Revista Brasileira de Informática na Educação, Volume 18, Número 2, 2010

Karma: um ambiente para o aprendizado de síntese de funções Booleanas

Carlos Eduardo Klock
ceklock@inf.ufrgs.br

Renato Perez Ribas
rpribas@inf.ufrgs.br

André Inácio Reis
andreis@inf.ufrgs.br

Instituto de Informática
Universidade Federal do Rio Grande do Sul
Caixa Postal 15064
91501-970 Porto Alegre - RS - Brasil
Fone +55 (51) 3308 6168

Resumo *Este artigo apresenta um ambiente computacional para síntese e análise de funções lógicas. O ambiente proposto, chamado de Karma, oferece um conjunto de ferramentas de apoio (CAD) para o ensino e formação de estudantes de computação e eletrônica na área de conhecimento de síntese lógica e circuitos digitais, mais precisamente em relação aos fundamentos de construção, otimização e análise de funções Booleanas. O ambiente foi desenvolvido em plataforma Java e não há similar disponível no mercado ou na academia. A ferramenta Karma, em suas mais diversas versões, vem sendo utilizada em cursos de graduação há alguns anos, e atualmente encontra-se estável e abrangente, embora sempre seja possível agregar novos métodos e algoritmos relacionados com o seu propósito. O Karma pode ser utilizado em aulas presenciais (versão GUI – interface gráfica do usuário), no ensino à distância (versão applet para navegadores WEB), na formação de novos pesquisadores na área de síntese lógica e programadores Java através do desenvolvimento de novos módulos, bem como para auxílio na produção de resultados científicos através do seu uso por comandos de console e scripts.*

Palavras-Chave: *funções Booleanas, síntese lógica, circuitos digitais, mapa de Karnaugh, CAD educacional, EAD*

Abstract *This paper presents an environment for logic synthesis and analysis of Boolean functions. The proposed environment, called Karma, provides a set of computer-aided design (CAD) tools for learning and formation of computing and electronic students in the field of logic synthesis and digital circuits, particularly in relation to the fundamentals of building, optimization and analysis of logic functions. This environment has been developed in Java platform and there is no similar environment available for academic or commercial use. Karma tool, in different versions, has been used in undergraduate courses for some years, and presents today a stable and complete release, although it is always possible to include new methods and algorithms related to the environment goal. Karma is quite suitable for presential classes through the graphical user interface (GUI) use, for distance teaching and E-learning through WEB browser applets, for education of young researchers in the logic synthesis field and beginner Java programmers through the development of new modules, as well as to support research data production through scripts and console commands.*

Keywords: *Boolean functions, logic synthesis, digital circuits, Karnaugh map, CAD tool, E-learning.*

1. Introdução

Conhecimentos sobre lógica Booleana fazem parte dos fundamentos da Computação e da Eletrônica Digital. Tais conhecimentos são explorados pela área de Síntese Lógica, na automatização da construção de circuitos digitais. Esta área estuda como abstrair e representar funções lógicas (ou Booleanas), bem como manipular, transformar, analisar e otimizar tais funções para o projeto de circuitos [1].

De forma simplista, pode-se dividir a síntese lógica em duas categorias ou níveis maiores: o tratamento de funções lógicas e a construção de circuitos digitais. No primeiro caso, o objetivo é obter equações lógicas que representem funções geralmente com poucas variáveis, e com características específicas segundo os métodos de síntese adotados: equações de dois níveis ou multi-nível, uso de função 'OU-exclusiva', número mínimo de literais e/ou termos, e assim por diante [2]. No caso da síntese de circuitos o foco principal é o mapeamento tecnológico, ou seja, como dividir de forma eficiente a construção de um circuito digital em portas lógicas que o compõe [3]. O ambiente Karma, aqui proposto, atua na primeira categoria, no nível das funções lógicas.

Os conceitos citados acima são um tanto abstratos, sendo difícil fornecer exemplos suficientemente didáticos para que os alunos possam adquirir confiança para resolver os problemas de forma independente. A disponibilidade de uma ferramenta computacional que possibilite ao estudante a interação com estes conceitos auxilia significativamente no processo de aprendizado [4][5]. Existe um grande número de ferramentas educacionais propostas na literatura [5]-[8] e disponíveis na Internet [9]-[11], principalmente em relação ao ensino do mapa de Karnaugh [12], talvez devido a questão visual inerente a este método de síntese de função. Porém, tais ferramentas ficam limitadas a tópicos específicos, não apresentando aos estudantes a abrangência da área. Por outro lado, há ferramentas mais completas do ponto de vista de métodos de síntese lógica, como o SIS [13] e o ABC [14], ambos da Universidade de Berkeley, mas que foram criadas para uso de pesquisadores e desenvolvedores de circuitos, sendo inadequadas para o ensino dos conceitos básicos deste tópico. Por exemplo, essas ferramentas não dispõem de interface gráfica para o usuário (GUI), isto é, os algoritmos são executados apenas por comandos no console.

Com o objetivo de auxiliar o aprendizado de síntese de funções lógicas, vem sendo desenvolvido há alguns anos um programa de computador chamado Karma [15], que integra várias ferramentas e módulos para os mais diferentes métodos de geração de equações, otimizações e análises. Este ambiente dispõe de um solucionador de mapas de Karnaugh, visualizador de diagramas de decisão binária (BDDs), conversor de formatos de descrição lógica, exercícios (jogos) didáticos sobre mapa de Karnaugh e tabelas de cobertura, verificador de equivalência lógica de funções, análise da probabilidade de ocorrência dos níveis lógicos dos sinais, entre outras, que serão discutidas ao longo deste artigo [1][2].

O ambiente Karma foi desenvolvido em plataforma Java para ser independente de sistema operacional, e poder ser executado em navegadores de Internet através de versão *applet*. Dessa forma, o Karma é útil tanto para aulas presenciais quanto para o ensino à distância (EAD). Uma vez que este ambiente está modularizado e há inúmeros métodos e técnicas de geração e análise de equações Booleanas, o projeto tem se mostrado bastante eficiente também para a formação de desenvolvedores Java, bem como para a introdução do conteúdo de síntese lógica para novos alunos de iniciação científica no grupo de pesquisa.

Neste artigo serão inicialmente apresentados, de forma sucinta, os fundamentos da área de síntese lógica para melhor compreensão das funcionalidades e benefícios do ambiente proposto. A seguir, na Seção 3, o Karma é descrito com uma breve apresentação dos seus módulos. Na Seção 4 é discutido o uso deste ambiente no contexto do ensino de graduação na Universidade Federal do Rio Grande do Sul. Por fim, na Seção 5, as conclusões do trabalho são apresentadas.

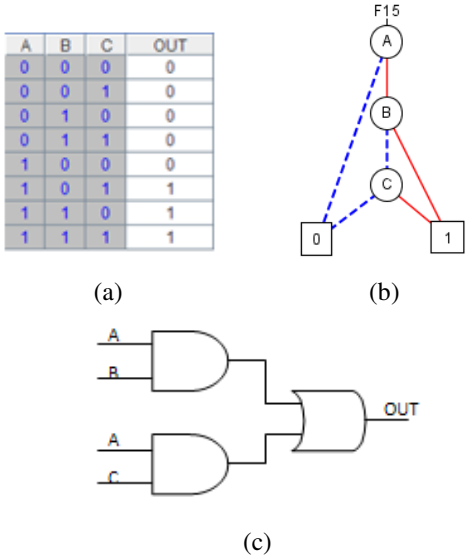
2. Fundamentos

Uma função Booleana é uma função onde tanto as entradas (domínio) quanto as saídas (imagem) são compostas de variáveis Booleanas. Uma variável Booleana é uma variável que assume valores binários B = {falso, verdadeiro}, ou {0, 1}. Assim, uma função de N variáveis de entrada associa o valor de saída '0' ou '1' a uma sequência de N bits (vetor) que representa uma combinação dos valores binários das variáveis de entrada. Quando o valor de saída associado a esta sequência de N bits é 0 ou 1, esta sequência é chamada de maxtermo ou mintermo, respectivamente [2].

Uma função Booleana pode ser completamente especificada ou incompletamente especificada. Uma função completamente especificada é uma função onde para cada combinação de entradas a saída pertence ao conjunto {0, 1}. Considere o símbolo '-'ou 'X' como um valor indefinido (*don't care*). Uma função incompletamente especificada é uma função onde para cada combinação de entradas a saída pertence ao conjunto {0, 1, X}, ou seja, existem combinações de entradas onde o valor da saída não importa (X). A não ser

que seja citado, é assumido por padrão que uma função Booleana é completamente especificada.

Funções Booleanas podem ser representadas de muitas maneiras diferentes, cada uma com suas vantagens e desvantagens dependendo do tipo de manipulação a ser executada. Dentre as formas mais comuns de representação de funções lógicas estão a tabela verdade, equação Booleana, portas lógicas, lista de termos (mintermos ou maxtermos) e diagrama de decisão binária (BDD). Algumas formas alternativas, menos usuais, são o número inteiro (binário, decimal, hexadecimal), BLIF [13], árvore binária e AIG [16]. Na Figura 1 são ilustrados alguns desses formatos.



(a) (b)

(c)

Figura 1 – Exemplos de descrição de funções lógicas: (a) tabela verdade, (b) BDD, (c) circuito com portas lógicas.

É importante distinguir as definições de 'variável' e 'literal'. Um literal é cada uma das instâncias de uma variável presente em uma expressão Booleana. Normalmente, o objetivo dos projetistas de circuitos digitais é reduzir o número de literais na expressão, pois isto tem relação com o número de chaves (transistores) utilizados, ou seja, corresponde ao tamanho do circuito (área física) e, por sua vez, pode ser associado também com o consumo de energia do circuito.

A técnica mais popular para ensino de minimização de funções lógicas é o uso de mapas de Karnaugh em conjunto com o método Quine-McCluskey [17]. A partir do mapa de Karnaugh é possível obter uma expressão Booleana na forma de 'soma-de-produtos' (SOP) e/ou 'produto-de-somas' (POS) [2]. Porém, devido ao procedimento visual desse método, este é limitado a funções de 4 a 5 variáveis. O algoritmo genérico para qualquer quantidade de variáveis na função é conhecido como método Quine-McCluskey [17]. Estes processos, mapa de Karnaugh e método Quine-McCluskey, podem ser divididos basicamente em dois passos principais [1]: (1) agrupamento de termos, eliminando alguns literais nos termos vizinhos e gerando o conjunto de implicantes primos da função; e (2) a tabela de cobertura onde é escolhido o conjunto mínimo de implicantes primos que cobrem todos os termos da função.

Funções na forma de SOP e POS correspondem a construção de circuitos digitais de dois níveis lógicos. Funções com lógica multi-nível podem ser obtidas através da aplicação de métodos de fatoração. A fatoração é amplamente utilizada para minimizar a quantidade de literais de uma função. Lógica multi-nível é preferida em circuitos integrados para redução de área e consumo de energia, sendo este normalmente o objetivo dos projetistas de circuitos integrados. Existem muitas técnicas para obter resultados de fatoração, sendo necessário o uso de heurísticas para reduzir o grau de complexidade, reduzindo também o esforço computacional (tempo de execução e consumo de memória) [18][19].

Implementações baseadas em multiplexadores, por sua vez, são aplicadas tipicamente em componentes FPGA [2]. Qualquer função Booleana de N variáveis pode ser construída usando multiplexadores de $2^{(N-1)}$ entradas. Considere um multiplexador de 4 entradas (E1, E2, E3, E4), ou 'MUX_4x1', que pode ser representado pela seguinte equação lógica:

S = !c1.!c2.E1 + !c1.c2.E2 + c1. !c2.E3 + c1. c2.E4 (1)

sendo as variáveis 'c1' e 'c2' os sinais de controle deste multiplexador. Na Tabela 1 é apresentada uma função lógica construída com o MUX_4x1, nas suas mais diversas formas de configuração. A escolha das variáveis a serem conectadas nos controles do multiplexador impacta diretamente no número de constantes (0 e 1) nas entradas do mesmo. Isso tem influência direta na redução do número de chaves (ou transistores) quando da construção física deste circuito.

Tabela 1 – Implementação da função lógica 'S=b.(a.c+!a.!c)+!b.!c', em diferentes versões, utilizando um MUX_4x1.

E1	E2	E3	E4	c1	c2	Função Lógica
!c	!c	!c	c	a	b	S = !a.!b.!c+!a.b.!c+a.!b.!c+a.b.c
1	0	!b	b	a	c	S = !a.!c+a.!c.!b+a.c.b
1	0	!a	a	b	c	S = !b.!c+b.!c.!a+b.c.a

A síntese de funções Boolenas baseada no operador lógico OU-exclusivo, ou que contempla este, pode ser utilizada para arquiteturas específicas, principalmente em operadores aritméticos, uma vez que um somador é construído utilizando preferencialmente este tipo de

operador lógico. Um método simples e direto é derivado do algoritmo Quine-McCluskey [20]. Este método encontra-se disponível no ambiente Karma.

A probabilidade dos valores das funções lógicas (sinais de saída dos circuitos digitais) de apresentarem os níveis lógicos alto (1) e baixo (0), segundo a probabilidade desta ocorrência nas variáveis da função (sinais de entrada do circuito), é bastante útil para a análise de circuitos em relação ao seu comportamento lógico e elétrico. Por exemplo, para uma função AND de duas variáveis, sendo que cada uma tem 50% de chance de apresentar o valor lógico '1', a probabilidade da função ser '1' é de 25%. Técnicas de construção de circuitos de baixo consumo estático e de teste de sistemas em hardware, entre outras, fazem bastante uso desta informação em seus métodos de análise.

Por fim, funções podem ser agrupadas em classes de equivalência [21] através de simples rearranjos, dando nome a classes de funções equivalentes como, por exemplo:

- Classe P – permutando as variáveis de entrada;
- Classe Ni – negando as variáveis de entrada;
- Classe NP – negando e/ou permutando variáveis de entrada;
- Classe No – negando a variável de saída da função;
- Classe NN - negando as variáveis de entrada, e/ou negando a variável de saída da função.
- Classe PN – permutando as variáveis de entrada e negando a variável de saída da função;
- Classe NPN - negando e/ou permutando variáveis de entrada, e/ou negando a variável de saída da função.

Exemplo de funções equivalentes a uma dada função 'S1 = (a.b)+c' são ilustradas na Tabela 2.

Tabela 2 – Exemplos de equivalências de funções lógicas: as funções indicadas possuem as seguintes equivalências para 'S1 = (a.b)+c'.

Função	Equivalência
S2 = (c.a)+b	P, NP, PN, NPN
S3 = (a.!b)+!c	Ni, NN, NP, NPN
S4 = !((a.b)+c)	No, NN, PN, NPN
S5 = !((!a.b)+c)	NN, NPN
S6 = (b.!c)+!a	NP, NPN
S7 = !((c.b)+a)	PN, NPN
S8 = !(c.!b)+!a)	NPN

3. Ambiente Karma

O ambiente Karma é dividido em vários módulos ou ferramentas, que serão brevemente apresentados a seguir. Na Figura 2 é mostrada a interface principal do ambiente.

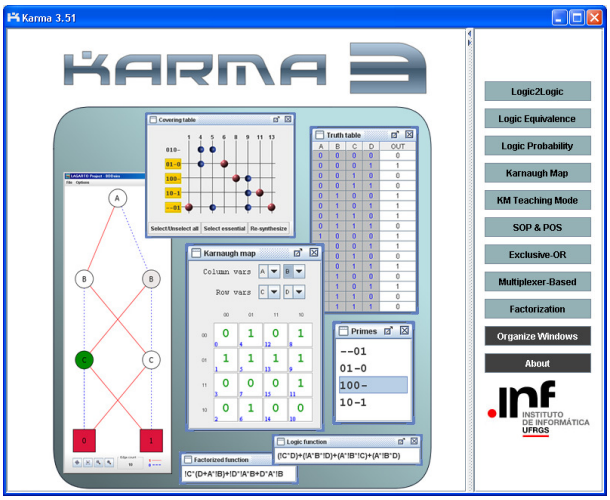


Figura 2 – Karma: interface gráfica do usuário (GUI).

3.1 *Logic2Logic* (Conversor de Formatos)

Conforme mencionado anteriormente, funções Booleanas podem ser representadas de diferentes formas. Para estudantes que estão tendo um primeiro contato com a área da Síntese Lógica, o entendimento da relação entre os diferentes formatos de representação de funções lógicas nem sempre é simples e evidente. Além disso, converter de um formato para outro é mais rápido e mais confiável usando uma ferramenta computacional de apoio para tal finalidade.

O módulo *Logic2Logic*, ilustrado na Figura 3, fornece essas funcionalidades. Os formatos suportados atualmente por este módulo são os seguintes:

- tabela verdade;
- BLIF [13];
- termos (índices ou linhas da tabela verdade);
- número inteiro (em qualquer base de 2 a 16);
- expressão Booleana;
- BDD (descrição textual).

Por ser extenso o número de possibilidades de formatos de descrição de funções, este módulo está permanentemente em desenvolvimento e sendo atualizado. Encontra-se em andamento o acréscimo dos formatos de árvore binária e diagrama AIG. Além disso, o *Logic2Logic* é utilizado para converter funções de

entrada definidas em vários outros módulos do Karma.

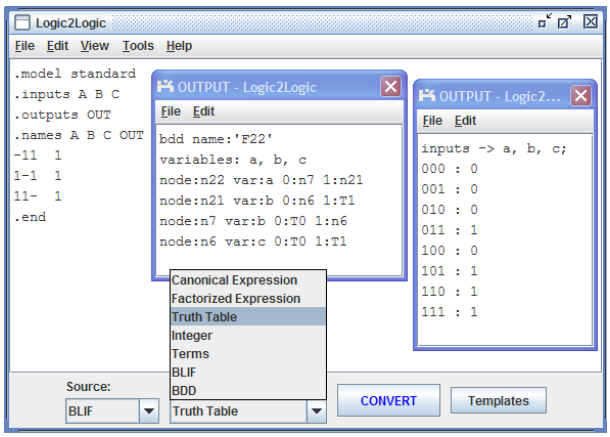


Figura 3 – Ilustração do módulo *Logic2Logic*.

3.2 *Logic Equivalence* (Equivalência Lógica)

Dadas duas representações de funções, o módulo de equivalência lógica permite identificar se há alguma equivalência entre as funções [21] segundo as classes descritas na Seção 2, inclusive se são funções idênticas, apenas descritas de forma diferente. Este módulo é muito útil para verificar resultados após manipulação e síntese de funções. A interface deste módulo é ilustrada na Figura 4.

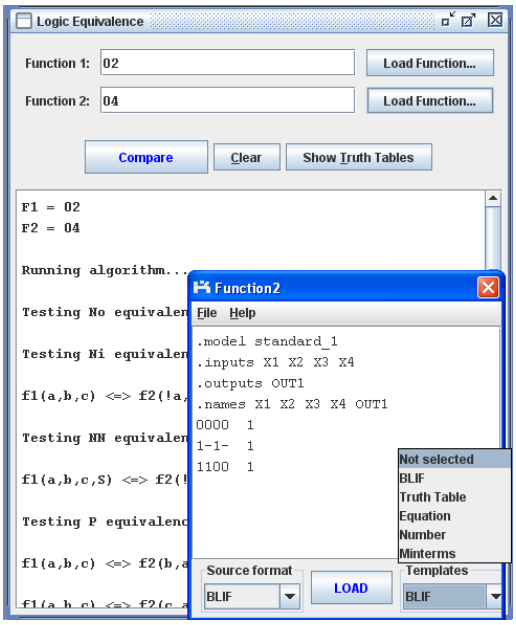


Figura 4 – Ilustração do módulo *Logic Equivalence*.

3.3 *Logic Probability* (Probabilidade Lógica)

Para a probabilidade de ocorrência do valor lógico '1' (nível lógico alto) relacionado a cada variável de entrada, é dada a probabilidade de ocorrência de cada combinação das variáveis nas linhas da tabela verdade, assim como a probabilidade de ocorrência do valor lógico '1' para a função especificada, conforme ilustrado na Figura 5.

Uma vez que diferentes formatos de descrição de função são permitidos, é interessante que os estudantes percebam que tais probabilidades de ocorrência são propriedades da função lógica, independente da maneira como esta é descrita. Este módulo também pode ser usado para ilustrar certas situações onde algumas combinações de entrada nunca acontecem.

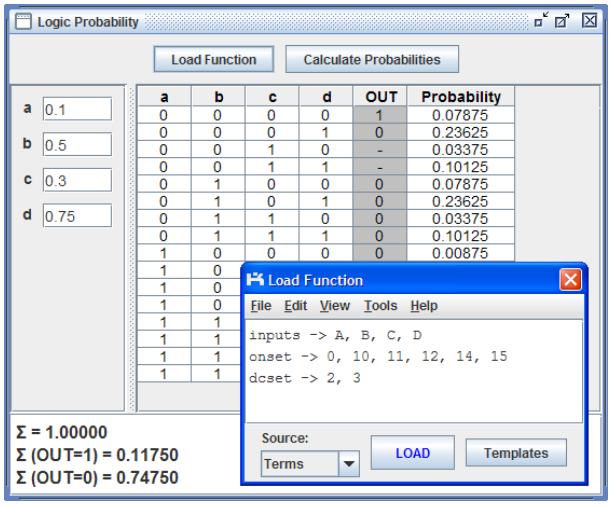


Figura 5 – Ilustração do módulo *Logic Probability*.

3.4 *Karnaugh Map* (Mapa de Karnaugh)

Este módulo representa a origem do projeto Karma [15], e é o mais aprimorado dos módulos do ambiente. É bastante útil para o aprendizado do método de mapa de Karnaugh [12] e do algoritmo Quine-McCluskey [17]. Uma das facilidades deste módulo, quando comparado com outras ferramentas de mapa de Karnaugh, é a possibilidade de definir o comportamento da função através da tabela verdade, diretamente no próprio mapa, ou através da descrição da função usando os formatos disponíveis no módulo *Logic2Logic*.

O conjunto de valores lógicos {0, 1, X} é permitido. A quantidade de variáveis também pode ser alterada, sendo que são suportadas até oito variáveis devido às limitações de visualização gráfica do mapa. Além disso, é possível mudar a ordem das variáveis para possibilitar outras formas de visualização de agrupamentos de

mintermos.

Após executar o algoritmo Quine-McCluskey, o módulo disponibiliza diferentes formas de visualização dos resultados, incluindo uma lista de implicantes primos que, quando selecionados, exibem os mintermos correspondentes no mapa, uma expressão na forma soma-de-produtos e na forma fatorada, ilustração do BDD da função, tabela de cobertura [22], e uma janela com os passos do algoritmo Quine-McCluskey. Veja ilustração na Figura 6.

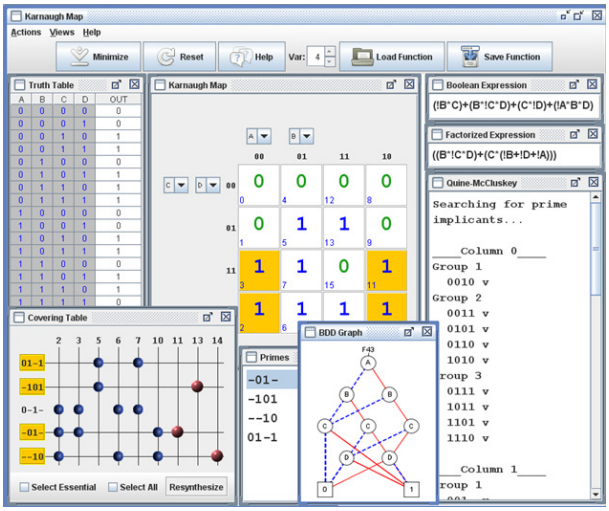


Figura 6 – Ilustração do módulo *Karnaugh Map*.

3.5 *KM Teaching Mode* (Exercícios de Ensino para o Mapa de Karnaugh)

Este módulo é muito interessante para praticar os conceitos que envolvem o método do mapa de Karnaugh na forma de jogos. Nos exercícios, as funções podem ter de 2 a 8 variáveis. O conceito de vizinhança (adjacência) de mintermos pode ser encontrado em diferentes exercícios. Os exercícios (jogos) disponíveis são os seguintes:

• **ver mintermos adjacentes**: clicando sobre uma posição no mapa são exibidos os mintermos vizinhos;

• **procurar mintermos adjacentes**: uma posição é realçada no mapa e o usuário deve encontrar os mintermos vizinhos;

• **ver cubo**: o usuário marca um ou mais mintermos no mapa, o computador verifica se eles formam um cubo, e exibe uma mensagem dizendo se está correto;

• **procurar cubo**: a expressão Booleana de um cubo é exibida e o usuário deve encontrar os mintermos que compõe o cubo no mapa de Karnaugh;

• **cubo implicante**: o usuário recebe uma função escolhida aleatoriamente pela ferramenta e deve indicar um conjunto de mintermos no mapa. A ferramenta diz as propriedades do conjunto selecionado, indicando se o conjunto de mintermos forma ou não um cubo. Caso um cubo seja formado é dito se o cubo é implicante, primo e/ou essencial;

• **questões sobre cubos**: Similar ao exercício anterior, mas a ferramenta escolhe o cubo aleatoriamente e o usuário indica as propriedades do cubo. O aprendiz deve dizer se o cubo é implicante, implicante primo ou implicante primo essencial;

• **cobertura de função**: o usuário deve responder questões sobre um conjunto de cubos que cobrem uma função, mas onde não necessariamente todos cubos são implicantes ou primos;

• **tabela de cobertura**: ajuda o usuário a entender o funcionamento de uma tabela de cobertura [22]. A tabela de cobertura é construída interativamente quando o usuário clica nas células do mapa de Karnaugh.

A Figura 7 ilustra uma das interfaces deste módulo, jogo 'questões sobre cubos'.

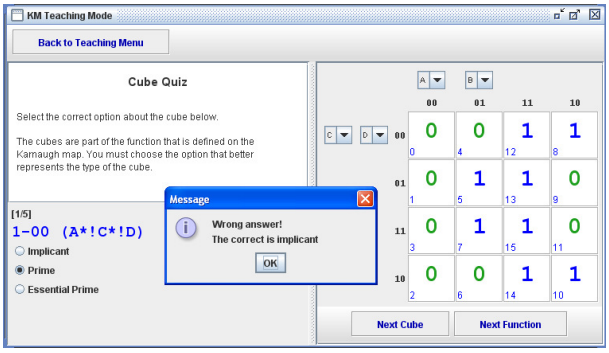


Figura 7 – Ilustração do módulo *KM Teaching Mode:* exercício *Cube Quiz*.

3.6 *SOP & POS* (Soma-de-Produtos e Produto-de-Somas)

Este módulo executa o algoritmo Quine-McCluskey e fornece a função Booleana, na sua forma direta e complementada (negada), nos formatos de soma-de-produtos (SOP) e produto-de-somas (POS). Estes formatos lógicos de dois níveis são geralmente minimizados em termos do número total de literais na função.

Outra otimização, não usual, pode ser obtida focando na minimização da quantidade de literais em cada termo da função, ou melhor, visando a eliminação dos termos com maior número de literais [23]. Neste caso podem existir duas soluções diferentes de acordo com a

minimização que se quer obter, e este módulo fornece as duas soluções, conforme ilustrado na Figura 8.

O módulo *SOP & POS* é útil para exercícios onde o usuário está interessado no resultado da síntese, e não em visualizar os agrupamentos, como ocorre no mapa de Karnaugh.

A síntese SOP é o princípio básico de configuração aplicado em dispositivos lógicos programáveis (PLDs) como PLA, PAL e CPLD [2].

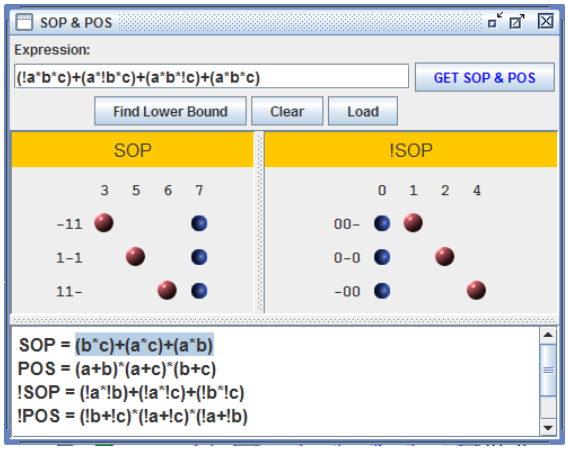


Figura 8 – Ilustração do módulo *SOP & POS*.

3.7 *Exclusive-OR* (OU-Exclusivo)

O algoritmo Quine-McCluskey pode ser usado para síntese de funções considerando operações com OU-exclusivo (EXOR). Para isso, uma estrutura de dados específica é utilizada [20].

Este módulo permite aos estudantes comparar os resultados do formato SOP convencional, que usa apenas operações OR (soma) e AND (produto), com os resultados da expressão que inclui a operação EXOR. O uso da operação EXOR representa uma possível minimização na quantidade de literais da equação.

3.8 *Multiplexer-Based* (Síntese com Multiplexador)

A síntese de funções lógicas baseada no uso de multiplexadores, conforme mencionado na Seção 2, é executada pelo módulo *Multiplexer-Based*. Para uma dada função lógica, definida em um dos formatos suportados pelo *Logic2Logic*, este módulo apresenta todas as possíveis soluções de acordo com as variáveis usadas como sinais de controle do multiplexador, e a variável usada para configurar as entradas do multiplexador (de forma direta ou complementar), juntamente com as constantes '0' e '1'. Todas essas possíveis configurações das conexões do multiplexador para a implementação de determinada função lógica são apresentadas na janela de saída (relatório) do módulo, conforme ilustrado na Figura 9.

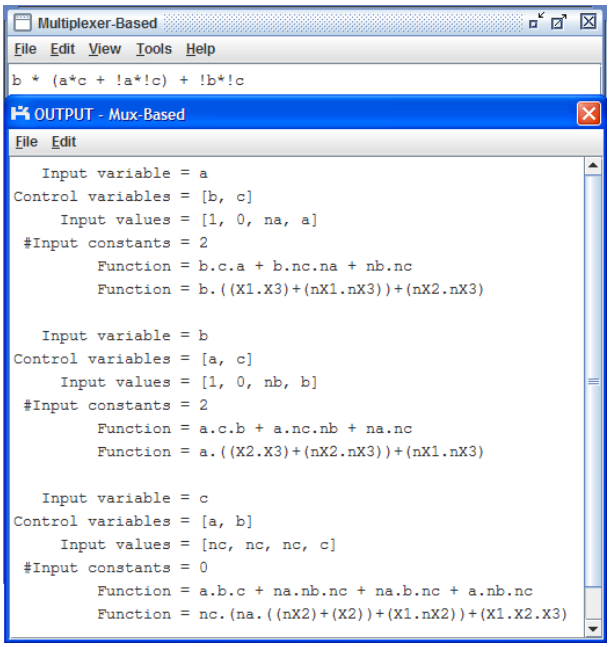


Figura 9 – Ilustração do módulo *Multiplexer-Based*, usando função descrita na Tabela 1 da Seção 2.

3.9 *Factorization* (Fatoração)

Formatos SOP, POS e *multiplexer-based* representam equações Booleanas de dois níveis lógicos. Uma maneira de reduzir a quantidade de literais na equação é através do procedimento de fatoração, resultando em uma representação com mais de dois níveis lógicos, conforme discutido na Seção 2.

Algoritmos exatos para fatoração são conhecidos apenas para funções na categoria *read-once*, ou seja, funções que podem ser descritas através de expressões onde cada variável aparece uma única vez (incluindo as polaridades direta e negada) [24]. O algoritmo proposto por Lawler [25] declara fornecer solução exata para qualquer função, mas tem uma complexidade que o torna inadequado para tratar funções com mais que 4 variáveis, podendo falhar também para funções com 4 ou menos variáveis, dependendo da função. A fatoração disponível no Karma é inédita e apresentou bons resultados em termos de minimização de equações e tempo de execução [26]. Também se pretende incluir na ferramenta o método de fatoração descrito em [19], que gera resultados ainda melhores.

3.10 Comandos de Console

Além da interface gráfica do usuário (GUI) e da versão *applet* desta interface para execução em navegadores WEB, por vezes é necessário executar um comando qualquer de forma rápida e direta. Com o uso de comandos de console (ou 'linhas de comando') não é necessário acessar um módulo específico do Karma, descrito acima, para obter resultados. Os comandos de console permitem a execução de vários algoritmos através da digitação de comandos numa janela de texto. Entre as funcionalidades disponíveis estão a minimização e fatoração de expressões Booleanas, resolução de tabelas de cobertura e a conversão de formatos lógicos.

4. Experiência do Karma no Ensino e Formação de Estudantes

A ferramenta Karma, em sua primeira versão, tinha por foco apenas a resolução do mapa de Karnaugh [15]. Desde então recebeu o módulo dos jogos pedagógicos e os demais módulos descritos foram sendo agregados no ambiente. Esta ferramenta foi sendo adotada desde a sua criação nas aulas de graduação das disciplinas de Técnicas Digitais (INF01118) para o curso de Ciência da Computação e de Circuitos Digitais (INF01056) para o curso de Engenharia de Computação, ambos da Universidade Federal do Rio Grande do Sul [27]. O Karma tem sido divulgado como material de apoio para a referência [2], especificamente para os capítulos 1 e 3.

De maneira geral, inicialmente o ambiente é utilizado para os alunos verificarem as respostas de exercícios feitos manualmente e para tirar dúvidas de resultados para situações menos correntes na aplicação dos métodos. Após, o Karma passa a ser uma ferramenta de apoio para desenvolvimento de circuitos digitais, quando é desejável alguma análise ou otimização lógica.

O uso do módulo *Exercícios de Ensino para o Mapa de Karnaugh* (seção 3.5), tem um aceite muito interessante, talvez pelo fato de tratarmos hoje de uma geração de jovens aficionados por vídeo games, e ser esta uma forma bastante lúdica para tratar do assunto [6]. Os exercícios *Questões sobre Cubos* e *Cobertura de Função*, foram utilizados em algumas aulas como 'jogo de auditório', onde o exercício era projetado na tela e o primeiro a responder disputava o ponto: 'acerto' significava um ponto para o participante que seguia no jogo; 'erro' eliminava o participante do jogo. A cada tentativa um novo desafio (nova função ou cubo) era apresentado. O resultado desta experiência para o aprendizado e retenção da informação foi bastante positivo, pois em um curto período de tempo os alunos dedicaram atenção especial (foco) nos princípios que envolvem o mapa de Karnaugh e com bastante entusiasmo e motivação [28].

Por fim, percebeu-se que o ambiente Karma tem sido um veículo bastante eficiente para a formação de novos bolsistas de iniciação científica (alunos de graduação) no que se refere a programação em linguagem Java e no entendimento dos conceitos básicos de síntese lógica. Os novos bolsistas do grupo de pesquisa recebem como tarefa o desenvolvimento de um novo módulo relacionado com o assunto, para ser então agregado ao ambiente. Isso faz com que o desafio do aprendizado tenha um objetivo prático e útil, contribuindo para a motivação do estudante.

Esta prática, desenvolvimento de novos módulos, é realizada desde os primórdios do Karma. Novos bolsistas desenvolvem módulos e/ou pacotes que podem ser executados independentes do Karma. Estes novos módulos são feitos utilizando algumas classes úteis do projeto, que já estão bem documentadas. Para tanto, os novos bolsistas dispõem do auxílio de outros bolsistas que já trabalharam com o desenvolvimento do Karma. Os módulos resultantes, que inicialmente são independentes, podem ser agregados ao Karma na forma de um módulo completo ou de funcionalidades para um módulo já existente.

5. Conclusão

Apesar de já existirem algumas ferramentas educacionais com o objetivo de ensinar mapas de Karnaugh e conceitos de síntese lógica, estas ferramentas têm um foco diferente da presente proposta. Enquanto muitas ferramentas são específicas para resolução de mapas de Karnaugh, o ambiente Karma dispõe de um conjunto bastante amplo de ferramentas para descrição, síntese e análise de funções Booleanas. Ambientes de síntese lógica, como as ferramentas SIS e ABC de Berkeley, são voltadas para a produção de resultados de pesquisa, não sendo apropriadas para o ensino de conceitos básicos e formação inicial de estudantes de graduação. O Karma vem sendo utilizado regularmente em disciplinas de graduação da Universidade Federal do Rio Grande do Sul, e está disponível através do site do Logic Circuit Synthesis labs [29].

6. Agradecimentos

Os autores agradecem a diversos alunos que, de uma forma ou outra, contribuíram para o desenvolvimento de parte deste ambiente computacional, em particular a Felipe Ribeiro Schneider, Marcos Ledur, Mateus

Volkmer Nunes, Dionatan de Souza Moura e Vinicius Callegaro. Este projeto foi parcialmente financiado pelo CNPq, pela empresa Nangate A/S e pelo Sétimo Framework Programme (FP7) da Comunidade Européia através do contrato 248538 - Synaptic.

7. Referências

[1] L. T. Wang, Y. W. Chang, K. T. Cheng. Electronic Design Automation: Synthesis, Verification and Test. Morgan Kaufmann, cap.6, pp.299-404, E.U.A., 2009.

[2] F. R. Wagner, A. I. Reis, R. P. Ribas. Fundamentos de Circuitos Digitais. Sagra Luzzatto, Porto Alegre, 2006.

[3] F. Marques, O. Martinello Jr., R. P. Ribas, L. Rosa Jr., A. I. Reis. Mapeamento Tecnológico no Projeto de Circuitos Integrados Digitais. *Desafios e Avanços em Computação - o estado da arte*, ed. da Universidade Federal de Pelotas, pp.177-196. 2009.

[4] P. A. Mlsna, E. Liszewski. Effectiveness of Karnaugh Mapplet Use in Student Learning of Digital Logic Skills. In *Proceedings of the 2005 American Society for Engineering Education Annual Conference & Exposition*, E.U.A., 2005.

[5] H. Pomares; C. García-García; I. Rojas; M. Damas; J. González; J. P. Florido; J. Urquiza. Teaching digital systems design with a new didactic environment through the Internet. *Research, Reflections and Innovations in Integrating ICT in Education*, vol.2, pp.1021-1026, 2009.

[6] L. A. Belfore II, A. B. Adcock, G. S. Watson. Introducing digital logic and electronics concepts in a game-like environment. In Proceedings of the 2009 Spring Simulation Multiconference, E.U.A., 2009.

[7] KMVQL: a visual query interface based on Karnaugh map. In *Proceedings of the working conference on Advanced visual interfaces*, pp.243-250, Itália, 2008.

[8] A. Sena, M. Torres. EasyKarnaugh 3.0 Uma ferramenta computacional para o auxílio no ensino de Mapas de Karnaugh em Lógica Digital. In *WEI - XVII Workshop sobre Educação em Computação*, Bento Gonçalves, RS, 2008.

[9] R. Kovacevic. Karnaugh Map Minimizer. http://k-map.sourceforge.net/, Abril 2010.

[10] R. Sasamori. KarnaughMap4. http://www.puz.com/sw/karnaugh/index.htm, Abril 2010.

[11] R. Sandige. Karnaugh Map Explorer 1.0. http://courseware.ee.calpoly.edu/~rsandige/KarnaughExplorer.html, Abril 2010.

[12] M. Karnaugh. The map method for synthesis of combinational logic circuits. *Trans. AIEE, Commun. & Electron.*, vol.72, no.1, pp.593-598, 1953.

[13] E. Sentovich; K. Singh; L. Lavagno; C. Moon; R. Murgai; A. Saldanha; H. Savoj; P. Stephan; R. K. Brayton; A. Sangiovanni-Vincentelli. SIS: A system for sequential circuit synthesis. *Memo. UCB/ERL M92/41*, 1992.

[14] Berkeley Logic Synthesis and Verification Group. ABC: A System for Sequential Synthesis and Verification, Release 70911.

[15] C. E. Klock; F. R. Schneider; M. N. Gomes; D. S. Moura; R. P. Ribas; A. I. Reis. KARMA: a didactic tool for two-level logic synthesis. *IEEE International Conference on Microelectronic Systems Education (MSE 07)*, 2007.

[16] L. Hellerman. A Catalog of Three-Variable Or-Invert and And-Invert Logical Circuits, *Electronic Computers, IEEE Transactions,* vol.EC-12, no.3, pp.198-223, June 1963.

[17] E. J. McCluskey. Minimization of Boolean functions. *Bell Syst. Tech. J.*, vol.35, no.5, pp.1417-1444, 1956.

[18] A. Mintz, M. C. Golumbic. Factoring Boolean functions using graph partitioning. *Discrete Applied Mathematics*, n.149, p.131-135, May 2005.

[19] A. I. Reis, A. Rasmussen, L. R. Junior, R. P. Ribas. Fast Boolean Factoring with Multi-Objective Goals. In *International Workshop on Logic & Synthesis - IWLS 2009,* Berkeley, CA, USA.

[20] B. H. Turton. Extending Quine-McCluskey for Exclusive-Or Logic Synthesis. *IEEE Trans. On Educ.*, vol.39, no.1, pp.81-85, 1996.

[21] P.Molitor and J.Mohnke. Equivalence checking of digital circuits: fundamentals, principles, methods. 265 pages. Springer, 2004.

[22] O. Coudert. On solving covering problems. In *Proceedings of the 33rd Annual Design Automation Conference,* pp.197-202, Las Vegas, United States, June, 1996.

[23] F. R. Schneider, R. P. Ribas, S. S. Sapatnekar, A. I. Reis. Exact Lower Bound For The Number Of Switches In Series To Implement A Combinational Logic Cell. *International Conference on Computer*

Design - ICCD 2005, San Jose, CA, USA, 2005.

[24] M. C. Golumbic, A. Mintz, U. Rotics. Factoring and recognition of read-once functions using cographs and normality and the readability of functions associated with partial k-trees, *Discrete Applied Mathematics*, vol. 154, issue 10, pages 1465-1477, June 2006.

[25] E. L. Lawler. An approach to multilevel Boolean minimization. *Journal of the ACM*, v.11, n.3, p.283-295, New York, USA, July 1964.

[26] V. Callegaro Et al. A Kernel-based Approach for Factoring Logic Functions. *Microelectronics Students Forum SForum 2009,* s.n., 2009, Natal, RN, Setembro 2009.

[27] Universidade Federal do Rio Grande do Sul. http://www.ufrgs.br, Abril, 2010.

[28] D. Fenker, H. Schütze. O Fascínio da Surpresa. *Revista Mente & Cérebro*, Ediouro Publicações, nº 193, Jan. 2009.

[29] Logic Circuit Synthesis labs. http://www.inf.ufrgs.br/logics, Abril 2010.