UM ALGORITMO PARA VALIDAÇÃO DE REDES DE TRANSISTORES GERADAS AUTOMATICAMENTE PELA FERRAMENTA SOPTIMIZER

DE SOUZA, Renato S.[1]; DOMINGUES JÚNIOR, Julio S.[1]; POSSANI, Vinicius N.[1] MARQUES, Felipe de S.[1]; DA ROSA JR, Leomar S.[1]

[1]Universidade Federal de Pelotas, Centro de Desenvolvimento Tecnológico – CDTec.
{rsdsouza, jsdomingues, vnpossani, felipem, leomarjr}@inf.ufpel.edu.br

1 INTRODUÇÃO

Os circuitos digitais estão cada vez mais presentes no dia-a-dia, causando um grande impacto na sociedade, devido ao fato de que estes se aplicam diretamente em diferentes áreas do conhecimento. Como exemplo, podemos citar computadores, telefones, sistemas automotivos computadorizados, aparelhos de GPS e equipamentos médicos. Nota-se que o avanço na área de desenvolvimento dos circuitos digitais é muito importante para a criação de novas tecnologias, como um auxílio à vida humana, entre outras coisas (Da Rosa Jr, 2008). Uma rede de transistores pode ser representada através de uma abordagem baseada em grafo (ZHU, 1993). Baseado nesta técnica uma ferramenta foi desenvolvida, a Soptimizer (POSSANI, 2010), na qual uma rede de transistores é representada através de um grafo e posteriormente é realizado um processo de otimização através de compartilhamento de arestas.

Este trabalho tem como finalidade a criação de um algoritmo que seja capaz de receber os caminhos de uma rede otimizada pela ferramenta Soptimizer, montar a equação lógica desta rede e, posteriormente, validá-la através da equação gerada, comparando com a soma-de-produtos (SOP) utilizada como entrada no Soptimizer. Este algoritmo utiliza algumas estruturas de dados, como vetores, os quais são estruturas de dados lineares e estáticas, assim como listas, que neste caso serviram como uma controladora de níveis para identificar quais as equações estão sendo manipuladas.

2 METODOLOGIA (MATERIAL E MÉTODOS)

A idéia inicial deste algoritmo consiste em receber como entrada os caminhos do grafo otimizado pela ferramenta Soptimizer. Cada literal presente em um caminho é retirado e armazenado em um vetor, que será chamado de vetor de literais. Assim, para cada caminho será criado um vetor de literais. Quando todos esses vetores forem obtidos, cada um deles será adicionado a outro vetor, descrito no texto por vetor de caminhos, o qual será utilizado para identificar, através de sua posição, com qual produto o algoritmo irá trabalhar no momento. A Fig.1 apresenta todos os caminhos recebidos pelo Soptimizer, esses caminhos foram organizados em vetores de literais e posteriormente foram adicionados ao vetor de caminhos.

Figura 1: Vetor de caminhos que armazena os vetores de literais que representam os caminhos do grafo.

Na sequência, é inicializada uma lista a qual tem o papel de armazenar os índices do vetor de caminhos, para informar a uma sub-rotina em qual posição ela deverá acessar este vetor, para trabalhar com o caminho correto. Como neste caso, a execução do algoritmo está no inicio, a lista é inicializada com todos os índices que pertencem ao vetor de caminhos, como ilustra a Fig. 2.

Figura 2: Lista que armazena os índices do vetor que contem os produtos da SOP.

A sub-rotina mencionada anteriormente tem como função realizar uma comparação de literais. Nela, usa-se um vetor que é inicializado de acordo com a quantidade de caminhos extraídos do grafo. Este vetor será indicado ao longo do texto como vetor de comparações. Seguindo nosso exemplo, o vetor de comparações será inicializado com três posições.

Está sub-rotina deverá verificar o conteúdo da última posição da lista que indica os índices que deverão ser acessados no vetor de caminhos. Então, o primeiro literal encontrado em cada um dos caminhos é copiado para o vetor de comparações. Este processo é representado na Fig. 3.

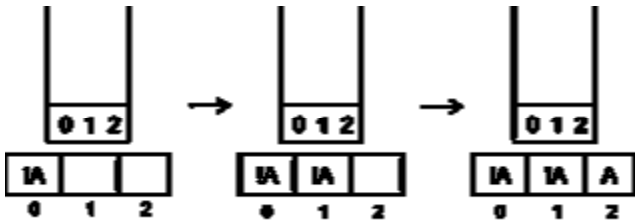


Figura 3: Vetor de comparação contendo o primeiro literal de cada caminho indicado pelo nível superior da lista.

Após acessar todas as posições indicadas pela lista, e ter copiado todos os primeiros literais de cada caminho, a sub-rotina irá procurar no vetor de comparações em quais posições os literais são iguais e retornar os índices desses literais para a rotina principal. Este retorno servirá para informar a rotina principal com quais posições do vetor de caminhos ela irá trabalhar neste momento.

Neste caso, a sub-rotina irá retornar os índices 0 e 1. Portanto o algoritmo irá colocar o literal equivalente a um dos retornos da sub-rotina, em um vetor, chamado equação, que armazenará os literais que irão compor a equação final. Então, será adicionado ao vetor equação o primeiro literal que corresponde ao caminho que, neste caso, está na posição 0 do vetor de caminhos. Removendo o primeiro literal nos caminhos que estão nas posições do vetor de caminhos, que são indicados pelo retorno da sub-rotina de comparação. A Fig. 4 ilustra o estado atual do vetor de caminhos após este procedimento. Por fim, será atualizada a lista para indicar qual serão os próximos caminhos a serem processados. Portanto, os índices retornados pela sub-rotina serão removidos do nível em que estão na lista, e posteriormente serão adicionados no nível superior da lista. Após atualizar a lista de níveis, seu estado atual está representado na Fig. 5.

Figura 4: Estado atual do vetor de caminhos.

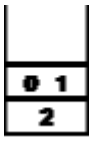


Figura 5: Lista de níveis atualizada, após processo de comparação de literais.

A atualização da lista é realizada toda vez em que a sub-rotina, responsável por comparar os literais, retorna os índices dos literais iguais encontrados nas equações comparadas. Esta atualização é importante, pois além de informar quais serão as próximas equações a serem tratadas, ela também é responsável pelo controle de parênteses da equação. Toda vez que o algoritmo sobe um nível na lista, os parênteses são controlados adicionando um " (" (abre parênteses) ao vetor equação. Outra relação importante com as atualizações que ocorrem na lista, é que quando é realizada em algum determinado tempo de execução do algoritmo, uma operação que retorna a um nível anterior na lista, isto indica que deve ser adicionada ao vetor equação o carácter ") " (fecha parêntese) e na sequência é adicionado um " + " (operador lógico OU).

Toda vez que é inserido um carácter no vetor equação, antes é testado qual o conteúdo da ultima posição do vetor equação, por exemplo, se estivéssemos adicionando um literal ao vetor equação, e se a ultima posição deste vetor também for um literal, então primeiro deverá ser adicionado um " * " (sinal que representa uma operação lógica E) para depois adicionar o literal em questão. Caso contrário, se a última posição do vetor equação não for um literal, então será adicionado apenas o literal no vetor equação. Voltando ao andamento do algoritmo, neste momento, será novamente verificado o topo da lista para saber quais posições do vetor de caminhos serão acessadas para copiar o primeiro literal presente em cada caminho.

Caso a lista não contenha mais índices, mas mesmo assim ela contém níveis, o algoritmo deve retornar todos os níveis da lista e a cada retorno de nível é adicionado ao vetor equação um ") ", até não restar mais níveis na lista. Caso contrário, o algoritmo encerra sua execução. Na Fig. 5 podemos verificar que ainda existem índices na lista. Portanto será executado novamente o processo explicado na Fig. 3, mas neste caso iremos cair em uma situação diferente. Pois, neste momento, como estamos trabalhando com as equações que estão nas posições '0' e '1' do vetor de caminhos, podemos conferir que o primeiro literal de cada um dos caminhos é diferente, como mostra a Fig. 6. Então neste caso a sub-rotina retornará que não existem literais iguais, portanto o algoritmo irá copiar todos os caminhos que estão sendo manipulados para o vetor equação, colocando entre cada caminho um sinal de " + ". Após ser feita toda a cópia dos caminhos eles são apagados do vetor de caminhos, e seus índices também serão apagados da lista de níveis. Na Fig. 7 é ilustrado o vetor de caminhos e a lista após este processo.

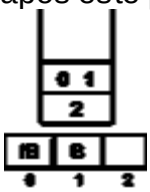
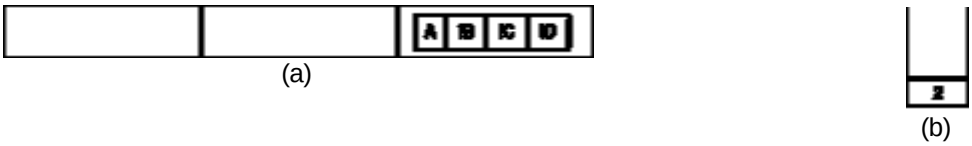


Figura 6: Vetor de comparação com os primeiros literais das equação 0 e 1.



(a) (b)

Figura 7: Lista de níveis(a) e vetor de comparações(b).

Como agora temos apenas um índice na lista, o caminho referente ao índice é removido do vetor de caminhos e adicionado no vetor equação e o índice também será removido da lista de níveis. Portanto, chegamos ao ponto em que a lista de níveis está vazia, e isso indica que o processo para criação da equação já foi finalizada. A equação gerada pelo algoritmo esta representada na Fig. 8.

Figura 8: Equação final.

3 RESULTADOS E DISCUSSÃO

O algoritmo proposto foi implementado na linguagem Java utilizando a IDE Netbeans 6.9.1. Para validar a abordagem proposta, foi utilizada um conjunto de funções lógicas de 4 entradas da p-class, que é composta por 3.982 funções. Todas as funções foram aplicadas na ferramenta Soptimizer e logo após a otimização de cada função, foi executado o algoritmo proposto neste trabalho, para gerar a equação representada pelo grafo. A validação das equações geradas, foi realizada a partir da ferramenta SwitchCraft (CALLEGARO, 2010), onde cada equação gerada era comparada com a soma-de-produtos utilizada como entrada no Soptimizer. Também foram avaliadas um conjunto de dez funções Booleanas randômicas com 7 variáveis, que foram utilizadas para base de resultados na ferramenta Soptimizer. Todas as dez funções foram validas da mesma forma que as 3.982 funções da p-class.

4 CONCLUSÃO

Este trabalho apresentou um algoritmo simples que tem por função extrair a equação de um grafo que foi otimizado pela ferramenta Soptimizer, com o intuito de validar este grafo otimizado. Apesar da solução proposta não conseguir extrair equações mínimas do grafo (com o menor número de literais possíveis), o trabalho desenvolvido atingiu o objetivo proposto. Como trabalhos futuros, pretende-se investigar formas de permitir a extração de equações mínimas e, assim, permitir que este algoritmo seja utilizado como uma ferramenta de fatoração para expressões Booleanas.

5 REFERÊNCIAS

DA ROSA JR., L. S. **Automatic Generation and Evaluation of Transistor Networks in Different Logic Styles**. PhD Thesis PGMicro/UFRGS, Porto Alegre, Brazil. (2008).

ZHU, J. et al. **On the Optimization of MOS Circuits**. IEEE Transactions on Circuits and Systems: Fundamental Theory and Applications. (1993), 412-422.

POSSANI, V. N. ; TIMM, E. F. ; AGOSTINI, L. V. ; ROSA JUNIOR, L. S. . TRANSISTOR NETWORKS DESIGN USING A GRAPH-BASED APPROACH. In: **10th Microelectronics Students Forum, 2010, São Paulo**. 10th Microelectronics Students Forum, 2010.

CALLEGARO, V. ; ROSA JR, L. ; MARQUES, F. S. ; RIBAS, R. P. ; REIS, A. I. . SwitchCraft - A Tool for Generating Switch Networks for Digital Cells. In: **SBCCI, 2010, São Paulo**. SBCCI, 2010