IGOR GADELHA PEREIRA

ESTRATÉGIAS DE TESTE APLICADAS À REDE DE INTERCONEXÃO DE FPGAS

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica - PPGEE, da Universidade Federal da Paraíba - UFPB, como requisito para a obtenção do título de Mestre em Engenharia Elétrica.

Orientador: Prof. Dr. Cleonilson Protásio de Souza

JOÃO PESSOA

2014

IGOR GADELHA PEREIRA

ESTRATÉGIAS DE TESTE APLICADAS À REDE DE INTERCONEXÃO DE FPGAS

Dissertação a presentada ao Programa de Pós-Graduação em Engenharia Elétrica - PPGEE, da Universidade Federal da Paraíba - UFPB, como requisito para a obtenção do título de Mestre em Engenharia Elétrica.

Orientador: Prof. Dr. Cleonilson Protásio de Souza

JOÃO PESSOA

2014

FICHA CATALOGRÁFICA

Pereira, Igor

ESTRATÉGIAS DE TESTE APLICADAS À REDE DE INTERCONEXÃO DE FPGAS– João Pessoa, 2014.

Nº de páginas: 78

Área de concentração: Teste de FPGAs.

Orientador: Prof. Dr. Cleonilson Protásio de Souza.

Dissertação (Mestrado) – UFPB/PPGEE.

1.Teste de circuito integrado; 2. BIST; 3. FPGAs

UNIVERSIDADE FEDERAL DA PARAÍBA – UFPB
CENTRO DE ENERGIAS ALTERNATIVAS E RENOVÁVEIS – CEAR
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA - PPGEE

A Comissão Examinadora, abaixo assinada, aprova a Dissertação

ESTRATÉGIAS DE TESTE APLICADAS À REDE DE INTERCONEXÃO DE FPGAS

Elaborada por

IGOR GADELHA PEREIRA

como requisito para obtenção do grau de
Mestre em Engenharia Elétrica.

COMISSÃO EXAMINADORA

PROF. DR. CLEONILSON PROTÁSIO DE SOUZA, (ORIENTADOR) PPGEE/UFPB

PROF. DR. ANTONIO AUGUSTO LISBOA DE SOUZA, PPGEE/UFPB

PROF. DR. JOSE ANTÔNIO GOMES DE LIMA, PPGI/UFPB

João Pessoa/PB, ____ de _____ de _____.

Aos meus pais, Eridon Celso Pereira e Carmem Lucia Moreira Gadelha, a minha irmã Iane Gadelha Pereira, ao futuro, e ao meu orientador Cleonilson Protásio de Souza.

Dedico

AGRADECIMENTOS

Aos meus pais, que foram verdadeiros mestres na arte da educação e consciência.

Ao professor Cleonilson Protásio de Souza pela orientação e empenho para com a pesquisa.

Aos técnicos dos laboratórios de elétrica, em especial Elder, e a estrutura disponibilizada pela UFPB.

Aos colegas de laboratório pela hora que dividimos juntos em especial Bruno Arruda, Verônica Lima e Marcos Bernardo. À vida, pois sem ela, nada seria possível.

À CAPES pela concessão da Bolsa de mestrado.

“A leitura após certa idade distrai excessivamente o espírito humano das suas reflexões criadoras. Todo o homem que lê demais e usa o cérebro de menos adquire a preguiça de pensar.”

Albert Einstein.

SUMÁRIO

LISTA DE ILUSTRAÇÕES VI
LISTA DE TABELAS VIII
ABSTRACT X
1 INTRODUÇÃO 12
1.1 OBJETIVOS ..... **ERRO! INDICADOR NÃO DEFINIDO.**
2 FUNDAMENTAÇÃO TEÓRICA 17
2.1 BIST (*BUILT IN SELF-TEST*) 17
2.2 MODELO DE FALHAS (*Stuck-At*) 20
2.3 MODELO DE FALHAS EM PONTE (*BRIDGE FAULTS*) 22
2.4 ARQUITETURA DO FPGA XILINX SPARTAN 3E 23
2.5 ESTRUTURA DA REDE DE INTERCONEXÃO DO FPGA XILINX SPARTAN 3 26
2.6 REVISÃO DE LITERATURA: ESTRATÉGIAS DE TESTES DE INTERCONEXÕES DE FPGAS 30
3 ESTRATÉGIA PROPOSTA 50
4 MATERIAL E MÉTODO 57
5 RESULTADOS E DISCUSSÃO 68
6 CONCLUSÕES 75
REFERÊNCIAS 77

LISTA DE ILUSTRAÇÕES

FIGURA 1 - EVOLUÇÃO DOS CIRCUITOS INTEGRADOS [1].....12
FIGURA 2 – EXEMPLOS DE DEFEITOS NA FABRICAÇÃO DE UM CHIP 17
FIGURA 3 - PROCESSO DE TESTE REALIZADO POR UM ATE [7].....18
FIGURA 4 - PROCESSO DE TESTE BIST [7].....18
FIGURA 5 - BIST PARA REDE DE INTERCONEXÃO.....19
FIGURA 6 - CARACTERIZAÇÃO DE FALHAS *STUCK AT* [7].....21
FIGURA 7 - *BRIDGE FAULTS* [4].....22
FIGURA 8 - FAMÍLIA SPARTAN 3E [12] ..... **ERRO! INDICADOR NÃO DEFINIDO.**
FIGURA 9 - ARQUITETURA FPGA SPARTAN 3E 24
FIGURA 10 - VISÃO DE UM CLB [12].....25
FIGURA 11 - VISÃO GERAL DE UM *SLICE* [12].....26
FIGURA 12 - ARQUITETURA DE UM PIP.....27
FIGURA 13 - ARQUITETURA DA REDE DE INTERCONEXÃO [13] 27
FIGURA 14 - ARQUITETURA DE UM MULTIPLEXADOR PIP [13].....28
FIGURA 15 - LINHAS QUE COMPÕE A REDE DE INTERCONEXÃO GLOBAL [14] 29
FIGURA 16 - ARQUITETURA DE TESTE DESENVOLVIDA POR STROUD & ABRAMOVICI, 1998 [15].....30
FIGURA 17 – ESTRATÉGIA DE TESTE RSTARS PARA TESTE *ONLINE* [15].....31
FIGURA 18 - ESTRATÉGIA DE TESTE RSTARS PARA TESTE *OFFLINE* [15].....31
FIGURA 19 - ESTRATÉGIAS PARA DIAGNÓSTICO DO LOCAL DA FALHA [15] 32
FIGURA 20 - EMULAÇÃO DE FALHA EM PONTE [16].....35
FIGURA 21 - RESULTADOS DE TESTE ∆IDDQ [16] 35
FIGURA 22 - TESTE BASEADO EM CONTADOR DE DOIS BITS [4].....36
FIGURA 23 - TESTE BASEADO EM UM CONTADOR DUPLO DE DOIS BITS [4] 37
FIGURA 24 - RESULTADOS DE SIMULÇÕES DE ESTRATÉGIAS BASEADAS EM CONTADORES [4].....37
FIGURA 25 - ESTRATÉGIA BASEADA EM PARIDADE [4] 38
FIGURA 26 - ESTRATÉGIA BASEADA EM PARIDADE MODIFICADA [4] 38
FIGURA 27 - ESTRATÉGIA DE TESTE BASEADA EM PARIDADE INVERTIDA [4] 39
FIGURA 28 - RESULTADOS DE SIMULÇÕES DE ESTRATÉGIAS BASEADAS EM PARIDADE [4].....40
FIGURA 29 - LFSR DE QUATRO BITS COM REALIMENTAÇÃO INTERNA [4] 41
FIGURA 30 - LFSR DE QUATRO BITS COM REALIMENTAÇÃO EXTERNA [4].....41
FIGURA 31 - RESULTADO DE SIMULAÇÕES DE ESTRATÉGIAS BASEADAS EM LFSR [4] 42
FIGURA 32 - REGISTRADOR CAR DE 8 BITS COM CONDIÇÕES DE FRONTEIRAS CÍCLICAS [4].....43
FIGURA 33 - REGISTRADOR CAR DE MAIOR COMPRIMENTO (*MAXIMAL LENGTH*) [4].....44
FIGURA 34 - RESULTADO DE SIMULAÇÕES DE ESTRATÉGIAS BASEADAS EM CAR [4].....44
FIGURA 35 - ESTRATÉGIA DE IMPLEMENTAÇÃO EM LINHAS *HEX LINES* [18].....45
FIGURA 36 - ARQUITETURA DE FIOS *DOUBLE LINES* COM DISCRIMINAÇÃO DE FIOS [18].....46
FIGURA 37- LUT CONFIGURADO COMO REGISTRADOR DE DESLOCAMENTO (SRL16) [12] ...50
FIGURA 38 - ARQUITETURA DE UM TPG UTILIZANDO SRL16.....51
FIGURA 39 - ESTRATÉGIA BIST BASEADA EM SRL16 51
FIGURA 40 - DIAGNÓSTICO DO LOCAL DA FALHA.....52
FIGURA 41 - ESTRATÉGIA PROPOSTA COM VALORES INICIAIS E FUNÇÕES LÓGICAS.....54

FIGURA 42 - RESULTADOS DE SIMULAÇÃO COM *SOFTWARE* AUSIM.....55
FIGURA 43 - SIMULAÇÃO COM O *SOFTWARE ISIM*.....57
FIGURA 44- ARQUIVO DE DESCRIÇÃO EM XDL [20].....58
FIGURA 45 - ARQUIVO DE CONFIGURAÇÃO EM XDL.....59
FIGURA 46 - CAPTURA DE TELA DO *SOFTWARE* FPGA EDITOR.....60
FIGURA 47 - ARQUIVO DE ALOCAÇÃO LÓGICA.....61
FIGURA 48 - PROJETO DA PLACA PARA TESTE DE FPGAS.....64
FIGURA 49 - PROCESSO DE TESTE BIST PARA FPGAS *XILINX*.....65
FIGURA 50 - VISUALIZAÇÃO DO ARQUIVO *.DATA*.....65
FIGURA 51 - ROTEAMENTO DOS WUTS CONSIDERANDO OS LIMITES DA MATRIZ DE CHAVEAMENTO.....68
FIGURA 52 - CONEXÕES END PARA BEG EM UM NÃO-CLB.....69
FIGURA 53 - REALIMENTAÇÃO INICIAL PARA O PRIMEIRO E SEGUNDO CLB DA LINHA.....70
FIGURA 54 - RESULTADO DA SIMULAÇÃO UTILIZANDO VALORES DA COMBINAÇÃO 1.....72
FIGURA 55 - RESULTADO DA SIMULAÇÃO UTILIZANDO VALORES DA COMBINAÇÃO 2.....72
FIGURA 56 - RESULTADO DA SIMULAÇÃO UTILIZANDO VALORES DA COMBINAÇÃO 3.....72
FIGURA 57 - RESULTADO DA SIMULAÇÃO UTILIZANDO VALORES DA COMBINAÇÃO 4.....73

LISTA DE TABELAS

TABELA 1 - CONFIGURAÇÕES DE TESTE UTILIZANDO RSTARS [15].....33
TABELA 2 - REGRAS PARA REGISTRADORES CAR [4].....43
TABELA 3 - DIAGNOSTICO DO LOCAL DA FALHA ATRAVÉS DE LEITURA DOS ORAS [18].....46
TABELA 4 - CONFIGURAÇÕES DE TESTE NECESSÁRIAS PARA TESTAR O FPGA VIRTEX4 [18].....47
TABELA 5 - ANALISE DA POSIÇÃO DA FALHA A PARTIR DOS RESULTADOS DOS ORAS.....52
TABELA 6 - CONFIGURAÇÕES DE TESTE NECESSÁRIAS PARA TESTAR COMPLETAMENTE O FPGA SPARTAN 3E XC3S100E.....69
TABELA 7 - ANALISE DE CONSUMO DE CORRENTE DINÂMICA PARA DIFERENTES COMBINAÇÕES DE VALORES INICIAIS DOS REGISTRADORES SRL16.....71

RESUMO

ESTRATÉGIAS DE TESTE APLICADAS À REDE DE INTERCONEXÃO DE FPGAS

Este trabalho objetiva realizar uma análise das principais estratégias de teste já existentes para FPGA, e propor uma nova estratégia aplicada à rede de interconexão do FPGA Xilinx Spartan 3E baseada em registradores de deslocamento com realimentação linear sintetizável pelo Algoritmo de Berlekamp-Massey e que possa diagnósticar com precisão o local da falha. Para isso, foram utilizados softwares da fabricante de FPGAs *Xilinx* (especificamente, XDL e FPGA_editor) para determinar precisamente a configuração do FPGA e, assim, criar uma nova proposta e avaliar sua empregabilidade. Como resultado, a partir da estratégia adotada foi possível rotear 7 WUTs (*Wires Under Test*) em um total de 8 para o FPGA em questão. Sendo assim, foram necessárias 24 configurações de teste para testar e diagnósticar todas as linhas do tipo *HexLine* e *DoubleLine*. Os resultados obtidos mostram que a estratégia proposta é capaz de testar 7 WUTs por vez e necessita de 24 configurações para testar e diagnósticar precisamente o local da falha na rede de interconexão.

Palavras-chave: Teste de FPGAs, Estratégias de teste BIST, Teste da rede de interconexão.

ABSTRACT

TEST STRATEGIES APPLIED TO FPGAS INTERCONNECT NETWORK

This work aims to carry out an analysis of the main existing testing strategies for FPGA, and propose a new strategy applied to the interconnection network of the Xilinx Spartan 3E FPGA based on linear feedback shift register synthesized by Berlekamp Massey Algorithm that can accurately localize the failure. For this, we used softwares from Xilinx manufacturer (specifically, XDL and FPGA_editor) to determine the FPGA based configuration and than create a new proposal and evaluate their employability. As a result of the proposed strategy, it was possible to route 7 WUTs (Wires Under Test) of total of 8 for the FPGA under investigation. Thus, it was necessary 24 test configurations to test and locate the failure on all hexlines and doublelines. The results show that this strategy is able to test 7 WUTs at a time and needs 24 test configurations to test and diagnose precisely the failure location.

Keywords: Test of FPGAs, FPGAs Bist Strategys, Interconnection network test.

1 INTRODUÇÃO

1 INTRODUÇÃO

Desde a invenção do primeiro transistor, a indústria eletrônica foi tomada pela constante evolução tecnológica. Em 1965, *Gordon E. Moore*, cofundador da Intel, previu que a quantidade de transistores que podem ser integrados em uma única pastilha de silício dobra aproximadamente a cada ano. A indústria de semicondutores progredio de circuitos integrados (CIs) com pouco mais de mil transistores, para dispositivos com bilhões de transistores atualmente [1], conforme pode ser visto na Figura 1.

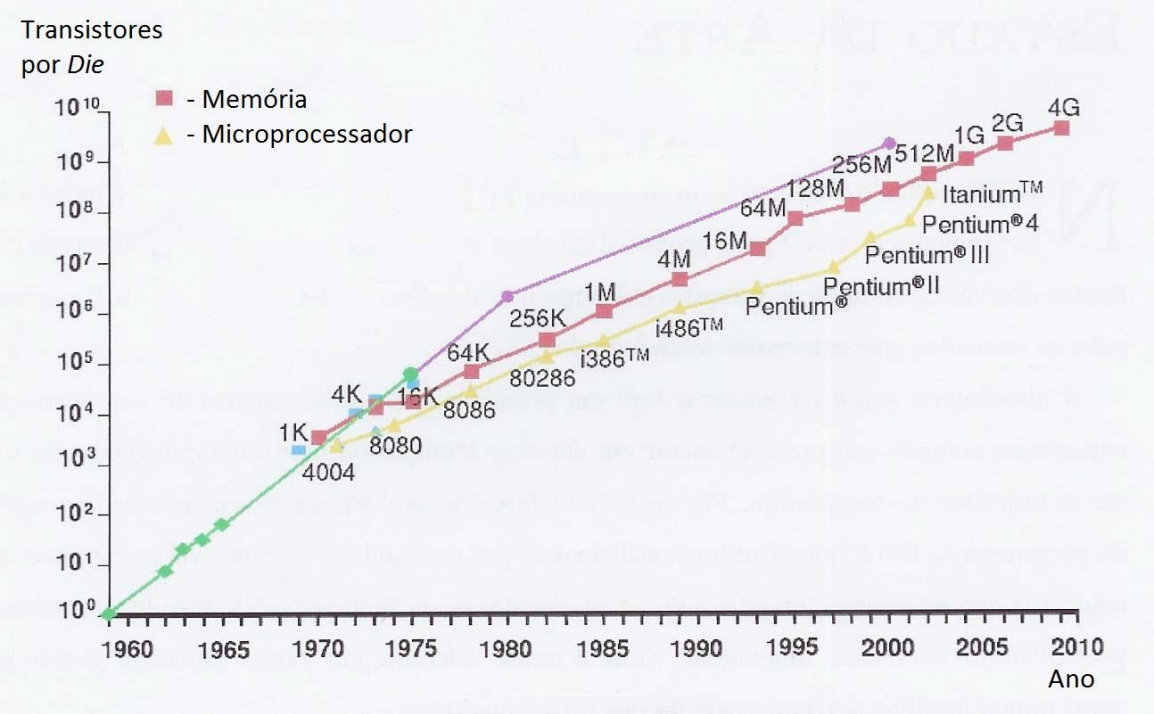


FIGURA 1 - EVOLUÇÃO DOS CIRCUITOS INTEGRADOS [1]

Devido a essa grande evolução dos CIs, houve uma desvantagem em relação a sua confiabilidade, desde que, CIs mais complexos e com transistores menores causaram novos tipos de defeitos físicos e consequentemente mais suscetibilidade de falhas no *chip* [2]. Tais CIs requerem mais tempo e complexidade de teste para atingir uma cobertura de nível de defeitos aceitável. Dessa forma, os gastos relacionados com o teste de CIs podem atingir até 50% do valor do projeto [2].

A grande imersão tecnológica dos tempos atuais nos torna cada vez mais dependentes dos sistemas eletrônicos. Nesse contexto, surge um dilema fundamental: quanto maior é o beneficio que esses sitemas oferecem ao nosso bem

estar e à nossa qualidade de vida, maior é o potencial de causarem algum dano quando esses falham na execução de suas funções ou as executam de forma incorreta [3]. Exemplos evidentes podem ser observados nas áreas como medicina e aviação. Quando esses sistemas falham, vidas e patrimônios são colocados em risco. Para evitar tais cenários, são necessários sistemas de alta confiabilidade.

Um circuito integrado bastante utilizado em sistemas de alta confiabilidade é o FPGA (da expressão em Inglês, *Field Programmable Gate Array*). O FPGA é um CI que implementa recursos lógicos programáveis em campo, tais como: portas lógicas, multiplexadores, *soft-cores,* etc. O FPGA apresenta algumas vantagens quando utilizado em sistemas de alta confiabilidade, são elas:

- Pode ser reprogramado para autoteste em campo.
- Possibilidade de ser reprogramado caso seja detectado algum erro funcional.
- Possibilidade de ser reprogramado em campo para atualizar a aplicação.

Para isso, o FPGA conta com recursos de *hardware* que aumentam a complexidade do *chip*, blocos lógicos programáveis (CLBs da sigla em Inglês – *Configurable Logic Blocks*) e rede de interconexão programável, que por sua vez chega a corresponder a 80% dos bits de configuração que serão carregados no FPGA, sendo assim mais suscetíveis a falhas. [4]

Em geral, testes de circuitos integrados são realizados utilizando-se equipamentos de teste automático ou ATE (do inglês, *Automatic Test Equipment*). Tais equipamentos são usados para aplicar vetores de teste ao circuito através de pontas de sondagem em contato direto com a pastilha de silicio, e ler a resposta enviada pelo circuito para cada novo vetor aplicado ao dispositivo, com o objetivo de identificar se o circuito contém falha ou não [5]. Esses vetores buscam excitar todos os pontos de interconexão do circuito e todas as portas lógicas para observar algum comportamento irregular que possa ser resultado de defeitos na fabricação.

O problema com o uso dos ATEs é o alto custo, tanto na aquisição do equipamento quanto aos relacionados com a operação, treinamento, manutenção e a compra e/ou desenvolvimento do *software* operacional [6]. Outro problema é que a maioria dos ATEs operam na faixa de algumas centenas de MHz, enquanto que,

atualmente, devido à evolução nos processos de fabricação, os circuitos operam na faixa de GHz [5].

A solução encontrada foi construir circuitos que já possuissem internamente, geradores de padrões de teste e analisadores de resposta, utilizando apenas um pino de saída para indicar se houve falha no teste. A essa solução foi atribuída o nome de BIST (da expressão em Inglês, *Built In Self-test*) [7].

Em geral, os testes de circuitos integrados são direcionados a um modelo de falha que irá representar um defeito físico do C.I. Um dos modelos de falha mais utilizados é o ***stuck-at***, em que por qualquer defeito físico que aconteça no circuito, este se manifestará numa conexão, podendo esta ficar fixa em “zero” (*stuck-at 0*) ou em “um” (*stuck-at 1*). Logo, deve-se excitar todas as conexões do circuito em busca de alguma conexão fixa em algum nível lógico. Outro modelo de falha bastante comum é o modelo de **falha em ponte**, em que um vazamento de corrente entre fios adjacentes em um circuito integrado pode se manifestar logicamente de três formas: **dominante**, quando um fio domina completamente o valor do outro, e **dominante AND/OR**, quando um fio domina o valor do outro em apenas um valor lógico [7].

Executar o teste em um FPGA consiste em programá-lo com uma lógica adequada para que esse contenha estruturas de teste e exercite sua rede de interconexão em busca de conexões fixas ou falhas em ponte. Ou seja, o CI que outrora seria programado para uma aplicação, agora é programado para executar um autoteste.

Em FPGAs mais modernos, há a possibilidade de programação parcial, enquanto o dispositivo está operacional. Porém, o tempo necessário para carregar a configuração no FPGA é bastante superior ao tempo utilizado para aplicar vetores de teste, logo, estratégias que maximizem a quantidade de recursos testados por configuração tendem a reduzir a quantidade de teste pela quantidade de configurações de teste [4].

Logo, novos algoritmos e estratégias de teste são um fator crítico na tentativa de aumentar a margem de lucro das empresas e a confiabilidade de sistemas eletrônicos, sendo um assunto bastante estudado em diversas pesquisas [1].

Este trabalho está dividido em tópicos como visto a seguir. 1. Introdução, 2. Fundamentação Teórica, 3. Estratégia Proposta, 4. Material e Método, 5. Resultados e Discussão, 6. Conclusão.

O primeiro tópico introduz e justifica o tema da pesquisa. No segundo tópico é apresentada uma base teórica acerca dos conceitos abordados durante a pesquisa na ordem em que foram trabalhados: Conceitos BIST, BIST para rede de interconexão bem como os modelos de falhas e vetores de teste estudados, Estrutura interna de um FPGA *XIlinx Sparan 3E* apresentando seus recursos lógicos e de interconexão programáveis. E por fim, uma revisão da literatura acerca dos principais trabalhos na área.

No terceiro tópico será apresentada uma proposta de estratégia de teste em hardware baseada em registrador de deslocamento. Em seguida, no quarto tópico, serão apresentadas as ferramentas utilizadas para criação dessa proposta. No quinto tópico, serão apresentados os resultados e algumas discussões, e por fim, a conclusão no sexto tópico.

2 FUNDAMENTAÇÃO TEÓRICA

2 FUNDAMENTAÇÃO TEÓRICA

Nesse capítulo, serão introduzidos alguns conceitos necessários para o entendimento da pesquisa.

2.1 BIST (*BUILT IN SELF-TEST*)

A fabricação de um circuito integrado é um processo delicado, e alguns requisitos são necessários para que seja feita com sucesso. Partículas de poeira e imprecisão no processo, bem como o envelhecimento dos componentes devido ao seu uso, podem provocar defeitos no *chip*, conforme visto na Figura 2, podendo influenciar em seu funcionando [7].

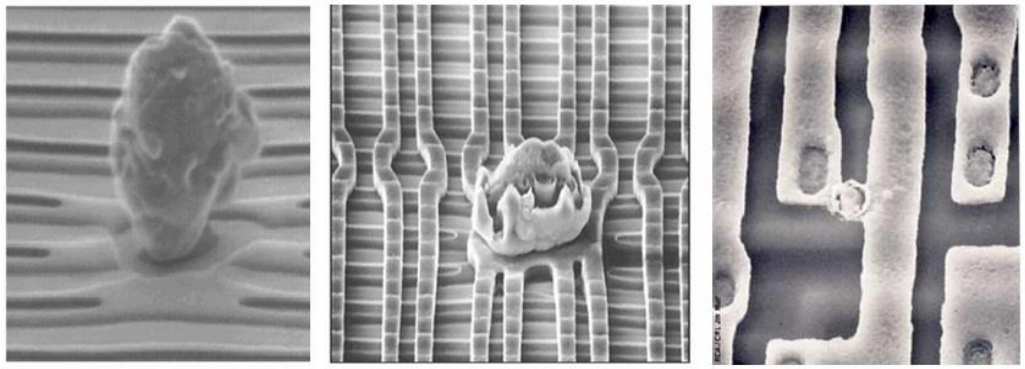


FIGURA 2 – EXEMPLOS DE DEFEITOS NA FABRICAÇÃO DE UM CHIP

Para detectar tais defeitos, é preciso aplicar vetores de teste nas entradas primárias (pinos) do circuito sob teste (CUT, da expressão em Inglês *Circuit Under Test*). Estes vetores se propagam pelo interior do *chip* a fim de transformar um defeito físico em uma falha perceptível, que por sua vez será propagada até suas saídas primárias, onde serão analisadas por um analisador de resposta produzindo uma resposta positiva ou negativa do teste, conforme visto na Figura 3.

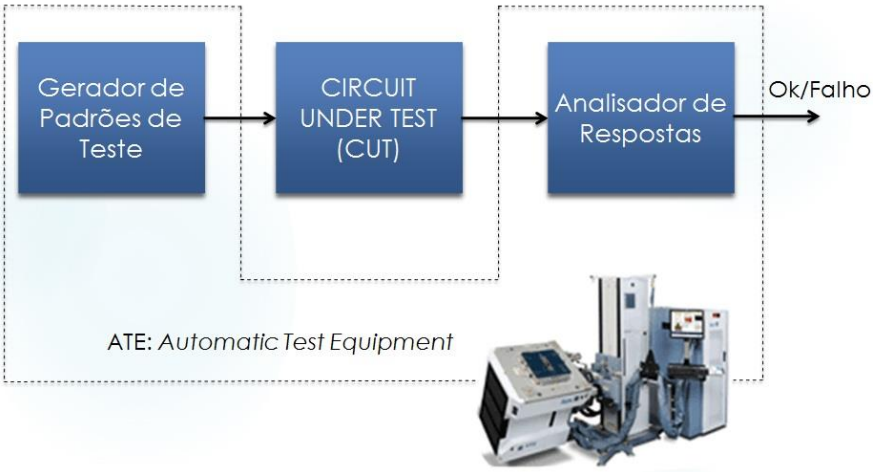


FIGURA 3 - PROCESSO DE TESTE REALIZADO POR UM ATE [7]

Para gerar tais vetores, é utilizado um gerador de padrões de teste (TPG, da expressão em Inglês *Test Pattern Generator*), e para analisar a resposta do CI é utilizado um analisador de resposta (ORA, da expressão em Inglês *Output Response Analyzer*). O equipamento que concentra o TPG e o ORA é o ATE (*Automatic Test Equipment*).

Conforme o aumento na velocidade de operação dos CIs e a alta densidade de transistores, ficou cada vez mais caro construir ATEs capazes de aplicar vetores de teste em altas velocidades. Além disso, CIs que apresentam vários estados internos requerem mais tempo de teste para controlar esses estados através dos pinos de entrada e saída [7]. Uma solução encontrada foi embutir o TGP e o ORA no CI, proporcionando uma maior cobertura de falha, bem como testes realizados na velocidade de operação do circuito. A essa técnica foi atribuído o nome BIST (*Built-in Self-Test*). Dessa forma, o CI executará seu teste e produzirá apenas uma resposta de sucesso ou falha, conforme visto na Figura 4.

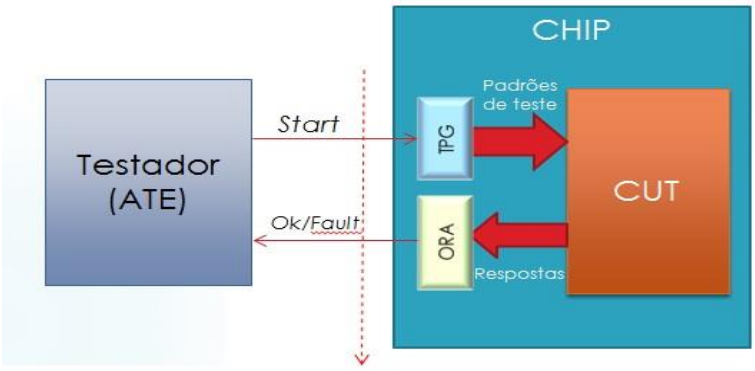


FIGURA 4 - PROCESSO DE TESTE BIST [7]

Para que seja possível detectar erros na rede de interconexão de um FPGA, deve-se utilizar uma estrutura de **TPG** para aplicar diferentes níveis de sinal nos fios sob teste (**WUTs** da expressão em Inglês, *Wires Under Test*), e por fim analisar sua resposta através de um **ORA**, que produzirá uma resposta positiva (com falha) ou negativa (sem falha) para o conjunto de WUTs analisados [4]. Uma metodologia BIST aplicado à rede de interconexão pode ser visualizada na Figura 5.

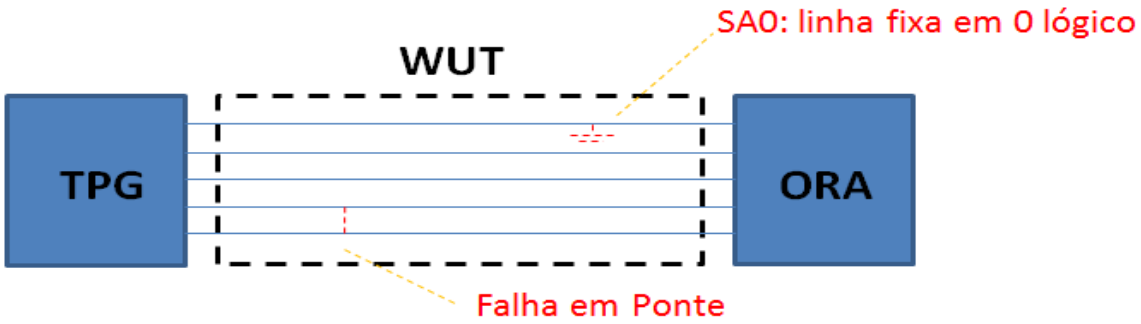


FIGURA 5 - BIST PARA REDE DE INTERCONEXÃO

Em geral, as falhas de um circuito são modeladas usando-se algum modelo de falha, que representará as falhas físicas do circuito [8]. Dois modelos são adotados nesse trabalho: *Stuck-AT* e *Bridge*. Ambos serão descritos no tópico seguinte. Modelos de falhas relacionados a tempo de propagação nos fios (do Inglês, *Delay Faults*) não serão considerados objetos de estudo nessa pesquisa.

BIST para FPGA normalmente é feito através de configurações de teste que envolve um gerador de padrões e um analisador de resposta para testar todos os seus recursos programáveis. Normalmente, as estruturas reprogramáveis a serem testadas podem ser divididas em dois grupos:

- **Grupo lógico**, responsável por implementar as funções lógicas e recursos de roteamento.
- Estruturas que compõe a **rede de interconexão programável**, que por sua vez se divide em rede de interconexão local e rede de interconexão global [9].

Este trabalho concentra-se na rede de interconexão global, logo, busca-se testar os recursos de roteamento globais que ligam os blocos lógicos programáveis entre si. Para isso, utilizar-se-á recursos lógicos para programar estruturas de teste, TPG e ORA, e estes recursos serão descritos no tópico 2.4.

Outro fator a se considerar é como os resultados dos testes serão extraídos dos ORAs. Alguns FPGAs contam com um recurso conhecido como *readback*, que permite a leitura da memória de configuração do FPGA a qualquer momento. Essa memória de configuração representa os estados dos registradores internos, onde serão armazenados os resultados dos teste. Quando este recurso não está presente, convém usar uma estrutura *scan chain*, onde os dados serão enviados de maneira serial para um pino externo do C.I. Um fator contra o *readback* é que praticamente dobra-se o tempo de teste, desde que o tempo necessário para executar o *readback* seja equivalente ao tempo que se leva para programar o FPGA, porém, este não necessita de estruturas extras utilizadas para implementar o *scan chain*, que aumentaria a quantidade de configurações de teste [4].

Testes BIST para rede de interconexão programável é um dos testes mais complexos para FPGA [4]. Essa complexidade é proporcionada pela grande quantidade de recursos que precisam ser testados, se comparados com outros tipos de teste BIST. Além disso, 80% dos bits de configuração carregados no FPGA estão relacionados aos recursos de roteamento. Dada essa complexidade, os FPGAs requerem muitas configurações de teste [9]. Por essa razão, o objetivo principal nas pesquisas da área é maximizar a quantidade de WUTs testados por configurações de teste, assim minimizando o total de configurações necessárias para testar todo o FPGA. Estes recursos de roteamento serão descritos no tópico 2.5.

Existem também dois tipos de teste de FPGA: testes *online*, em que o FPGA continua operacional, ou seja, continua realizando operações lógicas e aplica o teste em uma área restrita, e testes *off-line,* em que o FPGA utiliza todos os seus recursos programáveis para se autotestar [4].

2.2 MODELO DE FALHAS (*Stuck-At*)

O modelo abordado nesse tópico é o *Stuck-AT,* também conhecido como modelo clássico ou padrão, e é amplamente utilizado na área de teste de circuitos integrados. Nesse modelo, considera-se que por qualquer defeito físico, as falhas ocorrem sempre nas linhas de conexão entre as portas lógicas. Ou seja, considera-

se que uma dada linha esteja com falha quando essa torna-se permanentemente fixa em algum nível lógico de forma anormal. Dessa forma, tem-se dois tipos de falhas no modelo *Stuck-AT:* ***stuck-at 0***, quando a linha está fixa no nível lógico "zero", e ***stuck-at 1***, que representa uma linha fixa no nível lógico "um" [7]. (Figura 6)

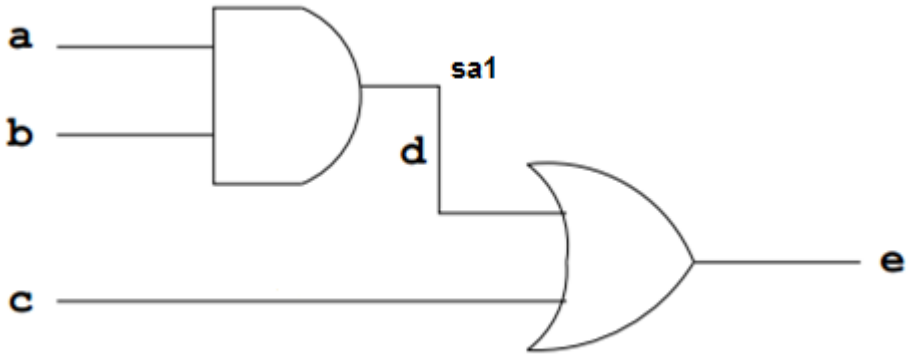


FIGURA 6 - CARACTERIZAÇÃO DE FALHAS *STUCK AT* [7]

No exemplo acima, pode-se verificar que a linha **d** apresenta uma falha do tipo *stuck-at 1*, (representada por sa1), de fato, isso não representa uma conexão entre a linha **d** e o terminal positivo da fonte de alimentação, mas talvez alguma falha física nos transistores que compõem a saída da porta *AND* e entrada da porta *OR*, que fixe essa linha em nível lógico "um".

Para que a falha seja detectada nesse circuito, é necessário aplicar vetores que detectem o problema e propaguem uma possível em resposta a isto para a saída. Por exemplo, o vetor (1, 1, 1) não detectará a falha porque a saída esperada na linha **e** sem falha (e = 1) é a mesma da saída com a falha. Para que este problema seja detectado, é necessário aplicar vetores que forcem a linha **d** para o nível lógico 0, ou seja, (0,0,0), (0,1,0),(1,0,0). Observa-se também que o valor de **c** deve ser 0 para que a falha seja observada na saída do circuito.

Logo, para cobrir a possibilidade de uma interconexão com falha *Stuck-AT* devem-se forçar ambos os valores lógicos na linha, e observar se ela não se encontra fixa em "um" ou em "zero" [7].

A seguir, será descrito o modelo de falha em ponte (*Bridge Faults*).

2.3 MODELO DE FALHAS EM PONTE (*BRIDGE FAULTS*)

Ligações em curto entre sinais normalmente não conectados são defeitos de fabricação que ocorrem em *chips* e em placas de circuito impresso. Estes curtos são modelados em forma de falha em ponte, *Bridge Faults* [10]. Nesse capítulo, será descrito mais especificamente esse tipo de falha.

Para esse modelo, existem algumas falhas lógicas comumente utilizadas, são elas: (a) Dominante (DOM), (b) Dominante-AND (DAND) e (c) Dominante-OR (DOR) [4], como pode ser visualizado na Figura 7.

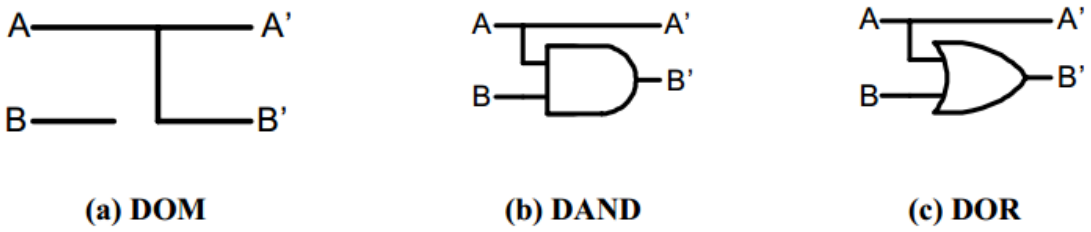


FIGURA 7 - *BRIDGE FAULTS* [4]

Uma falha DOM existe quando uma resistência baixa é aplicada entre dois fios adjacentes e a fonte mais forte domina a saída dos dois, resultando em uma falha Dominante (Figura 7a). Falhas DAND e DOR existem quando uma resistência baixa é aplicada entre dois fios adjacentes e a fonte mais forte domina o curto em apenas um valor lógico, assemelhando-se a uma porta lógica AND ou OR.

A quantidade total de possíveis falhas em ponte presentes em um FPGA cresce de forma exponencial quando são considerados, no teste, todos os fios, ou seja, são testados de forma exaustivamente dois a dois. Porém, a probabilidade desse tipo de falha acontecer é maior entre dois fios adjacentes, logo, a maioria dos testes realizados focam em WUTs adjacentes entre si [10].

Para detectar falhas do tipo *Bridge Faults*, são necessários vetores capazes de estimular essas falhas, e para isso, é necessário que se aplique valores lógicos opostos (1,0 e 0,1) e observar se os valores se mantêm [4].

2.4 ARQUITETURA DO FPGA XILINX SPARTAN 3E

O FPGA Spartan 3 da Xilinx foi lançado para o mercado em 2005 [11]. Naquela época, esse FPGA foi lançado para suprir o mercado de *chips* baratos e que pudessem ser consumidos em grande quantidade para serem utilizados em Roteadores, Modems, Televisão digital e outros, tentando tornar o FPGA comercialmente mais atraente frente aos ASICs. Os seguintes modelos compõem a família Spartan 3, são eles: Spartan 3, Spartan 3E, Spartan 3A, Spartan 3AN e Spartan 3A DSP. Neste trabalho, será utilizado um FPGA **XC3S100E** da família Spartan 3E (Figura 8), que apresenta as seguintes características:

- Quantidade de recursos lógicos e IO (do Inglês, *Input/Output*) reduzidos, minimizando a quantidade de pinos e facilitando a confecção de uma placa de teste.
- Não possuem estruturas de DSP que não serão testadas, ou seja, possui um hardware mais específico para as necessidades desse trabalho, sendo composto basicamente por recursos lógicos e recursos de roteamento.
- Apresenta rede de interconexão similar à do FPGA Xilinx Virtex4, modelo de maior capacidade de processamento e utilizado em diversas pesquisas na área de teste.

TABELA 1 – FAMÍLIA SPARTAN 3E [12]

DISPOSITIVO	PORTAS LOGICAS	EQUIVALENTE DE PORTAS LÓGICAS	MATRIZ DE CLB (UM CLB = 4 SLICES)			
			LINHAS	COLUNAS	TOTAL CLBS	TOTAL SLICES
XC3S100E	100K	2.160	22	16	240	960
XCS3250E	250K	5.508	34	26	612	2.448
XC3S500E	500K	10.476	46	34	1.164	4.656
XC3S1200E	1200K	19.512	60	46	2.168	8.672
XC3S1600E	1600K	33.192	76	58	3.688	14.752

Observa-se que todos os membros da família Spartan 3E apresentam arquitetura baseada em linhas e colunas (da expressão em inglês, ***Rows e***

Columns), onde estarão localizados os blocos programáveis, e que diferem em quantidade entre seus modelos. Uma visão geral dessa arquitetura está disponível na Figura 9.

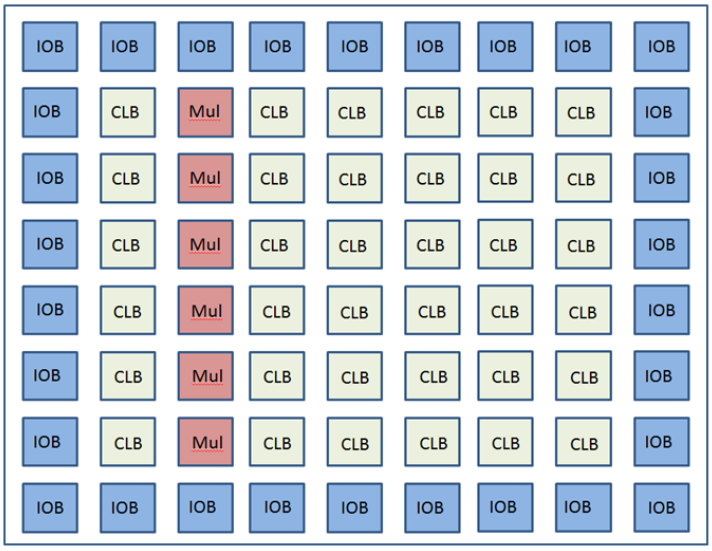


FIGURA 8 - ARQUITETURA FPGA SPARTAN 3E

Os IOBs (da expressão em Inglês, *Input Output Blocks*) são os blocos lógicos responsáveis pelo interfaceamento de sinais internos com os pinos externos do *chip*. Logo, para acessar os pinos de entrada e saída do CI, deve-se utilizar esses blocos. Os CLBs (da expressão em Inglês, *Configurable Logic Blocks*) são blocos programáveis responsáveis por executar a função lógica desejada, por exemplo, portas lógicas, tabela da verdade, multiplexadores, etc. Para isso, utilizam-se nos CLBs algumas estruturas que serão descritas no parágrafo seguinte. Além dos IOBs e CLBs, os FPGAs Xilinx Spartan 3E apresentam *hardware* multiplicadores, ou seja, estruturas específicas para executar funções de multiplicação em forma de blocos multiplicadores (representado por “Mul” na Figura 9) [12].

Um CLB é composto por 4 *slices* (Figura 10), cada *slice* é formado por duas LUTs (da expressão em Ingês, *Look-UP Table*) de 4 entradas, dois *flipflops*/*latches* e alguns componentes como portas lógicas e multiplexadores [12]. As LUTs contêm uma tabela verdade de qualquer função lógica que será representada. Os *flipflops* são utilizados para realizar funções sequenciais no *slice*.

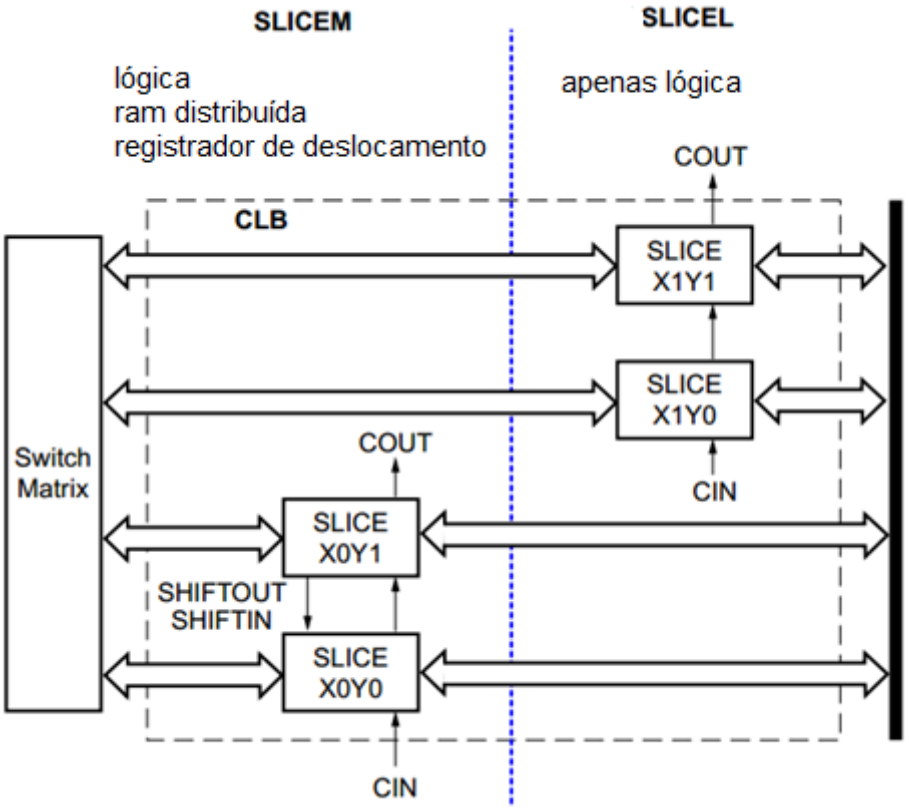


FIGURA 9 - VISÃO DE UM CLB [12]

Nessa família de FPGA, existem dois tipos de *slices*, SLICEL e SLICEM. O SLICEL é capaz de implementar apenas funções lógicas, já o SLICEM é mais complexo. Além de todos os componentes presentes no SLICEL, o SLICEM incorpora componentes adicionais que permitem sua utilização como registrador de deslocamento ou pequenas memórias RAM [12]. Uma visão geral de um *slice* pode ser visualizada na Figura 11. Observa-se que as estruturas em azul só estão presentes no SLICEM.

Em tese, esses *slices* serão configurados como TPG e ORA, necessários para testar a rede de interconexão programável.

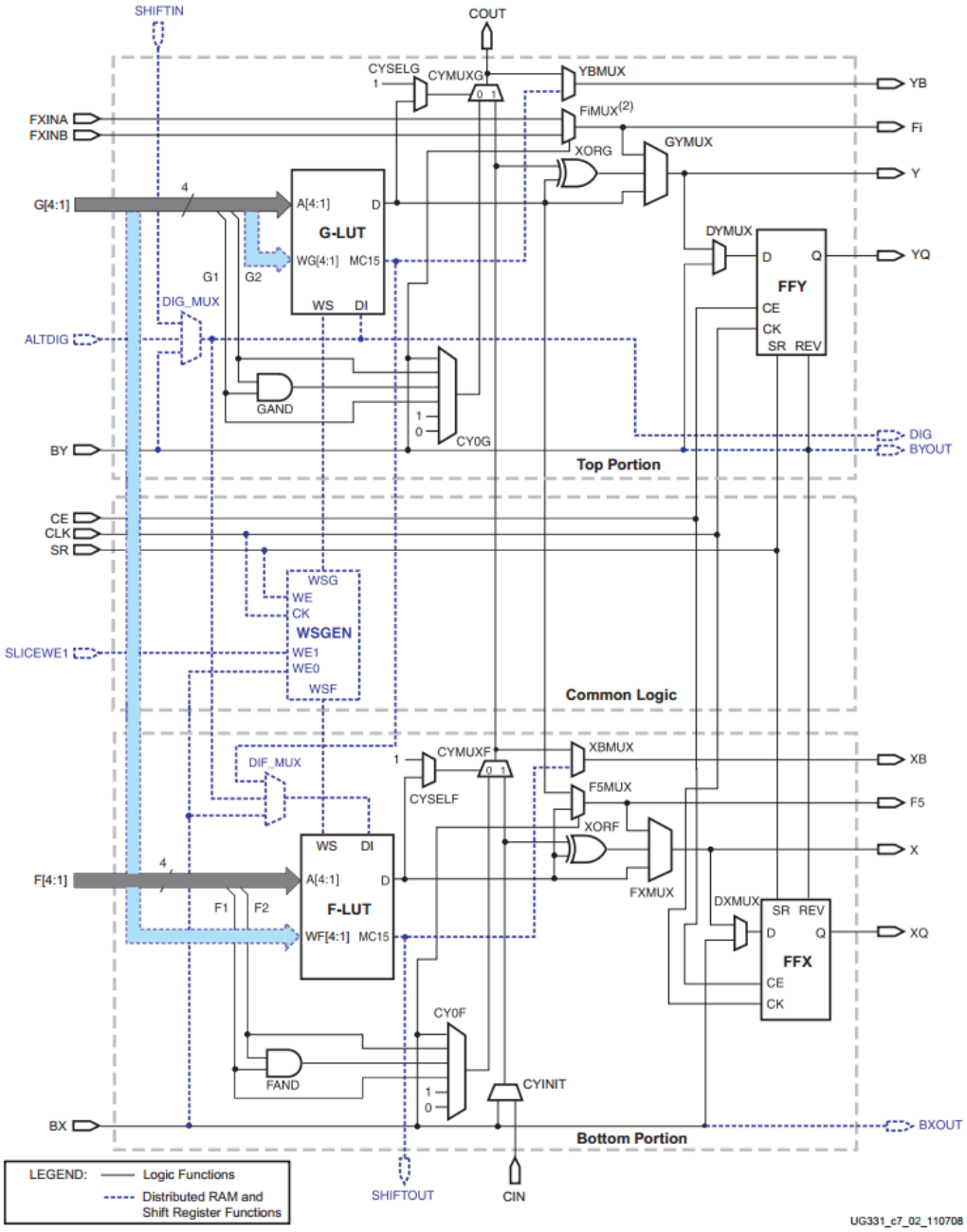


FIGURA 10 - VISÃO GERAL DE UM *SLICE* [12]

2.5 ESTRUTURA DA REDE DE INTERCONEXÃO DO FPGA XILINX SPARTAN 3

A função principal de uma rede de interconexão no FPGA é conectar os CLBs, IOBs e outros componentes internos. Normalmente, 80% dos bits de configuração que serão carregados no CI controlam a rede de interconexão [4]. Para que seja possível controlar vários segmentos de fios, utilizam-se os PIPs (da

expressão em inglês, *Programmable Interconnect Points*). Um PIP é uma chave que liga dois segmentos de fio e é controlado através de um bit de configuração na memória (Figura 12).

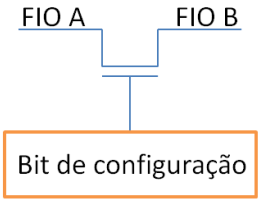


FIGURA 11 - ARQUITETURA DE UM PIP

A maioria desses PIPs e segmentos de fios ficam localizados na matriz de chaveamento (*switch matrix*). Cada CLB, IOB e componente embarcado têm uma matriz associada e cada sinal que será transmitido de um bloco lógico a outro e que deve ser roteado através da matriz de chaveamento, conforme figura 13.

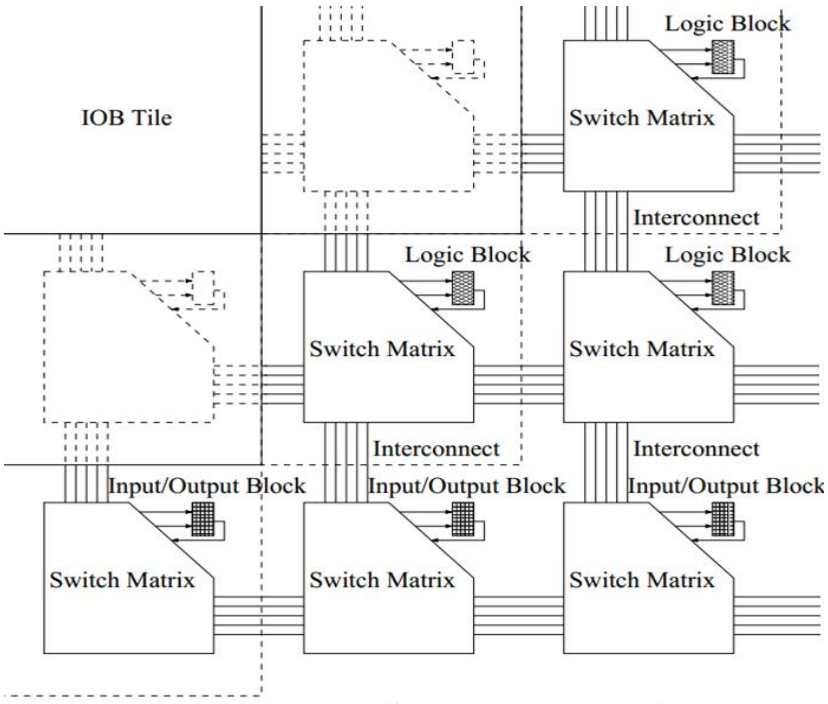


FIGURA 12 - ARQUITETURA DA REDE DE INTERCONEXÃO [13]

O FPGA Spartan 3E utiliza uma matriz de chaveamento composta por segmentos de fios e PIPs multiplexados. Multiplexadores de entrada ligam sinais da interconexão para os *slices*, já os multiplexadores de saída ligam os sinais do *slice* à rede de interconexão [13]. Um típico multiplexador PIP de dois níveis pode ser visualizado na figura 14.

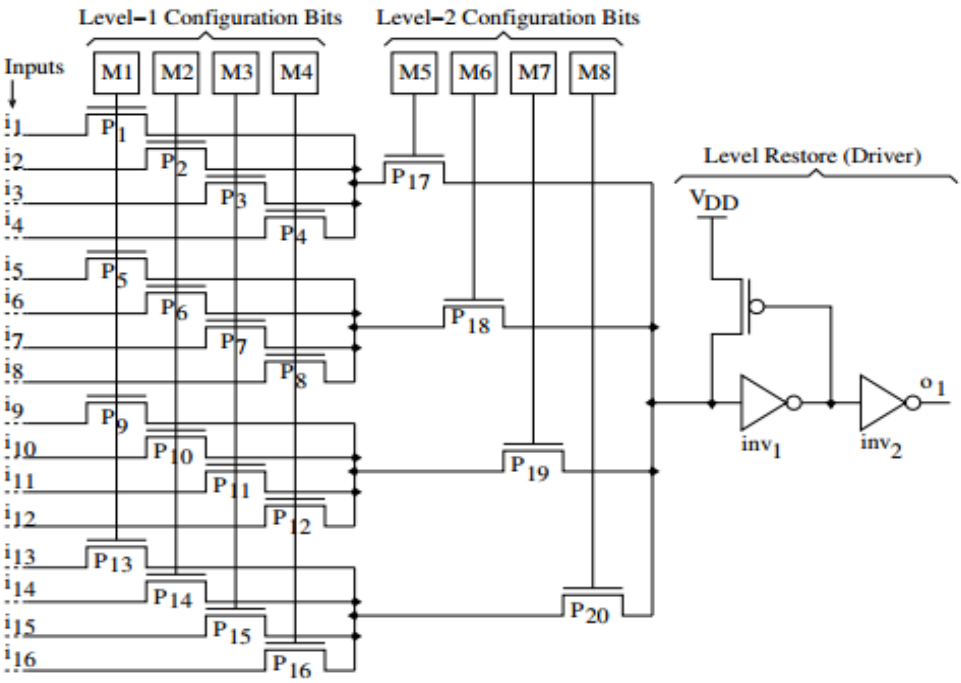


FIGURA 13 - ARQUITETURA DE UM MULTIPLEXADOR PIP [13]

Observa-se que quando dois bits de configuração no mesmo nível são ativados, ocorre um curto entre as entradas. Como esses multiplexadores são construídos com transistores da tecnologia NMOS, o valor lógico “zero” é dominante [13]. Logo, pode-se detectar por meio de falha equivalente a falhas em ponte e falhas *Stuck-At*, PIPs “travados” que podem estar ligados ou desligados. Caso esteja ligado, este se denomina PIP *stuck-on,* e PIP *stuck-off* caso esteja desligado [13].

Os segmentos de fios presentes no FPGA classificam a rede de interconexão em duas formas: Rede de interconexão local e rede de interconexão global. A primeira é responsável pelo roteamento de sinais de entrada e saída da matriz de chaveamento a sinais internos no CLB. A rede de interconexão global é composta por fios que não fazem parte da matriz de chaveamento, e que servem para ligar as matrizes de chaveamento entre si, e existem por todo FPGA. A estratégia de teste descrita nesse trabalho está direcionada à rede de interconexão global, sendo a que ocupa maior área no CI e por isso está mais suceptível a falhas.

A rede de interconexão global pode ser dividida em três tipos de segmentos de fios diferentes: *Double lines, hex lines e long lines*, conforme visualizado na Figura 15.

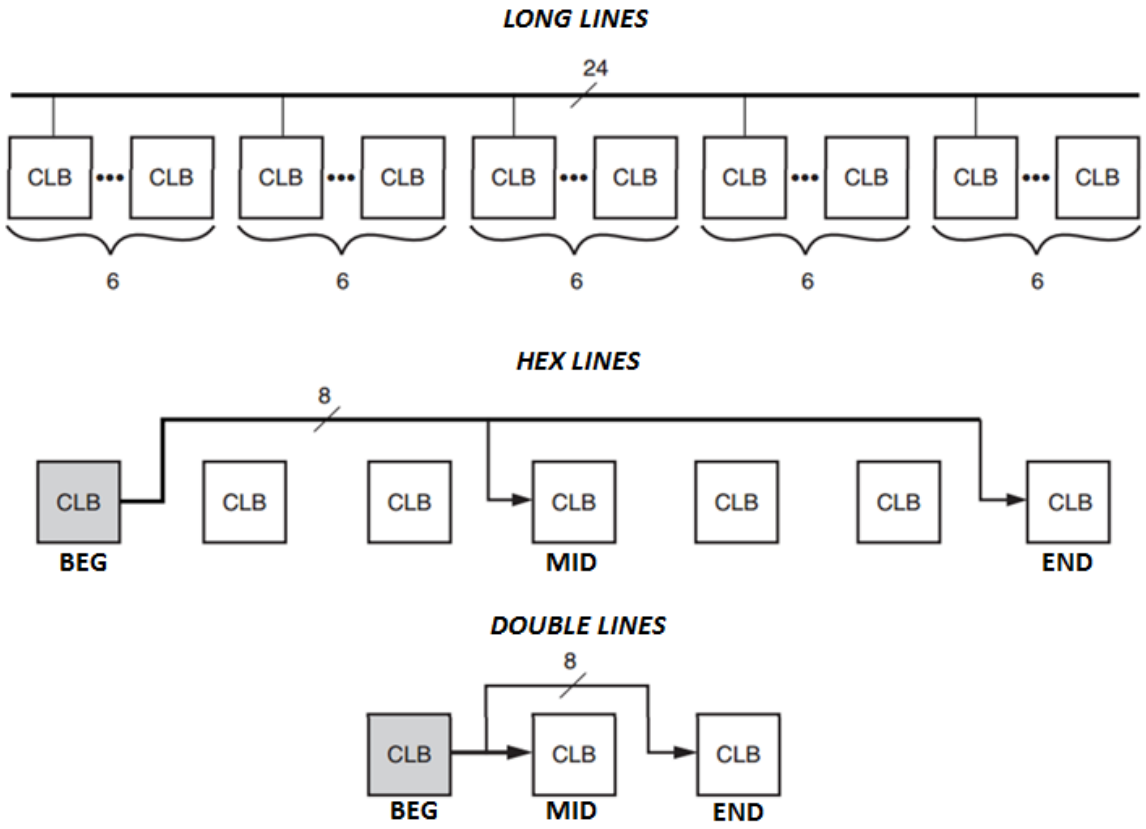


FIGURA 14 - LINHAS QUE COMPÕE A REDE DE INTERCONEXÃO GLOBAL [14]

As linhas *double lines* são aquelas que ligam três CLBs vizinhos, são unidirecionais e existem nas quatro direções: Norte, sul, leste e oeste. Observa-se que o CLB enviando sinais para a linha recebe a denominação de BEG e, o que se encontra no meio, recebe a denominação de MID. Já o END é o último CLB a receber esse sinal.

As linhas *hex lines* são semelhantes às *double lines*, a única diferença é que serão ligados os 3 CLBs com intervalos de 2 entre si, ou seja, supondo que o CLB BEG seja representado por 0 (o primeiro da linha), o terminal MID será o 3 e o END será o 6.

Linhas *long lines* perpassam por toda a extensão do C.I em dois sentidos, horizontal e vertical, e se conectam a um CLB para cada 6. Em qualquer CLB, apenas 4 fios *long lines* podem enviar ou receber sinais. Devido à maior quantidade de linhas *double lines* e *hex lines*, a estratégia proposta nessa trabalho focará o teste nessas linhas, sendo essas as que apresentam maior probabilidade de falha.

2.6 REVISÃO DE LITERATURA: ESTRATÉGIAS DE TESTES DE INTERCONEXÕES DE FPGAS

Recuperou-se o conhecimento já mencionado na literatura acerca das estratégias de teste já existentes aplicadas à rede de interconexão de FPGAs sendo descritos os trabalhos mais relevantes.

Em [15], é apresentada uma estratégia de teste aplicada à rede de interconexão de FPGAs que pode ser utilizada tanto em testes *on-line* como em testes *off-line*. Inicialmente, essa técnica foi desenvolvida para teste e diagnóstico *online* de aplicações tolerantes a falhas, ou seja, aplicações que podem sofrer uma falha e se reprogramar para evitar tais recursos defeituosos. Porém, observou-se que essa técnica poderia ser aplicada em testes *offline*, ou seja, o FPGA ficará fora de operação e se reprogramará para testar integralmente sua rede de interconexão, e na sequência, será carregada a aplicação nos recursos não defeituosos.

Para testar a rede de interconexão, [15] utilizou uma estratégia baseada em contador de N-bits associado a um gerador de paridade, conforme visto na figura 16.

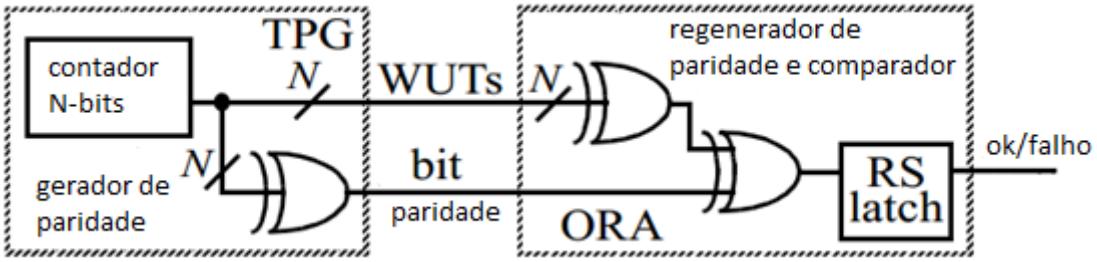


FIGURA 15 - ARQUITETURA DE TESTE DESENVOLVIDA POR STROUD & ABRAMOVICI, 2002 [15]

A estrutura apresentada acima foi implementada em FPGA na forma de STAR (da expresão em Inglês, *Self-testing area*). Essa implementação utiliza para o teste uma área do FPGA que não está operacional. Movendo-se o STAR para outras áreas, garante-se que todo o FPGA seja testado. Este método recebe o nome de RSTARS (da expressão em Inglês, *Roving Self-test areas*) (Figura 17).

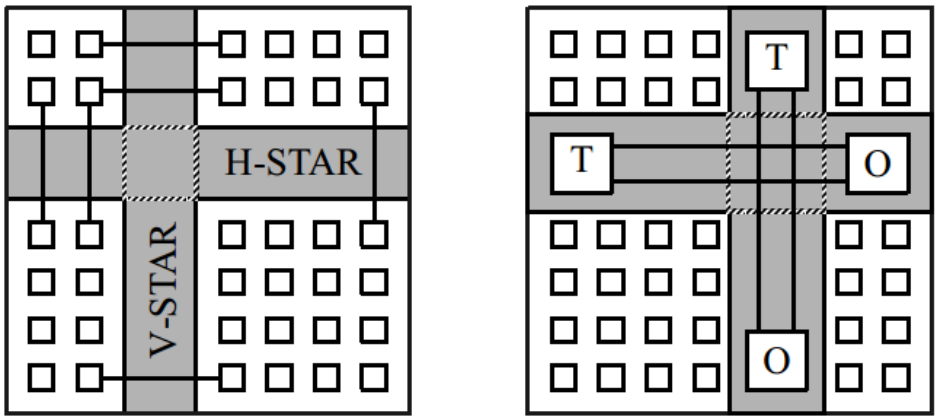


FIGURA 16 – ESTRATÉGIA DE TESTE RSTARS PARA TESTE *ONLINE* [15]

Observa-se que todos os fios presentes nos STARs estão reservados para teste, excluindo-se apenas os que estão roteando sinais da aplicação. Quando é reservado um STAR no sentido horizontal, a ele é atribuido o nome de H-STAR, e V-STAR para vertical. Nas áreas de STAR, estão presentes um TPG (representado pela letra “T”, na Figura 17) e um ORA (representado pela letra “O”, na Figura 17).

Para testes *offline*, é aplicado o conceito de STARs em todo FPGA, isto é, são reservados todos os fios em determinada direção e é aplicado o teste, conforme visualizado na Figura 18.

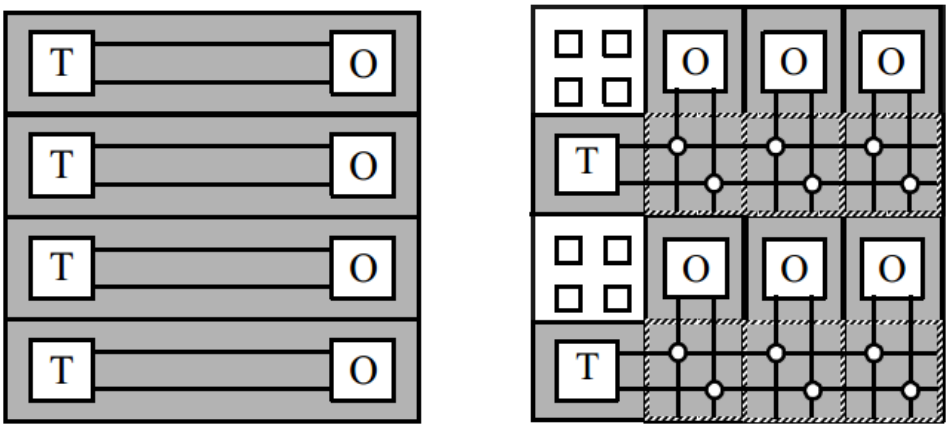


FIGURA 17 - ESTRATÉGIA DE TESTE RSTARS PARA TESTE *OFFLINE* [15]

Quando uma falha é detectada, o local desse defeito deve ser identificado com uma boa resolução para facilitar uma reprogramação mais eficiente da aplicação. Ou seja, é preciso conhecer o local da falha para que a aplicação não utilize recursos defeituosos. Para indentificar o local da falha, [15] utilizou configurações de teste extras, isto é, configurações além das já utilizadas para testar

todo o FPGA. Essas configurações foram feitas baseadas nas seguintes estratégias: 1) substituir WUTs (onde cria-se um caminho alternativo e se observa se o mesmo permanesse com falha), conforme Figura 19a; 2) adicionando ORAs que serão utilizados para testar essa linha em várias posições e assim identificar o local da falha, conforme Figura 19b; 3) dividir para conquistar, onde o teste é dividido em dois, e por meio de divisões sucessivas encontra-se o local da falha, conforme Figura 19c; e 4) re-rotear segmentos de fios a fim de encontrar um segmento específico com falha, conforme visualizado na Figura 19d.

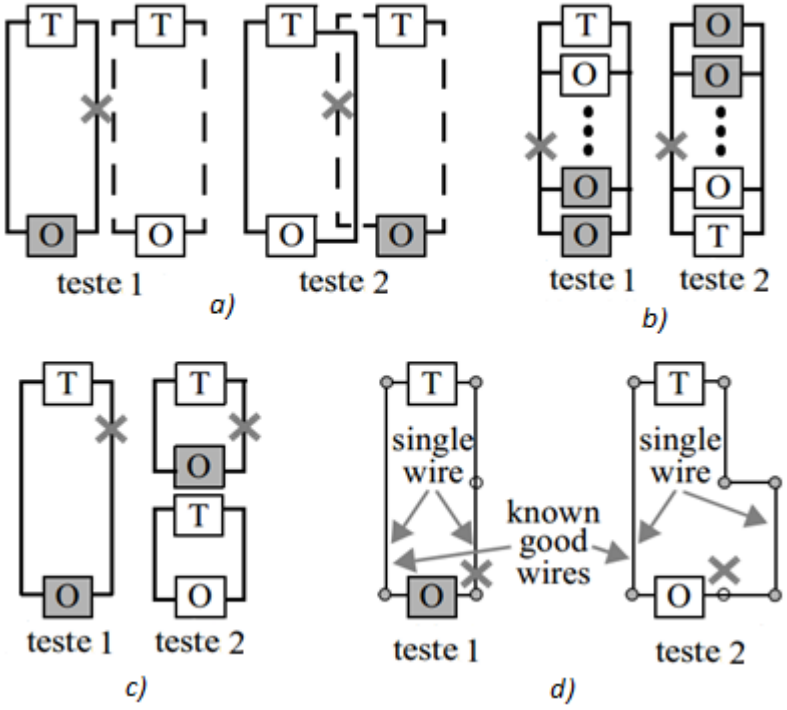


FIGURA 18 - ESTRATÉGIAS PARA DIAGNÓSTICO DO LOCAL DA FALHA [15]

Essa proposta apresentada em [15] foi implementada em um FPGA do fabricante ORCA modelo 2C15. A partir dessa implementação em FPGA, é possível identificar as configurações de teste necessárias, conforme visto na Tabela 1. Observa-se que uma **seção de teste** envolve várias **fases de teste** para identificar um determinado conjunto de falha em determinados recursos.

SESSÃO DE TESTE	FALHAS PROCURADAS	FASE V-STAR	FASE H-STAR
1	Curtos, abertos, Falhas de PIP na rede global	7	7
2	Falhas de PIP, curtos e abertos na rede local	4	4
3	Falhas de PIP entre rede global e local	3	3
4	Falha em PIPs multiplexados	7	0
5	Falha em PIPs na rede global	6	

TABELA 2 - CONFIGURAÇÕES DE TESTE UTILIZANDO RSTARS [15]

Como o FPGA utilizado em [15] para implementar as estratégias de teste é de fabricante diferente do utilizado nessa pesquisa, o termo CIP (da expressão em Inglês, *Configurable Interconnect Points*) foi utilizado, porém, este representa a mesma estrutura física dos PIPs não multiplexados.

Em 2004, [16] propôs para o teste de *Bridge Faults* em redes de interconexão, um método baseado em Iddq, que é a corrente de alimentação do FPGA em modo quiescente. Quando uma *Bridge Fault* é ativada, há um fluxo de corrente parasita entre os fios, que é refletida na corrente de alimentação geral do FPGA [16]. O teste de Iddq padrão consiste em carregar configurações de teste no FPGA ativando possíveis falhas em ponte. Após o estado transitório analisar a corrente de alimentação do FPGA, quando essa corrente ultrapassa um limiar (***threshold***) pré-estabelecido, esse dispositivo é rejeitado [16].

No entanto, a eficiência do teste padrão baseado em Iddq diminuiu devido ao aumento nas correntes parasitas (*Leakage*) de microestruturas. A diminuição no limiar da tensão de operação e o aumento da quantidade de transistores causaram um aumento exponencial na Iddq, mas não causaram um aumento nas correntes de falha [16]. A corrente de alimentaçao de um FPGA é bastante alta devido à grande quantidade de PIPs e à porção do FPGA que não está sendo utilizado, onde normalmente é fixado um valor lógico para evitar ruido e *crosstalk* [16]. Estudos realizados por [17] descobriram que 65% da potência dissipada está relacionada à rede de interconexão.

Para conseguir mais eficiência no teste baseado em corrente Iddq, foi criado o teste de corrente Iddq diferencial (ΔIddq), que aplica a primeira derivada discreta de Iddq através de um vetor numérico, ΔIddq[i] = Iddq[i] – Iddq[i-1], ou seja, a diferença entre sucessivas medições de Iddq. Após isso, é gerado um gráfico com a assinatura de consumo diferencial e uma descontinuidade nesse gráfico pode representar uma falha em ponte [16].

Antes de realizar o teste, é preciso carregar uma configuração que fixe toda a rede de interconexão em um dado nível lógico, por exemplo, 1, e então é medida uma corrente de consumo de referência, Iddq-ref. Como nenhuma falha em ponte foi ativada, a Iddq-ref representa a corrente de vazamento nos PIPs. Em seguida, são carregadas várias configurações que aplicam um conjunto da interconexão para valor lógico “zero” mantendo o resto em “um”. Para cada configuração, uma corrente de consumo total Iddq é medida, Iddq-tot. Como a corrente Iddq-tot envolve uma possível corrente de falha (Iddq-*fault*) e uma corrente de consumo da interconexão quando são aplicados valores opostos (Iddq-Int), logo: Iddq-tot = Iddq-*fault* + Iddq-int.

Para criar uma corrente de assinatura para cada configuração de teste, pode-se subtrair a corrente de consumo total da corrente de consumo de referência, conforme visto na Equação 1. Logo, a corrente de assinatura representa a soma da corrente consumida pela rede de interconexão e uma possível corrente de falha.

$$\begin{aligned} I_{DDQ_{sig}} &= I_{DDQ_{tot}} - I_{DDQ_{ref}} \\ &= (I_{DDQ_{ref}} + I_{DDQ_{int}} + I_{DDQ_{fault}}) - I_{DDQ_{ref}} \\ &= I_{DDQ_{int}} + I_{DDQ_{fault}}. \end{aligned}$$

EQUAÇÃO 1 - CORRENTE DE ASSINATURA IDDQ [16]

Testes foram conduzidos num FPGA Spartan 3 XC3S50. Foram utilizados PIPs que não estavam sendo utilizados para emular uma falha em ponte de aproximadamente 1kΩ. Na Figura 20*a*, pode-se visualizar uma estrutura de um PIP trandicional, já na figura 20*b*, é demonstrada uma falha em ponte e na Figura 20*c*, observa-se uma falha em ponte emulada através da ligação de um PIP.

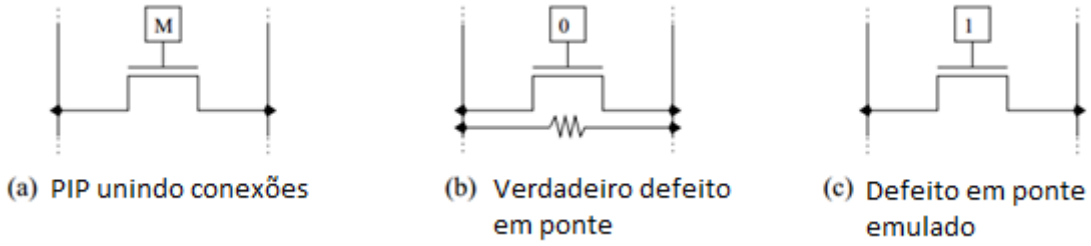


FIGURA 19 - EMULAÇÃO DE FALHA EM PONTE [16]

A Figura 21 mostra os resultados dos testes realizados. Observa-se que o teste Iddq padrão por utilizar um limiar (***threshold***) fixo apresenta dois resultados inválidos (***False Failure*** **e** ***Test Escape***). Para os teste realizados com ∆Iddq, há uma cobertura de defeito mais aceitável.

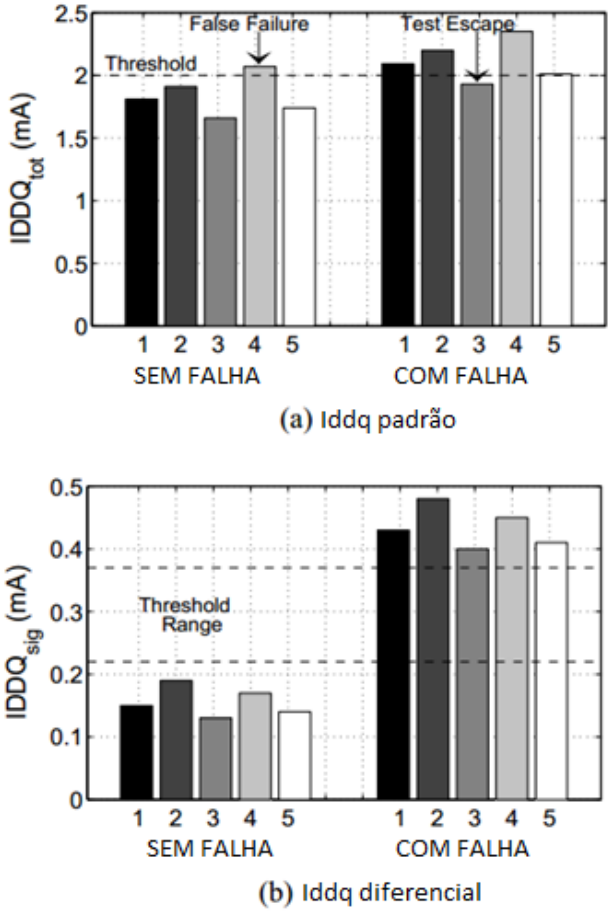


FIGURA 20 - RESULTADOS DE TESTE ∆IDDQ [16]

Em [4], apresenta-se uma análise das estratégias utilizadas na época, bem como resultados de simulações de cobertura de falhas para testes *offline*.

As estratégias descritas em [4] são baseadas em: contadores, contadores com paridade, contadores com paridade invertida, LFSR (da expressão em Inglês, *Linear Feedback Shift Register*) e CAR (*Cellular Automata Register*). Estas serão descritas a seguir.

Estratégia baseada em contadores consiste em utilizar um contador de dois bits para gerar os vetores de teste (***single counter***). Estes serão aplicados nos fios e transmitidos ao ORA que identifica a falha baseada na comparação entre dois fios.

Observa-se que esta estratégia é capaz de gerar 4 sinais e distribui-los em 8 WUTs, conforme Figura 22.

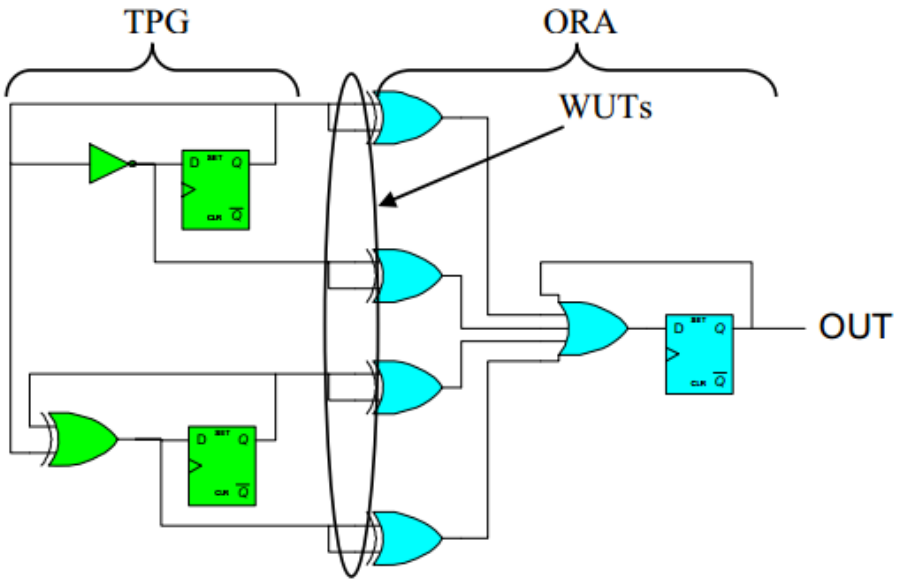


FIGURA 21 - TESTE BASEADO EM CONTADOR DE DOIS BITS [4]

Em seguida, foi criada a estratégia baseada em contador duplo (***dual counter***), que consiste em utilizar dois contadores de dois bits em conjunto para gerar 8 WUTs. A diferença em relação ao contador de dois bits simples é que os sinais são gerados de forma independente, utilizando o estado atual e o estado anterior do contador, proporcionando uma melhor cobertura de falha [4]. Essa estratégia pode ser visualizada na Figura 23, observa-se que CNTR0 e CNTR1 denominam os contadores, da expressão em inglês *Counter*.

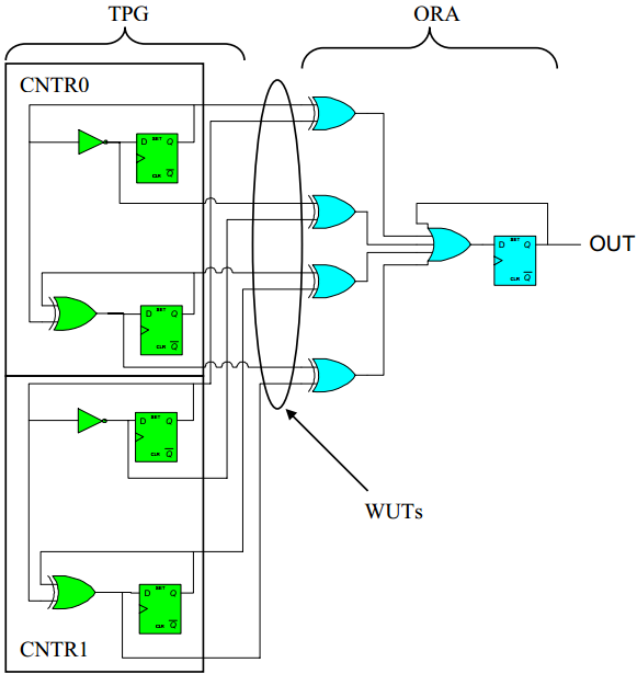


FIGURA 22 - TESTE BASEADO EM UM CONTADOR DUPLO DE DOIS BITS [4]

Essas estratégias baseadas em contadores foram descritas em ASL (da expressão em Inglês, *Auburn Simulation Language*) e simuladas utilizando o *software* AUSIM. A partir dessa simulação, foram obtidos os resultados que podem ser visualizados na Figura 24.

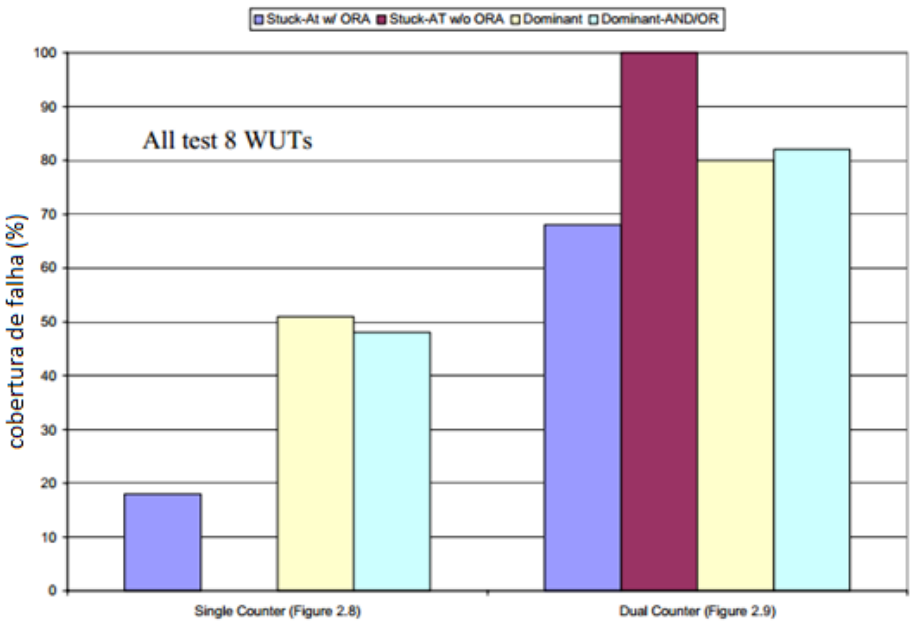


FIGURA 23 - RESULTADOS DE SIMULÇÕES DE ESTRATÉGIAS BASEADAS EM CONTADORES [4]

Conforme visualizado na figura acima, o contador simples não é capaz de detectar nenhuma falha *stuck-at.* Os resultados em marrom, no gráfico anterior, representam a cobertura de falhas *stuck-at* excluindo aquelas que podem ocorrer no interior do ORA, ou seja, as falhas geradas pelo TPG, ou rede de interconexão. Já o contador duplo é capaz de detectar 100% das falhas *stuck-at* excluindo falhas no ORA. Esse resultado foi um dos responsáveis pela decisão de investigar estratégias baseadas em uma combinação TPG/ORA. [4]

Além das estratégias baseadas em contadores, há estratégias baseadas em paridade (***dual parity***). Nelas, utiliza-se um contador como TPG e um gerador de paridade, que transmitirá esse sinal em outro fio. Logo, se algum bit for invertido, este poderá ser detectado pelo ORA, que acusará um erro de paridade. Uma grande desvantagem dessa estratégia é que se assume que o fio pelo qual será transmitido a paridade esteja livre de falhas [4]. Esta estratégia pode ser visualizada na Figura 25.

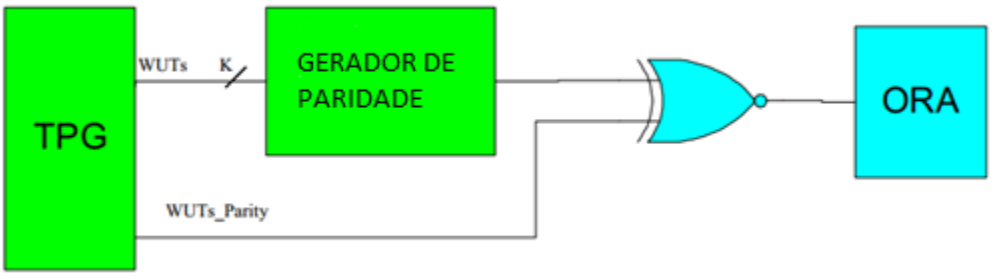


FIGURA 24 - ESTRATÉGIA BASEADA EM PARIDADE [4]

Essa suposição não era realista, uma vez que o fio que transmite a paridade também deve fazer parte do teste. Logo, essa estratégia foi modificada, e o sinal de paridade é enviado junto ao conjunto de WUTs, conforme visualizado na Figura 26.

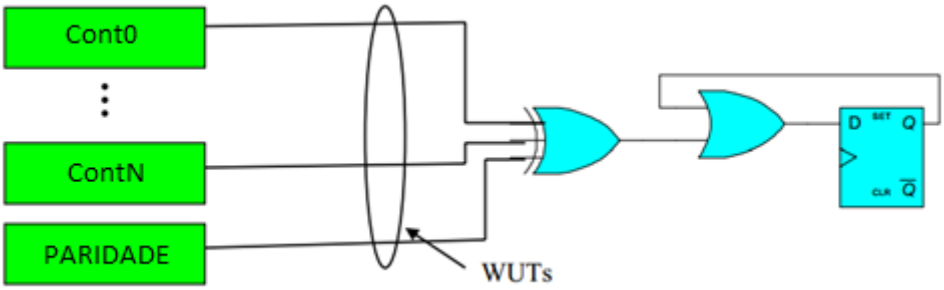


FIGURA 25 - ESTRATÉGIA BASEADA EM PARIDADE MODIFICADA [4]

Como as estratégias baseadas em paridade requerem sinais que realimentam os contadores, alguma falha nessa rede de realimentação pode modificar a sequência gerada pelo contador, e o circuito de paridade não a detectará.

Para corrigir esse problema, foi criada uma estratégia baseada em paridade invertida (***cross-coupled parity***). Esta apresenta dois contadores de dois bits, um deles iniciado em “zero” com contagem crescente, outro iniciado em “três” (em binário, 11) com contagem decrescente. Os bits de paridade são gerados para ambos os contadores, e quando enviados, são invertidos. Ou seja, a paridade do contador crescente é enviada ao ORA do contador decrescente, e paridade do contador decrescente é enviada ao ORA do contador crescente. Como estas são equivalentes, o ORA detectará a falha, ou seja, a paridade do contador crescente inciado em “zero” é a mesma desse contador quando iniciado em “três” (11 em binário), e assim para os demais vetores, o ORA detectará a falha. Esta estratégia pode ser visualizada na Figura 27.

Com essa estratégia é possível detectar falha em até 12 WUTs e expansível até 18, se os 6 WUTs que estão conectados aos ORAs forem duplicados na distribuição, como no contador de dois bits. Observa-se que isso ocorre apenas se considerados os fios de realimentação dos contadores como WUTs.

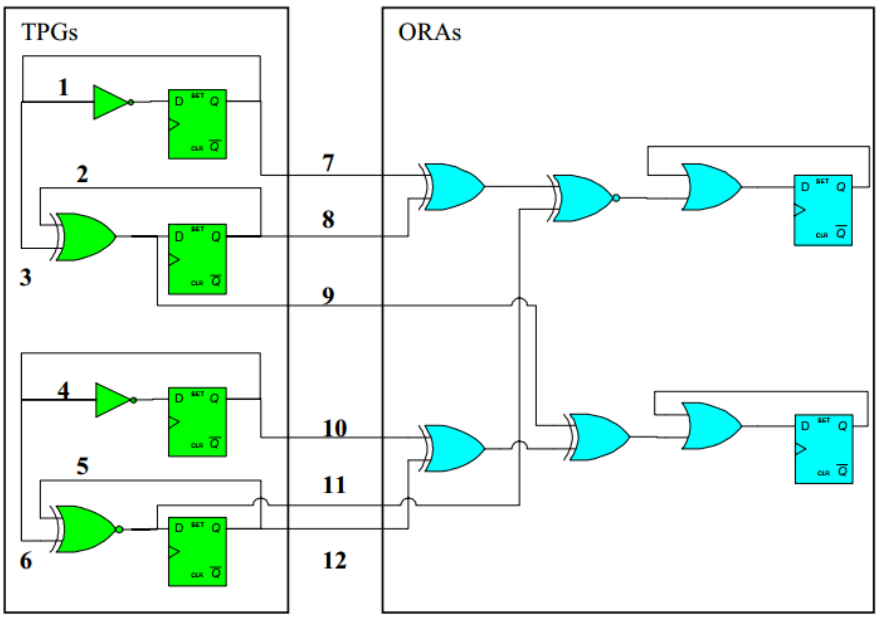


FIGURA 26 - ESTRATÉGIA DE TESTE BASEADA EM PARIDADE INVERTIDA [4]

Essas estratégias baseadas em paridade foram descritas em ASL e foram simuladas por [4] com o *software* AUSIM. Os resultados obtidos podem ser visualizados na Figura 28.

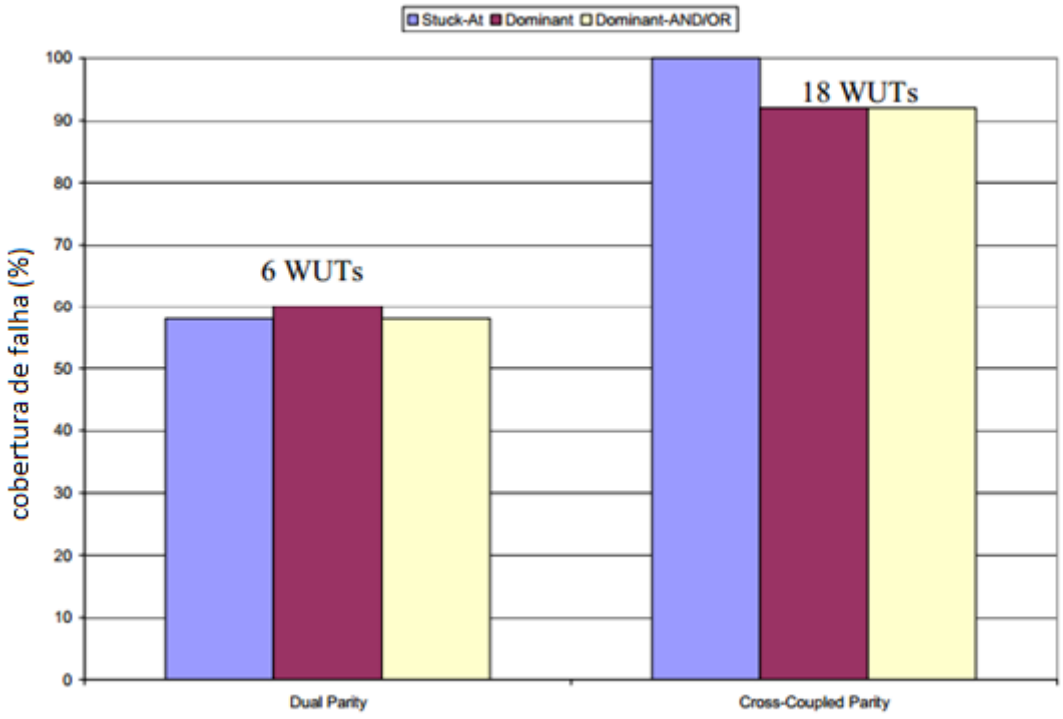


FIGURA 27 - RESULTADOS DE SIMULÇÕES DE ESTRATÉGIAS BASEADAS EM PARIDADE [4]

Observa-se que a estratégia baseada em paridade invertida é capaz de atingir 100% de cobertura *stuck-at* e apresenta uma melhora significativa quando comparada à estratégia com paridade comum.

Além dessas estratégias que envolvem contadores utilizados como TPG, e outro circuito configurado como ORA, surgiu a proposta de um testador misto TPG/ORA. Ou seja, ao mesmo tempo em que os vetores são gerados para testar a interconexão, essa estrutura é sensível a falhas e ocorrerá uma mudança de seus estados caso isso aconteça. A seguir, serão descritas essas estratégias mistas.

Um bom exemplo de testador misto é o LFSR (da expressão em Inglês, *Linear Feedback Shift Register*) de polinômio primitivo, conforme visto na Figura 29. Após pulsos de *clock* em número suficiente para que cada estado do registrador altere seu valor lógico (sendo 2^n – 1 a maior sequência de vetores), o resultado do

teste pode ser lido dos registradores. Caso esse apresente um comportamento incomum, se detectará a falha.

Existem dois tipos de LFSR e a diferença entre eles é como a realimentação é aplicada. Quando as estruturas de realimentação encontram-se dentro da cadeia dos registradores, podemos dizer que esse apresenta realimentação interna (Figura 29).

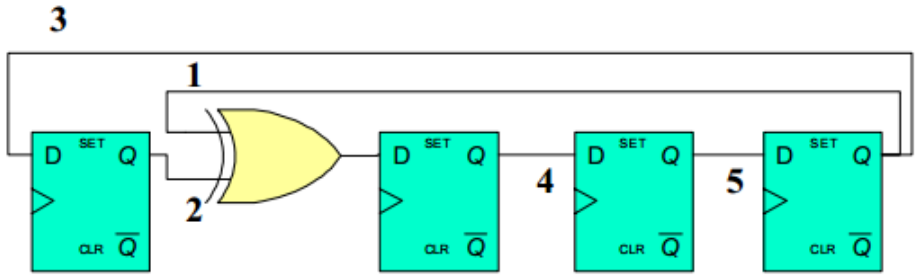


FIGURA 28 - LFSR DE QUATRO BITS COM REALIMENTAÇÃO INTERNA [4]

Quando este apresenta as estruturas de realimentação fora da cadeia de registradores, pode-se dizer que esse é um LFSR de realimentação externa, conforme Figura 30.

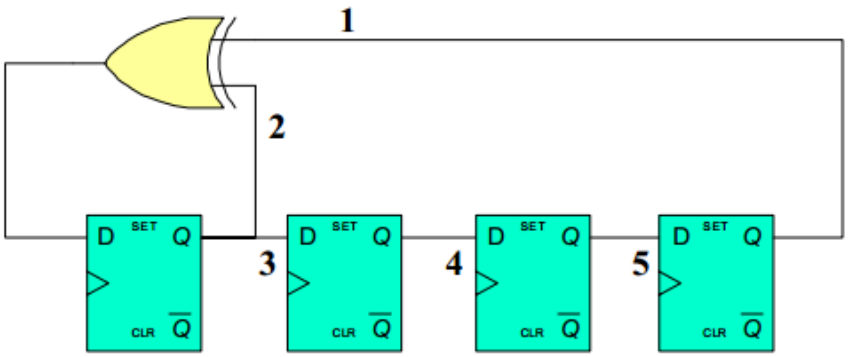


FIGURA 29 - LFSR DE QUATRO BITS COM REALIMENTAÇÃO EXTERNA [4]

É possível implementar em um mesmo CLB de uma Virtex4 dois LFSRs de quatro bits, que apresentam resultados similares a LFSRs de oito bits. Estes resutados são mostrados na Figura 31.

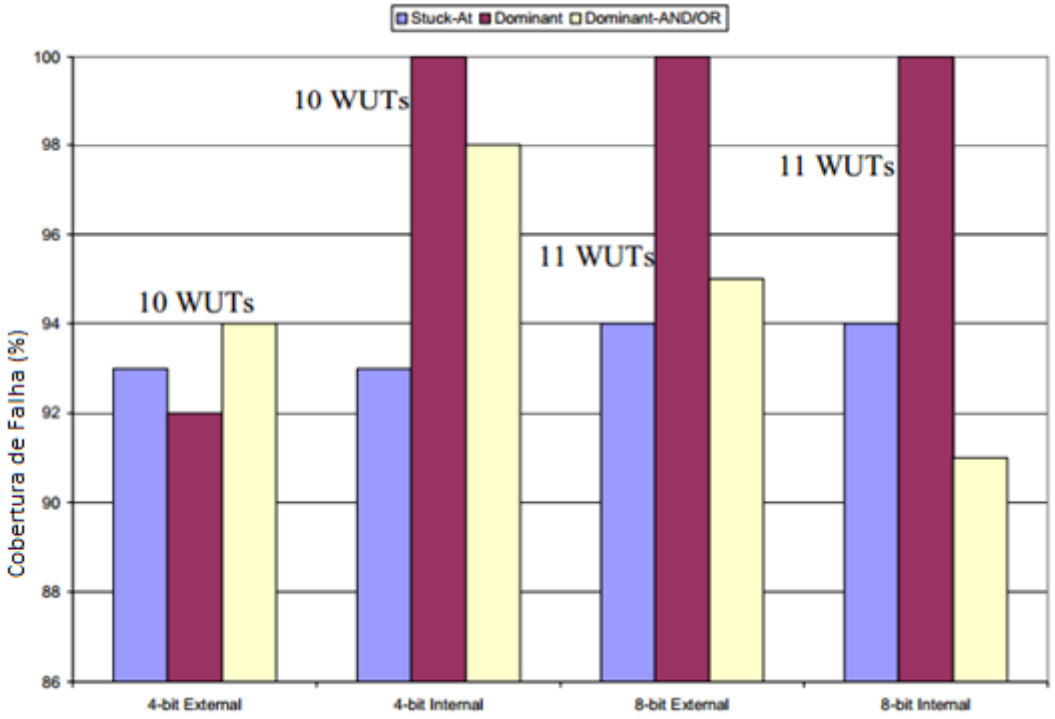


FIGURA 30 - RESULTADO DE SIMULAÇÕES DE ESTRATÉGIAS BASEADAS EM LFSR [4]

Como visto na figura acima, todas as estruturas baseadas em LFSR apresentam cobertura de falha razoável, acima de 90% para padrões acadêmicos, porém, apresentam uma deficiência em quantidade de fios sob teste. A cobertura de falha dominantes para o LFSR externo de 4 bits foi menor devido ao fio de realimentação do XOR no bit menos significativo, que dificultou a detectabilidade de possíveis falhas em ponte.

Outra alternativa de testador misto é baseado em CAR (da expressão em Inglês, *Cellula Automata Register*). Enquanto o LFSR é um registrador regido por um polinômio, o CAR é um registrador regido por regras. As regras mais utilizadas para construção de um CAR são as regras de 90 e regras de 150 e os detalhes para sua construção estão descritos na Tabela 2. Um determinado registrador *i* quando regido pela regra de 90 apresenta duas entradas, exceto quando $i = 0$ ou $i = N$, sendo 0, o primeiro registrador e N, o último.

Nesses casos, o registrador apresenta apenas uma entrada. Para a regra de 150, os registradores internos a cadeia apresentam três entradas, e duas entradas para quando $i = 0$ ou $i = N$. Logo, a regra de 150 sempre gera um WUT a mais do

que a regra de 90, sendo assim, buscou-se maximizar o uso das regras 150 na construção do CAR.

REGRA	Bit 1 (LSB)	i^{th} Bit	Bit N (MSB)
150	$Q_1^+ = Q1 \oplus Q2$	$Q_i^+ = Qi \oplus Q_{i-1} \oplus Q_{i+1}$	$Q_N^+ = Q_N \oplus Q_{N-1}$
90	$Q_1^+ = Q2$	$Q_i^+ = Q_{i-1} \oplus Q_{i+1}$	$Q_N^+ = Q_{N-1}$

TABELA 3 - REGRAS PARA REGISTRADORES CAR [4]

Devido à natureza do CAR, o uso da regra de 150 em todos os seus estágios torna-o restrito a apenas dois vetores de teste. Outro fator é que dependendo do vetor de inicialização, o CAR pode ter um de seus estágios travados em um valor lógico. Para compensar a falta de geração de padrões de teste únicos, foi desenvolvido um CAR de 8 bits com condições cíclicas de fronteira, ou seja, é um registrador CAR que apresenta uma realimentação, conforme visto na Figura 32.

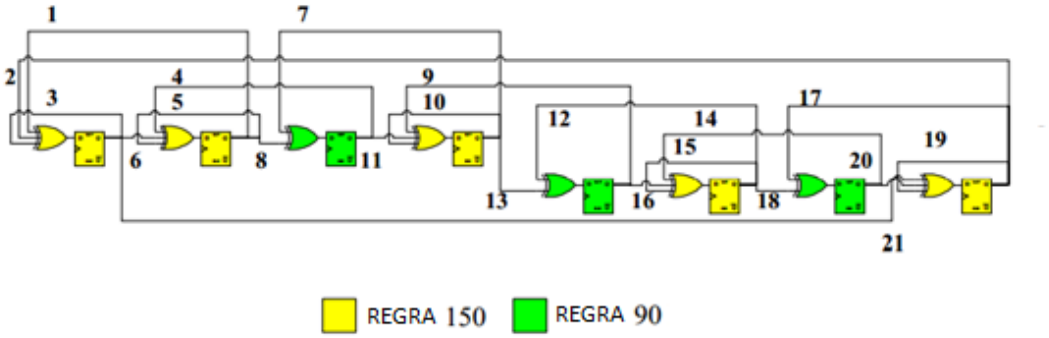


FIGURA 31 - REGISTRADOR CAR DE 8 BITS COM CONDIÇÕES DE FRONTEIRAS CÍCLICAS [4]

A única maneira de conseguir todas as combinações em um CAR é assumir condições de fronteira nulas [2]. Devido a isso, o CAR cíclico de 8 bits foi modificado para o CAR de maior comprimento (*Maximal Length*). Essa modificação proveu 2^n – 1 vetores de teste, porém, reduziu a quantidade de WUTs de 21 para 19, conforme pode ser visto na Figura 33.

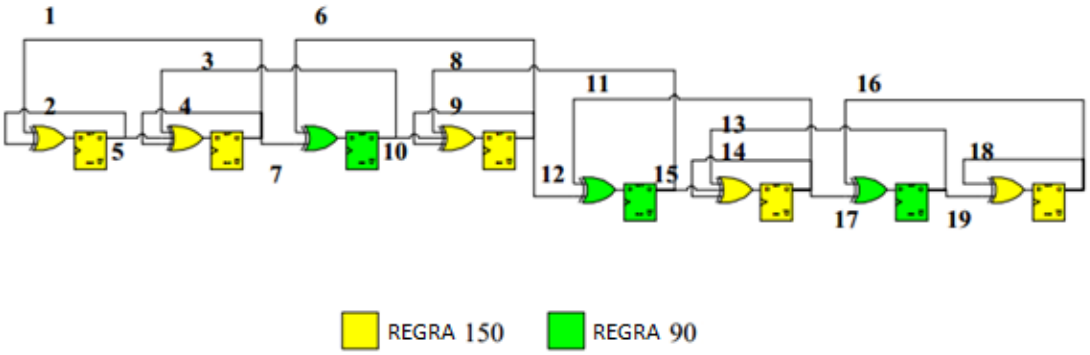


FIGURA 32 - REGISTRADOR CAR DE MAIOR COMPRIMENTO (*MAXIMAL LENGTH*) [4]

Estas estruturas acima foram descritas em ASL e simuladas por [4] através do *software* AUSIM. O resultado dos testes podem ser visualizados na Figura 34.

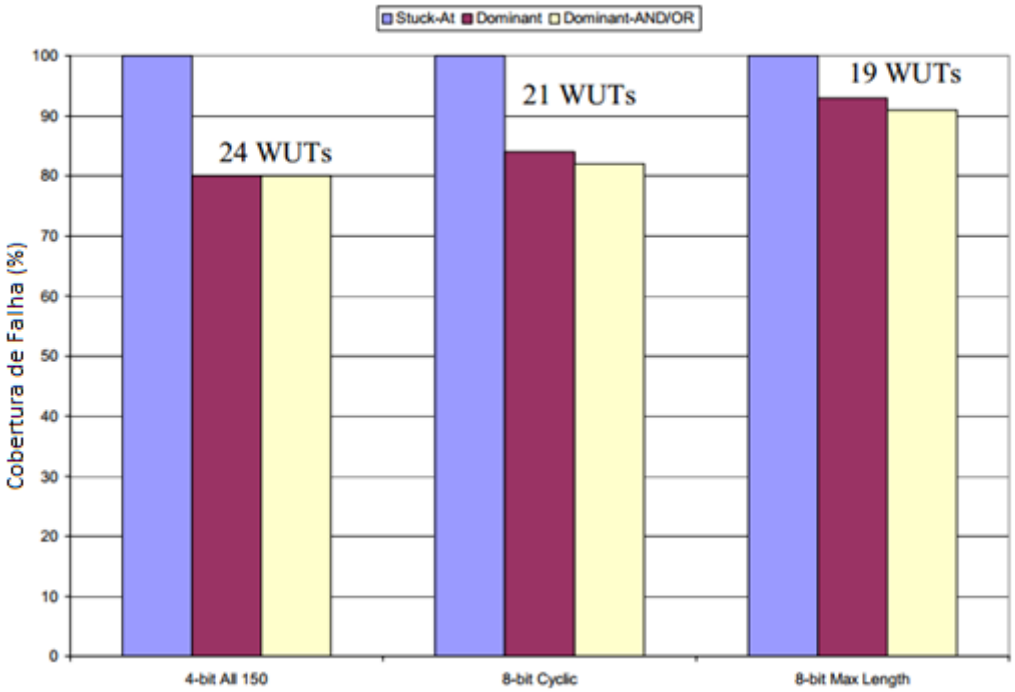


FIGURA 33 - RESULTADO DE SIMULAÇÕES DE ESTRATÉGIAS BASEADAS EM CAR [4]

O CAR cíclico de 8 bits não produz vetor suficiente para atingir uma boa cobertura de falha. Isso requer que um novo valor inicial seja carregado, necessitando outra configuração de teste, ou seja, isso dobrará o tempo de teste, já

que é necessário fazer o *readback* dos valores do teste e reprogramar o FPGA com um novo valor inicial no registrador.

Esses resultados mostraram que toda estratégia tem sua vantagem e desvantagem. A quantidade de WUTs e a quantidade de configurações de teste e *readback* devem ser levadas em consideração, já que são um fator crucial para diminuição do tempo de teste. Nesse aspecto, se destacaram as estratégias baseadas em contador com paridade invertida e o CAR de máximo comprimento de 8 bits, apresentando ótimas coberturas de falhas. A desvantagem encontrada nessas estratégias são as limitações físicas do FPGA para que seja possível programa-lás, por exemplo, o CAR de máximo comprimento de 8 bits não pode ser implementado em um único CLB. Dessa forma, a estratégia baseada em contador com paridade invertida é a mais indicada para teste de roteamento de FPGA.

[18] descreve em seu trabalho uma estratégia baseada em contador duplo com paridade invertida para a rede de interconexão global, bem como a implementação em um FPGA Virtex4 incluindo resultados experimentais. Tal arquitetura para teste de linhas *Hex Lines* pode ser visualizada na Figura 35.

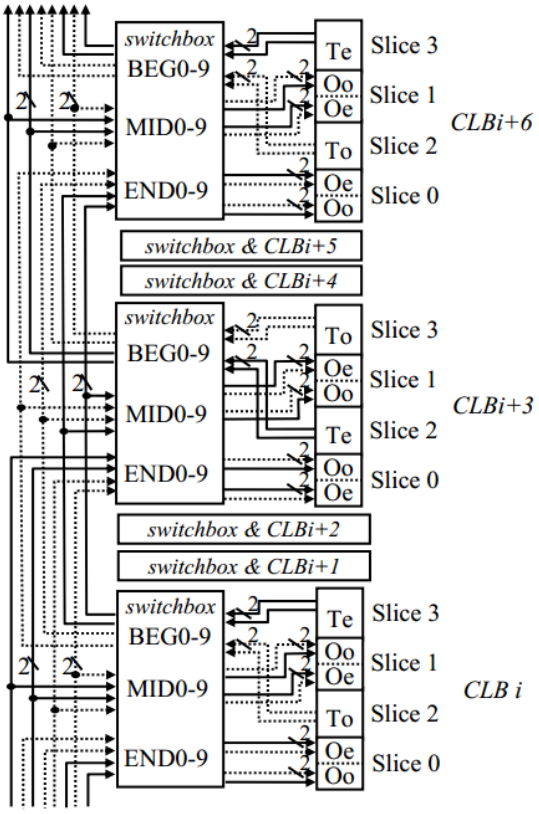


FIGURA 34 - ESTRATÉGIA DE IMPLEMENTAÇÃO EM LINHAS *HEX LINES* [18]

Observa-se que os *slices* que contêm o TPG têm seus sinais direcionados para o começo de uma linha *hex line* (BEG) no CLB i, este se propaga até o CLB i+3 (MID) onde será analisado por um ORA, e CLB i+6 (END) onde também será analisado por um ORA. Sendo assim, é possível diagnosticar com precisão o local da falha. Por exemplo, na Figura 36, pode-se visualizar uma arquitetura de teste de linhas *Double Lines*, em que os segmentos de fios são representados pelas letras A, B e C.

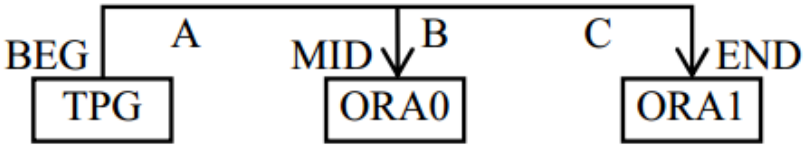


FIGURA 35 - ARQUITETURA DE FIOS *DOUBLE LINES* COM DISCRIMINAÇÃO DE FIOS [18]

Conforme o resultado indicado nos ORAs, pode-se identificar o local da falha. Ou seja, se apenas o ORA0 indicar erro no teste, a falha estará no segmento de fio B, se apenas o ORA1 indicar erro no teste, a falha estará no segmento de fio C. Caso tanto o ORA0, como o ORA1, apresentarem erros no teste, haverá uma falha em A. Esses resultados podem ser visualizados na Tabela 3.

ORA0	ORA1	Falha
OK	FALHA	C
FALHA	OK	B
FALHA	FALHA	A
OK	OK	Não há falha

TABELA 4 - DIAGNOSTICO DO LOCAL DA FALHA ATRAVÉS DE LEITURA DOS ORAS [18]

Esta estratégia foi implementada através de uma linguagem proprietária da *xilinx* conhecida como XDL (*Xilinx Database Language*). A partir dessa linguagem é possível ter todo o controle sob o *hardware* do FPGA, ou seja, controlam-se todos os segmentos de fios e suas interconexões. Na Tabela 4, pode-se visualizar a

quantidade de configurações de teste necessárias para diferentes modelos do FPGA Virtex4.

Observa-se que são necessárias pelo menos 2 configurações para cada sentido no teste de *double lines*.

RECURSO DE ROTEAMENTO	DIREÇÃO				TOTAL DE CONFIGURAÇÕES
	NORTE	SUL	LESTE	OESTE	
Double Lines para colunas de CLB	2	2	2	2	8
Double Lines para colunas de Não-CB	4	4	X	X	9 (1 extra)
Hex Lines para colunas de CLB	4	4	4	4	16
Hex Lines para colunas de Não-CLB, modelo LX	4	4	X	X	8
Hex Lines para colunas de Não-CLB, para modelo SX25/35	8	8	X	X	16
Hex Lines para colunas de Não-CLB, para modelo SX55	20	20	X	X	40
Long Lines para colunas de CLB	1	1	3	3	8
Long Lines Para colunas de Não-CLB	1	1	X	X	2
Total de Configurações	LX=51, SX25/35=59, SX55=83				

TABELA 5 - CONFIGURAÇÕES DE TESTE NECESSÁRIAS PARA TESTAR O FPGA VIRTEX4 [18]

Pode-se perceber que houve uma evolução nas estratégias quando considerado o número de WUTs e a cobertura de falha obtida. A primeira estratégia apresentada nesse capítulo [15] foi elaborada baseada em contador com paridade simples e foi implementada através de RSTARS, tornando-se eficaz para testes *on-line*. Porém, resultados obtidos mais tarde por [4] através de um *software* simulador de falhas provaram que uma estratégia baseada em contador com paridade simples apresentava uma cobertura de falha deficiente para aplicações de alta confiabilidade. Além disso, o diagnóstico do local da falha apresentado por [15] necessita de configurações de teste extras, aumentando o número de configurações necessárias para teste e o tempo para executa-lo.

Em 2004, foi proposto por [16] uma estratégia de teste para a rede de interconexão de FPGAs baseada em teste de Iddq diferencial e capaz de detectar falhas em ponte (*Bridge Faults*). Tal estratégia provou-se bastante eficaz, porém, necessita de uma grande quantidade de configurações de teste tornando-se um processo lento e inviável atualmente.

Em [4] foi feito um estudo de diferentes estratégias utilizadas na época a fim de se obter uma melhor cobertura de falha e quantidade de fios sob teste. Dentro das estratégias presentes no trabalho, encontram-se: Contadores, contadores com paridade simples e invertida, LFSR e CAR. Estas foram elaboradas em um nível lógico mais alto do que o *hardware* disponível nos FPGAs. Quando implementadas, as portas lógicas serão descritas em LUTs que apresentam limites em quantidade de sinais de entrada e *flip-flops* em número limitado. Dessa forma, além de uma boa cobertura de falha e quantidade de WUTs por configuração de teste, é necessário avaliar a viabilidade da síntese em FPGA. [4] conclui que a estratégia baseada em contador duplo com paridade invertida prova-se mais eficaz em cobertura de falha, quantidade de WUTs e viabilidade na implementação. [18] programa a estratégia escolhida por [4] em um FPGA *Xilinx* Virtex4 e provê detalhes de como foi sintetizado, quantidade de configurações necessárias e um método preciso para o diagnóstico do local da falha.

No próximo capítulo, será descrita a estratégia de teste para rede de interconexão elaborada nesse trabalho. A estratégia foi criada utilizando os menores componentes lógicos presentes no FPGA, ou seja, é uma estratégia que não precisa ser sintetizada, pois todos os componentes utilizados estão presentes da mesma forma no dispositivo, o que propicia uma maior viabilidade na implementação e maior números de WUTs. Para isso, utilizou-se registradores de deslocamento e LUTs, ambos presentes nos *Slices* do FPGA *Xilinx* Spartan3E.

ESTRATÉGIA PROPOSTA

3 ESTRATÉGIA PROPOSTA

Nesse tópico, será proposta uma estratégia de teste BIST para rede de interconexão de FPGAs Xilinx Spartan 3E baseada em registrador de deslocamento.

Como visto anteriormente, para aplicar um teste BIST A uma rede de interconexão, é necessário um gerador de padrões de teste (TPG) e um analisador de resposta (ORA). Nessa proposta, será utilizado um registrador de deslocamento para gerar os vetores de teste, ou seja, à medida que os dados estão sendo deslocados, são gerados novos vetores de teste. Para isso, convém utilizar as LUTs presentes no SLICEM na função de registrador de deslocamento com capacidade de 16 bits, representado por SRL16 (da expressão em Inglês, *Shift Register Length 16*), como visto na Figura 37.

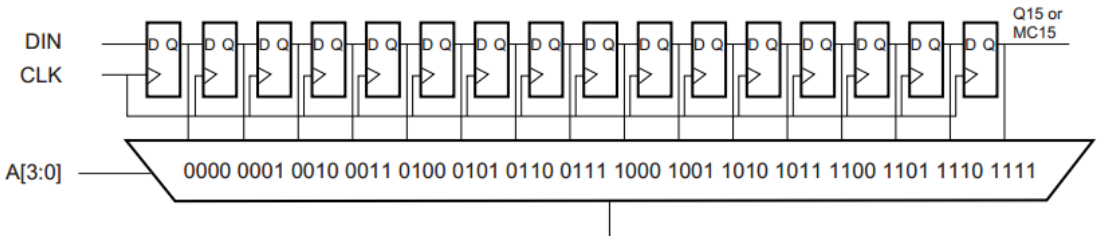


FIGURA 36- LUT CONFIGURADO COMO REGISTRADOR DE DESLOCAMENTO (SRL16) [12]

Na figura acima, podemos visualizar o conjunto de registradores conectados a um grande multiplexador, em que sua entrada de controle A[3:0] seleciona qual bit da cadeia de registradores será direcionado a saída, ou seja, um registrador de 16 bits com acesso a um de seus estados através do endereço fornecido na entrada A[3:0]. Observa-se que o último bit do estado do registrador está disponível, este está representado por Q15 ou MC15, bem como uma entrada DIN que alimenta o primeiro bit da cadeia de registradores.

Logo, para cada LUT configurada como SRL16, é possível obter duas fontes de sinal, uma saída D endereçada por A[3:0] e o último estado do registrador MC15. Como cada SLICEM contém dois SRL16, e cada CLB contém dois SLICEM, pode-se obter até 8 fontes de sinais independentes para controle do vetor de teste. Esta arquitetura pode ser visualizada na Figura 38.

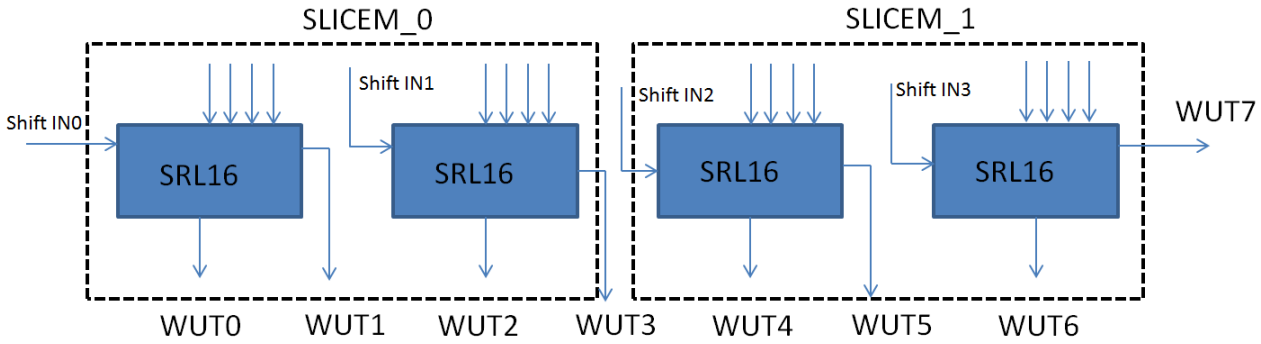


FIGURA 37 - ARQUITETURA DE UM TPG UTILIZANDO SRL16

Assim, pode-se carregar uma semente (valor inicial) em cada registrador SRL16 que corresponderá aos vetores de teste que serão enviados ao ORA. Observa-se que todos os registradores SRL16 apresentam uma entrada de dados *shift in*, onde é possível introduzir valores no registrador de deslocamento. Este será o principio para armazenar os resultados do teste. Ou seja, os vetores, outrora enviados pelo TPG, serão analisados pelo ORA, e o resultado do teste deslocado para esses registradores. Para analisar esses vetores de teste, serão utilizadas as outras LUTs, presentes nos outros dois SLICELs, conforme Figura 39.

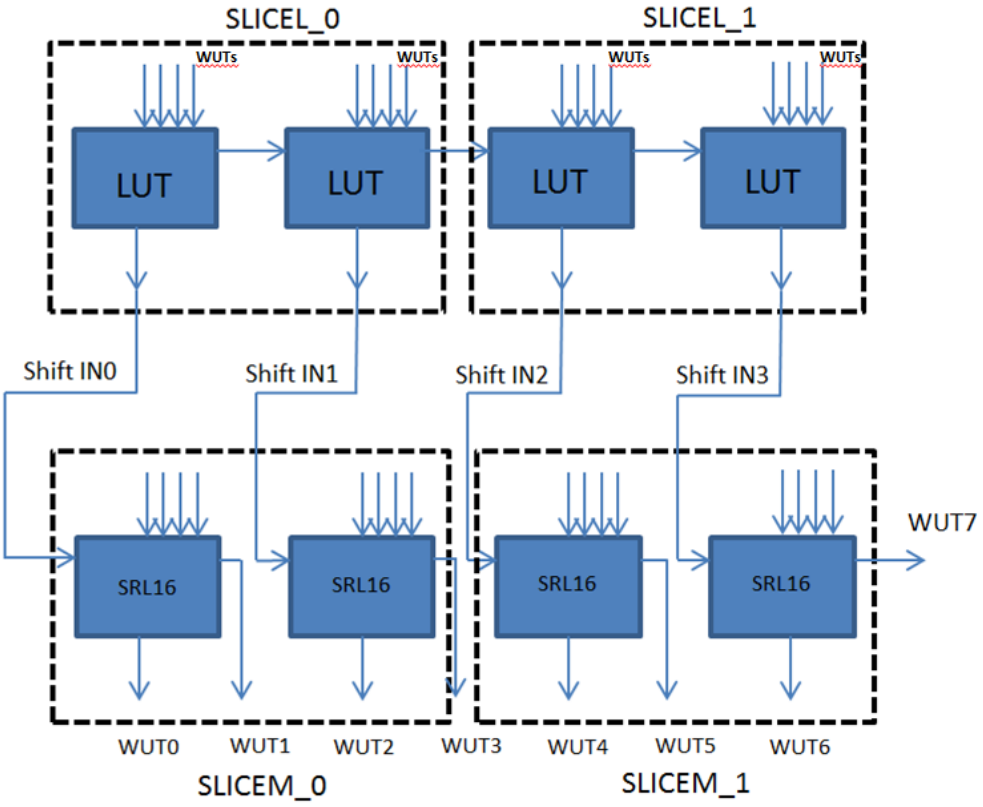
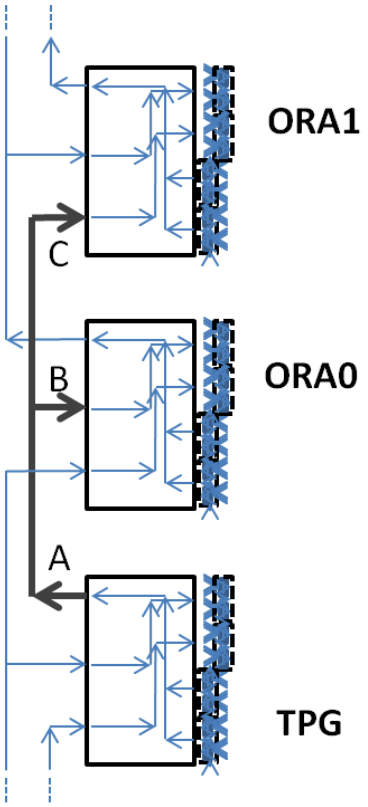


FIGURA 38 - ESTRATÉGIA BIST BASEADA EM SRL16

A partir dessa arquitetura, é possível observar a quantidade de fios que podem ser analisados pelo ORA. Como este se baseia em quatro LUTs, e cada LUT apresenta quatro entradas, é possível obter a leitura de até 16 WUTs. Observa-se que este valor corresponde a dois conjuntos de oito WUTs, logo, utiliza-se um SLICEL para analizar os 8 WUTs que chegam no CLB através da entrada MID e outro através da entrada END. Isso possibilita que se tenha duas leituras do mesmo WUT em posições diferentes, possibilitando o diagnóstico do local da falha, conforme visualizado na Figura 40 e Tabela 5.



ORA0	ORA1	Falha
OK	FALHA	C
FALHA	OK	B
FALHA	FALHA	A
OK	OK	Não há falha

TABELA 6 - ANALISE DA POSIÇÃO DA FALHA A PARTIR DOS RESULTADOS DOS ORAS

FIGURA 39 - DIAGNÓSTICO DO LOCAL DA FALHA

Ainda devido à modularidade em que os resultados são armazenados, é possível saber qual conjunto de fios e o lugar em que estes apresentaram falha. Por exemplo, se a última LUT que tem o resultado de seu teste enviados a um SRL16 (através do fio *shift in3*) apresentar erro, a falha encontra-se nesse conjunto de 4 fios analisados por essa LUT.

Esta estratégia, se comparada às outras apresentadas anteriormente em [4, 16, 17, 19, 20], demonstra melhorias significativas como maior quantidade de WUTs e diagnóstico mais preciso do local da falha. Teoricamente, seria necessário apenas uma configuração de teste para cada tipo de linha, *double lines* ou *hex lines*, e uma para cara direção, sendo assim, seriam necessárias 8 configurações de teste para testar todas as vias do tipo *double lines* e *hex lines*. Porém, devido a limitações na matriz de chaveamento, não é possível rotear todas as 8 fontes de sinais geradas pelo TPG para os WUTs. Apenas 7 fontes de sinais foram roteadas com sucesso, resultado melhor descrito no Capítulo 5.

Outro fator importante a se considerar nessa estratégia é o controle da quantidade de ciclos de *clock*, visto que o resultado do teste que está sendo direcionado para o registrador de deslocamento, após 16 cliclos de *clock* correspondente ao tamanho do registrador, poderá ser uma fonte de sinal para um WUT. Esta estratégia, com a falta de disponibilidade de um controle de *clock*, deve apresentar um comportamento cíclico, ou seja, deve-se carregar uma semente no registrador de deslocamento compatível com a função a ser realizada na LUT para que em algum momento do teste, esteja disponível uma assinatura válida no registrador. *A priori* foram utilizados os valores **0xAAAA** no **SRL G** e **0x5555** no **SRL F** como semente dos registradores SRL16 e as LUTs foram programadas com as seguintes funções **F = WUT7 ^ (WUT0 ^ (WUT2 ^ WUT6))** e **F = WUT1 ^ (WUT3 ^ WUT4)**, o SRL16 também foi configurado para que o sinal direcionado à saída seja equivalente ao último estágio do registrador, ou seja, sua entrada A[3:0] está conectada ao VCC e possui valor lógico **1111**, conforme Figura 41.

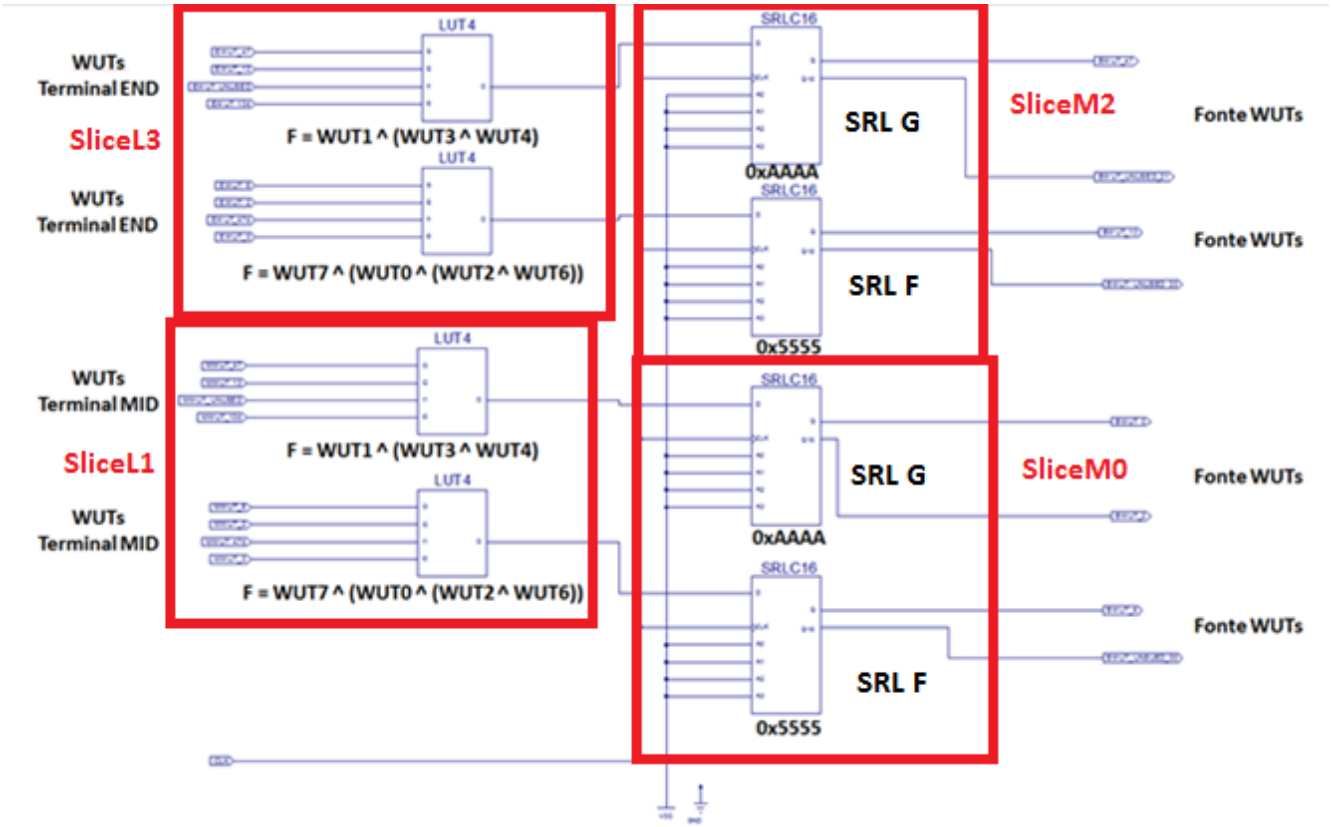


FIGURA 40 - ESTRATÉGIA PROPOSTA COM VALORES INICIAIS E FUNÇÕES LÓGICAS

Em geral, a quantidade de energia consumida por um dispositivo (corrente dinâmica) está diretamente relacionada ao número de chaveamentos realizados em um dado instante de tempo [19]. Considerando o comportamento cíclico dessa estratégia, podem-se deduzir valores iniciais dos registradores SRL16 energeticamente mais eficientes que apresentem menor consumo dinâmico. Resultados experimentais com outros vetores serão detalhados no Capítulo 5.

A estratégia de teste proposta aqui foi descrita em ASL e simulada com o *software* AUSIM. Os resultados podem ser visualizados na Figura 42.

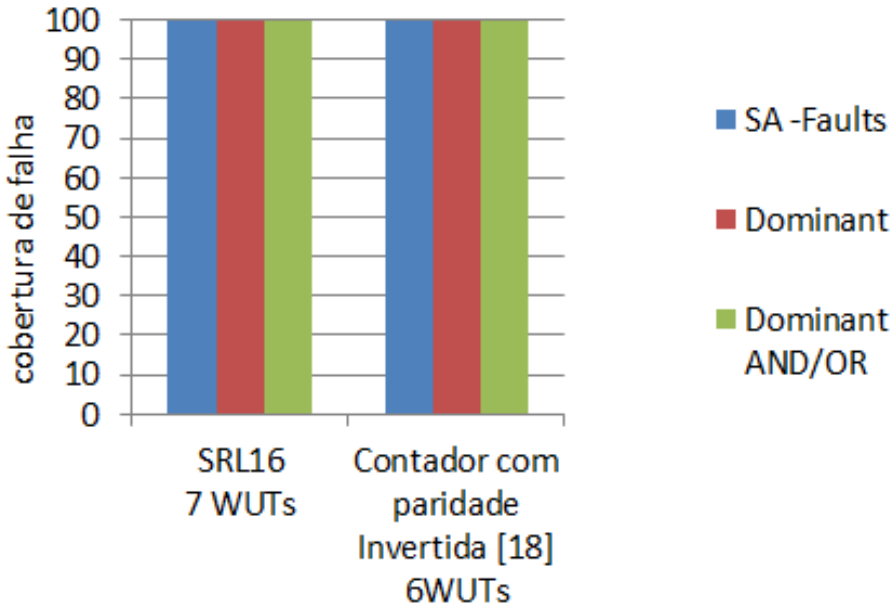


FIGURA 41 - RESULTADOS DE SIMULAÇÃO COM *SOFTWARE* AUSIM

Observa-se que para todas as falhas do tipo *Stuck-At* e falhas em ponte obteve-se uma cobertura de falha de 100% para os 7 fios roteados com sucesso, semelhante ao resultado obtido de um contador com paridade invertida capaz de testar 6 fios, implementado por [18]. Os resultados dessa simulação foram feitos considerando os limites da implementação no dispositivo, logo a quantidade de WUTs e cobertura de falhas são diferentes das relatadas por [4] no capítulo anterior.

No próximo capítulo, serão apresentadas as ferramentas utilizadas para implementação dessa estratégia em FPGA, e em seguida, os resultados.

4 MATERIAL E MÉTODO

4 MATERIAL E MÉTODO

A estratégia anterior foi implementada através de um esquemático de alto nível em um *software* da *Xilinx* denominado ISE 14.2, conforme Figura 41 apresentada no capitulo anterior. Através dessa implementação, foi possível simular a estratégia e obter dados sobre seu comportamento, bem como realizar injeção de falhas. O *software* utilizado pela *Xilinx* para fazer simulação lógica é denominado *iSim.* O resultado de uma simulação pode ser visualizado na Figura 43, onde se observa o comportamento cíclico dos vetores de teste.

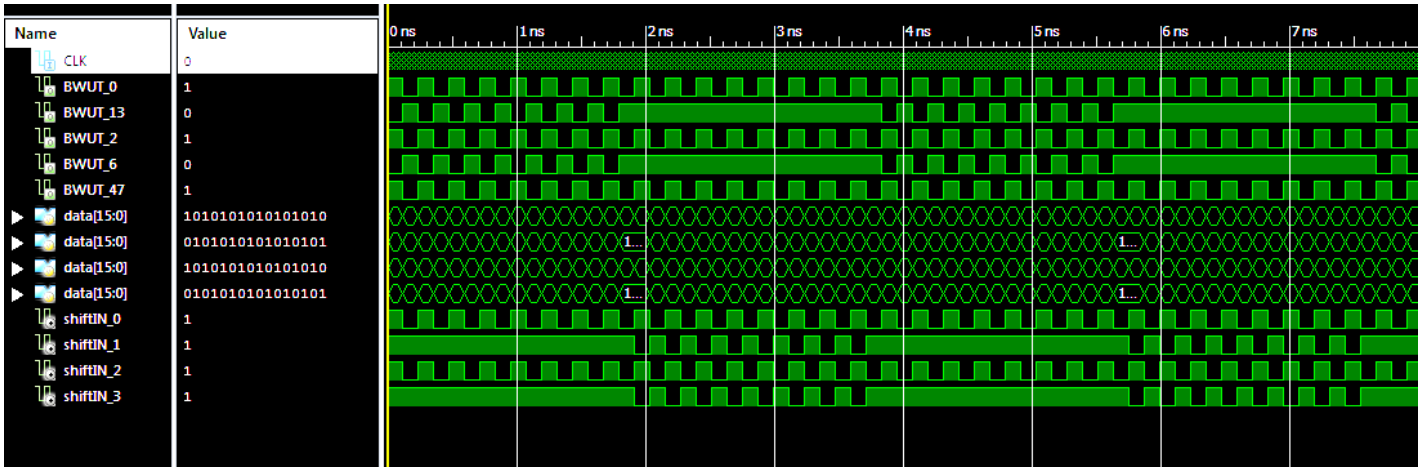


FIGURA 42 - SIMULAÇÃO COM O *SOFTWARE ISIM*

Para programar a estratégia proposta no capítulo anterior no dispositivo, é necessário utilizar uma ferramenta que seja capaz de fornecer controle específico sobre o *hardware* do FPGA. Essa ferramenta, para o caso de FPGAs da Xilinx, é conhecida como XDL (da expressão em Inglês, *Xilinx Design Language*) e será descrita a seguir.

O XDL provê uma interface entre o usuário e os recursos disponíveis no FPGA, por exemplo, componentes internos, CLBs, *Netlist*. Para obtenção de uma descrição destes recursos, pode-se utilizar o comando:

xdl –report –pips –all_conns xc6slx16

Após isso, é gerado um arquivo de descrição com as informações dos recursos do FPGA. Este arquivo é bastante extenso devido à forma com que estas

informações são escritas, ou seja, para cada CLB está associada uma descrição de seus recursos lógicos. Também é apresentado uma descrição da rede de interconexão demonstrando as conexões de entrada e saída em um dado CLB, bem como as possíveis conexões dos PIPs. Um exemplo do trecho do arquivo gerado pode ser observado na Figura 44.

```
# header
(xdl_resource_report v0.2 xc6slx16csg324-3 spartan6
# dimension
(tiles 73 62
 ...
 # configurable logic block with two slices
 (tile 4 6 CLEXL_X1Y61 CLEXL 2
   (primitive_site SLICE_X0Y61 SLICEL internal 45
    (pinwire A1 input L_A1)
    ...
    (primitive_site SLICE_X1Y61 SLICEX internal 43
    ...
    (pinwire D output XX_D)
  ...
 # interconnect tile
 (tile 4 5 INT_X1Y61 INT 1
   ...
   (wire EE2B0 2
     (conn CLEXM_X2Y61 CLEXM_EE2M0)
     (conn INT_BRAM_X3Y61 EE2E0)
    ...
   # switch matrxi multiplexers
   (pip INT_X1Y61 EE2E0 -> EE2B0)
   (pip INT_X1Y61 EE4E0 -> EE2B0)
 (pip INT_X1Y61 EL1E_S0 -> LOGICIN_B9)
  ...
# summary
(summary tiles=4526 sites=5378 sitedefs=46
   numpins=157962 numpips=5782505))
```

FIGURA 43- ARQUIVO DE DESCRIÇÃO EM XDL [20]

Para configurar o FPGA, é necessário escrever um arquivo de configuração. Nele estarão presentes informações em forma de texto que mais tarde serão transformadas em um conjunto de bits a serem carregados no FPGA. Um exemplo de arquivo de configuração pode ser visualizado na Figura 45.

```
1 design "clb_topo" xc3s100evq100-5 v3.2 ,
2   cfg "
3        _DESIGN_PROP::PK_NGMTIMESTAMP:1369330327";
4
5
6 inst "TPG0_X5Y21" "SLICEM",placed CLB_X5Y21 SLICE_X8Y40  ,
7   cfg " BXINV::BX BYINV::BY BYINVOUTUSED::#OFF BYOUTUSED::#OFF CEINV::#OFF
8        CLKINV::#OFF COUTUSED::#OFF CY0F::#OFF CY0G::#OFF CYINIT::#OFF CYSELF::#OFF
9        CYSELG::#OFF DIF_MUX::BX DIGUSED::#OFF DIG_MUX::BY DXMUX::#OFF
10       DYMUX::#OFF F:XLXI_11/SR3:#RAM:D=0xFEFD F5USED::#OFF FFX::#OFF FFX_INIT_ATTR::#OFF
11       FFX_SR_ATTR::#OFF FFY::#OFF FFY_INIT_ATTR::#OFF FFY_SR_ATTR::#OFF
12       FXMUX::F FXUSED::#OFF F_ATTR::SHIFT_REG G:XLXI_11/SR2:#RAM:D=0xFBF7
13       GYMUX::G G_ATTR::SHIFT_REG REVUSED::#OFF SHIFTOUTUSED::#OFF SLICEWE0USED::#OFF
14       SLICEWE1USED::#OFF SRFFMUX::#OFF SRINV::SR SYNC_ATTR::#OFF WF1USED::#OFF
15       WF2USED::#OFF WF3USED::#OFF WF4USED::#OFF WG1USED::#OFF WG2USED::#OFF
16       WG3USED::#OFF WG4USED::#OFF XBMUX::#OFF XUSED::0 YBMUX::0 YBUSED:B_WUT6_OBUF.SLICEM_YBUSED:0
17       YUSED::0 GMC15_BLACKBOX:SRL2_SR2.SLICEM_GMC15_BLACKBOX: WSGEN:XLXI_11/SR3.CE:
18       "
19  ;
20
21 ...
22
23 inst "ORA3_X5Y21" "SLICEL",placed CLB_X5Y21 SLICE_X9Y41  ,
24   cfg " BXINV::#OFF BYINV::#OFF CEINV::#OFF CLKINV::#OFF COUTUSED::#OFF CY0F::#OFF
25        CY0G::#OFF CYINIT::BX CYSELF::#OFF CYSELG::#OFF DXMUX::#OFF DYMUX::#OFF
26        F:LUT0:#LUT:D=((~A1*(A3*(~A2*A4)))+(A1*(~A3*(A2*~A4)))) F5USED::#OFF
27        FFX::#OFF FFX_INIT_ATTR::#OFF FFX_SR_ATTR::#OFF FFY::#OFF FFY_INIT_ATTR::#OFF
28        FFY_SR_ATTR::#OFF FXMUX::F FXUSED::#OFF G:LUT1:#LUT:D=((~A2*(A3*(A1*A4)))+(A2*((~A3*(A1*A4))+(A3*(A1@A4)))))
29        GYMUX::G REVUSED::#OFF SRINV::#OFF SYNC_ATTR::#OFF XBUSED::#OFF XUSED::0
30        YBUSED::#OFF YUSED::0 XORF:ORA3_X5Y21.XORF: "
31  ;
32
33 ...
34
35 net "SHIFT_IN0_X4Y21" ,
36   outpin "ORA3_X4Y21" Y ,
37   inpin "TPG2_X4Y21" BY ,
38   pip CLB_X4Y21 Y3 -> OMUX13 ,
39   pip CLB_X4Y21 OMUX13 -> BX3 ,
40   pip CLB_X4Y21 BX3 -> BY2 ,
41   pip CLB_X4Y21 BY2 -> BY_PINWIRE2 ,
42   ;
43
44 ...
45
```

FIGURA 44 - ARQUIVO DE CONFIGURAÇÃO EM XDL

A primeira linha descreve o nome do projeto, o modelo do FPGA que será utilizado e a versão do XDL utilizada (no caso v3.2). Observa-se que há sempre uma linha de configuração atrelada a cada componente lógico do XDL (representado por ***cfg***). Na linha 6, há uma instância de um módulo, que correspondem à configuração de um *slice*. Logo, para cada *slice* utilizado do FPGA, há uma instância e uma *string* de configuração associada. Essa *string* pode ser visualizada entre as linhas 7 e 18 e é utilizada para configurar todos os componentes internos do *slice*.

Entre as linhas 35 e 42 está a descrição de uma ligação. Pode-se observar que esta é formada por um pino de saída (***outpin***), um ou mais pinos de entrada

(***inpin***) e um conjunto de PIPs que quando ativados conduzem o sinal de um ponto a outro.

A *xilinx* utiliza como padrão para o arquivo binário, equivalente ao XDL escrito em linguagem de máquina, a extensão ***.ncd***. Logo, para converter de ***.ncd*** para ***.xdl***, utiliza-se o seguinte comando:

xdl –ncd2xdl design.ncd

Para converter de *.xdl* para *.ncd*, utiliza-se o comando:

xdl –xdl2ncd design.ncd

Nessa pesquisa, o arquivo *.ncd* é utilizado em outros *softwares* como BitGen e FPGA_editor. O FPGA_editor é utilizado para visualizar o arquivo *.ncd,* permitindo verificar as conexões realizadas no arquivo XDL. A Figura 46 mostra uma imagem obtida do *software* FPGA_editor feita a partir de um estudo de uma estratégia de teste para *double lines*. O *software* BitGen será descrito ainda nesse capítulo.

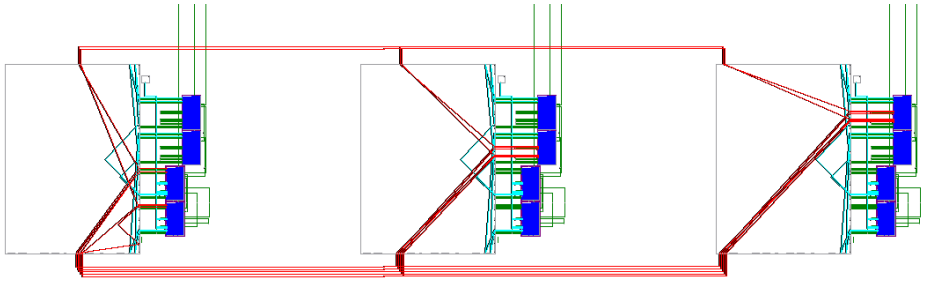


FIGURA 45 - CAPTURA DE TELA DO *SOFTWARE* FPGA EDITOR

Para cada configuração de teste, é necessário um arquivo XDL. Buscando facilitar a construção desses arquivos, foi desenvolvido um programa em C que manipula *strings* e formata os arquivos com as configurações necessárias para testar o FPGA. Esse programa em C foi desenvolvido no ambiente de programação ***eclipse*** e compilado através do compilador ***gcc***.

Após a geração dos arquivos XDL, é necessário convertê-los para o padrão ***.ncd*** e, na sequência, utilizar uma ferramenta chamada BitGen, que irá interpretar

esse arquivo ***.ncd*** e gerará um conjunto de bits que serão carregados no FPGA. Para essa ferramenta foi utilizado o seguinte comando:

bitgen –d -l -g Readback [DESIGN.NCD]

As opções –d, -l e –g *Readback* servem para instruir ao programa as seguintes funções:

- **-d**: Não fazer busca por erros de DRC (do inglês, *Design Rule Check*). Quando o *design* não tem fios totalmente roteados, utiliza-se essa opção para continuar com a geração do *Bitstream*.
- **-l**: Criar arquivo de alocação lógica (extensão .ll, do inglês, *logic location file*), que reporta como os bits estão distribuídos dentro do *Bitstream*. Este arquivo será bastante útil para interpretar os dados lidos do FPGA através do processo de *Readback*. Um exemplo do arquivo de alocação lógica pode ser visualizado na Figura 47.
- **-g Readback**: Esta opção serve para dizer ao programa que esta configuração que será carregada no FPGA pode ser lida, ou seja, habilita a possibilidade de *Readback*.

Após o uso do programa, serão criados os arquivos com extensão *.bit* (*Bistream*) e.ll (Alocação lógica).

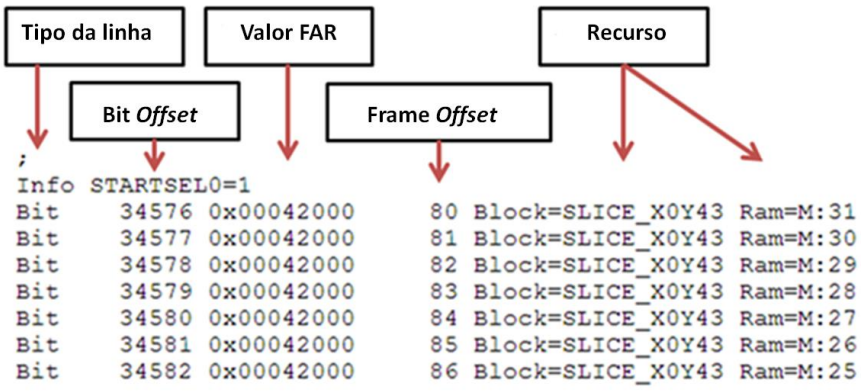


FIGURA 46 - ARQUIVO DE ALOCAÇÃO LÓGICA

No arquivo de alocação lógica, as informações estão distribuídas em cada linha do arquivo. Para cada linha, tem-se um tipo que dirá se o restante da linha representa algum recurso lógico ou apenas uma informação. Quando essa linha está descrevendo um recurso lógico, ela começa com a palavra “Bit”, que representará

um bit em um conjunto de Bits (*Bitstream*) extraídos do FPGA através do processo de *readback*, e a posição desse bit é a próxima informação apresentada na linha, descrita por Bit *Offset* na Figura 47.

O valor FAR (do inglês, *Frame Address Register*) representa o valor que está contido no registrador de endereço de quadro/janela, para isso, utiliza-se o termo em inglês denominado *frame*. Os dados contidos no *bitstream* estão organizados através de janelas, e dentro do FPGA, existe um registrador denominado FAR, que identifica em qual das janelas o programador está tentando escrever. Essa informação é importante quando se pretende utilizar reprogramação parcial do dispositivo, onde é possível escrever em apenas uma janela. O *frame Offset* identifica a posição do Bit a partir do ínicio da janela em que ele se encontra, e na sequência da linha está descrito o recurso atrelado a este determinado bit de configuração.

A partir do arquivo ***.bit,*** utiliza-se um *software* chamado **iMPACT** para programar e ler a memória do dispositivo (*readback)* através da interface JTAG. Antes de utilizar o *software* é necessário configurar duas variáveis de ambiente do sistema operacional: a variável **XIL_IMPACT_IGNORE_MASK_FILE=1,** que diz ao programa que ao verificar o dispositivo para saber se foi programado corretamente, o mesmo ignore um arquivo que mascara os bits que serão verificados, a depender do projeto isso é bastante útil, pois alguns possuem dados voláteis que não podem ser verificados. A outra variável é denominada **XIL_IMPACT_VIRTEX_DUMPBIN=1**, e diz ao programa para gerar um arquivo binário contendo o *bitstream* que foi lido do FPGA, consequentemente contendo os resultados do teste. Apesar de a variável ter o nome de outro modelo de FPGA (VIRTEX), funciona também para o Spartan 3E.

Após a configuração das variáveis de ambiente, utiliza-se o programa a partir do seguinte comando:

impact –batch

A opção *–batch* possibilita que o programa seja executado em linha de comando, onde o mesmo abre um terminal específico para ser utilizado. Em seguida, aplicam-se os seguintes comandos no terminal do iMPACT:

setmode –bs*:* o programa utiliza o modo de comunicação de *Boundary Scan*, sendo esse o utilizado para comunicação com o dispositivo através da interface JTAG.

setcable –p auto: configura o programa para utilizar o cabo de comunicação e encontrar em qual porta ele está conectado ao computador automaticamente.

identify: solicita ao programa para que ele identifique algum dispositivo que esteja conectado a interface JTAG. O mesmo retornará com o dispositivo e em qual posição ele se encontra dentro da cadeia JTAG.

assignfile –p 1 –file [Arquivo.bit]: Atribui um arquivo de configuração, ***Arquivo.bit***, para um determinado dispositivo na cadeia JTAG, identificado pela posição ***–p 1.***

program –p 1: Programa o dispositivo na posição 1 da cadeia JTAG.

verify –p 1: Verifica o dispositivo na posição 1 da cadeia JTAG, esse é o comando utilizado para ler a memória do dispositivo e gerar um arquivo ***.bin***, contendo o *bitstream* lido do FPGA. Nesse arquivo estarão os resultados do teste.

O programador se comunica com o dispositivo através de um cabo da *Xilinx* chamado *Platform Cable USB v2*, que pode prover essa conexão de algumas maneiras: JTAG e *slave serial*. Nesse projeto, optou-se por utilizar o modo JTAG devido à simplificação da construção da placa de testes, que será detalhada a seguir.

Para executar o teste no FPGA, foi desenvolvida uma placa que conta com um FPGA Spartan 3E xc3s100e e alguns outros componentes para que seja possível o seu funcionamento, uma visão do projeto dessa placa pode ser visualizado na Figura 48.

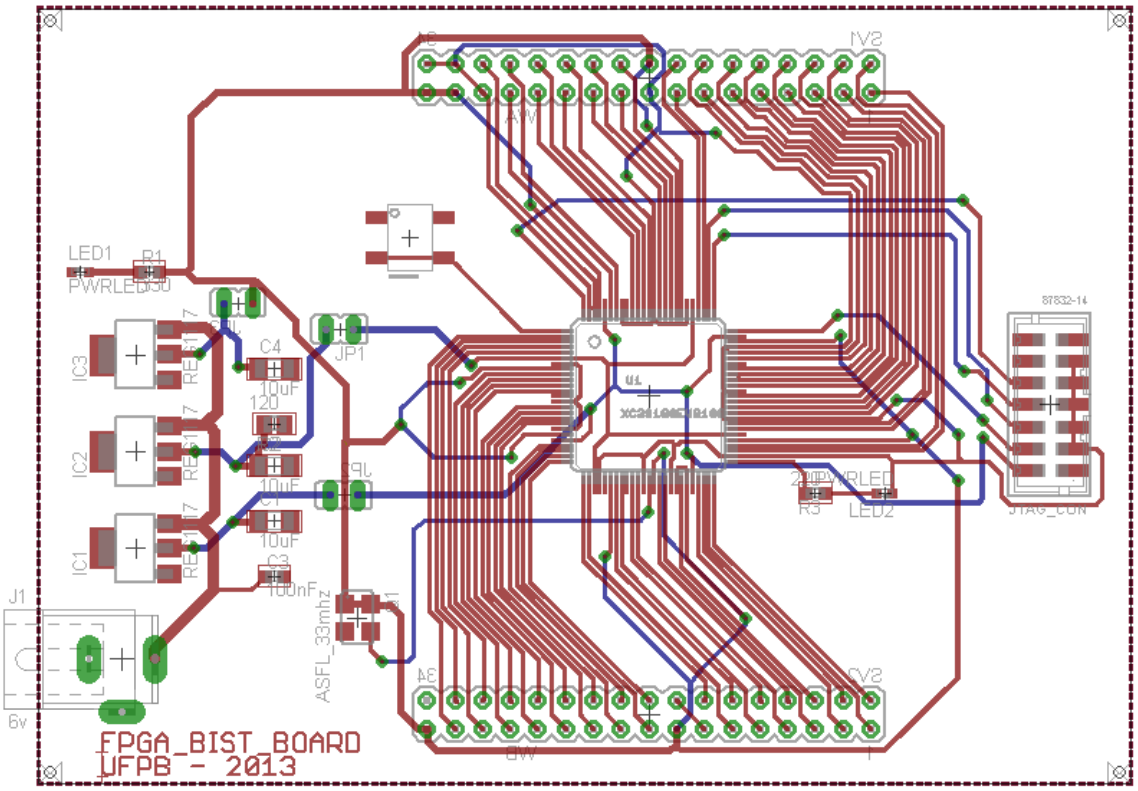


FIGURA 47 - PROJETO DA PLACA PARA TESTE DE FPGAs

Os CIs denominados por IC1, IC2 e IC3 na Figura 48 são reguladores de tensão, e provêm as tensões de 2.5v, 1.2v e 3.3v, respectivamente. O FPGA Spartan 3E utiliza alimentação de 1.2v para os sinais que trafegam internamente nos CLBs e na rede de interconexão, 2.5v como uma tensão de alimentação auxiliar que alimentará a interface de comunicação JTAG do C.I, e 3.3v para operar os pinos de entrada e saída. Para medir o consumo realizado pelo teste, convêm efetuar essa medição na tensão de 1.2v através do ponto de medição **JP1** (*jumper*)**.** A placa ainda conta com um gerador de *clock* a cristal de 33Mhz denominado **ASFL_33Mhz**, conectores de entrada e saída nas laterais da placa denominados **SV1** e **SV2,** e um conector para o cabo *Xilinx Platform Cable* denominado **JTAG_CON.**

Em geral, o fluxo adotado nesse trabalho para teste BIST em FPGAs pode ser visualizado na Figura 49.

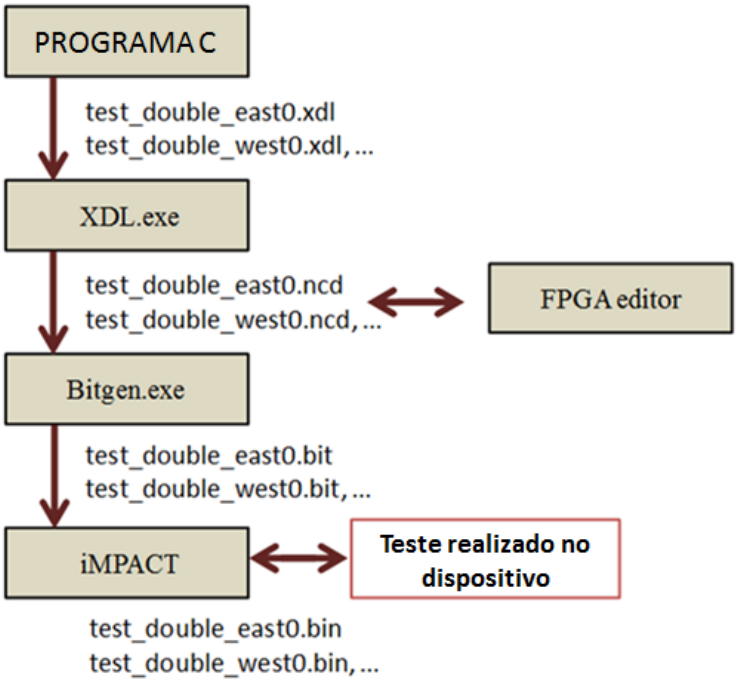


FIGURA 48 - PROCESSO DE TESTE BIST PARA FPGAs *XILINX*

Após gerar o arquivo ***.bin*** contendo o ***bitstream*** que foi lido do FPGA através do **iMPACT**, é necessário interpretá-lo. Para isso, foi desenvolvido um programa em C que procura e guarda o valor dos bits nas posições indicadas pelo arquivo de alocação lógica **.ll** e gera um arquivo ***.data***, que contém os valores que estão registrados nos SRL16. Um exemplo de um arquivo ***.data*** pode ser visualizado na Figura 50. Observa-se que nesse arquivo, cada linha contêm 8 bits, e para cada SRL16 são necessários duas linhas, logo, cada *slice* apresentam 4 linhas, duas equivalem a cada SRL16 neles contidos.

```
Bit 34576 | CLB X1Y22 - SLICE_X0Y43 - 32 => READ - 0xAA - INV - 0x55 - 01010101
Bit 34584 | CLB X1Y22 - SLICE_X0Y43 - 24 => READ - 0xAA - INV - 0x55 - 01010101
Bit 34592 | CLB X1Y22 - SLICE_X0Y43 - 16 => READ - 0x00 - INV - 0xFF - 11111111
Bit 34600 | CLB X1Y22 - SLICE_X0Y43 - 8 => READ - 0x15 - INV - 0xEA - 11101010
Bit 34608 | CLB X1Y22 - SLICE_X0Y42 - 32 => READ - 0xAA - INV - 0x55 - 01010101
Bit 34616 | CLB X1Y22 - SLICE_X0Y42 - 24 => READ - 0xAA - INV - 0x55 - 01010101
Bit 34624 | CLB X1Y22 - SLICE_X0Y42 - 16 => READ - 0x00 - INV - 0xFF - 11111111
Bit 34632 | CLB X1Y22 - SLICE_X0Y42 - 8 => READ - 0x15 - INV - 0xEA - 11101010
Bit 34640 | CLB X1Y21 - SLICE_X0Y41 - 32 => READ - 0xAA - INV - 0x55 - 01010101
Bit 34648 | CLB X1Y21 - SLICE_X0Y41 - 24 => READ - 0xAA - INV - 0x55 - 01010101
Bit 34656 | CLB X1Y21 - SLICE_X0Y41 - 16 => READ - 0x00 - INV - 0xFF - 11111111
Bit 34664 | CLB X1Y21 - SLICE_X0Y41 - 8 => READ - 0x15 - INV - 0xEA - 11101010
Bit 34672 | CLB X1Y21 - SLICE_X0Y40 - 32 => READ - 0xAA - INV - 0x55 - 01010101
Bit 34680 | CLB X1Y21 - SLICE_X0Y40 - 24 => READ - 0xAA - INV - 0x55 - 01010101
Bit 34688 | CLB X1Y21 - SLICE_X0Y40 - 16 => READ - 0x00 - INV - 0xFF - 11111111
Bit 34696 | CLB X1Y21 - SLICE_X0Y40 - 8 => READ - 0x15 - INV - 0xEA - 11101010
```

FIGURA 49 - VISUALIZAÇÃO DO ARQUIVO *.data*

A estratégia utilizada nesse trabalho para validar o teste em FPGA foi baseada em uma comparação com o modelo desenvolvido em alto nível, logo, os valores apresentados no arquivo .***data*** devem corresponder aos valores obtidos na simulação com o *software iSim.* Todos os valores lidos do FPGA corresponderam aos valores simulados, sendo assim, a estratégia de teste apresentada por esse trabalho foi validada em FPGA e os resultados e discussões acerca dessa estratégia serão detalhados no próximo capítulo.

5 RESULTADOS E DISCUSSÃO

5 RESULTADOS E DISCUSSÃO

Essa estratégia baseada em registrador de deslocamento foi implementada através do XDL. A partir disso, foi possível observar limitações na matriz de chaveamento, que por sua vez não permitiu que todos os 8 sinais gerados pelo TPG fossem roteados para as linhas que precisam ser testadas. Foram roteados com sucesso 7 desses sinais e dois tiveram que ser distribuídos a partir de uma mesma fonte, conforme Figura 51.

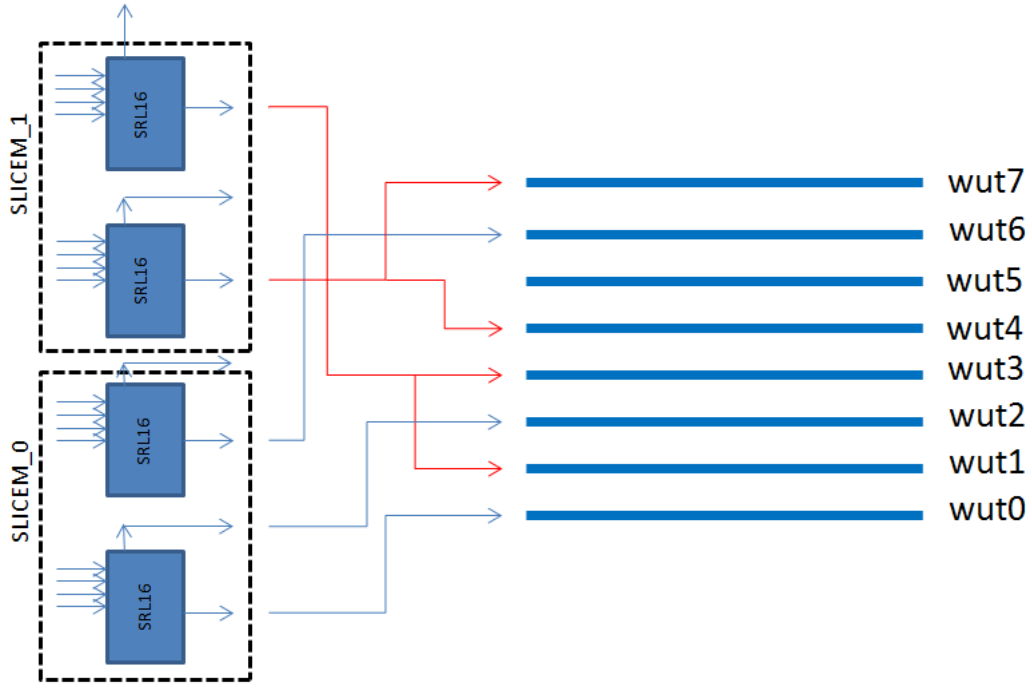


FIGURA 50 - ROTEAMENTO DOS WUTS CONSIDERANDO OS LIMITES DA MATRIZ DE CHAVEAMENTO

Devido a essa limitação, a quantidade de configurações de teste necessárias dobrou, ou seja, são necessárias 8 configurações para testar 7 dos 8 WUTs, e uma configuração adicional para testar apenas 1 WUT (WUT5, conforme figura acima).

Outra limitação encontrada foi em relação às colunas presentes no dispositivo que não eram compostas por CLBs e sim por outros blocos lógicos como multiplicadores e memórias, sendo necessário rotear o teste dessas linhas no sentido Norte e Sul através de uma coluna vizinha. Isso resulta em duas configurações de teste extras para cada sentido e para cada tipo de linha. No total,

são necessárias 8 configurações para testar linhas não-CLBs. Essas configurações podem ser visualizadas na Tabela 6.

Recurso De Roteamento	Direções				Qt. De Configurações
	NORTE	SUL	LESTE	OESTE	
Double Lines	2	2	2	2	8
Hex Lines	2	2	2	2	8
Double Lines Não-CLBs	2	2	0	0	4
Hex Lines Não-CLBs	2	2	0	0	4
Total	**Spartan 3e xc3s100e**				**24**

TABELA 7 - CONFIGURAÇÕES DE TESTE NECESSÁRIAS PARA TESTAR COMPLETAMENTE O FPGA SPARTAN 3E XC3S100E

Um bloco multiplicador, quando está localizado em um terminal END de uma linha, deve- se tornar um novo BEG para que seja possível fornecer sinais ao ORA do CLB vizinho e assim garantir o teste de todos os segmentos de fios, além de manter a consistência dos vetores na linha. Logo, pode-se observar na Figura 52, as conexões END para BEG em um bloco que não é CLB.

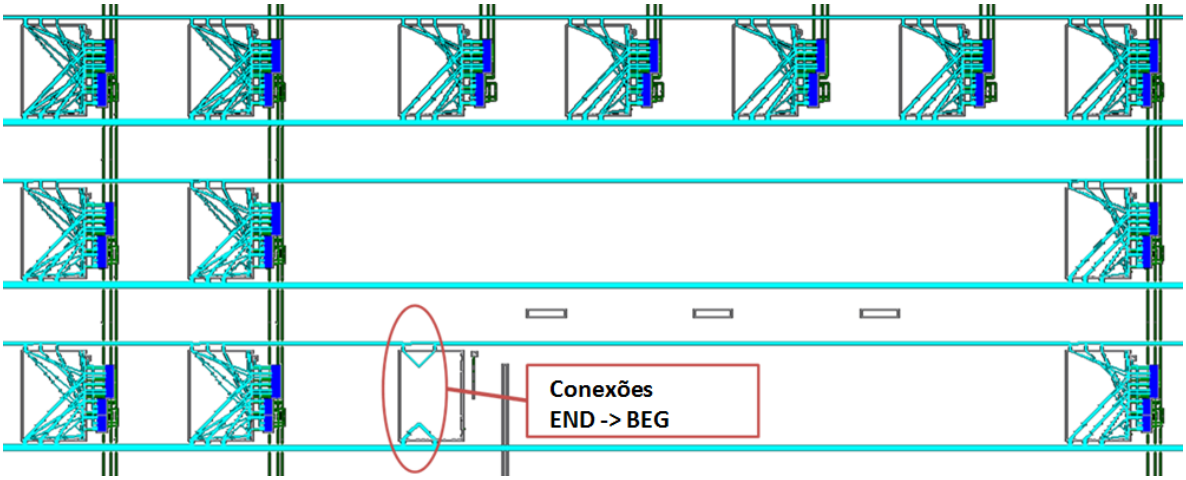


FIGURA 51 - CONEXÕES END PARA BEG EM UM BLOCO NÃO-CLB

Outra consideração a ser feita está relacionada aos CLBs que estão localizados na primeira linha, pois não possuem TPGs anteriores que possam prover sinais em suas conexões MID e END, e CLBs localizados na segunda linha, pois não possuem conexões END. Logo, foi desenvolvida uma rede de realimentação utilizando um bloco de entrada e saída (IOB) e os blocos terminadores de cada linha, conforme visto na Figura 53.

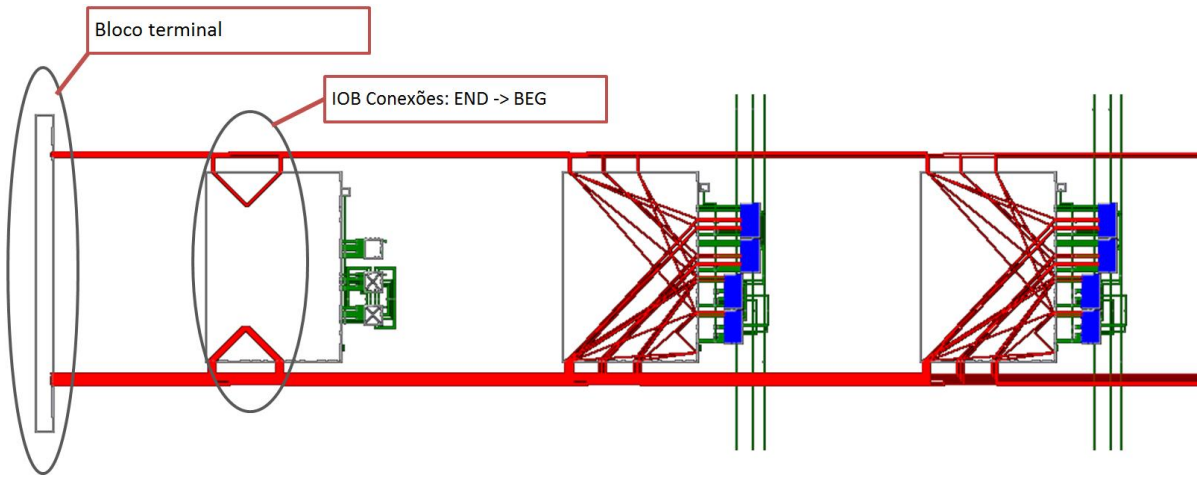


FIGURA 52 - REALIMENTAÇÃO INICIAL PARA O PRIMEIRO E SEGUNDO CLB DA LINHA

Os blocos de terminação, presentes em cada linha, provêm uma conexão entre os fios que chegam em um sentido e são direcionados para o sentido contrário. Por exemplo, um fio que parte do IOB (BEG) em sentido oeste encontra o bloco terminal e por meio de ativação de um PIP, esse sinal é redirecionado para o fio no sentido leste, encontrando o mesmo IOB na posição MID e o primeiro CLB da linha em END. Os IOBs tiveram suas conexões END para BEG acionadas em ambos os sentidos, quando a conexão END para BEG acontece no mesmo sentido da linha, no caso leste, ele provê sinais nas conexões MID do primeiro CLB da linha e END para o segundo CLB da linha. Quando a conexão é realizada no sentido contrário à linha, no caso oeste, esse sinal será retransmitido para o terminal END do primeiro CLB da linha através da conexão realizada em um bloco terminal.

Observa-se também que essa implementação baseada em registrador de deslocamento propicia o uso futuro de estratégias baseadas em LFSR. O LFSR é

uma estrutura bastante utilizada como gerador de padrões e análise de assinatura em teste de circuitos integrados [7]. Tal estrutura é governada por um polinômio primitivo que pode ser trabalhado para aplicar melhores vetores de teste, que podem propiciar uma melhor dissipação de potência e maior cobertura de falha.

Foram simulados e realizados experimentos com diferentes valores iniciais nos registradores SRL16 a fim de se obter diferentes valores de corrente de consumo. A Tabela 7 relaciona diferentes valores iniciais para os registradores SRL16, cobertura de falha para cada combinação (obtidas através do simulador de falha AUSIM) e corrente dinâmica consumida pelo dispositivo, medida através de um múltimetro Agilent modelo U1442B**,** sob a tensão de 1.2V.

Nº combinação	Valores iniciais para SRL16	Cobertura de falha	Corrente dinâmica
1	LUT G: 0xAAAA LUT F: 0x5555	100% Stuck-AT e Falhas em ponte	43.82 mA
2	LUT G: 0xFAFA LUT F: 0x5F5F	100% Stuck-AT e Falhas em ponte	35.13 mA
3	LUT G: 0xFAFA LUT F: 0xF5F5	100% Stuck-At e Falhas em ponte	30.22 mA
4	LUT G: 0xFFFA LUT F: 0xFFF5	100% Stuck-AT e Falhas em ponte	20.42 mA

TABELA 8 - ANALISE DE CONSUMO DE CORRENTE DINÂMICA PARA DIFERENTES COMBINAÇÕES DE VALORES INICIAIS DOS REGISTRADORES SRL16

Para cada combinação de valor inicial no SRL16, foi feita uma simulação com o *software* iSim. Através disso, foi possível visualizar o comportamento cíclico dos vetores e a quantidade de chaveamento nos WUTs. Essas simulações podem ser visualizadas nas Figuras 54, 55, 56 e 57. A primeira linha corresponde ao sinal de *clock*, da segunda até a quinta linha estão relacionadas as 5 fontes de sinais para os 7 WUTs, nas próximas quatro linhas estão os valores internos dos registradores SRL16 e as últimas quatro linhas representam o sinal de *ShiftIN*, que liga a saída da LUT a entrada de dados do registrador SRL16.

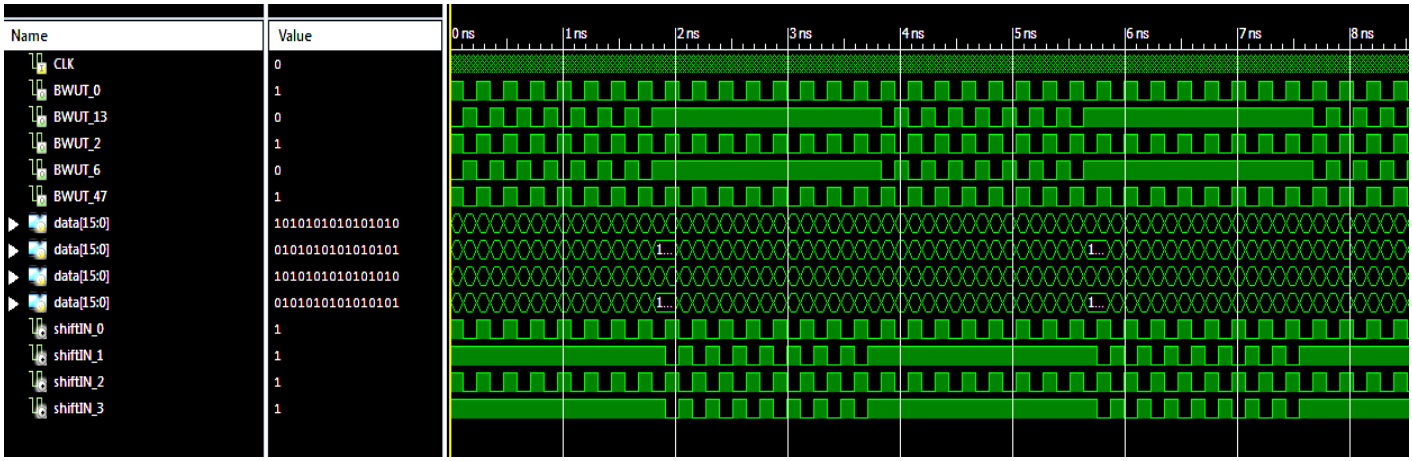


FIGURA 53 - RESULTADO DA SIMULAÇÃO UTILIZANDO VALORES DA COMBINAÇÃO 1

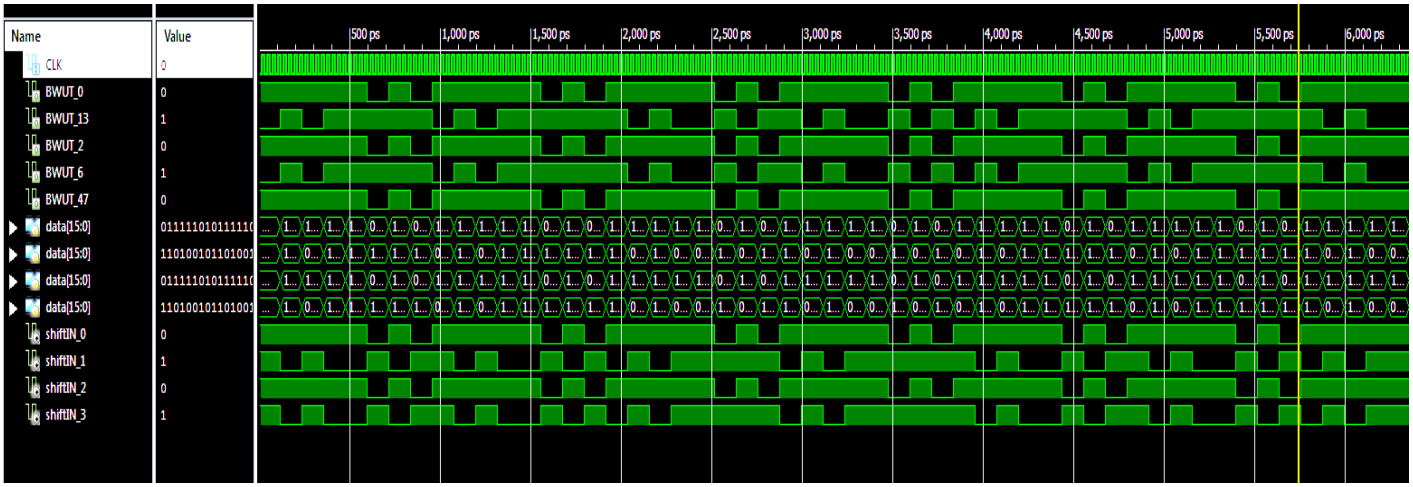


FIGURA 54 - RESULTADO DA SIMULAÇÃO UTILIZANDO VALORES DA COMBINAÇÃO 2

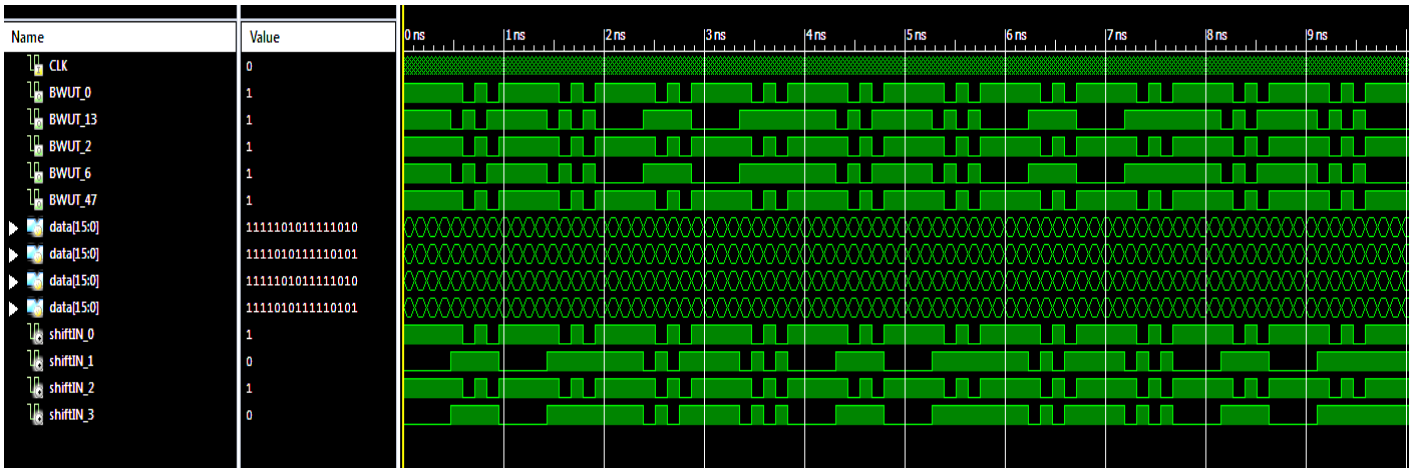


FIGURA 55 - RESULTADO DA SIMULAÇÃO UTILIZANDO VALORES DA COMBINAÇÃO 3

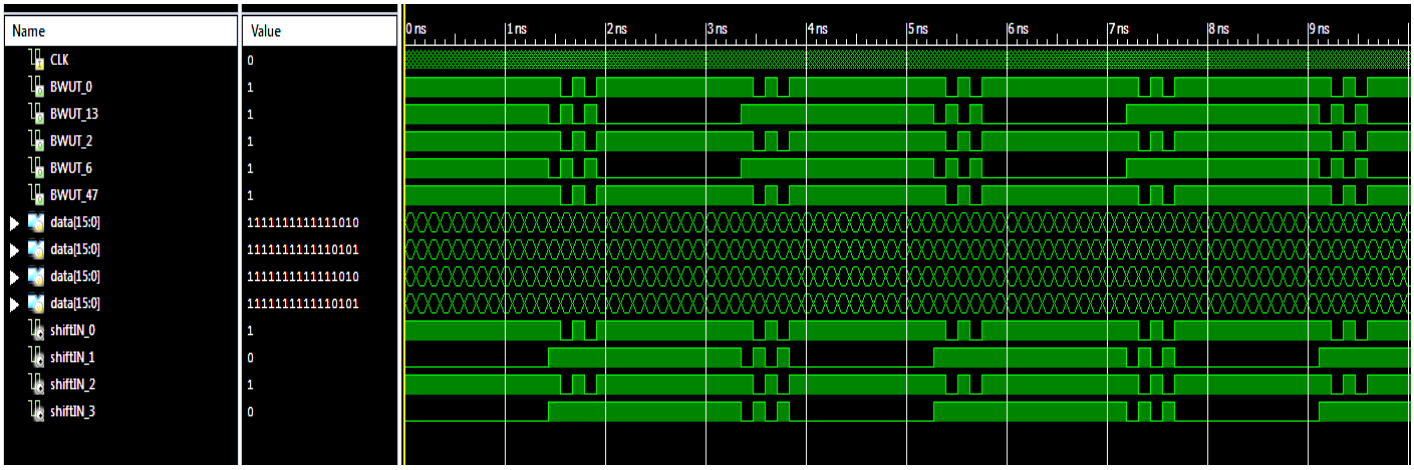


FIGURA 56 - RESULTADO DA SIMULAÇÃO UTILIZANDO VALORES DA COMBINAÇÃO 4

As figuras acima comprovam que a combinação que apresenta menor consumo de corrente dinâmica apresenta uma quantidade menor de chaveamento nos WUTs, e ainda assim consegue manter a consistência dos vetores de teste, garantindo uma cobertura de falha de 100% para o tipo Stuck-AT e falhas em ponte.

Outro ponto importante a se considerar para testes orientados às fábricas são os *multiple tests site*, ou seja, campos de teste em que podem ser testados vários FPGAs de uma vez. Isso só é possível devido a estratégias BIST, que não necessitam de um ATE, ou seja, testando vários FPGAs por vez, pode-se diminuir o tempo de teste. Para isso, é necessário um controlador de teste e estratégias que possam automatizá-lo.

6 CONCLUSÕES

6 CONCLUSÕES

O trabalho desenvolvido e implementado nesta pesquisa trata de uma proposta de uma arquitetura de teste para a rede de interconexão de FPGA utilizando uma estratégia baseada em registrador de deslocamento. Um foco específico é dado as linhas *double lines* e *hex lines*, que existem nas quatro direções (norte, sul, leste e oeste) e perpassam por 3 CLBs. Cada matriz de chaveamento apresenta conexão com 24 segmentos de fios do tipo *double line*. Estes podem ser classificados como BEG, MID e END, num total de 8 segmentos de fios para cada classificação. O mesmo acontece com as linhas *hex line*.

As primeiras estratégias de teste estudadas neste trabalho e desenvolvidas para FPGAs eram pouco eficientes, no sentido de cobertura de falha e tempo de teste, além de serem desenvolvidas em um nível de abstração maior, necessitando de síntese lógica, e utilizavam recursos que se suponham bons, como a rede de interconexão interna do CLB.

A estratégia proposta nesse trabalho foi desenvolvida através de uma análise dos recursos disponíveis no FPGA e apresenta melhorias significativas no tempo de teste devido à quantidade de configurações reduzidas, e precisão no diagnóstico da falha, além de que não utiliza a rede de interconexão interna, ou seja, não é projetada em cima de recursos que se assumem estarem bons. Essa estratégia é capaz de encontrar falhas em ponte e do tipo *stuck-at*. Além dessas, também é possível identificar, por meio de falha equivalente, falhas nos PIPs, como: *PIP stuck-on* e *PIP stuck-off.*

Devido a limitações existentes na matriz de chaveamento, foi possível rotear 7 dos 8 WUTs propostos com uma configuração, logo, a quantidade de configurações necessárias para realizar o teste em todo FPGA não pode ser reduzida, pois, ainda assim, é necessário uma configuração extra para testar um fio, ou seja, ainda são necessárias 16 configurações de teste para testar todas as linhas *double lines* e *hex lines,* e 8 configurações para testar as linhas *Double lines* e *Hex lines* em colunas de não-CLBs, totalizando 24 configurações de teste, em contraste com trabalhos anteriores que utilizaram 41 configurações de teste para testar todas as linhas *Double Lines e Hex lines* [18]. Além disso, houve um avanço no que diz

respeito à precisão no diagnóstico de falhas, sendo possível identificar com resolução mínima de 4 fios, o conjunto que apresentou falha.

A estratégia de teste proposta nesse trabalho foi validada em *hardware* através do FPGA *Xilinx* Spartan 3E xc3s100e, para isso, foram comparados os resultados de simulação do *software* iSim com os resultados obtidos do FPGA através do processo de *readback*. Houve equivalência total dos valores e a estratégia provou-se funcional no dispositivo.

Este trabalho pode ser estendido para o teste de outras linhas presentes no FPGA, como: *Long lines* e *Omux lines.* Bem como, esta estratégia pode ser aplicada para testes *online*, ou seja, utilizar linhas e colunas específicas para realizar o teste, enquanto o restante do FPGA permanece operacional para executar qualquer função.

REFERÊNCIAS

[1] Moore, G. Cramming More Components onto Integrated Circuits. **Proceedings** *of the IEEE*, vol. 86, no. 1, pp. 82-85, 1998.

[2] Stroud, E. C. **A Designer´s Guide to Built-in Self-Test**. *Boston: Springer*, 2002.

[3] Avizienis, A. Toward Systematic Design of Fault-Tolerant Systems. **IEEE Transactions on Computer***,* v. 30, p. 51-58, Apr 1997.

[4] Dixon, B. **Built-In Self-Test of The Programmable Interconnect In Field Programmable Gate Arrays**. *M.S. Thesis, Auburn University, Auburn, AL,* 2008.

[5] Zhang, S.; Choi, M.; Park, N. e Lombardi, F.. Probabilistic balancing of fault coverage and test cost in combined built-in self-test/automated test equipment testing environment. **Proceedings** of 19th IEEE International Symposium on Defect and Fault Tolerancein VLSI Systems (dft 2004), p. 48–56, october 2004.

[6] Ungar, L.Y. e Ambler, T. Economics of built in seflt-test. ***IEEE Design & Test of Computers****, v. 18, n.5, p.70-79, sept-oct 2001.*

[7] Souza, Cleonilson P. **Uma Arquitetura Autotestável para Circuitos Digitais baseada no Algoritmo de Berlekamp-Massey e em Sistemas Imunológicos Artificiais**. *Tese de Doutorado UFCG. Campina Grande*. 2005**.**

[8] Jas, A.; Krishna, C.V. e Touba, N. A Weighted pseudorandom hybrid BIST. **IEEE transactions on Very Large Scale Integration (VLSI) Systems**, p. 1277 – 1283, Dec 2004.

[9] Wang, L.; Stroud, C. e Touba, N. **System On Chip Test Architectures: Nanometer Design for Testability**, *Amsterdam: Elsevier*, 2007.

[10] Stroud, C.; Bailey, J.; Emmert, J.; Nickolic, D.; e Chhor, K. Bridging Fault Extraction from Physical Design Data for Manufacturing Test Development, **Proceedings**. *IEEE International Test Conf*. pp. 760-769, 2000.

[11] **"Spartan 3 Family Datasheet"**, Family Datasheet DS099, Xilinx, Inc., 2013. Disponibilizado em http://www.xilinx.com/support/documentation/data_sheets/ds099.pdf.

[12] **"Spartan 3E User Guide"**, User Guide UG331, Xilinx, Inc., 2011. Disponibilizado em http://www.xilinx.com/support/documentation/user_guides/ug331.pdf.

[13] Chmelar, E., Minimizing the number of test configurations for FPGAs, **Computer Aided Design, 2004**. ICCAD-2004. IEEE/ACM International Conference on , vol., no., pp.899,902, 7-11 Nov. 2004

[14] **"Spartan 3E Datasheet"**, Datasheet DS312, Xilinx, Inc., 2013. Disponibilizado em http://www.xilinx.com/support/documentation/data_sheets/ds312.pdf.

[15] Stroud, C.; Nall, J.; Lashinsky, M.; Abramovici, M., BIST-based diagnosis of FPGA interconnect, Test Conference, 2002. **Proceedings**. International , vol., no., pp.618,627, 2002

[16] Chmelar, E., "**The test and Diagnosis of FPGAs**", *Dissertation for degree on Doctor of Philosophy*, June 2004.

[17] Shang, L., Kaviani, A., and Bathala, K., Dynamic power consumption in Virtex-II FPGA family, **Proceedings**. Int. Symp. FPGAs, pp. 157–164, 2002.

[18] Yao, J.; Dixon, B.; Stroud, C.; Nelson, V., System-level Built-In Self-Test of global routing resources in Virtex-4 FPGAs, ***System Theory, 2009. SSST 2009****. 41st Southeastern Symposium on* , vol., no., pp.29,32, 15-17 March 2009

[19] Girard, P., Low power testing of VLSI Circuits: Problems and Solutions, **Proceedings**. *IEEE 2000 First International Symposium on Quality Electronic Design, 2000.*

[20] Beckhoff, C.; Koch, D.; Torresen, J., The Xilinx Design Language (XDL): Tutorial and use cases, ***Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)****, 2011 6th International Workshop on* , vol., no., pp.1,8, 20-22 June 2011.