Hoje em dia o desenvolvimento de circuitos integrados se baseia em um fluxo dividido em diversas etapas. Entre estas, uma estágio que consume muito tempo é denominada mapeamento tecnológico. Esta etapa monta um arranjo de células de forma a representar uma dada função Booleana. Logo, para executar este passo precisamos saber quais células da biblioteca são equivalentes a um bloco lógico requisitado.

O problema de determinar se duas funções Booleanas são equivalentes através de permutações de suas variáveis é denominado equivalência P (Permutação). No pior caso uma busca baseada em permutações para um problema de tamanho N levará N! passos até uma solução ser encontrada. Logo, em um mapeamento em biblioteca, uma busca baseada no caso exaustivo pode se tornar algo muito custoso.

Uma função lógica pode ser representada por diversas expressões lógicas. Uma expressão pode ser melhor do que outra, dado um critério de escolha. Sendo que o critério é minimizar o numero de literais, o exemplo Eq.2 é melhor que o exemplo Eq.1, pois Eq.1 tem 6 literais contra 4 literais de Eq.2.

Ex:

$$f = a * b * c + a * c * d \quad \text{(Eq.1)}$$

$$f' = a * c * (b + d) \quad \text{(Eq.2)}$$

Funções com expressões normalizadas aceleram o processo de busca por equivalência. Dentro desse conjunto, a classe Read-Once é composta com funções que apresentam cada literal somente uma vez em sua expressão lógica. Assim, *f'* no exemplo acima, é uma função Read-Once, pois suas variáveis {a,b,c,d} aparecem somente uma vez. Note ainda que uma expressão Read-Once tem estrutura canônica.

Assim, trabalhar com estruturas canônicas em funções booleanas possibilita normalização, isto é, mexer na ordem dos literais e operadores de modo que duas expressões se tornem equivalentes quanto à estrutura.

Portanto, o trabalho desenvolvido se concentra neste enfoque: um método para calcular equivalência P entre funções lógicas pertencentes à classe Read-Once. Nossa abordagem transforma uma expressão booleana em uma árvore lógica (estrutura com facilidades de manipulação) e constrói um código para ordenar a árvore gerada em uma forma não ambígua que coincida com outra árvore lógica equivalente (ordenando assim a expressão). Por usar funções Read-Once, duas expressões equivalentes terão uma estrutura canônica comum e, portanto podem ser normalizadas igualmente.

O método desenvolvido foi validado e comparado com [1-2] provando sua eficácia. O próximo passo será integrar o algoritmo ao atual fluxo de projeto para comprovar a eficácia do método.

[1] U. Hinsberger and R. Kolla, "Boolean matching for large libraries," Proc. Design Automation Conference (DAC), 1998.

[2] D. Debnath and T. Sasao, "Efficient computation of canonical form for Boolean matching in large libraries," Proc. Asia and South Pacific Design Automation Conference (ASP-DAC) 2004.

[3] Tsutomu Sasao and Jon T Butler. Progress in Applications of Boolean Functions. Synthesis Lectures on Digital Circuits and Systems, 2009, Vol. 4, No. 1 , Pages 1-153