Título do Artigo: Systematic Software-Based Self-Test for pipelinedd Processors.
Referência Bibliográfica: Mihalis Psarakis, Dimitris Gizopoulos, Miltiadis Hatzimihail, Antonis Paschalis, Anand Raghunathan, Srivaths Ravi; 43 rd Design Automation Conferencia (DAC/2006)**;** pp. 393- 398
Aluna: Cláudia Morgado
RA: 075738

O objetivo desse artigo foi analisar a metodologia SBST em dois modelos de processadores inteiramente *pipelined* RISC (miniMIPS e OpenRISC 1200), detectar as falhas na cobertura dos testes e propor melhorias.

A metodologia de SBST (Auto-teste baseado em software) consiste no desenvolvimento de um programa de teste eficiente que alcance alta cobertura de falhas. O programa de auto-teste é armazenado na memória de instruções enquanto os padrões e as respostas de teste são armazenados na memória de dados. Ao ser executado, o programa de auto-teste aplica os padrões aos componentes do processador, ou aos demais núcleos do SoC se for o caso, e as respostas obtidas são armazenadas na memória do sistema.

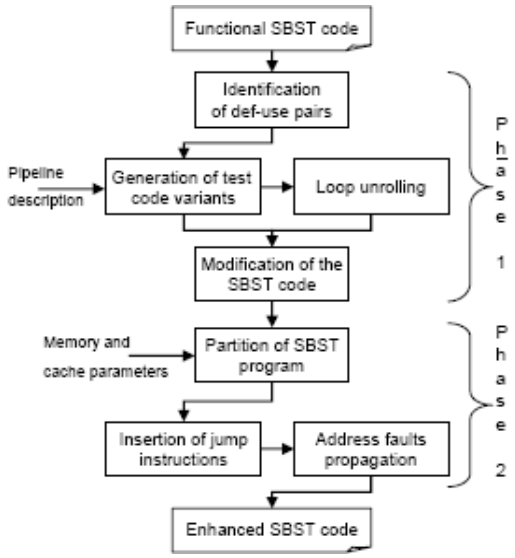
A técnica SBST apresenta uma série de vantagens sobre outros métodos de teste *online.* A que mais se destaca é o fato do SBST ser um método de teste não intrusivo, uma vez que utiliza as próprias instruções do processador, dispensando quaisquer alterações na sua estrutura ou no seu conjunto de instruções. Além disso, o SBST é uma estratégia de baixo custo, de baixa potência, bastante flexível e facilmente programável.

Analisando a lógica da testabilidade dos programas de SBST nos dois modelos de processadores, detectou-se que embora os componentes funcionais do processador sejam eficientemente testados, a cobertura total de falhas foi de apenas 86,58%. Uma análise profunda das falhas não detectadas em cada componente do *pipelined* revela os seguintes problemas:

Primeiro Problema: a lógica de carregamento da informação do endereço-relacionado através dos estágios do pipelined são mal testadas. O endereço da memória da instrução (IMA) é usado em todos os estágios do pipelined e o endereço de dados (DMA) é utilizados em caso de instruções de *load/store*. A informação do endereço usada nos estágios do pipelined é também usada por outros componentes como barramento, exceções, etc. Cada um desses estágios são acoplados a uma quantidade de lógica significativa para executar suas atividades, porém não são todas testadas.
Solução Proposta: uma solução genérica para excitação da lógica do endereço é requerer a execução do código de SBST encontrado, em diferentes regiões em todo espaço de memória. A colocação do código SBST deve ser feita sem afetar a velocidade de simulação da falha. A idéia chave proposta envolve dividir um dado programa de SBST em múltiplos segmentos de código, que são virtualmente armazenados e executados em regiões de memória diferentes.

Segundo Problema: baixa cobertura de testes na detecção e definição de *hazard* .
Solução Proposta: melhorar a cobertura da falha da lógica de controle e o trajeto do fluxo de dados da estrutura do pipelined. O conceito básico é aplicar as seqüências de instrução do teste que aumentam a atividade dos componentes do *pipelined*: causar *hazards* de tipos diferentes, e ativar todas as vezes múltiplos trajetos do despacho com dados diferentes.

Metodologia SBST com as soluções propostas



A metodologia consiste em duas fases: a fase 1 aplica a solução para os mecanismos da detecção de hazards e a fase 2 aplica a solução para os componentes da lógica de endereço. A fase 1 processa entrada dos programas SBST identificando os *def-use*, o qual representa a possibilidade de dependência entre as instruções. Para dependências gerais dos dados, a fase 1 faz a varredura do código, e gera as variantes correspondentes do código para cada par de dependência. Para os programas de SBST que têm uma dependência dos dados dentro de um laço, a metodologia faz o processo de " *unrolling* laço" para combinar eficazmente todos as variantes do código. Então ele seleciona as variantes do código e substitui as instruções originais a esse. Na fase 2 os programas gerados na fase 1 são divididos em segmentos Agora os segmentos do programas são carregados na memória física da instrução, em seguida são introduzidas as instruções de salto afim de transferir a execução do programa a uma segmento seguinte. Finalmente a fase 2 introduz as sequências de instruções apropriadas que garante a propagação das falhas nos trajetos dos despachos.

Essas alterações fizeram com a cobertura dos testes chegassem ao redor dos 93%, com poucas alterações com relação ao tamanho dos programas e no tempo de execução.