Método de Síntese Lógica Aproximada Dois-Níveis Baseado na Inserção e Remoção de Cubos.

Gabriel Ammes Pinho[1] **e Renato P. Ribas**[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{gabriel.ammes,rpribas}@inf.ufrgs.br

Resumo. *Computação aproximada é um paradigma que reduz a complexidade de um sistema ao custo de reduzir a sua precisão. Circuitos digitais aproximados podem ser tratados em diferentes níveis de projeto, desde a sua definição algorítmica e arquitetural até a definição final no nível de transistores. A Síntese lógica aproximada (ALS) é uma das etapas de projeto que busca construir um circuito lógico que não seja logicamente equivalente a definição original mas consegue otimizações em desempenho, área e consumo. Este trabalho propõe um novo método de ALS dois-níveis baseado nas técnicas de inserção e remoção de cubos. Resultados experimentas mostram reduções significantes no número de literais e no tempo de execução comparados ao estado-da-arte. As soluções obtidas apresentam uma redução no número de literais de até 38%, 56% e 93% com frequência de erro de 1%, 3% e 5%*

Abstract. *A Two-Level Approximate Logic Synthesis Method Based on Insertion and Removal of Cubes - Approximate computing is an attractive paradigm that trades off the system complexity and accuracy. Aproximate digital circuits can be treated at different design levels, from algorithm and architectural definition to transistor level. The approximate logic synthesis (ALS) represents a design step that aims to build a logic circuit that is not logically equivalent to the original definition but can further improve performance, area, and power. This work proposes a new two-level ALS method based on cube insertion and removal procedures. Experimental results have shown significant literal count and runtime reduction compared to the state-of-the-art approach. The obtained solutions have presented a literal number reduction up to 38%, 56%, and 93% concerning an error rate of 1%, 3%, and 5%, respectively.*

1. Introdução

Computação aproximada é um paradigma que permite que um sistema tenha uma execução imprecisa ou inexata com o objetivo de otimizar o seu desempenho e sua eficiência energética. Este paradigma pode ser aplicado em diversos níveis computacionais, desde o nível de algoritmos e compiladores, passando pelo nível arquitetural e chegando ao nível de circuitos e transistores [Mittal 2016]. Quando este paradigma é aplicado em sistemas que executam funções resilientes a erros, é possível otimizar o sistema sem degradar de forma crítica a operação desejada. Este trabalho foca no uso de computação aproximada no nível de circuitos, em particular, nos circuitos integrados digitais.

Ferramentas computacionais fornecem um fluxo altamente automatizado para o desenvolvimento de projetos de circuitos integrados. Este fluxo pode ser dividido em três

passos principais: síntese de alto nível, síntese lógica e síntese física. A síntese lógica, em particular, tem como objetivo otimizar a lógica do circuito e implementá-lo em uma dada tecnologia alvo. Esta etapa do fluxo de projeto é executada sobre representações dois-níveis ou multinível que implementam os blocos combinacionais de um dado circuito.

A aplicação de computação aproximada no nível de circuitos consiste em obter uma implementação que não é logicamente equivalente à especificação mas consegue realizar otimizações em área, desempenho e consumo de energia. Diversos trabalhos propõem técnicas para aproximar um circuito de forma automática através de modificação sistemática do funcionamento de um circuito genérico sem exceder uma dada restrição de erro. Devido à similaridade em técnicas, estruturas de dados e objetivos de otimização com a etapa de síntese lógica, a geração automática de circuitos aproximados é frequentemente chamada de síntese lógica aproximada.

Métodos de síntese lógica tradicional para construção de circuitos dois-níveis podem ser utilizados para síntese de componentes programáveis CPLDs, bem como parte de métodos para síntese de circuitos multinível. Além da geração de expressões aproximadas do tipo soma-de-produtos (SOP) e produto-de-somas (POS), técnicas de aproximação para circuitos dois-níveis poder ser exploradas nesses dois cenários. Além disso, o entendimento dos conceitos e técnicas relacionados à aproximação dois-níveis pode contribuir significativamente para futuros estudos sobre a aproximação de circuitos multinível.

Este trabalho propõe um método para aproximar circuitos dois-níveis que tem como entrada uma SOP e um dado limite de frequência de erro, e gera uma expressão aproximada com um número de literais reduzido e que respeita o dado limite de erro. O método proposto foi desenvolvido com a intenção de ser escalável em relação à quantidade de erros permitidos, possibilitando uma inserção de mais erros do que é observado em outros trabalhos que abordam o mesmo problema.

2. Conceitos Preliminares

Uma breve revisão de fundamentos de síntese lógica aproximada e de métricas de erro é apresentada a seguir, bem como a terminologia adotada, buscando facilitar o entendimento do método de síntese lógica aproximada dois-níveis proposto.

Uma *variável* corresponde ao símbolo utilizado para representar sinais de entrada e saída. A ocorrência de uma variável é chamada de *literal*, podendo ser um literal de entrada ou de saída, depende da respectiva variável. Um literal de entrada pode ser direto ou complementado (negado). Um produto de literais onde cada variável aparece no máxima uma vez é chamado de *cubo*. O caso particular onde um cubo contém um literal para cada literal de entrada e somente um literal de saída é chamado de *mintermo*. Um conjunto de cubos que cobre todos os mintermos é chamado de *cobertura*. O caso especial de cobertura onde todos os cubos são referentes a mesma saída, ou seja, uma cobertura de saída única, pode ser representado por uma *soma-de-produtos* (SOP). Em geral, quando se representa a cobertura com uma SOP os literais de saída são omitidos. O tamanho de um cubo é igual ao número de mintermos cobertos por ele. A expansão de um cubo consiste em remover um de seus literais de entrada, tornando-o em um cubo maior.

No contexto da aproximação de circuitos, múltiplas métricas foram propostas com o objetivo de quantificar e controlar a quantidade de erro inserido [Froehlich et al. 2019].

Neste trabalho utilizamos a métrica de *Error Rate* (ER) para limitar a ocorrência de erros. A métrica ER consiste na frequência em que os erros ocorrem ou na probabilidade de um erro ocorrer para um dada combinação de entrada. Desta forma, esta métrica é calculada considerando a razão entre o número combinações de entrada que levam a erros e o número total de possíveis combinações de entrada.

Considerando a síntese lógica aproximada, uma combinação de entrada que leva a uma ou mais saídas incorretas é definida como uma combinação de entrada errônea (EIC, do inglês *erroneous input combination*). Por exemplo, se dois mintermos levam a saídas incorretas mas contém os mesmo literais de entrada, somente um EIC será levado em conta. Além disso, o número de erros (NE) em uma dada cobertura é igual aos seu número de EICs. Quando o NE é utilizado como métrica de erro, o seu valor pode ser tanto arbitrário quanto relativo ao número de entradas do circuito. Para o segundo caso, o NE é igual a $2^n * er$, onde n é o número de entradas e er o limite de ER.

3. Estado-da-Arte

Em [Su et al. 2020], Su *et al.* apresentam um método de busca heurística para resolver o problema de síntese lógica aproximada dois-níveis (2L-ALS) usando ER como métrica de erro. Este trabalho pode ser considerado o estado-da-arte para este problema e é apresentado a seguir.

O principal objetivo do método de Su é identificar um conjunto de combinações de entrada para terem sua entrada complementada de 0 para 1 que maximize a redução de literais da cobertura. Este conjunto de combinações é chamada de SICC e é equivalente a um conjunto de EICs. Eles propõe neste trabalho a estrutura de dados *SICC-cube tree* (SCT) que agrupa um conjunto de EICs com os cubos dependem destes EICs para serem inseridos na cobertura. Uma SCT consiste em uma árvore de dois-níveis, onde a raiz contém o conjunto de EICs e as folhas contém os cubos a serem inseridos na cobertura. O número de EICs na raiz é igual ao número de erros inseridos na cobertura.

Um cubo deve satisfazer duas condições para ser inserido como folha de uma SCT. Inicialmente, pelo menos um cubo deve ser removido da cobertura quando este é adicionado na cobertura. Por fim, o número de literais do cubo removido deve ser maior do que do cubo a ser inserido. Estas condição são impostas para garantir que os cubos nas folhas da SCT levem a otimizações no número de literais.

Inicialmente, todos os possíveis cubos de uma dada função utilizando um diagrama de Hasse. Esses cubos são utilizados para construir as SCTs. Em seguida, as SCTs obtidas são combinadas, pois existem SCTs com menos erros que o limite estabelecido. Por fim, a SCT que leva a maior redução de literais é escolhida.

O procedimento para estimar a redução de literais de uma dada SCT contém três passos principais. Como a inserção de um cubo não garante que outros sejam removidos, o primeiro passo consiste em identificar quais cubos da cobertura podem ser removidos quando todos os cubos folhas de uma SCT são inseridos na cobertura. Além disso, inserir todos os cubos folhas pode aumentar o número de literais. Desta forma, o segundo passo é identificar quais cubos folhas são necessários para remover os cubos identificados no primeiro passo. Por fim, a redução de literais é calculada subtraindo o número de literais nos cubos removidos e nos cubos inseridos.

Além deste método base, os autores apresentam quatro técnicas de aceleração para permitir o uso em circuitos maiores.

1. Como a complexidade temporal do algoritmo-base cresce exponencialmente com o número de erros, o número de erros permitidos para cada execução é limitado em dois, gerando assim coberturas parcialmente aproximadas. Todas estas soluções parciais geradas são aproximadas iterativamente até que o erro máximo permitido seja atingido.
2. Com a utilização da primeira técnica, é gerada uma quantidade muito grande de soluções parciais é criado, tornando custosa a aproximação de todas estas soluções. Desta forma, somente as duas soluções parciais com a maior redução de literais para um dado número de erros são aproximadas.
3. Como o número de erros permitidos para cada aproximação é dois, é necessário combinar as SCTs que inserem somente um erro. Para diminuir o número de SCTs combinadas, somente um subconjunto de todas as SCTs são utilizadas na etapa de combinação. Para isso, inicialmente são estimados os literais das SCTs geradas sem combinação. Em seguida, duas SCTs são combinadas caso a primeira esteja entre as 25% melhores e a segunda entre as 80% melhores SCTs.
4. Considerar todos os cubos em um diagrama de Hasse implica em um alto custo computacional. Para reduzir este custo, eles consideram somente os cubos do diagrama de Hasse que são parentes dos cubos presentes na cobertura. Estes cubos podem ser vistos como os cubos obtidos removendo literais de entrada ou inserindo literais de saídas nos cubos da cobertura.

Os valores utilizados nas três primeiras técnicas de aceleração foram obtidas utilizando análises empíricas. Mesmo com a utilização destas técnicas, o aumento no número de erros permitidos continua impactando no tempo de execução desta técnica.

4. Trabalho Proposto

Em [Shin and Gupta 2010], os autores apresentam duas técnicas para a aproximação de SOPs: a inserção de cubos na SOP, invertendo as saídas de mintermos de 0 para 1, e a remoção de cubos da SOP, modificando estas saídas de 1 para 0. Eles também propõem um experimento para comparar estas duas técnicas e sugerem que a inserção de cubos leva a resultados melhores do que a sua remoção. Baseado nessa conclusão, trabalhos relacionados têm focado na inserção de cubos. A seguir, apresentamos um exemplo de aproximação utilizando estas duas técnicas individualmente e em conjunto.

Para a SOP $x_1 * \overline{x_2} + x_0 * x_2$ representada pelo mapa de Karnaugh (diagrama de Veitch) ilustrado na Figura 1a, suas aproximações considerando um limite de número de erros igual a dois através da inserção de cubos, da remoção de cubos e de ambas em conjunto, são mostradas na Figura 1b, na Figura 1c e na Figura 1d, respectivamente. Os mintermos destacados nos mapas de Karnaugh representam os mintermos aproximados. Na Figura 1b, quando os mintermos $x_0 * \overline{x_1} * \overline{x_2}$ e $\overline{x_0} * x_1 * x_2$ têm a saída complementada de 0 para 1, é possível inserir os cubos x_0 e x_1. Esta inserção implica na remoção dos cubos $x_1 * \overline{x_2}$ e $x_0 * x_2$, reduzindo o número de literais de 4 para 2. Na Figura 1c, quando os mintermos $\overline{x_0} * x_1 * \overline{x_2}$ e $x_0 * x_1 * \overline{x_2}$ têm a saída complementada de 1 para 0, é possível remover diretamente o cubo $x_1 * \overline{x_2}$, reduzindo o número de literais de 4 para 2. Na Figura 1d, quando os mintermos $x_0 * \overline{x_1} * \overline{x_2}$ e $\overline{x_0} * x_1 * \overline{x_2}$ têm a saída complementada de 0 para

1 e de 1 para 0, respectivamente, é possível inserir o cubo x_1 e remover os cubos $x_1 * \overline{x_2}$ e $x_0 * x_2$, reduzindo o número de literais de 4 para 1.

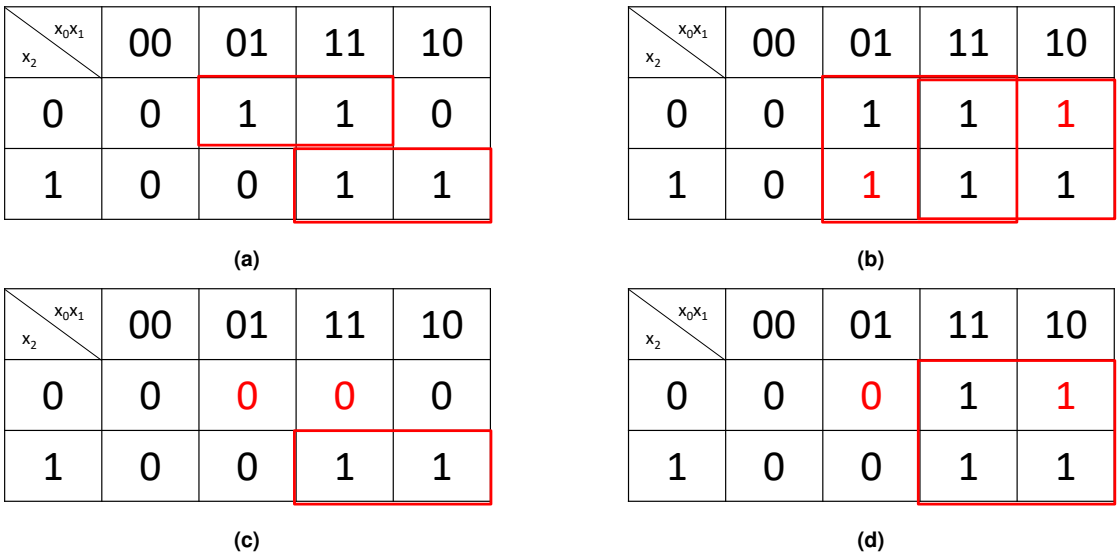

Figura 1. Exemplos de coberturas aproximadas comparando o uso da inserção de cubos, da remoção de cubos e de ambas em conjunto.

No presente trabalho é proposta uma abordagem para a resolução do problema de 2L-ALS usando ER como restrição de erro utilizando a inserção e remoção de cubos de forma conjunta como técnica de aproximação. O circuito dois-níveis é representado por um cobertura $\mathcal{C}$ e o limite de ER por er. Conforme mostrado na Figura 1, aplicar as duas técnicas em conjunto pode levar a melhores resultados do que utilizar somente a inserção de cubos, como é feito em outros trabalhos. Inicialmente, apresentamos uma descrição geral do método proposto e, em seguida, os algoritmos para aproximar a cobertura.

4.1. Descrição Geral

O fluxo geral é mostrado no Algoritmo 1. A cobertura é armazenada usando dois mapas da Karnaugh. O primeiro mapa liga os cubos aos mintermos que são cobertos por somente este cubo. O segundo mapa liga cada mintermo coberto pelo procedimento, contendo os cubos da solução que o cobrem. O método proposto aplica inicialmente um procedimento de inserção de cubos, gerando múltiplas coberturas parcialmente aproximadas com menos do limite de número de erros. A cobertura inicial $\mathcal{C}$ é aproximada utilizando um método de inserção com o limite de dois erros, criando duas coberturas parcialmente aproximadas (soluções parciais) com um e dois erros, respectivamente. Este comportamento é similar a primeira técnica de aceleração apresentada na Seção 3. Para cada solução parcial, o método de remoção de cubo é aplicado, considerando o número de erros ainda disponível, que corresponde a diferença entre o erro máximo permitido e o erro já inserido na solução parcial. Esta estratégia permite escapar de mínimos locais que ocorrem quando somente a técnica de inserção é aplicada.

Para evitar um aumento substancial na complexidade de tempo e de espaço, somente a cobertura inicial e as modificações necessária para transformá-la nas soluções parciais são armazenadas. Para isso, uma solução contém os cubos que devem ser inseridos e removidos da cobertura, além da redução de literais e dos EICs inseridos. Na

Algoritmo 1: Método de 2L-ALS Proposto

Entrada: Uma cobertura otimizada $\mathcal{C}$ e um limite de ER *er*
Saída: Cobertura aproximada $\mathcal{C}$'

```
1  e ← er * 2^n ;
2  Vetor sols com e+1 vetores de Soluções;
3  sols_0 ← ∅ (solução vazia);
4  para i ← 0 até e faça
5      topS ← as duas melhores soluções em sols_i;
6      para cada solution s em topS faça
7          modifyCover(C, s);
8          (s1, s2) ← cubeInsertion(C, 2, s);
9          sols_{i+1} ← sols_{i+1} ∪ s1;
10         sols_{i+2} ← sols_{i+2} ∪ s2;
11         s3 ← cubeRemoval(C, e-i, s);
12         sMax ← max(s1, s2, s3);
13         se sMax > bestSol então bestSol ← sMax;
14         restoreCover(C, s);
15     fim
16 fim
17 retorna espresso(modifyCover(C,best));
```

linha 2, o vetor de $sols$ contém todas as soluções parciais obtidas, enquanto o vetor $sols_i$ contém as soluções parciais com i erros. Desta forma, como o vetor $sols_0$ não insere nenhum erro na cobertura, sendo ele iniciado como vazio. Para cada vetor $sols_i$, as duas melhores soluções são selecionadas na linha 5 para serem aproximadas, similar a segunda técnica de aceleração apresentada na Seção 3.

Para modificar a cobertura $\mathcal{C}$ considerando uma solução s é aplicada a função modifyCover na linha 7 do Algoritmo 1. Esta função insere e remove os cubos de $\mathcal{C}$, tornando-a uma cobertura aproximada. Na direção contrária, a função restoreCover desfaz estas modificações. O método de inserção de cubos, na linha 8, retorna duas soluções (*s1* e *s2*) com um e dois erros que são armazenadas em $sols_{i+1}$ e $sols_{i+2}$, na linha 9 e 10. O método de remoção de cubos, na linha 11, pode inserir até $e - i$, onde e é o número máximo de erros permitidos, retornando a solução *s3*. Caso umas das três soluções geradas reduza mais literais do que a melhor solução obtida até o momento, esta solução é armazenada em $bestSol$. No final, a melhor solução encontrada é aplicada na cobertura inicial $\mathcal{C}$ e otimizada utilizando a ferramente Espresso [Brayton et al. 1984].

4.2. Método de Inserção de Cubos

O método de inserção de cubos desenvolvido é baseado no método de busca heurística apresentado em [Su et al. 2020]. Desta forma, a principal estrutura de dados para realizar a aproximação é a SICC-cube tree (SCT). A ideia geral é gerar múltiplas SCTs com cubos que insiram até dois erros e selecionar as SCTs que levam a uma maior redução de literais.

O Algoritmo 2 apresenta o fluxo do método de inserção de cubos. Na linha 1, as SCTs são geradas. Os cubos utilizados para a geração destas SCTs são todos os cubos obtidos através da expansão dos cubos da cobertura $\mathcal{C}$. A utilização da expansão de cubos é uma simplificação da quarta técnica de aceleração apresentada na Seção 3. Quando um cubo expandido é usado como folha de uma SCT, é garantido que este cubo irá remover pelo menos um cubo da cobertura e irá reduzir o número de literais. Desta forma, a

Algoritmo 2: Função cubeInsertion

Entrada: Uma cobertura otimizada $\mathcal{C}$, um limite de número de erros e e a solução atual s

Saída: Duas soluções com erro 1 e 2

1 *trees* ← generateSCT($\mathcal{C}$, e, s.EIC);
2 (*s1*, *s2*) ← combineAndEstimate($\mathcal{C}$, *trees*);
3 *s1* ← updateSolution(*s1*, *s*);
4 *s2* ← updateSolution(*s2*, *s*);
5 retorna (*s1*, *s2*);

verificação a ser feita é relacionada ao número de erros inseridos. O número de erros inseridos por um cubo é igual ao número de mintermos cobertos por ele que não são cobertos por $\mathcal{C}$ que não tiveram a combinação de entrada considerada como EIC anteriormente.

Na linha 2, a combinação de SCTs e a estimativa da redução de literais de todas as SCTs é realizada. Inicialmente, todas as SCTs obtidas sem combinação têm a redução de literais estimada. As SCT que inserem somente um erro são combinadas, considerando a quarta técnica de aceleração apresentada na seção 3. As SCTs com um e dois erros que removem mais literais são utilizadas para criar as soluções *s1* e *s2*, retornadas ao final.

4.3. Função de Remoção de Cubos

Remover um cubo implica que os literais deste cubo são removidos da cobertura. Sendo assim, o método de remoção de cubos escolhe de forma gulosa o cubo com maior razão entre número de literais e número de erros inseridos para ser removido a cada iteração. O Algoritmo 3 mostra o fluxo deste método.

No laço da linha 3, é realizada a escolha de qual cubo será removido. Para isso, são obtidos os EICs de cada cubo de $\mathcal{C}$ na linha 4 e o ganho de remover este cubo é calculado na linha 5. Na remoção de cubos, o EIC de um cubo é dado pelos mintermos que somente são cobertos por este cubo em $\mathcal{C}$ e que a combinação de entrada não foi adicionada como EIC anteriormente. Estes mintermos estão armazenados junto com cada cubo no primeiro mapa da estrutura de dados de $\mathcal{C}$. Caso o ganho de remover o cubo atual seja o maior encontrado até o momento, o ganho, o cubo e os EICs são armazenados. Caso o cubo não tenha EICs, o ganho é maximizado. Na linha 12, o cubo com maior ganho é removido de $\mathcal{C}$. Quando um cubo é removido, os mintermos que são cobertor por somente um cubo são atualizados, impactando nas iterações seguintes. Em seguida, o vetor de cubos removidos, número de erros acumulados nesta etapa e os EICs são atualizados.

Quando no erro máximo é atingido, o laço principal acaba. No fim, os cubos removidos são reinseridos em $\mathcal{C}$ e a solução *s3*, contendo os cubos removidos, os EICs atualizados e a redução de literais, é retornada.

5. Resultados Experimentais

Os algoritmos propostos foram implementados utilizando a linguagem de programação C++. Nossos experimentos foram executados utilizando o conjunto de benchmarks do IWLS'93 [Mcelvain 1993], em um computador com um processador quad-core i5-2400 @ 3.10GHz e 8GB de RAM. As tabelas utilizadas nesta seção estão simplificadas. As versões originais e outras análises podem ser vistas na dissertação.

Algoritmo 3: Função cubeRemoval

Entrada: Uma cobertura otimizada $\mathcal{C}$, um limite de número de erros e e a solução atual s
Saída: Uma solução com no máximo e erros

```
1  error ← e, newEIC ← s.EIC;
2  enquanto error > 0 faça
3      para each Cube in C faça
4          cubeEIC ← getCubeEIC(cube, C, newEIC);
5          gain ← litCount(cube) / max(0.01, #cubeEIC);
6          se gain > bestGain and error ≥ #cubeEIC então
7              bestGain ← gain;
8              bestCube ← cube;
9              bestEIC ← cubeEIC;
10         fim
11     fim
12     removeCubeFromCover(bestCube, C);
13     removedCubes ← removedCubes ∪ bestCube;
14     error ← error - #bestEIC;
15     newEIC ← updateEICs(newEIC, bestEIC);
16 fim
17 insertCubes(C, removedCubes);
18 s3 ← updateSolution(removedCubes, newEic, s);
19 retorna s3;
```

5.1. Comparação com o Estado-da-Arte

O método de Su [Su et al. 2020] pode ser considerado o estado-da-arte e será adotado aqui como a referência. Os circuitos utilizados são os mesmos utilizados por Su com uma restrição fixa de 16 erros. Como o código-fonte do método de Su não está publicamente disponível, nós comparamos os nossos resultados com os apresentados no artigo.

A Tabela 1 mostra a comparação dos resultados obtidos pelo método de Su e do método proposto. Coluna 1 apresenta o nome dos circuitos utilizados, assim como o número de entradas (i) e saídas (o). Coluna 2 apresenta o número de literais da cobertura original, enquanto a Coluna 3 e a Coluna 4 o número de literais do circuito aproximado por Su e pelo método proposto, respectivamente. Coluna 5 apresenta a razão entre o número de literais da nossa aproximação e da aproximação de Su. Coluna 6 apresenta o número de cubos da cobertura original, enquanto a Coluna 7 e a Coluna 8 o número de cubos do circuito aproximado por Su e pelo método proposto, respectivamente. Coluna 9 apresenta a razão entre o número de cubos da nossa aproximação e da aproximação de Su. Coluna 10 e Coluna 11 apresentam os tempos de execução da cada método.

O nosso método apresentou melhores resultados para todos os circuitos utilizados, com exceção do *b12* onde nenhum dos dois métodos foi capaz de aproximar. Os circuitos *con1*, *misex1* e *b12* não foram aproximados pelo método de Su pois não existem SCTs com dois ou menos erros. Apesar disso, os circuitos *con1* e *misex1* puderam ser aproximados pelo nosso método através da etapa de remoção de cubos. Em geral, os resultados da redução no número dos cubos é similar a redução no número de literais.

O método proposto apresenta um menor tempo de execução em todos os circuitos. Esta diferença de tempo é observada principalmente nos circuitos com mais de dez entradas, mostrando que o nosso método tem uma escalabilidade melhor.

Tabela 1. Comparação do método proposto com o método de Su [Su et al. 2020] sobre os circuitos do conjunto de benchmarks do IWLS93 considerando ER como métrica de erro.

Circuito	Literais				Cubos				Tempo (s)	
	Orig.	Su	Prop.	Prop./Su	Orig.	Su	Prop.	Prop./Su	Su	Prop.
con1 i:7;o:2	32	32	24	**0.75**	9	9	7	**0.77**	0.38	0.02
misex1 i:8;o:7	96	96	77	**0.80**	12	12	10	**0.83**	0.50	0.02
apex4 i:9;o:19	5419	5040	5024	**0.99**	436	421	418	**0.99**	109	31.9
sao2 i:10;o:4	496	231	165	**0.71**	58	29	22	**0.75**	2.48	1.47
table3 i:14;o:14	2644	2459	2347	**0.95**	175	165	159	**0.96**	513	5.47
b12 i:15;o:9	207	207	207	1.00	43	43	43	1.00	249	1.69
t481 i:16;o:1	5233	5105	4975	**0.97**	481	473	463	**0.97**	1570	3.41
table5 i:17;o:15	2501	2410	2270	**0.94**	158	154	147	**0.95**	7868	22.07

5.2. Resultados Considerando a Frequência de Erro

Utilizar um limite de erros fixo pode ser um problema pois o ER depende do número de combinações de entrada do circuito. Por exemplo, o limite de 16 erros utilizados anteriormente representam um ER de 12.5% nos circuitos de 7 entradas mas somente 0.0001% em um circuito de 17 entradas. Como o método proposto consegue atingir um boa escalabilidade temporal, é possível utilizar um número maior de erros durante a aproximação, permitindo utilizar um limite de ER em porcentagem. A Tabela 2 apresenta os resultados de aproximações considerando um limite de ER de 1%, 3% e 5%, utilizando os circuito utilizados anteriormente com mais de dez entradas. Coluna 1 apresenta o nome dos circuitos utilizados com o número de entradas e saídas. Coluna 2 apresenta o limite de ER, enquanto a Coluna 3 apresenta o número de erros (NE) utilizados na aproximação. Coluna 4 apresenta o número de literais do circuito original. Coluna 5 apresenta o número de literais do circuito aproximado pelo método proposto e a porcentagem de redução entre parênteses. Coluna 6 apresenta o número de cubos do circuito original. Coluna 7 apresenta o número de cubos do circuito aproximado pelo método proposto e a porcentagem de redução entre parênteses. Coluna 8 apresenta o tempo de execução.

O método proposto alcança uma redução de literais média de 38% com ER de 1%, 56% com ER de 3% e 64% com ER de 5%. Nos circuitos *sao2*, *table3*, *t481* e *table5*, foi possível obter perto de 90% de redução de literais com 5% de ER. Além disso, o circuito *b12* que não pôde ser aproximado com 16 erros, foi aproximado neste experimento, devido ao número maior de erros permitidos, alcançando até 26.1% e redução de literais. Mesmo nos circuitos com um número maior de erros devido ao número de entrada, o método de manteve escalável, executando em menos de 5 minutos.

6. Conclusão

Com o uso crescente de aplicações resilientes a erro, é de interesse o projeto de circuitos integrados utilize esta característica para otimizar área, desempenho e consumo energético dos circuitos. Desta forma, é importante focar no desenvolvimento de ferramentas para síntese de circuitos aproximados otimizados e que sejam escaláveis.

Neste trabalho, foi proposto um método escalável para síntese de circuitos aproximados dois-níveis considerando a métrica ER que utiliza as técnicas de inserção e

Tabela 2. Resultados do método proposto sobre os circuitos do conjunto de benchmarks do IWLS93 considerando ER como métrica de erro.

Circuitos	ER	NE	Literais		Cubos		Tempo (s)
			Orig.	Aproximado	Orig.	Aproximado	
sao2	1%	10	496	273 (0.55)	58	33 (0.56)	0.83
i: 10	3%	30		75 (0.15)		12 (0.20)	3.14
o: 4	5%	51		31 (0.06)		7 (0.12)	4.74
table3	1%	163	2644	1271 (0.48)	175	93 (0.53)	32.59
i: 14	3%	491		536 (0.20)		42 (0.24)	36.56
o: 14	5%	819		189 (0.07)		17 (0.10)	44.44
b12	1%	372	207	193 (0.93)	43	40 (0.93)	2.34
i: 15	3%	983		167 (0.80)		35 (0.81)	7.52
o: 9	5%	1638		145 (0.70)		28 (0.65)	15.71
t481	1%	655	5233	1992 (0.38)	481	212 (0.44)	4.67
i: 16	3%	1966		942 (0.18)		120 (0.24)	6.00
o: 1	5%	3276		578 (0.11)		84 (0.17)	8.85
table5	1%	1310	2501	720 (0.28)	158	55 (0.34)	152.81
i: 17	3%	3932		278 (0.11)		24 (0.15)	302.28
o: 15	5%	6553		152 (0.06)		14 (0.08)	375.89

remoção de cubos. Os resultados experimentais mostraram que o trabalho proposto supera o método estado-da-arte em qualidade de resultados e em escalabilidade. O código-fonte desenvolvido, os circuito originais utilizados e os circuitos aproximados gerados estão disponíveis no GitHub [1]. Pretende-se estender este trabalho em duas direções futuras:

- Modificar o método utilizado para gerar circuitos aproximados considerando diferentes métricas que adotem a magnitude do erro. O uso dessas métricas é interessante para a aproximação de circuitos aritméticos.
- Utilizar o trabalho proposto como um ”aproximador local”de um método ALS multinível. Ou seja, o uso de um método de ALS dois-níveis para aproximar subcircuitos de um circuito multinível pode ser relevante.

Referências

Brayton, R. K., Hachtel, G. D., McMullen, C., and Sangiovanni-Vincentelli, A. (1984). *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers.

Froehlich, S., Große, D., and Drechsler, R. (2019). One Method - All Error-Metrics: A Three-Stage Approach for Error-Metric Evaluation in Approximate Computing. In *2019 Design, Automation and Test in Europe Conference and Exhibition (DATE)*.

Mcelvain, K. (1993). Iwls93 benchmark set: Version 4.0.

Mittal, S. (2016). A survey of techniques for approximate computing. *ACM Computing Surveys*, 48(4).

Shin, D. and Gupta, S. (2010). Approximate logic synthesis for error tolerant applications. In *Design, Automation and Test in Europe Conference and Exhibition (DATE)*.

Su, S., Zou, C., Kong, W., Han, J., and Qian, W. (2020). A novel heuristic search method for two-level approximate logic synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(3):654–669.

[1] `https://github.com/GabrielAmmes/2LALS-IR`