Metodologia para síntese automática de máquinas de estados finitos baseada em descrição em alto nível de abstração

Eduardo Lopes da Cruz[1], Tiago da Silva Almeida[2], Alexandre César Rodrigues da Silva[3]

Abstract *— This paper present a methodology of synthesis of descriptions in high-level of abstraction for finite state machines. The synthesis process has a tools collection to perform the simulation, the reading the relevant instructions, the conversion of this instructions, the minimization and the generation of code and hardware implementation. To evaluate the methodology was used the line code HDB1 as case study. The results were obtained a minimum circuit of low cost.*

Index Terms — Automatic code generation, high-level of abstraction, minimization, translation tools, finite state machine.

INTRODUÇÃO

A busca pela redução de custos e tempo de fabricação de CIs (Circuitos Integrados), automação de processos industriais e maior eficiência em sistemas de aquisição de dados levaram a grandes avanços tecnológicos nas últimas décadas. Tais avanços fizeram surgir ASICs (*Application Specific Integrated Circuit*) mais complexos e desenvolvidos para uma grande diversidade de aplicações, como, por exemplo, a indústria automotiva, aeroespacial, telecomunicações, etc.

Porém, o desenvolvimento de novas tecnologias não é uma tarefa trivial e demanda muito esforço do projetista e das ferramentas que auxiliam na realização de projetos, fazendo-se necessário o surgimento de novas metodologias e ferramentas computacionais que possam ajudar a resolver problemas de projetos [1].

A ausência de um padrão bem definido para a especificação em alto nível de abstração fez surgir uma variedade muito grande de metodologias e ferramentas para sintetizar as descrições envolvidas na criação de um determinado projeto.

O primeiro trabalho a classificar os passos envolvidos em projetos de sistemas e circuitos em alto nível de abstração foi [2], posteriormente aperfeiçoado para um contexto mais atual por [3], [4].

A representação visual de sistemas e circuitos, em alto nível de abstração, facilita a verificação de maneira formal e metodologias podem ser desenvolvidas para auxiliar no processo de síntese [2-4]. Essa representação visual pode ser feita por meio de máquinas de estados finitos (MEF), o que vem sendo feito em muitos trabalhos [5-7].

Com base nesse contexto, este trabalho apresenta uma coleção de ferramentas computacionais para auxiliar o processo de síntese de circuitos, utilizando como alvo as MEFs. A partir de uma descrição textual da MEF, na linguagem XML (eXtensible Markup Language), uma das ferramentas descrita neste trabalho é capaz de gerar uma entrada para outra ferramenta que realiza a otimização da MEF. Tais ferramentas foram avaliadas com um código de linha HDB1.

O presente trabalho está organizado da seguinte forma: a Seção **Definição de máquinas de estados finitos** apresenta a definição e as formas de representação das MEFs, a Seção **Ferramentas utilizadas** descreve as ferramentas computacionais utilizadas na metodologia proposta. A Seção **Resultados** apresenta o funcionamento da ferramenta XML2TAB bem como os resultados obtidos e a Seção **Conclusão** apresenta as conclusões e perspectivas para trabalhos futuros.

DEFINIÇÃO DE MÁQUINAS DE ESTADOS FINITOS

MEFs são representações abstratas de circuitos sequenciais. Mais precisamente, Segundo [6], MEFs são representações abstratas de comportamentos sequenciais de sistemas, definindo a resposta do sistema em relação a uma sequência de eventos de entradas.

A Referência [7] afirmou que tais máquinas são amplamente utilizadas na modelagem de sistemas em diversas áreas como circuitos sequenciais, protocolos de comunicação, entre outros. A utilização de MEF possibilita ao projetista a capacidade de prever como o sistema se comportará em função da execução de uma determinada ação sincronizada por ciclos de relógio.

Uma MEF (circuito sequencial) possui um número finito de entradas, constituindo o conjunto N das variáveis de entradas, ou seja, $N = \{N_1, N_2,...,N_n\}$. Possui, também, um conjunto finito M de saída, sendo $M = \{M_1, M_2, ..., M_m\}$. O valor contido em cada elemento de memória forma o conjunto K das variáveis de estado, $K = \{K_1, K_2, ..., K_k\}$, e define o estado atual da máquina. As saídas e as funções de transições internas que geram o conjunto $S = \{S_1, S_2, ..., S_s\}$

[1]Eduardo Lopes da Cruz - Faculdade de Engenharia de Ilha Solteira, UNESP – Univ. Estadual Paulista, Av. Prof. José Carlos Rossi, 1370, Ilha Solteira, São Paulo, Brasil, cruz.duda@gmail.com
[2]Tiago da Silva Almeida - Faculdade de Engenharia de Ilha Solteira, UNESP – Univ. Estadual Paulista, Av. Prof. José Carlos Rossi, 1370, Ilha Solteira, São Paulo, Brasil, 77.tiago@gmail.com
[3]Alexandre César Rodrigues da Silva, Faculdade de Engenharia de Ilha Solteira, UNESP – Univ. Estadual Paulista, Av. Prof. José Carlos Rossi, 1370, Ilha Solteira, São Paulo, Brasil, acrsilva@dee.feis.unesp.br

(próximo estado) dependem das N entradas e dos K estados atuais da MEF e são definidos através dos circuitos combinacionais. As MEFs podem ser representadas pelas seguintes equações:

$$K(t + 1) = f[K(t),N(t)] . \quad (1)$$

onde *f* é a função de transição de estados. O valor da saída M(t) é obtido através de duas formas:

$$M(t) = g[K(t)]. \quad (2)$$

$$M(t) = g[K(t),N(t)]. \quad (3)$$

onde *g* é a função de saída.

Uma MEF com propriedades descritas nas equações (1) e (2) é denominada modelo de Moore e uma MEF descrita através das equações (1) e (3) é denominada modelo de Mealy [5].

Uma MEF pode ser representada por um diagrama de transição de estados (DTE) ou por uma tabela de transição de estados (TTE). O DTE mostrado na Figura 1, utilizado neste trabalho, representa o código de linha HDB1 utilizado em linhas de telecomunicações. Este código de linha tem a finalidade de limitar o número de zeros em uma sequência, pois tais sequências podem dificultar a recuperação de informações [8].

Em sua execução, o código de linha HDB1 transforma a informação digital em um sinal ternário onde zeros isolados são codificados pelo nível de tensão "0". Quando ocorrem dois ou mais zeros consecutivos, para cada par de zeros é transmitido como sinal +1 +1 se o último sinal anterior ao par for de paridade -1, se o último sinal for +1, o par será transmitido como -1 -1[8]. Sendo assim, esta codificação aumenta o número médio de pulsos, facilitando o sincronismo na recepção.

Uma TTE descreve o comportamento da MEF de maneira textual. A TTE pode organizar os dados na seguinte ordem: estado atual, entrada, próximo estado e saída, mas outras organizações dos dados também são aceitas.

FERRAMENTAS UTILIZADAS

Algumas ferramentas utilizadas neste trabalho são amplamente aceitas na academia e indústria, como exemplo o MATLAB / Simulink, e outras foram desenvolvidas para a realização da metodologia. Cada uma delas serão descritas nas subseções subsequentes.

MATLAB/Simulink - Stateflow

O Stateflow estende o Simulink com um ambiente de projeto para o desenvolvimento de diagramas de estados e fluxogramas. O Stateflow fornece os elementos necessários para descrever a lógica complexa de uma forma legível e compreensível. É totalmente integrado com o MATLAB e Simulink, proporcionando um ambiente para a concepção de sistemas embarcados que contém controle, supervisão e modo lógico [9].

Na Figura 1 ilustra-se a especificação da MEF de código de linha HDB1 na forma de DTE produzida a partir da *toolbox* Stateflow.

Os quadrados arredondados representam os estados, e as setas representam as transições. Para modelagem do diagrama é necessário obedecer algumas regras do Stateflow, como, a atribuição de um nome ao estado, contido dentro do símbolo, não deve começar com caracteres numéricos e a condição de transição deve obedecer a seguinte sintaxe:

$$[i[t] == 0]\{o = 0;\} . \quad (4)$$

onde, o ***i*** representa a variável que irá receber o valor de entrada, o ***t*** é uma variável que representa um tempo discreto, o ***o*** é uma variável de saída. Os colchetes mais externos representam a condição na entrada para que a saída entre chaves seja obtida, e transite de um estado para outro. Os sinais iguais duplos (==) representam a verificação de uma condição e o sinal igual sozinho (=) representa a atribuição de valor [9].

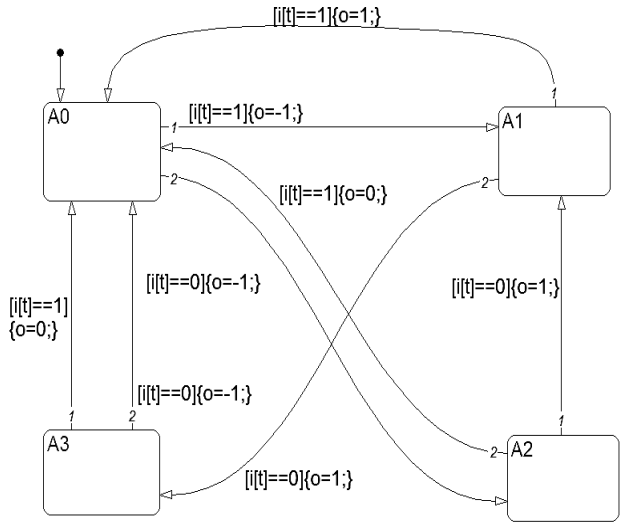

FIGURA. 1
DTE DESENVOLVIDO EM STATEFLOW, UTILIZADO COMO PASSO INICIAL DE SÍNTESE EM ALTO NÍVEL DE ABSTRAÇÃO.

Na Figura 2 ilustra-se o resultado da simulação do DTE apresentado na Figura 2.

Os valores de entrada estão dispostos no vetor T e os valores de saída mostrados no vetor Z.

T=00101001111011101010. (5)

S=-10-10-111-11-110-11-110-10-1 (6)

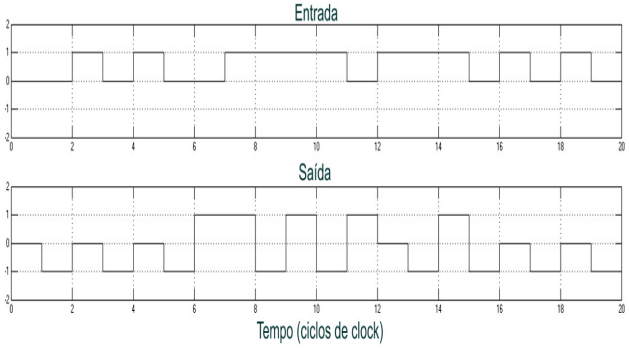

FIGURA 02.
RESULTADO DA SIMULAÇÃO EM STATEFLOW.

Padrão de criação de documentos XML

O XML é um padrão de criação de documentos com dados organizados de forma hierárquica. É baseado em texto, capaz de descrever vários tipos de dados, listas e árvores, criando uma infraestrutura única, permitindo sua utilização em diversas áreas como computação, engenharia, matemática, etc.

O XML utiliza-se de *tags* para marcar uma instrução e, uma *tag* deve conter marcas de início e de fim de instrução.

Neste trabalho foi utilizado uma derivação específica de XML para a representação de MEFs denominada SCXML (*StateChart XML*), que é a combinação do padrão em XML CCXML (controle de chamadas em XML) e diagramas de estados de *Harel*. Um arquivo SCXML permite ao projetista definir o estado inicial da MEF, bem como as entradas, transições de estados e saídas. Permite também a utilização do conceito de hierarquia na construção das MEFs [10].

Ferramenta TABELA

A ferramenta denominada TABELA também faz parte do conjunto de ferramentas utilizadas e é uma das motivações para a execução deste trabalho. A ferramenta TABELA gera a tabela de transição de estados de uma MEF a partir de seu DTE e minimiza as funções de transições internas correspondentes aos elementos de memória utilizados e as funções de saída do circuito [5].

Para a utilização do TABELA o usuário deve fornecer a quantidade de bits de saída, quantidade de *flip-flops*, tipo de cada *flip-flop* (D ou JK), quantidade de entradas e a tabela de estados na seguinte ordem: estado atual, próximo estado, entrada e saída. A TTE deve estar na representação decimal e a MEF representada preferencialmente na forma do modelo de *Mealy*.

O método de minimização utilizado pelo programa TABELA é o de Quine-McCluskey. O algoritmo de Quine-McCluskey é um método clássico que possui duas fases: a obtenção dos implicantes primos, e a cobertura irredundante, onde a partir do conjunto de implicantes primos são obtidos os implicantes essenciais para a realização da função. De acordo com [11], a intenção do método Quine-McCluskey é proporcionar procedimento algorítmico para fornecer os implicantes primos.

Um implicante primo é um termo que não pode ser combinado a outro para a formação de um novo termo e, um implicante primo essencial é um implicante que cobre pelo menos um termo da função que não é coberto por nenhum outro implicante primo.

A aplicação do método de Quine-McCluskey segue a seguinte ordem:

- Classificam-se e agrupam-se em conjuntos de termos da função booleana de acordo com seus índices (mesmo números de 1's em sua forma binária) de forma crescente.
- Comparam-se todos os termos de um dado grupo com cada termo do grupo seguinte, ou seja, de índice imediatamente superior, mediante a utilização do teorema $AB.A\overline{B}=A$. Aplica-se sucessivamente este teorema comparando cada termo do grupo do índice *i* com todos os termos do grupo do índice $i + 1$ até esgotarem as possibilidades. O termo resultante consiste na representação fixa original com o dígito diferente substituído por um *X*. Por outro lado, marcam-se com setas todos os termos comparados com ao menos outro termo.
- Após tabular os termos comparados, procede-se novamente conforme exposto no item acima até esgotarem-se as possibilidades. Os termos que ficarem sem a seta marcada formam o conjunto dos termos irredutíveis, ou seja, os termos da expressão simplificada.

Ferramenta SF^{2}XML

O SF^{2}XML (*Stateflow to XML*) é uma ferramenta computacional capaz de capturar as informações relevantes no arquivo de projeto do ambiente Stateflow e gerar uma descrição correspondente em XML. O arquivo resultante da conversão está de acordo com o padrão SCXML proposto pelo W3C (*World Wide Web Consortium*).

Ferramenta XML2TAB

Visando extinguir a necessidade do trabalho manual no processo de criação do arquivo de entrada a ser fornecido ao programa TABELA, foi desenvolvida a ferramenta chamada XML2TAB (*XML to TABELA*). A ferramenta é capaz de gerar automaticamente um arquivo de entrada padronizada para o programa TABELA, a partir de uma descrição SCXML gerado pela SF^{2}XML.

Todas estas informações formam as expressões de transição que serão realizadas pela MEF. A partir de tais transições, a ferramenta XML2TAB calcula a quantidade de *flip-flops* necessários para a implementação da MEF bem

como a quantidade de bits de entrada e bits de saída, deixando a cargo do projetista somente a especificação dos tipos de *flip-flops* que serão utilizados.

Sabe-se que cada *flip-flop* pode armazenar dois estados distintos (0 ou 1), obtemos a seguinte equação:

$$E \leq 2^n \qquad (7)$$

onde *n* é a quantidade de *flip-flops* necessários para a representação da MEF e *E* a quantidade de estados da MEF. Para atender esta restrição, foi criado a função especificada no Algoritmo 1, que calcula a quantidade de *flip-flops* necessários.

ALGORITMO 1: CÁLCULO DE FLOP-FLOPS

```
Início
  Flip-flop←1;
  S←0;
  Enquanto (flip-flop≤quant_estados) faça
    S←exp(2,flip-flop);
    Se S≥ quant_estados então
      Parar;
    Fim-se
    Senão
      Flip-flop←flip-flop+1;
  Fim-enquanto
Fim
```

A estrutura de dados utilizada para armazenar as transições em memória para que posteriormente possam ser manipuladas para criar o arquivo de entrada padrão para o TABELA é uma lista encadeada. Esta estrutura armazena as informações do estado atual, entrada, próximo estado e saída.

No cálculo relacionado a quantidade de bits de entrada e saída, os maiores de entrada e saída em decimal são convertidos para suas respectivas representações em binário, e a quantidade de bits necessária para representar tais números utilizados se tornam então os parâmetros de entrada e saída.

Resultados

Como foi apresentado na Seção **Padrão de criação e documentos XML**, foi utilizado o arquivo SCXML com a especificação de uma MEF do código de linha HDB1 como implementação de estudo de caso. A Figura 3 ilustra o arquivo gerado pelo programa XML2TAB que é também arquivo de entrada para o TABELA.

No arquivo de entrada para o TABELA, a primeira linha representa a quantidade de *flip-flops* necessários para a implementação da MEF, as linhas seguintes determinam os tipos de *flip-flops* a serem utilizados.

Após a especificação dos tipos de *flip-flops*, são determinadas as quantidades de bits de entrada e de saída respectivamente, e as linhas seguintes determinam as transições. Por fim, uma instrução -100 é aplicada informando a ferramenta o fim do arquivo.

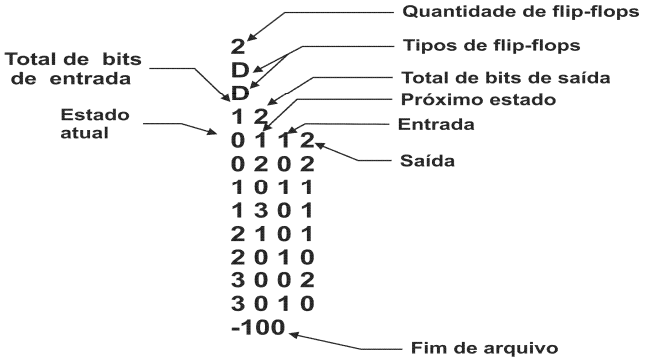

FIGURA 3.
ARQUIVO DE ENTRADA PADRÃO PARA O TABELA

Na Figura 4 apresenta-se o resultado obtido após a otimização realizada pelo TABELA. Na Figura 4 estão as informações de custo mínimo de cada função, bem como os mintermos, implicantes primos essenciais, sua redundância e o custo mínimo para a implementação de todas as funções da MEF do código de linha HDB1.

O TABELA fornece ao usuário as informações relacionadas às funções internas de entrada e saída, dos *flip-flops* utilizados, e também informa *don't care states*, que são implicantes primos não essenciais ou irrelevantes da MEF.

```
!DE!P/!ENTRADA !MINT!!Q1!Q1+!D1!!Q0!Q0+!D0!!Z1!Z0!
! 3! 0! 1 (1) !   7!! 1!  0! 0!! 1!  0! 0!! 0! 0!
! 3! 0! 0 (0) !   3!! 1!  0! 0!! 1!  0! 0!! 1! 0!
! 2! 0! 1 (1) !   6!! 1!  0! 0!! 0!  0! 0!! 0! 0!
! 2! 1! 0 (0) !   2!! 1!  0! 0!! 0!  1! 1!! 0! 1!
! 1! 3! 0 (0) !   1!! 0!  1! 1!! 1!  1! 1!! 0! 1!
! 1! 0! 1 (1) !   5!! 0!  0! 0!! 1!  0! 0!! 0! 1!
! 0! 2! 0 (0) !   0!! 0!  1! 1!! 0!  0! 0!! 1! 0!
! 0! 1! 1 (1) !   4!! 0!  0! 0!! 0!  1! 1!! 1! 0!

FUNCAO D1
==========
MINTERMOS :  0;  1;

IMPLICANTES PRIMOS ESSENCIAIS :
ESSENCIAL:  0    REDUNDANCIA:      1 ->  00X

CUSTO FINAL DE D1 = 2

FUNCAO D0
==========
MINTERMOS :  4;  1;  2;

IMPLICANTES PRIMOS ESSENCIAIS :
ESSENCIAL:  2    REDUNDANCIA:      0 ->  010
ESSENCIAL:  1    REDUNDANCIA:      0 ->  001
ESSENCIAL:  4    REDUNDANCIA:      0 ->  100

CUSTO FINAL DE D0 = 12

FUNCAO Z1
==========
MINTERMOS :  4;  0;  3;

IMPLICANTES PRIMOS ESSENCIAIS :
ESSENCIAL:  3    REDUNDANCIA:      0 ->  011
ESSENCIAL:  0    REDUNDANCIA:      4 ->  X00

CUSTO FINAL DE Z1 = 7

FUNCAO Z0
==========
MINTERMOS :  5;  1;  2;

IMPLICANTES PRIMOS ESSENCIAIS :
ESSENCIAL:  2    REDUNDANCIA:      0 ->  010
ESSENCIAL:  1    REDUNDANCIA:      4 ->  X01

CUSTO FINAL DE Z0 = 7

CUSTO TOTAL DAS 4 FUNCOES = 28
```

FIGURA 4.
RESULTADO MINIMIZAÇÃO OBTIDO PELO TABELA DA MEF PARA O ESTUDO DE CASO DO HDB1.

Para a geração do código correspondente ao circuito combinacional que representa o código de linha HDB1 foi utilizado o programa TAB2VHDL [12]. O TAB2VHDL é responsável por gerar o código VHDL (*VHSIC Hardware Description Language*), em domínio funcional, a partir do arquivo gerado pelo TABELA, inferindo os *flip-flops* utilizados e gerando o circuito combinacional referente ao comportamento da MEF. Inicialmente, o TAB2VHDL faz a leitura do arquivo gerado pelo TABELA e a partir dele é gerado um arquivo com extensão ".vhd" contendo o código VHDL, o qual é ilustrado na Figura 5.

```
ENTITY TESTE IS
    PORT(
     CLK, CLR : IN BIT;
     X0 : IN BIT;
     Q0, Q1 : OUT BIT;
     Z0 : OUT BIT );
END TESTE;

ARCHITECTURE RTL OF TESTE IS

SIGNAL  VE0, VE1: BIT;

SIGNAL D1, D0 : BIT;

 BEGIN

PROCESS(CLK, CLR)
BEGIN
    IF CLR = '0' THEN
       VE0 <= '0';
    ELSIF CLK'EVENT and CLK = '1' THEN
       VE0 <= D0;
    END IF;
    Q0 <= Ve0;
END PROCESS;

PROCESS(CLK, CLR)
BEGIN
    IF CLR = '0' THEN
       VE1 <= '0';
    ELSIF CLK'EVENT and CLK = '1' THEN
       VE1 <= D1;
    END IF;
    Q1 <= Ve1;
END PROCESS;

  D1 <=  ( NOT(X0) AND (VE0));
  D0 <=  ( NOT(X0) AND NOT(VE1) AND NOT(VE0));
  Z0 <=  ( NOT(X0) AND (VE1));

END RTL;
```

FIGURA 5.
CÓDIGO VHDL GERADO PELO TAB2VHDL DA MEF PARA O ESTUDO DE CASO DO HDB1.

CONCLUSÃO

Neste trabalho foi apresentada uma metodologia para a síntese automática de descrições em alto nível de abstração com a finalidade de se produzir um circuito com custo mínimo. Todas as ferramentas utilizadas estão totalmente relacionadas entre si, sendo que a cada passo no processo de síntese uma ferramenta diferente é utilizada para a execução de uma tarefa importante para a obtenção do resultado final.

Este trabalho reforça o conceito da utilização de ferramentas de síntese na execução de projetos de circuitos e sistemas digitais, visando cada vez mais a diminuição da intervenção humana nas etapas do projeto, aumentando a precisão e a possibilidade de várias formas de simulação e implementação do projeto final.

Uma melhoria que pode ser adicionada a ferramenta XML2TAB é a formulação automática de um problema de programação linear a partir das funções mínimas fornecidas pelos métodos de minimização existentes, permitindo a simulação e verificação através de outras ferramentas computacionais além do TABELA e do TAB2VHDL.

AGRADECIMENTOS

Os autores gostariam de agradecer ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq, processos: 307255/2009-3 141744/2010-3), ao Departamento de Engenharia Elétrica da Universidade Estadual Paulista - UNESP - Campus de Ilha Solteira e também ao FUNDUNESP (Fundação para o desenvolvimento da UNESP).

REFERÊNCIAS

[1] Silva, A., C.R, "Data converter design and synthesis using hardware description languages", *Technical Report,* University of Limerik, 2007.

[2] Gajski, D., D., Kuhn, R., H., "Introduction to new VSLI tools", *IEEE Computer,* 16, No 12., 1983, 11-14.

[3] Riesgo, T., Torroja, Y., Torre, E., "Design Methodologies bades on hardware description languages", *IEEE transactions on Industrial Eletronics ,* 46, No 1., 1999, 3-12.

[4] Gerstlauer, A., Haubert, C., Pimentel, A., D., Stefanov, T., P., Gajski, D., D., Teich, J., "Eletronic System – Level Synthesis Methodologies", *IEEE Transactions on Computer Aided Design of Integrated Circuit and Systems,* 28, No 10., 2009, 1517-1530.

[5] Silva, A. C., R., "Contribuição a minimização e simulação de circuitos lógicos", *Masther's thesis,* Universidade Estadual de Campinas,1989.

[6] Milligan, G., Vanderbaunwhede, W., "Implementation of Finite State Machine on a Reconfigurable Devices", *Conference on Adaptative Hardware and Systems,* No 7., 2007, 386-396.

[7] Lee, D., Yannakakis, M., "Principles and Methodsof Testing Finite State Machines – A Survey", *Proceedings of the IEEE,* 84, No 8., 1996, 1090-1123.

[8] Almeida, T., S., "Ambiente computacional para projetos de sistemas com tecnologia mista ", *Masther's thesis*, Universidade Estadual Paulista – UNESP, 2009.

[9] Karris, S., T., *Introduction to Stateflow with applications,* Orchard Publications.

[10] Auburn, R., J., Barnett, J., Bodell, M., Raman, T., V., "State Chart XML (SCXML): State Machine Notation for Control Abstraction 1.0", *site. http://www.w3.org/TR/2005/WD-scxml-20050705/.*

[11] Jain, T., K., Kushwaha, D., S., Misra, A., K., "Optimization of the Quine-McCluskey Method for the minimization of the Boolean expression", *International Conference on Automatic and Autonomous Systems,* No 4., 2008,164-168.

[12] Tancredo, L., O., "TAB2VHDL: um ambiente de síntese lógica para máquinas de estados finitos", *Masther's thesis,* Faculdade de Engenharia, Universidade Estadual Paulista,2002.