FERRAMENTA DE VISUALIZAÇÃO GRÁFICA PARA ESTRUTURA NETLIST

DETONI, Douglas; DA ROSA JR, Leomar S.; MARQUES, Felipe de S.

Grupo de Arquiteturas e Circuitos Integrados - GACI
CDTEC – Computação
Universidade Federal de Pelotas
{ddetoni, leomarjr, felipem}@inf.ufpel.edu.br

1 INTRODUÇÃO

Com a evolução dos circuitos digitais, tanto em tamanho quanto em desempenho, criou-se a necessidade de usar formatos e ferramentas que facilitem o processo de síntese lógica (DA ROSA JUNIOR, 2009). Atualmente existem diversos formatos diferentes que são usados para representar uma *netlist*, que é a estrutura usada na etapa de *design* de circuitos. Assim como existem formatos, também são necessárias ferramentas que os manipulem de diversas maneiras (GONÇALVES, 2012). Este trabalho propõe a implementação de uma ferramenta que seja capaz de manipular uma *netlist* de maneira visual, através de uma interface gráfica simples, porém com diversos recursos. Esta ferramenta tem como base uma estrutura de *netlist* implementada na linguagem Java, a qual possui suporte para os formatos *netblif e verilog.* É usado o *framework JUNG* para realizar a visualização da estrutura em forma de grafos, obedecendo aos padrões de desenho de circuitos que são utilizados nas demais ferramentas usadas no mercado de microeletrônica.

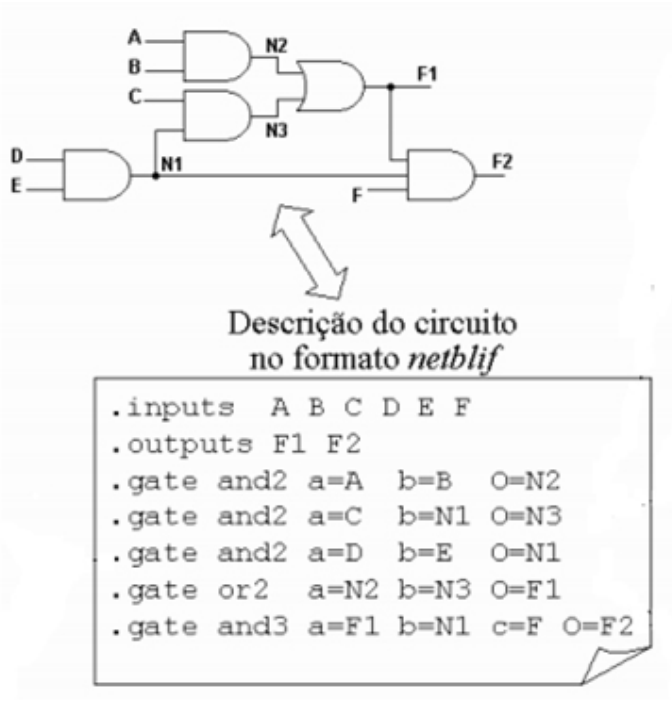
2 METODOLOGIA

Tendo como base uma *netlist,* a qual é uma estrutura capaz de representar a conectividade de um *design* de circuito tratando objetos como *nets, pin, terminal, cell,* entre outros componentes, fez-se necessário analisar estes objetos e transformá-los em uma estrutura de um grafo direcional acíclico (DAG). Isso foi necessário porque o *framework JUNG,* que foi o *framework* escolhido para auxiliar o desenvolvimento da ferramenta em relação à operação com estruturas de grafo, necessita da mesma para poder gerar uma visualização gráfica.

Como formatos de arquivo de entrada da ferramenta, inicialmente optou-se pelo *netblif* e *verilog.* O primeiro é uma extensão do formato *BLIF* (*Berkeley Logic Interchange Format)* que consiste de entradas, saídas e *latches* interconectados, de modo a representar um circuito combinacional (SENTOVICH, 1992). O *verilog*, de maneira semelhante, também especificada um circuito combinacional. Os dois formatos, apesar de representarem *netlist* possuem sintaxes diferentes, por isso seus *parsers* são independentes. Também é possível que a ferramenta abranja outros formatos, aumentando sua área de atuação. Exemplos de ambos os formatos podem ser vistos na Fig. 1.

Cada nodo do grafo possui uma identificação que permite gerar elementos gráficos diferentes. Isso ocorre, por exemplo, em nodos *terminal* e *instance.* Um nodo *terminal* representa uma entrada ou saída de determinado *design,* sua representação gráfica é um quadrado com uma identificação de qual entrada ou saída ele se refere. Entretanto, um nodo *instance* representa uma instância de um

design, isto é, um *design* interno o qual está sendo visualizado. Deste modo é necessária uma representação gráfica que possua entradas e saídas, assim como uma identificação do *design*. Outro aspecto visual é a representação das arestas do DAG que devem ser desenhadas de forma ortogonal devido ao padrão usado em *layout* de circuitos.



Descrição de um circuito no formato Verilog

```
module  mux_using_assign(
din_0       , // Mux first input
din_1       , // Mux Second input
sel         , // Select input
mux_out       // Mux output
);
//-----------Input Ports---------------
input din_0, din_1, sel ;
//-----------Output Ports---------------
output mux_out;
//------------Internal Variables--------
wire  mux_out;
//-------------Code Start-----------------
assign mux_out = (sel) ? din_1 : din_0;

endmodule //End Of Module mux
```

Figura 1. Exemplos de circuitos nos dois formatos suportados.

Deseja-se poder carregar um *netlist* e navegar entre as camadas do *design*, assim como adicionar e apagar outros elementos, organizar os elementos na tela, salvar as mudanças, entre outras operações.

3 RESULTADOS E DISCUSSÃO

A ferramenta, atualmente, é capaz de gerar uma visualização de um *netlist* pequeno, com as representações gráficas dos nodos bem definidas, assim como o desenho ortogonal das arestas. As dificuldades com estruturas grandes vêm da falta de algoritmos de *layout* eficientes que possam organizar na tela os diversos elementos de um *design* sem sobreposição de nodos, fios cruzados e muitos outros problemas encontrados no desenho de um *layout*. Um exemplo de visualização pode ser observado na Fig. 2.

A navegação entre *designs* internos ainda não está implementada. O trabalho está em andamento, onde se estuda uma maneira eficiente de manter o histórico da navegação, para que *netlists* com múltiplas camadas não sobrecarreguem a ferramenta.

O *framework JUNG* dispõe de várias opções para personalização de um grafo. Isso possibilitou que o padrão de desenho fosse usado sem que uma estrutura de grafo fosse implementada especificamente para isso. É importante ressaltar que a ferramenta baseia-se inteiramente neste *framework*, sendo pré-requisito entende-lo para compreender e acrescentar ao código da ferramenta (JUNG, 2009).

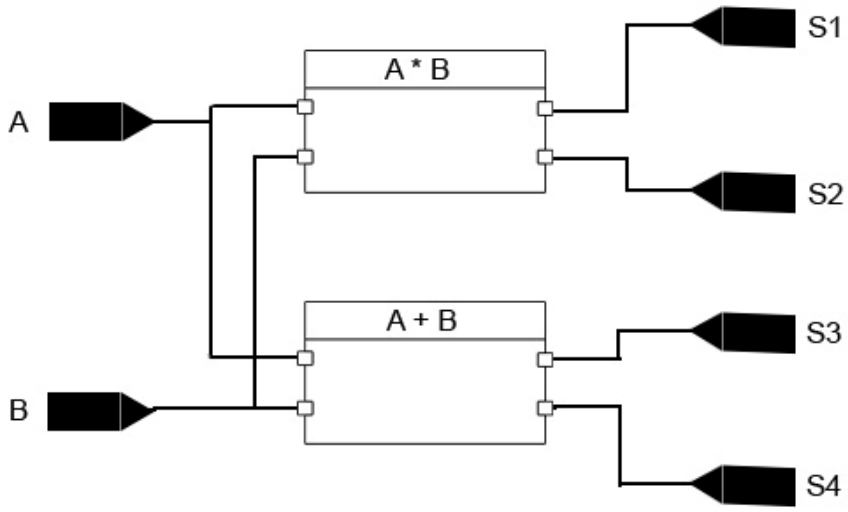


Figura 2. Exemplo de visualização de circuito com a ferramenta.

4 CONCLUSÃO

Existe muito trabalho ainda por ser feito. Contudo, a intenção é de que a ferramenta seja eficiente o suficiente para que *benchmarks* de tamanho grande possam funcionar corretamente. Algumas dificuldades em relação à junção e adaptação das estruturas de *netlist* e grafo foram encontradas. Por ser uma estrutura independente da ferramenta de visualização, a estrutura de *netlist* não facilita a extração do grafo que a representa. Isso certamente influenciará na eficiência da ferramenta.

A escolha da linguagem Java permitirá que qualquer plataforma com a JVM (*Java Virtual Machine*) instalada utilize a ferramenta. Isso amplia muito leque de pessoas que poderão trabalhar com ela.

A ferramenta tem um potencial de desenvolvimento muito bom. Podendo agregar bastante a projetos de circuitos digitais (POSSANI, 2012). Suas funcionalidades não se prendem somente as já citadas no artigo. Muita coisa poderá ser incrementada tornando-a cada vez mais completa. O trabalho realizado até o momento ainda encontra-se em fase de desenvolvimento, porém já é possível disponibilizar uma versão inicial capaz de auxiliar os projetistas na visualização e no desenvolvimento de circuitos digitais.

5 REFERÊNCIAS

DA ROSA JUNIOR, L. S.; MARQUES, F. S.; SCHNEIDER, F.; RIBAS, R. P.; REIS, A. I. “A Comparative Study of CMOS Gates with Minimum Transistor Stacks”. In: **20th ACM Symposium on Integrated Circuits and Systems Design**, Rio de Janeiro, 2007, p. 93-98.

SENTOVICH, E. et al. “SIS: A system for sequential circuit synthesis”. In: **Technical Report No. UCB/ERL M92/41, EECS Department, University of California, Berkeley**, 1992.

JUNG 2.0 Tutorial (2009). www.jung.sourceforge.net. Acessado em: 22/07/2012.

GONÇALVES, S.; MARTINS, M.; COLVARA, M.; REIS, A.; RIBAS, R.; DA ROSA JUNIOR, L. S.; MARQUES, F. “Technology Mapping for QCA Devices”. In: **XXVII Simpósio Sul de Microeletrônica**, 2012.

POSSANI, V. N.; SOUZA, R. S.; DOMINGUES JR., J. S.; AGOSTINI, L. V.; MARQUES, F. S.; DA ROSA JUNIOR, L. S. “Optimizing Transistor Networks Using a Graph-Based Technique”. In: **Journal of Analog Integrated Circuits and Signal Processing**, Springer, 2012.