Inserção Automática de Técnicas de Tolerância a Falhas em Descrições VHDL[1]

Ana Carla dos Oliveira Santos, Sérgio Vanderlei Cavalcante

{acos, svc}@cin.ufpe.br
Universidade Federal de Pernambuco - UFPE
Centro de Informática
Cx. Postal 7851 CEP 50732-970
Recife – PE – Brasil

Resumo

À medida que os usuários dependem mais do desempenho de máquinas, uma falha no funcionamento destas causa um prejuízo mais visível e mais grave. O uso de técnicas de tolerância a falhas pode aumentar a confiabilidade dos sistemas amenizando, assim, esse problema. Esse documento apresenta o desenvolvimento de uma ferramenta para inserir técnicas de tolerância a falhas em sistemas de hardware descritos em VHDL.

1 Introdução

Sistemas de computação vêm sendo mais empregados a cada dia, atingindo um maior número de usuários, que passam a depender mais fortemente do desempenho desses sistemas. À medida que mais pessoas são beneficiadas pelas máquinas, maior pode ser o prejuízo causado por problemas ocorridos no funcionamento destas.

Dessa forma, torna-se necessária a utilização de mecanismos para lidar com os problemas que potencialmente possam afetar o bom funcionamento dos sistemas. Tolerância a falhas é um desses mecanismos. Tolerar as falhas do sistema, implica em reconhecer que estas são inevitáveis e oferecer alternativas que permitam ao sistema manter o funcionamento desejado mesmo na ocorrência destas.

O desenvolvimento de sistemas embutidos vem sendo realizado de forma cada vez mais sistemática seguindo metodologias mais elaboradas de projeto. O desenvolvimento de técnicas de síntese de alto-nível e, agora mais recentemente, o emprego de metodologias de hardware/software co-design[2] vêm tornando o projeto de sistemas embarcados mais maduro e conseqüentemente diminuindo o tempo para comercialização dos produtos e aumentando a qualidade destes.

Uma forma de amadurecimento de metodologias é a utilização de ferramentas que auxiliem o projetista durante o desenvolvimento do produto. Uma vantagem do uso de ferramentas é que, geralmente, estas são bem testadas e robustas, e desse modo evitam a inserção de erros de projeto nas etapas em que atuam, que se tornam automáticas e corretas por construção.

Ferramentas automáticas envolvendo o projeto de hardware tolerante a falhas vêm sendo estudadas já há algum tempo. A ferramenta MEFISTO[4] por exemplo, foi elaborada para injetar automaticamente falhas em modelos VHDL, com o intuito de validar o mecanismo de tolerância a falhas utilizado. A injeção de falhas pode ser feita pela ferramenta por duas abordagens distintas: modificação do modelo VHDL, através da inserção de componentes sabotadores ou mutação dos compoonentes existentes; ou utilização de comandos nativos dos simuladores, para inserir erros nos sinais de saídas dos módulos. Outro

1 VHDL – Very High Speed Integrated Circuit Hardware Description Language

trabalho relacionado é a inserção automática de BIST (Built-In Self Test) em descrições VHDL, para aumento da testabilidade dos componentes, como apresentado em [5] .

Apesar do avanço em metodologias de automação de projeto nessa área, nota-se a ausência de ferramentas de auxílio à etapa de aplicação das técnicas de tolerância falhas, que ainda é feita manualmente. A análise da confiabilidade requerida e a escolha das técnicas a serem empregadas são ainda realizadas de forma intuitiva dependendo quase que exclusivamente da experiência da equipe de desenvolvimento do sistema. Como todo processo de desenvolvimento de sistemas computacionais, a tendência é que essas etapas de desenvolvimento sejam gradativamente automatizadas ou passem a receber auxílio de ferramentas para a escolha das técnicas e exploração das alternativas, que, somadas à capacidade de entendimento do problema e experiência prévia dos projetistas, tornem o projeto de sistemas embarcados críticos mais estável e confiável.

Além disso, após a escolha das técnicas, a implementação do sistema pode se tornar mais complexa devido à inserção de redundância que, eventualmente, pode exigir a utilização de protocolos de sincronização entre os módulos replicados para que as técnicas sejam corretamente aplicadas. Sistemas simples tendem a ser mais confiáveis que sistemas mais complexos; e partindo deste princípio, a aplicação de técnicas de tolerância a falhas a um projeto simples, pode torná-lo complexo o bastante de modo a não compensar a inserção de redundância já que a possibilidade de erros de projeto aumenta e a confiabilidade do sistema como um todo pode ser prejudicada. Uma forma sistemática e conhecidamente correta para a inserção das técnicas resolveria esse problema.

2 A ferramenta ToleranSE

Este documento vem apresentar o desenvolvimento da ferramenta ToleranSE – Tolerância a Falhas em Sistemas Embutidos – que visa a inserção automática de técnicas de tolerância a falhas em projetos de sistemas embutidos. Esse tipo de sistema, geralmente é constituído por módulos implementados em software e em hardware. Inicialmente, a ferramenta tratará da aplicação de técnicas na porção de hardware do projeto.

A especificação de hardware esperada como entrada para a ferramenta deve estar descrita na linguagem de descrição de hardware VHDL. Além da especificação do sistema em VHDL, a ferramenta também recebe como entrada a especificação das técnicas a serem implementadas no projeto. A técnica escolhida é especificada pelo usuário através da interface gráfica da ferramenta. A escolha da técnica fica sob a responsabilidade do projetista sendo apenas auxiliada pela ferramenta.

Como saída, a ferramenta gerará uma nova descrição VHDL do sistema, apresentando aspectos de tolerância a falhas, conferidos pela implementação das técnicas determinadas. A figura 1 mostra um esquema sobre o funcionamento geral da ferramenta apresentada.

Algumas técnicas de tolerância a falhas foram escolhidas para serem suportadas pela ferramenta. São elas: códigos de detecção e correção de erros de *Hamming*, *NMR*, *Mid-Value Select* e *Flux-Summing*[1] .

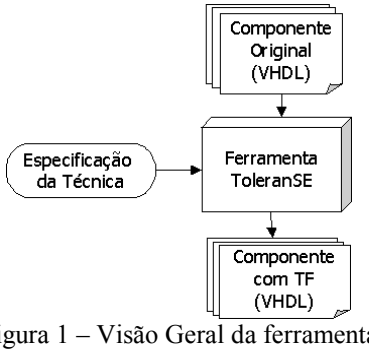

Figura 1 – Visão Geral da ferramenta ToleranSE

2.1 Técnicas de Tolerância a Falhas Abordadas

As técnicas a serem abordadas pela ferramenta foram escolhidas por serem muito utilizadas em projetos de hardware. Cada uma delas será brevemente descrita nesta seção.

NMR

A redundância modular ou NMR (*n-modular redundancy*) consiste na utilização de *N* módulos, com mesma funcionalidade, realizando a mesma computação e utilizando-se um mecanismo de voto por maioria para a escolha da saída correta.

Essa técnica considera que a probabilidade de mais de um módulo apresentar falhas durante a computação é muito pequena e desse modo a confiabilidade do sistema seria aumentada. Essa consideração exige que os módulos sejam independentes no que diz respeito às falhas[1] .

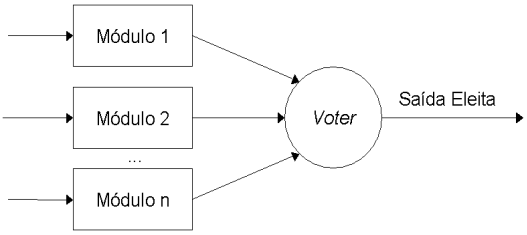

Figura 2 – Técnica NMR

Flux-Summing

Os sistemas de controle com realimentação, como por exemplo os controladores de temperatura, podem ser mais beneficiados pela técnica *flux-summing*, que utiliza propriedades inerentes desses sistemas (de controle de *loop* fechado) para a compensação de falhas.

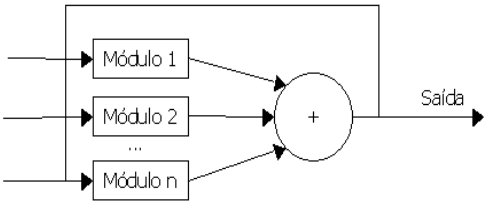

Figura 3 – Técnica Flux-Summing

A implementação desta técnica consiste em utilizar módulos redundantes e um transformador que recebe as saídas dos módulos como entrada, e sua saída é proporcional à soma das saídas dos módulos. Cada módulo é realimentado pela saída do transformador e desse modo a falha de um dos módulos é percebida e compensada pelos demais[1] .

Mid-Value Select

Em situações onde pequenas variações de valores da saída não são consideradas erro no sistema, uma variação conhecida como *mid-value select* pode ser utilizada. Nessa técnica, o sistema de votação majoritária é substituído por um componente que escolhe a saída que apresenta valor médio dentre as demais saídas do sistema.

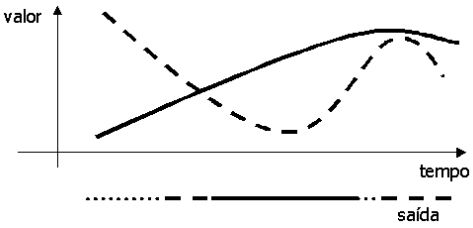

Figura 4 – Técnica Mid-Value Select

Um exemplo para a aplicação da técnica são os sistemas de conversão analógico-digital, onde as saídas podem sempre diferir nos bits menos significativos. A utilização da técnica TMR clássica neste caso não seria adequada, já que com muita freqüência acusaria erros inexistentes no sistema. Essa técnica só pode ser utilizada quando houver um número ímpar de módulos, tal que sempre uma saída seja a que se encontra no meio entre os valores considerados[1] .

Código de Hamming

Os códigos de detecção e correção de erros consistem na adição de informação redundante a conjunto de bits com o intuito de detectar e corrigir erros nos vetores de dados.

A informação redundante é produto de alguma operação realizada sobre os dados já existentes. Realizando novamente a operação e comparando o resultado com a informação redundante, pode-se detectar os erros.

No código de *Hamming*, bits redundantes são inseridos nas posições 2^n do dado codificado e as demais posições são preenchidas pelos bits originais. Cada bit de dados é utilizado para o cálculo dos bits de verificação cuja posição está contida em sua decomposição em potências de 2. Então um erro único em qualquer das posições de dados, alterará o valor dos bits redundantes que utilizaram o bit corrompido em seu cálculo. Dessa forma, para identificar a posição do bit corrompido, basta somar as posições dos bits de verificação que estão alterados. Identificada a posição do erro, o dado original pode ser facilmente recuperado.

2.2 Método de Aplicação Automática das Técnicas

A aplicação das técnicas propostas é realizada pela ferramenta através de diversas etapas. Primeiro, o código VHDL original desenvolvido pelo projetista da aplicação, que é o usuário da ferramenta, é tratado pelo analisador sintático que extrai informações necessárias para a geração do novo componente integrando os aspectos de tolerância a falhas. Essas informações, como a interface do componente, portas de I/O, e os sub-componentes instanciados, são armazenados na ferramenta. Foi utilizado, para a geração dos componentes específicos de tolerância a falhas – tais como os *voters* das técnicas NMR, Flux-Summing e Mid-Value Select, e os codificadores e decodificadores de *Hamming* – um gerador de componentes, que dependendo das características de cada aplicação, implementa o código VHDL dos componentes responsáveis pela implementação de cada uma das técnicas, específicos para a aplicação. Em seguida, a ferramenta gera as instâncias das réplicas do componente original, desenvolvido pelo projetista, além de instanciar os componentes específicos para tolerância a falhas.

É gerado ainda pela ferramenta todo o código VHDL adicional para a implementação da técnica escolhida, como as atribuições de sinais representando as ligações entre os módulos replicados e o componente de tolerância a falhas, além das conexões das instâncias com a interface do componente final, gerado pela ferramenta.

A figura 5 representa uma visão geral do funcionamento da ferramenta ToleranSE.

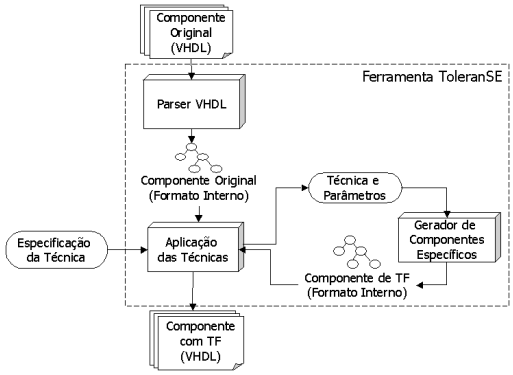

Figura 5 – Arquitetura da ferramenta ToleranSE

O método para geração da nova descrição VHDL do componente, incluindo tolerância a falhas, possui particularidades para cada uma das técnicas abordadas pela ferramenta. As técnicas que envolvem replicação de módulos – NMR, e suas variações, Mid-Value Select e Flux Summing – são aplicadas de forma similar, salvo pequenas particularidades apresentadas mais adiante.

É utilizada a arquitetura estrutural de VHDL para a geração do novo componente. O ideal é que o componente gerado possua a interface o mais parecida possível com a do componente original, minimizando o impacto da aplicação da técnica sobre o resto da implementação do sistema. Dessa forma, a interface do novo componente é inicialmente feita

igual à interface do componente original. A replicação dos módulos é feita instanciando-se o componente original tantas vezes quantas forem o número de réplicas utilizadas na técnica. Tal número será denominado *N*. Enquanto as entradas do componente original são distribuídas para as entradas dos módulos replicados, as saídas de cada réplica são agrupadas em um único vetor de bits, que será utilizado como entrada para o componente votante. Assim, são formados *N* vetores que agrupam as saídas de cada uma das réplicas. Uma instância do *voter* de *N* entradas é inserida no novo componente e os *N* vetores são fornecidos como entradas. A saída do *voter*, saída eleita dentre as saídas das réplicas, será também um vetor que representa um agrupamento de outros sinais e vetores. Esse vetor é então desagrupado nas saídas do componente gerado. Em alguns casos, o *voter* pode produzir saídas adicionais, indicando a ocorrência de falha (discordância entre os módulos) ou apontando qual dos módulos foi o discordante. Assim, essas saídas adicionais são acrescidas ao componente gerado, tendo seus valores simplesmente repassados para a interface.

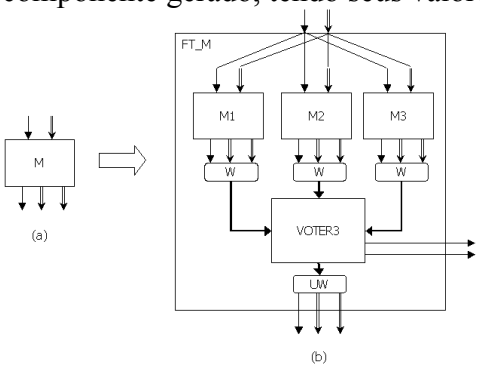

Figura 6 – Exemplo de Aplicação da Técnica com tripla replicação: (a) indica o componente original e (b) representa a arquitetura do componente gerado

A figura 6 representa o método geral de aplicação das técnicas que utilizam réplicas de componentes, considerando *N* igual à 3. Em (a) é representado o componente original, e em (b) o componente gerado pela ferramenta. Na figura, os componentes W representam os *wrappers,* responsáveis por agrupar os vetores, enquanto os componentes UW são os *unwrappers*, responsáveis pelo desagrupamento desses vetores. Tais componentes não são componentes físicos, mas apenas representam as atribuições de sinais, geradas por software e inseridas no código do novo componente, responsáveis pelo agrupamento e desagrupamento dos sinais.

Para aplicação das técnicas Mid-Value Select e Flux-Summing, nem todas as saídas dos módulos replicados devem ser levadas em consideração.

A saída do voter, nestes casos, depende operações de comparação ou soma sobre as entradas das réplicas. Assim, não faz sentido agrupar, no vetor utilizado como entrada para o voter, os sinais de saída do componente original que não representem valores numéricos. Sinais que não contenham valores numéricos, como flags por exemplo, não devem ser levados em consideração, para a correta aplicação das técnicas.

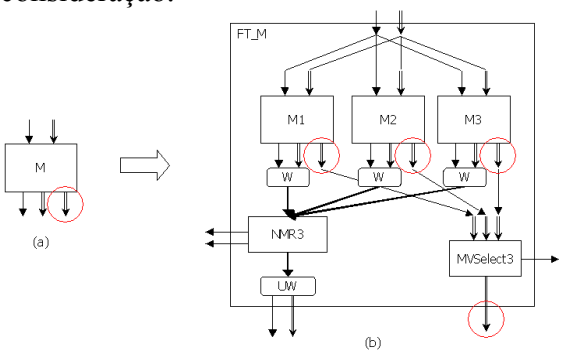

Figura 7 – Exemplo de Aplicação da Técnica Mid-Value com N = 3: (a)indica o componente original e (b) representa a arquitetura do componente gerado

Nestes casos, o projetista deve indicar a qual sinal de saída do componente original deve ser aplicada a técnica. Os demais sinais são então agrupados e votados por um voter NMR, onde os valores dos sinais não é importante. A figura 7 exemplifica o método de implementação da Mid-Value Select, com *N* igual à 3, destacando-se o vetor de saída do módulo original escolhido para aplicação da técnica.

A utilização de replicação de componentes para obtenção de tolerância a falhas, assume que os módulos replicados são independentes do que diz respeito à falhas. Para um melhor aproveitamento dessas técnicas, seria desejável que as réplicas fossem as mais independentes possíveis, sendo idealmente implementadas em diferentes abordagens, por diversas equipes de desenvolvimento e sintetizadas em chips separados. A implementação da técnica, encapsulando tanto as réplicas quanto o voter num único componente VHDL, a ser sintetizado num único chip, limita portanto o tipo de falhas que a configuração final deverá tolerar. Esse método, contudo, é eficaz para mascarar falhas parciais no chip ou componente programável onde o componente é sintetizado, além de servir como método de exploração e avaliação de alternativas para aplicação de tolerância a falhas, e para a prototipação rápida de projetos que, em ambiente de produção, serão desenvolvidos em diversos dispositivos.

Para a aplicação dos codificadores e decodificadores de *Hamming*, o procedimento utilizado é mais simples, sendo apenas necessário inserir uma instância do componente decodificador ou codificador de Hamming antes da entrada ou após a saída do componente original, dependendo das escolhas do projetista. As figuras 8 e 9 ilustram a aplicação dessa técnica a componentes genéricos. O componente *coder* recebe um vetor de dados e gera o vetor incluindo os bits redundantes para geração do código de *Hamming*, enquanto que o *decoder* faz a operação inversa. Desse modo, as interfaces dos componentes gerados diferem das interfaces dos componentes originais apenas pelo número de bits do vetor escolhido para aplicação da codificação/decodificação, que aumenta por conta da utilização dos bits redundantes.

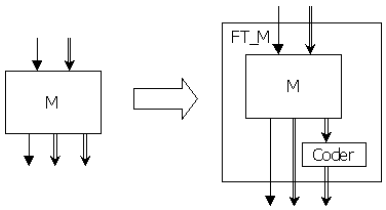

Figura 8 – Exemplo de Aplicação de um codificador de *Hamming*

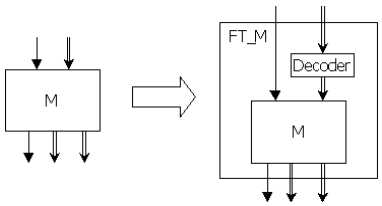

Figura 9 – Exemplo de Aplicação de um decodificador de *Hamming*

2.3 Gerador de Componentes Específicos

Um dos aspectos da inserção de tolerância a falhas é que cada solução é fortemente dependente da aplicação. Para a automatização da aplicação das técnicas, essa característica representa um problema à parte. A generalização das técnicas se torna necessária quando se pretende aplicá-las de forma automática, de modo que, para qualquer entrada fornecida pelo usuário, a ferramenta utilize um procedimento bem definido.

Assim, a implementação de uma biblioteca estática de componentes não é suficiente, já que não é possível prever todas as opções de componentes requeridas pelo usuário em ocasião da utilização da ferramenta. Sendo assim, um módulo da ferramenta desenvolvida é responsável por gerar dinamicamente, e sob demanda, os componentes VHDL específicos necessários para a aplicação da técnica escolhida pelo usuário.

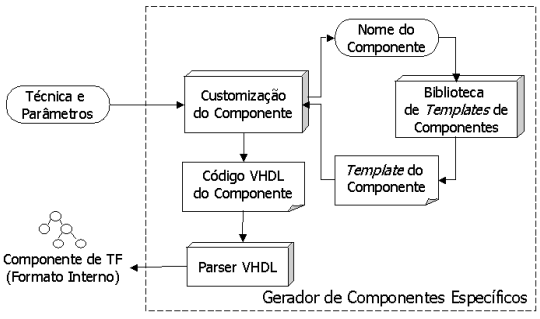

Figura 10 – Arquitetura Geral do Gerador de Componentes Específicos

O módulo gerador de componentes utiliza uma biblioteca de *templates* parametrizáveis de componentes VHDL, e instancia esses componentes de acordo com os argumentos passados por outros módulos da ferramenta. As saídas fornecidas pelo gerador representam os componentes específicos para a aplicação de tolerância a falhas – *voters* e codificadores, bem como os sub-componentes necessários para implementação destes.

2.4 Protocolo de Sincronização

Havendo replicação de módulos, é preciso garantir que os módulos replicados mantenham-se no mesmo passo da execução, evitando que a configuração tolerante a falhas – composta pelas réplicas e *voter* – detecte falhas inexistentes nos módulos. Assim, pode ser necessária a utilização de um protocolo de sincronização entre os módulos replicados e o *voter,* de forma que este não compute a votação entre as saídas nos intervalos em que uma das réplicas estiver ainda computando resultados ou uma das saídas encontre-se instável.

Nota-se que, sendo os *voters* das técnicas abordadas componentes puramente combinacionais, nos casos em que o componente original for implementado também de forma combinacional, não haverá necessidade de sincronismo, já que as réplicas receberão os mesmos sinais de entrada e portanto devem produzir as mesmas saídas sempre. Nesse caso podemos considerar que o efeito de não haver uma saída estável durante o período de computação do resultado final com o *voter* é semelhante ao efeito que se obtinha com o circuito original, apenas observando que o período que se deve esperar para se obter um resultado estável é igual ao maior período entre as N-versões acrescido do tempo utilizado pelo *voter*.

No caso de o componente original ter comportamento síncrono, regido por *clock*, há a necessidade de controle de sincronização e este é realizado através de um protocolo bem simples. O projetista deve implementar o componente original de forma que este indique através de um bit, denominado *ready*, que as saídas estão prontas para serem votadas. As saídas devem ser mantidas estáveis até que um outro bit, dessa vez de entrada, denominado *continue*, seja ativado. O bit de *continue* é enviado pelo *voter*, indicando que a votação foi realizada e que o módulo pode continuar executando sua máquina de estados. O *voter* por sua vez, espera que todas as réplicas ativem o bit de *ready*, indicando que suas saídas foram computadas e estão estáveis, para fazer a votação e liberar a saída da configuração tolerante a falhas. Após a votação ser realizada, o bit de *continue* deve ser enviado a todas as réplicas, e o componente volta a esperar pelos sinais de *ready*.

Para a implementação do protocolo de sincronização, alguns componentes VHDL adicionais são acrescentados aos *voters* para o controle de sincronização.

O principal destes componentes, o ControlSync, é regido por *clock*, que será conectado ao mesmo sinal de *clock* das réplicas, e é responsável pela execução de uma máquina de estados que implementa o protocolo. Além disso são utilizados um *timer*, para geração de um sinal de *timeout* quando uma das réplicas atrasar ou falhar no envio do sinal *ready,* e um registrador para manter a saída do *voter* estável até que ocorra a próxima votação. A figura 12 mostra um exemplo de *voter* 3MR encapsulado com o mecanismo de controle de sincronização em um único componente.

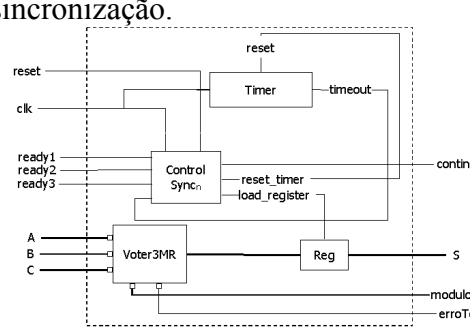

Figura 12 – *Voter* 3MR com controle de sincronização.

3 Resultados e Trabalhos Futuros

A ferramenta encontra-se atualmente em fase de testes. Um estudo de caso foi realizado aplicando-se a técnica 3MR no componente adicionador de um controlador de uma máquina de vender refrigerantes. O sistema foi validado através da injeção manual de falhas em vetores internos do projeto e a ferramenta de síntese de alto nível Max+plus II da Altera foi utilizada para simulação, que apresentou resultados satisfatórios. A tabela abaixo mostra o número de células lógicas utilizadas para a síntese e o atraso no caminho mais longo do projeto, antes e depois da aplicação automática da técnica. Pelos dados apresentados, nota-se que o número de células lógicas após aplicação de tolerância é pouco maior que três vezes o número original. Sendo a triplicação de recursos já esperada na implementação do 3MR, verifica-se que o *overhead* causado pela inserção da técnica é de 9 células lógicas. O aumento no atraso devido à utilização do *voter* foi de 5ns. Vale ressaltar que o componente utilizado para o estudo é bastante simples. Para componentes de mais alta complexidade, o *overhead*, de área e de tempo, causado pela utilização da técnica é ainda menos significativo, já que a complexidade do *voter* não é proporcional à complexidade do componente sendo replicado. Outros estudos de caso abrangendo todas as técnicas abordadas pela ferramenta deverão ainda ser realizados.

	Componente Original	Componente Tolerante a Falhas
Células lógicas utilizadas	21	72
Atraso no caminho mais longo	11,5 ns	16,5 ns

4 Conclusões

Com a automação da implementação das técnicas, o custo e o tempo de projeto dos sistemas de hardware tolerantes a falhas seriam diminuídos, além de a qualidade do sistema ser aumentada pela prevenção de inserção de erros de projeto que processos automáticos oferecem. A ferramenta também oferece auxílio ao projetista durante a fase de escolha das técnicas a serem aplicadas ao projeto, através da viabilidade de exploração no espaço de soluções. Num processo manual, mais lento e de custo mais elevado, a análise detalhada de qual técnica de redundância utilizar pode ser inviável. Assim, o projetista define a priori a técnica a ser utilizada podendo esta não ser a técnica ótima para o projeto. A ferramenta, oferecendo uma maneira de rápida implementação de várias técnicas, viabiliza a comparação do impacto que a aplicação de cada uma das técnicas pode causar na confiabilidade do sistema, permitindo uma busca mais eficiente da melhor técnica no espaço de soluções.

5 Referências

[1] Pradhan, D. K.; FAULT-TOLERANT COMPUTER SYSTEM DESIGN. Prentice Hall 1996.

[2] Vahid, F.; Givargis, T. EMBEDDED SYSTEM DESIGN: A UNIFIED HARDWARE/SOFTWARE APPROACH. Department of Computer Science and Engineering – University of California, 1999.

[3] Avizienis, A. TOWARD SYSTEMATIC DESIGN OF FAULT TOLERANT SYSTEMS. IEEE Computer, 30, No. 4 (1997), 51-58.

[4] Jenn, E.; Arlat J.; Rimen, M.; Ohlsson, J.; Karlsson, J. FAULT INJECTION INTO VHDL MODELS: THE MEFISTO TOOL. FTCS-24, pp. 66 – 75. IEEE, 1994.

[5] Kim, K.; Tront, J.G.; Ha, D.S. AUTOMATIC INSERTION OF BIST HARDWARE USING VHDL. IEEE Design Automation Conference, pp. 9 – 15. IEEE, 1988.