =>  Shift_DR,
                       Mode,         --=> Mode_iram,
                       carrega_BS,   --=> carrega_BS,
                       armazena_BS,--    => --armazena_BS,
                       reset_BS --     => reset_BS
 );


-- 19 Registrador IRAM_Address
IRAM_Address_Scan: reg8_int
 port map (
       Data_in      => iram_address,
       Data_out     => iram_address_BS,
       Scan_in      => TDI_iram_address_scan,
       Scan_out     => TDI_port2_scan,
                       clock,    -- =>       clock,
                       Shift_bs,    -- =>  Shift_DR,
                       Mode,         --=> Mode_iram,
                       carrega_BS,   --=> carrega_BS,
                       armazena_BS,--    => --armazena_BS,
                       reset_BS --     => reset_BS
 );

-- 20 Registrador Port2_Out
Port2_out_Scan: reg8_int
 port map (
       Data_in      => port2_out,
       Data_out     => port2_out_BS,
       Scan_in      => TDI_port2_scan,
       Scan_out     => TDO,
                       clock,    -- =>       clock,
                       Shift_bs,    -- =>  Shift_DR,
                       Mode,         --=> Mode_iram,
                       carrega_BS,   --=> carrega_BS,
                       armazena_BS,--    => --armazena_BS,
                       reset_BS --     => reset_BS
 );

-- Unidade Lógica e Aritmética
ULA: ula8
 port map(
          ula1_BS,
          ula2_BS,
          microinst(6 downto 0),
          carry_flag_in,
```

```
        ula_out,
        carry_flag_out
       );

-- Mux da entrada 1 da ula
  with microinst(8 downto 7) select
    mux_1 <= ACC_BS        when "00",
           iram_out_reg  when "01",--Alterado de iram_out para o teste --_reg
           eprom_out  when "10",
           PC_out_BS     when others;

-- Mux da entrada 2 da ula
  with microinst(10 downto 9) select
    mux_2 <= aux_mux2_BS when "00",
           iram_out_reg   when "01", --Alterado de iram_out para o teste --_reg
           eprom_out  when "10",
           aux2_mux2_BS when others;

-- Sinal de controle do gerador de endereço de programa

process (estado_atual)
           begin
           case estado_atual is
                    when est1 =>
                             PC_inc <= '1'; -- Incremento do PC após o opcode
                    when est4 =>
                             PC_inc <= microinst(11); -- Incremento do PC após o dado imediato
      when est7 =>
                             PC_inc <= microinst(12); -- Incremento do PC em instruções de 3 bytes
      when others =>
                             PC_inc <= '0';
                    end case;
end process;

-- Mux para seleção do endereço da RAM
  with microinst(14 downto 13) select
    mux_3 <= aux1_mux3_scan_data_out when "00", -- R0, R1, ... R7
           iram_out_reg            when "01", -- iram addr, alterador de iram_out para o teste
           eprom_out               when "10", -- #data, iram addr
           aux2_mux3_scan_data_out when others; -- @R0, @R1

-- Mux do sinal de escrita no registrador PC
  with IR_BS(7 downto 4) select
    PC_wr <= microinst(17)                 when "1000", -- Desvio incondicional
           (microinst(17) and zero_flag)  when "0110", -- Desvio condicional
           (microinst(17) and carry_flag_BS) when "0100", -- Desvio condicional
           '0'                            when others;

-- Mux do sinal de flag para a entrada ula
  with IR_BS(7 downto 4) select
    carry_flag_in <= carry_flag_BS when "0011", -- Instrução ADDC
                  '1'       when "0000", -- Instrução INC
                  '1'       when "1001", -- Instrução SUBB
                  '0'       when others;
```

```
-- Mux para seleção da entrada da RAM
  with eprom_data_BS select
    iram_in <= P1_in          when "10010000", -- Porta 1 (iram addr=90h)
--             Port0_in       when "10000000", -- Porta 0 (iram addr=80h) Desabilitada
               "0000" & P3_in when "10110000", -- Porta 3 (iram addr=B0h)
               ula_out        when others;    -- Ula

-- Mux para geração do flag de zero
  with ACC_BS select
    zero_flag <= '1' when "00000000",
                 '0' when others;

-- Armazenamento no acumulador
  process(estado_atual)
    begin
        case estado_atual is
          when est6 =>
            if microinst(16)='1' then
              acc <= iram_in;
            elsif microinst(18)='1' then   -- Instrução MOVX A, @R
              acc <= P0_in;
            end if;
          when others  =>
            acc <= acc_BS;
        end case;
  end process;

-- Armazenamento no registrador da entrada 1 da ula
  process(estado_atual)
    begin
        case estado_atual is
          when est4 =>
            ula_1 <= mux_1;
          when others  =>
            ula_1 <= ula1_BS;
        end case;
  end process;

-- Armazenamento no registrador da entrada 2 da ula
  process(estado_atual)
    begin
        case estado_atual is
          when est4 =>
            ula_2 <= mux_2;
          when others  =>
            ula_2 <= ula2_BS;
        end case;
  end process;

-- Armazenamento do endereço da memória de dados
  process(estado_atual)
    begin
        case estado_atual is
          when est2 =>
            iram_address <= aux_Iram_addess_scan_dt_out; -- Endereçamento indireto: @R
```

```
        when est3 =>
          iram_address <= mux_3;
        when others  =>
          iram_address <= iram_address_BS;
      end case;
  end process;

-- Armazenamento do flag de carry
  process(estado_atual)
    begin
      case estado_atual is
        when est6 =>
          carry_flag <= carry_flag_out;
        when others  =>
          carry_flag <= carry_flag_BS;
      end case;
  end process;

-- Sinal de escrita na RAM interna
  process(estado_atual)
    begin
      case estado_atual is
        when est6 =>
          iram_wr <= microinst(15);
        when others =>
          iram_wr <= Aux_iram_wr_BS;
      end case;
  end process;




-- Sinal de escrita no registrador PC
-- Mux de PC_Wr
process(estado_atual)
          begin
                  case estado_atual is
                          when est5 =>
                                  PC_write <= PC_wr;
                          when others         =>
                    PC_write <= '0';
                    end case;
end process;
-- Armazenamento do segundo operando da eprom
  process(estado_atual)
    begin
      case estado_atual is
        when est4 =>
          if microinst(11)='1' then   -- Armazena o 2º byte de instruções de 2 bytes
            eprom_data <= eprom_out;
          else
            eprom_data <= aux_eprom_data_BS;
          end if;
```

```
        when others  =>
          eprom_data <= eprom_data_BS;
      end case;
  end process;

-- Armazenamento da porta de saída 2
  process(estado_atual)
    begin
      case estado_atual is
        when est6 =>
          if (microinst(15)='1' and eprom_data_BS="10100000") then
            Port2_out <= ula_out;  -- Porta 2 (iram addr=A0h)
          end if;
        when others  =>
          Port2_out <= Port2_out_BS;
      end case;
  end process;




-- Armazenamento da porta de saída 3
  process(clock)
    begin
     if (clock='0' and not clock'stable) then
        case estado_atual is
        when est4 =>
          if (microinst(18) = '1') then  -- Instrução MOVX A, @R
            Port3_out <= "0100";  -- sinal de leitura na memória externa
          elsif (microinst(19) = '1')  then  -- Instrução MOVX @R, A
            Port3_out <= "1000";  -- sinal de escrita na memória externa
          end if;
        when est6 =>
          if (microinst(15)='1' and eprom_data_BS="10110000") then
            Port3_out <= ula_out(3 downto 0);
          end if;
        when est7 =>
          if (microinst(19) = '1' or microinst(18) = '1') then
            Port3_out <= "0000";
          end if;
        when others  =>
          Port3_out <= Port3_out;
      end case;
     end if;
  end process;




-- Mapeamento de sinais internos para externos
P0_out        <= ACC_BS;         -- Fornece um dado para periférico
P0_add        <= mux_1;       -- Fornece a parte baixa de um endereço de 16 std_logics
P2_out        <= Port2_out_BS;   -- Fornece a parte alta de um endereço de 16 std_logics
```

```
P3_out       <= Port3_out;   -- Gera sinais de leitura e escrita nas operações MOVX
end A_soc_8051;
```

Soc_8051_BS.vhd

```
--------------------------------------------------------------------------------
-- Dissertação de Mestrado 2001/02                                            --
-- Microcontrolador 8051 com teste IEEE1149.1                                 --
-- Versão para System-on-Chip 8051                                            --
-- Por Eduardo Santos Back em maio de 2001                                    --
-- UFRGS                                                                      --
--                                                                            --
--------------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_1164.all;

entity soc_8051_bs is
  port(
       clk      : in std_logic;                    -- Sinal de sincronismo externo
       reset      : in std_logic;                   -- Sinal de inicialização
       P0_in      : in std_logic_vector(7 downto 0);   -- Porta de entrada 0 (dado)
       P0_add     : out std_logic_vector(7 downto 0);  -- Porta de saída 0 (endereço)
       P0_out     : out std_logic_vector(7 downto 0);  -- Porta de saída 0 (dado)
       P1_in      : in std_logic_vector(7 downto 0);   -- Porta de entrada 1
       P2_out     : out std_logic_vector(7 downto 0);  -- Porta de saída 2
       P3_out     : out std_logic_vector(3 downto 0);  -- Porta de saída 3
       P3_in      : in std_logic_vector(3 downto 0);   -- Porta de entrada 3
----------Sinais necessários à TAP----------
       TDI                          : in std_logic;
       TDO                          : out std_logic;
       TMS                          : in std_logic;
       TRST       : in std_logic;
                    modo        :out std_logic;
                    armazena    :out std_logic;
       erro_eprom       : out std_logic;    -- indicacao de erro na memoria de programa
       erro_decod       : out std_logic;    -- indicacao de erro na memoria de decodificacao
                    erro_RAM_ext     : out std_logic;
       TCK                                      : in std_logic
  );

end soc_8051_bs;

architecture A_soc_8051_bs of soc_8051_bs is
```

```
-- Sinais das células BS
signal Data_in      : std_logic;
signal Shift_BS             : std_logic;
signal Mode_BS              : std_logic;
signal carrega_BS           : std_logic;
signal armazena_BS          : std_logic;
signal reset_BS             : std_logic;
signal Scan_out_1           : std_logic;
signal Scan_out_2           : std_logic;
signal Scan_out_3           : std_logic;
signal Scan_out_4           : std_logic;
signal Scan_out_5           : std_logic;
signal Scan_out_6           : std_logic;
signal Scan_out_7           : std_logic;
signal Scan_out_8           : std_logic;
signal Scan_out_9           : std_logic;
signal Scan_out_10          : std_logic;
signal Scan_out_11          : std_logic;
signal Scan_out_12          : std_logic;
signal Scan_out_13          : std_logic;
signal Scan_out_14          : std_logic;
signal Scan_out_15          : std_logic;
signal Scan_out_16          : std_logic;
signal Scan_out_17          : std_logic;
signal Scan_out_18          : std_logic;
signal Scan_out_19          : std_logic;
signal Scan_out_20          : std_logic;
signal Scan_out_21          : std_logic;
signal Scan_out_22          : std_logic;
signal Scan_out_23          : std_logic;
signal Scan_out_24          : std_logic;
signal Scan_out_25          : std_logic;
signal Scan_out_26          : std_logic;
signal Scan_out_27          : std_logic;
signal Scan_out_28          : std_logic;
signal Scan_out_29          : std_logic;
signal Scan_out_30          : std_logic;
signal Scan_out_31          : std_logic;
signal Scan_out_32          : std_logic;
signal Scan_out_33          : std_logic;
signal Scan_out_34          : std_logic;
signal Scan_out_35          : std_logic;
signal Scan_out_36          : std_logic;
signal Scan_out_37          : std_logic;
signal Scan_out_38          : std_logic;
signal Scan_out_39          : std_logic;
signal Scan_out_40          : std_logic;
signal Scan_out_41          : std_logic;
signal Scan_out_42          : std_logic;
signal Scan_out_43          : std_logic;
signal Scan_out_44          : std_logic;
signal Scan_out_45          : std_logic;
signal Scan_out_46          : std_logic;
signal Scan_out_47          : std_logic;
-- Sinais do soc_8051
```

```
signal P0_IN_8051          : std_logic_vector(7 downto 0);
signal P1_IN_8051          : std_logic_vector(7 downto 0);
signal P3_IN_8051          : std_logic_vector(3 downto 0);
signal P0_ADD_8051         : std_logic_vector(7 downto 0);
signal P0_OUT_8051         : std_logic_vector(7 downto 0);
signal P2_OUT_8051         : std_logic_vector(7 downto 0);
signal P3_OUT_8051         : std_logic_vector(3 downto 0);
signal TDI_8051                    : std_logic;
signal TDO_8051                              : std_logic;
signal carrega_bs_8051     : std_logic;
signal clock_bs_8051       : std_logic;
signal mode_bs_8051                : std_logic;
signal shift_bs_8051       : std_logic;
-- Sinais do TAP
signal Shift_BS_tap                : std_logic;
signal Mode_BS_tap                 : std_logic;
signal Clock_BS_tap                : std_logic;
signal Carrega_BS_tap      : std_logic;
signal Armazena_BS_tap     : std_logic;
signal Reset_test_cells    : std_logic;
signal     reset_bs_tap    : std_logic;
signal TDI_BS_tap                  : std_logic;
signal TDO_BS_tap                  : std_logic;
-- Sinais do TAP cadeia interna
signal Shift_int_tap       : std_logic;
signal Mode_int_tap                : std_logic;
signal Carrega_int_tap     : std_logic;
signal armazena_int_tap : std_logic;
signal TDI_int_tap                 : std_logic;
signal TDO_int_tap                 : std_logic;
-- Sinais da cadeia de BIST
signal Shift_BIST_tap      : std_logic;
signal Mode_BIST_tap       : std_logic;
signal carrega_BIST_tap : std_logic;
signal armazena_BIST_tap: std_logic;
signal TDI_BIST_tap                : std_logic;
signal TDO_BIST_tap                : std_logic;

-------------------------------------------------------------------------------------
------------------------------ Componentes externos à descrição --------------------------
-------------------------------------------------------------------------------------
-------------Controlador TAP--------------------

component tap
  port(
       TCK       : in std_logic;              -- Sinal de sincronismo externo
       TRST      : in std_logic;              -- Sinal de reset assíncrono (opcional)
       TDI       : in std_logic;             -- Porta de entrada de dados de teste
       TDO       : out std_logic;             -- Porta de saída de dados de teste
       TMS       : in std_logic;              -- Porta de entrada de seleção do teste
       --Sinais de saída para as células BS
       Shift_BS  : out std_logic;                                    -- Saída para Shift_DR

       Mode_BS      : out std_logic;                                 -- Saída para Mode
       clock_bs  : out std_logic;                                    -- Saída para clock_bs das células BS
```

```
                carrega_BS          : out std_logic;                    -- Habilita o primeiro flip-flop
                armazena_BS         : out std_logic;                    -- Habilita o segundo flip-flop
                reset_test_cells  : out std_logic;                  -- Coloca as saídas dos flip-flops em
zero
                TDI_BS              : out std_logic;                    -- TDI para células BS (OUT)
                TDO_BS                      : in std_logic;                         -- TDO das células BS
(IN)
    --Sinais de saída para as células BS da cadeia de teste interna
    Shift_int       : out std_logic;                    -- Saída para Shift_DR

    Mode_int      : out std_logic;                      -- Saída para Mode
        carrega_int           : out std_logic;                      -- Habilita o primeiro flip-flop
                armazena_int  : out std_logic;
                TDI_int             : out std_logic;                    -- TDI para células BS (OUT)
                TDO_int                     : in std_logic;                         -- TDO das células BS
(IN)
    --Sinais de saída para a cadeia de teste BIST das memórias
    Shift_BIST    : out std_logic;                      -- Saída para Shift_DR

    Mode_BIST  : out std_logic;                         -- Saída para Mode
                carrega_BIST  : out std_logic;                      -- Habilita o primeiro flip-flop
                armazena_BIST : out std_logic;                      -- Habilita o segundo flip-flop
                TDI_BIST          : out std_logic;                  -- TDI para células BS (OUT)
                TDO_BIST          : in std_logic                    -- TDO das células BS (IN)
    );
end component;

component bs_cell
   port(
     Data_in      : in std_logic;          -- Pino de entrada de dados na célula BS
     Data_out     : out std_logic;          -- Pino de saída de dados da célula BS
     Scan_in      : in std_logic;          -- Pino de entrada da cadeia BS
     Scan_out     : out std_logic;          -- Pino de saída da cadeia BS
                clock               : in std_logic;                 -- Entrada de clock_bs
                Shift_BS            : in std_logic;                 -- Seletor do MUX de entrada
da célula BS, que seleciona Data_In ou Scan_In
                Mode_BS                     : in std_logic;                     -- Seletor do MUX de
saída da célula BS; 0 -> Normal; 1 -> Em teste
                carrega_BS          : in std_logic;                 -- Habilita o primeiro flip-flop
                armazena_BS         : in std_logic;                 -- Habilita o segundo flip-flop
                reset_BS            : in std_logic                  -- Coloca as saídas dos flip-
flops em zero
    );
end component;

component soc_8051
        port (
                clock     : in std_logic;             -- Sinal de sincronismo externo
    reset     : in std_logic;             -- Sinal de inicialização
    P0_in     : in std_logic_vector(7 downto 0);   -- Porta de entrada 0 (dado)
    P0_add    : out std_logic_vector(7 downto 0);  -- Porta de saída 0 (endereço)
    P0_out    : out std_logic_vector(7 downto 0);  -- Porta de saída 0 (dado)
    P1_in     : in std_logic_vector(7 downto 0);   -- Porta de entrada 1
    P2_out    : out std_logic_vector(7 downto 0);  -- Porta de saída 2
    P3_out    : out std_logic_vector(3 downto 0);  -- Porta de saída 3
```

```
        P3_in      : in std_logic_vector(3 downto 0);   -- Porta de entrada 3
        TDI                        : in std_logic;
        TDO                        : out std_logic;
        reset_bs    : in std_logic;
        carrega_bs  : in std_logic;
                    armazena_bs       : in std_logic;
        mode                    : in std_logic;
        shift_bs       : in std_logic;
-- Sinais para a cadeia BIST
        Shift_BIST  : in std_logic;                                    -- Saída para Shift_DR

        Mode_BIST  : in std_logic;                          -- Saída para Mode
                    carrega_BIST  : in std_logic;                          -- Habilita o primeiro flip-flop
                    armazena_BIST : in std_logic;                          -- Habilita o segundo flip-flop
                    TDI_BIST           : in std_logic;                     -- TDI para células BS (OUT)
                    TDO_BIST           : out std_logic;
        erro_eprom       : out std_logic;                   -- indicacao de erro na memoria de programa
        erro_decod       : out std_logic;              -- indicacao de erro na memoria de decodificacao
                    erro_RAM_ext        : out std_logic
    );
end component;

begin

-- O caminho definido para o BS é:
-- P0_IN -> P1_IN -> P3_IN -> P0_ADD -> P0_OUT -> P2_OUT -> P3_OUT.

--  P0_in(8)

celula_bs1: bs_cell -- P0_in
          port map(
                    Data_in             =>        P0_in(7), -- do 8051_bs
        Data_out    =>          P0_in_8051(7), -- do soc_8051
        Scan_in     =>          TDI_BS_tap,
        Scan_out    =>  Scan_out_1, -- Nome com signal para referenciar à próxima BS_Cell
                    clock     =>        Clock_BS_tap,
                    Shift_BS            =>        Shift_BS_tap,
                    Mode_BS                     =>        Mode_BS_tap,
                    carrega_BS          =>        Carrega_BS_tap,
                    armazena_BS         =>        armazena_BS_tap,
                    reset_BS            =>  reset_BS_tap
          );

celula_bs2: bs_cell -- P0_in
          port map(
                    Data_in             =>        P0_in(6), -- do 8051_bs
        Data_out    =>          P0_in_8051(6), -- do soc_8051
        Scan_in     =>          Scan_out_1,
        Scan_out    =>  Scan_out_2,
                    clock     =>        Clock_BS_tap,
                    Shift_BS            =>        Shift_BS_tap,
                    Mode_BS                     =>        Mode_BS_tap,
                    carrega_BS          =>        Carrega_BS_tap,
                    armazena_BS         =>        armazena_BS_tap,
                    reset_BS            =>  reset_BS_tap
```

```
        );

celula_bs3: bs_cell -- P0_in
        port map(
                Data_in         =>      P0_in(5), -- do 8051_bs
    Data_out    =>      P0_in_8051(5), -- do soc_8051
    Scan_in     =>      Scan_out_2,
    Scan_out    =>  Scan_out_3,
                clock   =>      Clock_BS_tap,
                Shift_BS        =>      Shift_BS_tap,
                Mode_BS         =>      Mode_BS_tap,
                carrega_BS      =>      Carrega_BS_tap,
                armazena_BS     =>      armazena_BS_tap,
                reset_BS        =>  reset_BS_tap
        );

celula_bs4: bs_cell -- P0_in
        port map(
                Data_in         =>      P0_in(4), -- do 8051_bs
    Data_out    =>      P0_in_8051(4), -- do soc_8051
    Scan_in     =>      Scan_out_3,
    Scan_out    =>  Scan_out_4, -- Nome com signal para referenciar  à próxima BS_Cell
                clock   =>      Clock_BS_tap,
                Shift_BS        =>      Shift_BS_tap,
                Mode_BS         =>      Mode_BS_tap,
                carrega_BS      =>      Carrega_BS_tap,
                armazena_BS     =>      armazena_BS_tap,
                reset_BS        =>  reset_BS_tap
        );

celula_bs5: bs_cell -- P0_in
        port map(
                Data_in         =>      P0_in(3), -- do 8051_bs
    Data_out    =>      P0_in_8051(3), -- do soc_8051
    Scan_in     =>      Scan_out_4,
    Scan_out    =>  Scan_out_5, -- Nome com signal para referenciar  à próxima BS_Cell
                clock   =>      Clock_BS_tap,
                Shift_BS        =>      Shift_BS_tap,
                Mode_BS         =>      Mode_BS_tap,
                carrega_BS      =>      Carrega_BS_tap,
                armazena_BS     =>      armazena_BS_tap,
                reset_BS        =>  reset_BS_tap
        );

celula_bs6: bs_cell -- P0_in
        port map(
                Data_in         =>      P0_in(2), -- do 8051_bs
    Data_out    =>      P0_in_8051(2), -- do soc_8051
    Scan_in     =>      Scan_out_5,
    Scan_out    =>  Scan_out_6, -- Nome com signal para referenciar  à próxima BS_Cell
                clock   =>      Clock_BS_tap,
                Shift_BS        =>      Shift_BS_tap,
                Mode_BS         =>      Mode_BS_tap,
                carrega_BS      =>      Carrega_BS_tap,
                armazena_BS     =>      armazena_BS_tap,
```

```
                reset_BS        =>  reset_BS_tap
        );

celula_bs7: bs_cell -- P0_in
        port map(
                Data_in         =>      P0_in(1), -- do 8051_bs
    Data_out    =>      P0_in_8051(1), -- do soc_8051
    Scan_in     =>      Scan_out_6,
    Scan_out    =>  Scan_out_7, -- Nome com signal para referenciar  à próxima BS_Cell
                clock   =>      Clock_BS_tap,
                Shift_BS        =>      Shift_BS_tap,
                Mode_BS                 =>      Mode_BS_tap,
                carrega_BS      =>      Carrega_BS_tap,
                armazena_BS     =>      armazena_BS_tap,
                reset_BS        =>  reset_BS_tap
        );

celula_bs8: bs_cell -- P0_in
        port map(
                Data_in         =>      P0_in(0), -- do 8051_bs
    Data_out    =>      P0_in_8051(0), -- do soc_8051
    Scan_in     =>      Scan_out_7,
    Scan_out    =>  Scan_out_8, -- Nome com signal para referenciar  à próxima BS_Cell
                clock   =>      Clock_BS_tap,
                Shift_BS        =>      Shift_BS_tap,
                Mode_BS                 =>      Mode_BS_tap,
                carrega_BS      =>      Carrega_BS_tap,
                armazena_BS     =>      armazena_BS_tap,
                reset_BS        =>  reset_BS_tap
        );

--   P1_in(8)

celula_bs9: bs_cell -- P1_IN
        port map(
                Data_in         =>      P1_in(7), -- do 8051_bs
    Data_out    =>      P1_in_8051(7), -- do soc_8051
    Scan_in     =>      Scan_out_8,
    Scan_out    =>  Scan_out_9, -- Nome com signal para referenciar  à próxima BS_Cell
                clock   =>      Clock_BS_tap,
                Shift_BS        =>      Shift_BS_tap,
                Mode_BS                 =>      Mode_BS_tap,
                carrega_BS      =>      Carrega_BS_tap,
                armazena_BS     =>      armazena_BS_tap,
                reset_BS        =>  reset_BS_tap
        );

celula_bs10: bs_cell -- P1_IN
        port map(
                Data_in         =>      P1_in(6), -- do 8051_bs
    Data_out    =>      P1_in_8051(6), -- do soc_8051
    Scan_in     =>      Scan_out_9,
    Scan_out    =>  Scan_out_10, -- Nome com signal para referenciar  à próxima BS_Cell
                clock   =>      Clock_BS_tap,
                Shift_BS        =>      Shift_BS_tap,
```

```
                Mode_BS                 =>      Mode_BS_tap,
                carrega_BS      =>      Carrega_BS_tap,
                armazena_BS     =>      armazena_BS_tap,
                reset_BS        =>  reset_BS_tap
        );

celula_bs11: bs_cell -- P1_IN
        port map(
                Data_in         =>      P1_in(5), -- do 8051_bs
    Data_out    =>      P1_in_8051(5), -- do soc_8051
    Scan_in     =>      Scan_out_10,
    Scan_out    =>  Scan_out_11, -- Nome com signal para referenciar  à próxima BS_Cell
                clock     =>    Clock_BS_tap,
                Shift_BS        =>      Shift_BS_tap,
                Mode_BS                 =>      Mode_BS_tap,
                carrega_BS      =>      Carrega_BS_tap,
                armazena_BS     =>      armazena_BS_tap,
                reset_BS        =>  reset_BS_tap
        );

celula_bs12: bs_cell -- P1_IN
        port map(
                Data_in         =>      P1_in(4), -- do 8051_bs
    Data_out    =>      P1_in_8051(4), -- do soc_8051
    Scan_in     =>      Scan_out_11,
    Scan_out    =>  Scan_out_12, -- Nome com signal para referenciar  à próxima BS_Cell
                clock     =>    Clock_BS_tap,
                Shift_BS        =>      Shift_BS_tap,
                Mode_BS                 =>      Mode_BS_tap,
                carrega_BS      =>      Carrega_BS_tap,
                armazena_BS     =>      armazena_BS_tap,
                reset_BS        =>  reset_BS_tap
        );

celula_bs13: bs_cell -- P1_IN
        port map(
                Data_in         =>      P1_in(3), -- do 8051_bs
    Data_out    =>      P1_in_8051(3), -- do soc_8051
    Scan_in     =>      Scan_out_12,
    Scan_out    =>  Scan_out_13, -- Nome com signal para referenciar  à próxima BS_Cell
                clock     =>    Clock_BS_tap,
                Shift_BS        =>      Shift_BS_tap,
                Mode_BS                 =>      Mode_BS_tap,
                carrega_BS      =>      Carrega_BS_tap,
                armazena_BS     =>      armazena_BS_tap,
                reset_BS        =>  reset_BS_tap
        );

celula_bs14: bs_cell -- P1_IN
        port map(
                Data_in         =>      P1_in(2), -- do 8051_bs
    Data_out    =>      P1_in_8051(2), -- do soc_8051
    Scan_in     =>      Scan_out_13,
    Scan_out    =>  Scan_out_14, -- Nome com signal para referenciar  à próxima BS_Cell
                clock     =>    Clock_BS_tap,
```

```
                Shift_BS        =>      Shift_BS_tap,
                Mode_BS                 =>      Mode_BS_tap,
                carrega_BS      =>      Carrega_BS_tap,
                armazena_BS     =>      armazena_BS_tap,
                reset_BS        =>  reset_BS_tap
        );

celula_bs15: bs_cell -- P1_IN
        port map(
                Data_in         =>      P1_in(1), -- do 8051_bs
    Data_out    =>      P1_in_8051(1), -- do soc_8051
    Scan_in     =>      Scan_out_14,
    Scan_out    =>  Scan_out_15, -- Nome com signal para referenciar  à próxima BS_Cell
                clock   =>      Clock_BS_tap,
                Shift_BS        =>      Shift_BS_tap,
                Mode_BS                 =>      Mode_BS_tap,
                carrega_BS      =>      Carrega_BS_tap,
                armazena_BS     =>      armazena_BS_tap,
                reset_BS        =>  reset_BS_tap
        );

celula_bs16: bs_cell -- P1_IN
        port map(
                Data_in         =>      P1_in(0), -- do 8051_bs
    Data_out    =>      P1_in_8051(0), -- do soc_8051
    Scan_in     =>      Scan_out_15,
    Scan_out    =>  Scan_out_16, -- Nome com signal para referenciar  à próxima BS_Cell
                clock   =>      Clock_BS_tap,
                Shift_BS        =>      Shift_BS_tap,
                Mode_BS                 =>      Mode_BS_tap,
                carrega_BS      =>      Carrega_BS_tap,
                armazena_BS     =>      armazena_BS_tap,
                reset_BS        =>  reset_BS_tap
        );

-- P3_in (4)
celula_bs17: bs_cell -- P3_IN
        port map(
                Data_in         =>      P3_in(3), -- do 8051_bs
    Data_out    =>      P3_in_8051(3), -- do soc_8051
    Scan_in     =>      Scan_out_16,
    Scan_out    =>  Scan_out_17, -- Nome com signal para referenciar  à próxima BS_Cell
                clock   =>      Clock_BS_tap,
                Shift_BS        =>      Shift_BS_tap,
                Mode_BS                 =>      Mode_BS_tap,
                carrega_BS      =>      Carrega_BS_tap,
                armazena_BS     =>      armazena_BS_tap,
                reset_BS        =>  reset_BS_tap
        );

celula_bs18: bs_cell -- P3_IN
        port map(
                Data_in         =>      P3_in(2), -- do 8051_bs
    Data_out    =>      P3_in_8051(2), -- do soc_8051
    Scan_in     =>      Scan_out_17,
```

```
        Scan_out    =>  Scan_out_18, -- Nome com signal para referenciar  à próxima BS_Cell
                        clock     =>        Clock_BS_tap,
                        Shift_BS          =>        Shift_BS_tap,
                        Mode_BS                   =>        Mode_BS_tap,
                        carrega_BS        =>        Carrega_BS_tap,
                        armazena_BS       =>        armazena_BS_tap,
                        reset_BS          =>  reset_BS_tap
            );

celula_bs19: bs_cell -- P3_IN
            port map(
                        Data_in           =>        P3_in(1), -- do 8051_bs
        Data_out    =>        P3_in_8051(1), -- do soc_8051
        Scan_in     =>        Scan_out_18,
        Scan_out    =>  Scan_out_19, -- Nome com signal para referenciar  à próxima BS_Cell
                        clock     =>        Clock_BS_tap,
                        Shift_BS          =>        Shift_BS_tap,
                        Mode_BS                   =>        Mode_BS_tap,
                        carrega_BS        =>        Carrega_BS_tap,
                        armazena_BS       =>        armazena_BS_tap,
                        reset_BS          =>  reset_BS_tap
            );

celula_bs20: bs_cell -- P3_IN
            port map(
                        Data_in           =>        P3_in(0), -- do 8051_bs
        Data_out    =>        P3_in_8051(0), -- do soc_8051
        Scan_in     =>        Scan_out_19,
        Scan_out    =>  Scan_out_20, -- Nome com signal para referenciar  à próxima BS_Cell
                        clock     =>        Clock_BS_tap,
                        Shift_BS          =>        Shift_BS_tap,
                        Mode_BS                   =>        Mode_BS_tap,
                        carrega_BS        =>        Carrega_BS_tap,
                        armazena_BS       =>        armazena_BS_tap,
                        reset_BS          =>  reset_BS_tap
            );

--   P0_ADD(8)

celula_bs21: bs_cell -- P0_ADD
            port map(
                        Data_in           =>        P0_ADD_8051(7),
        Data_out    =>        P0_ADD(7), -- do soc_8051
        Scan_in     =>        Scan_out_20,
        Scan_out    =>  Scan_out_21, -- Nome com signal para referenciar  à próxima BS_Cell
                        clock     =>        Clock_BS_tap,
                        Shift_BS          =>        Shift_BS_tap,
                        Mode_BS                   =>        Mode_BS_tap,
                        carrega_BS        =>        Carrega_BS_tap,
                        armazena_BS       =>        armazena_BS_tap,
                        reset_BS          =>  reset_BS_tap
            );

celula_bs22: bs_cell -- P0_ADD
            port map(
```

```
                Data_in         =>      P0_ADD_8051(6),
        Data_out    =>      P0_ADD(6), -- do soc_8051
        Scan_in     =>      Scan_out_21,
        Scan_out    =>  Scan_out_22,
                clock   =>      Clock_BS_tap,
                Shift_BS        =>      Shift_BS_tap,
                Mode_BS                 =>      Mode_BS_tap,
                carrega_BS      =>      Carrega_BS_tap,
                armazena_BS     =>      armazena_BS_tap,
                reset_BS        =>  reset_BS_tap
            );

celula_bs23: bs_cell -- P0_ADD
            port map(
                Data_in         =>      P0_ADD_8051(5),
        Data_out    =>      P0_ADD(5), -- do soc_8051
        Scan_in     =>      Scan_out_22,
        Scan_out    =>  Scan_out_23,
                clock   =>      Clock_BS_tap,
                Shift_BS        =>      Shift_BS_tap,
                Mode_BS                 =>      Mode_BS_tap,
                carrega_BS      =>      Carrega_BS_tap,
                armazena_BS     =>      armazena_BS_tap,
                reset_BS        =>  reset_BS_tap
            );

celula_bs24: bs_cell -- P0_ADD
            port map(
                Data_in         =>      P0_ADD_8051(4),
        Data_out    =>      P0_ADD(4), -- do soc_8051
        Scan_in     =>      Scan_out_23,
        Scan_out    =>  Scan_out_24, -- Nome com signal para referenciar  à próxima BS_Cell
                clock   =>      Clock_BS_tap,
                Shift_BS        =>      Shift_BS_tap,
                Mode_BS                 =>      Mode_BS_tap,
                carrega_BS      =>      Carrega_BS_tap,
                armazena_BS     =>      armazena_BS_tap,
                reset_BS        =>  reset_BS_tap
            );

celula_bs25: bs_cell -- P0_ADD
            port map(
                Data_in         =>      P0_ADD_8051(3),
        Data_out    =>      P0_ADD(3), -- do soc_8051
        Scan_in     =>      Scan_out_24,
        Scan_out    =>  Scan_out_25, -- Nome com signal para referenciar  à próxima BS_Cell
                clock   =>      Clock_BS_tap,
                Shift_BS        =>      Shift_BS_tap,
                Mode_BS                 =>      Mode_BS_tap,
                carrega_BS      =>      Carrega_BS_tap,
                armazena_BS     =>      armazena_BS_tap,
                reset_BS        =>  reset_BS_tap
            );

celula_bs26: bs_cell -- P0_ADD
```

```
        port map(
                Data_in         =>      P0_ADD_8051(2), -- do 8051_bs
    Data_out    =>      P0_ADD(2), -- do soc_8051
    Scan_in     =>      Scan_out_25,
    Scan_out    =>  Scan_out_26, -- Nome com signal para referenciar  à próxima BS_Cell
                clock   =>      Clock_BS_tap,
                Shift_BS        =>      Shift_BS_tap,
                Mode_BS                 =>      Mode_BS_tap,
                carrega_BS      =>      Carrega_BS_tap,
                armazena_BS     =>      armazena_BS_tap,
                reset_BS        =>  reset_BS_tap
        );

celula_bs27: bs_cell -- P0_ADD
        port map(
                Data_in         =>      P0_ADD_8051(1), -- do 8051_bs
    Data_out    =>      P0_ADD(1), -- do soc_8051
    Scan_in     =>      Scan_out_26,
    Scan_out    =>  Scan_out_27, -- Nome com signal para referenciar  à próxima BS_Cell
                clock   =>      Clock_BS_tap,
                Shift_BS        =>      Shift_BS_tap,
                Mode_BS                 =>      Mode_BS_tap,
                carrega_BS      =>      Carrega_BS_tap,
                armazena_BS     =>      armazena_BS_tap,
                reset_BS        =>  reset_BS_tap
        );

celula_bs28: bs_cell -- P0_ADD
        port map(
                Data_in         =>      P0_ADD_8051(0), -- do 8051_bs
    Data_out    =>      P0_ADD(0), -- do soc_8051
    Scan_in     =>      Scan_out_27,
    Scan_out    =>  Scan_out_28, -- Nome com signal para referenciar  à próxima BS_Cell
                clock   =>      Clock_BS_tap,
                Shift_BS        =>      Shift_BS_tap,
                Mode_BS                 =>      Mode_BS_tap,
                carrega_BS      =>      Carrega_BS_tap,
                armazena_BS     =>      armazena_BS_tap,
                reset_BS        =>  reset_BS_tap
        );

--      P0_OUT(8)

celula_bs29: bs_cell -- P0_OUT
        port map(
                Data_in         =>      P0_OUT_8051(7), -- do 8051_bs
    Data_out    =>      P0_OUT(7), -- do soc_8051
    Scan_in     =>      Scan_out_28,
    Scan_out    =>  Scan_out_29, -- Nome com signal para referenciar  à próxima BS_Cell
                clock   =>      Clock_BS_tap,
                Shift_BS        =>      Shift_BS_tap,
                Mode_BS                 =>      Mode_BS_tap,
                carrega_BS      =>      Carrega_BS_tap,
                armazena_BS     =>      armazena_BS_tap,
                reset_BS        =>  reset_BS_tap
```

```
        );

celula_bs30: bs_cell -- P0_OUT
        port map(
                Data_in         =>      P0_OUT_8051(6), -- do 8051_bs
    Data_out    =>      P0_OUT(6), -- do soc_8051
    Scan_in     =>      Scan_out_29,
    Scan_out    =>  Scan_out_30, -- Nome com signal para referenciar  à próxima BS_Cell
                clock   =>      Clock_BS_tap,
                Shift_BS        =>      Shift_BS_tap,
                Mode_BS         =>      Mode_BS_tap,
                carrega_BS      =>      Carrega_BS_tap,
                armazena_BS     =>      armazena_BS_tap,
                reset_BS        =>  reset_BS_tap
        );

celula_bs31: bs_cell -- P0_OUT
        port map(
                Data_in         =>      P0_OUT_8051(5), -- do 8051_bs
    Data_out    =>      P0_OUT(5), -- do soc_8051
    Scan_in     =>      Scan_out_30,
    Scan_out    =>  Scan_out_31, -- Nome com signal para referenciar  à próxima BS_Cell
                clock   =>      Clock_BS_tap,
                Shift_BS        =>      Shift_BS_tap,
                Mode_BS         =>      Mode_BS_tap,
                carrega_BS      =>      Carrega_BS_tap,
                armazena_BS     =>      armazena_BS_tap,
                reset_BS        =>  reset_BS_tap
        );

celula_bs32: bs_cell -- P0_OUT
        port map(
                Data_in         =>      P0_OUT_8051(4), -- do 8051_bs
    Data_out    =>      P0_OUT(4), -- do soc_8051
    Scan_in     =>      Scan_out_31,
    Scan_out    =>  Scan_out_32, -- Nome com signal para referenciar  à próxima BS_Cell
                clock   =>      Clock_BS_tap,
                Shift_BS        =>      Shift_BS_tap,
                Mode_BS         =>      Mode_BS_tap,
                carrega_BS      =>      Carrega_BS_tap,
                armazena_BS     =>      armazena_BS_tap,
                reset_BS        =>  reset_BS_tap
        );

celula_bs33: bs_cell -- P0_OUT
        port map(
                Data_in         =>      P0_OUT_8051(3), -- do 8051_bs
    Data_out    =>      P0_OUT(3), -- do soc_8051
    Scan_in     =>      Scan_out_32,
    Scan_out    =>  Scan_out_33, -- Nome com signal para referenciar  à próxima BS_Cell
                clock   =>      Clock_BS_tap,
                Shift_BS        =>      Shift_BS_tap,
                Mode_BS         =>      Mode_BS_tap,
                carrega_BS      =>      Carrega_BS_tap,
                armazena_BS     =>      armazena_BS_tap,
```

```
                reset_BS        =>  reset_BS_tap
        );

celula_bs34: bs_cell -- P0_OUT
        port map(
                Data_in         =>      P0_OUT_8051(2), -- do 8051_bs
    Data_out    =>      P0_OUT(2), -- do soc_8051
    Scan_in     =>      Scan_out_33,
    Scan_out    =>  Scan_out_34, -- Nome com signal para referenciar  à próxima BS_Cell
                clock   =>      Clock_BS_tap,
                Shift_BS        =>      Shift_BS_tap,
                Mode_BS                 =>      Mode_BS_tap,
                carrega_BS      =>      Carrega_BS_tap,
                armazena_BS     =>      armazena_BS_tap,
                reset_BS        =>  reset_BS_tap
        );

celula_bs35: bs_cell -- P0_OUT
        port map(
                Data_in         =>      P0_OUT_8051(1), -- do 8051_bs
    Data_out    =>      P0_OUT(1), -- do soc_8051
    Scan_in     =>      Scan_out_34,
    Scan_out    =>  Scan_out_35, -- Nome com signal para referenciar  à próxima BS_Cell
                clock   =>      Clock_BS_tap,
                Shift_BS        =>      Shift_BS_tap,
                Mode_BS                 =>      Mode_BS_tap,
                carrega_BS      =>      Carrega_BS_tap,
                armazena_BS     =>      armazena_BS_tap,
                reset_BS        =>  reset_BS_tap
        );

celula_bs36: bs_cell -- P0_OUT
        port map(
                Data_in         =>      P0_OUT_8051(0), -- do 8051_bs
    Data_out    =>      P0_OUT(0), -- do soc_8051
    Scan_in     =>      Scan_out_35,
    Scan_out    =>  Scan_out_36, -- Nome com signal para referenciar  à próxima BS_Cell
                clock   =>      Clock_BS_tap,
                Shift_BS        =>      Shift_BS_tap,
                Mode_BS                 =>      Mode_BS_tap,
                carrega_BS      =>      Carrega_BS_tap,
                armazena_BS     =>      armazena_BS_tap,
                reset_BS        =>  reset_BS_tap
        );

--  P2_OUT(8)

celula_bs37: bs_cell -- P2_OUT
        port map(
                Data_in         =>      P2_OUT_8051(7), -- do 8051_bs
    Data_out    =>      P2_OUT(7), -- do soc_8051
    Scan_in     =>      Scan_out_36,
    Scan_out    =>  Scan_out_37, -- Nome com signal para referenciar  à próxima BS_Cell
                clock   =>      Clock_BS_tap,
                Shift_BS        =>      Shift_BS_tap,
```

```
                Mode_BS                 =>      Mode_BS_tap,
                carrega_BS      =>      Carrega_BS_tap,
                armazena_BS     =>      armazena_BS_tap,
                reset_BS        =>  reset_BS_tap
        );

celula_bs38: bs_cell -- P2_OUT
        port map(
                Data_in         =>      P2_OUT_8051(6), -- do 8051_bs
    Data_out    =>      P2_OUT(6), -- do soc_8051
    Scan_in     =>      Scan_out_37,
    Scan_out    =>  Scan_out_38, -- Nome com signal para referenciar  à próxima BS_Cell
                clock   =>      Clock_BS_tap,
                Shift_BS        =>      Shift_BS_tap,
                Mode_BS                 =>      Mode_BS_tap,
                carrega_BS      =>      Carrega_BS_tap,
                armazena_BS     =>      armazena_BS_tap,
                reset_BS        =>  reset_BS_tap
        );

celula_bs39: bs_cell -- P2_OUT
        port map(
                Data_in         =>      P2_OUT_8051(5),
    Data_out    =>      P2_OUT(5), -- do soc_8051
    Scan_in     =>      Scan_out_38,
    Scan_out    =>  Scan_out_39, -- Nome com signal para referenciar  à próxima BS_Cell
                clock   =>      Clock_BS_tap,
                Shift_BS        =>      Shift_BS_tap,
                Mode_BS                 =>      Mode_BS_tap,
                carrega_BS      =>      Carrega_BS_tap,
                armazena_BS     =>      armazena_BS_tap,
                reset_BS        =>  reset_BS_tap
        );

celula_bs40: bs_cell -- P2_OUT
        port map(
                Data_in         =>      P2_OUT_8051(4),
    Data_out    =>      P2_OUT(4), -- do soc_8051
    Scan_in     =>      Scan_out_39,
    Scan_out    =>  Scan_out_40, -- Nome com signal para referenciar  à próxima BS_Cell
                clock   =>      Clock_BS_tap,
                Shift_BS        =>      Shift_BS_tap,
                Mode_BS                 =>      Mode_BS_tap,
                carrega_BS      =>      Carrega_BS_tap,
                armazena_BS     =>      armazena_BS_tap,
                reset_BS        =>  reset_BS_tap
        );

celula_bs41: bs_cell -- P2_OUT
        port map(
                Data_in         =>      P2_OUT_8051(3),
    Data_out    =>      P2_OUT(3), -- do soc_8051
    Scan_in     =>      Scan_out_40,
    Scan_out    =>  Scan_out_41, -- Nome com signal para referenciar  à próxima BS_Cell
                clock   =>      Clock_BS_tap,
```

```
                Shift_BS        =>      Shift_BS_tap,
                Mode_BS                 =>      Mode_BS_tap,
                carrega_BS      =>      Carrega_BS_tap,
                armazena_BS     =>      armazena_BS_tap,
                reset_BS        =>  reset_BS_tap
        );

celula_bs42: bs_cell -- P2_OUT
        port map(
                Data_in         =>      P2_OUT_8051(2),
    Data_out    =>      P2_OUT(2), -- do soc_8051
    Scan_in     =>      Scan_out_41,
    Scan_out    =>  Scan_out_42, -- Nome com signal para referenciar  à próxima BS_Cell
                clock     =>    Clock_BS_tap,
                Shift_BS        =>      Shift_BS_tap,
                Mode_BS                 =>      Mode_BS_tap,
                carrega_BS      =>      Carrega_BS_tap,
                armazena_BS     =>      armazena_BS_tap,
                reset_BS        =>  reset_BS_tap
        );

celula_bs43: bs_cell -- P2_OUT
        port map(
                Data_in         =>      P2_OUT_8051(1),
    Data_out    =>      P2_OUT(1), -- do soc_8051
    Scan_in     =>      Scan_out_42,
    Scan_out    =>  Scan_out_43, -- Nome com signal para referenciar  à próxima BS_Cell
                clock     =>    Clock_BS_tap,
                Shift_BS        =>      Shift_BS_tap,
                Mode_BS                 =>      Mode_BS_tap,
                carrega_BS      =>      Carrega_BS_tap,
                armazena_BS     =>      armazena_BS_tap,
                reset_BS        =>  reset_BS_tap
        );

celula_bs44: bs_cell -- P2_OUT
        port map(
                Data_in         =>      P2_OUT_8051(0),
    Data_out    =>      P2_OUT(0), -- do soc_8051
    Scan_in     =>      Scan_out_43,
    Scan_out    =>  Scan_out_44, -- Nome com signal para referenciar  à próxima BS_Cell
                clock     =>    Clock_BS_tap,
                Shift_BS        =>      Shift_BS_tap,
                Mode_BS                 =>      Mode_BS_tap,
                carrega_BS      =>      Carrega_BS_tap,
                armazena_BS     =>      armazena_BS_tap,
                reset_BS        =>  reset_BS_tap
        );

--          P3_OUT(4)

celula_bs45: bs_cell -- P3_OUT
        port map(
                Data_in         =>      P3_OUT_8051(3),
    Data_out    =>      P3_OUT(3), -- do soc_8051
```

```
        Scan_in    =>          Scan_out_44,
        Scan_out   =>  Scan_out_45, -- Nome com signal para referenciar  à próxima BS_Cell
                       clock     =>        Clock_BS_tap,
                       Shift_BS          =>        Shift_BS_tap,
                       Mode_BS                   =>        Mode_BS_tap,
                       carrega_BS        =>        Carrega_BS_tap,
                       armazena_BS       =>        armazena_BS_tap,
                       reset_BS          =>  reset_BS_tap
            );

celula_bs46: bs_cell -- P3_OUT
            port map(
                       Data_in           =>        P3_OUT_8051(2),
        Data_out   =>          P3_OUT(2), -- do soc_8051
        Scan_in    =>          Scan_out_45,
        Scan_out   =>  Scan_out_46, -- Nome com signal para referenciar  à próxima BS_Cell
                       clock     =>        Clock_BS_tap,
                       Shift_BS          =>        Shift_BS_tap,
                       Mode_BS                   =>        Mode_BS_tap,
                       carrega_BS        =>        Carrega_BS_tap,
                       armazena_BS       =>        armazena_BS_tap,
                       reset_BS          =>  reset_BS_tap
            );

celula_bs47: bs_cell -- P3_OUT
            port map(
                       Data_in           =>        P3_OUT_8051(1),
        Data_out   =>          P3_OUT(1), -- do soc_8051
        Scan_in    =>          Scan_out_46,
        Scan_out   =>  Scan_out_47, -- Nome com signal para referenciar  à próxima BS_Cell
                       clock     =>        Clock_BS_tap,
                       Shift_BS          =>        Shift_BS_tap,
                       Mode_BS                   =>        Mode_BS_tap,
                       carrega_BS        =>        Carrega_BS_tap,
                       armazena_BS       =>        armazena_BS_tap,
                       reset_BS          =>  reset_BS_tap
            );

celula_bs48: bs_cell -- P3_OUT
            port map(
                       Data_in           =>        P3_OUT_8051(0),
        Data_out   =>          P3_OUT(0), -- do soc_8051
        Scan_in    =>          Scan_out_47,
        Scan_out   =>  TDO_BS_tap,
                       clock     =>        Clock_BS_tap,
                       Shift_BS          =>        Shift_BS_tap,
                       Mode_BS                   =>        Mode_BS_tap,
                       carrega_BS        =>        Carrega_BS_tap,
                       armazena_BS       =>        armazena_BS_tap,
                       reset_BS          =>  reset_BS_tap
            );


  -- pino_inside = nome do pino dentro do 8051
  -- pino_outside = nome do pino fora do 8051. Este deve ser declarado
```

```
  -- como um signal no arquivo que chama o 8051.
  -- pino_inside => pino_outside

  -- pino_in = entrada do chip que chama o 8051 e que corresponde
  -- a pino_outside e pino_inside

tap_port: tap
          port map(
                    TCK,
        TRST,
        TDI,
        TDO,
        TMS,
        Shift_BS    =>      Shift_BS_tap,
        Mode_BS             =>      Mode_BS_tap,
        clock_bs    =>      Clock_BS_tap,
                    carrega_BS      =>      Carrega_BS_tap,
                    armazena_BS     =>      armazena_BS_tap,
                    reset_test_cells    =>      reset_BS_tap,
                    TDI_BS          =>      TDI_BS_tap,
                    TDO_BS                  =>      TDO_BS_tap,
        Shift_int   =>      Shift_int_tap,
        Mode_int    =>      Mode_int_tap,
                    carrega_int     =>      Carrega_int_tap,
                    armazena_int =>armazena_int_tap,
                    TDI_int         =>      TDI_int_tap,
                    TDO_int                 =>      TDO_int_tap,
        Shift_BIST  =>      Shift_BIST_tap,
        Mode_BIST=>         Mode_BIST_tap,
                    carrega_BIST    =>      Carrega_BIST_tap,
                    armazena_BIST   =>      armazena_BIST_tap,
                    TDI_BIST        =>      TDI_BIST_tap,
                    TDO_BIST        =>      TDO_BIST_tap
          );

processador: soc_8051
          port map(
          clock     =>      clk,
        reset,
        P0_in       => P0_in_8051,
        P0_ADD      => P0_ADD_8051,
        P0_out      => p0_out_8051,
        P1_in       => p1_in_8051,
        P2_out      => p2_out_8051,
        P3_out      => p3_out_8051,
        P3_in       => p3_in_8051,
        -- Sinais para a cadeia interna do soc_8051
                    TDI             => TDI_int_tap,
        TDO                 => TDO_int_tap,
        reset_bs    =>      reset_bs_tap,
        carrega_bs  =>      carrega_int_tap,
                    armazena_bs     =>      armazena_int_tap,
        mode        =>      mode_int_tap,
        shift_bs    =>      shift_int_tap,
                    -- Sinais para a cadeia BIST
```

```
        Shift_BIST  =>      shift_bist_tap,
        Mode_BIST=>         mode_bist_tap,
                    carrega_BIST    =>      carrega_BIST_tap,
                    armazena_BIST   =>      armazena_BIST_tap,
                    TDI_BIST                =>      TDI_BIST_tap,
                    TDO_BIST                =>      TDO_BIST_tap,
        erro_eprom,
        erro_decod,
                    erro_RAM_ext
    );

modo<=mode_int_tap;
armazena<=armazena_int_tap;
end A_soc_8051_bs;
```

Glossário

ASIC	Application Specific Integrated Circuit.
Bed-of-Nails	Técnica de teste baseada no uso de ponteiras que são colocadas sobre as trilhas com a finalidade de injetar vetores e extrair resultados de testes.
BIST	Built In Self Test - Autoteste integrado da lógica central de um componente.
Boundary-Scan	Estruturas internas em um circuito integrado, que formam um caminho serial na sua periferia com a finalidade de produzir testes estruturais.
BS	Boundary Scan
CI	Circuito Integrado
Core	Núcleo de hardware.
DFT	Design for Testability.
Extest	Instrução para realizar um teste externo.
FPGA	Field Programmable Gate Array, componente de lógica programável.
Idcode	Código de identificação.
Ident	Instrução utilizada para identificação do componente.
IEEE	Institute of Electrical and Electronics Engineers.
IEEE std. 1149.1	Norma que regulamenta o padrão de teste boundary-scan como um padrão internacional.
Intest	Instrução para realizar um teste interno.
IP	Internet Protocol.
IR	Nome dado ao registrador de instruções.
JEDEC	Joint Electronic Devices Engineering Council Formato padrão para a transferência de dados entre sistemas de preparação e dispositivos lógico-programáveis.
JETAG	Joint European Test Action Group; grupo de companhias européias de sistemas eletrônicos que se reuniram com a finalidade de desenvolver estruturas de testes funcionais. Este grupo criou o Boundary-Scan.
JTAG	Joint Test Action Group; nome dado à união do grupo JETAG com companhias norte-americanas de sistemas eletrônicos. Este grupo tornou o boundary-scan um padrão internacional.
Stuck-at	Nome dado quando um elemento está fixado em um determinado valor lógico.
TAP	Test Access Port, ou porta de acesso e controle da máquina de estados de teste compatível com a norma IEEE 1149.1
TCK	Pino de clock dedicado ao teste.
TDI	Test Data In - Nome dado ao pino de entrada de dados de teste.
TDO	Test Data Out. Nome dado ao pino de saída de dados de teste.

TMS	Test Mode Select. Nome dado ao pino que seleciona o modo de teste.
TRST	Test Reset ou pino de reset assíncrono.
VHDL	Very High Speed Integrated Circuit Hardware Description Language - Linguagem utilizada para programar dispositivos do tipo FPGA.

Referências

[CAR96] CARRO, L.; PEREIRA, G.; SUZIM, A. Prototyping and Reengineering of Microcontroller-based Systems. In: IEEE INTERNATIONAL WORKSHOP ON RAPID SYSTEMS PROTOTYPING, 7., 1996. **Shortening the path from specification to prototype**. Los Alamitos: IEEE Computer Society, 1996. p.178 - 182.

[CAR97] CARRO, L. et al. Ambiente ágata de projeto : versão beta 2.0. In: WORKSHOP IBERCHIP, 3., 1997, México. **[Anais]**. México: Departamento de Ingineria Eléctrica, 1997. p. 494 - 503.

[CAS2000] CASSOL, L. **Relatório do Estágio Supervisionado.** Porto Alegre: Escola de Engenharia – Departamento de Engenharia Elétrica, UFRGS, 2000. 45p.

[COT99a] COTA, E. et al. Implementing a Self-Testing 8051 Microprocessor. In: SYNMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, 12., 1999, Natal. **Proceedings...** Los Alamitos: IEEE Computer Society, 1999. p.202 - 205.

[COT99b] COTA, E.; KRUG, M.; LUBASZEWSKI, M. Teste e Projeto Visando Teste de Circuitos e Sistemas Integrados. In: ESCOLA DE MICROELETRÔNICA da SBC-SUL, EMICRO, 1., 1999. **Livro texto**. Porto Alegre: Instituto de Informática, 1999. p.181 - 200.

[FRA2000] FRANCO, D. Estudo de Caso de um System-on-Chip: sistema para tradução automática de palavras impressas. In: SEMANA ACADÊMICA do PPGC, 4., 1999, Porto Alegre. **Anais...** Porto Alegre: PPGC da UFRGS, 1999.

[IEE90] IEEE. **IEEE Standard Test Access Port and Boundary Scan Architecture**, IEEE Standard 1149.1. New York,1990.

[JAR89] JARWALA, N.; YAU, C. A New Framework for Analyzing Test Generation and Diagnosis Algorithms for Wiring Interconnects. In: IEEE INTERNATIONAL TEST CONFERENCE. **Proceedings…** [S.l.:s.n.], 1989.

[JON91] JONG, F. et al. Boundary Scan Test, Test Methodology and Fault Modeling. **Journal of Electronic Testing: Theory and Applications**, [S.l.], v. 2, p. 77 - 88, 1991.

[JTA95] JTAG TECHNOLOGIES. **Boundary-Scan Test.** Netherlands, 1995.

[KAP99] KAPUR, Rohit et al. P1500-CTL: Towards a Standard Core Test Language. In: VLSI TEST SYMPOSIUM, 17., 1999, Dana Point. **Proceedings...** Los Alamitos: IEEE Computer Society, 1999. p. 489 - 490.

[LAL97] LALA, P. G. **Digital Circuit Testing and Testability**. [S.l.]Academic Press, 1997.

[LIM2000] LIMA, F.G. et al. Designing a radiation hardened 8051-like micro-controller. In: SYMPOSIUM ON INTEGRATEDCIRCUITS AND SYSTEMS DESIGN, 13., 2000, Manaus. **Proceedings...** Los Alamitos: IEEE Computer Society, 2000. p.255 - 60.

[LUB92] LUBASZEWSKI, M.; COURTOIS, B. On The Design of Self-Checking Boundary Scannable Boards. In: INTERNATIONAL TEST CONFERENCE, 1992. **Proceedings...** [S.l.:s.n.], 1992. p. 372 - 381.

[MAU90] MAUNDER, C. M.; TULOSS, R. **The Test Access Port and Boundary-scan Architecture**. Los Alamitos: IEEE Computer Society Press, 1990.

[SIL97] SILVA, L.G. P.S. et al. Synthesis of the FPGA version of 8051. INTERNAL MICROELECTRONICS GROUP SEMINAR, SIM, 1997, Porto Alegre. **Proceedings…** [S.l.:s.n.], 1997. p. 115 - 120.

[TUL89] TUL, R. E.; YAU, C. W. BIST & Boundary Scan for Board Level Test: Test Program Pseudocode. In: EUROPEAN TEST CONFERENCE. **Proceedings…** [S.l.:s.n.], 1989. p. 106.

[ZOR98] ZORIAN, Y. et al. Testing Embedded-Core Based System Chips. In: INTERNATIONAL TEST CONFERENCE. **Proceedings…** [S.l.:s.n.], 1998. p. 130 - 143.

[ZOR99] ZORIAN, Y. **Testing Semiconductor Chips:** Trends and Solutions. [S.l.:s.n.]: IEEE, 1999.

Obras Consultadas

[BAC2000] BACK, E. S. **Tecnologias Industriais de Teste**. 2000. Trabalho Individual (Mestrado em Ciência da Computação) - Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[CAR2000] CARRO, L. et al. Using Reconfigurability to Break Down Test Costs: a case study. In: IEEE LATIN AMERICAN TEST WORKSHOP, 1., 2000, Rio de Janeiro. **Digest of papers**. [Amissville: IEEE Computer Society], 2000. p.209 – 214.

[DER92] DERVISOGLU, B. IEEE 1149.2 Description and Status Report. **IEEE Design & Test of Computers**, Los Alamitos, v. 9, p.79 - 81, Sept. 1992.

[FRA2000] FRANCO, D. T. **Estudo de Caso de um *System-on-Chip* para Validação de um Modelo de Sistemas**. 2000. Dissertação (Mestrado em Ciência da Computação) - Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[GRO2000] HALES, A.; MARINISSEN, E. J. **IEEE P1500 Web Site**. Disponível em: < http://grouper.ieee.org/groups/1500/ >. Acesso em: 17 mar. 2000.

[KAU74] KAUTZ, W. Testing for Faults in Wiring Networks. **IEEE Transactions on Computers**, New York, v. C-23, n. 4, p. 358-636, Apr. 1974.

[LUB95] LUBASZEWSKI, M. Boundary Scan: The Pioneer of Test Standards. In: CONGRESS OF THE BRAZILIAN MICROELECTRONICS SOCIETY, SBMICRO, 10., 1995, Canela. **Proceedings...** Porto Alegre: Instituto de Informática da UFRGS, 1995. p. 145-166.

[MAL88] MALY, W. ; NIGH, P. Built-In Current Testing – Feasibily Study, In: INTERNATIONAL CONFERENCEON COMPUTER-AIDED DESIGN, 1988, Santa Clara. **Digest of technical papers**. New York: IEEE, 1988. p.340-343.

[MCH92] MCHUGH, P. IEEE 1149.5 Module Test and Maintenance Bus. **IEEE Design & Test of Computers**, Los Alamitos, v. 9, p. 62 - 65, Dec. 1992.

[REN98] RENOVELL, M. SRAM-based FPGAs: a structural test approach. In: BRAZILIAN SYMPOSIUM ON INTEGRATED CIRCUIT DESIGN, SBCCI, 11., 1998, Rio de Janeiro. **Proceedings...** Los Alamitos: IEEE Computer Society, 1998. p. 67-72.

[VSI96] VSI ALLIANCE. **Virtual Socket Interface Architectural Document**, Nov. 1996.