IMPLEMENTAÇÃO E OTIMIZAÇÃO DE PORTAS LÓGICAS COMBINACIONAIS NO NÍVEL DE TRANSISTORES UTILIZANDO UMA ABORDAGEM BASEADA EM GRAFOS

POSSANI, Vinicius N.; TIMM, Éric F.;
AGOSTINI, Luciano V.; DA ROSA JR, Leomar S.

Grupo PET Computação UFPel
Universidade Federal de Pelotas
Pelotas – RS – Brasil

1 INTRODUÇÃO

Atualmente, o desenvolvimento de circuitos VLSI tem dominado uma boa parte da indústria de microeletrônica. Ferramentas de automação têm ajudado projetistas a manipular mais transistores durante o projeto ao mesmo tempo em que reduz o ciclo de desenvolvimento. Este processo se tornou mais eficiente através de algoritmos para a geração automática de redes de transistores.

A técnica mais comum para otimizar redes de transistores é baseada em fatoração (BRAYTON, 1987) (MINTZ, 2005). Porém, na literatura existem métodos alternativos baseados em otimizações de grafo, onde cada aresta do grafo mantém uma associação com um transistor da rede. A idéia principal é tentar minimizar as arestas de um dado grafo, ou compor um novo grafo com um número de arestas reduzido (ZHU, 1993). Com isso, este trabalho apresenta a ferramenta Soptimizer, que implementa um método baseado em grafo para gerar redes de transistores. Nessa abordagem, temos como entrada uma expressão Booleana que é transformada em um grafo, o qual posteriormente será otimizado através do compartilhamento de arestas.

2 METODOLOGIA

A proposta baseada em grafos presente na ferramenta Soptimizer, aceita como entrada uma expressão representada na forma de uma soma de produtos (SOP). A idéia básica consiste, inicialmente, em separar todos os produtos que compõem a SOP. Na sequência, cada literal presente em um produto é extraído e colocado em um vetor. Para cada produto, é criado um vetor. A Figura 1 ilustra todos os vetores obtidos da Expressão (1), que representa a SOP de entrada.

!A*C*E*F*!G*H + !A*B*F*!H + A*!B*C*!G*H (1)

!A	C	E	F	!G	H
!A	B	F	!H		
A	!B	C	!G	H	

Figura 1. Vetores que representam os produtos da Expressão (1).

Quando a lista de vetores é obtida, então os vetores são organizados de acordo com o número de literais que os compõem. A Figura 2 exemplifica esta organização. Após, inicia-se a montagem do grafo, removendo da lista os vetores de mesmo tamanho e criando uma aresta no grafo para cada literal encontrado no vetor. Isto é demonstrado na Figura 3.

!A	C	E	F	!G	H
A	!B	C	!G	H	
!A	B	F	!H		

Figura 2. Vetores que representam os produtos da Expressão 1, ordenados.

Figura 3. Primeiro produto como um grafo.

Na sequência, outro produto é carregado da lista de vetores e colocado no grafo. A Figura 4 ilustra este processo para o segundo vetor da lista. Todos os caminhos do grafo são percorridos com o objetivo de encontrar arestas idênticas, as quais representam mesmas variáveis de controle. Se esta condição for satisfeita, então as arestas idênticas são levadas para o início do grafo onde será realizado o compartilhamento entre os vértices que as conectam. Isto é exemplificado na Figura 5, onde as arestas ‘C’, ‘!G’ e ‘H’ foram unificadas.

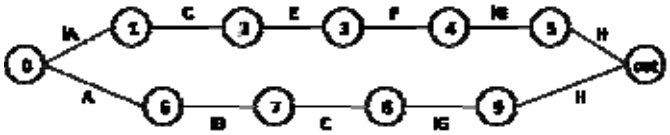

Figura 4. Grafo composto pelos dois primeiros produtos.

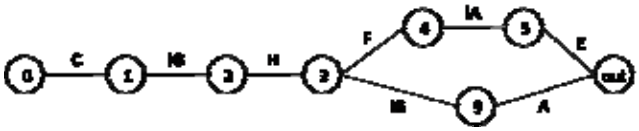

Figura 5. Grafo otimizado para os dois primeiros produtos.

Este procedimento é executado até que a lista de vetores esteja completamente vazia. A Figura 6 mostra o último vetor adicionado ao grafo. As próximas arestas candidatas a serem unidas são ‘!A’ e ‘F’, neste caso esta otimização não é permitida, já que o vértice 3 é um ponto de separação entre dois caminhos.

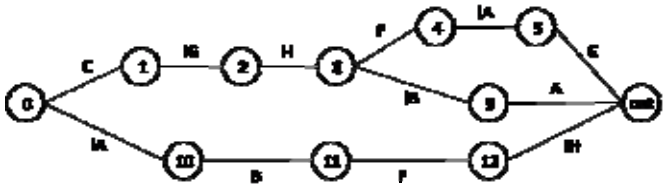

Figura 6. Grafo com o último produto adicionado.

Na sequência desta etapa de otimização, o processo é iniciado novamente. Mas agora o grafo é percorrido em sentido contrário, do final para o início, com o objetivo de mover e unir os literais que não puderam ser otimizados anteriormente. Para isso os vértices ‘0’ e ‘out’ são trocados de lugar.

Agora os literais ‘!A’ e ‘F’ podem ser levados para o início do grafo para serem unidos, cada um com seu literal equivalente. A Figura 7 e a Figura 8 ilustram este procedimento.

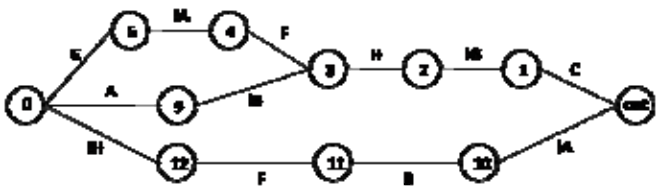

Figura 7. Grafo com os vértices ‘0’ e ‘out’ invertidos.

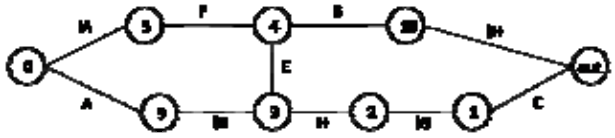

Figura 8. Grafo obtido após a otimização final.

Para garantir que caminhos inválidos não sejam introduzidos no grafo durante o processo de compartilhamento de arestas, uma rotina, que percorre o grafo comparando os caminhos com os produtos originais da expressão, é invocada regularmente após cada otimização. Isto é necessário porque se um caminho inválido for introduzido, o grafo não será uma representação verdadeira da expressão Booleana recebida como entrada.

É possível notar que todos os produtos da SOP são representados no grafo através dos caminhos ‘!A F B !H’, ‘!A F E H !G C’ e ‘A !B H !G C’. No entanto, devido ao compartilhamento de arestas um novo caminho, ‘A !B E B !H’, também foi introduzido. Este caminho é aceito se ele não alterar o comportamento lógico da rede. Em redes de transistores, este novo caminho não é sensibilizado, pois ele contém transistores controlados pela variável ‘B’ em ambas as polaridades. Em outras palavras este é um caminho nulo.

Um detalhe interessante é que a abordagem proposta pode gerar redes do tipo *“bridge”*, como o método proposto por (ZHU, 1993). A Figura 8 ilustra uma configuração do tipo *“bridge”* (através da aresta ‘E’). Isto é uma vantagem em relação a otimizações baseadas em fatoração que podem apenas gerar redes série-paralelo. Em geral, para algumas funções Booleanas, redes série-paralelo não são a solução mais otimizada possível (DA ROSA JR, 2008).

3 RESULTADOS E DISCUSSÕES

A ferramenta com o algoritmo proposto foi implementado na linguagem Java usando a IDE NetBeans 6.8. A interface gráfica da ferramenta foi desenvolvida com a biblioteca Prefuse (PREFUSE.ORG, 2004) e pode ser observada na Figura 9, onde o grafo resultante do processo de otimização da Expressão (1) é ilustrado. Como saída, a ferramenta também gera um arquivo Spice da rede de transistores, o qual pode ser usado em simuladores elétricos.

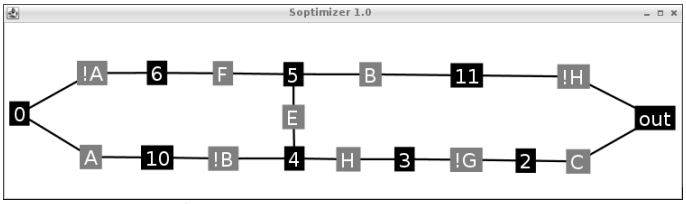

Figura 9. Interface gráfica desenvolvida para a ferramenta Soptimizer.

Com o objetivo de avaliar a abordagem proposta, 10 funções Booleanas com 7 variáveis de entrada foram escolhidas de forma aleatória. Elas foram aplicadas na ferramenta comercial SIS (SENTOVICH, 1992) e extraídas na forma de SOP. A Tabela 1 apresenta os resultados obtidos, onde o número total de literais para a forma de SOP é descritos na coluna “Nº de literais SOP”. As expressões foram fatoradas usando o algoritmo *“quick-factor”* do SIS. Os resultados são exibidos na coluna “Nº de literais SIS”. Os resultados obtidos usando o método proposto são exibidos na coluna “Nº de arestas Soptimizer”. O ganho obtido sobre a ferramenta SIS é exibido na coluna “% de ganho”.

Na maioria dos casos, o método proposto neste trabalho alcançou resultados melhores. Analisando as redes obtidas é possível identificar varias configurações do tipo *"bridge"* nos arranjos gerados pelo Soptimizer. Já a ferramenta SIS gerou expressões otimizadas compostas por operadores 'AND' e 'OR'. Neste caso, apenas rede série-paralelo podem ser implementadas, representando um grande consumo em área. Para a função F7 o algoritmo proposto não obteve o melhor resultado, pois para esta expressão não foi possível atingir a configuração do tipo *"bridge"*. Este resultado está relacionado com a ordem em que os produtos e os literais são organizados para fazer a montagem e a otimização do grafo.

Tabela 1. Resultados para as 10 funções Booleanas com 7 variáveis, escolhidas aleatoriamente.

Funções	Nº de literais SOP	Nº de literais SIS	Nº de arestas Soptimizer	% de ganho
F1	133	80	67	16,25
F2	92	70	48	31,42
F3	78	62	39	37,09
F4	150	78	75	3,48
F5	119	82	62	24,39
F6	71	44	35	20,45
F7	170	76	81	-6,17
F8	135	78	66	15,38
F9	111	74	62	16,21
F10	97	64	44	31,25

4 CONCLUSÕES

Este trabalho apresentou uma abordagem baseada em grafo para gerar redes de transistores otimizadas. A solução proposta pode atingir redes do tipo *"bridge"*. Como *"benchmark",* 10 expressões Booleanas com 7 variáveis foram usadas. Os resultados obtidos demonstram que o método proposto pode gerar redes com uma redução de transistores de até 37,09% se comparado ao algoritmo *"quick-factor"* da ferramenta SIS.

5 REFERÊNCIAS

BRAYTON, R. K. **Factoring logic functions**. IBM J. Res. Dev. 31, 2 (1987), 187-198.

MINTZ, A. and Golumbic, M. C. **Factoring boolean functions using graph partitioning**. Discrete Appl. Math. 149, 1-3 (2005), 131-153.

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

DA ROSA JR, L. S. **Automatic Generation and Evaluation of Transistor Networks in Different Logic Styles**. PhD Thesis PGMicro/UFRGS, Porto Alegre, Brazil. (2008), 147 p.

PREFUSE.ORG. **The Prefuse Visualization Toolkit**. [Online] Avaliable: http://prefuse.org/ [Acessed: Mar. 25, 2010].

SENTOVICH, E. et al. SIS: **A system for sequential circuit synthesis**. Tech. Rep. UCB/ERL M92/41. UC Berkeley, Berkeley. (1992).