Evaluation of Clause-Column Table method to design of MIC hazard-free in asynchronous finite state machines

Tiago da Silva Almeida[1] e André Luiz Gomes de Freitas[1]

[1] *Universidade Federal do Tocantins, Curso de Ciência da Computação, Tocantins, Brasil*

Data de recebimento do manuscrito: 18/05/2020
Data de aceitação do manuscrito: 08/06/2020
Data de publicação: 12/06/2020

Abstract— Asynchronous finite state machines are of great interest because they use fewer transistors to manufacture them. However, find the minimum resources in logic modeling is an NP-Problem, since the search space increase exponentially. Although this problem is studied for some time, there is still space for newer researches. Mostly, in new computational methods to improve performance in solving logical optimization in finite state machines. Thus, this paper presents a study and evaluation of heuristic algorithms for the optimization of asynchronous finite state machines, to obtain the smallest possible circuit. Thus, the Clause-Column Table and Quine-McCluskey algorithms are combined in order to propose an algorithm capable of minimizing asynchronous sequential circuits. Tests and results show that it is possible to synthesize circuits in a reasonable time, but with some logical errors. It may be concluded that it still needs research, even though it is not such a recent line of research.

Keywords—Assincronous Finite State Machine, Clause-Column Table, Heuristic, Quine-McCluskey

I. Introdução

Com o avanço da tecnologia, o uso de sistemas digitais tem sido amplamente difundido, principalmente dos circuitos eletrônicos digitais. Como resultado desse avanço é necessário lidar com circuitos digitais cada vez mais complexos. Consequentemente ainda é necessário que os custos e o tempo de implementação sejam cada vez menores [1]. Além disso, os softwares mais robustos, como os que são baseados em aprendizagem profunda ou que processam grandes quantidades de dados, necessitam de hardwares cada vez mais robustos, às vezes até hardwares especializados como os ASICs (*Application Specific Integrated Circuits*), para que possam obter um desempenho melhor. Por causa disso, a área de otimização é cada vez mais utilizada para obter circuitos menores e mais eficientes sem a necessidade de utilizar componentes menores, ou seja, diminuir a escala de integração.

A área de minimização de lógica não é nova, mas é cada vez mais necessária uma lógica eficiente e de baixo custo em diversas áreas, como por exemplo, na minimização de circuitos e sistemas VLSI (*Very Large Scale Integration*), ULSI (*Ultra Large Scale Integration*), além de outros [2].

Na maioria desses sistemas são necessários circuitos que sejam capazes de salvar dados e realizar operações aritméticas, lógicas sobre esses dados, ou mesmo algum controle especializado dentro do sistema. Esse tipo de circuito é chamado de circuito sequenciais, porque a sequência de operações define o comportamento do sistema. As Máquinas de Estados Finitos (MEF) é um modelo abstrato que descreve o comportamento dos circuitos sequenciais [3].

Com as MEFs podemos abstrair determinados circuitos, e pensar em seu comportamento como um algoritmo. Onde o algoritmo pode ser representado como uma máquina, ou grafo, que possui vários estados, e entre cada estado possui transições que ocorrem de acordo com as entradas recebidas. A partir dessa representação é possível chegar ao modelo físico do circuito.

O modelo físico obtido através de uma MEF pode ser internamente ou externamente síncronos em relação ao sistema ou assíncrono. O modelo abstrato que descreve o circuito assíncrono também é chamado de Máquina de Estado Finito Assíncrona (MEFA), ou também chamada como Máquina de Estados Finitos em Modo Fundamental.

Os circuitos assíncronos são uma alternativa para resolver alguns dos problemas que os sistemas síncronos complexos possuem, que são geralmente relacionados ao sinal do relógio (*clock*) global, entre outros. Além de possuir outras vantagens em relação aos síncronos, como sem distorções no relógio, sem distribuição de relógio, menor consumo de energia, maior modularização e mais robustez a variações de temperatura e menor interação com efeitos eletromagnéticos [4].

Como normalmente todo hardware, seja ele VLSI ou ASIC ou um processador convencional, possui partes ou são inteiramente feitos utilizando MEFA, obtê-los com circuitos

Dados de contato: Tiago da Silva Almeida, tiagoalmeida@uft.edu.br

de tamanho ótimo ou o menor possível pode levar a estruturas mais eficientes e mais baratas. Porém, minimização de MEF ou MEFA é complexo e possui um alto custo computacional, por causa disso o estudo de novos métodos para realização dessa tarefa está cada vez mais presente, para obter ferramentas com um custo computacional menor e que geram resultados satisfatório.

Uma MEFA pode ser expressa somente por funções booleanas, pois, é composta apenas por componentes com estruturas simples como as portas lógicas. O que torna possível aperfeiçoar tanto o espaço, utilizando minimização lógica combinatória, como também o custo, visto que os componentes lógicos possuem um custo inferior ao de estruturas mais complexas utilizadas em MEF (como elementos de memória).

Na literatura existem algoritmos de minimização de funções booleanas considerados exatos, que geram implicantes primos para então obter uma solução ótima [1]. Com base nisso, esse projeto tem como objetivo desenvolver um algoritmo baseado no método *Clause-Column Table* e no método Quine-McCluskey de forma a utiliza-los para otimizar uma MEFA.

O número de variáveis e termos de uma expressão booleana estão relacionados com o tamanho do circuito combinacional da MEFA, o que está relacionado com custo e o desempenho. Assim, objetivo geral deste trabalho é apresentar um algoritmo baseado em minimização de funções booleanas de *Clause-Column Table* e no algoritmo de Quine-McCluskey, com o propósito de realizar sínteses das MEFA e realizar uma avaliação comparativa dos resultados produzidos com os apresentados na literatura.

II. CIRCUITOS SEQUENCIAIS ASSÍNCRONOS

A MEF em modo fundamental, também chamada de circuito sequencial assíncrono ou MEF Assíncrona (MEFA), utiliza-se da ideia que os *flip-flops* apenas proporciona um atraso entre as variações nos níveis lógicos por um período de relógio. Portanto, na MEFA os *flip-flops* foram substituídos por elementos que introduzem um atraso. O tipo de atraso proporcionado é o atraso decorrente da propagação do sinal elétrico transmitido através de um fio, ou uma cascata de portas lógicas [5].

Existe diversas formas de representar uma MEFA. O modelo da Figura 1 é o mais simples para entender como o circuito funciona. Também chamado de máquina de Huffman, o circuito é caracterizado pelo fato que as entradas podem mudar a qualquer momento e pelo uso de elementos de atraso como dispositivos de memória [3].

A combinação dos sinais de entrada $x_1, x_2, ..., x_l$ são chamados de estado de entrada. Na saída dos elementos de atraso D (*delay*) existem as variáveis $y_1, y_2, ..., y_k$, que são chamadas de variáveis internas ou secundárias. Da mesma forma o conjunto dessas variáveis é chamado de estado secundário ou interno, e as variáveis $Y_1, Y_2, ..., Y_k$ são chamadas de variáveis de excitação. As saídas geradas pela lógica combinacional determina as variáveis de saída como também o estado secundário, que o sistema irá assumir em seguida. Um estado de entrada é dito estar em um estado estável, se e somente se $y_i = Y_i$ para todo $i = 1, 2, ..., k$ [3].

Quando o estado de entrada muda, ou seja, alguma de

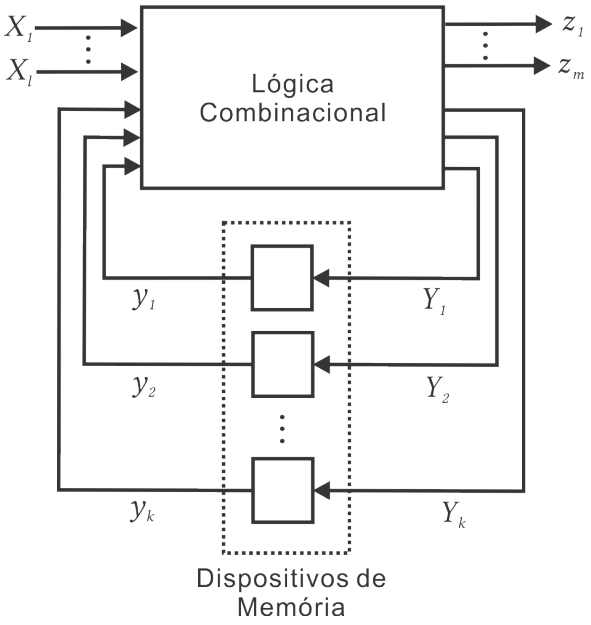

Figura 1: Modelo básico para uma MEF em modo Fundamental [3].

suas variáveis são alteradas, a lógica combinacional produz um novo conjunto de valores para as variáveis de excitação, como consequência o circuito entra no que é chamado de estado instável. Após um instante de tempo de atraso as variáveis secundárias assumirão o novo valor das variáveis de excitação, como resultado o circuito vai para o próximo estado estável [5]. Portanto, uma transição de um estado estável para outro só irá ocorrer quando o estado de entrada sofre alguma alteração.

A ideia é que, depois que um valor de entrada muda, não ocorra nenhuma outra mudança nas entradas enquanto o circuito não entra no estado estável. Tal forma de operação é chamada de modo fundamental. Se somente uma variável de entrada puder mudar, é chamado de modo fundamental SIC (*Single-Input-Change*), no entanto se mais de uma entrada puder mudar, é chamado de circuito em modo fundamental MIC (*Mutiple-Input-Change*). Uma generalização da MIC é chamada de circuito *Burst-Mode* [3].

Assim como no caso da MEF, existe algumas maneiras convenientes de demonstrar o comportamento desejado de um circuito. Para as MEFA, o método para descrever o comportamento é chamado de tabela de fluxo. A Tabela 1 é um exemplo de tabela de fluxo de um circuito que conta em módulo 4 (de 0 a 3). A Tabela 1 apresenta o número de inversões de nível lógico sobre a entrada X. Logo, quando X passar por $X = 0 \rightarrow 1 \rightarrow 0 \rightarrow 1 \rightarrow 0$ etc.. a saída esperada será $z_1 z_0 = 00 \rightarrow 01 \rightarrow 10 \rightarrow 11 \rightarrow 00$ assim por diante [5].

Logo, a partir da Tabela **??** nota-se que possui 4 estados estáveis, que são representados pelos estados em negrito. Os demais estado são instáveis. Portanto, quando $Xmudapara1$ e o circuito está no estado 00, ou seja, $y_1 y_0 = 00$, $Y_1 Y_0$ mudara para 00 , neste momento $y_1 y_0 \neq Y_1 Y_0$. Portanto, a MEFA irá para um estado instável, mas após o atraso $y_1 y_0 = Y_1 Y_0 = 01$, levando o circuito para o próximo estado estável. A saída não está representada, pois nesta atribuição $z_1 z_0 = Y_1 Y_0$. De forma semelhante a MEF, pode se obter as equações de excitação a partir da Tabela 1 e construir o circuito lógico.

TABELA 1: TABELA DE FLUXO DE ESTADOS.

Estado,Saida		
	X	
y_1y_0	0	1
00	**00**	01
01	10	**01**
10	**10**	11
11	00	**11**

1. Circuito Bust-Mode

As máquinas MIC possuem uma maior flexibilidade, pois permite que diversas entradas mudem de uma vez antes que a máquina entre no estado estável. Uma generalização é chamado de circuito de *Bust-Mode* (BM). o circuito BM permite que múltiplas entradas mudem simultaneamente, mas diferente da MIC que todas as entradas devem chegar em um período de tempo limitado, pode chegar em qualquer ordem e a qualquer momento no chamado *input-bust* [6].

Ela é representada por meio de um diagrama de estado chamado de especificação *bust-mode*. O diagrama possui um número finito de estados, cada transição é representada por um arco conectando a um par de estados e possui um estado inicial distinguível. O arco é rotulado com possíveis transições, levando o sistema de um estado para outro. Cada transição consiste de um conjunto não vazio de entradas *input-bust* e um de saída *output-bust*, se nenhuma entrada muda então o sistema está estável. Em um dado estado quando todas as entradas mudaram de valor, a máquina gera o conjunto de saída *output-bust* correspondente e então muda de estado. O conjunto de entrada *input bust* muda somente uma vez por transição de estado [7, 4].

O circuito em BM possui duas restrições na sua especificação. Primeiro, nenhum conjunto de entrada de um determinado estado pode ser subconjunto de outro ou então o comportamento será ambíguo. Esta restrição é chamada de propriedade do conjunto máximo. Segundo, em um dado estado sempre entrará com o mesmo conjunto de entrada, ou seja, cada estado possui apenas um ponto de entrada. Chamada de propriedade do ponto de entrada único. No caso da propriedade do ponto de entrada único, se um diagrama não a satisfaz, pode ser transformado em um diagrama equivalente que o satisfaz dividindo os estados [7]. Um exemplo da MEFA em BM pode ser visto na Figura 2, onde cada transição é rotulada com um conjunto de entrada e um de saída divido por um $/$. Uma transição de ascendente $0->1$ é representada por "$+$" e uma transição de descendente $1->0$ por "$-$".

2. Corridas

Qualquer transição alternativa realizada do estado de origem ao estado de destino, que envolva mudança de duas ou mais variáveis é chamada de corrida. Por exemplo se quiser ir do estado 00 para o 11, terá que passar pelo 01 ou 10. Esse evento acontece porque as variações nos $Y's$ ocorrem através de tempos de atraso aleatórios (devido a natureza física dos componentes) e acabam chegando nos $y's$ em tempos diferentes. Pois, os circuitos para cada um possuem tamanhos diferentes e quantidade de portas lógicas distintas. Logo, o tempo necessário para cada uma das variáveis percorrer o seu circuito é diferente.

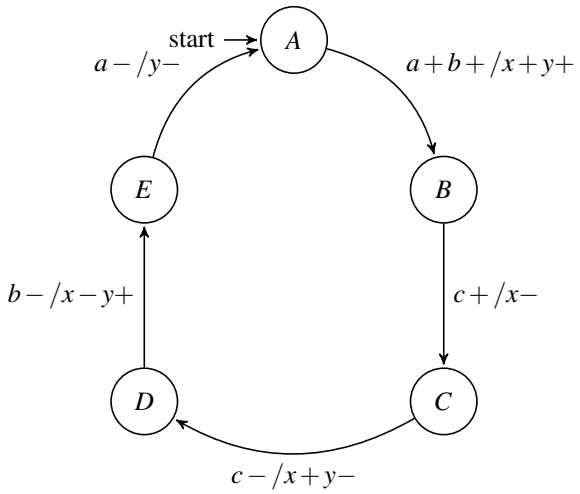

Figura 2: Um exemplo de circuito *Bust-Mode* [7].

Em alguns casos essas corridas são permitidas, pois pode fazer parte ou ser essencial para a lógica da MEFA projetada. Uma corrida que possa resultar em uma transição que se estabilize em um estado errado é chamado de corrida crítica. Esse defeito deve ser corrigido ou eliminado, visto que não pode ser permitido tal defeito no circuito [5, 8].

3. Hazard de Funções

A presença de *hazard* coloca um desafio a mais na hora de desenvolver circuitos sequencias assíncronos. Os *hazard* são as falhas (*glitches* ou *spikes*) que podem se manifestar na saída por algum instante de tempo. Uma função f que não muda monotonicamente durante uma transição de entradas é dito ter um *hazard* de função na transição. As seguinte definições são de [9, 7].

Definição II.1. Uma função booleana f contém um *hazard* de função estático para a transição de entrada de A para C, se e somente se:

1. $f(A) = f(C)$, e;
2. Existe algum estado de entrada $B \in [A,C]$ tal que $f(A) \neq f(B)$.

Definição II.2. Uma função booleana f contém um *hazard* de função dinâmico para a transição de entrada A para D, se e somente se:

1. $f(A) \neq f(D)$.
2. Existe um par de estado de entrada B e C, onde $A \neq B$ e $C \neq D$ tal que
   (a) $B \in [A,D]$ e $C \in [B,D]$ e;
   (b) $f(B) = f(D)$ e $f(A) = f(C)$.

Se uma transição possui um *hazard* de função, nenhuma implementação da função é garantida ser livre de *glitches* durante uma transição, assumindo um atraso arbitrário nos fios e portas logicas.

4. Hazards Lógicos

Se f é uma função é livre de *hazard* para uma transição do estado de entrada A para B, ele ainda pode conter *hazards* por causa do atraso na lógica realizada. As seguintes definições foram tiradas do [7].

Definição II.3. Um circuito combinacional para uma função f contém um *hazard* lógico estático para a transição de entradas de um mintermo A para o mintermo B se e somente se:

1. $f(A) = f(B)$;
2. Não existe nenhum *hazard* de função estático na transição de A para B;
3. Para alguma alocação de atraso, a saída do circuito não é monotônica durante o intervalo de transição.

Definição II.4. Um circuito combinacional para uma função f contém *hazard* lógico dinâmico para a transição de estado de entrada A para o B, somente se:

1. $f(A) \neq f(B)$;
2. Não existe *hazard* dinâmico de função na transição de A para B.
3. Para uma certa alocação de atraso, a saída do circuito não é monotônica durante o intervalo de transição.

As definições formalizam que o *hazard* lógico ocorre para alguma combinação particular de portas e fios de atraso no circuito, devido aos fatos das alterações dos estados de entradas não serem instantâneo. Portanto, ocorrerá um *glich* ou *spike* na saída durante a transição.

III. TRABALHOS RELACIONADOS

Um dos principais problemas da minimização é achar um algoritmo adequado e eficiente. Para resolver esse problema o [2] utiliza de uma abordagem em duas fases distintas. Na primeira fase todos os implicantes primos são gerados, enquanto na segunda gera-se um subconjunto de implicantes primos do conjunto original onde a sua união forma a função booleana minimizada. Sendo o algoritmo proposto o *Clause-Column Table* que é computacionalmente muito eficiente em funções dadas na forma soma de produto, produto da soma, na sua forma canônica ou não.

Outra abordagem ao mesmo algoritmo relevante para esse projeto foi a de [1] onde foi proposto uma melhoria no *Clause-Column Table*, adicionando o teorema da adjacência e um novo critério de parada. Além disso, também realizou a segunda fase por meio de programação linear inteira. Os resultados obtidos mostraram que o método *Clause-Column Table* aprimorado pode ser mais eficiente que o método original de [2].

Outro método proposto por [10] é o de minimização de expressões booleanas na forma de soma de produtos de ou exclusivo (XOR). É uma área bastante explorada na literatura por causa de suas propriedades que facilitam o processo de testes e o circuito final nessa forma é consideravelmente menor que na soma de produtos. Porém, ainda não foi encontrado uma solução eficiente para minimizar funções como mais de 6 variáveis nos casos gerais.

Seguindo essa mesma abordagem, [11] propuseram uma heurística para minimização de expressões booleanas na forma soma de produtos de ou exclusivo, para funções incompletamente especificadas de múltiplas saídas. Pois, utilizando essa abordagem reduz consideravelmente a complexidade do circuito.

[10] desenvolveram um algoritmo baseado no DCMIN (*Don't Care MINimization*) que usa decomposição funcional e lógica de múltiplos valores para produzir expressões quase mínimas para essas funções chamadas QuickDCMIN, que demonstrou ser mais eficiente.

Já no trabalho de [12] foi proposto um método de aprendizagem de máquina para minimização de funções booleanas chamada de MLBM (*Machine-Learning-Based Minimization*). Porém, o MLBM é diferente dos métodos de minimização comuns na literatura, como já apresentado. A abordagem possui uma grande habilidade de processamento, já que não possui limite de variável, e sempre produz um resultado excelente.

Diferentemente dos outros trabalhos o [4] propôs dois novos algoritmos baseados no algoritmo genético, para minimização e alocação de estado de uma MEFA. A arquitetura da MEFA utilizada foi da máquina no modo *Bust-mode*. Tal arquitetura não possui realimentação por parte do circuito de saída. O que possibilita um baixo tempo de resposta, além da capacidade de alteração de mais de uma variável de entrada no mesmo instante. O algoritmo apresentou uma alta eficiência em seus testes quando comparado com a ferramenta *Minimalist* que é o estado da arte para tal arquitetura.

[13] desenvolveram um algoritmo atrelado a um sistema CAD (*Computer-Aided Design*) para lidar com máquina de estados em nível lógico. O sistema foi desenvolvido para analisar a MEFA na linguagem de descrição de hardware VHDL. Assim, como o algoritmo realiza a minimização e retorna um modelo minimizado em VHDL. O sistema foi modelado tanto para computação com um único processador, quanto para computação de alto desempenho.

A eficiência da ferramenta CAD foi avaliada utilizando diferentes números de nós computacionais e MEFAs. Com base nisso, o aumento de eficiência com base no aumento de nós é diretamente ligado a complexidade da MEFA. Quanto mais complexa, mais significativa era o ganho de eficiência com o aumento de nós. Da mesma forma, quanto menos complexa menos significativa era o aumento.

IV. METODOLOGIA

Como as MEFAs são compostas basicamente de lógica combinacional com realimentação, pode-se utilizar otimização lógica para minimizar tais máquinas. O processo de minimização de uma MEFA é uma síntese que possui três etapas, sendo a minimização lógica a última delas. Como as duas primeiras etapas, minimização de estados e alocação de estados estão fora do escopo desse trabalho, foram realizadas utilizando a ferramenta *minimilist*, uma ferramenta que possui uma coletânea de algoritmos, de todas as três etapas, ela possui o objetivo de auxiliar no desenvolvimento de novos algoritmos e síntese de MEFAs [6].

Nesse projeto é proposto um algoritmo para minimização lógica de MEFA baseados nos métodos *Clause-Column Table* aprimorado desenvolvido por [1] e no algoritmo Quine-

McCluskey [14].

A escolha do *Clause-Column Table* foi feita, pois ele é eficiente mesmo com um número grande de variáveis, além de aceitar expressões booleanas tanto na forma soma de produtos e produto de somas na forma canônica e não canônica.

a. Clause-Column Table

O método *Clause-Column Table* é uma técnica tabular e iterativa que permite a geração de implicantes primos. O resultado converge rapidamente mesmo no caso de funções com muitos termos e variáveis [2]. Apesar disso o método possui algumas deficiências.

[1] desenvolveram um nova versão nomeada de *Clause-Column Table* aprimorado, em que foram feitos algumas alterações para evitar a geração de termos nulos e diminuir a quantidade de iterações utilizada para geração dos implicantes primos.

Foi adicionado, a propriedade algébrica $AB + A\overline{B} = A$. Quando aplicado a tabela inicial formada pelos maxtermos da função booleana muitos termos são eliminados. Outra alteração foi a adição de um critério de parada com o objetivo de evitar a geração de termos nulos. Se houver apenas uma coluna a partir da segunda iteração o algoritmo deve ser interrompido. O algoritmo *Clause-Column Table* aprimorado desenvolvido pela [1] é composto pelos seguintes passos:

1. Passo: Forme uma tabela em que cada coluna é composta por um maxtermo da função. Se a função booleana inicial estiver na forma de produto de soma, execute o próximo passo, se não obtenha o seu dual.
2. Passo: Execute as seguintes operações até que todas as colunas sejam excluídas, caso não for possível excluir todas vá para o passo 3:
   (a) Se algum critério do passo 6 for atendido pare a execução, senão continue;
   (b) Se houver colunas dominadas, elimine-as;
   (c) Se existir colunas redundantes elimine aleatoriamente uma delas;
   (d) Simplifique as colunas restantes com o teorema da adjacência (apenas uma vez por iteração);
   (e) Se houver literal essencial selecione-os. Se houver mais de um literal essencial selecione um deles aleatoriamente;
   (f) Verifica novamente o critério de para 6(c);
   (g) Elimine todas as colunas que contém literais essenciais;
   (h) Elimine o complemento do literal selecionado das colunas restantes;
3. Passo: Se algum literal foi selecionado no passo anterior execute o passo 5, senão execute o passo 4.
4. Passo: Verifique se na tabela reduzida a quantidade de incidência de cada um dos literais na forma complementada ou não:
   (a) Se houver literais com a mesma incidência selecione um deles aleatoriamente;
   (b) Se o literal com a maior incidência estiver contido em todas as colunas ele deve ser considerado implicante primo da iteração atual;
   (c) Execute o passo 6.
5. Passo:
   (a) Todas as colunas que contêm o literal selecionado devem ser eliminadas;
   (b) O complemento do literal selecionado também deve ser eliminado das demais colunas;
   (c) Forme a tabela reduzida com o restante das colunas;
   (d) Se na tabela reduzida tiver mais de uma coluna selecione o literal essencial, senão repita o passo 5(a) e 5(b).
   (e) Realize a operação lógica AND entre o literal selecionado e a tabela reduzida cujo o termo resultantes são implicantes primos.
6. Passo: Verifique os critérios de parada.
   (a) Todas as colunas sejam excluídas;
   (b) Dois literais, complementados ou não se tornem essenciais ao mesmo tempo;
   (c) Um literal torna-se essencial e na iteração seguinte o seu complemento também;
   (d) Reste apenas uma coluna a partir da segunda iteração.
7. Passo: Quando atingido algum critério de parada reúna os implicantes primos resultantes e forme o conjunto de solução.

O Algoritmo deve então ser executado iterativamente até que um dos critérios de parada seja atingido. Com os implicantes primos obtidos pode-se então usar de outros algoritmos para gerar a expressão mínima da função booleana.

b. Quine-McCluskey

Usualmente a abordagem para o problema de minimização de expressões booleanas envolve duas fases distintas. Na primeira fase, o algoritmo encontra todas os implicantes primos. Já na segunda fase, a partir do conjunto de implicantes primos obtidos na primeira fase, um subconjunto mínimo é selecionado de uma forma que a sua união é equivalente a função original [2]. O Quine-McCluskey segue essa mesma lógica, possuindo duas fases: na primeira ele encontra os implicantes primos, na segunda ,ele gera o conjunto mínimo para minimizar a função.

O método começa com uma lista de todos os n mintermos da função e continuamente deriva todos os implicantes primos com $n-1$, $n-2$ variáveis até que todos os implicantes tenham sido identificados. O algoritmo possui 4 passos, sendo o 1° e 2° para encontrar os implicantes primos e o 3° e 4° para realizar a simplificação.

Na primeira etapa todos os mintermos da função são listados em uma coluna com sua representação binária. Em seguida, são separados em grupos de acordo com o número de bits 1 em sua forma binária, pois isso simplifica a procura

de mintermos logicamente adjacentes. Já que para ser logicamente adjacentes os minitermos se diferenciam em apenas um literal, logo na forma binária ele tem que ter um bit 1 a mais ou um a menos.

No segundo passo é realizado uma busca por força bruta entre os grupos vizinhos procurando por mintermos adjacentes e combinando todos eles em uma coluna de $n-1$ implicantes primos, marcando todos os que foram possíveis de combinar. Após isso repita para todas as colunas, combinando as colunas de $n-1$, $n-2$ até quando não for possível combinar mais nenhum implicante, marcando todos eles. Após isso todos os implicantes não marcados são primos, tornando uma lista de implicantes primos.

Obtido a lista de implicantes primos, o passo 3 constrói se uma tabela que lista todos os mintermos na horizontal e implicantes primos na vertical, marque onde um certo implicante primo (linha) for capaz de cobrir um mintermos (coluna). No último passo selecione o menor número de implicantes que é capaz de representar todos os mintermos da função [9].

c. Síntese de MEFA

Assim como as abordagens usuais, que divide a síntese de circuitos lógicos combinacionais em algumas etapas [2]. O método proposto tem como objetivo tentar melhorar o desempenho do processo de síntese utilizando de dois algoritmos conhecidos na literatura (CCT/QM –*Clause-Column Table* e Quine-McCluskey). Tal método possui três etapas como pode ser visto na Figura 3, uma etapa para realizar a minimização de estados e alocação dos mesmos. A segunda etapa utilizada para obter os implicantes primos de cada equação de excitação, pôr fim a última etapa selecionar o menor subconjunto que consiga representar a função lógica de cada equação, desta maneira fornecendo o circuito lógico da MEFA.

O circuito foi operado em modo fundamental MIC, mais especificamente na sua generalização BM (*Bust-mode*), por ter uma grande quantidade de estudos e ferramentas, além de ser comercialmente mais utilizado. O algoritmo recebe como entrada uma tabela de estados, o número de variáveis de entrada, número de variáveis de saída, número de estados usados e número de transições entre os estados. As etapas demostradas na Figura 3 funcionam da seguinte maneira:

- Etapa 1: Na primeira etapa recebe um arquivo com todas as especificações da MEFA, foi feito então a minimização de estados e alocação dos mesmos utilizando a ferramenta chamada *chasm* presente na *minimilist* [6]. O *chasm* realiza alocação de estado livre de corrida crítica próximo do ótimo. Após a alocação é construída a tabela de transição de estados final e dela gerado o conjunto de equações de excitação que representam o circuito da MEFA;
- Etapa 2: A partir do conjunto de equações de excitação obtido é encontrado os implicantes primos de cada equações, utilizando o método do *Clause-Column Table*, obtendo então o conjunto de implicantes primos;
- Etapa 3: Com o conjunto de implicantes primos, utiliza-se como base os passos 3 e 4 do algoritmo Quine-McCluskey para gerar expressão booleana mínima de cada equação de excitação da MEFA, assim retornando o circuito combinacional na forma de soma de produtos.

d. Benchmark

O algoritmo foi avaliado utilizando o *benchmark* da Tabela 2. O *benchmark* é um conjunto com 10 MEFAs amplamente utilizados em estudos na utilizado na literatura [4, 15, 6], sendo algumas das MEFAS especificadas são modelos reais utilizados na indústria.

TABELA 2: DESCRIÇÃO DOS CASOS CONTIDOS NO *benchmark*.

Benchmark	Entradas	Saidas	Estados
dme-e	3	3	8
it-control	5	7	10
hp-if-rf-control	6	5	12
stetson-p2	8	12	25
stetson-p3	4	2	8
isend	4	3	10
scsi-tsend-bm	5	4	11
opt-token	4	4	12
hp-ir	3	2	6
isend-bm	5	4	10

O algoritmo utilizado nesse projeto foi incorporado no processo de síntese de MEFA na sua última etapa, para geração do circuito combinacional reduzido. Para as partes de minimização de estado e alocação dos mesmos foi utilizado a ferramenta *minimalist*, já que essas etapas fogem do escopo do trabalho.

V. RESULTADOS

As configurações utilizadas para minimização de estado e alocação dos mesmo realizado com o *minimalista* de forma heurística para as MEFAs não garantem possuir uma alocação ótima, apenas ser livre de corrida crítica. Para diminuir as variáveis de estado e conseguintemente gerar um circuito equivalente reduzido, foi utilizada uma variação da máquina de Huffman, onde só as variáveis de estados são realimentadas (modelo de Moore), como algumas das variáveis de saída realimenta o circuito como entrada de estado. Após a alocação é gerado um arquivo no formato de PLA (*Programmable Logic Array*), onde informa uma tabela verdade com os mintermos de onde pode ser obtida as funções de excitação do circuito, e assim realizar a minimização lógica.

De forma a avaliar o desempenho do método foi utilizado o *benchmark* com 10 máquinas com o circuito em modo BM. Sendo alguns projetos reais de circuitos assíncronos e por ser bem utilizado na literatura. Para mostrar a eficiência do algoritmo foi feito uma comparação com os resultados de [4] que propõe a ferramenta SAGAAs. A minimização e alocação de estados usando Algoritmos Genéticos, a minimização lógica foi realizada com o HFMIN que é um algoritmo exato para quantidades de literais [6].

Todos os *benchmark* foram executados em um processador AMD A12-9720p Quad-Core com até 3,60 GHz e 8GB de SDRAM DDR4. Quanto aos resultados obtidos do [4] não foi informado as especificações onde os *benchmark* foram executados. A comparação dos dois resultados pode ser vista na Tabela 3.

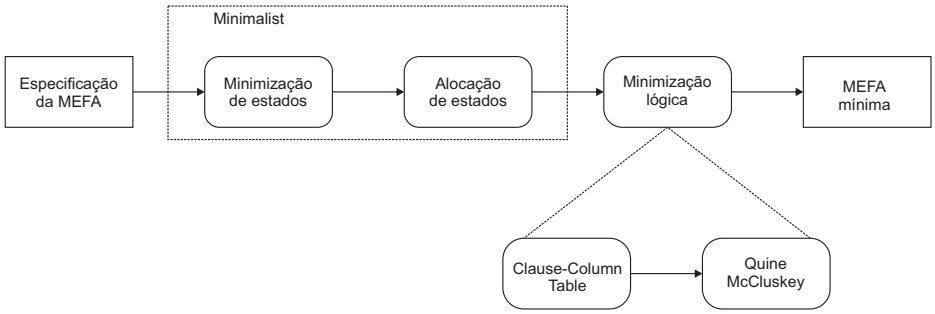

Figura 3: Fluxo de minimização de uma MEFA proposto neste trabalho.

As especificações E, S e EST da MEFA contida na Tabela 3 são respectivamente a quantidade de entrada, saída e estados, os resultados Pr, Li e T também são respectivamente, o número de produtos, de literais das funções finais do circuito é o tempo de execução do algoritmo em relação ao tempo de CPU. O tempo de execução do SAAGA's é o tempo de toda a síntese (minimização de estado, alocação de estado e minimização lógica) diferente do CCL/QM a onde o tempo do *minimalist* não foi considerado, apenas o do método proposto por esse projeto.

O CCT/QM em relação ao SAAGAS/HFMIN não obteve um desempenho tão bom, houve um aumento de $14,6\%$ no número de produtos gerado na solução final como pode se ver na Tabela 3 em que o total de produtos foram de 178 para o CCT/QM contra 152 do SAAGAs//HFMIN e um aumento de $12,6\%$ no número de literais também, com 650 contra 568 literais gerado no *benchmark*. Sendo as maiores diferenças no número de produtos e literais são respectivamente na máquina *steson-p2* com uma diferença de 10 produtos e na *scsi-tsend-bm*.

Em relação ao tempo de execução, incluindo o *benchmark steson-p2* houve uma diminuição no tempo de execução em $91,7\%$ e se não consideramos o mesmo será de 64% de redução. Porém, o tempo demostrado pelo SAAGAs é o tempo execução das três etapas do processo de síntese, em quanto o do CCT/QM é tempo somente da minimização lógica. No geral os tempos foram semelhantes, em relação as MEFs menores, já com as MEFs com mais variáveis de entradas e saída ambas houve um salto no tempo de execução, sendo CCT/QM com um tempo inferior ao do SAAGAS/HFMIN, no entanto este gerou um circuito final menor.

O desempenho do método em relação ao trabalho do [4], o circuito final gerado foi inferior, apesar de ter um tempo parcial melhor. Além disso o *Clause-Column table* não garante que o circuito seja livre de *hazards* de função de lógica, ou seja, pode ocorrer *glitches* na MEFA final, enquanto o HFMIN utilizado no SAGAAS garante ser livre de *hazards* e é exato no número de literais.

VI. Considerações finais

O desenvolvimento deste projeto possibilitou a análise dos tipos de MEFs e suas aplicações. Entre elas a MEFA apresentou algumas vantagens econômicas e energéticas e em relação à MEF. Além disso, como a MEFA é composta de apenas de circuitos combinacionais possibilita utilização de algoritmos de otimização lógicas para gerar um circuito equivalente mínimo.

Durante o desenvolvimento do projeto as MEFAs foram modeladas em modo fundamental MIC, mais especificamente o circuito BM, pois está mais próximo dos circuitos reais, utilizados na indústria. Além de que possui diversos estudos dessa modelagem, assim como ferramentas que ajudam no processo de síntese. Logo, trouxe o projeto para problemas mais próximos dos problemas do mundo real.

Neste projeto foi mostrado um método de minimização lógica de MEFA utilizando os algoritmos *Clause-Column Table* e o Quine-McCluskey, na tentativa de obter um minimizador lógico eficiente. No entanto, os resultados demostraram que o método não é tão eficiente já que houve um aumento no número de literais e produtos do circuito final. Contudo, a diferença de soluções não é tão distante se comparados empiricamente. Embora, seja necessária uma análise estatística para comprovar essa hipótese.

Apesar do método garantir que o circuito seja livre de corrida crítica, uma deficiência é o fato que o *Clause-Column table* não garante que o circuito final seja livre de *hazards*. Diferentes de outros métodos já existentes na literatura, podendo a MEFA final possuir alguns *glitches* na saída, até que a MEFA entre em estado estável. Essa limitação é devido ao fato de ele só suportar funções completamente especificadas, ou seja, em que todas as entradas possuem um comportamento esperado e previsível.

Em relação ao tempo de execução, o desempenho foi satisfatório nos *benchmarks*. Porém, necessita de mais estudos e experimentos em MEFAs mais robustas, onde possuem um número maior de entradas e saídas para que tenha um resultado mais preciso. Pois, um aumento de 2 variáveis de entrada fez o tempo de execução que era de menos de um segundo saltar para mais de 6 segundos. Devido ao fato de o problema de gerar um conjunto de implicantes primos mínimo que represente todos os mintermos da função ser um problema NP-difícil e o Quine-Mccluskey ter complexidade exponencial, quanto mais variáveis a função tiver mais rapidamente o tempo execução aumentará.

Para futuros trabalhos, visando melhorar o método, é conveniente avaliar as meta-heurísticas para gerar o conjunto mínimo de implicantes primos da função para MEFAs maiores. Outro é adaptar o *Clause-Colum Table* para que possa garantir que o circuito final seja livre de *hazard* de função e lógico, utilizando expressões booleanas na forma de notação cubica.

Referências

[1] C. D. P. do Nascimento Barbieri, J. V. De Paulo, and A. C. Rodrigues, "Aperfeiçoamento do método clause-column table para a geração eficiente de implicantes primos," *A*, vol. 1, p. M2, 2015.

TABELA 3: RESULTADOS DA METODOLOGIA PROPOSTA.

Benchmark	Especificações			CCT/QM			SAAGAs/HFMIN		
	E	S	EST	Pr	Li	T(s)	Pr	Li	T(s)
dme-e	3	3	8	8	30	0,11	8	24	0,19
it-control	5	7	10	19	68	0,22	19	60	0,66
hp-if-rf-control	6	5	12	13	46	0,26	7	63	0,82
stetson-p2	8	12	25	42	160	6,3	32	154	88,60
stetson-p3	4	2	8	5	18	0,13	5	13	0,21
isend	4	3	10	21	81	0,22	19	59	0,27
scsi-tsend-bm	5	4	11	28	100	0,20	24	72	0,55
opt-token	4	4	12	15	48	0,25	14	50	0,68
hp-ir	3	2	6	3	11	0,07	4	12	0,05
isend-bm	5	4	10	24	88	0,2	20	61	0,23
Total				178	650	7,5	152	568	92,26

[2] S. R. Das, A.-A. Amin, S. N. Biswas, M. H. Assaf, E. M. Petriu, and V. Groza, "An algorithm for generating prime implicants," in *Instrumentation and Measurement Technology Conference Proceedings (I2MTC), 2016 IEEE International*. IEEE, 2016, pp. 1–6.

[3] Z. Kohavi and N. K. Jha, *Switching and finite automata theory*. Cambridge University Press, 2009.

[4] T. Curtinhas, T. C. Cavalcante, D. L. Oliveira, L. A. Faria, and O. Saotome, "Minimization and encoding of high performance asynchronous state machines based on genetic algorithm," in *2015 28th Symposium on Integrated Circuits and Systems Design (SBCCI)*. IEEE, 2015, pp. 1–6.

[5] H. Taub, *Circuitos Digitais e Microprocessadores*. Editora McGraw-Hill, 1984.

[6] R. M. Fuhrer and S. M. Nowick, *Sequential optimization of asynchronous and synchronous finite-state machines: Algorithms and tools*. Springer Science & Business Media, 2012.

[7] S. M. Nowick, "Automatic synthesis of burst-mode asynchronous controllers," Ph.D. dissertation, Stanford University PhD thesis, 1993.

[8] R. F. Tinder, "Asynchronous sequential machine design and analysis: A comprehensive development of the design and analysis of clock-independent state machines and systems," *Synthesis Lectures on Digital Circuits and Systems*, vol. 4, no. 1, pp. 1–236, 2009.

[9] V. P. Nelson, H. T. Nagle, J. D. Irwin, and B. D. Carroll, *Digital logic circuit analysis and design*. Prentice Hall, 1995.

[10] S. Stergiou, K. Daskalakis, and G. Papakonstantinou, "A fast and efficient heuristic esop minimization algorithm," in *Proceedings of the 14th ACM Great Lakes symposium on VLSI*. ACM, 2004, pp. 78–81.

[11] M. Kalathas, D. Voudouris, and G. Papakonstantinou, "A heuristic algorithm to minimize esops for multiple-output incompletely specified functions," in *Proceedings of the 16th ACM Great Lakes symposium on VLSI*. ACM, 2006, pp. 357–361.

[12] Y. Lin and R. Geng, "Mlbm: Machine-learning-based minimization algorithm for boolean functions," in *Industrial Electronics, 2009. ISIE 2009. IEEE International Symposium on*. IEEE, 2009, pp. 1160–1165.

[13] F. Kudlačák and E. Gramatová, "Asynchronous sequential circuits synthesis by high-performance computing," in *2014 14th Biennial Baltic Electronic Conference (BEC)*. IEEE, 2014, pp. 65–68.

[14] E. J. McCluskey Jr, "Minimization of boolean functions," *Bell system technical Journal*, vol. 35, no. 6, pp. 1417–1444, 1956.

[15] H. M. Jacobson and C. J. Myers, "Efficient algorithms for exact two-level hazard-free logic minimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 11, pp. 1269–1283, 2002.