BOOLEAN REPRESENTATION CODE – UM MÉTODO EFICIENTE PARA REPRESENTAR FUNÇÕES BOOLEANAS

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

Universidade Federal de Pelotas, Centro de Desenvolvimento Tecnológico – CDTec,
Grupo de Arquitetura e Circuitos Integrados – GACI.
{rsdsouza, vnpossani, jsdomingues, felipem, leomarjr}@inf.ufpel.edu.br

1 INTRODUÇÃO

Os aparelhos eletrônicos estão cada vez mais presentes em nosso dia a dia, causando um grande impacto na sociedade, devido ao fato de que estes se aplicam diretamente em diferentes áreas do conhecimento. Consequentemente, grandes dificuldades acabam sendo encontradas devido à adaptação de novos parâmetros da tecnologia, como por exemplo, a complexidade de desenvolver um chip em um tempo curto o suficiente para que o produto seja lançado no mercado. Neste contexto, as ferramentas de CAD (*Computer Aided Design*) vêm contribuindo para que os desenvolvedores aumentem a eficiência e diminuam a complexidade em um projeto (DA ROSA JUNIOR, 2009).

Neste contexto, foi desenvolvida uma ferramenta chamada *Soptimizer* (POSSANI, 2012) que implementa um método baseado em grafo para gerar redes de transistores. A partir de uma expressão booleana, obtém-se um grafo, onde cada aresta deste grafo representa um transistor e posteriormente, um processo de otimização através de compartilhamento de arestas é realizado, chegando a uma rede otimizada. Devido aos compartilhamentos de arestas realizados durante o processo de otimização, podem ser introduzidos novos caminhos no grafo, que podem mudar o comportamento lógico do circuito. Sendo assim, é preciso garantir que estes novos caminhos não alterem o comportamento lógico do circuito que o grafo representa.

Este trabalho propõe um método para a geração de um *Boolean Representation Code* (BRC) de uma função lógica. O método proposto é incorporado na ferramenta *Soptimizer* para verificar, a partir de uma forma rápida e segura, se os novos caminhos no grafo são válidos e não alteram o comportamento lógico do circuito. Com a utilização do método proposto a ferramenta *Soptimizer* torna-se também capaz de realizar algumas otimizações algébricas que não eram possíveis ao utilizar a solução anterior.

2 BOOLEAN REPRESENTATION CODE

A ideia principal deste método é gerar um BRC para cada função Booleana. O primeiro passo consiste em verificar quantas variáveis existem na função. O método proposto representa um BRC através de um inteiro. Assim, para descobrir quantos inteiros são necessários é computado 2^n, onde n representa o número de variáveis presentes na função. Após, o resultado é dividido por 32, grandeza que representa o tamanho de um inteiro. Caso seja necessário utilizar mais de um inteiro para representar um BRC, será utilizada uma estrutura de vetores para armazenar cada inteiro. Por exemplo, no caso de uma função conter mais de seis variáveis, o resultado do calculo $2^6/32$ é igual a 2. Assim, é necessário dois inteiros para gerar o BRC básico para cada variável. Um vetor é utilizado para garantir que, durante as operações lógicas as comparações são realizadas

corretamente, onde cada inteiro no vetor seja comparado com outro inteiro em uma posição equivalente.

Depois de verificar quantos inteiros são necessários para criar um BRC, o método gera o BRC básico, que é o BRC de cada variável na função de entrada. Se a função não tem mais de cinco variáveis, é realizado um processo de atribuição simples, onde cada variável recebe um BRC, como mostrado na Fig.1. Os dados presentes na Fig.1 foram gerados através de concatenações de bits, similar ao processo de montagem de uma tabela verdade. A Fig. 2 exemplifica a geração ao considerar duas variáveis.

✓ 1 variável:	✓ 2 variáveis :	✓ 3 variáveis :	✓ 4 variáveis :	✓ 5 variáveis :
var1 = 1	var1 = 5 var2 = 3	var1 = 85 var2 = 51 var3 = 15	var1 = 21845 var2 = 13107 var3 = 3855 var4 = 255	var1 = 1431655765 var2 = 858993459 var3 = 252645135 var4 = 16711935 var5 = 65535

Figura 1: Valores padrões do BRC básico para cada variável de acordo com o número de variáveis presentes na função de entrada.

✓ 2 variáveis:
var1 = 0000 0000 0000 0101 = 5
var2 = 0000 0000 0000 0011 = 3

Figura 2: BRC considerando duas variáveis.

Quando a função contem mais de cinco variáveis, o método proposto de geração de BRC básico é modificado. Assim, é utilizado um vetor, como fora explicado anteriormente. Para as primeiras cinco variáveis, são utilizados os mesmo valores correspondentes para as cinco variáveis, indicados na Fig.1. Estes valores são armazenados em todas as posições do vetor de acordo com a variável correspondente. Então, para a primeira variável é atribuído o valor 1431655765 em todas as posições do vetor. Isto é realizado para todas as quatro próximas variáveis, alterando apenas o valor da atribuição para cada caso. Para as próximas variáveis, é realizado um processo onde é concatenado os valores 0 e -1 em cada posição do vetor. O número de concatenações de 0 e -1 necessárias é indicado pelo cálculo de 2n-5. Este processo assemelha-se ao método de montagem de uma tabela verdade, onde cada variável é representado por uma sequência de bits nas colunas da tabela. A Fig.3 justifica o motivo de ser usado os valores 0 e -1 durante a geração do BRC básico.

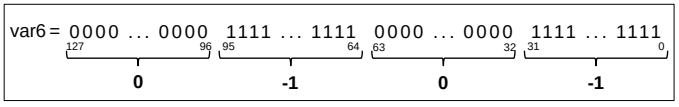

Figura 3: Divisão de uma sequência de bits e associando-se a um número inteiro equivalente.

Todo este processo de geração do BRC básico para uma função que contém mais de cinco variáveis é ilustrado na Fig.4(a). Se alguma variável da função for negada, o processo de geração do BRC básico é o mesmo. Será atribuído, para a variável negada, o mesmo valor apresentado na Fig.1. A principal diferença é que os bits que são 0 tornam-se 1, e aqueles que são 1 tornam-se 0. No caso em que a função tenha mais do que cinco variáveis, é realizado o mesmo procedimento de concatenação explicado anteriormente. A única diferença é a ordem de concatenação dos valores que são atribuídos no vetor. Em primeiro lugar, é

concatenado o valor -1. Depois o valor 0 é concatenado. A Fig.4(b) mostra uma função aleatória que contém duas variáveis negadas, a terceira e a sétima.

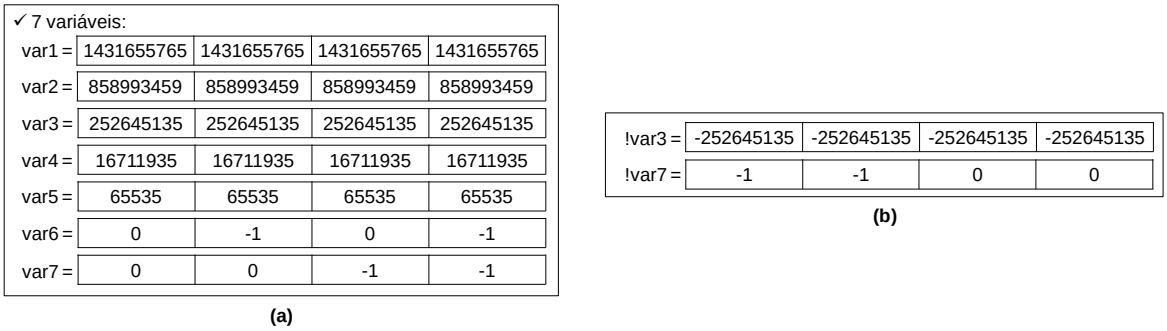

Figura 4: Vetores com o valor para cada variável: (a) variáveis não negadas; (b) variáveis negadas.

Após gerado o BRC básico de cada variável, é realizado um procedimento utilizando as operações lógicas AND e OR para gerar o BRC da função de entrada. A Exp.1 mostra a função utilizada como exemplo.

A*C*E*F + A*B*F + A*B*!C + D*E*!G (Exp. 1)

Primeiramente, é obtido o BRC dos produtos pela operação AND bit a bit, entre cada inteiro presente em uma posição do vetor com outro inteiro da posição correspondente do vetor seguinte. Depois, é realizada a operação OR bit a bit entre os BRC gerados anteriormente. A Fig.6(a) mostra a geração do BRC de um produto através da operação lógica AND realizada entre cada inteiro presente no vetor. Depois de gerar todos os BRC de cada produto, é realizada a operação lógica OR entre cada posição dos vetores de cada BRC desses produtos. Este processo é ilustrado na Fig.6(b), o que também mostra o BRC da função indicada na Exp.1.

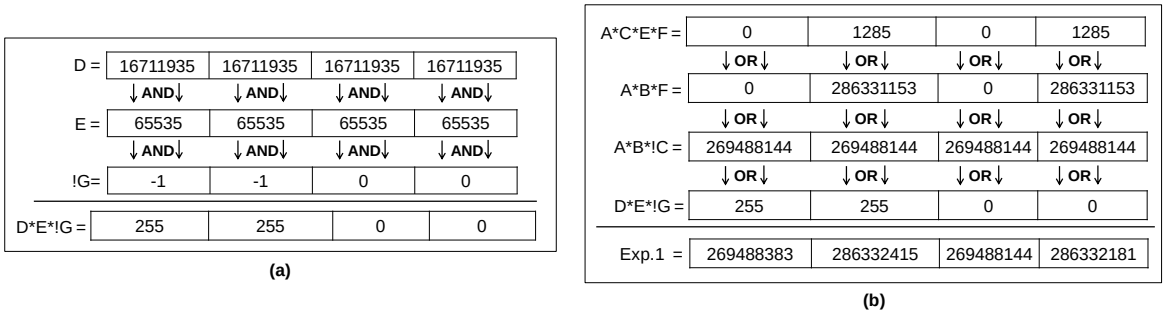

Figura 6: Geração do BRC: (a) para o produto D*E*!G; (b) para a Exp.1.

3 RESULTADOS EXPERIMENTAIS

O método proposto foi integrado na ferramenta *Soptimizer*. Com o objetivo de avaliar a eficiência do método proposto, foram utilizados como *benchmarks* todas as funções *p-class* de quatro entradas. Este conjunto de funções é composto por 3982 funções Booleanas. Também foram escolhidas, randomicamente, 54 funções lógicas com seis entradas, chamado de Random6 *benchmark*. Separadamente, mais três funções foram escolhidos para análise. A XOR 4 foi escolhida porque é extremamente utilizada em vários circuitos, tais como somadores e multiplicadores. Funções F5 e F13 (KAGARIS, 2007), foram escolhidas porque contém um grande número de variáveis, se comparando com as funções de 4 entradas da *p-class*. A Tab.1 apresenta o resultado obtido em tempo total de execução. A coluna “Sem BRC” informa o tempo total de execução da ferramenta Soptimizer utilizando a antiga versão de algoritmo para a comparação de

equivalência das funções. A coluna “Com BRC” mostra o tempo total de execução onde o método proposto é utilizado. A coluna “Redução” reporta o percentual de ganho e perda em cada caso. Estes testes foram realizados em um computador com um processador Intel Pentium Dual Core T2360 1.73GHz, 2Gb de memória e Windows 7 Ultimate 64bits.

Como pode ser percebido nos resultados da Tab.1, para os *benchmarks p-class*, Random6, XOR 4 e F13, o tempo total de execução da ferramenta *Soptimizer* é menor quando se utiliza o algoritmo proposto. No entanto, para o *benchmark* F5, o tempo total de execução obtido é maior quando utilizado o algoritmo proposto. A principal razão para isso, é que o *benchmark* F5 contém cubos grandes, com poucas variáveis. Nesta situação, o algoritmo proposto apresenta uma desvantagem se comparado com a antiga estratégia usada pela ferramenta *Soptimizer*. Todo o processo para gerar o BRC e compará-los quando necessário é mais demorado do que apenas comparar diretamente os produtos armazenados em estruturas vetoriais. Nosso método é capaz de fornecer melhores resultados quando há vários cubos a serem verificados na expressão.

Tabela 1: Tempo total de execução obtido pela ferramenta *Soptimizer* com e sem o método proposto.

Benchmark	Número de Funções	Número de Variáveis	Sem BRC	Com BRC	Redução
p-class	**3.982**	**4**	2193 ms	1599 ms	27,1%
Random6	**54**	**6**	189 ms	156 ms	17,5%
XOR 4	**1**	**4**	57 ms	47 ms	17,6%
F5	**1**	**8**	78 ms	100 ms	-28,3%
F13	**1**	**10**	546 ms	320 ms	41,4%

4 CONCLUSÃO

Este trabalho apresentou um método para geração de um *Boolean Representation Code* que pode ser eficientemente utilizado para representar funções Booleanas. Em um primeiro momento, o método proposto foi integrado à ferramenta *Soptimizer* para validar o processo de otimização de rede de transistores. O método foi validado utilizando diversas funções com diferentes quantidades de variáveis.

Os resultados demonstram que o algoritmo pode minimizar o tempo total de execução quando incorporado em uma ferramenta de CAD. No estudo de caso realizado, foi possível alcançar uma taxa de ganho de 27,1% em tempo total de execução quando considerando o *benchmark p-class* de quatro entradas.

Como trabalho futuro, mais testes serão realizados considerando diferentes funções de referência. Além disso, pretende-se incorporar o método proposto em outros algoritmos de avaliação Booleanas desenvolvidos pelo grupo, especialmente os relacionados com mapeamento tecnológico.

5 REFERÊNCIAS

Da Rosa Junior et al. *Switch Level Optimization of Digital CMOS Gate Networks*. In: **10th IEEE International Symposium on Quality Electronic Design**, 2009, p. 324-329.

Possani, V. N.; Souza, R. S.; Domingues Jr., J. S.; Agostini, L. V.; Marques, F. S.; Da Rosa Junior, L. S. *Optimizing Transistor Networks Using a Graph-Based Technique*. In: **Journal of Analog Integrated Circuits and Signal Processing**, Springer, 2012.

D. Kagaris et al. *A Methodology for Transistor-Efficient Supergate Design*. In: **IEEE Transactions on Very Large Scale Integration Systems**, 2007, p. 488-492