MANIPULAÇÃO EFICIENTE DE FUNÇÕES BOOLEANAS EM ARQUITETURAS DE 64BITS ATRAVÉS DO USO DE BOOLEAN REPRESENTATION CODE

RENATO SOUZA; FELIPE MARQUES; LEOMAR DA ROSA JR

Universidade Federal de Pelotas – {rdssouza, felipem, leomarjr}@inf.ufpel.edu.br

1. INTRODUÇÃO

Nos atuais cenários de competitividade, as empresas são obrigadas a lançar novos produtos inovadores para conquistar cada vez mais consumidores. Consequentemente, grandes dificuldades acabam sendo encontradas devido à adaptação de novos parâmetros da tecnologia. Nesse cenário, as ferramentas de CAD (*Computer Aided Design*) vêm contribuindo para que os desenvolvedores aumentem a eficiência e diminuam a complexidade em um projeto.

Neste contexto, foi desenvolvida uma ferramenta chamada *Soptimizer* (POSSANI, 2012) que implementa um método baseado em grafos para gerar redes de transistores. A partir de uma expressão Booleana, obtém-se um grafo, onde cada aresta 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.

Portanto, este trabalho propõe um método para representar funções lógicas de forma eficiente. O método proposto é denominado de *Boolean Representation Code* (BRC). Para verificar a eficiência do método, ele foi incorporado à ferramenta *Soptimizer* com o objetivo de verificar, de uma forma rápida e segura, as otimizações realizadas pela mesma.

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 de 64 bits. 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 64, grandeza que representa o tamanho do 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 sete variáveis, o resultado do calculo $2^7/64$ 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 sejam 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 seis 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:	✓ 6 variáveis:
var1 = 2L	var1 = 12L	var1 = 240L	var1 = 65280L	var1 = 4294901760L	var1 = -4294967296L
	var2 = 10L	var2 = 204L	var2 = 61680L	var2 = 4278255360L	var2 = -281470681808896L
		var3 = 170L	var4 = 43690L	var3 = 4042322160L	var3 = -71777214294589696L
			var3 = 52428L	var4 = 3435973836L	var4 = -1085102592571150096L
				var5 = 2863311530L	var5 = -368934881474191032 4L
					var6 = -6148914691236517206L

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 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0011 = 12L

var2 = 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0101 = 10L

Figura 2: BRC considerando duas variáveis.

Quando a função contem mais de seis variáveis, o método proposto de geração de BRC básico é modificado. Assim, é utilizado um vetor, como fora explicado anteriormente. Para as primeiras seis variáveis, são utilizados os mesmo valores correspondentes para as seis 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 -4294967296L em todas as posições do vetor. Isto é realizado para todas as cinco próximas variáveis, alterando apenas o valor da atribuição para cada caso. Para as próximas variáveis, é realizado um processo onde é concatenado os valores -1 e 0 em cada posição do vetor. O número de concatenações de -1 e 0 necessários é indicado pelo cálculo de 2^{n-6}. 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 -1 e 0 durante a geração do BRC básico.

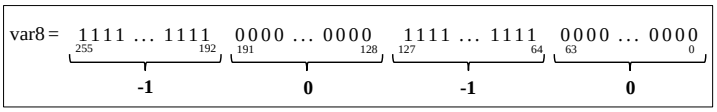

Figura 3: Divisão de uma sequência de bits e associando-se a um número inteiro equivalente.

Todo este processo de geração do BRC básico para uma função que contém mais de seis 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 0, depois o -1. A Fig.4(b) mostra os BRC básicos negados.

✓ 7 variáveis:

var1 =	-1	0
var2 =	-4294967296L	-4294967296L
var3 =	-281470681808896L	-281470681808896L
var4 =	-71777214294589696L	-71777214294589696L
var5 =	-1085102592571150096L	-1085102592571150096L
var6 =	-368934881474191032 4L	-368934881474191032 4L
var7 =	-6148914691236517206L	-6148914691236517206L

(a)

✓ 7 variáveis negadas:

!var1 =	0	-1
!var2 =	4294967295L	4294967295L
!var3 =	281470681808895L	281470681808895L
!var4 =	71777214294589695L	71777214294589695L
!var5 =	1085102592571150095L	1085102592571150095L
!var6 =	368934881474191032 3L	368934881474191032 3L
!var7 =	614891469123651720 5L	614891469123651720 5L

(b)

Figura 4: Vetores com o valor para cada variável: (a) variáveis diretas; (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), resultando no BRC da função indicada na Exp.1.

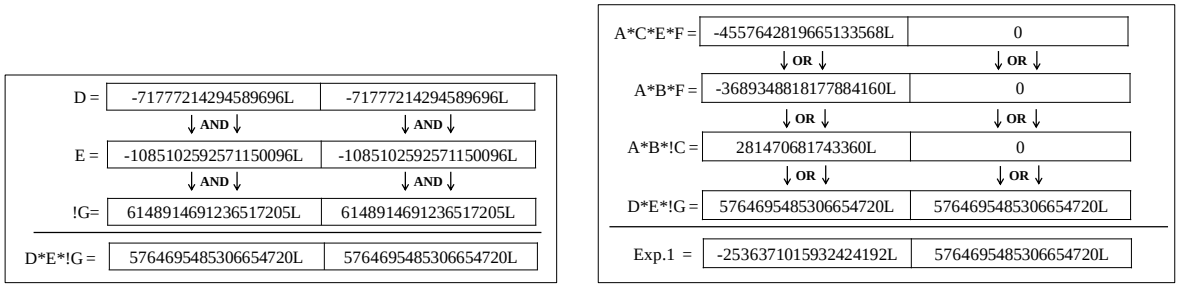

Figura 6: Geração do BRC: (a) para o produto D*E*!G; (b) para a Exp.1.

3. RESULTADOS EXPERIMENTAIS

O método proposto foi integrado na ferramenta *Soptimizer*. Para avaliar a eficiência do método proposto, foram utilizados como *benchmarks* todas as funções *p-class* de quatro entradas, no qual é composta por 3982 funções Booleanas. O *benchmark* demonimado de Random6 é composto por 54 funções lógicas com seis entradas cada, escolhidas randomicamente. Também foram escolhidas duas funções que contém um grande número de variáveis, que é as funções F5 e F13 (KAGARIS, 2007). Por último, foi utilizada a XOR 4 que foi escolhida porque é utilizada em vários circuitos, tais como somadores e multiplicadores.

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.

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. Porem, 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 BRC.

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%

Além disso, foi realizado um estudo para avaliar o tempo de execução individual para cada uma das funções lógicas da *p-class* de quatro entradas. Desta forma, foi possível realizar uma melhor análise em quais funções obteve-se uma redução na execução. A Figura 8 mostra um gráfico que resume estes resultados.

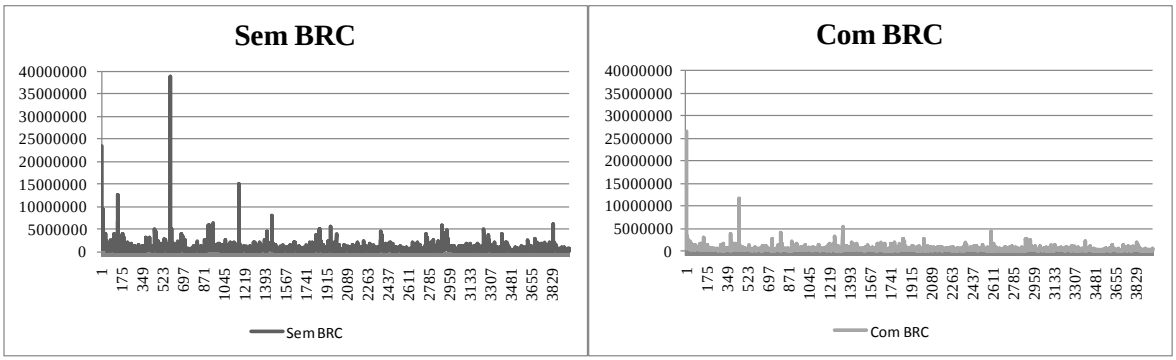

Figura 7: Comparação do tempo de execução de cada função da *p-class* na ferramenta *Soptimizer*.

4. CONCLUSÕES

Este trabalho apresentou um método eficiente, denominado *Boolean Representation Code*, com a finalidade de representar funções Booleanas. O método proposto foi integrado à ferramenta *Soptimizer* para validar o processo de otimização de rede de transistores. Para validar o método foi utilizado diferentes funções com diversos números de transistores.

Os resultados mostram que o algoritmo pode reduzir o tempo total de execução da ferramenta. No estudo de caso realizado, foi possível obter uma taxa de ganho de 27,1% em tempo total de execução quando considerando o *benchmark p-class* de quatro entradas. Também foi possível notar que com a utilização do método proposto a ferramenta *Soptimizer* torna-se, capaz de realizar algumas otimizações algébricas que não eram possíveis ao utilizar a solução anterior.

5. REFERÊNCIAS BIBLIOGRÁFICAS

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