# EXPLORANDO UMA SOLUÇÃO TRADICIONAL PARA O CÁLCULO DOS ARCOS DE TRANSIÇÃO EM CIRCUITOS BOOLEANOS

MATHEUS RENAN FREITAS DE FREITAS[1]; GUSTAVO CUNHA KNEIP[2]; ANTÔNIO ARAÚJO DE BRUM[3]; MAICON SCHNEIDER CARDOSO[4]; LEOMAR SOARES DA ROSA JUNIOR[5]

[1]*Universidade Federal de Pelotas – mrffreitas@inf.ufpel.edu.br*
[2]*Universidade Federal de Pelotas - gckneip@inf.ufpel.edu.br*
[3]*Universidade Federal de Pelotas - aadbrum@inf.ufpel.edu.br*
[4]*Universidade Federal de Pelotas - mscardoso@inf.ufpel.edu.br*
[5]*Universidade Federal de Pelotas - leomarjr@inf.ufpel.edu.br*

## 1. INTRODUÇÃO

O estudo dos arcos de transição para uma dada uma expressão booleana é etapa fundamental no processo de caracterização elétrica de um circuito, como visto em WUERDIG et al. (2023). tem-se por arco de transição a mudança de estado de uma única variável de entrada - tomando as outras variáveis por estados constantes - capaz de provocar uma mudança de estado na saída do circuito (LIMA et al., 2020).

Neste trabalho, é proposta uma metodologia prática para calcular os arcos de transição para uma única variável de uma determinada expressão booleana, dada em soma de produtos. Essa metodologia foi implementada em um programa utilizando a linguagem *C* (o qual refere-se, no presente trabalho, como ”AT”), com posterior avaliação qualitativa dessa solução.

## 2. METODOLOGIA

A solução proposta trata-se de um processamento completo da expressão, que culmina no cômputo de todos os arcos de transição via força bruta. Na figura 1, observa-se as etapas propostas para o desenvolvimento completo da solução.

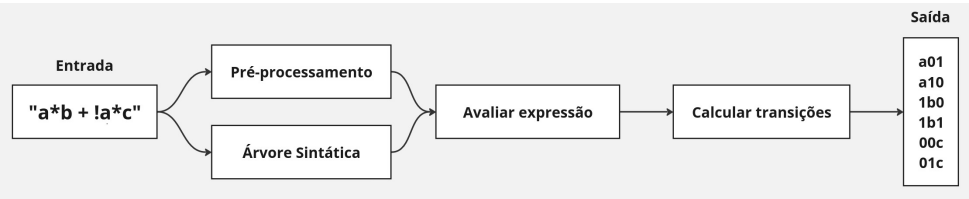


Figura 1: Diagrama metodológico da solução proposta.

Inicialmente realiza-se um pré-processamento da entrada, obtendo a quantidade de variáveis lógicas que o programa recebe e os seus símbolos. Como a entrada é uma soma de produtos (*SOP*), é necessário avaliar logicamente o significado da expressão, o que se faz construindo sua tabela verdade. Para tal, constrói-se a árvore sintática dessa expressão e avalia-se o resultado operando sobre ela, para cada conjunto de entradas possível. A abordagem escolhida consiste na interpretação da *SOP* como uma operação infixa, construindo a árvore sintática através de uma versão simplificada do algoritmo ”*shunting yard*”conforme descrito por DIJKSTRA (1961).

Para obter a tabela verdade da expressão, é realizada uma busca em profundidade na árvore. Busca-se um valor verdadeiro para qualquer nodo na camada dois (somas) da árvore; de forma análoga, pode-se negar imediatamente um nodo na camada três (produtos) se um de seus filhos apresentar ao menos uma falsidade. Se é possível percorrer a árvore inteira sem encontrar a condição de parada antecipada na camada 2, então o valor para esse conjunto de entradas é falso, caso contrário, é verdadeiro. Repete-se esse processo para todas as entradas possíveis, armazenando os resultados em um vetor.

Após esse processo, são combinados a tabela verdade gerada com os dados obtidos no pré-processamento da entrada para encontrar os arcos de transição, que são impressos conforme são encontrados. Isso é feito considerando os índices do vetor como palavras binárias que representam o estado de cada entrada, sendo o valor armazenado no vetor (zero ou um) a saída obtida. Esse processo é descrito no algoritmo 1 a seguir.

**Algoritmo 1** Encontrar os arcos de transição no vetor $v$

**Requer:** $v \leftarrow$ vetor contendo a tabela verdade da expressão
**Requer:** $x \leftarrow$ quantidade de operandos em $v$
**Requer:** $y \leftarrow$ vetor com os símbolos de cada operando, de maior a menor peso
**Garantir:** $arco$ como um vetor de caracteres alocado, de tamanho $x$

$l \leftarrow 0$
indice $\leftarrow 0$
**for** i $\leftarrow 0$ **até** $x - 1$ **faça**
    **for** j $\leftarrow 0$ **até** $2^{x-1} - 1$ **faça**
        **for** k $\leftarrow x - 1$ **até** $0$ **faça**
            peso $\leftarrow x - 1 - k$
            **se** $k \neq i$ **então**
                indice $\leftarrow$ indice $+ l$-ésimo bit de $j$ deslocado $peso$ vezes à esquerda
                $arco_k \leftarrow l$-ésimo bit de $j$
            **senão**
                $arco_k \leftarrow y_i$
            **fim do se**
        **fim do for**
    **fim do for**
    **se** $v_{\text{indice}} \neq v_{2^{(x-1)-i}}$ **então**
        Imprima o conteúdo do vetor $arco$
    **fim do se**
    $l \leftarrow 0$
    indice $\leftarrow 0$
**fim do for**

## 3. RESULTADOS E DISCUSSÃO

O programa desenvolvido está disponível na internet sob domínio público, através do *GitHub*, no repositório "*gckneip/AT*". As limitações teóricas para o programa são o número de variáveis utilizadas, cuja quantidade não pode exceder a disponibilidade de *bits* que a máquina-alvo pode processar; e também a memória disponível. Além

disso, como aceita caracteres latinos maiúsculos e minúsculos como símbolos de variáveis atômicas, está limitada a 52 variáveis. Na prática, dificilmente será alcançada essa limitação, haja vista que a quantidade de memória necessária para armazenar a tabela verdade do pior caso alcança $2^n\ B = 2^{52} B \approx 4,5\ PB$ (quatro petabytes e meio) de memória disponível necessária.

A figura 2 mostra o tempo de execução do programa - medido - para operações com diferentes quantidades de entradas (utilizou-se como expressão geratriz um único produto contendo todas as variáveis necessárias). A análise preliminar da complexidade algorítmica de execução resultou em exponencial, equivalente a $O(2^n)$, em que $n$ representa a quantidade de variáveis; resultado que é confirmado também na figura 2.

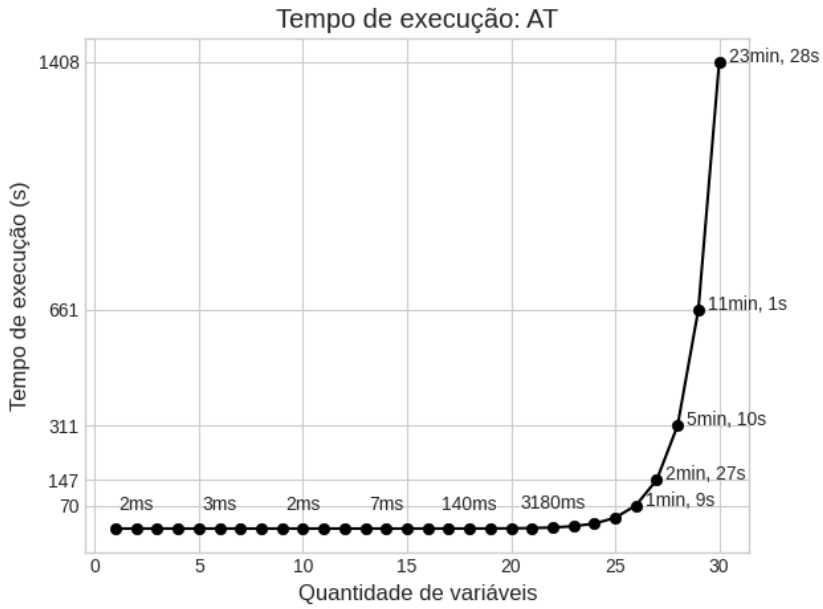


Figura 2: Métricas obtidas para a execução do algoritmo em diferentes condições.

## 4. CONCLUSÕES

A metodologia proposta se mostrou plenamente funcional e o *software* desenvolvido obteve boas métricas de desempenho.

A contribuição deixada por esse trabalho reflete-se na possibilidade de explorar soluções diferentes - menos triviais - para abordar esse mesmo problema. Já há soluções em desenvolvimento que utilizam técnicas como redes neurais para o cálculo dos arcos de transição, utilizando a ferramenta *AT* como **benchmark** e também como fonte de dados para os *datasets* de treino. Além disso, os módulos desenvolvidos individualmente para esse trabalho podem servir como base para outras aplicações e problemáticas abordadas na área da microeletrônica. Para trabalhos futuros, especificamente no âmbito da ferramenta aqui discutida, sugere-se a refatoração do código desenvolvido para favorecer o uso didático, através de uma documentação mais robusta. Por fim, observa-se que a solução desse problema, ainda que incipiente, foi um importante ponto de partida para estudos mais aprofundados nas de áreas de simulação elétrica, síntese lógica de circuitos e afins, servindo como marco inicial para outras linhas de pesquisa no contexto da microeletrônica.

## 5. REFERÊNCIAS BIBLIOGRÁFICAS

DIJKSTRA, Edsger Wybe. Algol 60 translation : An Algol 60 translator for the x1 and Making a translator for Algol 60. **ALGOL Bulletin**, 1961.

LIMA, Vitor Gonçalves et al. Enhancing Side Channel Attack-Resistance of the STTL Combining Multi-Vt Transistors with Capacitance and Current Paths Counterbalancing. **Journal of Integrated Circuits and Systems**, v. 15, n. 1, p. 1–11, 26 mai. 2020. ISSN 1872-0234. DOI: 10.29292/jics.v15i1.100. Disponível em: <https://jics.org.br/ojs/index.php/JICS/article/view/100>. Acesso em: 10 out. 2024.

WUERDIG, Rodrigo N. et al. LEX - A Cell Switching Arcs Extractor: A Simple SPICE-Input Interface for Electrical Characterization. In: 2023 IEEE Computer Society Annual Symposium on VLSI (ISVLSI). [S.l.: s.n.], 2023. P. 1–6. DOI: 10.1109/ISVLSI59464.2023.10238671.