UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Campus de Ilha Solteira

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

"Ferramentas para a Integração de Redes de Petri e VHDL na Síntese de Sistemas Digitais"

GIORJETY LICORINI DIAS

Orientador: Prof. Dr. Alexandre César Rodrigues da Silva

Co-orientador: Prof. Dr. Norian Marranghello

Dissertação apresentada à Faculdade de Engenharia - UNESP – Campus de Ilha Solteira, para obtenção do título de Mestre em Engenharia Elétrica.
Área de Conhecimento: Automação.

Ilha Solteira – SP

Fevereiro/2007

FICHA CATALOGRÁFICA

Elaborada pela Seção Técnica de Aquisição e Tratamento da Informação
Serviço Técnico de Biblioteca e Documentação da UNESP - Ilha Solteira.

Dias, Giorjety Licorini
D541f Ferramentas para a integração de redes de Petri e VHDL na síntese de sistemas digitais / Giorjety Licorini Dias. -- Ilha Solteira : [s.n.], 2007
180 p. : il.

Dissertação (mestrado) - Universidade Estadual Paulista. Faculdade de Engenharia de Ilha Solteira, 2007

Orientador: Alexandre César Rodrigues da Silva
Co-orientador: Norian Marranghello
Bibliografia: p. 156-159

1. Máquina de estados finitos. 2. Síntese de sistemas digitais. 3. Ferramentas de conversão. 4. Redes de Petri. 5. VHDL (Linguagem descritiva de hardware).

UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
FACULDADE DE ENGENHARIA DE ILHA SOLTEIRA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

GIORJETY LICORINI DIAS

FERRAMENTAS PARA A INTEGRAÇÃO DE REDES DE PETRI E VHDL NA SÍNTESE DE SISTEMAS DIGITAIS.

_____
Prof. Dr. Aledir Silveira Pereira

_____
Prof. Dr. Marius Strum

_____
Dr. Norian Marranghello
(Co- orientador)

Dissertação apresentada à Faculdade de Engenharia - UNESP – Campus de Ilha Solteira, para obtenção do título de Mestre em Engenharia Elétrica.
Área de Conhecimento: Automação.

ILHA SOLTEIRA – SP
Fevereiro de 2007.

DEDICATÓRIA

Dedico este trabalho aos meus pais Genésio e Amélia, ao meu irmão Geverson.

AGRADECIMENTOS

Agradeço principalmente a Deus, por ter me dado força e proteção nos momentos difíceis, sabedoria para escolher os caminhos corretos e também pelos momentos de alegria que tive durante esta caminhada.

Com atenção especial, aos meus orientadores Alexandre César Rodrigues da Silva e Norian Marranghelo, pela confiança, dedicação e profissionalismo apresentados durante os três anos em que trabalhamos juntos. Por ter me oferecido a oportunidade de crescer pessoal e profissionalmente.

Especialmente a minha família por ter acreditado e depositado total confiança em mim. Agradeço a eles por tudo que fizeram por mim, principalmente por estarem sempre ao meu lado oferecendo amor, carinho e compreensão.

Aos colegas Ricardo Vieira, Henrique Dezani, Hélio Clementino, Thiago Prado e Tércio Filho.

Com um carinho especial a Lenildo Luis Melo que foi quem mais esteve ao meu lado durante a etapa final de conclusão deste trabalho. Seu apoio e compreensão foram muito importantes para mim.

Aos funcionários da UNESP pelo carinho e profissionalismo apresentados durante os anos de mestrado. Em especial a Vanessa, Antônio e Edílson.

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico – CNPq.

Por fim agradeço Roberto Almeida Gabriel e Eduardo Yamamoto Barbosa, pela confiança, dedicação, compreensão e carinho oferecidos. Por ter sido as pessoas que incentivaram o início deste trabalho.

RESUMO

Neste trabalho apresentam-se quatro ferramentas de síntese digital, capazes de converter máquinas de estados finitos modeladas em rede de Petri para uma descrição VHDL correspondente à maquina modelada. As máquinas de estados finitos nos modelos de Mealy ou Moore são representadas em rede de Petri Lugar/Transição através de duas metodologias de modelagem desenvolvidas. Uma das metodologias modela apenas máquinas do tipo Mealy, enquanto que a outra modela máquinas de Mealy e Moore. As metodologias e o tipo de tradução da rede de Petri que se deseja obter são fatores essenciais para definir as ferramentas que serão utilizadas. Duas das ferramentas desenvolvidas traduzem o modelo da rede de Petri em uma tabela de transição de estados e as outras duas ferramentas traduzem o modelo da rede de Petri em uma descrição comportamental na linguagem VHDL. Dependendo da ferramenta utilizada é necessário integrar outras ferramentas de síntese, desenvolvidas em trabalhos anteriores, no processo de tradução da rede de Petri para VHDL. A aplicabilidade das ferramentas e metodologias desenvolvidas foi concluída através de simulações dos códigos VHDL obtidos.

Palavras Chave: Máquina de estados finitos. síntese de sistemas digitais. ferramentas de conversão. redes de Petri. VHDL

ABSTRACT

In this work we present four digital synthesis tools capable of converting finite state machines modeled in Petri nets into a corresponding VHDL description. Mealy or Moore finite state machine models are represented in Place/Transition Petri nets through two possible methodologies, developed during this work. With one of the methodologies only Mealy machines can be modeled, while the with other both Mealy and Moore type machines can be dealt with. The methodologies and the kind of Petri net translation one desires to obtain are essential factors to determine the tools to be used. Two among the tools we developed translate a Petri net description into a state transition table, while the other two translate the Petri net description into a VHDL behavioral one. Depending on which of them is used it is also necessary to use some other synthesis tools developed by members of our research group. The adequacy of the developed methodologies an tools to the synthesis process has been verified through the simulation of the VHDL codes generated by our tools.

Keywords: Finite State Machine. synthesis digital systems. convertion tools. Petri net. VHDL.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

1.1 Motivação

O diagrama de estados é uma das metodologias utilizadas para a descrição do comportamento das Máquinas de Estados Finitos (FSM do inglês *Finite State Machines*), ele permite escrever de maneira eficiente máquinas síncronas ou assíncronas especificadas no modelo de Mealy ou no de Moore [1]. As FSM são intensivamente utilizadas no projeto de controladores de sistemas digitais. Porém, devido à complexidade dos sistemas, e por apresentarem cada vez mais atividades paralelas, o diagrama de estados tornou-se inadequado. Assim, metodologias mais recentes utilizam descrições situadas em um alto nível de abstração, como as redes de Petri (RdP), que possibilitam a verificação, validação e a implementação da FSM.

Entre os paradigmas de modelagem, a RdP permite fácil especificação de co-operação dos sub-sistemas e o uso de métodos de validação formal, que são baseados na teoria matemática. Estes métodos devem ser aplicados antes da fase de implementação e em paralelo com a fase de simulação, possibilitando uma minimização dos erros de projeto [2].

Vários tipos de RdP podem ser sugeridos para especificar e modelar sistemas digitais, tanto por impor restrições para o modelo básico quanto por acrescentar extensões para ela [2], [3], [4]. Entretanto, somente o seu uso não é o suficiente para o projeto e implementação de um sistema. Faz-se necessário a integração de outra ferramenta que possibilite a simulação e síntese do sistema, mais especificamente a utilização de linguagens de descrição de *hardware* (HDLs).

Escolheu-se utilizar neste trabalho a linguagem de descrição de *hardware* VHDL, por ser uma linguagem padronizada e bem conhecida na comunidade de projetistas de

sistemas digitais o que evitaria a resistência às mudanças. Essa linguagem está disponibilizada na grande maioria dos ambientes comerciais de síntese. Dessa forma as descrições resultantes geradas nesta linguagem, poderão ser facilmente sintetizadas em diferentes tecnologias alvo e o desempenho de cada uma delas será comparado.

O interesse em se utilizar RdP deve-se, principalmente às suas características de modelagem. Pesquisas realizadas comprovam a eficiência do uso desta ferramenta na modelagem, especificação e simulação de sistemas paralelos, sendo uma ferramenta capaz de apoiar o desenvolvimento de sistemas heterogêneos, constituídos pela integração de *hardware* e *software* [5], [6], [7].

Assim, a integração das etapas de projeto de *hardware* e *software*, técnica conhecida como *codesign*, é cada vez mais necessária no desenvolvimento de sistemas digitais. Esta técnica, atualmente está sendo muito difundida e aplicada, pois o mercado de produtos computacionais almeja a redução de custos e tempo de projeto, o que leva os projetistas a deslocar maior parte da funcionalidade dos sistemas embarcados para o *software*, deixando os elementos de *hardware* dedicados apenas às funcionalidades que necessitam de alto desempenho [8].

Contudo, as RdP são vistas como uma extensão natural da FSM, proporcionando assim uma migração muito fácil de uma especificação em FSM para RdP [2].

1.2 Objetivo do Trabalho

Este trabalho teve como objetivo desenvolver metodologias de modelagem que especificassem FSM nos modelos de Mealy ou Moore em RdP, assim desenvolveram-se duas metodologias. Uma modela apenas máquinas do tipo Mealy, a qual denominou-se 4M, a outra modela máquinas de Mealy e Moore, a esta denominou-se 5M.

Para testar a viabilidade das metodologias desenvolveram-se ferramentas de síntese digital, que convertem as FSM modeladas em RdP, ou seja, em alto nível de abstração para um nível inferior, mas especificamente em linguagem de descrição de *hardware*. Desta forma desenvolveram-se quatro ferramentas que utilizam as descrições da RdP obtidas pelas metodologias de modelagem 4M e 5M, afim de gerar a descrição VHDL correspondente a FSM modelada.

As descrições utilizadas como entrada pelas ferramentas, apresenta uma linguagem padrão para RdP, a PNML (*Petri Net Markup Language*) [9]. Por esta razão utilizou-se o software PIPE (*Platform Independent Petri net Editor*) para modelar as RdP.

As ferramentas, denominadas PIPE2VHDL4M e PIPE2VHDL5M geram a descrição comportamental na linguagem VHDL para o sistema modelado em RdP e as PIPE2TAB4M e PIPE2TAB5M determinam a tabela de transição de estados, por intermédio desta tabela é possível obter a descrição RTL do sistema na linguagem VHDL.

Na figura 1.1 apresenta-se como as ferramentas desenvolvidas se inserem no contexto da síntese de sistemas digitais, através do diagrama Y de Gajski and Kuhn [10]. Observa-se que a seta na figura 1.1 indica os níveis de abstração em que os programas desenvolvidos atuam. As metodologias encontram-se no mais alto nível do diagrama.

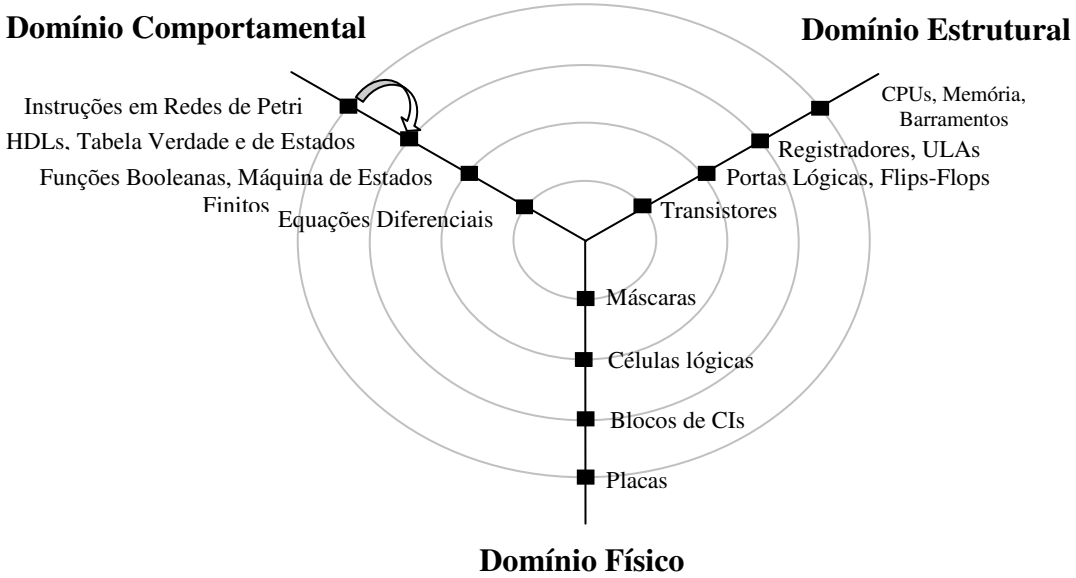


Figura 1.1: Ferramentas desenvolvidas vista como uma transição no Diagrama Y.

Desta forma, tem-se que as ferramentas desenvolvidas, tomam como entrada a descrição da RdP e a converte para um código VHDL do sistema modelado. A simulação dos códigos VHDL gerados pelas ferramentas, nos ambientes Max+PlusII ou QuatusII, validam as metodologias desenvolvidas.

1.3 Estado da Arte

Hady *et al.* [11] publicou o primeiro trabalho realizado com o propósito de integrar as linguagens RdP e VHDL. Neste os autores apresentam metodologias e ferramentas que permitem que um sistema seja analisado usando RdP. Diferentes metodologias e ferramentas são avaliadas para permitir a análise e verificação dos projetos interpretados em linguagem de descrição de *hardware* (HDL). Uma das metodologias apresentadas permite que o projetista crie modelos não interpretados em um ambiente já capaz de interpretar a modelagem em VHDL.

No trabalho "geração de VHDL a partir da especificação de controladores paralelos em redes de Petri hierárquicas" [12] são relatadas as vantagens da utilização de RdP, relativamente a outros paradigmas de modelagem na especificação de controladores que tenham comportamento paralelo. Os autores também propõem alterações no comportamento das RdP, de forma a obter uma modelagem eficiente de controladores. Foi desenvolvido um compilador que gera o código VHDL, a partir da especificação de um controlador baseado em RdP. O compilador foi desenvolvido para gerar o código em determinado subconjunto VHDL, que permite a simulação e a síntese no pacote Alliance (domínio público) [13].

Uma metodologia para a especificação de sistemas digitais, baseada em RdP orientadas a objetos é apresentada por Machado *et al.* [14]. Neste trabalho é proposto um

exemplo de sistema digital que foi especificado no modelo RdP-shobi e a partir do qual o código VHDL é gerado automaticamente.

Um dos trabalhos que propõe uma técnica para a síntese de sistemas, com especificações comportamentais escritas em VHDL é “síntese de sistemas assíncronos baseada no mapeamento direto usando-se VHDL e redes de Petri” [15]. Neste trabalho os circuitos assíncronos independentes de velocidade são construídos usando células de David. Esta técnica combina as vantagens da síntese lógica e das técnicas de tradução. As RdP coloridas e as RdP etiquetadas são usadas como formatos intermediários para controlar a representação dos circuitos assíncronos independentes de velocidade.

O trabalho intitulado “modelagem em redes de Petri de um controlador de aprendizagem adaptável aplicado em uma cadeira de rodas elétrica” [16] apresenta uma metodologia de *codesign* baseada na utilização de RdP, a fim de obter soluções otimizadas para o *hardware/software*. Um exemplo da aplicação é dado para uma alavanca de controle neural de uma cadeira de rodas elétrica.

2 REDES DE PETRI NA MODELAGEM E ESPECIFICAÇÃO DE SISTEMAS

As redes de Petri (ou simplesmente RdP) atualmente tem-se mostrado linguagens poderosas para especificar e modelar o comportamento de sistemas paralelos, em particular sistemas digitais.

Os conceitos apresentados nesta seção são relevantes para o entendimento do funcionamento desta linguagem e necessário para o desenvolvimento das metodologias de modelagem apresentadas neste trabalho, que possibilitam que FSM sejam modeladas em RdP Lugar/Transição. Sendo assim, apresentam-se neste capítulo definições, estutura, funcionamento, principais propriedades e classificações pertinentes as RdP.

2.1 Definições

A RdP é uma linguagem gráfica e matemática, desenvolvida em 1962 por Carl Adam Petri em sua tese de doutorado intitulada “Comunicação com autômatos”, defendida na Universidade de Darmstadt, na Alemanha. Essa linguagem tem-se mostrado bastante adequada para a modelagem de sistemas que exibem atividades assíncronas ou concorrentes [17]. As áreas de aplicação têm sido diversas, das quais se salientam: modelagem de protocolos de comunicação; gerenciamento de base de dados; sistemas de controle industrial; sistemas computacionais de multiprocessamento; entre outras.

Pode-se apresentar a RdP como um modelo formal, de três maneiras diferentes [18]:

a. um grafo com dois tipos de nós e comportamento dinâmico;

b. um conjunto de matrizes de inteiros positivos ou nulos, cujo comportamento dinâmico é descrito por um sistema linear e

c. estruturalmente, um sistema de regras baseado na representação do conhecimento, sob a forma condição→ação.

O grafo de uma RdP possui como elementos básicos: lugares, transições, arcos e marcas (*tokens* ou senhas). Na figura 2.1 tem-se o modelo gráfico de uma RdP, onde são explicitados os elementos básicos que constituem a rede.

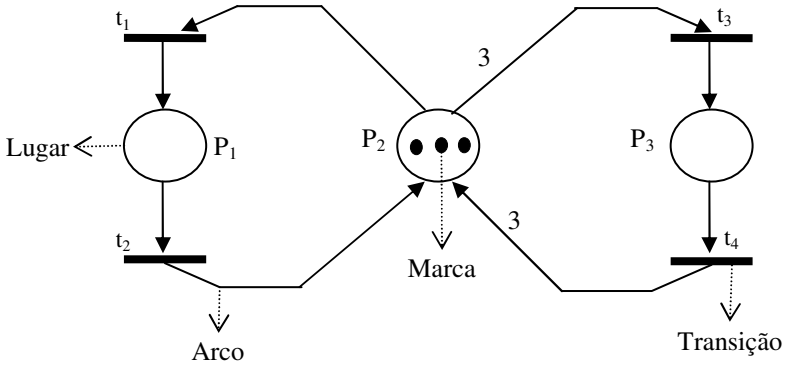


Figura 2.1: Representação de uma Rede de Petri [18].

Um lugar (representado por círculo) modela condição ou estado de um agente (componente de hardware ou software) que corresponde às condições que devem ser certificadas para os eventos acontecerem. Este componente é denominado passivo e corresponde às variáveis de estado.

Uma transição (representado por barra ou retângulo) modela um evento ou ação, que determinam as mudanças de estado do sistema. As transições são componentes ativos.

O disparo de uma transição está associado às condições dos seus lugares de entrada (pré-condições) e de saída (pós-condições).

Os arcos são elementos que interligam lugares a transições ou vice-versa, nunca lugar a lugar ou transição a transição, encadeando assim condições e eventos. Os arcos também possuem pesos, estes representam a quantidade de marcas que serão inseridas ou retiradas dos lugares que o arco conecta.

As marcas são representadas graficamente por um ponto no lugar. Ela simula um recurso disponível ao agente, o posicionamento dessas marcas em alguns lugares do grafo, constitui a marcação. A evolução da marcação permite modelar o comportamento dinâmico do sistema.

A partir de todas as definições dadas suponha-se que a figura 2.1 representa um programa que executa apenas duas tarefas, leitura e escrita de arquivos. Ressalta-se que apenas uma tarefa é realizada por vez, ou seja, enquanto o programa efetua a leitura fica impedido de fazer a escrita.

Assim tem-se no modelo da figura 2.1, P_1 representando a tarefa de leitura, P_3 a escrita, P_2 o estado do programa (livre ou não para executar determinada atividade), "t_1" o início da atividade leitura, "t_2" fim da leitura, "t_3" início da atividade de escrita e "t_4" fim da escrita. Se houver três marcas em P_2 o programa estará livre para executar qualquer atividade, caso contrário ele já estará realizando alguma tarefa. Suponhamos que a transição t_1 dispare assim uma marca será retirada do Lugar P_2 e inserida no lugar P_1, isto significa que o programa está efetuando a leitura de um arquivo. Caso t_2 dispare, uma marca será retirada de P_1 e inserida em P_2, assim este estará livre para realizar outra atividade. Se t_3 disparar, três marcas serão retiradas de P_2 e apenas uma será inserida em P_3, desta forma o programa estará realizando a escrita em determinado arquivo. Quando t_4 disparar uma marca será retirada de P_3 e três serão inseridas em P_2 e o programa estará livre para efetuar qualquer uma das tarefas.

É importante ressaltar que não existe uma seqüência única para a construção de um modelo em RdP. No entanto, uma metodologia comumente adotada para a modelagem de um sistema produtivo através de uma representação em RdP pode ser [19]:

a. Identificação dos recursos, operações ou atividades no sistema;

b. Estabelecimento de seqüências das atividades em cada processo;

c. Representação das atividades, geralmente, por lugares que são conectados às transições, as quais indicam o início e término destas atividades;

d. Especificações da marcação inicial (estado inicial do sistema);

e. Definição dos recursos através de lugares e conexão destes às transições de início e término de cada operação.

Antes de elaborar um modelo em RdP é necessário conhecer a estrutura e o funcionamento de uma RdP. Na subseção seguinte apresenta-se a estrutura de uma RdP.

2.2 Estrutura de uma Rede de Petri

O modo gráfico é uma das alternativas utilizadas para representar uma RdP, o que pode ser feito por meio de um grafo bipartido, também é interessante representá-la pela sua estrutura e dinâmica, de forma analítica. Estas representações permitem a utilização de métodos formais de análise que proporcionam a determinação de propriedades do modelo da RdP [20].

As RdP são compostas por quatro conjuntos: um conjunto de lugares P, um conjunto de transições T, uma aplicação de entrada I ou Pré, e uma aplicação de saída O ou Pós.

As ligações entre lugares e transições são especificadas através de duas funções: a função de entrada I ou de incidência direta (Pré) e a função de saída O ou de incidência reversa (Pós) [21].

Enquanto a função de entrada I especifica, para todas as transições t_j, os seus conjuntos de lugares de entrada, a função de saída O define os seus conjuntos de lugares de saída. Uma RdP diz-se ordinária se as funções de incidência só podem tomar os valores 1 ou 0; diz-se generalizada quando as funções de incidência podem tomar valores positivos ou nulos [20]. Essas funções são representadas em forma de matrizes. Assim nesta seção são apresentadas definições pertinentes as matrizes de Incidência Direta e Reversa.

2.2.1 Matriz de Incidência Direta

Esta matriz apresenta a estrutura da rede quanto às informações dos lugares de entrada, isto é aqueles que incidem sobre cada transição específica, e o respectivo peso do arco de entrada da transição em questão. É formada por NL (quantidade de lugares) linhas e NT (quantidade de transições) colunas.

Os elementos da linha "i" e coluna "j" (i =1, 2,..., NL; j =1, 2,..., NT) denotado por A_{ij} contém as seguintes informações:

a. Se $A_{ij} = 0$, o lugar P_i não é lugar de entrada da transição t_j;

b. Se $A_{ij} \neq 0$, o lugar P_i é um lugar de entrada da transição t_j com $p_e = A_{ij}$, onde p_e é o peso de entrada, do arco que liga o lugar a transição em questão. Na figura 2.2 é apresentada a matriz de incidência direta, ou de pré-condições, para a RdP da figura 2.1.

$$\text{Pr } \acute{e} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{matrix} P_1 \\ P_2 \\ P_3 \end{matrix}$$

Figura 2.2: Matrizes de incidência direta (Pré) da rede de Petri da Figura 2.1

Observa-se na figura 2.2 que P_2 é o lugar de entrada das transições t_1 e t_3, com p_e=3 para o elemento A_{23}.

2.2.2 Matriz de Incidência Reversa

Esta matriz apresenta a estrutura da rede quanto às informações dos lugares de saída de cada transição específica e o respectivo peso do arco de saída da transição em questão. É formada por NL linhas e NT colunas.

Os elementos da linha i e coluna j (i =1, 2,..., NL; j =1, 2,..., NT) denotado por $B_{ij,}$ contém as seguintes informações:

a. Se $B_{ij} = 0$, o lugar P_i não é lugar de saída da transição t_j;

b. Se $B_{ij} \neq 0$, o lugar P_i é um lugar de saída da transição t_j com peso $p_s = B_{ij}$, onde p_s é o peso de saída, do arco que liga a transição ao lugar.

A matriz de incidência reversa, ou de pós-condições, da RdP da figura 2.1 é exibida na figura 2.3.

$$Pos = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{matrix} P_1 \\ P_2 \\ P_3 \end{matrix}$$

Figura 2.3: Matrizes de incidência reversa (Pos) da rede de Petri da Figura 2.1.

Nota-se na figura 2.1 que P_2 é o lugar de saída das transições t_2 e t_4, com p_s=3 para o elemento B_{24}.

Através das duas matrizes apresentas nas figuras 2.2 e 2.3 é possível encontrar a matriz de incidência (W), onde: W= Pos –Pré.

Na figura 2.4 é mostrada a matriz de incidência W.

$$W = \begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 1 & -3 & 3 \\ 0 & 0 & 1 & -1 \end{bmatrix}$$

Figura 2.4: Matrizes de incidência W da rede de Petri da Figura 21.

2.3 Funcionamento de uma Rede de Petri

Nesta subseção são apresentadas as regras indispensáveis para o funcionamento de uma RdP. As regras estão divididas nos seguintes tópicos: marcação de uma RdP, sensibilização de uma transição, disparo da transição e seqüência de disparos.

2.3.1 Marcação para a Rede de Petri

A marcação de uma RdP indica o número de marcas que cada lugar contém, uma marcação é denotada por um vetor M, de dimensão n (número total de lugares). O p-ésimo componente de M (denotado por $M(P_i)$), é o número de marcas no lugar P_i. Formalmente, a

marcação pode ser definida como um vetor M = ($M(P_1)$, ..., $M(P_n)$). A marcação inicial (M_0) corresponde ao estado inicial do sistema, a partir da qual a rede será analisada. Para a figura 2.1, M_0 = (0,3,0), corresponde que os lugares P_1 e P_3 não possuem marcas, enquanto P_2 tem 3 marcas. A presença de marcas em um lugar pode ser interpretada como a presença de um recurso de um determinado tipo.

Marcação decorrente é qualquer marcação acessível após um ou mais disparos, partindo de uma dada marcação inicial M_0.

2.3.2 Sensibilização de uma Transição

Uma transição t é sensibilizada por uma marcação M, se e somente se [22]:

$$\forall pi \in P, M(p_i) \geq \text{Pr}\acute{e}(p_i, t_j)$$

Por exemplo, na RdP da figura 2.1 as transições t_1 e t_3 estão sensibilizadas, pela marcação inicial $M_0 = [0\ 3\ 0]^T$, pois:

Pré $(p_i, t_1) = [0\ 1\ 0]^T$, Pré $(p_i, t_3) = [0\ 3\ 0]^T$ e $M(p_i) = [0\ 3\ 0]^T$ onde i = 1, 2,3.

Portanto:

$$M(p_i) > \text{Pré}\ (p_i, t_1) \text{ e } M(p_i) = \text{Pré}\ (p_i, t_3)$$

2.3.3 Disparo de uma Transição

O comportamento dinâmico de cada sistema modelado com RdP pode ser descrito pelas mudanças nos seus estados. A simulação do comportamento dinâmico acontece com o disparo das transições, onde as marcações da RdP correspondente são modificadas.

Em uma RdP, $t_i \in T$ só pode disparar se estiver sensibilizada. Disparando uma transição sensibilizada t_i, uma nova marcação M' é obtida, tal que [23]:

$$M'(p_i) = M_0(p_i) - Pré\ (p_i, t_j) + Pós\ (t_j, p),\ \forall p \in P$$

Dada a RdP da figura 2.1, após o disparo da transição t_1 sensibilizada, a partir da marcação inicial M_0 obtém-se a seguinte marcação:

$$M'(pi) = M_0(p_i) - Pré\ (p_i,t_1) + Pós\ (p_i,t_1) = [0\ 3\ 0]^T - [0\ 1\ 0]^T + [1\ 0\ 0]^T = [1\ 2\ 0]^T$$

Para o disparo da transição t_3 sensibilizada, obtém-se a seguinte marcação:

$$M'(pi) = M_0(p_i) - Pré\ (p_i,t_3) + Pós\ (p_i,t_3) = [0\ 3\ 0]^T - [0\ 3\ 0]^T + [0\ 0\ 1]^T = [0\ 0\ 1]^T$$

2.3.4 Seqüência de Disparo

Uma seqüência de transições "s" que devem disparar para atingir uma marcação M' partindo de M é denominada de seqüência de disparo.

Se $M_0 \xrightarrow{t_1} M_1$ e $M_1 \xrightarrow{t_2} M_2$, diz-se que a seqüência $s = t_1\ t_2$ é disparável a partir de M_0 com a seguinte notação: $M_0 \xrightarrow{t_1 t_2} M_2$.

Com uma seqüência "s" é associado um vetor característico $S = t_{jk}$, onde o j indica quais as transições disparadas e o k o número de vezes que ela disparou. Sua dimensão é igual ao número de transições da rede.

Um exemplo de seqüência de disparo pode ser dado pela RdP da figura 2.1, para alcançar a marcação $M = (0, 0, 1) = [0, 0, 1]^T$ partindo de $M = (0, 3, 0)$, deve-se disparar a seqüência $s = t_1\ t_2\ t_1\ t_2\ t_3 = t_{12}\ t_{22}\ t_{31}$ cujo vetor característico é $S = (t_1, t_2, t_3, t_4)\ (2, 2, 1, 0) = [2\ 2\ 1\ 0]^T$, a seqüência de disparos na forma matricial é dada por:

$$\begin{bmatrix} 0 \\ 3 \\ 0 \end{bmatrix} \xrightarrow{t_1} \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix} \xrightarrow{t_2} \begin{bmatrix} 0 \\ 3 \\ 0 \end{bmatrix} \xrightarrow{t_1} \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix} \xrightarrow{t_2} \begin{bmatrix} 0 \\ 3 \\ 0 \end{bmatrix} \xrightarrow{t_3} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Um vetor característico pode corresponder a diversas seqüências de disparo, por exemplo: (2, 1, 0, 0) corresponde a ambas as seqüências $t_1\ t_2\ t_1$ e $t_1\ t_1\ t_2$, observando-se assim

uma perda de informação na evolução da rede, pois em S não está representada a ordem de disparo das transições. A seqüência $t_2t_1t_1$ não é uma seqüência de disparo do vetor característico, pois t_2 não é habilitada a partir da marcação inicial, atribuída na RdP mostrada na figura 2.1.

Observa-se também que da execução a RdP resulta dois tipos de seqüências:

a. Seqüência de disparo(s);

b. Seqüência de marcações (M_0, M_1, M_2, ...).

A marcação de uma RdP é essencial para analisar suas propriedades, na subseção 2.3 são apresentadas as principais propriedades do seu modelo.

2.4 Principais Propriedades de uma RdP

As propriedades de uma RdP podem ser divididas em dois grupos [19]:

a. Propriedades comportamentais, que dependem da marcação inicial;

b. Propriedades estruturais, que dependem apenas da topologia da RdP.

A seguir, faz-se uma apresentação breve de um conjunto de propriedades comportamentais e estruturais relevantes. Visto que o conhecimento dessas propriedades da RdP reveste-se de grande importância na medida que permite a análise de várias características e problemas associados aos sistema modelados por ela.

2.4.1 Propriedades Comportamentais

As propriedades dependentes de um estado inicial e que estão ligadas à evolução da rede são mais comumente chamadas de comportamentais. Sua verificação se faz geralmente pela construção do grafo de marcações acessíveis [23].

As principais propriedades comportamentais das RdP que possibilitam a análise do sistema modelado são: vivacidade, reinicialização, alcançabilidade, limitação e persistência.

a) Vivacidade

Uma RdP é dita viva para uma marcação inicial M_0 se, para qualquer marcação decorrente M_n, existir uma seqüência de disparos "s" que torne disparável qualquer transição da rede, ou quase viva se for somente disparável por uma vez. Assim uma RdP marcada é viva se e somente se todas as suas transições são vivas.

O conceito de vivacidade está relacionado com a total ausência de *deadlock* (bloqueios) na operação do sistema.

b) Reinicialização ou Reversibilidade

Uma RdP é reiniciável para uma dada marcação inicial M_0 se, para qualquer marcação decorrente M_n, existir uma seqüência de disparo "s" que faça a rede voltar à marcação inicial.

Considerando a RdP da figura 2.5 cujo grafo de marcações é dado pela figura 2.6 ela não é reinicializável, pois não existe nenhuma seqüência que permita voltar à marcação inicial $M_0 = P_1\ P_4 = (1,0,0,1)$ após o disparo da transição t_1.

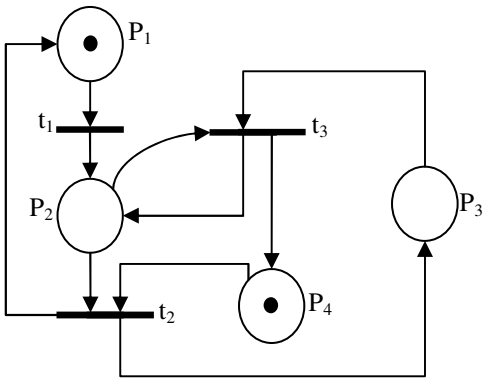


Figura 2.5: Exemplo de uma Rede de Petri não reinicializável [23].

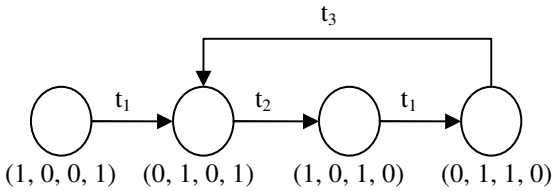


Figura 2.6: Grafo de Marcações Acessíveis da rede não reinicializável [23].

Portanto, se considerar para a rede da figura 2.5 a marcação $M_0 = P_2\ P_4 = (0, 1, 0, 1)$, ela é ao mesmo tempo viva e reinicializável. Assim, conclui-se que a reinicialização depende da marcação inicial e da estrutura da rede.

c) Alcançabilidade

A alcançabilidade é fundamental e básica para estudar as propriedades dinâmicas de qualquer sistema. Indica a possibilidade de atingirmos uma determinada marcação pelo disparo de um número finito de transições, a partir de uma marcação inicial. O disparo de uma transição habilitada mudará a distribuição das marcas (marcação) na rede de acordo com as regras de disparo de uma transição, descritas anteriormente.

Na RdP denomina-se de alcançabilidade de uma marcação M (representada por A(R;M)) o multi conjunto de todas as marcações geradas a partir de M. Onde A(R, M) é igual ao grafo de marcações.

Assim, uma marcação M_n é dita ser alcançável a partir de uma marcação M_0, se existir uma seqüência de disparos que transforma M_0 em M_n. Uma seqüência de disparos pode ser denotada por $s = M_0\ t_1\ M_1\ t_2\ M_2 \ldots t_n\ M_n$, ou simplesmente $s = t_1\ t_2\ t_3 \ldots$. Neste caso M_n é alcançável desde M_0 por "s", podendo-se escrever então: $M_0\ [s > M_n$ [23].

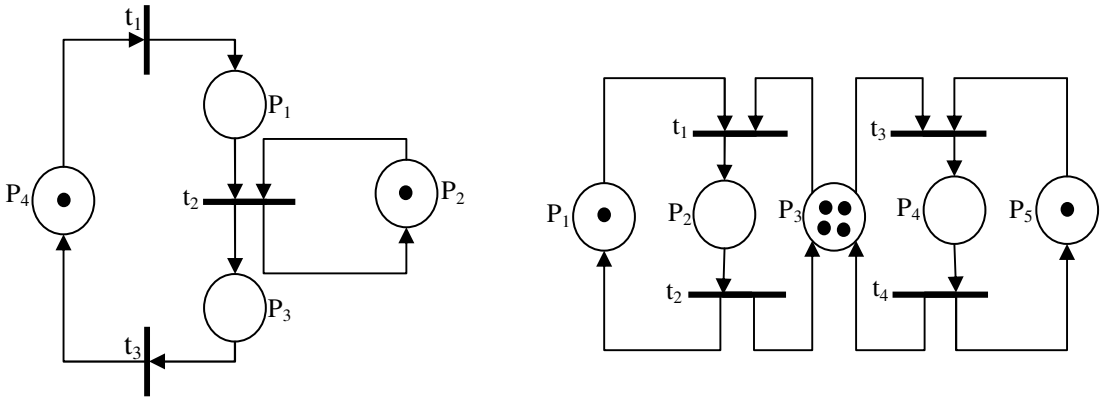
d) Limitação

Em uma RdP limitada, o número de marcações é finito. Entretanto uma RdP não limitada possui infinitas marcações, significando que o sistema físico correspondente é impossível de ser implementado.

Assim uma RdP é dita ser k-limitada ou simplesmente limitada se o número de marcas em cada lugar não excede um número finito para qualquer marcação alcançável desde M_0.

Se k = 1 diz-se que o lugar é seguro, salvo ou binário. Por exemplo, na rede da figura 2.7(a), cada marcação M', a qual pode ser alcançável desde M_0, tem no máximo uma marca em $P_{i,}$ sendo esta uma rede segura [23].

Uma rede marcada é k-limitada se e somente se todos os seus lugares são k limitados. Apresenta-se na figura 2.7 (b) uma rede K-limitada, com k = 4.



a) limitada com k = 1 b) limitada com k = 4

Figura 2.7: Rede de Petri marcada e k-limitada [23].

Uma rede marcada é segura se e somente se todos os seus lugares são seguros.

e) Persistência

Uma RdP é chamada persistente se para quaisquer duas transições habilitadas, o disparo de uma não desabilita a outra, isto é, uma transição após habilitada permanece neste estado até disparar. Esta propriedade é importante no âmbito de circuitos assíncronos independentes de velocidade [24]. As redes que possuem essa propriedade estão livres de conflitos, como os grafos marcados [25]. Todos os grafos marcados são persistentes embora nem todas as redes persistentes sejam grafos marcados [25].

É importante salientar que as propriedades que foram definidas são fortemente ligadas à marcação inicial.

2.4.2 Propriedades Estruturais

As propriedades estruturais são aquelas que dependem das características topológicas das RdP e são independentes da marcação inicial M_0. Estas propriedades podem ser, muitas vezes, caracterizadas pela matriz de incidência da RdP e pelas equações ou desigualdades que lhe estão associadas e são cruciais na análise das RdP, já que tornam possível a investigação da estrutura de uma RdP independentemente do comportamento [18].

a) Componentes conservativos, invariantes de lugar

Os invariantes de lugar (*place invariants*) são conjuntos de lugares, cuja soma das marcas neles contidas é constante para todas as marcações possíveis. São representados por vetores inteiros de dimensão n, onde n é o número de lugares da rede. Os elementos não nulos correspondem aos lugares que pertencem a um determinado invariante.

Como exemplo desta propriedade considera-se a RdP apresentada na figura 2.8, formada pelos lugares P_1, P_2, P_3, P_4 e P_5 e pelas transições t_1, t_2, t_3 e t_4.

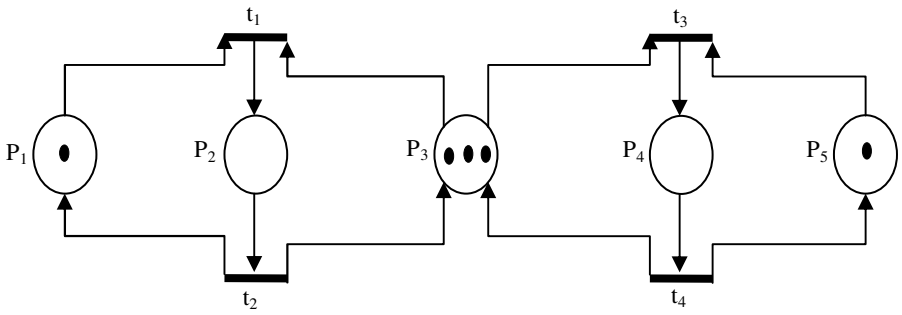


Figura 2.8: Rede de Petri com invariante de lugar [18].

Observa-se que no circuito formado apenas pelos lugares P_1 e P_2 e pelas transições t_1 e t_2, a soma $M(P_1) + M(P_2)$ vale 1 para a marcação inicial $M_0 = [\ 1\ 0\ 3\ 0\ 1\ ]^T$. O disparo de t_1 não modifica em nada esta soma, da mesma forma que o de t_2, embora a marcação de cada lugar seja modificada a cada disparo de transição. O disparo das transições t_3 e t_4 também não modificam esta soma. Para este exemplo, pode-se verificar que, para todas as marcações acessíveis a partir da marcação inicial, tem-se $M(P_1) + M(P_2) = 1$

A forma linear $M(P_1) + M(P_2) = M_0(P_1) + M_0(P_2)$ é chamada invariante linear de lugar, visto que a soma das marcas se conserva para estes lugares. O conjunto de lugares P_1 e P_2 formam um componente conservativo da rede. Assim, invariantes de lugar permitem, sem enumerar todas as marcações possíveis, obter-se informações importantes sobre a propriedade de limitabilidade de uma determinada RdP.

Se considerarmos outro circuito para a rede da figura 2.8, formado apenas pelos lugares P_2, P_3 e P_4 e pelas transições t_1, t_2, t_3 e t_4. Verifica-se que o conjunto P_2, P_3 e P_4 formam um conjunto conservativo, com o invariante de lugar, pois as marcações acessíveis após os disparos de qualquer uma das transições t_1, t_2, t_3 e t_4 não alteram a soma das marcas dos respectivos lugares.

b) Componentes repetitivos, invariantes de transição

Em dualidade com os invariantes de marcação estão os invariantes de transição (*transition invariants*).

As transições invariantes representam seqüências de disparos que reinicializam uma marcação. Eles realçam a componente cíclica de um processo.

Com o propósito de aprimorar o entendimento desta propriedade considera-se como exemplo a sub-rede da figura 2.8, composta pelas transições t_3 e t_4 juntamente com seus lugares de entrada e saída (P_3, P_4 e P_5). Deve-se observar que o disparo da seqüência $s = t_3\ t_4$ a partir da marcação inicial leva de volta à mesma marcação.

A referida seqüência $s = t_3\ t_4$ é um invariante de transição, já que o disparo desta seqüência não modifica a marcação da rede. O invariante de transição corresponde a uma seqüência cíclica de eventos que pode ser repetida indefinidamente. O conjunto das transições invariantes forma, por sua vez, um componente repetitivo estacionário da rede. A seqüência $s = t_1\ t_2$ da rede apresentada na figura 2.8 também é um invariante de transição.

2.5 Classificação das redes de Petri

Há várias maneiras possíveis de se classificar as redes de Petri, uma maneira muito utilizada, consiste em agrupá-las quanto ao seu grau de abstração. Neste caso, pode-se separa-las em RdP de baixo nível e de alto nível [25].

As RdP de baixo nível são aquelas cujo significado de suas marcas não são diferenciáveis a não ser pela estrutura da rede à qual estão associadas. Elas ainda podem ser subdivididas em Elementares e Lugar/Transição. As redes Elementares são extremamente restritivas do ponto de vista de modelagem, pois permitem a existência de apenas uma marca em cada elemento. As redes Lugar/Transição minimizam as restrições impostas pelas redes Elementares, fundamentalmente, permitindo a utilização de mais de uma marca em cada elemento da rede.

As redes de alto nível são aquelas cujas marcas incorporam alguma semântica, viabilizando sua diferenciação. Esta semântica pode ir desde a atribuição de valores ou cores às marcas, até a adoção de noções de tipos de dados abstratos, conferindo-lhes um grande poder de expressão. A principal característica das redes de alto nível [26], comparativamente às redes elementares, é que nas de alto nível as marcas apresentam condições puramente booleanas e na outra as marcas representam dados individualizados. Os dois principais modelos de redes de alto nível são as redes Predicado/Transição, que se baseiam na lógica de predicados de primeira ordem, e as redes Coloridas, as quais originalmente foram orientadas a representações algébrico-lineares [25].

Além destas categorias têm-se extensões, as quais podem ser aplicadas tanto em redes de baixo nível quanto de alto nível. As principais extensões visam a inclusão de hierarquias e de aspectos temporais às RdP.

2.5.1 Redes Elementares:

As redes de Petri elementares [27], [28] constituem a versão proposta para sintetizar as diversas variações que surgiram sobre o modelo proposto por Petri em 1962, conhecidas na literatura por redes Clássicas ou Condição/Evento, ainda conservam as características originais das RdP.

Uma rede Elementar é uma quádrupla $RE = (P, T, F, C_{in})$, onde se tem:

P = {p1, p2, ..., pm}é um conjunto finito de lugares,

T = {t1, t2, ..., tn} é um conjunto finito de transições,

$F \subseteq (PxT) \cup (TxP)$ é um conjunto de arcos (relação de fluxo),

C_{in} é um caso inicial, que representa a situação dinâmica inicial do sistema, imediatamente antes de ser acionado.

Portanto, a representação gráfica de uma rede Elementar consiste da notação gráfica da rede subliminal acrescida da marcação do caso inicial, por meio de marcas. A esta associação de uma estrutura topológica de rede com um conjunto de marcas denomina-se rede marcada.

2.5.2 Rede de Petri Lugar/Transição

O grafo da rede Lugar/Transição é como o de qualquer outra RdP, direcionado (arcos orientados) e bipartido (constituído por dois componentes, lugares e transição). Entretanto, este grafo permite que pesos sejam atribuídos nos arcos que conectam um lugar a uma transição ou vice-versa.

As principais características deste tipo de rede atribuem-se a quantidade de marcas que cada lugar pode possuir e a ponderação dos arcos. A quantidade de marcas está associada a uma capacidade k, onde k é o número máximo de marcas que cada lugar suportará. Os arcos

ponderados são representados pelos pesos de cada arco, os pesos indicam quantas marcas serão retiradas e (ou) inseridas no lugar com o disparo da transição à qual o arco é excedente. Desta forma, se o peso (valor inteiro positivo) de um arco for igual a Q, pode-se, interpretar que há um conjunto de Q arcos ligando dois componentes distintos. Atribuindo pesos aos arcos é possível aumentar o poder de modelagem das RdP.

Devido à inserção de pesos nos arcos algumas restrições devem ser consideradas para o disparo das transições associadas. Para que uma transição esteja habilitada a disparar, é necessário que o número de marcas em cada lugar de entrada dessa transição seja superior ou igual ao peso do arco que conecta o lugar à transição em questão. Se tal condição se verificar e a transição disparar, é subtraído de cada lugar de entrada, um número Q de marcas, idêntica ao peso do arco que liga o lugar a transição. Analogamente, a cada lugar de saída é adicionado um número de marcas igual ao peso do arco que conecta a transição ao lugar.

Para exemplificar uma RdP Lugar/Transição, modelou-se a fórmula do perímetro (soma dos lados) para um triângulo isósceles (possui a mesma medida para apenas dois lados). Assim um triângulo isóscele, com um dos lados iguais medindo “a” e o outro medindo “b”, tem seu perímetro dado por P = 2a + b. Aplicando-se a modelagem em RdP para a fórmula do perímetro (P), obtém-se o modelo apresentado na figura 2.9, que mostra quando uma transição está habilitada e a marcação resultante após o seu disparo.

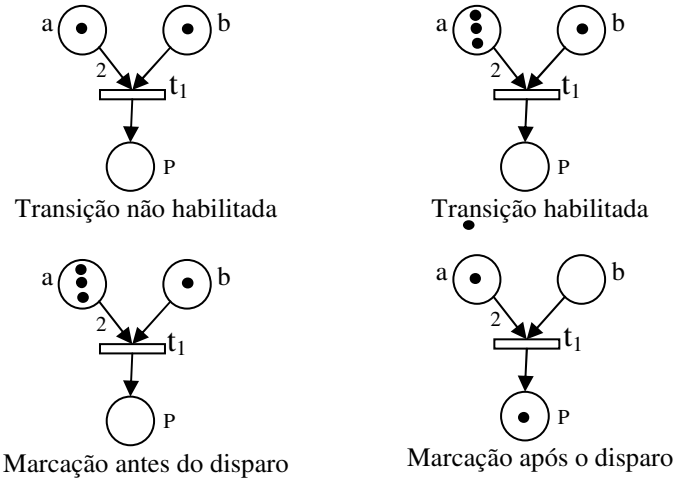


Figura 2.9: Representação do Disparo de uma Rede de Petri Lugar/Transição.

Observa-se na figura 2.9 que as medidas dos lados (a e b) e do perímetro (P) do triângulo são representadas por lugares, enquanto que a adição é representada pela transição t_1. Com o disparo da transição t_1 é obtida a medida do perímetro, visto que duas marcas são retiradas do lugar "a" e uma de "b", assim P=2a+b.

De acordo com os conceitos já apresentados, tem-se que uma RdP Lugar/Transição pode ser definida formalmente como uma quíntupla, RdP = (P,T,F,W,M_0) [22], onde:

P = {p1, p2, ..., pm}é um conjunto finito de lugares,

T = {t1, t2, ..., tn} é um conjunto finito de transições,

$F \subseteq (PxT) \cup (TxP)$ é um conjunto de arcos (relação de fluxo),

W: F {1, 2, 3, ...}é uma função de peso,

M_0: P→ {0, 1, 2, 3, ...}é a marcação inicial e

$P \cap T = \emptyset$ e $P \cup T \neq \emptyset$

Para um grande número de sistemas reais, é lícito pensar-se que cada lugar tenha um limite superior para o número de marcas que pode conter. Por exemplo, uma sala de aula tem um número limite de alunos, um estacionamento tem um número limite de carros. Entretanto nos exemplos dados e nas metodologias de modelagens desenvolvidas não se considerou a capacidade limite de cada lugar. Entretanto a RdP Lugar/Transição possibilita limitar a capacidade dos lugares, indicando o número máximo de marcas que cada lugar deverá possuir.

2.5.3 Redes de Petri Colorida

Nas RdP de baixo-nível existem apenas um tipo de marca, o que não permite a diferenciação de recursos em um lugar, sendo necessários lugares distintos para expressar recursos similares.

O objetivo das RdP Coloridas é promover a redução do tamanho do modelo, permitindo que marcas individualizadas (cores) representem diferentes recursos em uma mesma sub-rede.

Inicialmente, as marcas das redes Coloridas eram representadas por cores. Em trabalhos mais recentes, as marcas são representadas por estruturas de dados complexas, definidas como tipos de dados e é possível efetuar operações complexas sobre estes dados.

2.5.4 Extensões

Nesta subseção apresentam-se as extensões mais significativas das redes de Petri, quais sejam, as extensões hierárquicas e as temporizadas. Ambas extensões podem ser aplicadas tanto em redes de baixo nível quanto de alto nível.

a) RdP com Arcos Inibidores

Quando duas transições estão em conflito, a priorização é um problema comum em uma RdP. Para dar solução ao mesmo, aumentando assim o poder de modelagem das RdP foram criados os arcos inibidores.

Um arco inibidor é um arco dirigido que une um lugar P_i a uma transição t_j.O extremo final é marcado por um círculo pequeno como mostrado na figura 2.10. O arco inibidor entre P_2 e t_4 significa que a transição t_4 pode disparar se o lugar P_2 não contém nenhuma marca. O disparo de t_4 consiste em tomar uma marca de cada lugar de entrada de t_4, com exceção de P_2, e depositar uma marca em cada lugar de saída de t_4. As expressões teste zero e RdP estendidas são freqüentemente usadas na literatura [23], para se referir aos arcos inibidores.

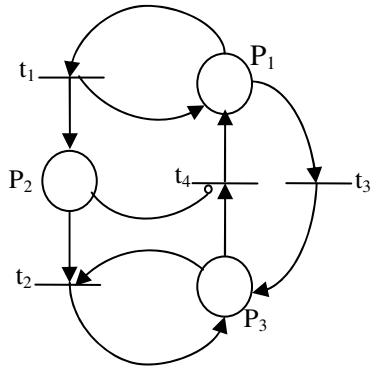


Figura 2.10: Rede de Petri com Arco Inibidor [23].

b) Redes de Petri Contínuas

A característica principal em relação às RdP Ordinárias é que a marcação de uma posição é um número real (positivo) e não mais um inteiro. Sendo o disparo de uma transição realizado como um fluxo contínuo. Estas redes representam sistemas que não podem ser modelados por RdP Ordinárias, obtendo um modelo muito apropriado quando o número de marcações da RdP Ordinária torna-se muito grande

d) Redes de Petri Sincronizadas.

Em uma RdP autônoma, sabe-se que uma transição pode ser disparada se ela é habilitada, mas não sabemos quando ela será disparada. No caso da RdP Sincronizada, um evento é associado a cada transição e o disparo desta transição acontecerá se a transição estiver habilitada e quando o evento associado ocorrer.

A figura 2.11, por exemplo, representa os estados de um motor. Esta é uma RdP Sincronizada porque os disparos das transições são sincronizados sobre eventos externos (o evento externo corresponde a uma mudança no estado do mundo externo).

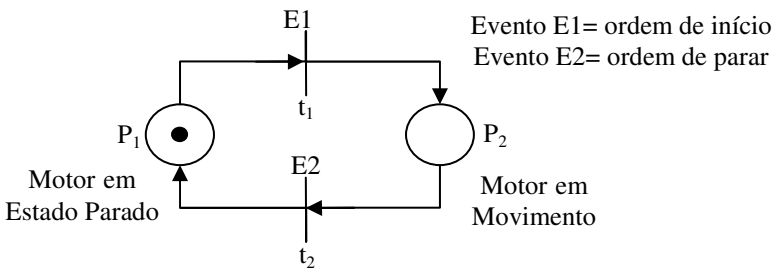
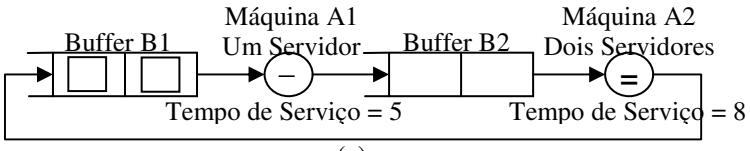


Figura 2.11: Rede de Petri Sincronizada [23].

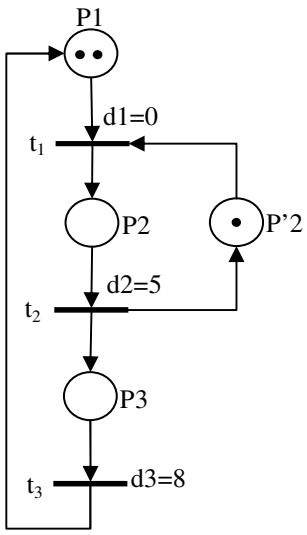
e) Rede de Petri T-Temporizada

Apresentada por Ramchandani [29] em sua tese de Doutorado em 1973 no MIT, a RdP T-Temporizada associa a cada transição da rede um único parâmetro temporal (sua duração de disparo).

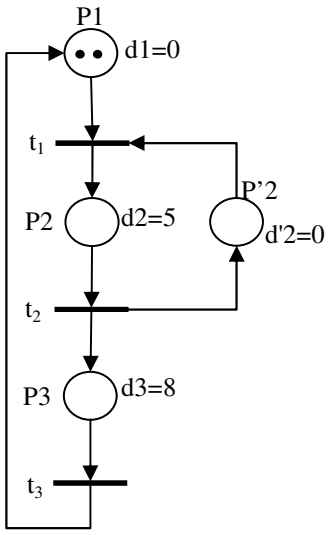
Um tempo d_j, possivelmente de valor zero, é associado com cada transição t_j, neste caso uma marca deverá ser reservada durante um intervalo d_j para permitir o disparo de uma transição. Considerando que o exemplo da figura 2.12 (a) é modelado por uma RdP T-Temporizada, onde uma marca pode ter dois estados: ou ela pode ser reservada para o disparo de uma transição tj ou ela pode ser não-reservada. Somente marcas não reservadas são consideradas para habilitar condições. Na figura 2.12 (b), a transição t_1 está habilitada porque todas as marcas são não-reservadas no tempo inicial. Após o disparo de t1 existe então uma marca reservada em P_2, e uma marca não-reservada em P_1. A transição t_2 está habilitada. A marca em P_2 está então reservada para o disparo de t_2, e este disparo acontecerá 5 unidades de tempo depois, já que $d_2 = 5$.



(a)



(b) RdP T-Temporizada



(c) RdP P-Temporizada

Figura 2.12: Rede de Petri Temporizada [23].

Em outras palavras, desde que uma transição torna-se habilitada, seu disparo absorve as marcas correspondentes aos seus lugares de entrada, as quais permanecem na transição durante o tempo da execução do disparo. Quando a duração do disparo termina, então as marcas são depositadas em cada lugar de saída da transição.

f) Rede de Petri P-Temporizada

Contrário ao modelo de Ramchandani [29], associa a cada lugar um tempo, possivelmente de valor zero. Quando uma marca é depositada no lugar, a mesma deverá permanecer no mínimo um tempo neste lugar (esta marca é dita ser indisponível por este tempo). Quando o tempo decorreu, as marcas então tornam-se disponíveis. Somente marcas disponíveis são consideradas para habilitar condições. A figura 2.12 (c) mostra um exemplo de rede P-Temporizada.

Das classificações de RdP existentes, nas quais algumas delas foram apresentadas nesta seção, utilizou-se para o desenvolvimento deste trabalho o tipo Lugar/Transição, por oferecer características que atendessem as necessidades iniciais deste trabalho, além de ser o único tipo de RdP tratado (modelado) pelo ambiente PIPE, este gera o código PNML, no formato XML, do modelo gráfico da RdP. A descrição PNML permite a análise de toda a rede e possibilita que o modelo da RdP seja interpretado e utilizado pelas ferramentas desenvolvidas neste trabalho.

3 FERRAMENTAS COMPUTACIONAIS UTILIZADAS NO PROJETO

Neste capítulo são abordadas as ferramentas computacionais empregadas nos quatro ambientes de síntese digital desenvolvidos neste trabalho, para validar as metodologias de modelagem.

Na seção 3.1 são expostos conceitos elementares sobre a linguagem de descrição de hardware VHDL, procurou-se enfatizar as vantagens proporcionadas pelo uso da linguagem, além dos tipos de descrições e construtores utilizados nos modelos VHDL gerados nos arquivos de saída, por parte dos ambientes desenvolvidos. Para encerrar esta subseção realizou-se uma comparação entre a descrição comportamental e RTL, com o objetivo de destacar as diferenças existentes entre elas.

Em seguida são apresentados todos os programas utilizados por alguns dos ambientes desenvolvidos neste trabalho. Em particular, são citados programas desenvolvidos por outros membros de nosso grupo de pesquisa.

Finalmente apresenta-se o ambiente PIPE [30], utilizado para modelar FSM dos modelos de Mealy ou Moore em RdP Lugar/Transição, através das metodologias de modelagem propostas nesta dissertação.

3.1 VHDL

VHDL é uma linguagem utilizada para descrever sistemas eletrônicos digitais. Originalmente seu propósito foi apenas para especificar *hardware*, atualmente ela é bem mais ampla, sendo utilizada para simulação e síntese [31].

Surgiu a partir de um programa de Circuitos Integrados de Velocidade Muito Alta (VHSIC), financiado pelo Departamento de Defesa Americano (1980). No curso do programa houve um esforço para padronizar as descrições das estruturas e funções de Circuitos Integrados (IC's). Posteriormente, a Linguagem de Descrição de *Hardware* VHSIC foi desenvolvida e tornou-se um padrão do Instituto de Engenheiros Elétricos e Eletrônicos (IEEE) dos EUA.

Como toda linguagem de descrição de *hardware*, VHDL possui duas principais aplicações: documentação e modelagem de projeto [32].

Uma boa documentação ajuda a garantir a precisão e a portabilidade de um projeto, enquanto a modelagem é usada para validar o projeto. A validação, neste caso, é obtida através da simulação, a qual vem rapidamente substituindo o custoso processo de prototipação [32].

Sendo assim a descrição de um sistema em VHDL apresenta inúmeras vantagens, tais como:

a. Os projetos em suas fases iniciais podem ser desenvolvidos em nível alto de abstração, independentes da tecnologia (implementação física);
b. O objetivo do projeto fica mais claro que na representação por esquemáticos, nos quais a implementação se sobrepõe à intenção do projeto [33];
c. Os projetos são fáceis de serem modificados [31];
d. O volume de documentação diminui, já que um código bem comentado em VHDL substitui o esquemático e a descrição funcional do sistema [33];
e. Permite através de simulação verificar o comportamento do sistema digital [31];
f. Reduz, consideravelmente, o tempo de projeto e implementação [31];

Quanto às desvantagens apenas uma é relevante: a VHDL não gera um *hardware* otimizado.

3.1.1 Estrutura de um programa VHDL

A estrutura básica de um programa VHDL, baseia-se em três blocos: *package*, *entity*, *achitecture*.

Cada módulo em VHDL tem sua própria *entity* e *architecture*. As arquiteturas podem ser descritas tanto em nível comportamental quanto estrutural ou uma mistura desses níveis, a arquitetura é responsável por definir os componentes ou comportamento da entidade (*entity*) e suas conexões. Assim o bloco "*architecture*" especifica o aspecto funcional do modelo.

Toda a comunicação ocorre através das portas (*port*) declaradas em cada entidade, observando-se o tipo, tamanho, se trata de sinal ou barramento e a direção. A lista "*port*" é responsável por definir os sinais externos. Assim as *entity* declaram as interfaces do projeto (pinos de entrada e saída).

Nos pacotes são declarados tipos de dados e subprogramas, e constantes, com o objetivo de reutilização do código.

A figura 3.1 apresenta a estrutura básica de um programa descrito em VHDL.

LIBRARY IEEE; **USE IEEE.STD_LOGIC_1164.all;** **USE IEEE.STD_LOGIC_UNSIGNED.all;**	**PACKAGE** **(BIBLIOTECAS)**
ENTITY exemplo **IS** **PORT (** < descrição dos pinos de entrada e saída > **);** **END** exemplo;	**ENTITY** **(PINOS DE I/O)**
ARCHITECTURE teste **OF** exemplo **IS** **BEGIN** **PROCESS(** <pinos de entrada e signal > **)** **BEGIN** < descrição do circuito integrado > **END PROCESS**; **END** teste;	**ARCHITECTURE** **(ARQUITETURA)**

Figura 3.1: Representação da estrutura básica de um programa descrito em VHDL.

Resumidamente, a função de uma entidade é determinada pela sua arquitetura, desta forma, podem ser definidas múltiplas arquiteturas para a mesma entidade.

3.1.2 Principais Construtores

Apresentam-se nesta subseção alguns dos construtores da linguagem VHDL, que são utilizados pelos ambientes de síntese desenvolvidos, estes ambientes são responsáveis por gerar o código VHDL da FSM modelada em RdP. Os construtores citados são: atribuição de sinal, comparações lógicas e estruturas de decisões lógicas.

3.1.2.1 Atribuição de Sinal

As atribuições de sinais em VHDL são feitas normalmente com o operador "<=", por ser tratado analogamente a um pino de entrada e saída. Na atribuição de sinais pode-se adicionalmente gerar um atraso programado através da palavra-chave "*after*". No exemplo: y <= *not* x a*fter* 5ns. O sinal y recebe o valor negado de x após um atraso de 5ns.

A estrutura flexível da VHDL permite a atribuição de valores em processos vinculada a uma ou mais condições, ou seja, a atribuição só se efetivará se uma condição (ou conjunto de condições) for verdadeira. Para se realizar esta operação em uma atribuição deve-se utilizar a palavra-chave *when*, seguida da condição que se deseja verificar para a validade da atribuição. Na figura 3.2, tem-se listada parte da descrição de um comparador binário para palavras de quatro bits, utilizando-se este recurso. Nota-se que o sinal de saída *equals* receberá um valor de acordo com a condição dos pinos "a" e "b". Caso o sinal no pino "a" seja igual ao presente em "b", o pino *equals* receberá o bit 1, caso contrário receberá o bit 0.

```
equals <= '1' when (a=b)
          else '0';
```

Figura 3.2: Atribuição de sinal para várias condições.

É importante destacar que os sinais podem ser declarados dentro das seções de uma entidade, arquiteturas e pacotes. Sinais dentro de pacotes são também referenciados como sinais globais porque podem ser compartilhados entre as entidades.

3.1.2.2 Comparações e Operações Lógicas

As comparações lógicas utilizadas na linguagem VHDL estão apresentadas na tabela 3.1 e são semelhantes às utilizadas em outras linguagens de programação.

Tabela 3.1: Representa os operadores de comparação e os tipos de operadores [34].

Operador	Operação	Tipo do operador da esquerda	Tipo do operador da direita	Tipo do resultado
=	Igualdade	Qualquer	Qualquer	*Boolean*
/=	Desigualdade	Qualquer	Qualquer	*Boolean*
<	Menor que	Escalar ou array	Escalar ou array	*Boolean*
<=	Menor ou igual	Escalar ou array	Escalar ou array	*Boolean*
>	Maior que	Escalar ou array	Escalar ou array	*Boolean*
>=	Maior ou igual	Escalar ou array	Escalar ou array	*Boolean*

O símbolo “<=” é utilizado na linguagem VHDL para duas funções distintas, uma função é a atribuição de sinal e a outra, comparação lógica de dois elementos. Ao utilizar o símbolo para atribuição de sinal, a variável localizada a sua esquerda deverá estar declarada dentro do código como uma porta ou um sinal de saída. Quando utiliza-lo para comparação os elementos da sua esquerda e direita deverão ser declarados como portas ou sinal de entrada, estes elementos serão apenas dos tipos escalares ou vetores.

Em VHDL também são utilizadas algumas operações lógicas, estas estão representadas na tabela 3.2.

Tabela 3.2: Representa os operadores das operações lógicas e os tipos de operadores [34].

Operador	Operação	Tipo do operador da esquerda	Tipo do operador da direita	Tipo do resultado
and	Lógica *AND*	Bit, boolean ou array (bit, boolean)	Bit, boolean ou array (bit, boolean)	Boolean
or	Lógica *OR*	Bit, boolean ou array (bit, boolean)	Bit, boolean ou array (bit, boolean)	Boolean
nand	Lógica *AND* negada	Bit, boolean ou array (bit, boolean)	Bit, boolean ou array (bit, boolean)	Boolean
nor	Lógica *OR* negada	Bit, boolean ou array (bit, boolean)	Bit, boolean ou array (bit, boolean)	Boolean
xor	Lógica *OR* exclusivo	Bit, boolean ou array (bit, boolean)	Bit, boolean ou array (bit, boolean)	Boolean
xnor	Lógica XOR negada	Bit, boolean ou array (bit, boolean)	Bit, boolean ou array (bit, boolean)	Boolean

3.1.2.3 Estruturas de Decisões Lógicas

Nesta subseção serão apresentadas as estruturas de decisões lógicas que podem ser utilizadas no escopo de um processo, estrutura *If e Case*. Elas possuem um estilo semelhante à sintaxe de outras linguagens de programação.

a) Estrutura *IF*

Em aplicações onde o estado de alguma variável (ou conjunto de variáveis) determina as operações que devem ser realizadas é muito comum o uso da estrutura *If*. A forma mais simples de utilização é através da estrutura *If-Then-End if,* como apresentado na figura 3.3, onde a condição (x < 10) é testada para permitir a operação necessária. Se a condição for verdadeira, "b" é atribuído à variável "a", caso contrário nada é feito.

```
IF (x < 10) THEN
        a := b;
END IF
```

Figura 3.3: Estrutura *If-Then-End if* [34].

Em alguns casos é desejado realizar-se uma operação (ou seqüência de operações) para quando a condição for verdadeira e outra quando a condição for falsa, utilizando desta forma a estrutura *If-Then-Else-End if*.

Outra variação desta estrutura consiste na utilização da palavra chave *Elsif*, que funciona como uma nova estrutura de comparação dentro da estrutura *If* corrente. São possíveis tantos *Elsif* quantos forem necessários para refinar as comparações. Observe a figura 3.4, onde a estrutura *Elsif* é utilizada para comparar o valor da variável *input*. Caso *input* for igual a 1 o sinal *State* recebe S0, caso *input* seja igual a 0 o sinal *State* recebe S1.

```
IF input = '1' THEN
        State <= S0;
ELSIF input = '0' THEN
        State <= S1;
END IF;
```

Figura 3.4: Utilizando a estrutura Elsif .

b) Estrutura *CASE*

Outra estrutura de comparação de valores para seleção de operações é a estrutura *Case-is-When-End Case.* Esta estrutura é mais utilizada para aplicações onde uma determinada variável pode assumir um número limitado de valores, cada qual associado a um conjunto de operações.

Para se implementar a função "ou" em estruturas *Case* usa-se o caracter "|".

Para a área de sistemas digitais a grande utilidade da estrutura *Case* é na implementação de máquinas de estado. Para exemplificar seu uso, considere-se a máquina de estados do tipo Moore, mostrada na figura 3.5, utilizando a estrutura apresentada nesta subseção. Os estados S0 e S1 da máquina fornecem saídas de valores 0 e 1 respectivamente. Qualquer entrada em S0 deslocará a máquina para o estado S1. Estando neste estado, se a máquina perceber entrada de valor lógico 0 ela permanecerá em S1 caso contrário irá para o estado S0.

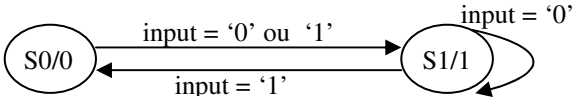


Figura 3.5: Máquina de Estados [33].

Na figura 3.6 é apresentada a descrição VHDL, da máquina apresentada na figura 3.5. Nota-se que a estrutura *Case* apresentada na figura 3.6 está dentro de um processo sincronizado, indicado pelo comando *Process(clk)*, o que normalmente é subentendido em máquinas de estados. A *Case* controla a mudança de estados da máquina conforme o tipo do sinal *state*, assim quando o sinal for igual a S0 será atribui a este o tipo de estado S1, caso *state* e a entrada identificada, sejam S1 e 1 respectivamente o *state* receberá S0, caso contrário será atribuído ao sinal o estado S1. Sendo assim, a estrutura *Case* orienta a mudança de estados, que só é realizada quando o evento clk ocorrer, que é representada pela instrução *ELSEIF (clk'EVENT AND clk='1').*

```
ENTITY state_machine IS
        PORT (
                clk     : IN    BIT;
                input   : IN    BIT;
                reset   : IN    BIT;
                output  : OUT   BIT);
END state_machine;

ARCHITECTURE a OF state_machine IS
        TYPE STATE_TYPE IS (S0, S1);
        SIGNAL state : STATE_TYPE;
BEGIN
        PROCESS (clk)
        BEGIN
                IF reset = '1' THEN
                        state <= S0;
                ELSIF (clk'EVENT AND clk = '1') THEN
                        CASE state IS
                                WHEN S0 =>
                                        state <= S1;
                                WHEN S1 =>
                                        IF input = '1' THEN
                                                state <= S0;
                                        ELSE
                                                state <= S1;
                                        END IF;
                        END CASE;
                END IF;
        END PROCESS;
        output <= '1' WHEN state = S1 ELSE '0';
END a;
```

Figura 3.6: Descrição da máquina de estados da figura 3.5, utilizando o comando Case [35].

3.1.3 Descrição RTL

Uma descrição nível de transferência entre registradores (RTL – *Register Transfer Level*) é caracterizada por um estilo que especifica todos os registradores de um projeto e a

lógica combinacional entre eles. O estilo da descrição RTL é mostrado na figura 3.7 pelo diagrama “nuvem e registrador” [36]. Entretanto o “objeto nuvem”, referenciado na figura 3.7, representa apenas a lógica combinacional, mas na prática está não é a representação convencional de nenhum circuito.

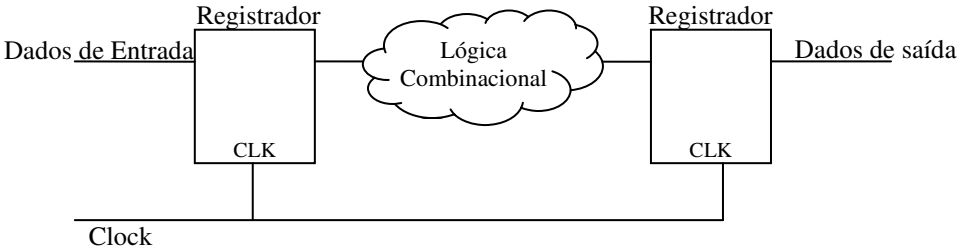


Figura 3.7: Diagrama nuvem e registrador[35].

O projetista quando utiliza a descrição RTL está preocupado em como os dados fluem entre os registradores. Desta forma, o projeto contém informações sobre a arquitetura, mas não contém detalhes sobre a tecnologia.

A lógica combinacional é descrita na implementação VHDL por equações lógicas, indicações de controle seqüencial (*Case, If then Else, etc*.), subprogramas, ou através de indicações concorrentes, esta lógica é representada pelo objeto “nuvem” entre os registradores da figura 3.7. Resumidamente pode-se dizer que a descrição RTL foca a implementação.

As descrições RTL são usadas para sincronizar projetos e descrever o comportamento do projeto a cada ciclo de relógio do sistema.

Neste tipo de descrição, os valores de saída são atribuídos diretamente, através de expressões lógicas.

Na figura 3.8 é dado um exemplo de descrição RTL, esta representa um comparador binário para palavras de quatro bits [31].

```
Entity comp4 is
        port ( a, b: in bit_vector (3 downto 0);
                equals: out bit);
End comp4;
Architecture RTL of comp4 is
Begin
        equals <= ‘1’ when (a=b) else ‘0’;
End RTL;
```

Figura 3.8: Descrição RTL para o comparador binário de palavras de quatro bits [31].

3.1.4 Descrição Comportamental

O nível mais alto de descrição de um sistema é o comportamental. Neste caso o projeto é descrito sem especificar a arquitetura ou os registradores, sendo assim, o projeto pode ser descrito em termos das funcionalidades desejadas, sem a necessidade de se entrar em detalhes de implementação. Resumidamente, tem-se, que a VHDL comportamental foca o comportamento.

Projetar um sistema utilizando-se da descrição comportamental é similar a programar em outra linguagem de programação, por exemplo, linguagem C. Portanto, esta é a forma mais flexível e poderosa de descrição. São definidos processos concorrentes (*process*), sendo que a cada processo é associada uma lista de sensibilidade, que indicam quais são as variáveis cuja alteração deve levar à reavaliação da saída.

No simulador funcional, quando uma variável da lista é modificada, o processo é simulado novamente.

O código da figura 3.9 representa a descrição comportamental de um comparador binário para palavras de quatro bits [33]. Observa-se na figura 3.9 que o comando *process* esta associado à lista de sensibilidade (a,b), esta lista indica que as variáveis a e b serão analisadas para a alteração da saída representada pela variável *equals*.

```
Entity comp4 is
        port (a, b: in bit_vector (3 downto 0);
                equals: out bit);
End comp4;

Architecture comport of comp4 is
Begin
        comp: process (a,b) -- lista de sensibilidade
                Begin
                        If a = b then
                                equals <= '1' ;
                        Else
                                equals <= '0' ;
                        End If;
                End process comp;
End comport;
```

Figura 3.9: Descrição comportamental para o comparador binário de palavras de quatro bits [33].

3.1.5 Características das Descrições RTL e Comportamental

Nesta seção são apresentadas as características pertinentes às descrições RTL e comportamental [37]:

a. A maioria das ferramentas de síntese que existem hoje requer que a descrição de circuitos esteja escrito em RTL, está é a razão para a RTL ser tão importante, nesse nível o projetista ainda mantém controle sobre a arquitetura dos registradores do projeto;
b. Por outro lado, as ferramentas de síntese comportamental geram automaticamente arquiteturas de portas lógicas e de registradores direto de uma descrição comportamental;
c. Descrição comportamental é mais rápida de descrever e mais simples.
d. Descrição comportamental aumenta desempenho da simulação.
e. Geralmente precisa-se escrever projeto de FPGA ou PLD em RTL para usar as ferramentas de síntese disponíveis;
f. O nível comportamental é usado para criar estruturas de estímulos, para modelar partes padrão do projeto, ou para criar especificações simuláveis do seu sistema;
g. Algumas estruturas geralmente não utilizadas em descrição RTL, porém úteis em código comportamental são: funções e procedimentos, tipo *string* e *time*, arquivos, registros, matrizes, listas, ponteiros e alocação dinâmica.

Atualmente, as descrições são feitas utilizando conjuntamente os níveis RTL e comportamental para dar maior flexibilidade ao projetista. E para isso os fabricantes já dispõem de ferramentas de síntese lógica que suporta esses dois níveis de descrição (se for necessário o de portas lógicas também).

3.2 Programa Algoritmo Genético com Propriedades de Substituição (AGPS)

O programa AGPS elaborado por Santos [38] em sua dissertação de mestrado, foi utilizado neste trabalho para realizar a alocação dos estados das FSM modeladas em RdP. Assim tem-se que o AGPS desenvolvido em linguagem de programação C, emprega métodos do algoritmo genético em sua implementação.

Em geral, um algoritmo genético básico consiste nos seguintes passos [38]:

a. Codificação, forma como a solução é representada;
b. Geração de uma população inicial aleatória;
c. Cálculo do custo de cada indivíduo (solução);
d. Seleção dos indivíduos que irão participar da produção da próxima geração e
e. Operador mutação que realiza uma modificação aleatória nos indivíduos para assegurar que o algoritmo não fique preso em um mínimo local.

Assim o algoritmo genético desenvolvido para o programa AGPS, realiza a alocação de estados em máquinas incompleta ou completamente especificadas.

Na figura 3.10 é apresentado o fluxograma que ilustra o comportamento do programa AGPS, desde o início da sua execução até o seu final, verificando as condições necessárias para executar cada etapa. No fluxograma da figura 3.10 os retângulos de cantos arredondados representam apenas o início e o fim da execução do programa, o losango indica quais decisões serão tomadas conforme a condição imposta no mesmo, os retângulos com o canto esquerdo cortado (cartão) lê o arquivo de entrada e gera o de saída. O círculo representa o fim da condição que verifica se a propriedade de substituição (P.S.) é realizada. Os

processamentos realizados pelo AGPS, como cálculos, determinação de conjuntos e verificação de informações são representados no diagrama por retângulos normais.

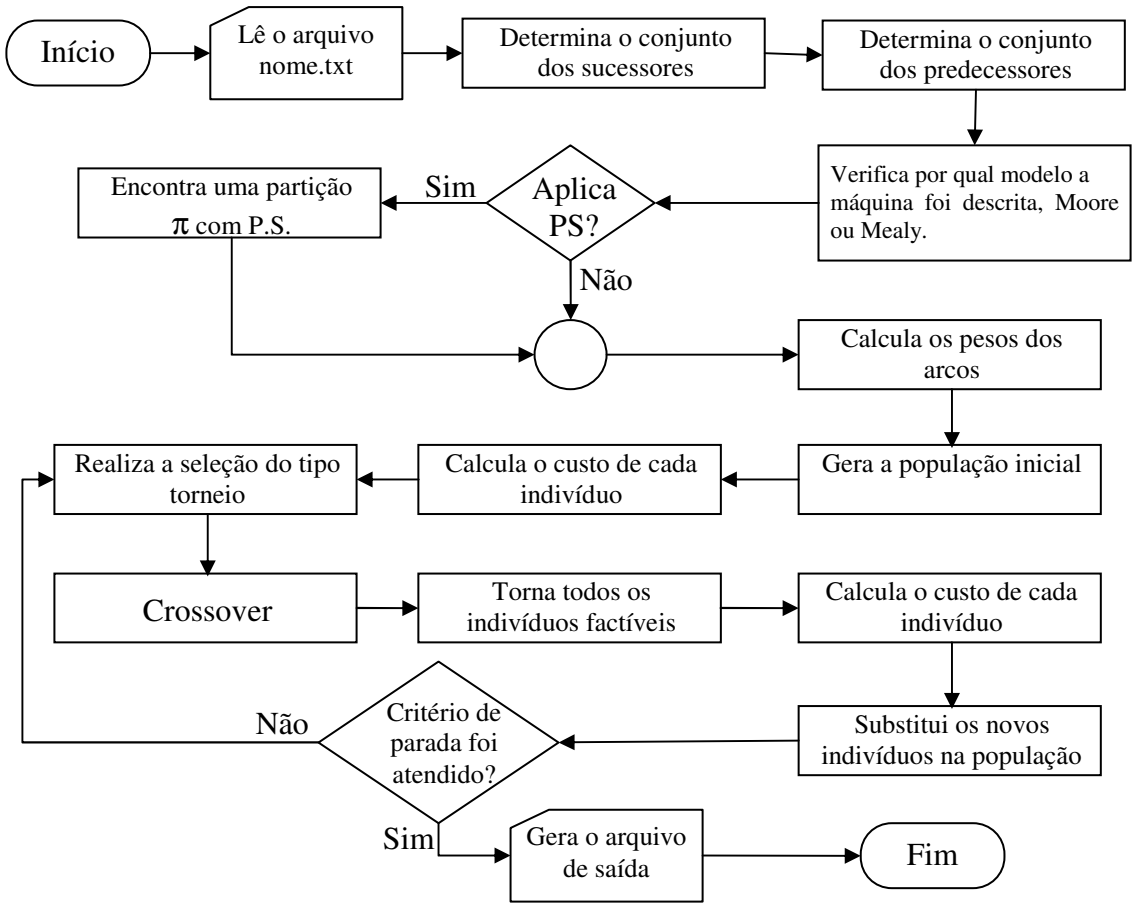


Figura 3.10: Fluxograma do Algoritmo Genético com Propriedade Substituição [38].

O programa AGPS deve ser executado a partir da linha de comando (modo texto) do sistema operacional DOS. Tem-se o seguinte padrão de comando: genetico i [opção] arq_entrada arq_saída. Sendo i a quantidade de iterações. A opção pode ser “A” para gerar o grafo proposto por Amaral [38] ou “P” para considerar a propriedade substituição [38] na geração do grafo.

O arquivo executável e de entrada do programa devem estar no mesmo diretório na qual é feita a chamada ao programa.

A configuração do arquivo de entrada que o programa AGPS utiliza, deve possuir extensão txt. O arquivo de saída possui extensão tab, tal arquivo é utilizado como entrada pelo programa TABELA.

3.3 Programa TABELA

O programa TABELA, implementado por Silva [37] em sua dissertação de mestrado, utilizado neste trabalho para sintetizar FSM modeladas em RdP Lugar/Transição, obtém funções booleanas nas suas formulas mínimas.

Os dados solicitados pelo programa são:

a. nome do dispositivo de saída para os resultados (não deve conter extensão);

b. número de flip-flops;

c. tipo de cada um dos flip-flops (podem ser do tipo D ou JK);

d. número de variáveis de entradas seguido pelo número de variáveis de saída;

e. Tabela de próximo estado, que deve conter os respectivos dados: estado atual, próximo estado, entrada e saída. Os dados devem estar na mesma ordem em que foram citados.

É importante salientar que os estados, as entradas e as saídas devem estar na forma decimal. As máquinas podem ser incompleta ou completamente especificadas e podem estar no modelo de Mealy ou Moore.

O programa gera a tabela de transição de uma máquina seqüencial a partir de seu diagrama de estados, armazenando-a no arquivo de saída. A partir desta tabela são obtidos os mintermos e os *don´t care states* das funções internas (controle) de todos os flip-flops e da saída do circuito, ou seja, minimiza as funções de transições internas correspondentes aos elementos de memória utilizados e as funções de saída do circuito.

Utilizando-se do algoritmo de Quine-McCluskey (Algoritmo de minimização de funções booleanas) estas funções são obtidas nas suas formulas mínimas [39].

O diagrama de blocos do programa TABELA é apresentado na figura 3.11. Observa-se que o diagrama ilustra o funcionamento do TABELA por intermédio de “blocos”,

onde os retângulos de cantos arredondados representam apenas o início e o fim da execução do programa, os retângulos com o canto esquerdo cortado (cartão) indica os dados solicitados pelo programa ao usuário. Os processamentos realizados pelo programa TABELA e seus arquivos de entrada e saída, são representados no diagrama por retângulos normais.

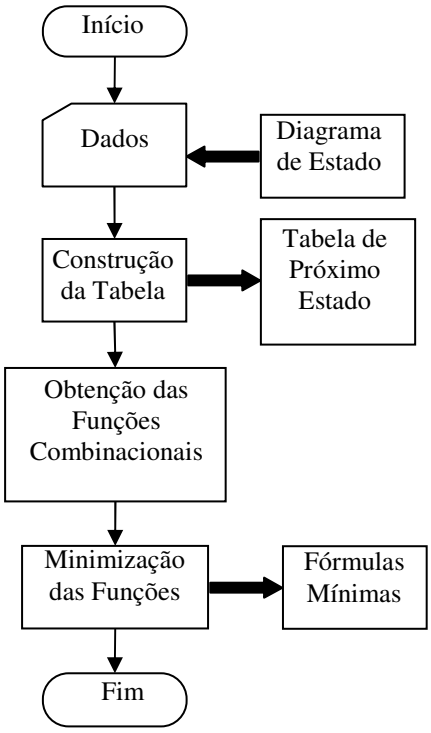


Figura 3.11: Diagrama de blocos do programa TABELA [39].

Neste trabalho o arquivo de saída do programa AGPS é utilizado como arquivo de entrada do programa TABELA e o arquivo de saída deste é usado pelo programa TAB2VHDL como arquivo de entrada.

3.4 Programa TAB2VHDL

O programa TAB2VHDL desenvolvido por Tancredo [40], é uma ferramenta de síntese, utilizada neste trabalho com o propósito de gerar a descrição otimizada na linguagem VHDL do sistema modelado em RdP Lugar/Transição. É importante salientar que a descrição gerada pelo programa TAB2VHDL auxilia o projeto de sistemas digitais e obtém somente a

descrição VHDL do sistema modelado, e especificado a partir da sua descrição em diagrama de transição de estados de uma máquina finita, apresentada no modelo de Mealy ou Moore.

O TAB2VHDL recebe como entrada a saída gerada pelo programa TABELA (funções booleanas e registros do arquivo) e cria um modelo funcional do circuito projetado no modelo RTL descrito em VHDL. Dessa forma, o arquivo contendo a descrição VHDL do circuito poderá ser sintetizado por ambientes de sínteses comerciais e implementado na tecnologia alvo desejada.

A figura 3.12 exibe o diagrama de blocos do programa TAB2VHDL. No diagrama os retângulos de cantos arredondados representam o início e o fim da execução do programa, os retângulos com o canto esquerdo cortado (cartão) indicam os dados solicitados pelo programa ao usuário, os retângulos normais representam os arquivos de entrada e saída utilizados e gerados pelo programa TAB2VHDL.

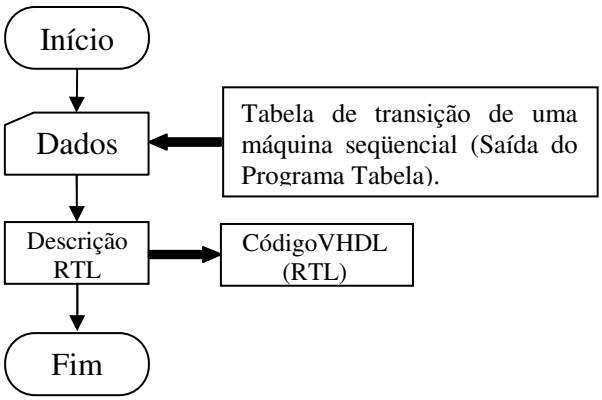


Figura 3.12: Diagrama de blocos do programa TAB2VHDL.

3.5 O editor PIPE

O PIPE (*Platform Independent Petri Net Editor* – Editor de Plataforma Independente para Rede de Petri) [30] foi desenvolvido por um grupo de pesquisadores do Departamento de Computação da Faculdade Imperial de Londres em dezembro de 2002 e atualmente, é mantido pela James D. Bloom. Este software é conhecido como um editor

multi-plataforma e tem como objetivo principal melhorar o suporte para modelar uma RdP. Pode-se destacar como características próprias do PIPE.

a) Implementação em Java;

b) Disponibilidade do código fonte (open source) à comunidade de programadores;

c) Plataforma independente;

d) Utiliza como formato padrão a PNML (Petri Net Markup Language) para salvar e carregar as RdP;

e) A descrição da RdP é obtida em linguagem XML (eXtensible Markup Language).

4 METODOLOGIAS DESENVOLVIDAS PARA MODELAR MÁQUINAS DE ESTADOS FINITOS EM REDES DE PETRI

A modelagem de sistemas em RdP envolve a construção de um formalismo gráfico e/ou matemático que expressa o comportamento do sistema modelado. Assim as RdP garantem o poder de decisão, além de tratar processos concorrentes e paralelos, itens que não são tratados pelas FSM, representadas por diagramas de estados [41]. Os diagramas de estados têm como característica principal a unicidade do "próximo estado", ou seja, nunca habilita mais de um estado simultaneamente.

Nesta seção apresentam-se as duas metodologias desenvolvidas para a modelagem de sistemas. Uma das metodologias desenvolvidas, denominada 4M, modela em RdP apenas FSM do tipo Mealy e a outra metodologia, denominada 5M, modela em RdP máquinas de Mealy ou Moore. As RdP são modeladas no ambiente PIPE utilizando os conceitos das metodologias desenvolvidas. O PIPE possibilita representar graficamente uma RdP Lugar/Transição e gera a descrição no formato XML obedecendo os padrões de descrição PNML da rede esquematizada .

4.1 Metodologia de Modelagem para Máquinas de Mealy

Para modelar uma FSM do tipo Mealy em uma RdP equivalente, desenvolveu-se uma Metodologia de Modelagem para Máquinas de Mealy, denominada 4M, a qual utiliza os componentes básicos (lugares, transições e arcos) de uma RdP Lugar/Transição, da seguinte forma:

a. Os lugares da RdP representam as entradas, saídas e estados da máquina de Mealy;

b. As transições são responsáveis por realizar um evento ou uma mudança de estado;

c. Os arcos da RdP são responsáveis por conectarem lugares que representam o estado atual e a entrada à transição, e a transição aos lugares que representam o próximo estado e a saída da máquina modelada.

Cada arco da máquina de estados do tipo Mealy representa uma mudança de estado, neste mesmo arco são representadas a entrada e saída da máquina, portanto os lugares que modelam o estado atual e a entrada são pré-condições para o disparo da transição (RdP) que indica a mudança de estado, e os que modelam o próximo estado e a saída são pós-condições dessa transição, isto é, são as condições resultantes da ação. A mudança de estado, bem como o próximo estado atingido, depende tanto do estado atual quanto das entradas da máquina. Dessa forma, ao representar as entradas, saídas e estados da máquina por lugares na RdP ter-se-á que cada transição da rede apresentará dois lugares como pré-condições (representando o estado atual e entrada do sistema) e outros dois como pós-condições (próximo estado e saída) do seu disparo. O disparo de uma transição sempre terá dois lugares como pré e pós-condições, independentemente da quantidade de entradas e saídas existentes no modelo, pois cada transição controla apenas uma entrada e fornece uma única saída de acordo com os estados atuais e próximos. O lugar que representa o estado atual é marcado, inserindo-se uma marca no lugar que o representa, enquanto os demais lugares que representam outros estados não são marcados na modelagem. As marcas nos lugares que representam a entrada da máquina serão inseridas apenas durante a simulação da RdP.

Com intuito de apresentar a metodologia de modelagem 4M, escolheu-se uma máquina de estados que representa um sistema capaz de detectar uma seqüência de três zeros

consecutivos sem sobreposição para uma seqüência lógica de zeros e uns. Para representar o detector utilizou-se a máquina de Mealy mostrada na figura 4.1, através do seu diagrama de estados. O estado inicial da máquina é o A. Se um caractere de nível lógico 1 chegar à entrada, a máquina continuará no estado A e a saída será zero. Na figura 4.1 esta condição está representada no arco que tem como origem o estado A e como destino o mesmo estado, rotulado pelos caracteres "1/0". Caso chegue à entrada um caractere 0, a máquina vai para o estado B e a saída continuará sendo zero. O que é representado pelo arco com origem em A e destino em B. Caso a máquina esteja no estado B, e chegue à entrada um caractere 0, a máquina vai para o estado C e a saída continuará 0. No entanto, se a máquina estiver no estado B e chegar à entrada um caractere com valor lógico 1, a máquina retorna ao estado A e a saída será 0. Se a máquina estiver no estado C e a entrada seja 1, a máquina retorna ao estado A e a saída será 0, esta condição está representada na figura 4.1 no arco que tem como origem o estado C e como destino o A, rotulado pelos caracteres "1/0". Se a máquina estiver no estado C e a entrada seja 0, a máquina retorna ao estado inicial A e a saída será o caractere 1, indicando que a seqüência de três zeros consecutivos foi identificada, esta condição está representada no arco rotulado pelos caracteres "0/1", que tem como origem o estado C e destino o A. A saída com valor lógico 1 será fornecida novamente somente quando forem identificados mais três zeros consecutivos na entrada.

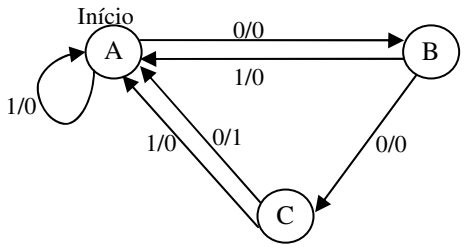


Figura 4.1: Máquina de Mealy para o detector de três zeros consecutivos sem sobreposição.

Na figura 4.2, apresenta-se a RdP Lugar/Transição correspondente a máquina mostrada na figura 4.1, ou seja, uma RdP que detecta a entrada de três zeros consecutivos sem sobreposição. Os lugares A, B e C mostrados na figura 4.2 representam os estados da máquina

da figura 4.1 e os lugares E0, E1, S0, S1 representam as entradas de valores lógicos 0, 1 e as saídas de valores 0 e 1 respectivamente.

As transições T0, T1, T2, T3, T4 e T5 são responsáveis pela evolução da rede, ou seja, pela mudança de estados, obedecendo a suas pré e pós-condições.

Os arcos orientam o seguimento das marcas na RdP, que constituirá a "nova" marcação, esta é responsável por indicar o estado atual da máquina, as entradas fornecidas pelo usuário e as saídas geradas no decorrer da simulação.

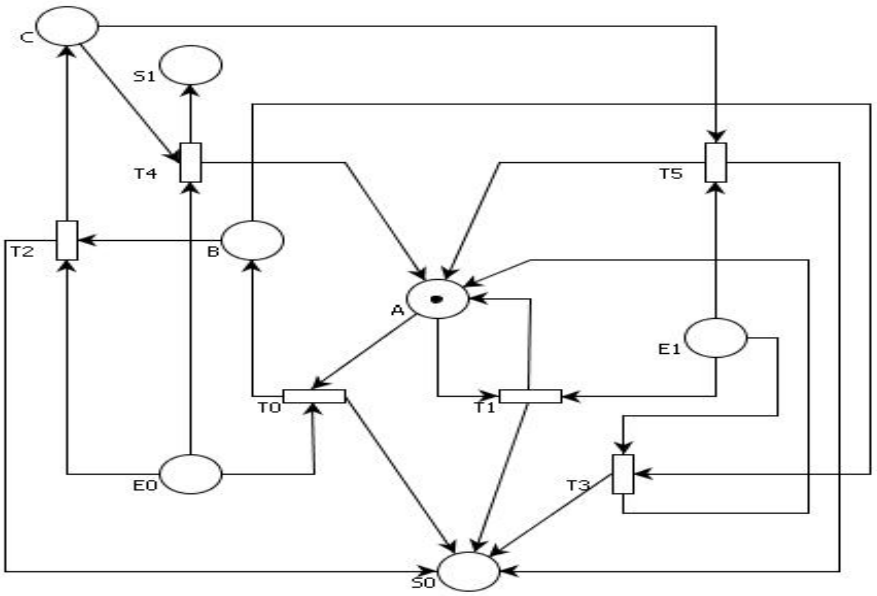


Figura 4.2: Representação da FSM do tipo Mealy modelada em RdP Lugar/Transição.

Entretanto foi necessário estabelecer alguns critérios para denominar os lugares que representam as entradas, saídas e estados da máquina, devido a utilização da descrição PNML gerada pelo modelo da RdP da máquina modelada pelos programas PIPE2TAB4M, PIPE2VHDL4M que geram a tabela de estados e a descrição VHDL. Assim os critérios estabelecidos são os seguintes:

a. Cada um dos estados correspondentes à FSM do tipo Mealy devem ser representados por um lugar específico na RdP. Os nomes atribuídos a esses lugares não devem conter caracteres numéricos;

b. As entradas e saídas também devem ser representadas por lugares na RdP;

c. O nome do lugar que representa a entrada da máquina deve possuir o caractere “E” como primeiro caractere, seguido de um número decimal que corresponde ao valor da entrada, por exemplo, E0 representa a entrada de valor lógico 0;

d. O nome do lugar que representa a saída da máquina deve possuir o caractere “S” como primeiro caractere, seguido de um número decimal que corresponde ao valor da saída, por exemplo, S0 representa a saída de valor lógico 0;

e. O estado inicial da FSM deve ser destacado acrescentando-se somente uma marca no lugar que o representa.

Todos os padrões estipulados anteriormente devem obrigatoriamente ser aplicados para a modelagem de qualquer FSM do tipo Mealy, utilizando-se a metodologia de modelagem 4M.

No caso da RdP Lugar/Transição apresentada na figura 4.2 os lugares S0 e S1 são depósitos de marcas e E0 e E1 possuirão marcas apenas quando estas forem inseridas pelo usuário durante a simulação. Assim a cada simulação são inseridas novas marcas nos lugares E0 e E1 e com o disparo das transições as marcas serão retiradas desses lugares e inseridas em S0 e S1. Assim os estados E0, E1, S0 e S1 são utilizados apenas para controlar as entradas e saídas durante a simulação.

A fim de mostrar o funcionamento da RdP apresentada na figura 4.2, utilizar-se-á o ambiente PIPE para realizar a simulação.

O PIPE além de simplificar e facilitar a modelagem da rede oferece alguns atrativos na simulação. Toda simulação da rede, seja ela pseudo-aleatória ou controlada pelo usuário, pode ser acompanhada pela documentação produzida pelo ambiente, esta é apresentada no lado esquerdo da tela e indica todas as transições disparadas durante a simulação.

Apresenta-se inicialmente a simulação pseudo-aleatória, ou seja, a realizada pelo ambiente PIPE. Antes de ativar a simulação é necessário inserir marcas nos lugares que representam as entradas do sistema, a adição de cinco marcas no lugar E0 e três no lugar E1 está ilustrada na figura 4.3.

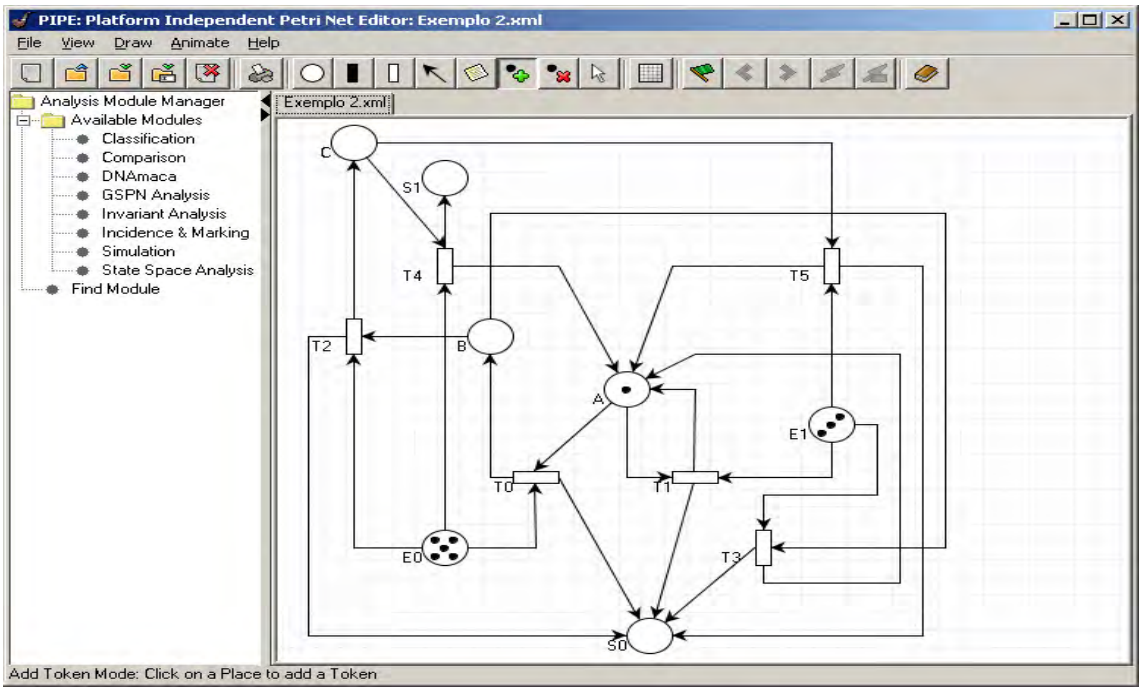


Figura 4.3: Representando a adição de entradas realizada no software PIPE.

Dois passos são necessários para iniciar a simulação do sistema.

1º Acionar o botão para o modo animação;

2º Ativar o botão para o disparo pseudo-aleatório das transições. Neste passo é necessário fornecer a quantidade de transições que se deseja disparar durante a simulação.

Os dois passos necessários para iniciar a simulação e o resultado da simulação pseudo-aleatória para as entradas fornecidas estão ilustrados na figura 4.4. Observa-se que as transições disparadas durante a simulação pseudo-aleatória são mostradas no final da simulação no lado esquerdo na tela do ambiente PIPE em seu histórico de animação, a seqüência das transições permite que o usuário identifique à seqüência lógica de entradas utilizadas pelo PIPE durante a simulação. Cada transição está relacionada a uma única entrada, este fato facilita a identificação da entrada através da análise do modelo da RdP. A

seqüência de transições disparadas é definida pelo ambiente PIPE conforme a simulação realizada pelo mesmo.

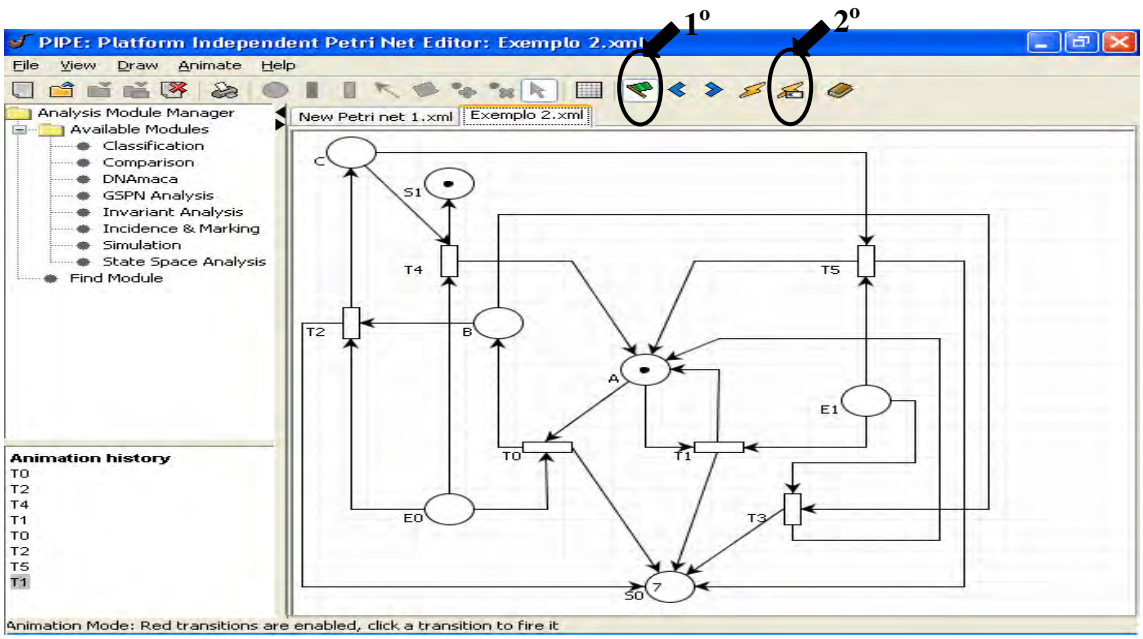


Figura 4.4: Simulação pseudo-aleatória realizada pelo software PIPE.

Para disparar a transição T0 da figura 4.4 é necessária pelo menos uma marca nos lugares A e E0 (representa a entrada de valor lógico 0), assim analisa-se que a primeira entrada considerada na simulação pseudo-aleatória é de valor 0. Para disparar T2 os lugares B e E0 têm que possuir pelo menos uma marca, assim a segunda entrada considerada durante a simulação foi de valor 0. Fazendo essa análise para todas as transições apresentadas no histórico da simulação pseudo-aleatória confere-se que a seqüência de entrada para este caso específico foi 00010011. Observando o histórico da simulação é possível identificar uma única seqüência de entradas, pois cada transição esta relacionada a somente uma entrada no modelo e o disparo das mesmas são indicados em ordem de execução. Assim a simulação da RdP da figura 4.4, gerou sete saídas com valor lógico 0 e uma saída com valor lógico 1.

A outra simulação apresentada é a controlada pelo usuário, esta por sua vez é mais trabalhosa, pois exige que o usuário insira a cada instante uma marca no lugar que representa a entrada desejada, obedecendo à seqüência que deseja seguir. Esta simulação é oposta à pseudo-aleatória, onde o usuário não sabe qual a seqüência que será atribuída pelo ambiente.

Considerando que o lugar A (estado inicial) da RdP possua uma marca e que outra seja inserida no lugar E0 (entrada de valor 0), habilitando a transição T0, conforme está apresentada na figura 4.5. O ambiente atribui a cor vermelha às transições habilitadas, para indicá-las.

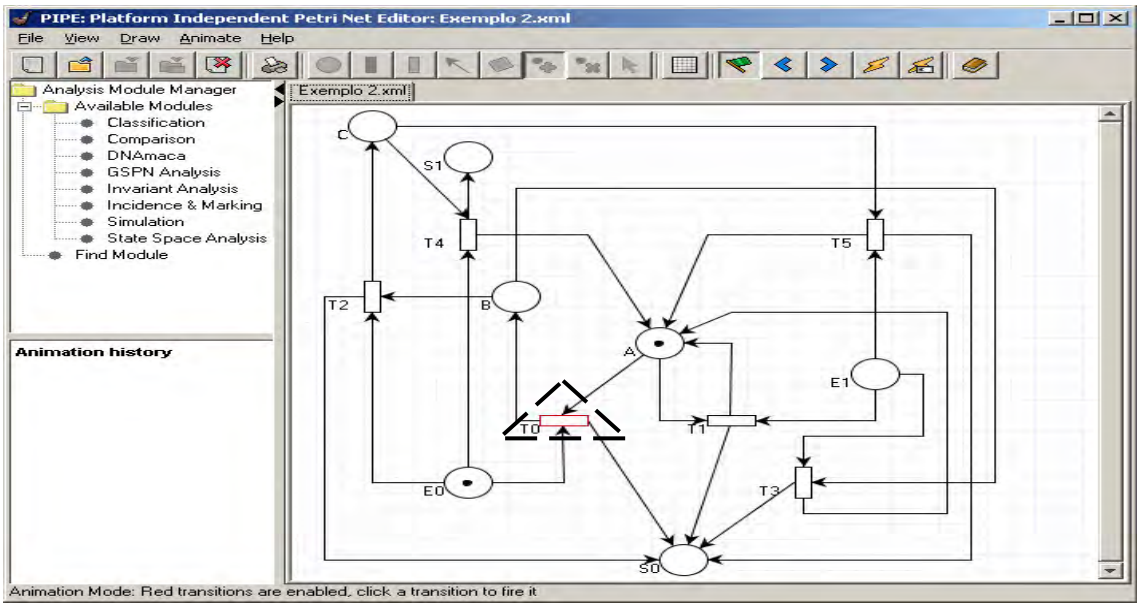


Figura 4.5: Transição T0 habilitada.

Com o disparo de T0, que será efetuado selecionando-se a transição habilitada ou pelo componente de disparo aleatório da transição (1), os lugares B e S0 (saída com valor lógico 0) são marcados, como é ilustrado na figura 4.6.

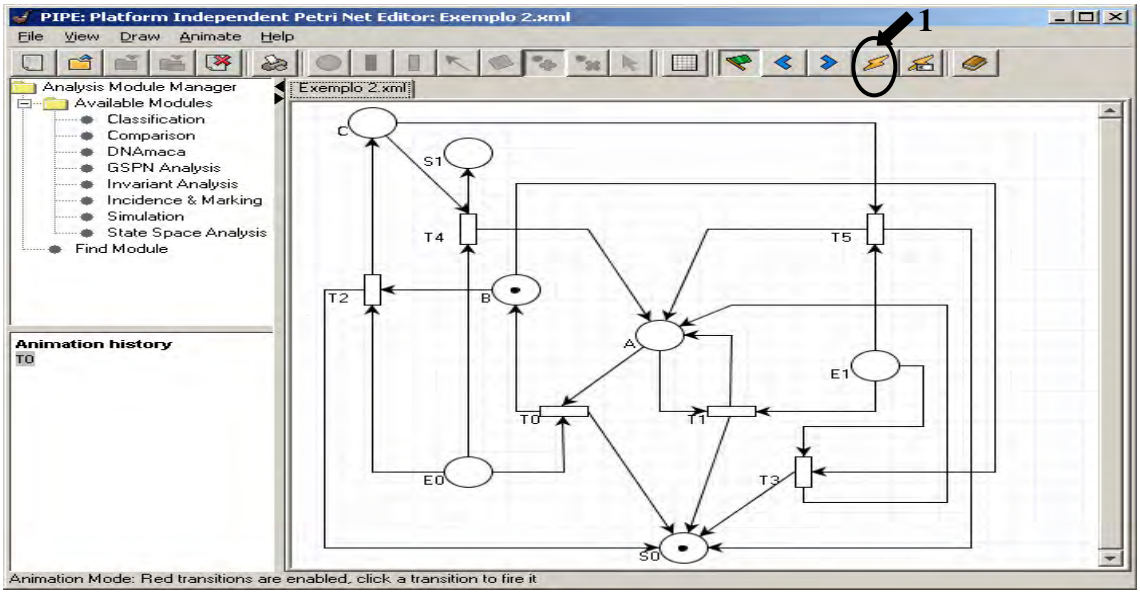


Figura 4.6: Marcação nos estados B e S0 (saída 0) após o disparo de T0.

Após o disparo de T0 insere-se uma marca no lugar E1, habilitando a transição T3. A figura 4.7 mostra a marca no lugar E1 e a transição T3 habilitada, destacada por um triangulo a envolvendo vermelho.

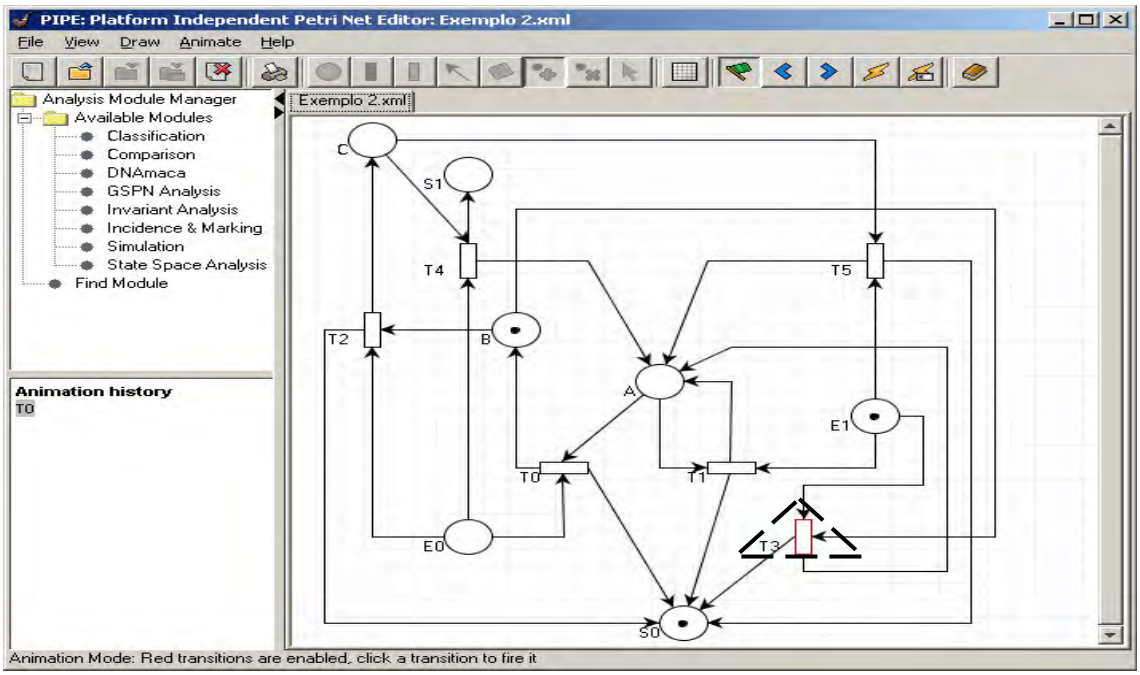


Figura 4.7: Transição T3 habilitada.

Os lugares marcados após o disparo de T3 são A e S0, conforme a representação da figura 4.8, onde o lugar S0 já possui duas marcas e A uma marca.

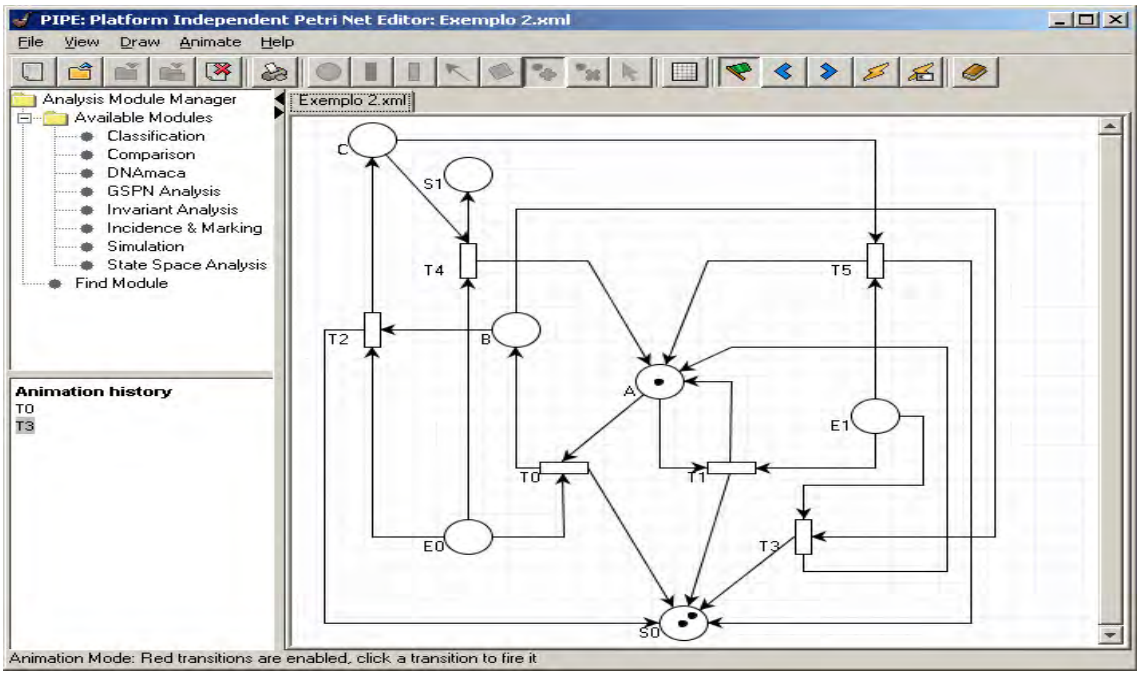


Figura 4.8: Marcação nos estados A e S0 (saída 0) após o disparo de T3.

Se posteriormente for inserida a entrada 0 observa-se, na figura 4.9 que a transição habilitada é T0.

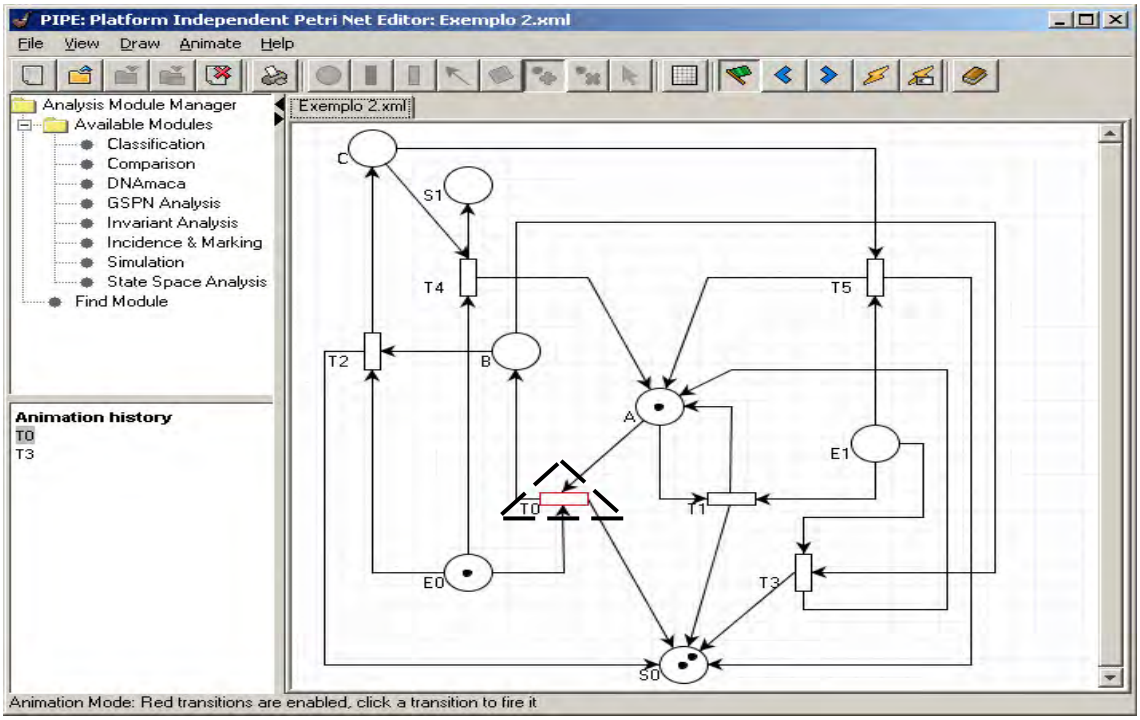


Figura 4.9: Transição T0 habilitada.

Com o disparo de T0 a RdP possuirá uma nova marcação. Na figura 4.10 nota-se que o lugar B possui uma marca e S0 três marcas.

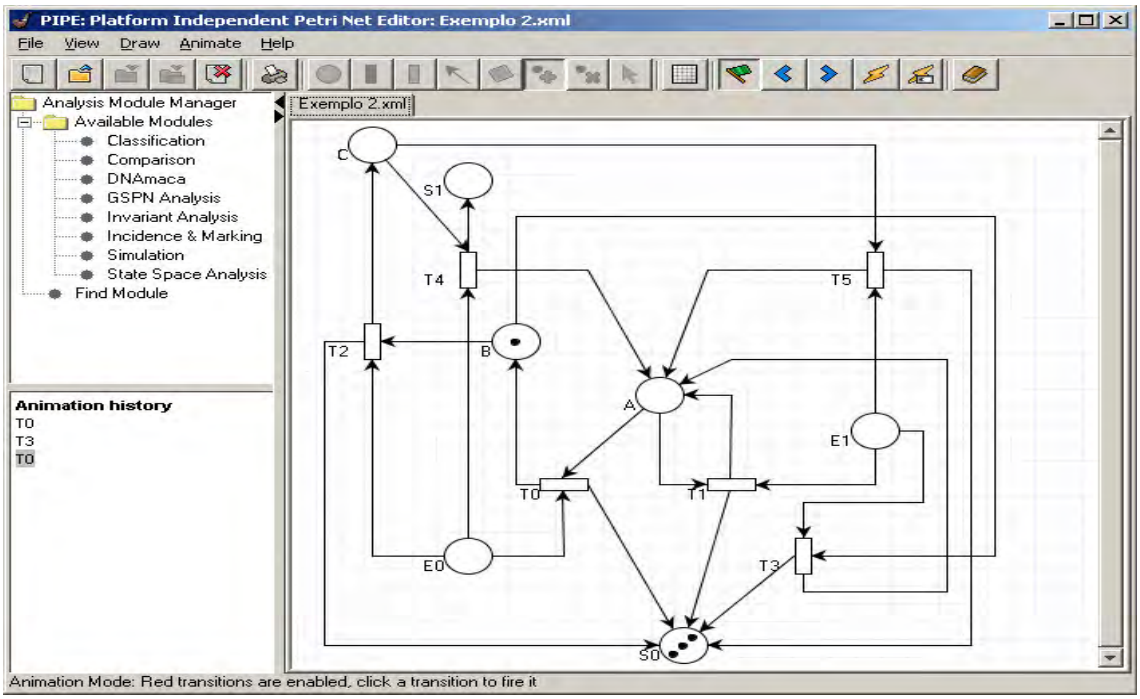


Figura 4.10: Marcação nos estados B e S0 (saída 0) após o disparo de T0.

Seguindo aos passos anteriores e inserindo consecutivamente as entradas 0 e 0 e disparando as transições T2 e T4, uma marca será inserida em cada um dos lugares S0 e S1, respectivamente. A figura 4.11 mostra o resultado da simulação para a seqüência de entradas 01000, definida pelo usuário.

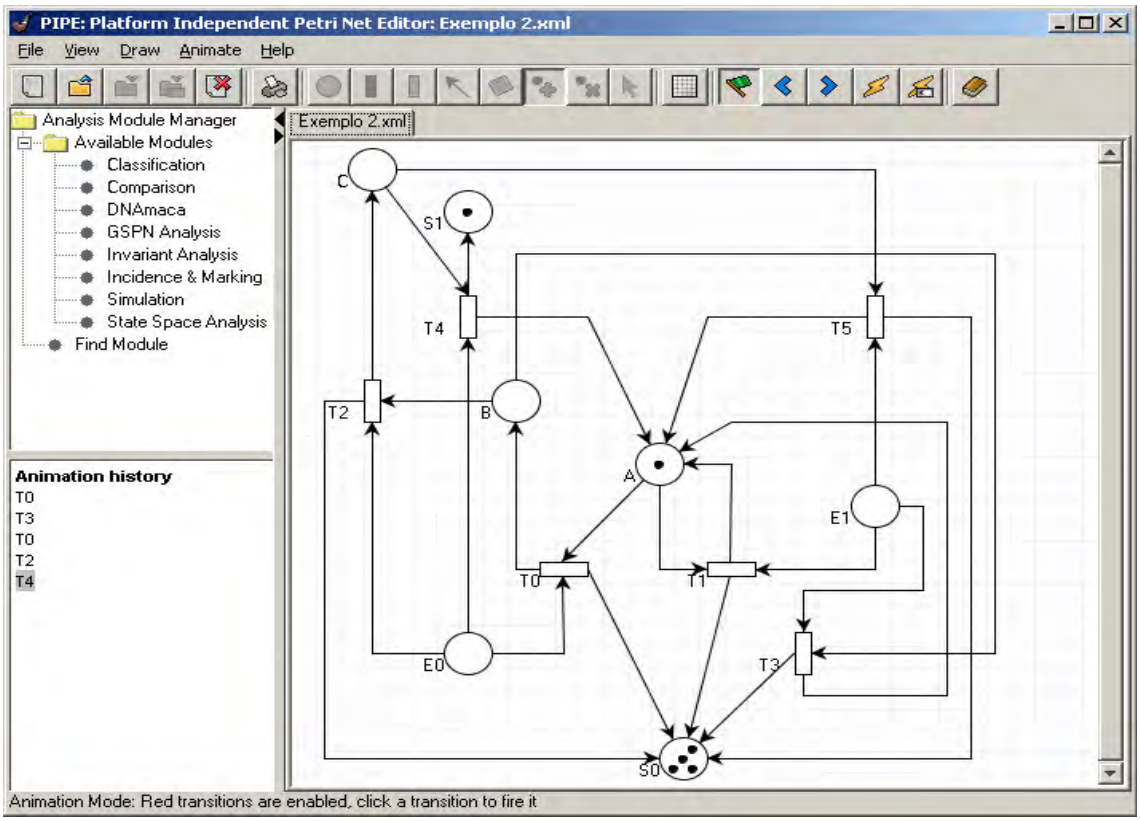


Figura 4.11: Simulação final realizada pelo usuário para a seqüência 01000.

A nova marcação da RdP é constituída por uma marca nos lugares S1 e A e quatro marcas no lugar S0, assim a simulação da RdP obteve uma saída de valor lógico 1 e quatro de valor lógico 0.

O usuário possui outro modo de simulação, onde uma quantidade de marcas é inserida em cada lugar, de acordo com a quantia de cada entrada desejada. A marcação resultante habilitará uma ou mais transições, assim o usuário poderá escolher a que deseja disparar de acordo com a entrada que a habilita, clicando sobre a transição escolhida. Considerando a marcação da figura 4.3, tem-se que as transições T0 e T1 estão habilitadas, se o usuário desejar que a entrada tenha valor lógico 0 ele disparará T0 caso contrário T1.

4.2 - Metodologia Desenvolvida para a Modelagem de Máquinas de Mealy ou Moore

Com o estudo de alguns casos, observou-se que a metodologia descrita anteriormente era inadequada para modelar FSM com muitos estados, pois eram utilizados muitos arcos para ligar as informações necessárias para realizar apenas uma "ação" do sistema especificado. Por exemplo, para modelar uma máquina simples como a da figura 4.1 com apenas três estados uma entrada e uma saída são necessários 7 lugares, 6 transições e 24 arcos para conectarem lugares a transições e vice-versa. Assim a metodologia 4M é inviável para modelar uma máquina com 32 estados, 1 entrada e 2 saídas (caso do HDB3 - *High Desinty Bipolar 3*[40]), pois seriam necessários 38 lugares, 64 transições e 128 arcos.

Sendo assim houve a necessidade de desenvolver uma nova metodologia que minimizasse a quantidade de lugares e arcos, facilitando a modelagem de máquinas com várias entradas, saídas e estados. Nesta seção apresenta-se a metodologia 5M (Metodologia para Modelagem de Máquinas de Mealy ou Moore), desenvolvida com esta finalidade.

4.2.1 - Metodologia Desenvolvida para a Modelagem das Máquinas de Mealy

Na metodologia 5M a modelagem das máquinas de Mealy difere da metodologia 4M, apresentada na seção anterior, apenas na representação das saídas do sistema. Na 4M as saídas são representadas por lugares na RdP, enquanto na 5M são representadas nas transições da rede, pelo simples fato da saída da máquina de Mealy depender do estado atual e do valor da entrada.

Nesta metodologia os estados e as entradas da FSM do tipo Mealy são representados por lugares na RdP.

Tem-se que o estado atual e a entrada da máquina (representados por lugares na RdP) são pré-condições para o disparo da transição e o próximo estado sua pós-condição. A representação da transição se dá pelos caracteres “TX/SX”, “TX” indica que o componente da rede é uma transição, o caractere X que acompanha o “T” corresponde ao número da transição, este é determinado pelo ambiente PIPE, obedecendo a ordem que a transição foi inserida na modelagem, ou seja, a primeira transição inserida é T0, a segunda T1 e assim por diante. Os caracteres “SX” representam a saída, o caractere X será substituído por um caractere numérico que corresponde ao valor lógico que representará o valor da saída. A barra (/) é utilizada na terminologia de cada transição com o propósito de separar a representação da transição e saída no modelo da RdP.

As transições na modelagem são responsáveis por realizar eventos na RdP, elas são habilitadas conforme o estado atual e a entrada da máquina. Assim, quando uma transição dispara, uma marca é retirada simultaneamente dos lugares que representam o estado atual e a entrada, sendo uma nova marca inserida no lugar que representa o próximo estado, que após o disparo passará a ser o estado atual da máquina.

Aplicando-se a metodologia de modelagem 5M para a máquina que detecta a entrada de três zeros consecutivos sem sobreposição, obtém-se a RdP mostrada na figura 4.12. Note-se que houve uma redução de cerca de 22% de elementos gráficos desta rede em relação aos daquela da figura 4.2.

Define-se como elementos gráficos de uma RdP todos os lugares, transições e arcos que a representam.

Os lugares A, B e C, da figura 4.12, representam os estados da máquina da figura 4.1 e os lugares E0 e E1 indicam as entradas de valores lógicos 0 e 1 respectivamente. A transição fornece a saída da máquina de acordo com estado atual e a entrada. A transição T1/S0 que conecta os lugares A e E0 ao lugar B corresponde ao arco da figura 4.1 que sai do

estado A com destino ao estado B, tendo como entrada e saída o valor lógico 0. Da mesma forma a transição T0/S0 que conecta os lugares B e E1 ao lugar A corresponde ao arco da figura 4.1 que sai do estado B para o A, tendo como entrada o valor lógico 1 e saída o valor 0. Assim cada transição da RdP corresponde a um arco do diagrama de estados.

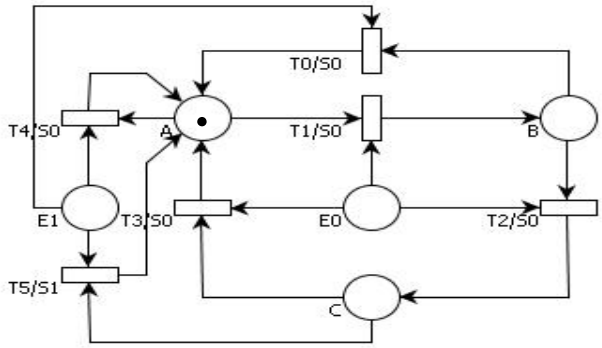


Figura 4.12: Modelagem 5M para a máquina de Mealy do detector de três zeros consecutivos sem sobreposição.

Observa-se, na figura 4.12, que para a transição T1/S0 ser habilitada é necessário que existam marcas nos lugares A e E0, sendo esta a pré-condição para o disparo de T1/S0. Após o disparo dessa transição uma marca deverá ser inserida no lugar B que passará a representar o estado atual da máquina, assim B é pós-condição do disparo de T1/S0. Todas as transições encontradas na figura 4.12 possuem dois lugares como pré-condições e um lugar como pós-condições de disparo.

A modelagem da máquina de Mealy em RdP é utilizada pelos programas PIPE2TAB5M e PIPE2VHDL5M, responsáveis por gerar a tabela de estados e a descrição VHDL da máquina modelada, desta forma, foi necessário estabelecer os seguintes critérios para modelagem das máquinas de Mealy:

a. Os estados e entradas correspondentes à FSM do tipo Mealy devem ser representados por lugares distintos na RdP;

b. As nomeações dos lugares que representam os estados da máquina não devem possuir o caractere “E”, pois este é utilizado exclusivamente na nomeação dos lugares que indicam as entradas da máquina;

c. As nomeações dos lugares que representam as entradas da máquina devem obrigatoriamente possuir o caractere “E”, como primeiro caractere, seguido do número decimal que representa o valor da entrada, por exemplo, “E1” representa a entrada de valor lógico 1;

d. A saída da FSM do tipo Mealy deve ser representada na transição da RdP, conforme seus lugares de entrada e saída, que representam o estado atual, a entrada e o próximo estado da máquina. A nomeação da transição (saída) obrigatoriamente deve possuir o caractere “TX” (X corresponde ao número da transição inserida) nos primeiros caracteres e em seguida os caracteres “/SX” (X representará o valor lógico da saída na forma decimal), por exemplo, “T5/S1” representa a sexta transição (T5) inserida no grafo da RdP modelado no ambiente PIPE, onde a saída de valor lógico 1 é indicada por S1;

e. Os valores lógicos que representam as entradas e saídas da máquina devem estar expressos na forma decimal;

f. O estado inicial da FSM deve ser destacado acrescentando-se somente uma marca no lugar que o representa na RdP.

4.2.2 - Metodologia Desenvolvida para a Modelagem das Máquinas de Moore

A metodologia 5M, desenvolvida para a modelagem das máquinas de Moore utiliza o tipo RdP Lugar/Transição para modelar o funcionamento desse tipo de FSM. Nas máquinas de Moore a saída depende somente do estado atual da máquina, assim sua modelagem difere das máquinas de Mealy.

O diagrama de estados é uma forma de representar FSM do tipo Moore, na figura 4.13 é apresentado um exemplo de máquina de Moore, para o detector de três zeros

consecutivos sem sobreposição. O estado inicial da máquina é representado pelo caractere A, este sempre fornece a saída de valor lógico 0. Se o caractere 1 chegar à entrada, e a máquina estiver no estado A ela continuará no mesmo. Caso esteja em A e chegue à entrada um caractere 0 a máquina vai para o estado B que sempre fornecerá saída 0. Se a próxima entrada do estado B for 0 a máquina vai para o estado C que fornece saída 0. Estando em C e a entrada for novamente 0 a máquina vai para o estado D que tem como saída o valor lógico 1.

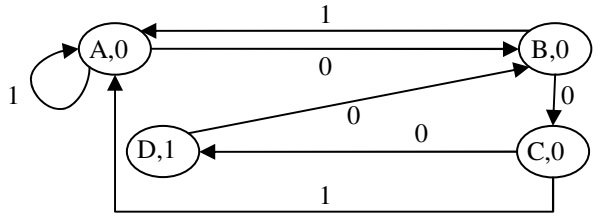


Figura 4.13: Máquina de Moore para o detector de três zeros consecutivos sem sobreposição.

Observa-se, na figura 4.13, que as evoluções dos estados ocorrem de acordo com a entrada, sendo que determinado estado sempre fornece uma saída fixa.

Para modelar as máquinas de Moore, utilizando a metodologia 5M é necessário representar seus estados e entradas por lugares distintos na RdP. As saídas são indicadas juntamente com os lugares que representam os estados da máquina.

A figura 4.14 mostra a modelagem em RdP Lugar/Transição para a FSM apresentada na figura 4.13. Observa-se que os lugares que representam os estados (A, B, C, D) da máquina são nomeados pelos mesmos caracteres alfabéticos, acompanhados pelos caracteres (/S) e pela saída daquele estado, por exemplo, o lugar A/S0 da RdP ilustrada na figura 4.14, representa o estado A da máquina que tem como saída o valor lógico 0, assim o S0 indica a saída e a barra (/) separaram as representações do estado da saída.

Os lugares B/S0 e C/S0 correspondem aos estados B e C apresentados na figura 4.13, que fornecem saída 0. Sendo que o estado D, que fornece saída 1, é representado na RdP pelo lugar D/S1.

As entradas são modeladas na RdP pelos lugares E0 (entrada com valor lógico 0)

e E1 (entrada com valor lógico 1).

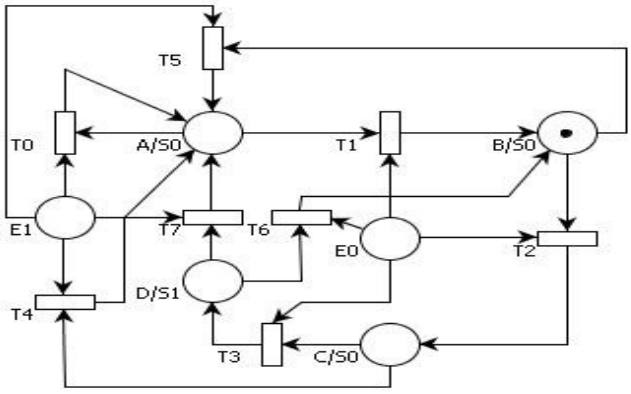


Figura 4.14: Modelagem em RdP para a máquina de Moore do detector de três zeros consecutivos sem sobreposição.

O lugar que representa o estado inicial da máquina é identificado pela inserção de uma marca.

As transições na RdP são responsáveis por executar as ações ou mudanças de estado da máquina de Moore. Os arcos conectam os lugares que representam o estado "atual" e a entrada a transição e à transição ao "próximo" estado do sistema modelado. O disparo das transições ocorre de acordo com estado atual em que a máquina se encontra e a entrada fornecida. Na figura 4.14 observa-se, por exemplo, que a transição T5 é habilitada quando existir uma marca no lugar B/S0 e no mínimo uma marca no lugar E1 que representa a entrada de valor lógico 1. Após o disparo da transição T5 é marcado o lugar que representa o próximo estado da máquina de Moore, neste caso o lugar marcado é o A/S0. Assim os lugares que indicam o estado atual e a entrada são pré-condições para o disparo da transição e o lugar que indica o próximo estado pós-condição.

Foi necessário estabelecer os seguintes critérios para a metodologia de modelagem 5M, pois a descrição da RdP gerada pelo modelo é utilizada pelos programas PIPE2TAB5M e PIPE2VHDL5M:

a. Cada um dos estados e entradas correspondentes à FSM do tipo Moore devem ser representados por lugares distintos na RdP;

b. Os nomes dos lugares que representam os estados não devem possuir o caractere “E”, pois este caractere é utilizado exclusivamente para nomear os lugares que representam as entradas da máquina e não os estados;

c. Os nomes dos lugares que representam as entradas da máquina devem obrigatoriamente possuir o caractere “E”, como primeiro caractere e depois um número decimal que indica o valor lógico da entrada, por exemplo, E0;

d. A saída da FSM do tipo Moore deve ser indicada no mesmo lugar que seu estado que a gera é representado. Os nomes dos lugares que representam o estado e a saída devem obrigatoriamente possuir os caracteres “/S”, depois do nome do estado indicado. A barra foi definida na metodologia de modelagem para separar a representação do estado e da saída da máquina, por exemplo, “A/S0” representa o estado A com saída de valor lógico 0. Sendo que a o valor lógico da saída tem que estar obrigatoriamente em número decimal;

e. Não é necessário que os lugares que representam os estados da máquina sejam nomeados apenas com caracteres alfabéticos;

f. O estado inicial da FSM deve ser destacado, acrescentando-se somente uma marca no lugar que o representa na RdP;

As restrições impostas para a metodologia de modelagem 5M são essenciais para a execução dos programas, sem nenhum problema.

4.3 - Comparação entre as Metodologias de Modelagem desenvolvidas

A quantidade de lugares para representar uma mesma FSM do tipo Mealy na metodologia 5M é inferior ao da 4M, visto que na 4M as saídas da máquina são representadas por lugares distintos na RdP e na 5M as saídas são representadas nas transições (máquinas de

Mealy) ou nos lugares (Máquinas de Moore). Conseqüentemente, a quantidade de arcos é inferior, pois as saídas não são mais representadas por lugares distintos, não necessitando assim de arcos que realizem a ligação entre as transições e os lugares de saída (saída da FSM).

Na metodologia 5M é fácil distinguir o tipo de FSM modelada, pois se a saída estiver representada no mesmo lugar (RdP) que o estado da máquina modelada, tem-se uma máquina do tipo Moore e se a saída estiver representa junto as transições da RdP a máquina modelada será do tipo Mealy. Visto que apenas a metodologia 5M modela máquinas de Mealy e de Moore.

Como na metodologia 5M não há um lugar específico que represente a saída da máquina, torna-se mais difícil identificar a quantidade de saídas que cada valor lógico obteve durante a simulação. Para obter esta informação é necessário analisar o histórico das transições disparadas.

Na modelagem das máquinas de Mealy é possível saber a quantidade de saídas, contabilizando as saídas indicadas no histórico do ambiente PIPE, visto que as saídas são representadas juntamente com as transições. O PIPE produz o histórico de todas as transições disparadas durante a simulação da rede.

Para identificar as saídas das máquinas de Moore modeladas em RdP, utilizando a metodologia 5M, é mais complicado que a identificação das máquinas de Mealy, pois é necessário observar as transições disparadas, referenciadas no histórico do ambiente PIPE e analisar na RdP qual é o lugar que cada transição marca após o seu disparo, este lugar representa o estado e a saída da máquina.

Apresenta-se na seção posterior os programas implementados com o intuito de analisar as descrições obtidas pelas metodologias de modelagem 4M e 5M.

5 DESCRIÇÃO DO FUNCIONAMENTO DOS PROGRAMAS DESENVOLVIDOS

Descrevem-se nesta seção o funcionamento dos programas PIPE2TAB4M, PIPE2VHDL4M, PIPE2TAB5M e PIPE2VHDL5M. Estes programas são considerados ferramentas de síntese digital, pois possibilitam que as FSM modeladas em RdP Lugar/Transição, por intermédio das metodologias de modelagem 4M ou 5M, sejam representas em tabela de transição de estados ou em descrição VHDL comportamental.

Todos os programas foram desenvolvidos em linguagem de programação Python, por esta ser gratuita, portátil, de tipagem dinâmica (não é necessário declarar variáveis), de sintaxe clara e limpa.

5.1 Programas Desenvolvidos para a Metodologia 4M

Nesta subseção apresentam-se os programas denominados PIPE2TAB4M e PIPE2VHDL4M, implementados para analisar a descrição PNML do modelo da RdP, gerado pela modelagem da rede no ambiente PIPE, utilizando-se da metodologia de modelagem 4M.

5.1.1 Programa PIPE2TAB4M

O programa PIPE2TAB4M gera a tabela de transição de estados, através da descrição PNML que representa a FSM modelada em RdP Lugar/Transição pela metodologia 4M.

Quando se executa o programa e, conseqüentemente, são passados os argumentos solicitados, tais como endereço do arquivo de entrada e de saída, o programa abre o arquivo de entrada para analisar a descrição PNML e cria o arquivo de saída para salvar a tabela gerada. Caso ocorra algum erro durante o processo de abertura, criação de arquivos ou na análise dos parâmetros do arquivo de entrada, o programa é encerrado sem realizar a geração da tabela e fornece a mensagem do referido erro para o usuário.

Se não houver nenhum erro nos parâmetros o arquivo de entrada é analisado. Os lugares descritos no código PNML dentro das etiquetas <*place id* = "rótulo do lugar"> e </*place*>, são os primeiros componentes analisados, pois pela identificação do valor (nomeação atribuída aos lugares durante a modelagem) dos seus nomes, encontrado dentro das etiquetas <*value*> e </*value*>, é possível computar as entradas, saídas e estados existentes na FSM modelada em RdP Lugar/Transição.

Na figura 5.1 é ilustrado o trecho do código PNML que descreve os lugares da RdP apresentada na figura 4.2. Observa-se na figura 5.1 que os lugares P2, P4 e P5 representam os estados da máquina, pois os valores atribuídos a esses lugares são A, B e C respectivamente. Enquanto que os lugares P0 e P1 representam as entradas de valores lógicos 0 e 1, nomeados na descrição pelos valores E0 e E1. As saídas são representadas pelos lugares P3 e P6, observa-se que estes possuem na descrição os valores S0 e S1.

O número mínimo de *flip-flops* é calculado de acordo com a quantidade de estados existentes no modelo da RdP. Nota-se na figura 5.1 que a RdP representada na descrição possui apenas três estados (A, B e C) assim a quantidade mínima de *flip-flops* é dois para este caso específico.

Após, identificar a quantidade de estados existentes no modelo da RdP e calcular o número mínimo de *flip-flops* necessários, é solicitado ao usuário que indique o tipo de todos os *flip-flops* que se deseja utilizar. A escolha fica a critério do usuário.

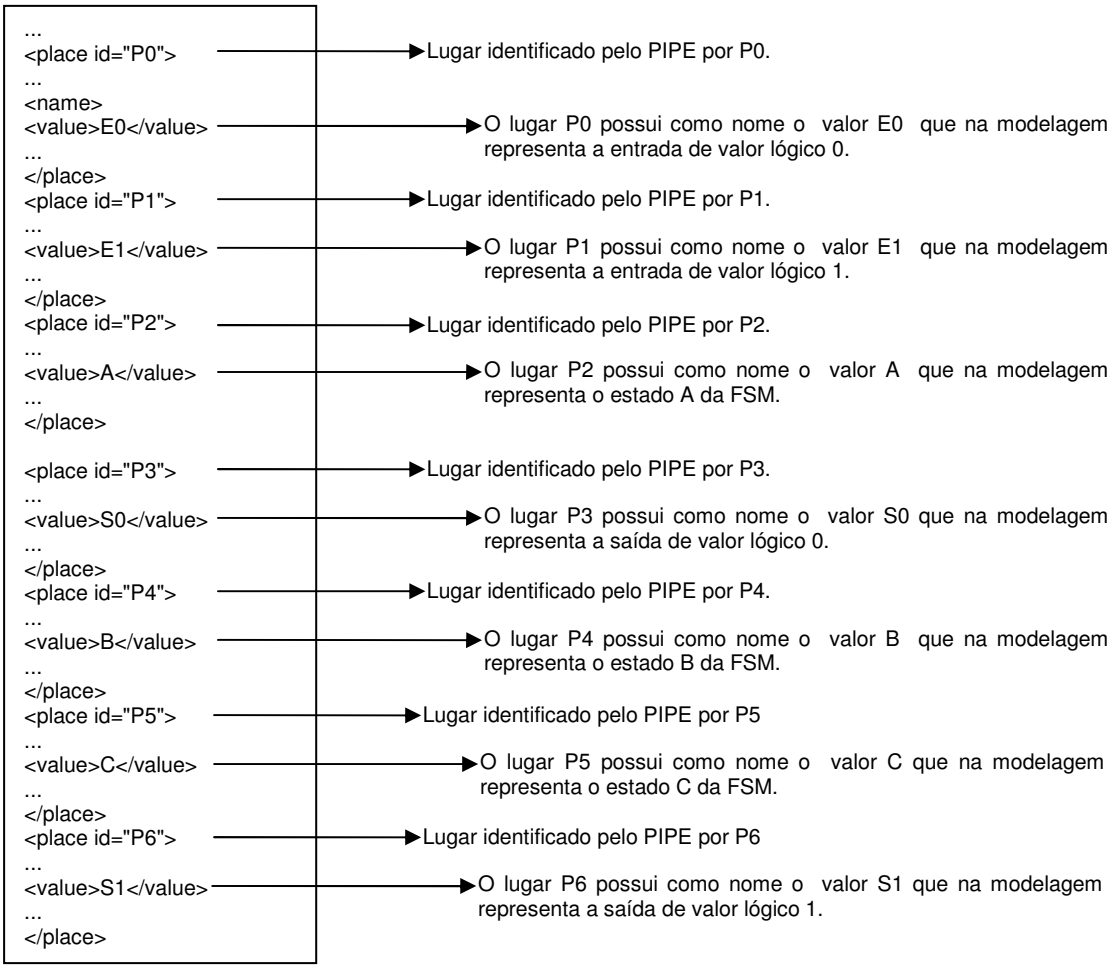


Figura 5.1: Trecho do código PNML onde estão descritos os lugares da RdP modelada pelo ambiente PIPE.

Os arcos descritos no código PNML nas etiquetas *<arc...> e </arc>,* são utilizados para obter a tabela de transição de estados, pois através da descrição dos arcos é possível detectar o estado atual, próximo estado, entrada e saída da FSM. O estado atual e a entrada são localizados nos arcos que representam a ligação dos lugares às transições, identificados no código PNML pelo trecho *id*=”rótulo do lugar” *to* “ rótulo da transição”, sendo que o próximo estado e a saída são localizados nos arcos que realizam a ligação das transições aos lugares, identificados no trecho de código PNML *id*="rótulo da transição” *to* “rótulo do lugar". Analisando, identificando e processando esses componentes, a tabela de transição da FSM é gerada. A fim de exemplificar a identificação da entrada, saída, próximo e estado atual da FSM modelada pela RdP da figura 4.2 é apresentada na figura 5.2 a parte do código PNML que descreve alguns arcos da rede. Observa-se na figura 5.2 que os lugares de

entrada da transição T0 são P0 e P2, que na modelagem representam os lugares E0 (entrada de valor lógico 0) e A (estado A da FSM). Sendo que os lugares de saída dessa mesma transição são P3 e P4, que correspondem na modelagem S0 (saída de valor lógico 0) e B (estado B da FSM). Assim para este caso temos que A é o estado atual, B o próximo estado, 0 o valor lógico da entrada e da saída, para a FSM modelada pela RdP da figura 4.2.

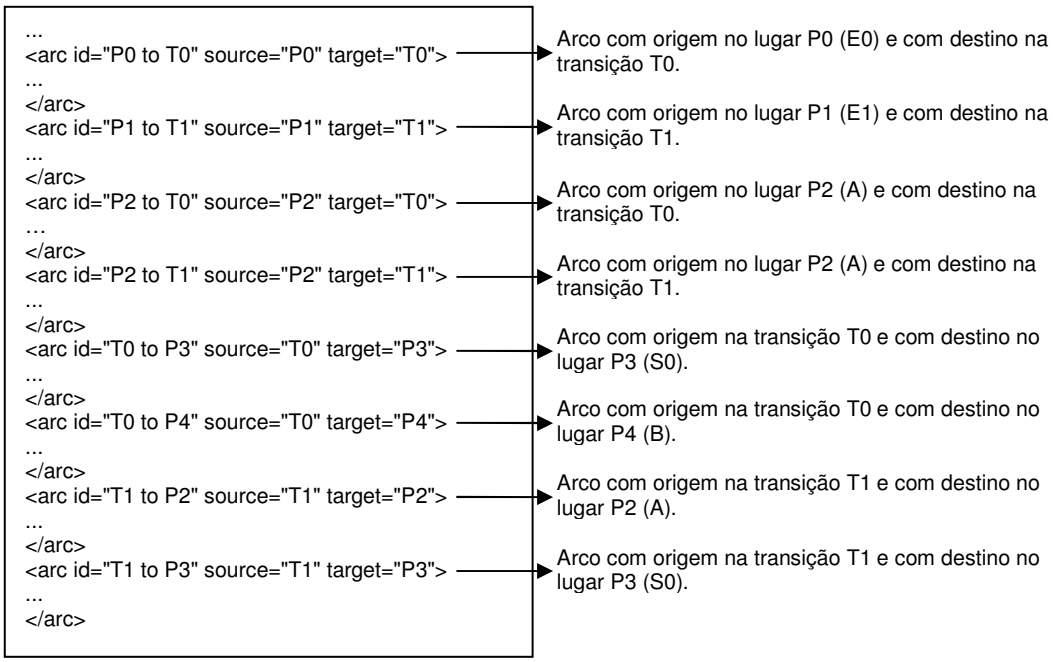


Figura 5.2: Trecho do código PNML onde estão descritas os arcos da RdP modelada pelo ambiente PIPE.

Como estamos trabalhando com um código XML, a análise do código se torna mais rápida e fácil, pois não precisamos criar um analisador para verificar cada caractere do código, basta apenas identificar as etiquetas que contêm as informações necessárias para o programa e capturar as informações existentes dentro delas.

O arquivo gerado pelo PIPE2TAB4M é utilizado como arquivo de entrada pelo programa AGPS e o arquivo de saída do programa AGPS é utilizado no TABELA, este último gera o arquivo que é usado pelo programa TAB2VHDL. Desta forma, o programa desenvolvido possibilita que os sistemas digitais especificados em alto nível de abstração, mas especificamente em RdP, sejam descritos em nível inferior, ou seja , em descrição RTL.

Os dados solicitados pelo programa PIPE2TAB4M são:

a. Estrutura de diretório (Diretório:\Pasta\Nome do arquivo.extenção) do arquivo de entrada, para captura dos dados ;

b. Estrutura de diretório do arquivo de saída para armazenar os resultados gerados pelo programa;

c. Tipo de cada *flip-flop* que se deseja utilizar. Os *flip-flops* utilizados são do tipo D e JK, sendo este último representado apenas pelo caractere J.

O diagrama de blocos do programa PIPE2TAB4M é mostrado na figura 5.3. Notam-se no diagrama que os retângulos de cantos arredondados representam apenas o início e o fim da execução do programa, os retângulos com o canto esquerdo cortado (cartão) indica os dados solicitados pelo programa ao usuário e a leitura dos arquivos de entrada, este arquivo é representado pelo componente do fluxograma que indica documentos. Os processamentos dos dados realizados pelo PIPE2TAB4M, como cálculos e construção de arquivo (saída) são representados no diagrama por retângulos normais. O programa PIPE2TAB4M utiliza como arquivo de entrada o documento que contém a descrição PNML da RdP modelada no ambiente PIPE, utilizando a metodologia 4M e por intermédio deste arquivo gera o de saída, contendo a descrição da tabela de estados.

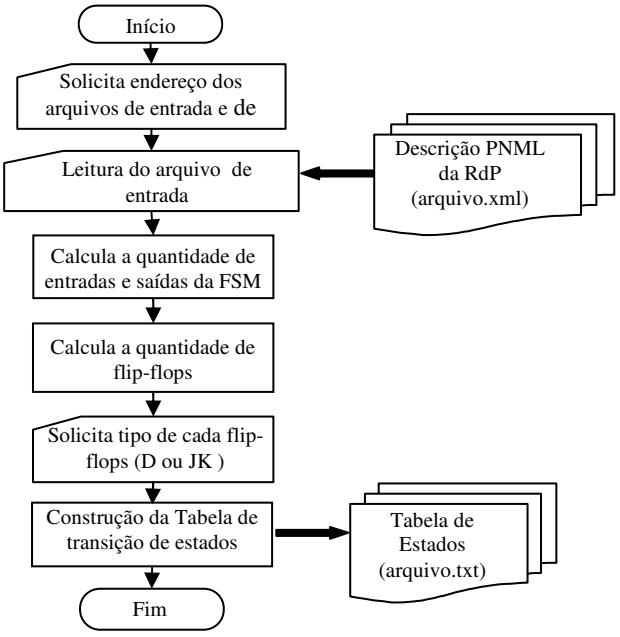


Figura 5.3: Diagrama de Blocos do programa PIPE2TAB4M.

Para executar o programa PIPE2TAB4M não é necessário que seu arquivo executável esteja no mesmo diretório do arquivo de entrada. Os passos necessários para executá-lo são os seguintes:

a. Abrir o *prompt* de comando (DOS);

b. Sair do comando raiz, entrar no diretório onde se encontra o programa e indicar o caminho, por exemplo, D:\MaterialAlexandre-Norian\Meus Programas-Python, em seguida teclar *Enter*;

c. Após o caminho identificado no *prompt* digitar c:\python25\python.exe PIPE2TAB4M.py. O caminho indicado é o local onde está salvo o arquivo executável do Python;

d. O programa será executado após o usuário teclar o *Enter* e os dados necessários para o funcionamento do programa serão requisitados.

Na figura 5.4 são apresentadas as etapas executadas no *prompt* do DOS para o usuário executar o programa PIPE2TAB4M e fornecer os dados de entrada requisitados pelo programa.

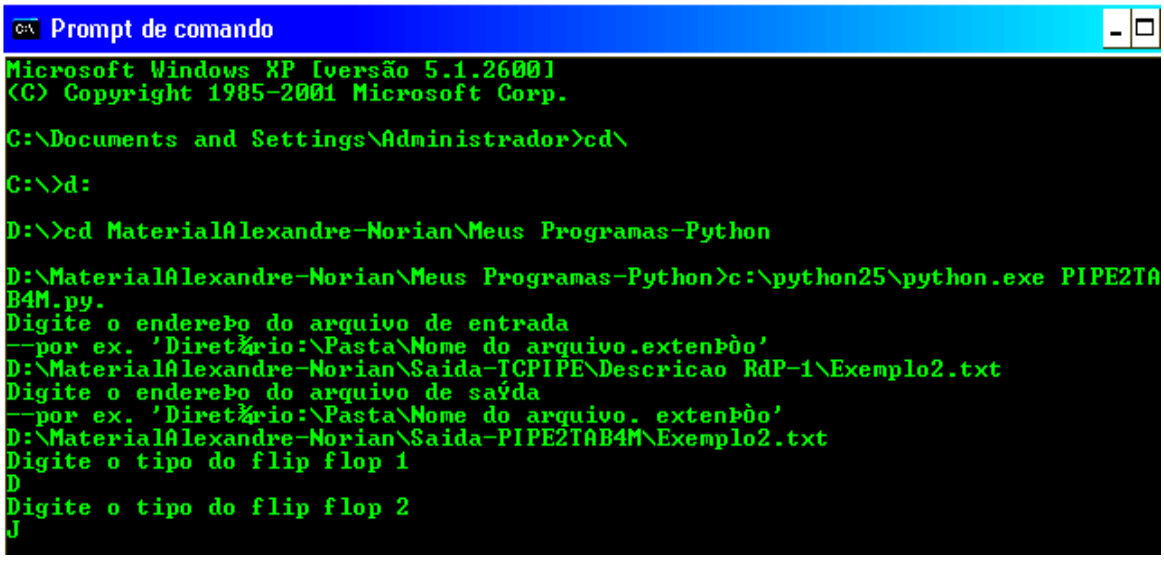


Figura 5.4: Ambiente do prompt DOS onde o programa PIPE2TAB4M foi executado.

O arquivo gerado pelo programa PIPE2TAB4M é apresentado na figura 5.5, este por sua vez descreve as informações da FSM, especificada pelo modelo da RdP, ilustrado na

figura 4.2, em forma de tabela. A referida FSM trata-se do detector de três zeros consecutivos sem sobreposição.

Figura 5.5: Arquivo gerado pelo programa PIPE2TAB4M para o detector de três zeros consecutivos sem sobreposição.

Na primeira linha do arquivo apresentado na Figura 5.5 especifica-se a quantidade mínima de elementos de memórias. Neste caso necessita-se no mínimo de dois elementos de memórias, visto que o sistema possui três estados. Nas duas linhas imediatamente abaixo se especificam o tipo de cada elemento que será utilizado, ou seja, o tipo dos *flip-flops*. Os escolhidos pelo usuário foram seqüencialmente do tipo D e JK para o 1º e 2º *flip-flop*. Especifica-se na quarta linha a quantidade de entradas e de saídas. O sistema projetado tem um porto de entrada e um porto de saída.

Nas linhas subseqüentes especificam-se as transições dos estados e a saída gerada em cada transição de acordo com a entrada fornecida para o sistema. Portanto, a partir da quinta linha, a primeira coluna representa o estado atual, a segunda o próximo estado, nas terceira e quarta colunas as respectivas entradas e saídas.

O arquivo gerado pelo programa PIPE2TAB4M é utilizado pelo programa AGPS, como arquivo de entrada, este por sua vez realiza a alocação de todos os estados da máquina. O arquivo de saída criado pelo AGPS está ilustrado na figura 5.6. Observa-se que o arquivo apresentado na figura 5.6 possui a mesma formatação do arquivo mostrado na figura 5.5, sendo que as informações descritas nas quatro primeiras linhas são idênticas nas duas figuras, a única alteração ocorre na alocação dos estados que são representados por caracteres

decimais e não mais por alfabéticos, ou seja, a tabela é modificada a partir da quinta linha, onde as alocações 1,3 e 0 foram efetuadas pelo programa AGPS para os estados A, B e C.

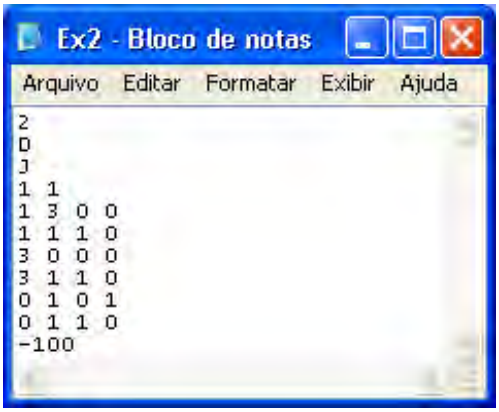


Figura 5.6: Arquivo gerado pelo programa AGPS

A utilização do AGPS auxilia o ambiente de síntese, pois o arquivo gerado por este é utilizado como entrada do programa TABELA, este por sua vez realiza a minimização das funções booleanas do sistema.

Na figura 5.7 está descrito o trecho do código de saída do programa TABELA. Nota-se que a primeira tabela gerada pelo programa é a de transição de estados, desta tabela são extraídas a função de controle dos elementos de memória e das saídas da máquina. Na primeira tabela a coluna DE apresenta os estados atuais, na coluna P/ os próximos estados, na MINT os mintermos e na Z_i as saídas da máquina, obtidas de acordo com os estados e entradas apresentados nas colunas DE e ENTRADA, as entradas são convertidas para valores binários. Os estados não representados na primeira tabela são indicados na tabela imediatamente abaixo, ou seja, na tabela dos *don't care state*.

O custo total para a implementação das quatro funções combinacionais (J1, K1, D0 e Z0) é igual a 8, considerando-se como critério de custo total a soma do custo final de cada função.

O arquivo gerado pelo programa TABELA é utilizado como entrada do programa TAB2VHDL, este por sua vez gera a descrição RTL na linguagem VHDL para o detector da seqüência de três zeros consecutivos sem sobreposição.

```
!DE!P/!ENTRADA !MINT!!Q1!Q1+!J1!K1!!Q0!Q0+!D0!!Z0!
! 0! 1! 1 (1)  !   4!! 0!  0! 0! X!! 0!  1! 1!! 0!
! 0! 1! 0 (0)  !   0!! 0!  0! 0! X!! 0!  1! 1!! 1!
! 3! 1! 1 (1)  !   7!! 1!  0! X! 1!! 1!  1! 1!! 0!
! 3! 0! 0 (0)  !   3!! 1!  0! X! 1!! 1!  0! 0!! 0!
! 1! 1! 1 (1)  !   5!! 0!  0! 0! X!! 1!  1! 1!! 0!
! 1! 3! 0 (0)  !   1!! 0!  1! 1! X!! 1!  1! 1!! 0!


DON'T CARE STATES GERAIS :
!DE!P/!ENTRADA !DONT!!Q1!Q1+!J1!K1!!Q0!Q0+!D0!!Z0!
! 2! X! 0 (0)  !   2!! 1! X ! X! X!! 0! X ! X!! X!
! 2! X! 1 (1)  !   6!! 1! X ! X! X!! 0! X ! X!! X!

FUNCAO J1
=========

MINTERMOS :  1;

DON'T CARE STATES :  3;  7;  6;  2;

IMPLICANTES PRIMOS ESSENCIAIS :

ESSENCIAL:  1     REDUNDANCIA:        2 ->  0X1

CUSTO FINAL DE J1 = 2

FUNCAO K1
=========

MINTERMOS :  3;  7;

DON'T CARE STATES :  1;  5;  0;  4;  6;  2;

IMPLICANTES PRIMOS ESSENCIAIS :

ESSENCIAL:  0     REDUNDANCIA:        7 ->  XXX

CUSTO FINAL DE K1 = 0

FUNCAO D0
=========

MINTERMOS :  1;  5;  7;  0;  4;

DON'T CARE STATES :  6;  2;

IMPLICANTES PRIMOS ESSENCIAIS :

ESSENCIAL:  4     REDUNDANCIA:        3 ->  1XX
ESSENCIAL:  0     REDUNDANCIA:        5 ->  X0X

CUSTO FINAL DE D0 = 4

FUNCAO Z0
=========

MINTERMOS :  0;

DON'T CARE STATES :  6;  2;

IMPLICANTES PRIMOS ESSENCIAIS :

ESSENCIAL:  0     REDUNDANCIA:        2 ->  0X0

CUSTO FINAL DE Z0 = 2


CUSTO TOTAL DAS 4 FUNCOES = 8
```

Figura 5.7: Descrição gerada pelo programa TABELA para o detector de três zeros consecutivos sem sobreposição.

O código RTL descrito pelo programa TAB2VHDL é apresentado na figura 5.8, com a finalidade de mostrar a descrição VHDL gerada pelo programa para o detector da seqüência de três zeros consecutivos sem sobreposição. Nota-se que na entidade da descrição, apresentada no código RTL da figura 5.8, foram definidos os portos de entrada (x0) e saída (z0) e os sinais de sincronismo (CLK e CLR). Os portos denominados Q0 e Q1 permitem ao projetista observar através da simulação, os estados que a máquina está transitando.

```
Arquivo  Editar  Formatar  Exibir  Ajuda
ENTITY Exemplo2 IS
    PORT(

    CLK, CLR : IN BIT;
    X0 : IN BIT;
    Q0, Q1 : OUT BIT;
    Z0 : OUT BIT );

END Exemplo2;

ARCHITECTURE RTL OF Exemplo2 IS
SIGNAL  VE0, VE1: BIT;
SIGNAL J1, K1, D0 : BIT;

 BEGIN
-- Inferindo flip-flop tipo D
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;
-- Inferindo flip-flop tipo JK
PROCESS(CLK, CLR)
BEGIN
   IF CLR = '0' THEN
      VE1 <= '0';
   ELSIF CLK'EVENT and CLK = '1' THEN
      IF J1 = '0' and K1 = '1' THEN
         VE1 <= '0';
      ELSIF J1 = '1' and K1 = '0' THEN
         VE1 <= '1';
      ELSIF J1 = '1' and K1 = '1' THEN
         VE1 <= not(VE1);
      END IF;
   END IF;
   Q1 <= VE1;
END PROCESS;

 J1 <=  ( NOT(X0) AND (VE0));
 K1 <=  '1';
 D0 <=  ( (X0)) OR (NOT(VE1));
 Z0 <=  ( NOT(X0) AND NOT(VE0));

END RTL;
```

Figura 5.8: Descrição RTL na linguagem VHDL gerada pelo programa TAB2VHDL.

Os sinais auxiliares, denominados V_{EN}, mostrados na descrição da figura 5.8 são definidos com o objetivo de evitar que se gerem vários *drivers* para um mesmo sinal.

Para cada *flip-flop* utilizado é criado um processo na descrição, dois tipos de processos foram criados na descrição da figura 5.8, um para modelar o *flip-flop* D e outro para modelar o *flip-flop* JK, ambos sensíveis a transição (borda) de subida.

Na figura 5.9 é apresentado o digrama de blocos referente ao processo de síntese digital, realizado com auxílio dos ambientes PIPE, Quartus II [42] ou Max+Plus [42] e dos programas PIPE2TAB4M, AGPS, TABELA e TAB2VHDL. Estes ambientes e programas permitem que sistemas modelados em alto nível de abstração, RdP, utilizando a metodologia

de modelagem 4M, sejam descritos em um nível inferior, mas especificamente em linguagens de descrição de *hardware* (modelo RTL em VHDL). No diagrama de blocos o retângulo de cantos arredondados representa a metodologia de modelagem utilizada para modelar FSM em RdP e os cubos representam os programas ou ambientes utilizados, o ambiente PIPE é empregado para a modelagem e simulação das RdP, o programa PIPE2TAB4M traduz o arquivo contendo a descrição PNML da RdP modelada pelo PIPE para uma tabela de transição de estados, o AGPS realiza a alocação dos estados da tabela gerada pelo PIPE2TAB4M, o arquivo gerado pelo AGPS é usado pelo TABELA para a minimização das funções booleanas, este por sua vez gera o arquivo que é utilizado pelo TAB2VHDL para a geração do modelo RTL em VHDL do sistema modelado em RdP. Por fim a descrição gerada pelo TAB2VHDL pode ser compilada e simulada nos ambientes da Altera.

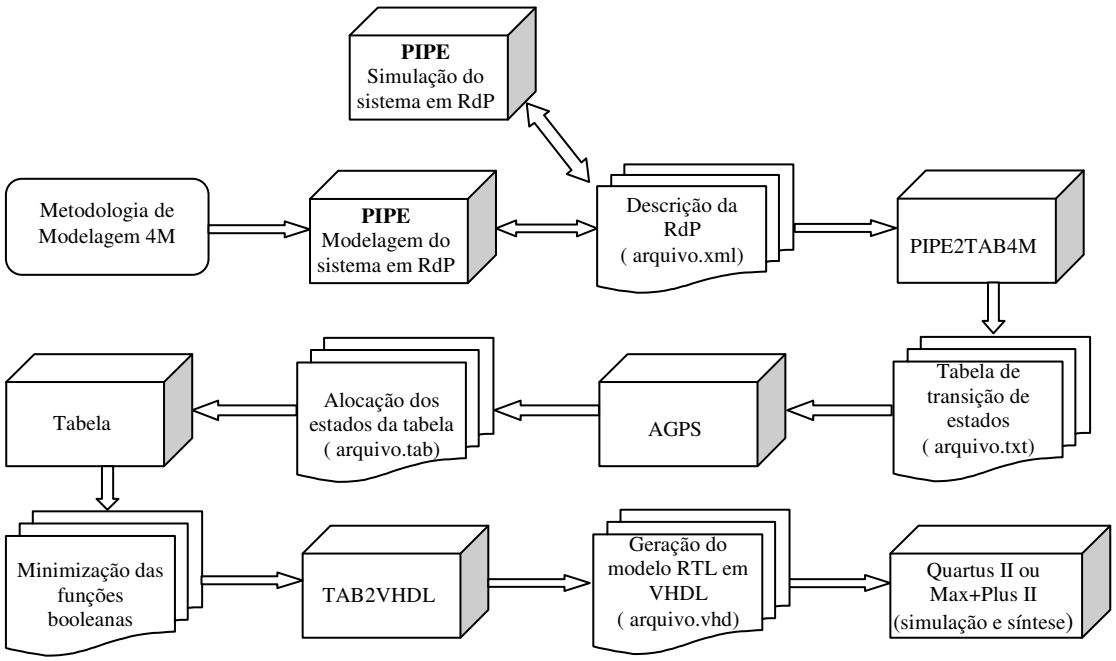


Figura 5.9: Diagrama de Blocos para o processo de síntese.

5.1.2 Programa PIPE2VHDL4M

O programa PIPE2VHDL4M gera automaticamente a descrição comportamental na linguagem VHDL, através da análise do código PNML. Este código é obtido pela

modelagem da FSM do tipo Mealy em RdP Lugar/Transição, utilizando-se do ambiente PIPE e da metodologia 4M.

Ao executar o programa e, conseqüentemente, fornecerem-se os argumentos solicitados, tais como endereço do arquivo de entrada e de saída, o programa abre o arquivo de entrada para analisar a descrição PNML e cria o arquivo de saída para salvar a descrição comportamental na linguagem VHDL gerada. Caso ocorra algum erro durante o processo de abertura ou criação de arquivos ou na análise dos parâmetros do arquivo de entrada, o programa é encerrado sem realizar a geração da descrição e fornece a mensagem do referido erro para o usuário.

Caso não houver nenhum erro nos parâmetros, o arquivo de entrada é analisado. Os lugares descritos no código PNML nas etiquetas <*place id* = "rótulo do lugar"> *e* <*/place*>*,* são os primeiros componentes analisados, pois pela identificação do "valor" dos seus nomes, encontrado dentro das etiquetas <*value*> e <*/value*>, é possível computar a quantidade de entradas, saídas e estados existentes na FSM modelada em RdP Lugar/Transição. Assim as entradas e saídas representadas no modelo por valores decimais são convertidas para valores binários e armazenadas no arquivo de saída dentro da entidade, representada no trecho de código *ENTITY* da descrição VHDL*,* como portas do tipo *bit*.

Após, definidas as entradas e saídas, é solicitado ao usuário que indique o nome da arquitetura que deseja utilizar no código VHDL a ser gerado.

Os estados identificados na análise dos lugares são declarados como tipo "tipo-estado", dentro da arquitetura representada pela etiqueta *ARCHITECTURE.*

Os arcos descritos no código PNML nas etiquetas <*arc...*> e <*/arc*>*,* são utilizados para descrever o processo do código VHDL, apresentado no trecho de código *PROCESS (Atual,* x_i*)* e *END PROCESS*, dentro deste são descritas as transições dos estados.

Através da descrição dos arcos é possível detectar o estado atual, próximo estado, entrada e saída.

Na figura 5.10 é ilustrado o diagrama de blocos para o programa PIPE2VHDL4M. Os retângulos de cantos arredondados da figura 5.10 representam apenas o início e o fim da execução do programa. Nos retângulos com o canto esquerdo cortado (cartão) são indicadas as solicitações de dados ao usuário pelo programa e a leitura dos arquivos de entrada, este arquivo é representado pelo componente de um fluxograma que indica documentos. O processo para a conversão das entradas e saídas para valores binários e a construção da descrição comportamental VHDL do sistema modelado, são representados no diagrama por retângulos normais. O programa PIPE2VHDL4M utiliza como arquivo de entrada o documento que contém a descrição PNML da RdP modelada no ambiente PIPE, utilizando a metodologia 4M e por intermédio deste arquivo gera o de saída, contendo a descrição comportamental na linguagem VHDL.

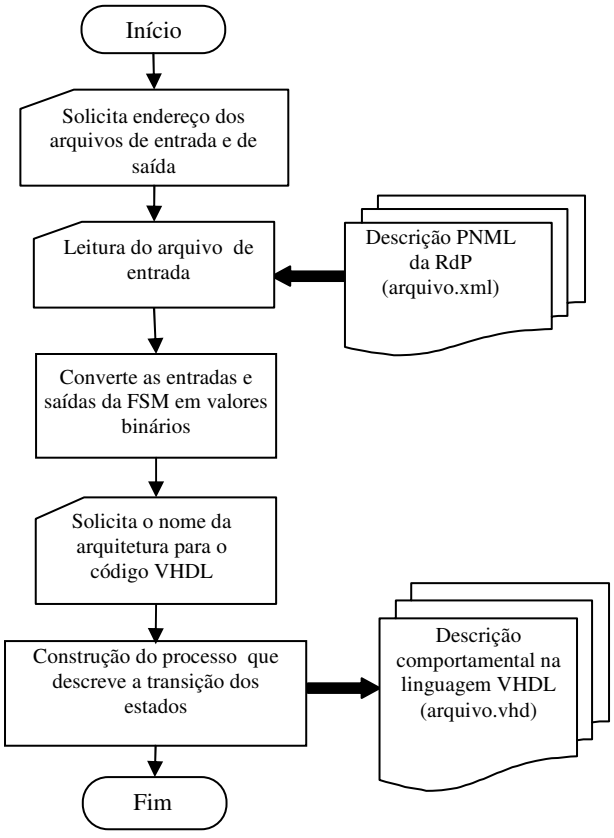


Figura 5.10: Diagrama de Blocos do programa PIPE2VHDL4M.

Ao executar o programa PIPE2VHDL4M não é necessário que seu executável esteja na mesma pasta do arquivo utilizado como entrada.

As seguintes etapas são necessárias para executar o PIPE2VHDL4M:

a. Abrir o DOS;

b. Sair do comando raiz, entrar no diretório em que o programa se encontra e indicar o caminho, por exemplo, D:\Material Alexandre- Norian\Meus Programas-Python, em seguida teclar *Enter*;

c. Digitar o caminho para executar o Python e nome do programa que será executado, c:\python25\python.exe PIPE2VHDL4M.py;

d. Após digitar os comandos para executar o PIPE2VHDL4M, os seguintes dados serão requisitados para o usuário:

- Endereço do arquivo de entrada para captura dos dados, por exemplo, D:\MaterialAlexandre-Norian\Saida-TCPIPE\DescriçãoRdP-\Exemplo2.txt;

- Endereço do arquivo de saída para armazenar a descrição comportamental na linguagem VHDL gerada pelo programa;

- Nome da arquitetura para a descrição VHDL.

A figura 5.11 ilustra as etapas recém descritas, executadas no *prompt* do DOS para rodar o programa PIPE2VHDL4M e fornecer os dados de entrada requisitados para o usuário.

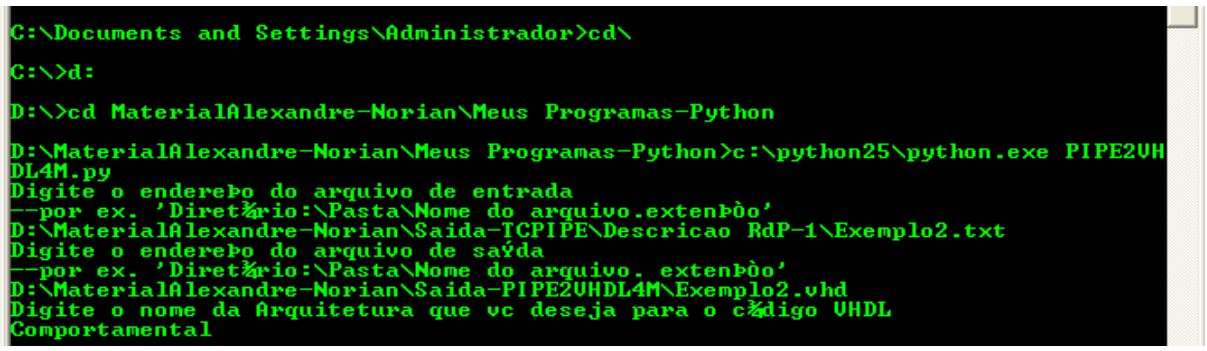


Figura 5.11: Ambiente do *prompt* DOS onde o programa PIPE2VHDL4M foi executado.

Na figura 5.12 é exibida a descrição comportamental em linguagem VHDL, gerada pelo programa PIPE2VHDL4M para o detector da seqüência de três zeros consecutivos sem sobreposição. Observa-se que os portos de entrada e saída e o sinal de sincronismo (*clock*) são definidos dentro da entidade na descrição comportamental, apresentada na figura 5.12. Os portos denominados x0 e z0 representam as entradas e saídas da máquina modelada em RdP. Os estados da máquina são representados dentro da arquitetura como um tipo de variável "tipo_estado".

Arquivo Editar Formatar Exibir Ajuda

```
ENTITY Exemplo2 IS
        PORT
        (
                --xn representão os sinais de entrada
                --zn representão os sinais de saída
                clock:IN BIT;
                x0:IN BIT;
                z0:OUT BIT
        );
END Exemplo2;

ARCHITECTURE Comportamental OF Exemplo2 IS

        TYPE tipo_estado IS (A, B, C);
        SIGNAL Atual, Proximo: tipo_estado;
BEGIN

        PROCESS (Atual, x0)

        BEGIN
                CASE Atual IS
                        WHEN A =>
                                IF x0= '0' THEN
                                        z0<= '0';
                                        Proximo <=B;
                                ELSE
                                        z0<= '0';
                                        Proximo <=A;
                                END IF;
                        WHEN B =>
                                IF x0= '0' THEN
                                        z0<= '0';
                                        Proximo <=C;
                                ELSE
                                        z0<= '0';
                                        Proximo <=A;
                                END IF;
                        WHEN C =>
                                IF x0= '0' THEN
                                        z0<= '1';
                                        Proximo <=A;
                                ELSE
                                        z0<= '0';
                                        Proximo <=A;
                                END IF;
                END CASE;
        END PROCESS;

        PROCESS
        BEGIN
                WAIT UNTIL clock'Event AND clock = '1';
                        Atual <= Proximo;
        END PROCESS;

END Comportamental;
```

Figura 5.12: Descrição comportamental obtida pelo programa PIPE2VHDL4M para o detector de três zeros consecutivos sem sobreposição.

Na descrição comportamental apresentada na figura 5.12, são criados dois tipos de processos, um para controlar o próximo estado e saída, obedecendo ao estado atual e entrada da máquina, este processo tem como lista de sensibilidade as variáveis Atual e x0 e outro para controlar a mudança dos estados da máquina, assim o próximo estado passa a ser o atual, este processo é sensível à transição de subida.

A descrição gerada pelo programa PIPE2VHDL4M pode ser compilada e simulada nos ambientes da Altera.

5.2 Programas Desenvolvidos para a Metodologia 5M

Nesta subseção apresentam-se os programas denominados PIPE2TAB5M e PIPE2VHDL5M, implementados para analisar a descrição PNML do modelo da RdP Lugar/Transição, esquematizado no ambiente PIPE, utilizando-se da metodologia de modelagem 5M. Estes programas geram a tabela de transição de estados e a descrição comportamental na linguagem VHDL para as FSM do tipo Mealy e/ou Moore especificadas em RdP.

5.2.1 Programa PIPE2TAB5M

O programa PIPE2TAB5M gera a tabela de transição de estados, através da análise do código PNML da RdP Lugar/Transição, que modela a FSM do tipo Mealy ou Moore, por intermédio da metodologia 5M.

Quando o programa é executado e, seguidamente os argumentos solicitados são informados, tais como endereço do arquivo de entrada e de saída, o programa abre o arquivo de entrada para analisar a descrição PNML e cria o arquivo de saída para salvar a tabela

gerada. Caso ocorra algum erro durante o processo de abertura, criação de arquivos ou na análise dos parâmetros do arquivo de entrada, o programa é encerrado sem realizar a geração da tabela e fornece a mensagem do referido erro para o usuário.

Se não houver nenhum erro nos parâmetros o arquivo de entrada é analisado. Os lugares descritos no código PNML dentro das etiquetas <*place id* = "rótulo do lugar"> e </*place*>, são os primeiros componentes analisados com o propósito de identificar se as FSM modeladas são do tipo Mealy ou Moore. Se a identificação do "valor" dos seus nomes, encontrados dentro das etiquetas <*value*> e </*value*>, apresentarem os caracteres "/S", a máquina modelada será de Moore, pois as saídas estão sendo representadas nos lugares (estados da FSM), caso contrário será de Mealy. A descrição PNML será analisada conforme o tipo de máquina modelada pela RdP Lugar/Transição. Observa-se na figura 5.13 que a máquina descrita é do tipo Mealy, pois nenhum dos lugares apresentados nessa descrição PNML possuem os caracteres "/S" no "valor" dos lugares descritos. A descrição PNML mostrada na figura 5.13 refere-se à RdP apresentada na figura 4.12, que modela o detector da seqüência de três zeros consecutivos sem sobreposição, por intermédio da metodologia 5M.

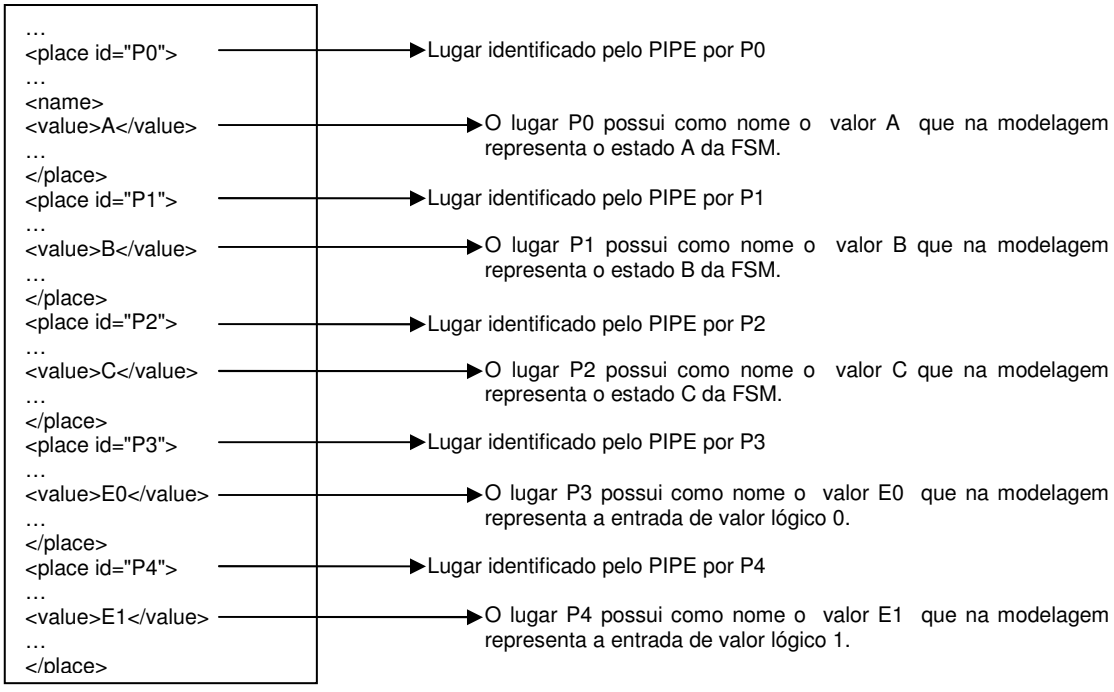


Figura 5.13: Trecho do código PNML onde estão descritos os lugares da RdP modelada pelo ambiente PIPE.

Nos modelos das máquinas de Moore é possível computar as entradas, saídas e estados existentes através da descrição dos lugares, no código PNML. Enquanto que no modelo de Mealy as saídas estão representadas nas transições, localizadas no código dentro das etiquetas <*transition id*=”rótulo da transição”> e <*/transition*>, desta forma para este caso é necessário analisar o “valor” das transições, encontrado dentro das etiquetas <*value*> *e* <*/value*>, para identificar as saídas. Os estados e entradas nesse último modelo são representados nos lugares da RdP. Na figura 5.14 é ilustrado o trecho do código PNML que descreve as transições da RdP apresentada na figura 4.12. Nota-se que no “valor” de cada transição também é representado a saída da máquina, por exemplo, as transições T0, T1, T2, T4 e T5 representam a saída de valor lógico 0 e T3 a saída de valor lógico 1.

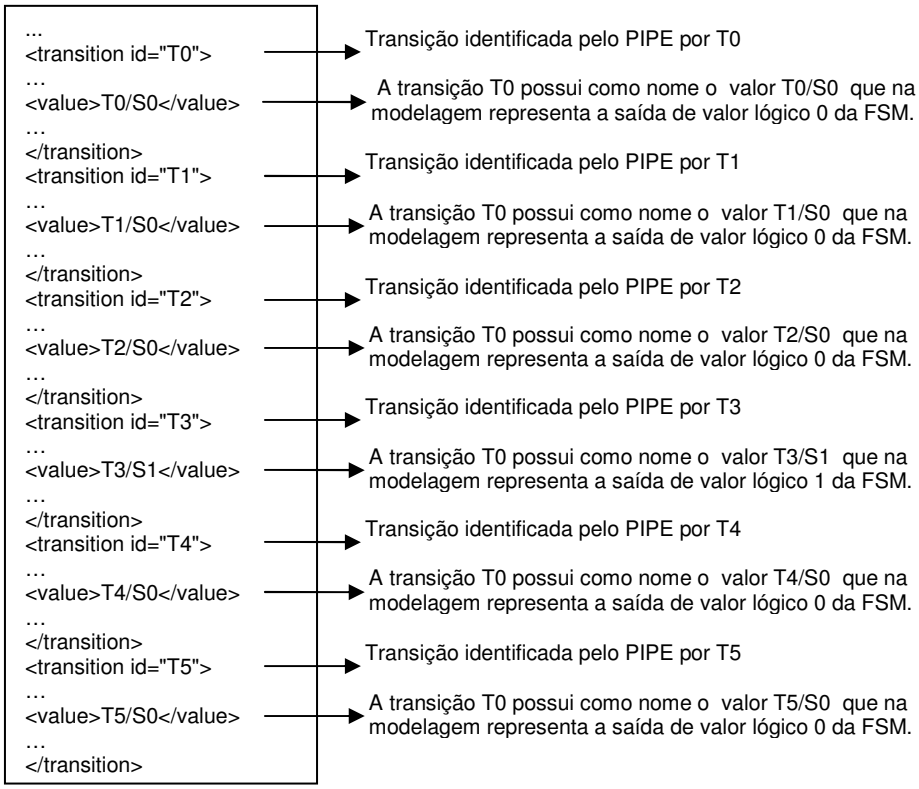


Figura 5.14: Trecho do código PNML onde estão descritas as transições da RdP modelada pelo ambiente PIPE.

O número mínimo de *flip-flops* é calculado de acordo com a quantidade de estados existentes no modelo da RdP. Após, calculado o número de *flip-flops* necessários, é solicitado ao usuário que indique o tipo de todos os *flip-flops* que se deseja utilizar.

Os arcos descritos no código PNML dentro das etiquetas <*arc..*> e <*/arc*>*,* são utilizados para obter a tabela de transição de estados, pois através da descrição dos arcos é possível detectar o estado atual, o próximo estado e a entrada da FSM. O estado atual e a entrada são localizados nos arcos que representam a ligação dos lugares às transições, identificados no código PNML pelo trecho *id*="rótulo do lugar" *to* "rótulo da transição", sendo que o próximo estado é localizado nos arcos que realizam a ligação das transições aos lugares, identificados no trecho de código PNML *id*="rótulo da transição" *to* "rótulo do lugar". As saídas no modelo Mealy são identificadas nas transições que conectam os lugares que representam o estado atual e a entrada ao lugar que modela o próximo estado. Já no modelo de Moore as saídas são modeladas no mesmo lugar que representa o estado atual. Analisando, identificando e processando esses componentes, a tabela de transição da FSM é gerada. Observa-se na figura 5.15 que os lugares de entrada da transição T0 são P1 e P4, que na modelagem representam os lugares B (estado B da FSM) e E1 (entrada de valor lógico 1). Sendo que o lugar de saída dessa mesma transição é P0, que correspondem na modelagem o lugar A (estado A da FSM). Assim para este caso temos que o estado atual, próximo estado, entrada e saída da FSM modelada pela RdP da figura 4.12 são B, A, 1 e 0. Neste caso a saída recebe o valor lógico 0, pois na descrição das transições, apresentada na figura 5.14, tem-se que o valor apresentado na transição T0 é T0/S0, assim S0 representa a saída 0.

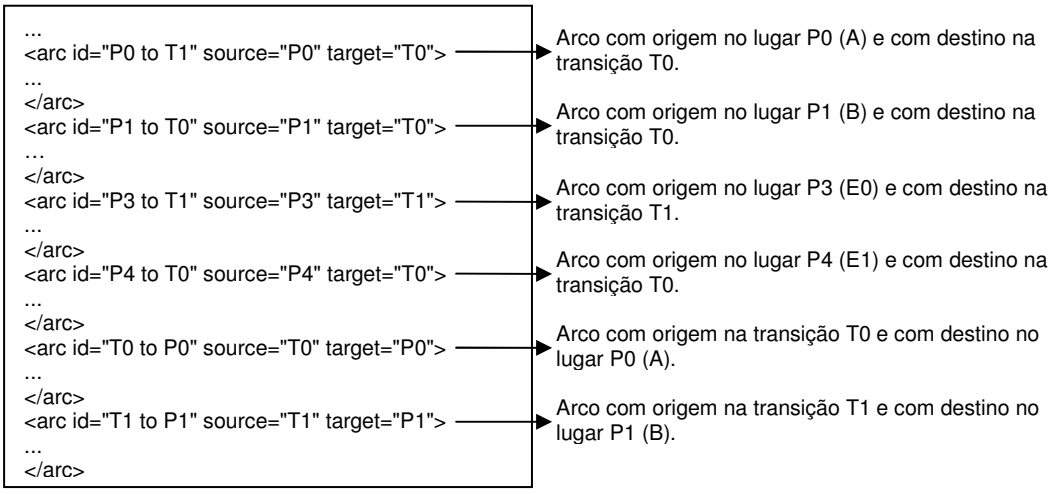


Figura 5.15: Trecho do código PNML onde estão descritas os arcos da RdP modelada pelo ambiente PIPE.

Para executar o programa PIPE2TAB5M são necessárias as seguintes etapas:

a. Abrir o *prompt* de comando (DOS);

b. Sair do comando raiz, entrar no diretório e pastas em que o programa PIPE2TAB5M se encontra, ou seja, indicar o caminho, por exemplo, D:\Material Alexandre- Norian\Meus Programas-Python;

c. Digitar o caminho para executar o Python e nome do programa que será executado, c:\python25\python.exe PIPE2TAB5M.p**y;**

d. Após teclar o *Enter* o programa será executado e os seguintes dados serão requisitados ao usuário;

- Estrutura de diretório (Diretório:\Pasta\Nome do arquivo.extensão) do arquivo de entrada, por exemplo, D:\MaterialAlexandre-Norian\Saida-TCPIPE\Descrição RdP-2\Mealy.txt;

- Estrutura de diretório do arquivo de saída para armazenar os resultados gerados pelo programa, por exemplo, D:\MaterialAlexandre-Norian\Saida-PIPE2TAB5M\Mealy.txt;

- Tipo de cada um dos *flip-flops* que se deseja utilizar. Os *flip-flops* que podem ser utilizados são do tipo D e JK, este último tipo é representado apenas pelo caractere J.

A figura 5.16 mostra as etapas relacionadas acima, que devem ser executadas pelo usuário no *prompt* do DOS para executar o programa PIPE2TAB5M.

```
C:\Documents and Settings\Administrador>cd\

C:\>d:

D:\>cd MaterialAlexandre-Norian\Meus Programas-Python

D:\MaterialAlexandre-Norian\Meus Programas-Python>c:\python25\python.exe PIPE2TA
B5M.py
Digite o endereÞo do arquivo de entrada
--por ex. 'Diret¾rio:\Pasta\Nome do arquivo.extenÞòo'
D:\MaterialAlexandre-Norian\Saida-TCPIPE\Descricao RdP-2\Mealy.txt
Digite o endereÞo do arquivo de saýda
--por ex. 'Diret¾rio:\Pasta\Nome do arquivo. extenÞòo'
D:\MaterialAlexandre-Norian\Saida-PIPE2TAB5M\Mealy.txt
Digite o tipo do flip flop
J
Digite o tipo do flip flop
D
```

Figura 5.16: Ambiente do prompt DOS onde o programa PIPE2TAB5M foi executado.

O diagrama de blocos do programa PIPE2TAB5M é demonstrado na figura 5.17. Observam-se no diagrama que os retângulos de cantos arredondados representam apenas o início e o fim da execução do programa, os retângulos com o canto esquerdo cortado (cartão) indica os dados solicitados pelo programa ao usuário e a leitura dos arquivos de entrada. Os arquivos de entrada e saída são indicados pelo componente utilizado em fluxogramas que representa documentos. Os cálculos, a verificação dos tipos de máquinas modeladas em RdP e a construção do arquivo de saída são representados no diagrama por retângulos normais. O programa PIPE2TAB5M gera um arquivo de saída, contendo a descrição da tabela de transição de estados, por intermédio da descrição PNML da RdP modelada no ambiente PIPE.

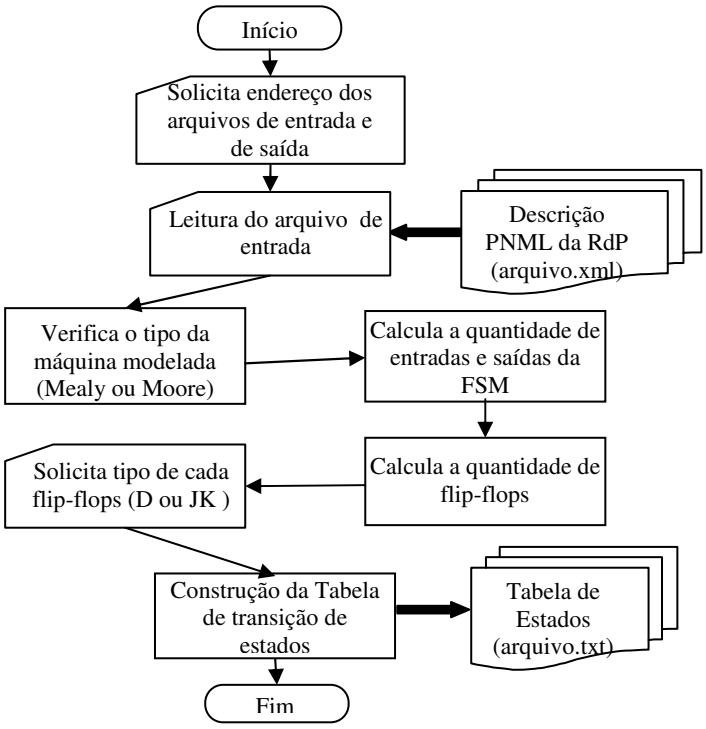


Figura 5.17: Diagrama de Blocos do programa PIPE2TAB5M.

O arquivo gerado pelo programa PIPE2TAB5M para a descrição PNML da RdP Lugar/Transição que modela o detector da seqüência de três zeros consecutivos sem sobreposição, é exibido na figura 5.18. Observa-se na figura 5.18 que a descrição gerada pelo programa PIPE2TAB5M é semelhante à gerada pelo PIPE2TAB4M, apresentada na figura 5.3, pois a tabela representada em ambos os arquivos tratam a mesma FSM, o que as

diferencia é a metodologia de modelagem aplicada para modelá-la em RdP Lugar/Transição. Assim o arquivo da figura 5.18 difere da figura 5.5 apenas pelo tipo dos *flip-flops* fornecidos pelo usuário. Na figura 5.18 os tipos JK e D foram escolhidos pelo usuário para o 1^o e 2^o *flip-flop*, enquanto que na figura 5.5 foram escolhidos para o 1^o e 2^o *flip-flop* os tipos D e JK.

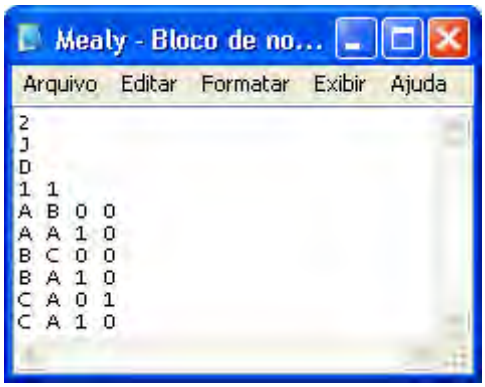


Figura 5.18: Arquivo gerado pelo programa PIPE2TAB5M para o detector de três zeros consecutivos sem sobreposição.

O arquivo gerado pelo PIPE2TAB5M também permite que os programas AGPS, TABELA e TAB2VHDL, sejam utilizados. Entretanto, o TAB2VHDL só deve ser utilizado quando as máquinas modeladas forem do tipo Mealy, visto que a descrição RTL obtida por este é restrita apenas a essas máquinas.

5.2.2 Programa PIPE2VHDL5M

O programa PIPE2VHDL5M é responsável por gerar a descrição comportamental na linguagem VHDL da FSM modelada em RdP Lugar/Transição, por intermédio do ambiente PIPE e da metodologia de modelagem 5M. A descrição comportamental é obtida através da análise do código PNML da RdP modelada.

Ao executar o programa e, seguidamente, informar os argumentos solicitados, tais como endereço do arquivo de entrada e de saída, o programa abre o arquivo de entrada para analisar a descrição PNML e cria o arquivo de saída para salvar a tabela gerada. Caso ocorra

algum erro durante o processo de abertura, criação de arquivos ou na análise dos parâmetros do arquivo de entrada, o programa é encerrado sem realizar a geração do código VHDL comportamental e fornece a mensagem do referido erro para o usuário.

Caso não exista nenhum erro nos parâmetros o arquivo de entrada é analisado. Os lugares descritos no código PNML nas etiquetas <*place id* = "rotulo do lugar"> e <*/place*>, são os primeiros componentes analisados com o propósito de identificar se as FSM modeladas são do tipo Mealy ou Moore. Se a identificação do "valor" dos seus nomes, encontrados dentro das etiquetas <*value*> e <*/value*>, apresentarem os caracteres "/S" a máquina modelada será de Moore, caso contrário será de Mealy. A descrição PNML será analisada conforme o tipo de máquina modelada em RdP Lugar/Transição.

Nos modelos das máquinas de Moore é possível computar as entradas, saídas e estados existentes através da descrição dos lugares no código. Enquanto que no modelo de Mealy as saídas estão representadas nas transições, localizadas no código PNML dentro das etiquetas <*transition id*="rótulo da transição"> e <*/transition*>, desta forma para este caso é necessário analisar o "valor" das transições, encontrado dentro das etiquetas <*value*> e <*/value*>, para identificar as saídas. Os estados e entradas nesse último modelo são representados nos lugares da RdP.

As entradas e saídas identificadas no modelo são convertidas para valores binários e armazenadas no arquivo de saída na entidade, representada dentro do trecho de código *ENTITY* da descrição VHDL, como portas do tipo *bit*.

Após, definidas as entradas e saídas, é solicitado ao usuário que indique o nome da arquitetura que se deseja utilizar no código VHDL a ser gerado.

Os estados identificados na análise dos lugares são declarados dentro da arquitetura, representada pela etiqueta *ARCHITECTURE,* com o tipo "tipo-estado".

Os arcos descritos no código PNML dentro da etiqueta <*arc...*> e </*arc*>, são utilizados para descrever o processo do código VHDL, descrito dentro do trecho de código *PROCESS (Atual, x_i)* e *END PROCESS*, onde a transição dos estados são especificadas. Através da descrição dos arcos é possível detectar o estado atual, próximo estado e entrada da FSM. As saídas nas máquinas de Mealy são identificadas nas transições, enquanto que nas de Moore são modeladas no mesmo lugar que representa o estado atual.

As etapas utilizadas para executar o programa PIPE2VHDL5M são semelhantes às utilizadas pelo programa PIPE2TAB5M, o que modifica é o nome do programa que está sendo executado, neste caso o nome do programa a ser fornecido é PIPE2VHDL5M.

Os dados requeridos pelo programa aos usuários são:

a. Endereço do arquivo de entrada para captura dos dados, por exemplo, D:\MaterialAlexandre-Norian\Saida-TCPIPE\Descrição RdP-2\Mealy.txt;

b. Endereço do arquivo de saída para armazenar a descrição comportamental na linguagem VHDL gerada pelo programa;

c. Nome da arquitetura para a descrição VHDL.

A figura 5.19 ilustra as etapas mencionadas, que devem ser executadas no *prompt* do DOS para rodar o programa PIPE2VHDL5M e fornecer os dados de entrada requisitados para o usuário.

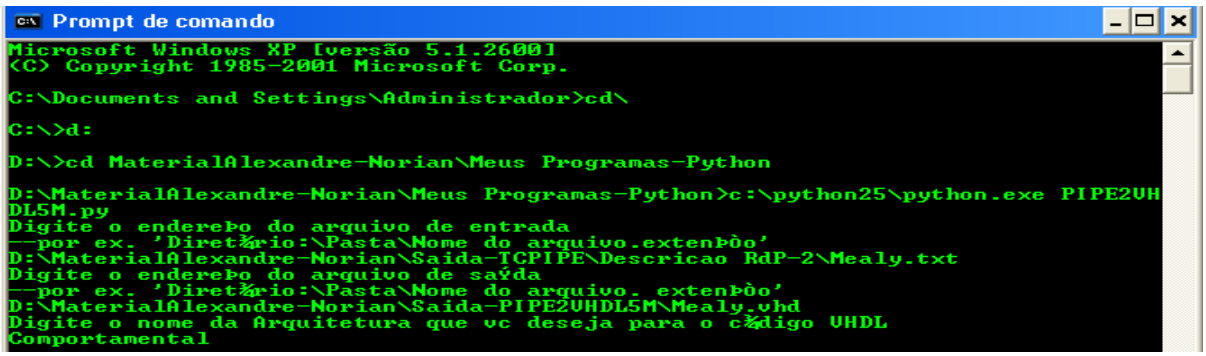


Figura 5.19: Ambiente do *prompt* DOS onde o programa PIPE2VHDL5M foi executado.

O diagrama de blocos do programa PIPE2VHDL5M é demonstrado na figura 5.20. Neste diagrama os retângulos com o canto esquerdo cortado (cartão) indicam os dados

solicitados pelo programa ao usuário e a leitura dos arquivos de entrada. Os arquivos de entrada e saída são representados no diagrama pelo componente do fluxograma que indica documentos. O processo para a verificação do tipo de máquina modelada, a conversão das entradas e saídas para valores binários e a construção da descrição comportamental VHDL do sistema modelado, são representados por retângulos normais. Os retângulos de cantos arredondados do diagrama representam apenas o início e o fim da execução do programa PIPE2VHDL5M.

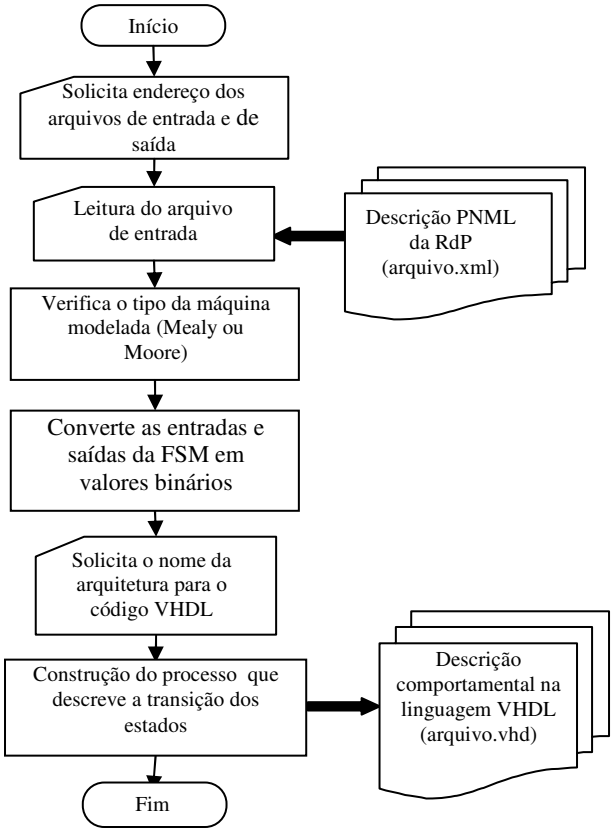


Figura 5.20: Diagrama de Blocos do programa PIPE2VHDL5M.

O arquivo gerado pelo programa PIPE2VHDL5M, contendo descrição comportamental em linguagem VHDL para o detector da seqüência de três zeros consecutivos sem sobreposição é semelhante à descrição obtida pelo PIPE2VHDL4M, pois especificam a mesma FSM modelada em metodologias diferentes.

Na figura 5.21 é ilustrado um trecho da descrição comportamental obtida pelo PIPE2VHDL5M. Nota-se na descrição que os portos de entrada e saída e o sinal de

sincronismo (*clock*) são definidos dentro da entidade. Os portos denominados x0 e z0, definidos como *bit,* representam as entradas e saídas da máquina modelada em RdP. Os estados da máquina são representados dentro da arquitetura como um tipo de variável "tipo_estado". Dois tipos de processos são criados na descrição comportamental, um para controlar o próximo estado e saída, obedecendo ao estado atual e entrada da máquina, este processo tem como lista de sensibilidade as variáveis Atual e x0 e outro para controlar a mudança do próximo estado da máquina para o atual, este processo é sensível à transição (borda) de subida.

Arquivo Editar Formatar Exibir Ajuda

```
ENTITY Mealy IS
        PORT
        (
                --xn representão os sinais de entrada
                --zn representão os sinais de saída
                clock:IN BIT;
                x0:IN BIT;
                z0:OUT BIT
        );
END Mealy;

ARCHITECTURE Comportamental OF Mealy IS

        TYPE tipo_estado IS (A, B, C);
        SIGNAL Atual, Proximo: tipo_estado;
BEGIN

        PROCESS (Atual, x0)

        BEGIN
                CASE Atual IS
                        WHEN A =>
                                IF x0= '0' THEN
                                        z0<= '0';
                                        Proximo <=B;
                                ELSE
                                        z0<= '0';
                                        Proximo <=A;
                                END IF;
                        WHEN B =>
                                IF x0= '0' THEN
                                        z0<= '0';
                                        Proximo <=C;
                                ELSE
                                        z0<= '0';
                                        Proximo <=A;
                                END IF;
                        WHEN C =>
                                IF x0= '0' THEN
                                        z0<= '1';
                                        Proximo <=A;
                                ELSE
                                        z0<= '0';
                                        Proximo <=A;
                                END IF;
                END CASE;
        END PROCESS;

        PROCESS
        BEGIN
                WAIT UNTIL clock'Event AND clock = '1';
                        Atual <= Proximo;
        END PROCESS;

END Comportamental;
```

Figura 5.21: Descrição comportamental obtida pelo programa PIPE2VHDL5M para o detector de três zeros consecutivos sem sobreposição, modelado pela metodologia 5M.

6 ANÁLISE DOS TESTES

Neste capítulo são discutidos os resultados dos testes, obtidos utilizando-se as metodologias de modelagem e os programas de síntese apresentados nesta dissertação.

Os resultados descritos na seção 6.1, para ambas as metodologias de modelagem, 4M e 5M, são gerados pelos testes do detector da seqüência de *bits* 10010 com sobreposição, do código de linha AMI, da FSM com duas entradas e uma saída e do comparador de série. O teste do código HDB3 foi realizado apenas com a metodologia 5M devido à sua capacidade de modelagem.

Através da modelam das máquinas de Moore em RdP, utilizando-se da metodologia 5M, discute-se na seção 6.2 os resultados obtidos pelos testes. As máquinas apresentadas nesta seção projetam o detector da seqüência de bits 11, o código de linha AMI e a FSM com duas entradas e uma saída, estas máquinas compõem o espaço amostral dos testes dessa subseção.

A fim de validar as metodologias e os programas desenvolvidos procurou-se escolher casos diferentes de testes, assim utilizou-se como espaço amostral dos testes FSM com apenas uma entrada e saída, com duas entradas e uma saída, com uma entrada e duas saídas e com várias entradas e saídas. Desta forma com o processamento correto de todos os testes do espaço amostral, fornecendo as saídas desejadas, tem-se que as metodologias e ferramentas desenvolvidas podem ser utilizadas para modelar qualquer FSM nos modelos de Mealy ou Moore.

6.1 Testes Aplicados a Metodologia 4M e 5M

Nesta seção apresentar-se-ão os resultados obtidos pelos programas PIPE2TAB4M, PIPE2VHDL4M, PIPE2TAB5M, PIPE2VHDL5M, AGPS, TABELA e

TAB2VHDL, para os testes do detector da seqüência de bits 10010 com sobreposição, do código de linha AMI, da FSM com duas entradas e uma saída, do comparador de série e do código de linha HDB3.

6.1.1 Detector para a seqüência 10010 com sobreposição

O detector apresentado nesta subseção gera a sua saída como nível lógico 1 toda vez que a seqüência 10010 for encontrada na seqüência de pulsos (1 e 0) fornecida como entrada, mesmo que essa seqüência seja com sobreposição.

Com o propósito de esclarecer o funcionamento do detector, é ilustrada na tabela 6.1 uma seqüência de entradas que exibe a atuação do detector da seqüência 10010 com sobreposição de acordo com as entradas fornecidas. Na linha E são apresentadas as entradas e na linha S as saídas fornecidas pela máquina obedecendo a seqüência de entradas indicada na linha E.

Tabela 6.1: Demonstra o funcionamento do detector para a seqüência 10010 com sobreposição.

E	1	0	0	1	0	0	1	0	0	1	0	0	0	1	0	1	0	1	0	1	0	0	0	1	1	0	0	1	0
S	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

A FSM do tipo Mealy para o detector da seqüência de *bits* 10010 com sobreposição é apresentada na figura 6.1. O estado A é o estado inicial da máquina. O deslocamento para os estados A, B, C, D, E, F ou G, ocorre de acordo com o valor lógico da entrada, ou seja, se a máquina estiver no estado A e a entrada for 0 a máquina deslocará para o B e produzirá uma saída de valor lógico 0, caso a entrada seja 1 a máquina permanecerá em A e determinará saída 0. Caso a máquina estiver no estado B e a entrada for 0 a máquina deslocará para o C e produzirá a saída 0, caso a entrada seja 1 a máquina deslocará para o estado F e determinará saída 0. A mesma lógica é usada para todos os estados da máquina, assim, cada estado analisa obrigatoriamente todas as entradas e fornece, para cada uma delas, uma saída.

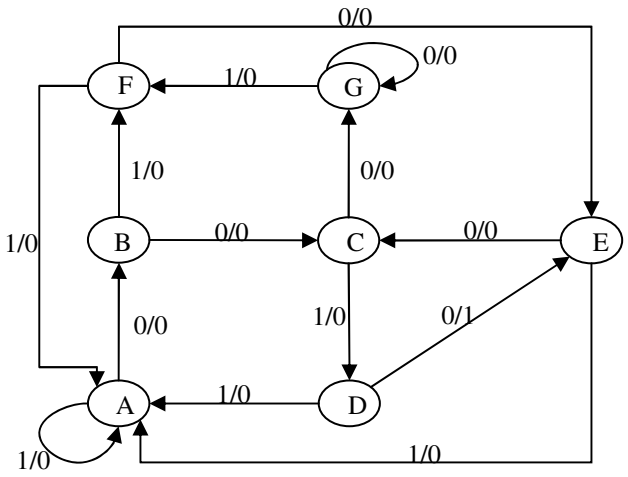


Figura 6.1: Máquina de Mealy para o detector da seqüência 10010 com sobreposição [25].

Utilizando-se o ambiente PIPE e a metodologia 4M para modelar o detector da seqüência 10010 com sobreposição, obtém-se a RdP Lugar/Transição apresentada na figura 6.2. Observa-se que cada um dos estados da máquina ilustrada na figura 6.1 são representados por lugares distintos na RdP que recebem a mesma denominação dos estados (A, B, C, D, E, F, G). Tem-se que as entradas e saídas da máquina correspondem respectivamente aos lugares E0 (entrada de valor lógico 0), E1 (entrada de valor lógico 1), S0 (saída de valor lógico 0) e S1(saída de valor lógico 1).

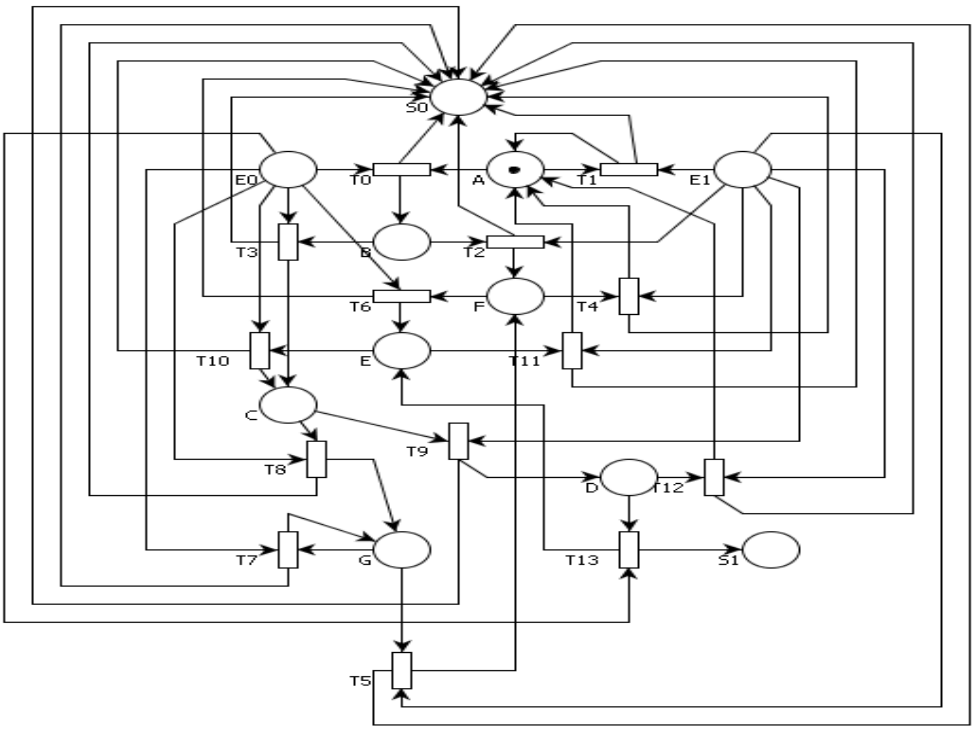


Figura 6.2: Modelo da RdP obtido pela metodologia 4M para o detector da seqüência 10010 com sobreposição.

Aplicando a metodologia 5M para modelar o detector ilustrado na figura 6.1, obtém-se a RdP Lugar/Transição mostrada na figura 6.3. Nota-se que os estados do detector são representados por lugares distintos na rede, e recebem a mesma nomeação (A, B, C, D, E, F, G). As entradas são representadas pelos lugares E0 e E1 que correspondem aos valores lógicos 0 e 1. As saídas são representadas nas transições de acordo com a entrada identificada em cada estado, por exemplo, a transição T0/S0 representa a saída de valor lógico 0, esta é habilitada quando o estado atual da máquina for A e a entrada for 0, o próximo estado atingido com o disparo dessa transição é B. Enquanto a transição T1/S0 "gera" a saída 0 quando o estado atual for A e a entrada 1, o próximo estado atingido com o disparo dessa transição é o próprio estado A. Observa-se que houve uma redução de cerca de 14% dos elementos gráficos na modelagem da RdP da figura 6.3 em relação a rede da figura 6.2.

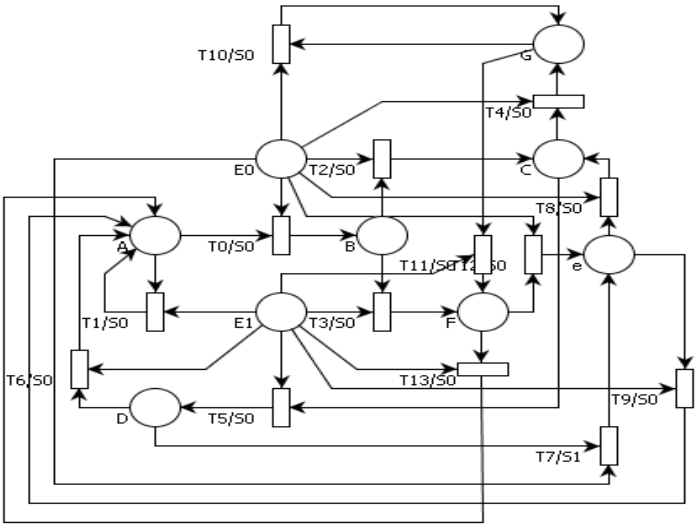
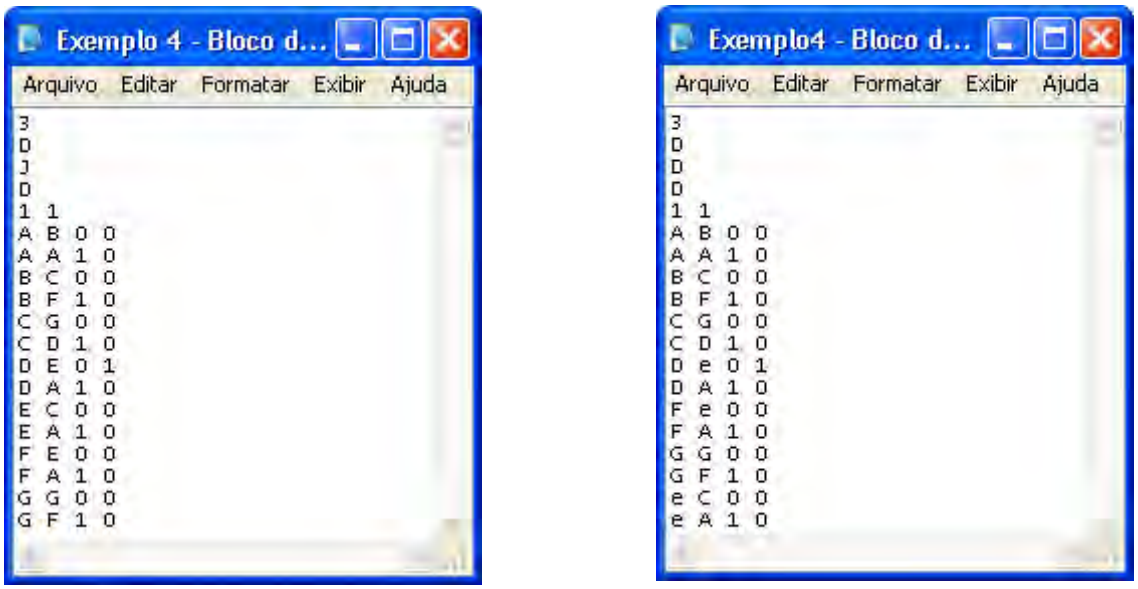


Figura 6.3: Modelo da RdP para o detector da seqüência 10010 com sobreposição, obtido através da metodologia 5M.

A modelagem ilustrada na figura 6.2 e figura 6.3 geram descrições PNML que podem ser utilizadas pelos programas PIPE2TAB4M e PIPE2TAB5M, os quais, por sua vez, transformam as informações da FSM especificadas pelos modelos da RdP em forma de tabela. A configuração dos arquivos criados pelos PIPE2TAB4M e PIPE2TAB5M, está ilustrada na figura 6.4. Nota-se que ambos os arquivos possuem configurações e descrições das transições dos estados idênticas, pois as duas metodologias modelam a mesma FSM em RdP distintas,

assim através da metodologia 4M e do programa PIPE2TAB4M é possível obter o arquivo da figura 6.4(a) e por intermédio da metodologia de modelagem 5M e do programa PIPE2TAB5M é possível obter o arquivo da figura 6.4(b). O estado representado pelo caractere "E" na figura 6.4 (a), está apresentado como "e" na figura 6.4 (b) devido a nomeação dada ao lugar que o representada durante a modelagem da FSM em RdP, lembrando-se que uma das restrições da metodologia 5M é que o nome dado aos lugares que representam os estados da máquina não deve possuir o caractere "E".



(a) (b)

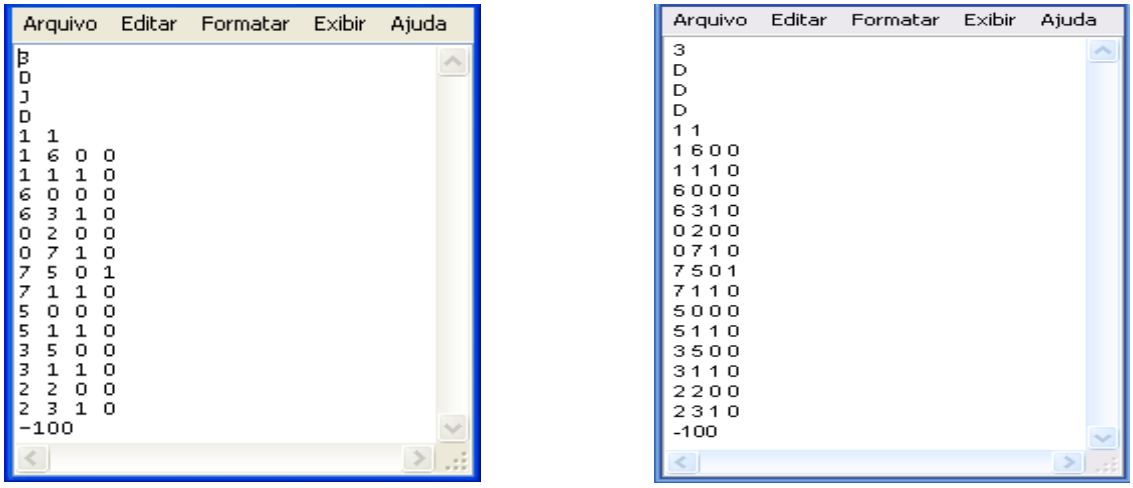
Figura 6.4: Arquivos criados pelos programas PIPE2TAB4M (a) e PIPE2TAB5M(b).

Na primeira linha dos arquivos apresentados na Figura 6.4 especifica-se a quantidade mínima de elementos de memórias. Neste caso necessita-se de no mínimo três elementos de memórias, visto que o sistema possui sete estados. Nas três linhas imediatamente abaixo se especificam o tipo de cada elemento que será utilizado, ou seja, o tipo dos *flip-flops*. Os escolhidos pelo usuário no programa PIPE2TAB4M, figura 6.4(a), foram seqüencialmente do tipo D, JK e D para o 1º, 2º e 3º *flip-flops*. Enquanto que na tabela gerada pelo programa PIPE2TAB5M, figura 6.4(b), escolheu-se o *flip-flops* tipo D, para os três elementos de memórias. Não há nenhum critério para a escolha dos flip-flops, estes são informados pelo usuário de acordo com sua vontade.

Especifica-se, na quinta linha, a quantidade de entradas e de saídas. O sistema projetado tem um porto de entrada e um porto de saída.

Nas linhas subseqüentes especificam-se as transições dos estados e a saída gerada em cada transição, de acordo com a entrada fornecida para o sistema. Portanto, a partir da sexta linha, tem-se que a primeira coluna representa o estado atual, a segunda o próximo estado, na terceira e na quarta colunas as respectivas entradas e saídas.

Os arquivos gerados pelos programas PIPE2TAB4M e PIPE2TAB5M são utilizados pelo programa AGPS, como arquivo de entrada, este programa por sua vez realiza a alocação de todos os estados da máquina. O arquivo gerado pelo AGPS possui a mesma formatação da fornecida pelos PIPE2TAB4M e PIPE2TAB5M, a única alteração ocorre na alocação dos estados que são representados por caracteres decimais e não mais por alfabéticos. Tem-se que o AGPS realizou, respectivamente, as alocações 1,6, 0, 7, 5, 3, 2 para os estados A, B, C, D, E, F e G, apresentados no arquivo da figura 6.4(a). O arquivo de saída criado pelo AGPS está ilustrado na figura 6.5 (a). Observa-se que o AGPS conserva a configuração dos arquivos gerados pelos programas PIPE2TAB4M e PIPE2TAB5M. A alocação dos estados é a mesma para os dois arquivos apresentados na Figura 6.4 pois descrevem a mesma FSM modelada em metodologias diferentes.



(a) (b)

Figura 6.5: Arquivo gerado pelo programa AGPS para os arquivos da Figura 6.4.

A utilização do AGPS auxilia o ambiente de síntese, pois o arquivo gerado por ele é utilizado como entrada pelo programa TABELA, por intermédio das informações do arquivo gerado pelo AGPS o TABELA realiza a minimização das funções booleanas do sistema.

Na figura 6.6 está descrito um trecho do código de saída do programa TABELA. Observa-se na figura 6.6 que todos os mintermos e *don't care states* das funções booleanas *e* os seus respectivos implicantes primos são apresentados em cada função combinacional. Por exemplo, a função J1 é composta pelos mintermos 1, 0 e 8 e pelos *don't care states* 6, 14, 7, 15, 3, 11, 2, 10, 12 e 4. Os implicantes primos geram as funções mínimas. O custo total para a implementação das quatro funções combinacionais (D2, J1, K1, D0 e Z0) é igual a 31.

```
FUNCAO D2
═════════

MINTERMOS :  1;  8;  7;  3;

DON'T CARE STATES : 12;  4;

IMPLICANTES PRIMOS ESSENCIAIS :

ESSENCIAL:  3     REDUNDANCIA:        4 ->  0X11
ESSENCIAL:  8     REDUNDANCIA:        4 ->  1X00
ESSENCIAL:  1     REDUNDANCIA:        2 ->  00X1

CUSTO FINAL DE D2 = 12

FUNCAO J1
═════════

MINTERMOS :  1;  0;  8;

DON'T CARE STATES :  6; 14;  7; 15;  3; 11;  2; 10; 12;  4;

IMPLICANTES PRIMOS ESSENCIAIS :

ESSENCIAL:  0     REDUNDANCIA:       14 ->  XXX0
ESSENCIAL:  0     REDUNDANCIA:        3 ->  00XX

CUSTO FINAL DE J1 = 5

FUNCAO K1
═════════

MINTERMOS :  6;  7; 15;  3; 11;

DON'T CARE STATES :  1;  9;  0;  8;  5; 13; 12;  4;

IMPLICANTES PRIMOS ESSENCIAIS :

ESSENCIAL:  1     REDUNDANCIA:       14 ->  XXX1
ESSENCIAL:  4     REDUNDANCIA:        3 ->  01XX

CUSTO FINAL DE K1 = 5

FUNCAO D0
═════════

MINTERMOS :  9; 14;  8;  7; 15; 13;  3; 11; 10;

DON'T CARE STATES : 12;  4;

IMPLICANTES PRIMOS ESSENCIAIS :

ESSENCIAL:  3     REDUNDANCIA:       12 ->  XX11
ESSENCIAL:  8     REDUNDANCIA:        7 ->  1XXX

CUSTO FINAL DE D0 = 5

FUNCAO Z0
═════════

MINTERMOS :  7;

DON'T CARE STATES : 12;  4;

IMPLICANTES PRIMOS ESSENCIAIS :

ESSENCIAL:  7     REDUNDANCIA:        0 ->  0111

CUSTO FINAL DE Z0 = 4

CUSTO TOTAL DAS 5 FUNCOES = 31
```

Figura 6.6: Descrição gerada pelo programa Tabela para o detector da seqüência 10010.

O arquivo gerado pelo programa TABELA é utilizado como entrada do programa TAB2VHDL, este por sua vez gera a descrição RTL na linguagem VHDL para o detector da seqüência 10010 com sobreposição.

O código descrito pela ferramenta TAB2VHDL está apresentado na figura 6.7, com a finalidade de mostrar a descrição VHDL gerada pela ferramenta para o detector da seqüência 10010 com sobreposição. Nota-se que na entidade da descrição apresentada no código RTL da figura 6.7, foram definidos os portos de entrada (X0) e saída (Z0) e os sinais de sincronismo (CLK e CLR). Os sinais auxiliares (VE0, VE1 e VE2) são definidos com o objetivo de evitar que se gerem vários *drivers* para um mesmo sinal.

Para cada *flip-flop* utilizado é criado um processo na descrição, assim três tipos de processos foram criados na descrição da figura 6.7, dois para modelar o *flip-flop* D e outro para modelar o *flip-flop* JK, ambos sensíveis a transição de subida. Entretanto como apenas parte do código RTL foi apresentada é possível observar apenas dois processos.

```
ENTITY Ex4TAB IS
    PORT(
    CLK, CLR : IN BIT;
    X0 : IN BIT;
    Q0, Q1, Q2 : OUT BIT;
    Z0 : OUT BIT );
END Ex4TAB;

ARCHITECTURE RTL OF Ex4TAB IS

SIGNAL VE0, VE1, VE2: BIT;

SIGNAL D2, J1, K1, D0 : BIT;


 BEGIN

-- Inferindo flip-flop tipo D
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;

-- Inferindo flip-flop tipo JK
PROCESS(CLK, CLR)
BEGIN
   IF CLR = '0' THEN
     VE1 <= '0';
   ELSIF CLK'EVENT and CLK = '1' THEN
     IF J1 = '0' and K1 = '1' THEN
       VE1 <= '0';
     ELSIF J1 = '1' and K1 = '0' THEN
       VE1 <= '1';
     ELSIF J1 = '1' and K1 = '1' THEN
       VE1 <= not(VE1);
     END IF;
   END IF;
   Q1 <= VE1;
END PROCESS;

-- Inferindo flip-flop tipo D
PROCESS(CLK, CLR)
BEGIN
   IF CLR = '0' THEN
     VE2 <= '0';
   ELSIF CLK'EVENT and CLK = '1' THEN
     VE2 <= D2;
   END IF;
   Q2 <= VE2;
END PROCESS;
```

Figura 6.7: Descrição RTL na linguagem VHDL gerada pelo programa TAB2VHDL.

Todo o processo de síntese digital é realizado para as metodologias de modelagem 4M e 5M, utilizando-se dos programas apresentados nesta subseção. Pode-se observar que cinco programas são utilizados para realizar a síntese do sistema modelado em RdP. Os programas PIPE2TAB4M, AGPS, TABELA e TAB2VHDL são usados para metodologia de modelagem 4M. Enquanto, os programas PIPE2TAB5M, AGPS, TABELA e TAB2VHDL são utilizados para a 5M.

O programa PIPE2VHDL4M foi desenvolvido com o propósito de gerar automaticamente a descrição comportamental na linguagem VHDL para os sistemas modelados em RdP Lugar/Transição, este ambiente de síntese foi desenvolvido para ser aplicado à metodologia de modelagem 4M. Enquanto, o programa PIPE2VHDL5M gera a descrição comportamental na linguagem VHDL para a RdP modelada através da metodologia 5M.

Os programas PIPE2VHDL4M e PIPE2VHDL5M utilizam como arquivo de entrada a descrição PNML gerada pelo ambiente PIPE, onde o sistema foi modelado em RdP Lugar/Transição, utilizando-se umas das metodologias desenvolvidas.

Parte da descrição gerada pelo PIPE2VHDL4M é apresentada na figura 6.8. Observa-se que os portos de entrada e saída denominados x0 e z0 respectivamente, e o sinal de sincronismo (*clock*) são definidos dentro da entidade (*ENTITY*). Os portos representam as entradas e saídas do detector da seqüência 10010 com sobreposição modelado em RdP. Os estados da máquina (A, B, C, D, E, F E G) são representados dentro da arquitetura como um tipo de variável “tipo_estado”. Dentro da estrutura *Case* é defina a mudança de estados e saídas da máquina modelada em RdP que representa o funcionamento do detector da seqüência 10010 com sobreposição.

Os códigos VHDL gerados pelos programas PIPE2VHDL4M e PIPE2VHDL5M são os mesmos para a mesma máquina modelada, a diferença está nas metodologias de modelagem aplicadas e na descrição PNML obtidas por elas.

```
ENTITY Exemplo4 IS
        PORT
        (
                --xn representão os sinais de entrada
                --zn representão os sinais de saída
                clock:IN BIT;
                x0:IN BIT;
                z0:OUT BIT
        );
END Exemplo4;

ARCHITECTURE Detector OF Exemplo4 IS

        TYPE tipo_estado IS (A, B, C, D, E, F, G);
        SIGNAL Atual, Proximo: tipo_estado;
BEGIN

        PROCESS (Atual, x0)

        BEGIN
                CASE Atual IS
                        WHEN A =>
                                IF x0= '0' THEN
                                        z0<= '0';
                                        Proximo <=B;
                                ELSE
                                        z0<= '0';
                                        Proximo <=A;
                                END IF;
                        WHEN B =>
                                IF x0= '0' THEN
                                        z0<= '0';
                                        Proximo <=C;
                                ELSE
                                        z0<= '0';
                                        Proximo <=F;
                                END IF;
                        WHEN C =>
                                IF x0= '0' THEN
                                        z0<= '0';
                                        Proximo <=G;
                                ELSE
                                        z0<= '0';
                                        Proximo <=D;
                                END IF;
                        WHEN D =>
                                IF x0= '0' THEN
                                        z0<= '1';
                                        Proximo <=E;
                                ELSE
                                        z0<= '0';
                                        Proximo <=A;
                                END IF;
                        WHEN E =>
                                IF x0= '0' THEN
                                        z0<= '0';
```

Figura 6.8: Descrição Comportamental na linguagem VHDL criada pelo programa PIPE2VHDL4M.

Na figura 6.9 é mostrado o resultado da simulação da descrição VHDL comportamental realizada no ambiente Quartus II 6.0 da Altera. Pode-se observar que no instante 30 ns ocorre a mudança do clock (sensível à borda de subida), neste instante a entrada (x0) está no nível lógico 0, assim a máquina transita do estado “a” para o estado "b". Pode-se

notar que a saída permaneceu em nível lógico 0 durante esta transição. Já na transição seguinte, ocorrida no instante 50 ns, nota-se novamente a mudança do clock e a entrada em nível lógico 0, assim a máquina transita do estado “b” para o estado "c". No mesmo instante observa-se que a saída continua no nível lógico "0". No instante 90 ns ocorre outra mudança do clock, a máquina transita do estado “d” para o estado "e", neste instante observa-se que a saída está em nível lógico "1". Dessa forma pode-se constatar que a simulação está de acordo com o comportamento da descrição apresentada na figura 6.8, considerando-se o atraso de cerca de 2 ns para a mudança dos estados e das saídas da máquina.

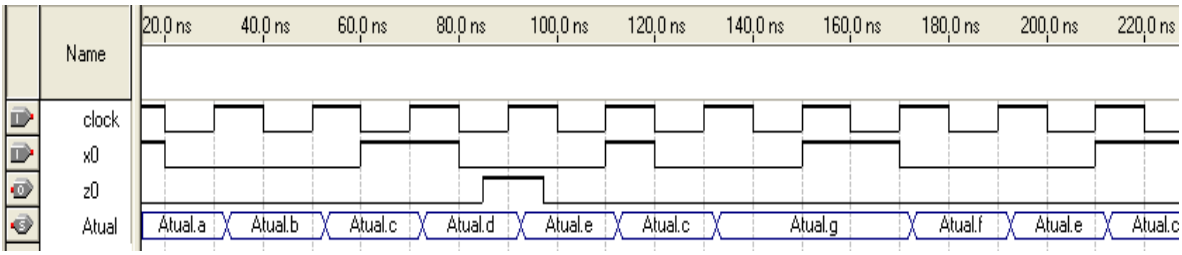


Figura 6.9: Simulação da descrição comportamental para o detector da seqüência 10010 com sobreposição.

6.1.2 Código de linha AMI

O código AMI (*Alternate Mark Inversion*) foi utilizado nas antigas linhas T1 (1,544 Mbps) e canais de 64Kbps americanos. O objetivo deste é eliminar o nível DC na linha de transmissão, assim transforma a informação digital em um sinal ternário onde os “zeros” são codificados pelo nível de tensão 0, enquanto que os “uns” são alternadamente codificados por dois níveis de tensão simétricos +V e –V [37].

O projeto do código de linha AMI codifica os pulsos 0 em “0”, cuja alocação foi adotada como “0”, e os pulsos 1, alternadamente em pulsos “+1” e “-1” onde se adotou respectivamente as alocações “1” e “2”. Na figura 6.10 é apresentado o diagrama de transição de estados para este código.

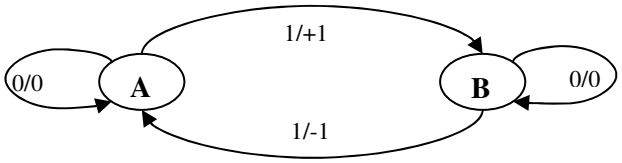


Figura 6.10: Diagrama de transição de estados para o código de linha AMI [37].

Observa-se no diagrama que quando ocorre a entrada de vários “uns” consecutivos a máquina alterna entre os estados A → B, B → A e as saídas entre “+1” e “-1”.

As informações contidas no diagrama de estados são transcritas para a modelagem em RdP, utilizando-se a metodologia de modelagem 4M. Na figura 6.11, apresenta-se a RdP Lugar/Transição correspondente à máquina mostrada na figura 6.10. Observa-se que os lugares A e B correspondem aos estados do diagrama que possuem a mesma denominação e os lugares E0 e E1 representam as entradas 0 e 1 respectivamente, enquanto S0, S1, S2 as saídas 0,1 e 2 que correspondem os pulsos 0, +1 e -1.

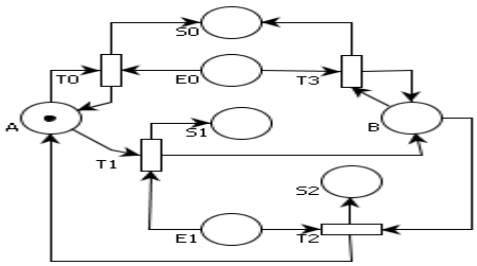


Figura 6.11: Modelo da RdP Lugar/Transição para o código de linha AMI, aplicando a metodologia 4M.

Utilizando-se a metodologia 5M para modelar o código de linha AMI, obtém-se a RdP Lugar/Transição ilustrada na figura 6.12. Nota-se que as saídas da máquina são representadas nas transições, ao invés de serem representadas por lugares como na RdP apresentada na figura 6.11. As saídas são indicadas de acordo com o estado atual e a entrada identificada. Observa-se na figura 6.12 que houve uma redução de cerca de 30% nos elementos gráficos desta RdP em relação a rede ilustrada na figura 6.11.

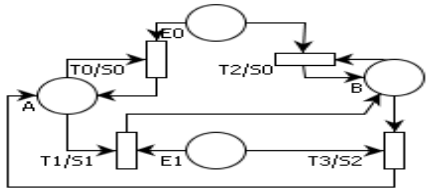


Figura 6.12: Modelo da RdP Lugar/Transição para o código de linha AMI, aplicando a metodologia 5M.

Os lugares A e B correspondem aos estados do diagrama, os lugares E0 e E1 as entradas 0 e 1 respectivamente e as saída são representadas nas transições da RdP.

Através das descrições PNML obtidas pelos modelos das RdP é possível utilizar os programas PIPE2TAB4M e PIPE2TAB5M, estes por sua vez geram as tabelas de transição de estados das RdP que modelam o código de linha AMI. A figura 6.13 mostra os arquivos gerados pelos programas PIPE2TAB4M e PIPE2TAB5M.

Na primeira linha especifica-se a quantidade mínima de elemento(s) de memória(s). No caso necessita-se no mínimo um elemento de memória, visto que o sistema contém somente dois estados. Na linha imediatamente abaixo se especifica o tipo de elemento de memória que será utilizado, neste caso o *flip-flop* tipo D foi informado para o arquivo gerado pelo programa PIPE2TAB4M e o tipo JK para o gerado pelo programa PIPE2TAB5M.

Menciona-se na terceira linha a quantidade de entradas e de saídas. O sistema projetado possui 1 porto de entrada e 2 portos de saídas.

Após a terceira linha são especificadas as transições dos estados e a saída gerada por cada um deles de acordo com a entrada.

(a) (b)

Figura 6.13: Arquivo gerado pelo programa PIPE2TAB4M (a) e PIPE2TAB5M (b) para o código de linha AMI.

O programa AGPS utiliza os arquivos gerados pelos PIPE2TAB4M e PIPE2TAB5M, como informação de entrada. O arquivo de saída criada pelo AGPS para a figura 6.13 (a) é mostrado na figura 6.14. Observa-se nesta que Figura os estados A e B receberam a alocação dos valores 1 e 0.

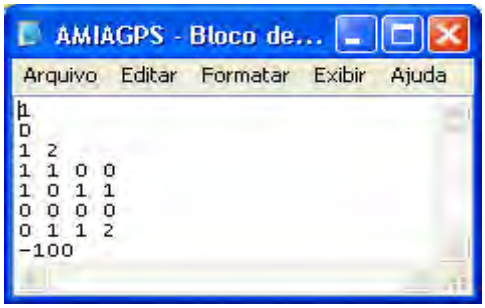


Figura 6.14: Arquivo com a tabela de transição de estados gerada pelo programa AGPS para o código de linha AMI.

A tabela gerada pelo AGPS é utilizada como arquivo de entrada no programa TABELA. Na figura 6.15 é apresentada a descrição obtida pelo programa TABELA, todos os mintermos e *don't care states* das funções booleanas e os seus respectivos implicantes primos são apresentados em cada função combinacional. Por exemplo, a função D0 é composta pelos mintermos 1e 2. Os implicantes primos geram as funções mínimas. O custo total para a implementação das três funções combinacionais (D0, Z1 e Z0) é igual a 10.

```
!DE!P/!ENTRADA  !MINT!!Q0!Q0+!D0!!Z1!Z0!
! 0! 1! 1 (1)   !    2!!  0!   1!  1!!  1!  0!
! 0! 0! 0 (0)   !    0!!  0!   0!  0!!  0!  0!
! 1! 0! 1 (1)   !    3!!  1!   0!  0!!  0!  1!
! 1! 1! 0 (0)   !    1!!  1!   1!  1!!  0!  0!

FUNCAO D0

MINTERMOS :   1;   2;


IMPLICANTES PRIMOS ESSENCIAIS :

ESSENCIAL:   2      REDUNDANCIA:         0 ->   10
ESSENCIAL:   1      REDUNDANCIA:         0 ->   01

CUSTO FINAL DE D0 = 6

FUNCAO Z1

MINTERMOS :   2;


IMPLICANTES PRIMOS ESSENCIAIS :

ESSENCIAL:   2      REDUNDANCIA:         0 ->   10

CUSTO FINAL DE Z1 = 2

FUNCAO Z0

MINTERMOS :   3;


IMPLICANTES PRIMOS ESSENCIAIS :

ESSENCIAL:   3      REDUNDANCIA:         0 ->   11

CUSTO FINAL DE Z0 = 2


CUSTO TOTAL DAS 3 FUNCOES = 10
```

Figura 6.15: Descrição gerada pelo programa Tabela para o código de linha AMI.

O arquivo gerado pelo programa TABELA é utilizado como entrada no programa TAB2VHDL, o qual gera a descrição RTL na linguagem VHDL para o código de linha AMI.

Na figura 6.16 é exibida a descrição. Observa-se que é criado na descrição da figura 6.16 apenas um processo para modelar o *flip-flop* tipo D.

```
ENTITY AMITAB IS
     PORT(
     -- CLK e CLR estao presentes em todos os modelos
     -- CLK e CLR estao presentes em todos os modelos
     CLK, CLR : IN BIT;

     -- Os parametros seguintes sao definidos de acordo
     -- com a descricao contida no programa tabela.
     -- Xn representam as variaveis de entrada
     -- Qn representam as variaveis de estado
     -- Zn representam as funcoes de saida
     X0 : IN BIT;
     Q0 : OUT BIT;
     Z0, Z1 : OUT BIT );
END AMITAB;

-- RTL e a designacao para todas as arquiteturas
ARCHITECTURE RTL OF AMITAB IS

-- VEn sao sinais auxiliares que assumem os mesmos valores
-- das variaveis de estado. Eles sao utilizados para permitir
-- um melhor modelamento do sistema
SIGNAL  VE0: BIT;

-- Jn, Kn e Dn representam as funcoes de controle e sao definidas
-- no arquivo gerado pelo Programa TABELA
SIGNAL D0 : BIT;


 BEGIN
-- Nesta parte do modelo teremos a descricao de cada um dos Flip-flops.
-- Deve-se observar que os sinais auxiliares sao utilizados, sendo que
-- no final de cada processo seus valores sao transferidos para as variaveis
-- de estados

-- Importante lembrtar que em relacao ao software, existem duas procedures,
-- uma que implementa o flip-flop D e outra que implementa o JK e estas procedures
-- recebem os indices que representam o respectivo elemento de memoria. Tal indice
-- definido no arquivo gerado pelo programa TABELA

-- Inferindo flip-flop tipo D
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;

 -- Processos que implementam as funcoes combinacionais de controle
  -- dos Elementos de memoria e de Saidas.

  D0 <=  ( (X0) AND NOT(VE0)) OR (NOT(X0) AND (VE0));
  Z1 <=  ( (X0) AND NOT(VE0));
  Z0 <=  ( (X0) AND (VE0));

END RTL;
```

Figura 6.16: Descrição RTL gerada pelo programa TAB2VHDL para o código AMI.

Através das descrições PNML, os programas PIPE2VHDL4M e PIPE2VHDL5M são utilizados a fim de obter uma descrição comportamental na linguagem VHDL para o código de linha AMI modelado em RdP Lugar/Transição, pelas metodologias de modelagem 4M e 5M. A figura 6.17 mostra a descrição comportamental obtida pelo programa

PIPE2VHDL4M, sendo esta idêntica à descrição gerada pelo PIPE2VHDL5M para o mesmo diagrama de estados. Nota-se que os portos de entrada (x0) e saída (z0 e z1) e o sinal de sincronismo (*clock*) são definidos dentro da entidade (*ENTITY*), na descrição comportamental apresentada na figura 6.17. Os estados da máquina (A e B) são representados dentro da arquitetura como um tipo de variável "tipo_estado". Dentro da estrutura *Case* é defina a mudança de estados e saídas da máquina descrita.

```
ENTITY AMI IS
        PORT
        (
                --xn representão as variáveis de entrada
                --zn representão as variáveis de saída
                clock:IN BIT;
                x0:IN BIT;
                z0, z1:OUT BIT
        );
END AMI;

ARCHITECTURE Codigo OF AMI IS

        TYPE tipo_estado IS (A, B);
        SIGNAL Atual, Proximo: tipo_estado;
BEGIN

        PROCESS (Atual, x0)

        BEGIN
                CASE Atual IS
                        WHEN A =>
                                IF x0= '0' THEN
                                        z0<= '0';
                                        z1<= '0';
                                       Proximo <=A;
                                ELSIF x0= '1' THEN
                                        z0<= '0';
                                        z1<= '1';
                                       Proximo <=B;
                                END IF;
                        WHEN B =>
                                IF x0= '0' THEN
                                        z0<= '0';
                                        z1<= '0';
                                      Proximo <=B;
                                ELSIF x0= '1' THEN
                                        z0<= '1';
                                        z1<= '0';
                                      Proximo <=A;
                                END IF;
                END CASE;
        END PROCESS;

        PROCESS
        BEGIN
                WAIT UNTIL clock'Event AND clock = '1';
                        Atual <= Proximo;
        END PROCESS;

END Codigo;
```

Figura 6.17: Descrição comportamental na linguagem VHDL para o código AMI.

A descrição obtida pelo PIPE2VHDL4M, é simulada tanto no ambiente Quartus II como no Max+Plus II. O resultado da simulação no Quartus II 6.0 é ilustrado na figura 6.18. Observa-se que os estados A e B assumiram durante a simulação o estado lógico 0 e 1. No instante 5 ns ocorre a mudança do clock (sensível à borda de subida), neste instante a entrada (x0) está no nível lógico 1, assim a máquina transita do estado "A" para o estado "B" e gera a saída z0="0" e z1="1". Na transição ocorrida no instante 10 ns não há transição do clock para

a borda de subida, assim a máquina permanece no estado “B” representado na simulação pelo nível lógico 1. No mesmo instante pode-se notar que a saída passa para o nível lógico z0=”1” e z1=”0”, pois a entrada neste instante é novamente 1 (x0), para esta transição considera-se o atraso de cerca de 2 ns. Já na transição seguinte, ocorrida no instante 65 ns, nota-se a mudança do clock para o nível lógico 1 e a entrada com o valor lógico 1, assim a máquina transita do estado “B” para o estado "A", neste instante observa-se que a saída passa para o nível lógico z0=”1” e z1=”0”. Analisando as transições apresentadas na figura 6.18 pode-se constatar que a simulação está de acordo com o comportamento da descrição apresentada na figura 6.17.

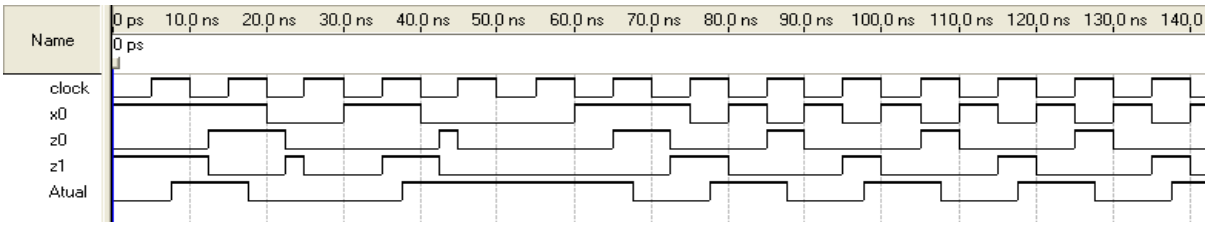


Figura 6.18: Simulação da descrição comportamental para o código de linha AMI.

6.1.3 FSM com duas entradas e uma saída

A FSM com duas entradas e uma saída apresentada nesta subseção, gera o valor lógico 1 como saída do circuito 1 se as entradas 01, 10 ou 11 forem detectadas a partir do seu estado inicial, neste caso representado pelo caractere “A”. A saída de valor lógico 0 sempre será atribuída à entrada 00.

Entretanto há uma restrição para a entrada “10”. Caso o detector constate anteriormente a ela, a entrada “00”, ele processará a saída como 0 na primeira vez que detectar a entrada 10, assim a saída será processada como 1 somente quando esta for identificada pela segunda vez consecutiva.

Na figura 6.19 é ilustrado o diagrama de transição de estados da FSM recém descrita. Os estados A e C geram a mesma saída, de valor lógico 1, para três entradas diferentes, 01, 10 e 11.

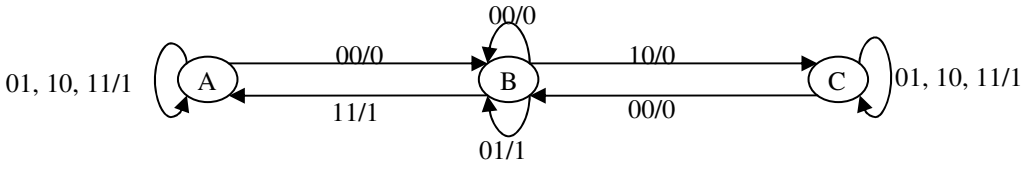


Figura 6.19: Diagrama de transição de estados para uma FSM com duas entradas e uma saída.

Para modelar a FSM apresentada na figura 6.19 em RdP Lugar/Transição foi necessário converter as entradas de valores binários para decimais, assim ao converter as entradas 00, 01, 10 e 11 em decimais obteve-se respectivamente os valores 0,1,2 e 3.

A modelagem da FSM em RdP Lugar/Transição, pela metodologia 4M está mostrada na figura 6.20. Nota-se que os estados A, B e C do diagrama são representados por lugares distintos na RdP e recebem a mesma nomenclatura atribuída no diagrama. As entradas de valores lógicos binários 00, 01, 10 e 11 convertidos para os valores decimais 0, 1, 2 e 3, são representados por E0, E1, E2, E3. Enquanto as saídas 0 e 1 correspondem respectivamente aos lugares S0 e S1.

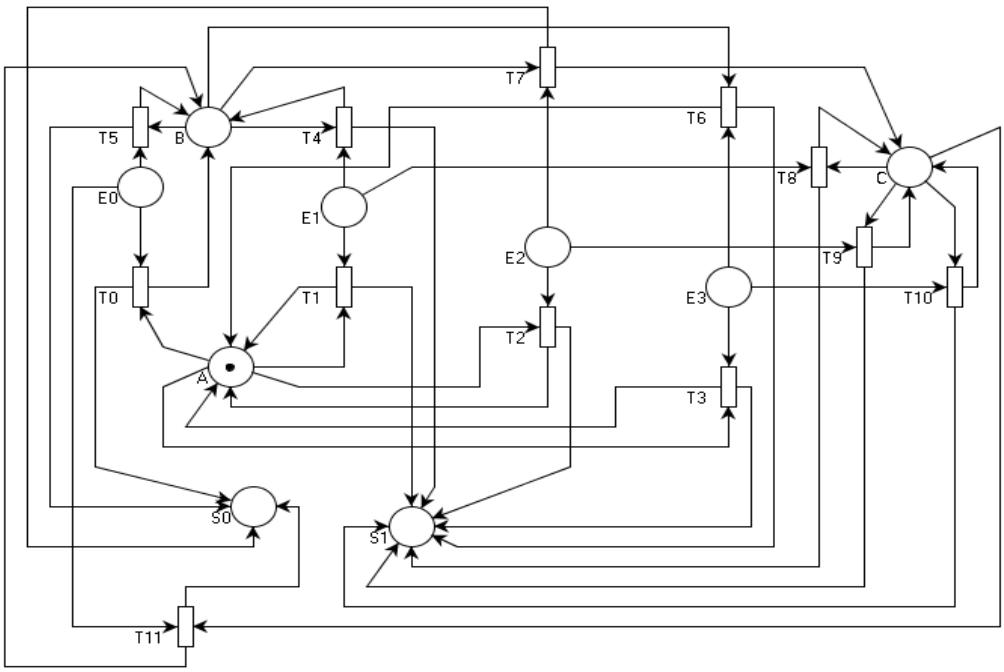


Figura 6.20: RdP Lugar/Transição para a FSM com duas entradas e uma saída, obtida pela metodologia 4M.

Por intermédio da metodologia de modelagem 5M é possível obter outro modelo de RdP Lugar/Transição para a mesma FSM descrita nesta subseção, na figura 6.21 é apresentada a RdP para o diagrama ilustrado na figura 6.19. Note-se que os lugares A, B e C da rede correspondem aos estados e os lugares E1, E2, E3 e E4 às entradas de valores decimais 1, 2, 3 e 4. As saídas na RdP são representadas junto às transições, conforme cada estado e entrada. A RdP apresentada na figura 6.21 apresenta uma redução de cerca de 26% dos elementos gráficos utilizados na modelagem aplicando-se a metodologia 5M em relação a rede ilustrada na figura 6.20 modelada através da metodologia 4M.

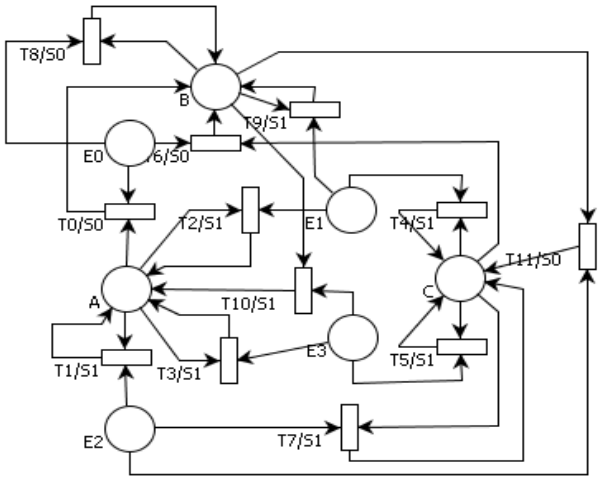
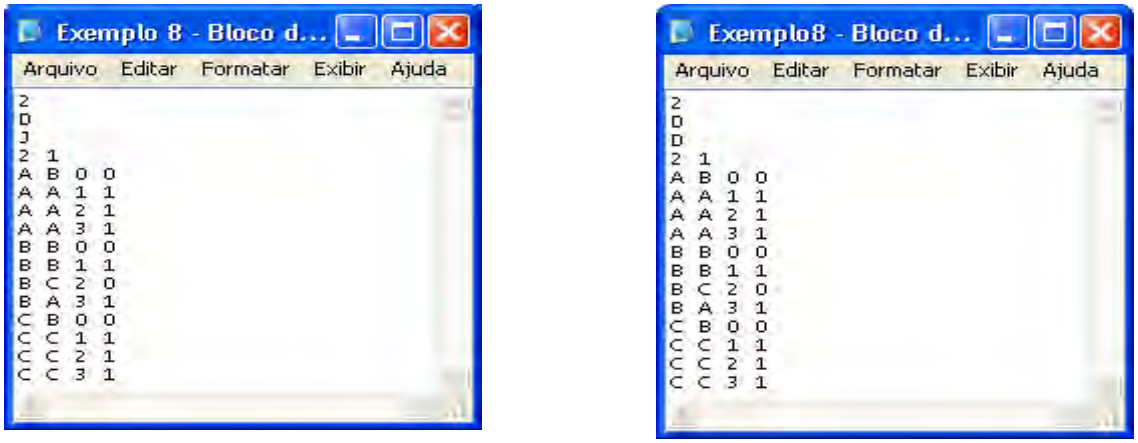


Figura 6.21: RdP Lugar/Transição para a FSM com duas entradas e uma saída, obtida pela metodologia 5M.

Os modelos esquematizados no ambiente PIPE geram descrições PNML distintas, que são utilizadas pelos programas PIPE2TAB4M e PIPE2TAB5M, estas são responsáveis por transcrever as informações das FSM do tipo Mealy especificadas na RdP Lugar/Transição em uma notação tabular, tal notação está representada na figura 6.22. Observa-se que na primeira linha da notação especificam-se dois elementos de memória, que são mencionados nas duas linhas abaixo, como do tipo D e JK para o 1º e 2º *flip-flop* da figura 6.22(a) e como do tipo D para os *flip-flops* da figura 6.22(b). Na quarta linha têm-se especificados dois portos de entrada e um porto de saída.



(a) (b)

Figura 6.22: Arquivos gerados pelas ferramentas PIPE2TAB4M e PIPE2TAB5M para a FSM com duas entradas e uma saída.

A configuração dos arquivos gerados pelos programas PIPE2TAB4M e PIPE2TAB5M é utilizada como entrada pelo programa AGPS. O arquivo obtido pelo programa AGPS é apresentada na figura 6.23. Observa-se no arquivo que os estados A, B e C da figura 6.22 receberam respectivamente as alocações 2, 0 e 1.

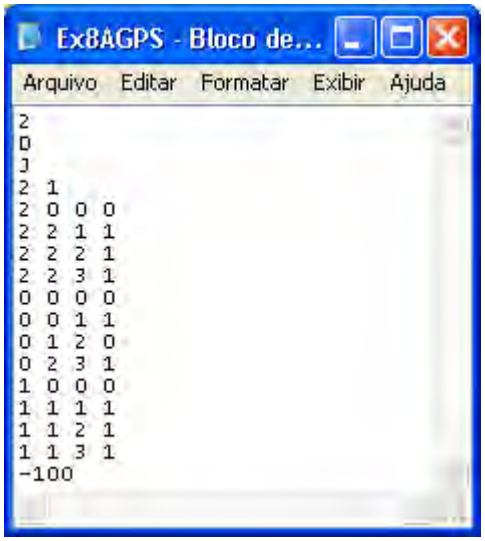


Figura 6.23: Arquivo gerado pelo programa AGPS para a FSM com duas entradas e uma saída.

O arquivo mostrado na figura 6.23 é utilizado como entrada pelo programa TABELA. Assim, a descrição da FSM, gerada pelo TABELA está exibida na figura 6.24. Onde vemos que todos os mintermos e *don't care states* das funções booleanas e os seus respectivos implicantes primos são apresentados em cada função combinacional. Por exemplo, a função K1 é composta pelo mintermo 2 e pelos *don't care states* 0, 4, 8, 12, 1, 5, 9, 13, 15, 11, 7 e 3, o custo final para a implementação desta função é igual a 2. As

funções combinacionais J1, D0 e Z0 possuem o custo final da sua implementação igual a 3, 7 e 8. Totalizando a 20 o custo para a implementação das quatro funções. O custo total de todas as funções é obtido através da soma do custo final de cada função. Os *don't care states* gerais são apresentados em forma de tabela na descrição obtida pelo programa TABELA.

A descrição gerada pelo programa TABELA é utilizado como entrada pelo programa TAB2VHDL, assim sendo, obtém-se a descrição RTL na linguagem VHDL para a FSM citada na figura 6.19.

```
DON'T CARE STATES GERAIS :
!DE!P/!ENTRADA  !DONT!!Q1!Q1+!J1!K1!!Q0!Q0+!D0!!Z0!
! 3! X! 0 (00)  !   3!! 1! X ! X! X!! 1! X ! X!! X!
! 3! X! 1 (01)  !   7!! 1! X ! X! X!! 1! X ! X!! X!
! 3! X! 2 (10)  !  11!! 1! X ! X! X!! 1! X ! X!! X!
! 3! X! 3 (11)  !  15!! 1! X ! X! X!! 1! X ! X!! X!

FUNCAO J1
=========

MINTERMOS : 12;

DON'T CARE STATES :  2;  6; 10; 14; 15; 11;  7;  3;

IMPLICANTES PRIMOS ESSENCIAIS :

ESSENCIAL: 12     REDUNDANCIA:       2 ->  11XO

CUSTO FINAL DE J1 = 3

FUNCAO K1
=========

MINTERMOS :  2;

DON'T CARE STATES :  O;  4;  8; 12;  1;  5;  9; 13; 15; 11;  7;  3;

IMPLICANTES PRIMOS ESSENCIAIS :

ESSENCIAL:  O     REDUNDANCIA:       3 ->  OOXX

CUSTO FINAL DE K1 = 2

FUNCAO DO
=========

MINTERMOS :  8;  5;  9; 13;

DON'T CARE STATES : 15; 11;  7;  3;

IMPLICANTES PRIMOS ESSENCIAIS :

ESSENCIAL:  5     REDUNDANCIA:      10 ->  X1X1
ESSENCIAL:  8     REDUNDANCIA:       1 ->  100X

CUSTO FINAL DE DO = 7

FUNCAO ZO
=========

MINTERMOS :  6; 10; 14;  4; 12;  5;  9; 13;

DON'T CARE STATES : 15; 11;  7;  3;

IMPLICANTES PRIMOS ESSENCIAIS :

ESSENCIAL:  9     REDUNDANCIA:       6 ->  1XX1
ESSENCIAL: 10     REDUNDANCIA:       5 ->  1X1X
ESSENCIAL:  4     REDUNDANCIA:      11 ->  X1XX

CUSTO FINAL DE ZO = 8


CUSTO TOTAL DAS 4 FUNCOES = 20
```

Figura 6.24: Descrição gerada pelo programa Tabela para a FSM com duas entradas e uma saída.

Parte do código determinado pelo TAB2VHDL está exibida na figura 6.25. Nota-se que na descrição da entidade apresentada no código RTL da figura 6.25, foram definidos os portos de entrada (X0, X1) e saída (Z0) e os sinais de sincronismo (CLK e CLR). Dois tipos de processos foram criados na descrição da figura 6.25, um para modelar o *flip-flop* D e outro para modelar o *flip-flop* JK, ambos os processos são sensíveis a transição de subida.

Arquivo Editar Formatar Exibir Ajuda

```
ENTITY Ex8TAB IS
    PORT(
    CLK, CLR : IN BIT;
    X0, X1 : IN BIT;
    Q0, Q1 : OUT BIT;
    Z0 : OUT BIT );
END Ex8TAB;

ARCHITECTURE RTL OF Ex8TAB IS

SIGNAL  VE0, VE1: BIT;
SIGNAL J1, K1, D0 : BIT;

 BEGIN

-- Inferindo flip-flop tipo D
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;

-- Inferindo flip-flop tipo JK
PROCESS(CLK, CLR)
BEGIN
   IF CLR = '0' THEN
      VE1 <= '0';
   ELSIF CLK'EVENT and CLK = '1' THEN
      IF J1 = '0' and K1 = '1' THEN
         VE1 <= '0';
      ELSIF J1 = '1' and K1 = '0' THEN
         VE1 <= '1';
      ELSIF J1 = '1' and K1 = '1' THEN
         VE1 <= not(VE1);
      END IF;
   END IF;
   Q1 <= VE1;
END PROCESS;

 J1 <=  ( (X1) AND (X0) AND NOT(VE0));
 K1 <=  ( NOT(X1) AND NOT(X0));
 D0 <=  ( (X0) AND (VE0)) OR ((X1) AND NOT(X0) AND NOT(VE1));
 Z0 <=  ( (X1) AND (VE0)) OR ((X1) AND (VE1)) OR ((X0));

END RTL;
```

Figura 6.25: Descrição RTL criada pelo programa TAB2VHDL para FSM com duas entradas e uma saída.

Por fim, utilizam-se os programas PIPE2VHDL4M e PIPE2VHDL5M para criar a descrição comportamental da FSM com duas entradas e uma saída, na linguagem VHDL. O

emprego destes programas possibilita a síntese da máquina especificada em RdP Lugar/Transição utilizando a metodologia 4M e/ou 5M.

A descrição obtida pelo programa PIPE2VHDL4M é apresentada na figura 6.26, esta descrição é idêntica à gerada pelo PIPE2VHDL5M, pois a mesma FSM é modela utilizando as metodologias 4M e 5M. Ao gerar a descrição o programa converte as entradas e saídas da FSM, assim as entradas 0, 1, 2 e 3 são tratadas com 00, 01, 10 e 11.

Arquivo Editar Formatar Exibir Ajuda

```
PROCESS (Atual, x0, x1)

BEGIN
        CASE Atual IS
                WHEN A =>
                        IF x0= '0' and x1= '0' THEN
                                z0<= '0';
                                Proximo <=B;
                        EISIF x0= '0' and x1= '1' THEN
                                z0<= '1';
                                Proximo <=A;
                        EISIF x0= '1' and x1= '0' THEN
                                z0<= '1';
                                Proximo <=A;
                        EISIF x0= '1' and x1= '1' THEN
                                z0<= '1';
                                Proximo <=A;
                        END IF;
                WHEN B =>
                        IF x0= '0' and x1= '0' THEN
                                z0<= '0';
                                Proximo <=B;
                        ELSIF x0= '0' and x1= '1' THEN
                                z0<= '1';
                                Proximo <=B;
                        ELSIF x0= '1' and x1= '0' THEN
                                z0<= '0';
                                Proximo <=C;
                        EISIF x0= '1' and x1= '1' THEN
                                z0<= '1';
                                Proximo <=A;
                        END IF;
                WHEN C =>
                        IF x0= '0' and x1= '0' THEN
                                z0<= '0';
                                Proximo <=B;
                        ELSIF x0= '0' and x1= '1' THEN
                                z0<= '1';
                                Proximo <=C;
                        ELSIF x0= '1' and x1= '0' THEN
                                z0<= '1';
                                Proximo <=C;
                        EISIF x0= '1' and x1= '1' THEN
                                z0<= '1';
                                Proximo <=C;
                        END IF;
        END CASE;
END PROCESS;
```

Figura 6.26: Descrição comportamental obtida pelo programa PIPE2VHDL4M.

Na figura 6.27 é exibido o resultado da simulação realizada no ambiente Max+Plus II 10.2 da Altera, para a descrição VHDL comportamental da FSM com duas

entradas e uma saída. Os valores 0, 1 e 2 correspondem aos estados A, B e C do modelo. Assim no instante 25 ns ocorre a mudança do clock (sensível à borda de subida), neste instante a entrada está no nível lógico x0=”0” e x1=”0”, assim a máquina transita do estado “A” para o estado "B" e gera a saída z0=”0”. Na transição ocorrida no instante 75 ns ocorre outra mudança no clock e a entrada possui valor lógico x0=”1” e x1=”0”, assim a máquina transita do estado B para o C e gera a saída de valor lógico 0. Analisando as transições da figura 6.27 pode-se constar que a simulação ocorre conforme a descrição apresentada na figura 6.26.

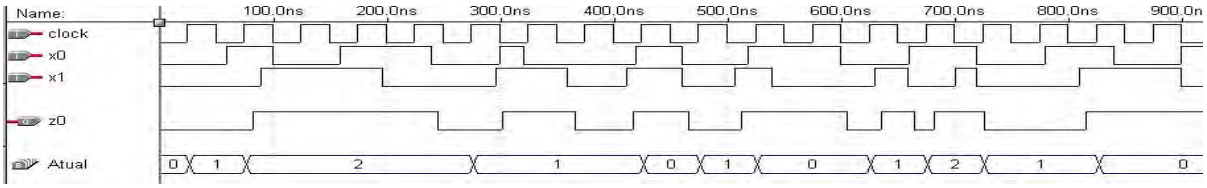


Figura 6.27: Simulação da descrição comportamental para a FSM com duas entradas e uma saída.

6.1.4 Comparador de série

O comparador de série, descrito nesta subseção, foi projetado para comparar o maior valor entre dois números binários, A e B. Os números são recebidos em série (bit a bit, AB), onde o bit de menor peso (B) é recebido primeiro. O comparador analisa se o segundo *bit* recebido (A) é igual, maior ou menor que o anterior.

Na figura 6.28 é apresentada a máquina de Mealy que exibe o funcionamento do comparador de série através do diagrama de estados. Nota-se que, caso os dois bits de entrada sejam iguais, a máquina irá gerar a saída de valor lógico 100 e transitará para o estado Igual. Se o segundo bit for menor à saída fornecida será 001, a máquina deslocar-se-á para o estado Menor. Se o segundo bit for maior, a saída fornecida terá o valor 010 e a máquina irá para o estado Maior.

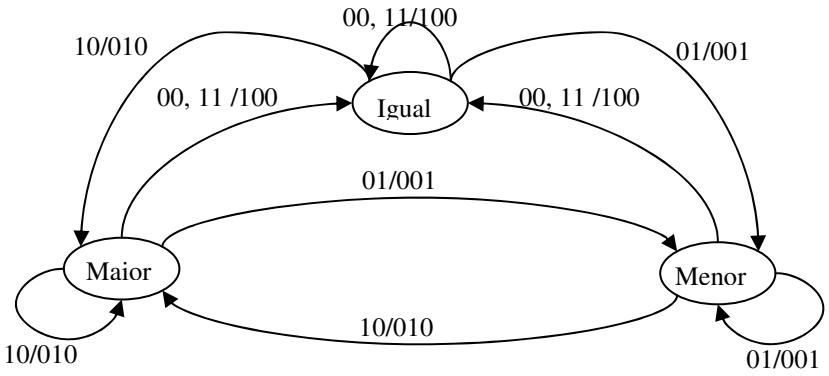


Figura 6.28 : Máquina de Mealy para o comparador de série.

As informações contidas na máquina de Mealy são modeladas em RdP, utilizando- se a metodologia de modelagem 4M e 5M.

Na figura 6.29, apresenta-se a RdP Lugar/Transição, obtida através da metodologia 4M, correspondente a máquina mostrada na figura 6.28. Observa-se que os lugares Igual, Menor e Maior representam os estados da máquina, enquanto que os lugares E0, E1, E2 e E3 as entradas que correspondem aos valores binários 00, 01, 10 e 11. As saídas 001, 010 e 100 são representadas pelos lugares S1, S2 e S4 da rede. A marca existente no lugar Igual representa a marcação inicial da RdP e o estado inicial da FSM do tipo Mealy.

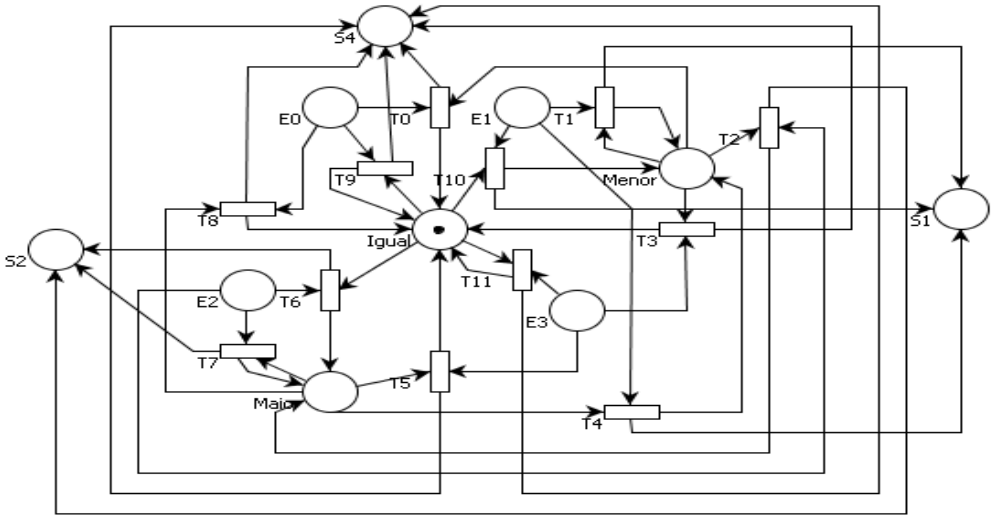


Figura 6.29: Modelagem em RdP Lugar/Transição, obtida pela metodologia 4M, para o comparador de série.

Apresenta-se na figura 6.30 a RdP Lugar/Transição do comparador de série, obtida utilizando-se a metodologia de modelagem 5M. Os lugares Igual, Menor e Maior representam os estados do diagrama e os lugares E0, E1, E2 e E3 as entradas. As saídas estão

representadas junto às transições. Observa-se que houve uma redução de cerca de 20% dos elementos gráficos na modelagem da RdP da figura 6.30 em relação a rede da figura 6.29.

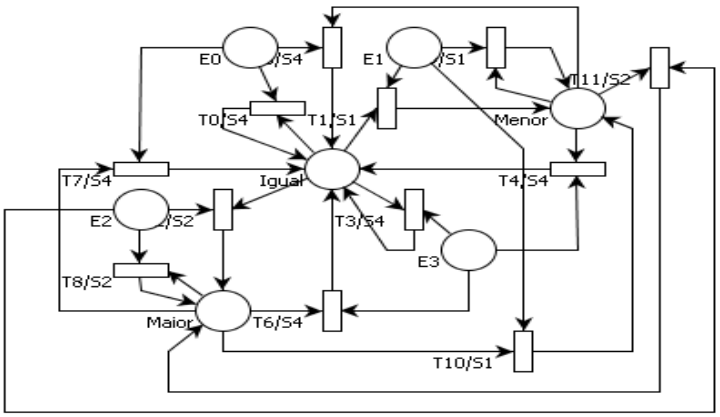
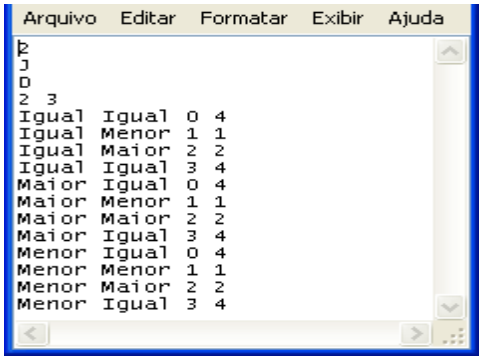


Figura 6.30: Modelagem em RdP Lugar/Transição para o comparador de série, obtida pela metodologia 5M.

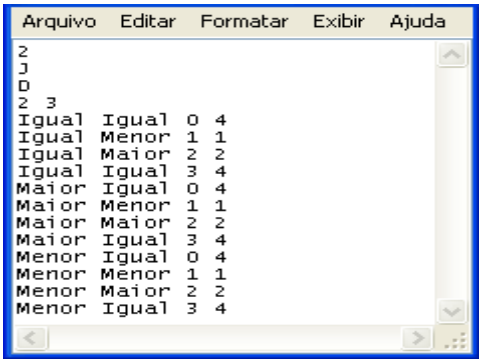
Os programas PIPE2TAB4M e PIPE2TAB5M utilizam as descrições PNML obtidas pelos modelos das RdP como entrada, estes por sua vez geram as tabelas de transição de estados das RdP que modelam o comparador de série. A figura 6.31 mostra os arquivos gerados pelos programas. Observa-se que os dois elementos de memórias mencionados nos arquivos da figura 6.31(a) e 6.31(b) são respectivamente do tipo JK e D para o 1º e 2º *flip-flop*.

Especifica-se na quarta linha a quantidade de entradas e saídas do comparador. Têm-se dois portos de entrada e três portos de saída.

A partir da quinta linha são especificadas as transições dos estados e a saída gerada por cada um deles de acordo com cada entrada.



(a)



(b)

Figura 6.31: Arquivos gerados pelos programas PIPE2TAB4M(a) e PIPE2TAB5M(b) para o comparador de série.

A configuração dos arquivos gerados pelos PIPE2TAB4M e PIPE2TAB5M é utilizada como entrada pelo programa AGPS. A descrição da tabela obtida pelo AGPS para o arquivo 6.31 está ilustrada na figura 6.32. Observa-se que o AGPS aloca para os estados Igual, Maior e Menor os valores decimais 1, 3 e 0, nesta ordem.

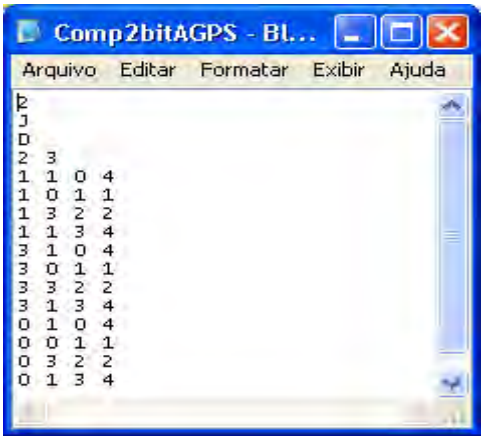


Figura 6.32: Arquivo gerado pelo programa AGPS.

O programa TABELA utiliza o arquivo mostrado na figura 6.32 como arquivo de entrada. Assim a descrição do comparador, gerada pelo TABELA está exibida na figura 6.33. Nota-se que todos os mintermos e *don't care states* das funções booleanas e os seus respectivos implicantes primos são apresentados em cada função combinacional. Por exemplo, a função K0 é composta pelos mintermos 5 e 7 e pelos *don't care states* 0, 4, 8, 12, 14, 10, 6, 2 respectivamente, o custo final para a implementação desta função é igual a 2. As funções combinacionais D1, J0, K0, Z2, Z1 e Z0 possuem o custo final da sua implementação igual a 2, 4, 2, 6, 2 e 2. O custo total para a implementação das seis funções é igual a 18. O custo total de todas as funções é obtido através da soma do custo de cada função.

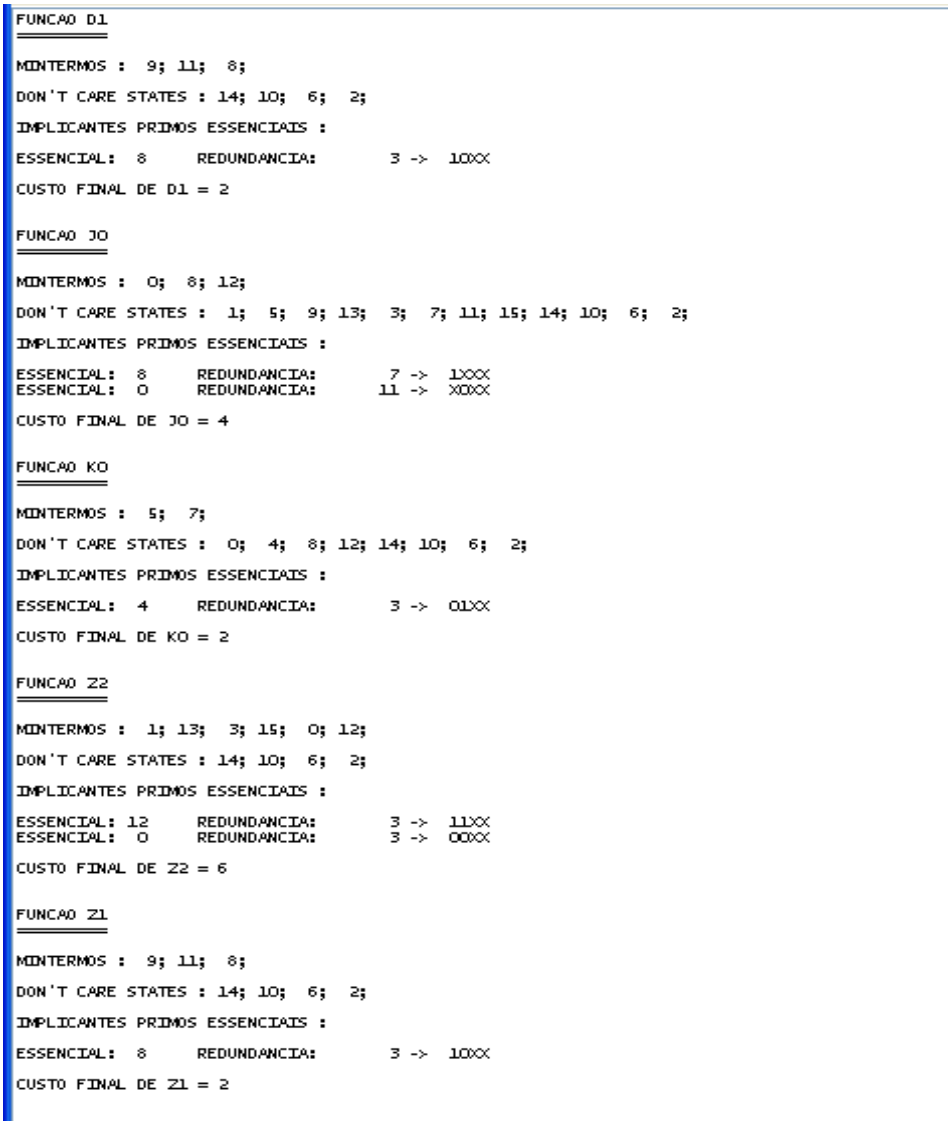


Figura 6.33: Descrição gerada pelo programa TABELA para o comparador de série.

A descrição gerada pelo programa TABELA é utilizada como entrada pelo programa TAB2VHDL, assim sendo, obtém-se a descrição RTL na linguagem VHDL para o comparador de série ilustrado na figura 6.28.

Na figura 6.34 é exibida a descrição RTL gerada pelo programa TAB2VHDL. Observa-se nesta figura que dois tipos de processos foram criados para modelar os elementos de memória, o primeiro para modela o *flip-flop* JK e o segundo para modelar o *flip-flop* D, ambos os processos são sensíveis a transição de subida.

Arquivo Editar Formatar Exibir Ajuda

```
ENTITY Comp2bit IS
    PORT(
    CLK, CLR : IN BIT;
    X0, X1 : IN BIT;
    Q0, Q1 : OUT BIT;
    Z0, Z1, Z2 : OUT BIT );
END Comp2bit;

ARCHITECTURE RTL OF Comp2bit IS

SIGNAL  VE0, VE1: BIT;
SIGNAL D1, J0, K0 : BIT;

 BEGIN
-- Inferindo flip-flop tipo JK
PROCESS(CLK, CLR)
BEGIN
   IF CLR = '0' THEN
      VE0 <= '0';
   ELSIF CLK'EVENT and CLK = '1' THEN
      IF J0 = '0' and K0 = '1' THEN
         VE0 <= '0';
      ELSIF J0 = '1' and K0 = '0' THEN
         VE0 <= '1';
      ELSIF J0 = '1' and K0 = '1' THEN
         VE0 <= not(VE0);
      END IF;
   END IF;
   Q0 <= VE0;
END PROCESS;

-- Inferindo flip-flop tipo D
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 <=  ( (X1) AND NOT(X0));
  J0 <=  ( (X1)) OR (NOT(X0));
  K0 <=  ( NOT(X1) AND (X0));
  Z2 <=  ( (X1) AND (X0)) OR (NOT(X1) AND NOT(X0));
  Z1 <=  ( (X1) AND NOT(X0));
  Z0 <=  ( NOT(X1) AND (X0));
END RTL;
```

Figura 6.34: Parte da descrição gerada pelo programa TAB2VHDL.

Por fim utilizam-se os programas PIPE2VHDL4M e PIPE2VHDL5M para obter as descrições comportamentais na linguagem VHDL do comparador de série, através da descrição PNML obtida pelo ambiente PIPE e pelas metodologias 4M e 5M. A figura 6.35 mostra a descrição comportamental obtida pelo PIPE2VHDL5M, esta descrição é idêntica à gerada pelo PIPE2VHDL4M, pois a mesma FSM é modela utilizando as metodologias distintas (4M e 5M). Observa-se que os portos de entrada (x0 e x1) e saída (z0, z1 e z2) e o sinal de sincronismo (*clock*) são definidos dentro da entidade (*ENTITY*), na descrição comportamental apresentada na figura 6.35. Os estados da máquina (Igual, Maior e Menor)

são representados dentro da arquitetura como um tipo de variável definida “tipo_estado”. Na estrutura *Case* é definida a mudança de estados e saídas da máquina descrita.

```
Arquivo  Editar  Formatar  Exibir  Ajuda
ENTITY Comp2bit1 IS
        PORT
        (
                --xn representão as variáveis de entrada
                --zn representão as variáveis de saída
                clock:IN BIT;
                x0, x1:IN BIT;
                z0, z1, z2:OUT BIT
        );
END Comp2bit1;

ARCHITECTURE Comparador OF Comp2bit1 IS

        TYPE tipo_estado IS (Igual, Maior, Menor);
        SIGNAL Atual, Proximo: tipo_estado;
BEGIN

        PROCESS (Atual, x0, x1)

        BEGIN
                CASE Atual IS
                        WHEN Igual =>
                                IF x0= '0' and x1= '0' THEN
                                        z0<= '1';
                                        z1<= '0';
                                        z2<= '0';
                                        Proximo <=Igual;
                                ELSIF x0= '0' and x1= '1' THEN
                                        z0<= '0';
                                        z1<= '0';
                                        z2<= '1';
                                        Proximo <=Menor;
                                ELSIF x0= '1' and x1= '0' THEN
                                        z0<= '0';
                                        z1<= '1';
                                        z2<= '0';
                                        Proximo <=Maior;
                                ELSIF x0= '1' and x1= '1' THEN
                                        z0<= '1';
                                        z1<= '0';
                                        z2<= '0';
                                        Proximo <=Igual;
                                END IF;
                        WHEN Maior =>
                                IF x0= '0' and x1= '0' THEN
                                        Begin
                                        z0<= '1';
                                        z1<= '0';
                                        z2<= '0';
                                        Proximo <=Igual;
                                        End
                                ELSIF x0= '0' and x1= '1' THEN
                                        z0<= '0';
                                        z1<= '0';
                                        z2<= '1';
                                        Proximo <=Menor;
                                ELSIF x0= '1' and x1= '0' THEN
                                        z0<= '0';
                                        z1<= '1';
                                        z2<= '0';
                                        Proximo <=Maior;
```

Figura 6.35: Descrição comportamental obtida pelo programa PIPE2VHDL4M.

Na figura 6.36 é exibido o resultado da simulação realizada no ambiente Max+Plus II 10.2 da Altera, para a descrição VHDL comportamental do comparador de série. Os valores 0, 1 e 2 correspondem aos estados Igual, Menor e Maior. Observa-se na figura 6.36 que no instante 150ns ocorre a mudança do clock (sensível a borda de subida) e a entrada neste instante tem valor lógico x0=’1’ e x1=’0’, assim a máquina transita do estado Igual (0) para o Menor (1). Já na transição do estado Maior (2) para o Menor (1), ocorrida no instante 350ns incide devido a mudança do clock para o valor lógico 1 e pela entrada ser x0=’1’ e x1=’0’. Já no instante 450 ns com uma nova mudança no clock a máquina transita do estado

Menor (1) para o Igual (0), sendo que a entrada neste instante é x0=’0’ e x1=’0’ e a saída da máquina passa a ser z2=”0’, z1=”0” e z0= “1”. Considerando-se o atraso de 2ns.

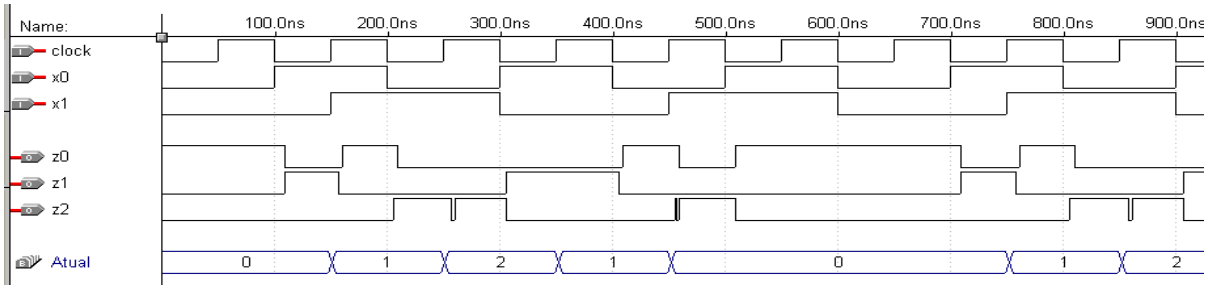


Figura 6.36: Simulação da descrição comportamental para o comparador de série.

6.1.5 Código de Linha HDB3

O Código HDB3 (*High Desinty Bipolar 3*) é uma técnica de sinalização bipolar, ou seja, depende tanto dos pulsos positivos quanto dos negativos.

O HDB3 é uma derivação do AMI (*Alternate Mark Inversion*), onde o "zero lógico" é representado pela não alteração da polaridade e o "um lógico" pela alternância da polaridade. As regras de codificação seguem as do AMI, com exceção de quando surge uma seqüência de quatro zeros consecutivos onde é utilizado um bit especial de violação, isto é, 4 zeros consecutivos são substituídos pela seqüência B00V ou 000V. Isto previne longas seqüências de zeros no fluxo de dados. É utilizado em taxas de transmissão de 2,8 e 34 Mb/s.

Na figura 6.37 é mostrado o diagrama de estado do código HDB3 em hexadecimal [37]. Nos nós do diagrama são representados os estados, esta representação é dada em forma hexadecimal. Cada estado possui dois tipos de entradas e uma saída para cada entrada fornecida. Por exemplo, se o estado atual for 10 e este processar a entrada 0, a saída + (01) é gerada e o estado atual passará a ser 0C, caso contrário a saída gerada é 0 e novo estado será 1C.

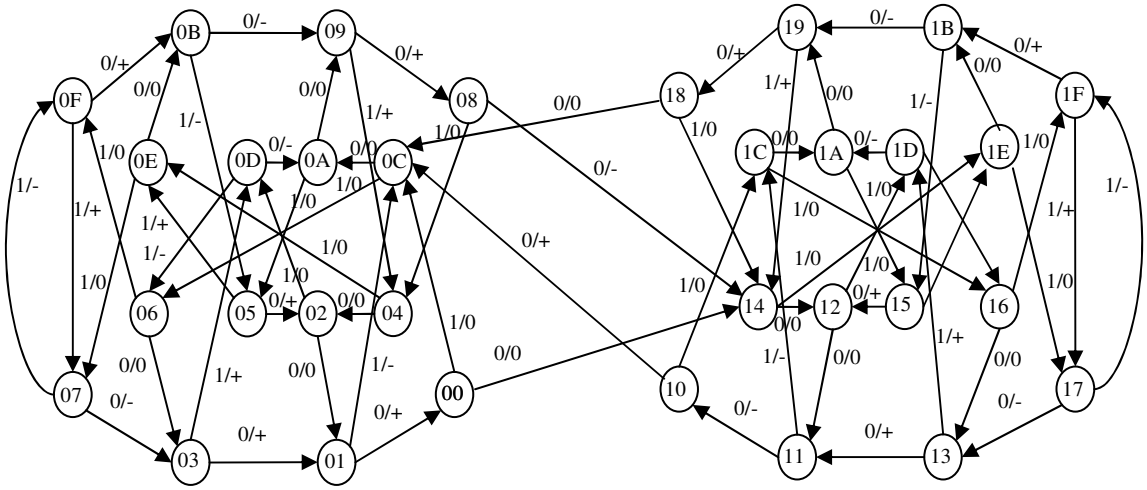


Figura 6.37: Diagrama de estado do código HDB3 em Hexadecimal [40].

Transforma-se o diagrama de estado (de representação hexadecimal) para a representação decimal, devido a sua modelagem em RdP Lugar/Transição, que será utilizada pelos programas PIPE2TAB5M e PIPE2VHDL5M. A tabela 6.2 apresenta essa transformação.

Tabela 6.2: Tabela de Conversão de Hexadecimal para Decimal

HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
00	0	08	8	10	16	18	24
01	1	09	9	11	17	19	25
02	2	0A	10	12	18	1A	26
03	3	0B	11	13	19	1B	27
04	4	0C	12	14	20	1C	28
05	5	0D	13	15	21	1D	29
06	6	0E	14	16	22	1E	30
07	7	0F	15	17	23	1F	31

Para a conversão do nível de tensão +, - e 0 estabeleceu-se que o binário 0 representará o sinal + e o binário 1 o sinal -, portanto +1 será 01 e -1 será 11 em binário, convertendo para decimal, tem-se 1 e 3.

A modelagem em RdP Lugar/Transição do código de linha HDB3 foi efetuada utilizando-se a metodologia 5M e o ambiente PIPE. Utilizou-se apenas a metodologia 5M, devido a sua capacidade de modelagem ser superior, pois permite a modelagem de FSM com mais estados. Como o HDB3 trata-se de uma FSM com 32 estados, modela-la utilizando a metodologia 4M, torna inviável para a análise do modelo da RdP. Caso o HDB3 fosse

modelado utilizando a metodologia 4M, seriam necessários 37 lugares para representar seus estados, entradas e saídas, 64 transições para efetuar a mudança de estados e 256 arcos para conectar o estado atual e a entrada à transição e a transição ao próximo estado e a saída da máquina. Enquanto, na metodologia 5M utilizaram-se 34 lugares, 64 transições e 192 arcos.

Na figura 6.38 é apresentada a modelagem em RdP Lugar/Transição para o código de linha HDB3. Os arcos que ligam os lugares de entrada às transições que representam a saída ficaram sobrepostos devido ao espaço de modelagem disponibilizado pelo ambiente PIPE e a disposição dos lugares de estados que utilizam a mesma entrada.

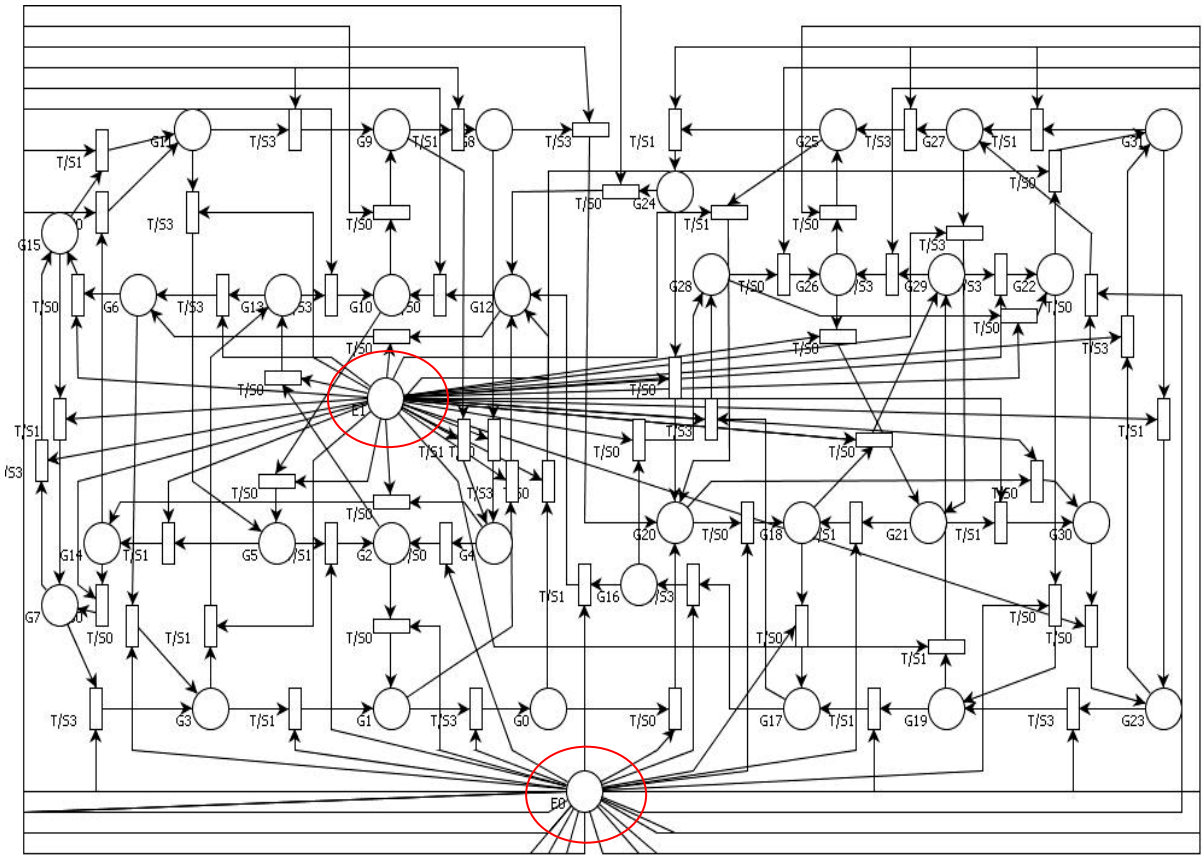


Figura 6.38: Modelagem do código HDB3 em RdP.

Observa-se na figura 6.38 que as entradas da máquina são representadas pelos lugares nomeados E0 e E1, que correspondem às entradas de valores lógicos 0 e 1 respectivamente. Os 32 estados são representados por lugares nomeados com o caractere “G” e acompanhado de um número decimal que corresponde ao valor hexadecimal apresentado nos estados do diagrama ilustrado na figura 6.37. Por se tratar de uma máquina de Mealy as saídas são

representadas nas transições. As saídas 00, 01 e 10 são representadas nas transições da RdP por S0, S1 e S3, que correspondem aos valores decimais 0,1 e 3.

A descrição PNML gerada pelo PIPE, para a RdP Lugar/Transição da figura 6.38 que modela o código de linha HDB3 é utilizada como arquivo de entrada pelo programa PIPE2TAB5M, para gerar a tabela de transição de estados. A tabela gerada pelo programa é apresentada na figura 6.39. Observa-se na figura que a quantidade mínima de *flip-flops* utilizado para representar o HDB3 é cinco, sendo atribuído os tipos D, D, JK, D e D.

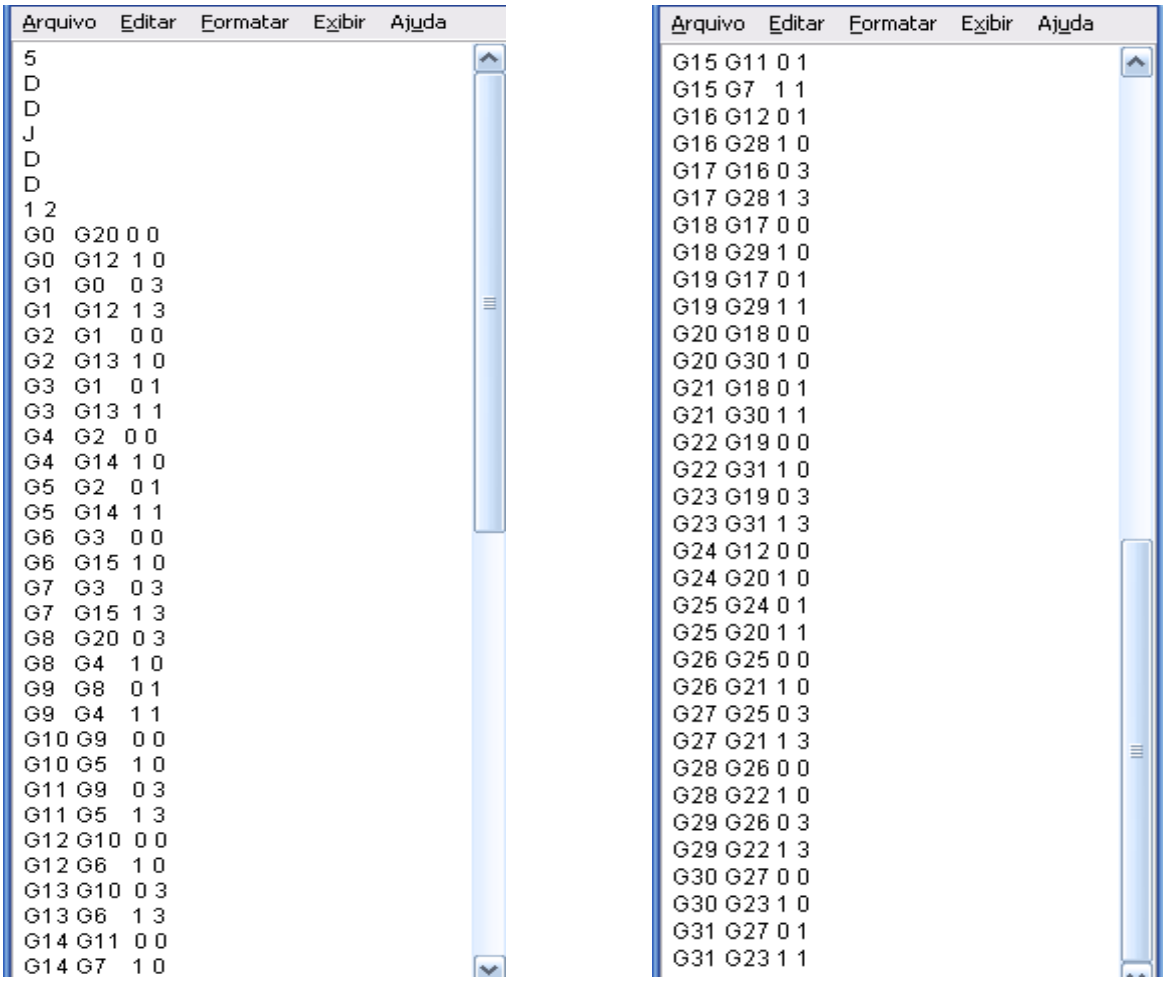


Figura 6.39: Tabela gerada pelo programa PIPE2TAB5M para o código de linha HDB3.

O arquivo gerado pelo programa PIPE2TAB5M é utilizado como entrada pelo programa AGPS. A descrição da tabela obtida pelo AGPS é mostrada na figura 6.40. Observa-se na tabela que os estados G0, G1, G2, G3, G4, G5, G6, G7, G8, G9, G10, G11,

G12, G13, G14, G15, G16, G17, G18, G19, G20, G21, G22, G23, G24, G25, G26, G27, G28, G29, G30, G31 receberam respectivamente as alocações 26, 19, 4, 0, 12, 8, 6, 18, 9, 17, 10, 16, 22, 24, 2, 1, 13, 25, 30, 7, 14, 5, 15, 23, 3 , 29, 27, 21, 31, 11, 20 e 28.

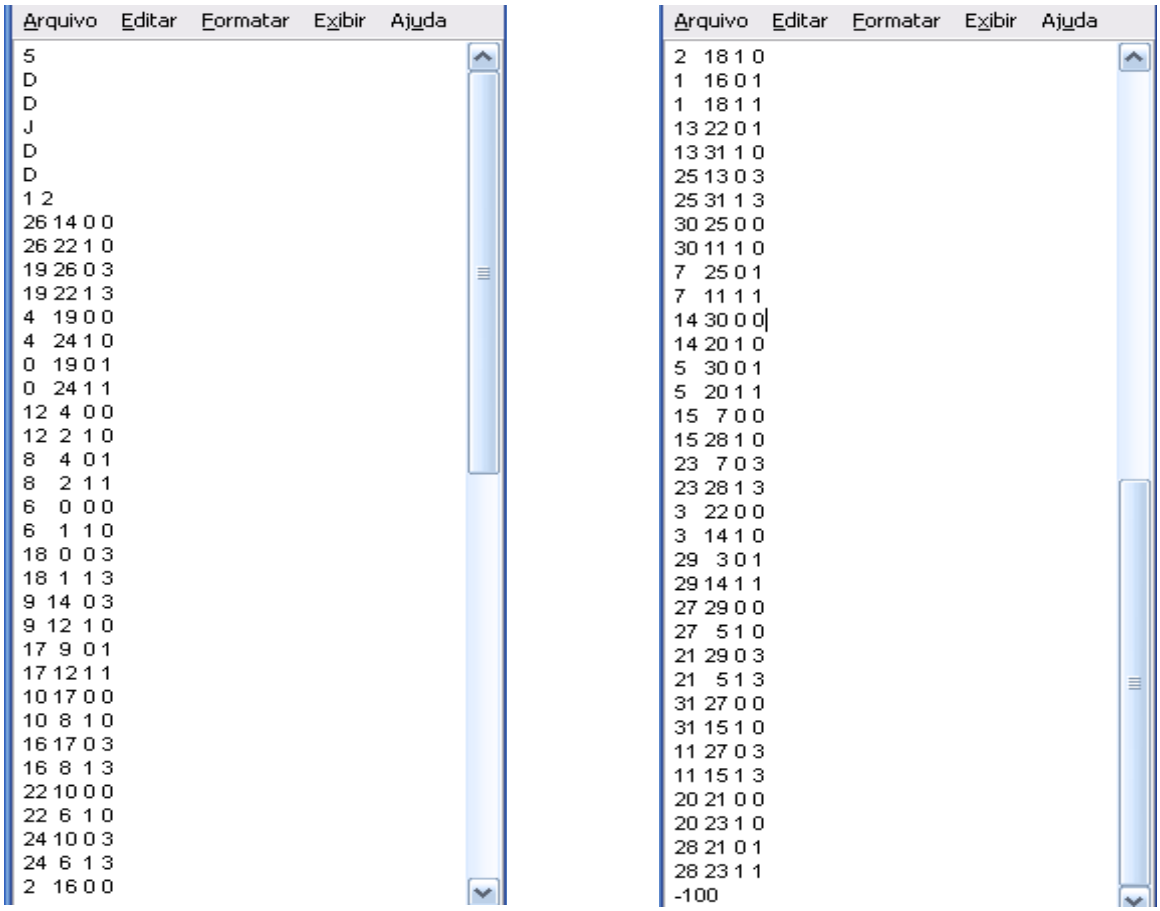


Figura 6.40: Tabela gerada pelo programa AGPS.

A tabela gerada pelo AGPS é utilizada como arquivo de entrada para o programa TABELA.

A descrição gerada pelo TABELA é utilizada pelo programa TAB2VHDL, como arquivo de entrada. Na figura 6.41 é exibida a escrição RTL na linguagem VHDL gerada pelo programa TAB2VHDL, nesta descrição são criados cinco processos para descrever os *flip-flops* D, D, Jk D e D.

Arquivo Editar Formatar Exibir Ajuda

```
-- Inferindo flip-flop tipo D
PROCESS(CLK, CLR)
BEGIN
   IF CLR = '0' THEN
     VB0 <= '0';
   ELSIF CLK'EVENT and CLK = '1' THEN
     VB0 <= D0;
   END IF;
   Q0 <= VB0;
END PROCESS;

-- Inferindo flip-flop tipo D
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;

-- Inferindo flip-flop tipo JK
PROCESS(CLK, CLR)
BEGIN
   IF CLR = '0' THEN
     VE2 <= '0';
   ELSIF CLK'EVENT and CLK = '1' THEN
     IF J2 = '0' and K2 = '1' THEN
       VE2 <= '0';
     ELSIF J2 = '1' and K2 = '0' THEN
       VE2 <= '1';
     ELSIF J2 = '1' and K2 = '1' THEN
       VE2 <= not(VE2);
     END IF;
   END IF;
   Q2 <= VE2;
END PROCESS;

-- Inferindo flip-flop tipo D
PROCESS(CLK, CLR)
BEGIN
   IF CLR = '0' THEN
     VE3 <= '0';
   ELSIF CLK'EVENT and CLK = '1' THEN
     VE3 <= D3;
   END IF;
   Q3 <= VE3;
END PROCESS;

-- Inferindo flip-flop tipo D
PROCESS(CLK, CLR)
BEGIN
   IF CLR = '0' THEN
     VE4 <= '0';
   ELSIF CLK'EVENT and CLK = '1' THEN
     VE4 <= D4;
```

Figura 6.41: Parte da descrição gerada pelo programa TAB2VHDL.

O programa PIPE2VHDL5M é utilizado para obter a descrição comportamental na linguagem VHDL do código de linha HDB3, através da sua descrição PNML obtida pela modelagem da sua RdP Lugar/Transição no ambiente PIPE, utilizando a metodologia 5M. A figura 6.42 mostra a descrição comportamental para o HDB3. Observa-se que os portos de entrada (x0) e saída (z0 e z1) e o sinal de sincronismo (*clock*) são definidos dentro da entidade (*ENTITY)*. Dentro da estrutura *Case* é definida a mudança de estados e saídas da máquina descrita.

Arquivo Editar Formatar Exibir Ajuda

```
ENTITY HDB3Mealy IS
        PORT
        (
                --xn representão as variáveis de entrada
                --zn representão as variáveis de saída
                clock:IN BIT;
                x0:IN BIT;
                z0, z1:OUT BIT
        );
END HDB3Mealy;

ARCHITECTURE Codigo OF HDB3Mealy IS

        TYPE tipo_estado IS (G0, G1, G2, G3, G4, G5, G6, G7, G8, G9, G10, G11, G12, G13
        SIGNAL Atual, Proximo: tipo_estado;
BEGIN

        PROCESS (Atual, x0)

        BEGIN
                CASE Atual IS
                        WHEN G0 =>
                                IF x0= '0' THEN
                                        z0<= '0';
                                        z1<= '0';
                                        Proximo <=G20;
                                ELSIF x0= '1' THEN
                                        z0<= '0';
                                        z1<= '0';
                                        Proximo <=G12;
                                END IF;
                        WHEN G1 =>
                                IF x0= '0' THEN
                                        z0<= '1';
                                        z1<= '1';
                                        Proximo <=G0;
                                ELSIF x0= '1' THEN
                                        z0<= '1';
                                        z1<= '1';
                                        Proximo <=G12;
                                END IF;
                        WHEN G2 =>
                                IF x0= '0' THEN
                                        z0<= '0';
                                        z1<= '0';
                                        Proximo <=G1;
                                ELSIF x0= '1' THEN
                                        z0<= '0';
                                        z1<= '0';
                                        Proximo <=G13;
                                END IF;
                        WHEN G3 =>
                                IF x0= '0' THEN
                                        z0<= '0';
                                        z1<= '1';
```

Figura 6.42: Parte da descrição comportamental gerada pelo programa TAB2VHDL.

Na figura 6.43 é ilustrado o resultado da simulação realizada no ambiente Quartus II 6.0 da Altera, para a descrição VHDL comportamental do código HDB3. Onde observa-se no instante 10 ns, o clock transitando para o nível lógico 1 e a entrada (x0) com valor lógico 0, conseqüentemente a máquina transita do estado “G0” para o estado "G20". No mesmo instante pode-se notar que a saída permaneceu em nível lógico "00". Já no instante 30 ns, nota-se que há outra mudança do clock e que a entrada possui valor lógico 1, assim a máquina transita do estado “G20” para o estado "G30". No mesmo instante observa-se que a saída permaneceu em nível lógico "00". No instante 90 ns, observa-se que a mudança do clock para

o nível lógico 1 e a entrada possuindo o valor lógico 0, faz com que a máquina transite do estado “G21” para o estado " G18". No mesmo instante observa-se que a saída está em nível lógico "01". Dessa forma pode-se constatar que a simulação está de acordo com o comportamento da descrição apresentada na figura 6.43, considerando-se o atraso de cerca de 2 ns para a mudança dos estados e das saídas.

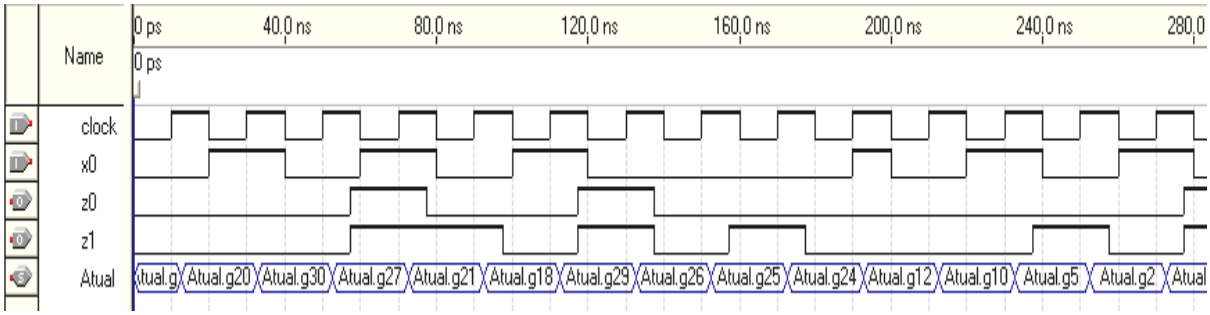


Figura 6.43: Simulação da descrição comportamental para o código de linha HDB3.

É importante salientar que a síntese de alto nível é atingida pelos ambientes computacionais desenvolvidos, onde se obtêm as descrições VHDL dos sistemas modelados em RdP. Portanto as descrições RTL e comportamental dos sistemas são obtidas a partir de sua especificação em alto nível de abstração e convertidas para um nível inferior.

6.2 Testes Aplicados à Metodologia 5M

Utilizando-se a metodologia 5M para a modelagem das FsM do tipo Moore apresentar-se-ão nesta subseção os resultados obtidos pelos programas PIPE2TAB5M, PIPE2VHDL5M, AGPS e Tabela, através da utilização da modelagem das FSM do tipo Moore. O detector da seqüência de bits 11, o código de linha AMI e a FSM com duas entradas e uma saída, compõem o espaço amostral dos testes utilizados nesta subseção.

6.2.1 Detector para a seqüência 11 com sobreposição

O detector apresentado nesta subseção possui a finalidade de identificar a seqüência 11, em uma seqüência de pulsos (1 e 0) fornecida como entrada, assim sua saída terá o nível lógico 1 toda vez que a seqüência for detectada, mesmo quando for identificada com sobreposição.

Na figura 6.44 é apresentada a FSM do tipo Moore que descreve o detector. O estado A é o estado inicial da máquina e fornece a saída de nível lógico 0. O deslocamento para os demais estados ocorre de acordo com o valor lógico da entrada. Caso a máquina esteja no estado A e receba como entrada "1", a máquina passará para o estado B que fornece a saída "0", se em seguida for identificada outra entrada "1", a máquina deslocará para o estado C que tem como saída o valor lógico 1. A mesma lógica é utilizada para todos os estados da máquina, onde cada um analisa todas as entradas possíveis.

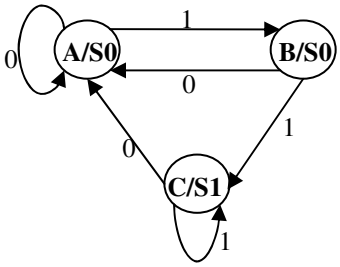


Figura 6.44: Máquina de Moore para o detector da seqüência 11 com sobreposição.

Utilizando-se o ambiente PIPE e a metodologia 5M para modelar o detector desta subseção, obtém-se a RdP Lugar/Transição apresentada na figura 6.45. Os estados da máquina são representados na rede pelos lugares A/S0, B/S0 e C/S1, que também representam suas respectivas saídas, de valores lógicos 0 (S0) e 1 (S1). As entradas de valores lógicos 0 e 1 são representadas pelos lugares E0 e E1.

A modelagem ilustrada na figura 6.45, realizada no ambiente PIPE, gera uma descrição PNML que é utilizada pelo programa PIPE2TAB5M, que transcreve as informações

da FSM especificada pelo modelo em RdP na forma de tabela. O arquivo gerado pelo PIPE2TAB5M é mostrado na figura 6.46.

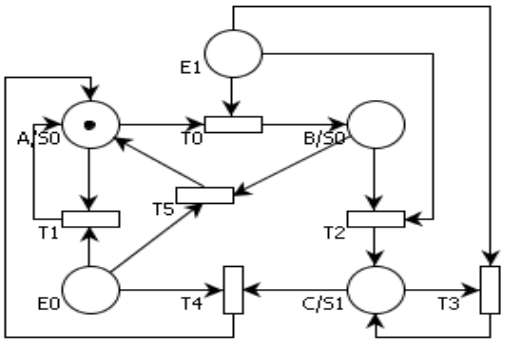


Figura 6.45: RdP Lugar/Transição para o detector da seqüência 11 com sobreposição, obtida pela metodologia 5M.

Observa-se na figura 6.46 que a primeira linha do arquivo especifica-se a quantidade mínima dos elementos de memórias, neste caso são dois elementos utilizados, esses são mencionados nas duas linhas abaixo, como do tipo JK e D respectivamente para o 1º e 2º *flip-flop*.

Na quarta linha tem-se a quantidade de entradas e saídas, ou seja, um porto de entrada e um porto de saída.

As transições dos estados e a saída gerada por cada um deles são especificadas a partir da quinta linha, conforme cada entrada.

Figura 6.46: Arquivo gerado pelo programa PIPE2TAB5M

O arquivo obtido pelo PIPE2TAB5M é utilizado como entrada pelo programa AGPS. Na figura 6.47 é ilustrada o arquivo gerado pelo AGPS. Nota-se que o AGPS alocou os valores 2, 0 e 3 para os estados A, B e C, exatamente na ordem que foram descritos.

Figura 6.47: Arquivo gerado pelo programa AGPS para o detector da seqüência 11 com sobreposição.

O programa TABELA utiliza o arquivo ilustrado na figura 6.47 como arquivo de entrada. Assim a descrição do detector, gerada pelo programa TABELA está apresentada na figura 6.48. as funções combinacionais D1, J0, K0 e Z0 possuem o custo final da sua implementação igual a 6, 2, 1 e 1. O custo total para a implementação das quatro funções é igual a 10. O custo total de todas as funções é obtido através da soma do custo final de cada função.

```
FUNCAO D1
═══════

MINTERMOS :  2;  0;  4;  3;  7;

DON'T CARE STATES :  5;  1;

IMPLICANTES PRIMOS ESSENCIAIS :

ESSENCIAL:  1     REDUNDANCIA:       6 ->  XX1
ESSENCIAL:  0     REDUNDANCIA:       5 ->  X0X
ESSENCIAL:  0     REDUNDANCIA:       3 ->  0XX

CUSTO FINAL DE D1 = 6

FUNCAO J0
═══════

MINTERMOS :  4;

DON'T CARE STATES :  3;  7;  5;  1;

IMPLICANTES PRIMOS ESSENCIAIS :

ESSENCIAL:  4     REDUNDANCIA:       1 ->  10X

CUSTO FINAL DE J0 = 2

FUNCAO K0
═══════

MINTERMOS :  3;

DON'T CARE STATES :  2;  6;  0;  4;  5;  1;

IMPLICANTES PRIMOS ESSENCIAIS :

ESSENCIAL:  0     REDUNDANCIA:       3 ->  0XX

CUSTO FINAL DE K0 = 1

FUNCAO Z0
═══════

MINTERMOS :  3;  7;

DON'T CARE STATES :  5;  1;

IMPLICANTES PRIMOS ESSENCIAIS :

ESSENCIAL:  1     REDUNDANCIA:       6 ->  XX1

CUSTO FINAL DE Z0 = 1


CUSTO TOTAL DAS 4 FUNCOES = 10
```

Figura 6.48: Descrição gerada pelo programa Tabela para o detector da seqüência 11 com sobreposição.

Com o propósito de obter a descrição comportamental na linguagem VHDL do detector da seqüência 11 com sobreposição, através da descrição PNML obtida pelo ambiente PIPE, utiliza-se o programa PIPE2VHDL5M. A figura 6.49 mostra a descrição comportamental obtida pelo PIPE2VHDL5M. Nota-se que dentro da estrutura *Case* é definida a mudança de estados e saídas da máquina descrita, de acordo com a entrada que é processada por cada estado. As variáveis Atual e x0 constituem a lista de sensibilidade do processo que descreve a estrutura *Case*.

```
PROCESS (Atual, x0)
BEGIN
        CASE Atual IS
                WHEN A =>
                        IF x0= '0' THEN
                                Proximo <=A;
                        ELSIF x0= '1' THEN
                                Proximo <=B;
                        END IF;
                        z0<= '0';
                WHEN B =>
                        IF x0= '0' THEN
                                Proximo <=A;
                        ELSIF x0= '1' THEN
                                Proximo <=C;
                        END IF;
                        z0<= '0';
                WHEN C =>
                        IF x0= '0' THEN
                                Proximo <=A;
                        ELSIF x0= '1' THEN
                                Proximo <=C;
                        END IF;
                        z0<= '1';
        END CASE;
END PROCESS;
```

Figura 6.49: Descrição comportamental para máquina de Moore gerada pelo programa PIPE2VHDL5M.

Observa-se que as descrições VHDL gerada para as máquinas de Moore são diferentes das de Mealy, devido às saídas dependerem apenas do estado atual da máquina, assim a cada estado é atribuída uma única saída, que é representada pela variável “z0” dentro da função c*ase*.

Na figura 6.50 é mostrado o resultado da simulação realizada no ambiente Quartus II 6.0 da Altera, para da descrição comportamental do detector da seqüência 11 com sobreposição na linguagem VHDL. Nota-se na figura 6.50, na transição ocorrida no instante 30 ns que a máquina transita do estado “a” para o estado "b". No mesmo instante pode-se notar que a saída permaneceu em nível lógico "0" e o *clock* assumiu nível lógico "1". Já na transição seguinte, ocorrida no instante 50 ns, nota-se que a máquina transita do estado “b” para o estado "a". No mesmo instante observa-se que a saída permaneceu em nível lógico "0"

e o *clock* assumiu nível lógico "1". No instante 90 ns, a máquina transita do estado “b” para o estado "c". No mesmo instante observa-se que a saída passa para o nível lógico "1". Dessa forma pode-se constatar que a simulação está de acordo com o comportamento da descrição apresentada na figura 6.49, considerando-se o atraso de cerca de 2 ns para a mudança dos estados e das saídas.

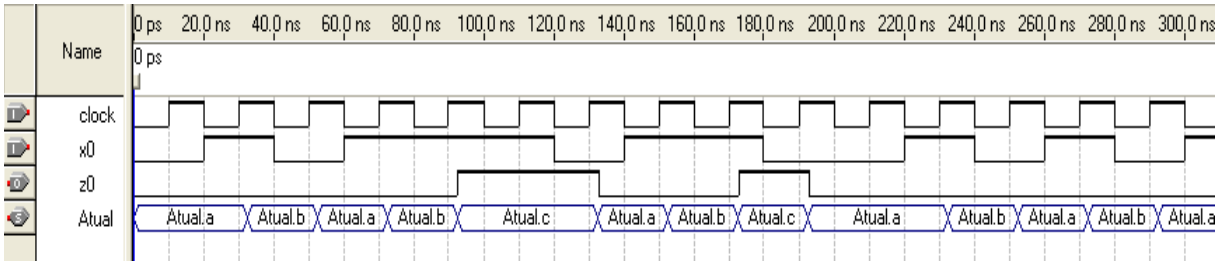


Figura 6.50: Simulação da descrição comportamental para o detector da seqüência 11 com sobreposição.

6.2.2 Código de linha AMI

O projeto do código de linha AMI apresentado nesta subseção é semelhante ao descrito na seção 6.1.2, pois possui o mesmo funcionamento, a diferença está na representação do diagrama de estados, ou seja, o tipo de máquina utilizada.

A representação ilustrada na figura 6.51 é diferente da figura 6.10 da seção 6.1.2, apenas pelo modelo da máquina utilizada no diagrama de estados, que em vez de ser do tipo Mealy é de Moore, porém a lógica do seu funcionamento é a mesma. Na figura 6.52 o AMI está representado em máquina de Moore.

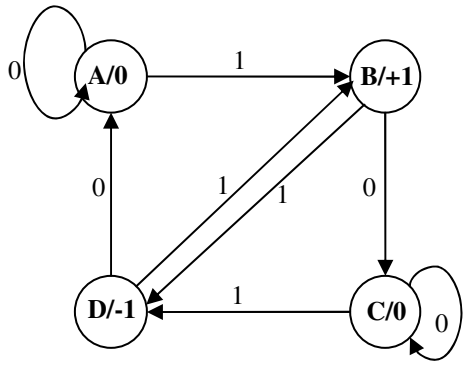


Figura 6.51: Diagrama de estados representado em máquina de Moore para o código de linha AMI.

Utilizando-se a metodologia de modelagem 5M e o ambiente PIPE para modelar o código de linha AMI, obtém-se a RdP Lugar/Transição apresentada na figura 6.52. Observa-se que cada um dos estados da máquina ilustrados na figura 6.51 são representados por lugares distintos na RdP, que recebem a denominação A/S0, B/S1, C/S0 e D/S2, para os estados A, B, C e D, esses representam ao mesmo tempo as saídas 0, +1 e -1 (0, 1 e 2).

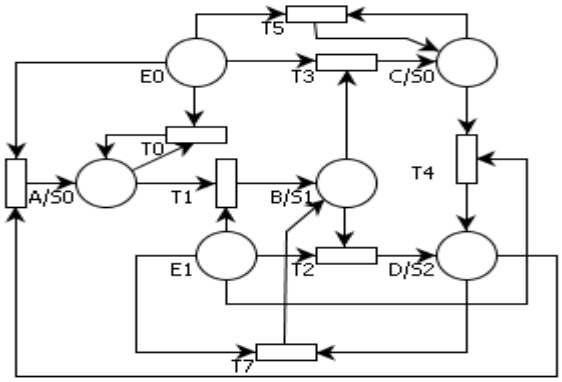


Figura 6.52: RdP Lugar/Transição para o código de linha AMI, representada em máquinas de Moore.

A descrição PNML obtida pelo modelo da RdP Lugar/Transição é utilizada pelo programa PIPE2TAB5M, para gerar a tabela de transição de estados da RdP que modela a máquina de Moore para o código de linha AMI. A figura 6.53 mostra o arquivo gerado pelo programa PIPE2TAB5M. Observa-se que os dois elementos de memórias mencionados nesta tabela são respectivamente do tipo D e JK para o 1º e 2º *flip-flops*.

Especifica-se, na quarta linha, a quantidade de entradas e saídas do AMI. Têm-se um porto de entrada e dois portos de saída.

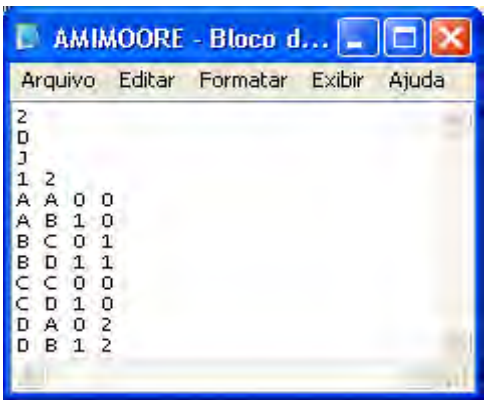


Figura 6.53: Arquivo gerado pelo programa PIPE2TAB5M para o código de linha AMI.

O programa AGPS utiliza o arquivo gerado pelo PIPE2TAB5M, como informação (arquivo) de entrada. A tabela de saída criada pelo AGPS é mostrada na figura 6.54. Observa-se nesta que os estados A, B, C e D receberam a alocação dos valores 0, 1, 3 e 2 respectivamente.

Figura 6.54: Tabela gerada pelo programa AGPS.

A tabela gerada pelo AGPS é utilizada como arquivo de entrada para o programa Tabela. Na figura 6.55 é apresentada parte da descrição dos mintermos e das funções da FSM modeladas em RdP. Para a descrição da figura 6.55 tem-se que as funções combinacionais J1, K1, D0, Z1 e Z0 possuem o custo final da sua implementação igual a 1, 1, 6, 2 e 2. O custo total para a implementação das cinco funções é igual a 12. O custo total de todas as funções é obtido através da soma do custo final de cada função.

```
FUNCAO J1
═══════

MINTERMOS :  1;  5;
DON'T CARE STATES :  3;  7;  2;  6;
IMPLICANTES PRIMOS ESSENCIAIS :
ESSENCIAL:  1     REDUNDANCIA:       6 ->  XX1
CUSTO FINAL DE J1 = 1

FUNCAO K1
═══════

MINTERMOS :  2;  6;
DON'T CARE STATES :  0;  4;  1;  5;
IMPLICANTES PRIMOS ESSENCIAIS :
ESSENCIAL:  0     REDUNDANCIA:       6 ->  XX0
CUSTO FINAL DE K1 = 1

FUNCAO D0
═══════

MINTERMOS :  4;  1;  3;  6;
```

Figura 6.55: Descrição gerada pelo programa Tabela para o código de linha AMI

O programa PIPE2VHDL5M também utiliza a descrição PNML gerada pelo PIPE, com o propósito de obter a descrição comportamental na linguagem VHDL para o código de linha AMI modelado em RdP, utilizando-se da definição de máquina de Moore. A figura 6.56 mostra a descrição comportamental obtida pelo programa. Observa-se que os portos de entrada (x0) e saída (z0 e z1) e o sinal de sincronismo (*clock*) são definidos dentro da entidade (*ENTITY*). Os estados da máquina (A, B, C e D) são representados dentro da arquitetura como um tipo de variável definida "tipo_estado". Dentro da estrutura *Case* é definida a mudança de estados e saídas da máquina descrita.

Arquivo Editar Formatar Exibir Ajuda

```
ENTITY AMIMOORE IS
        PORT
        (
                --xn representão as variáveis de entrada
                --zn representão as variáveis de saída
                clock:IN BIT;
                x0:IN BIT;
                z0, z1:OUT BIT
        );
END AMIMOORE;

ARCHITECTURE MOORE OF AMIMOORE IS

        TYPE tipo_estado IS (A, B, C, D);
        SIGNAL Atual, Proximo: tipo_estado;
BEGIN

        PROCESS (Atual, x0)

        BEGIN
                CASE Atual IS
                        WHEN A =>
                                IF x0= '0' THEN
                                        Proximo <=A;
                                ELSIF x0= '1' THEN
                                        Proximo <=B;
                                END IF;
                                z0<= '0';
                                z1<= '0';
                        WHEN B =>
                                IF x0= '0' THEN
                                        Proximo <=C;
                                ELSIF x0= '1' THEN
                                        Proximo <=D;
                                END IF;
                                z0<= '0';
                                z1<= '1';
                        WHEN C =>
                                IF x0= '0' THEN
                                        Proximo <=C;
                                ELSIF x0= '1' THEN
                                        Proximo <=D;
                                END IF;
                                z0<= '0';
                                z1<= '0';
```

Figura 6.56: Descrição comportamental para máquina de Moore gerada pelo programa PIPE2VHDL5M.

A descrição obtida pelo programa PIPE2VHDL5M foi simulada no ambiente Quatus II 6.0. O resultado da simulação é ilustrado na figura 6.57. Nota-se na transição ocorrida no instante 30 ns, a máquina transita do estado "a" para o estado "b". No mesmo instante pode-se notar que a saída permaneceu em nível lógico "00" e o *clock* assumiu nível lógico "1". Já na transição seguinte, ocorrida no instante 50 ns, a máquina transita do estado

“b” para o estado "c". No mesmo instante observa-se que a saída permaneceu em nível lógico "01" e o *clock* assumiu nível lógico "1". No instante 70 ns, nota-se que a máquina transita do estado “c” para o estado "d". No mesmo instante observa-se que a saída está em nível lógico "00". Dessa forma pode-se constatar que a simulação está de acordo com o comportamento da descrição apresentada na figura 6.56, considerando-se o atraso de cerca de 2 ns para a mudança dos estados e das saídas.

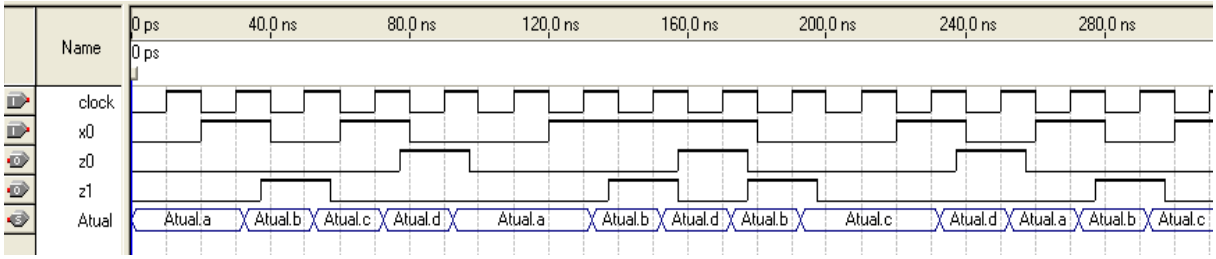


Figura 6.57: Simulação da descrição comportamental da máquina de Moore, para o código de linha AMI.

6.2.3 FSM com duas entradas e uma saída

A FSM apresentada nesta subseção é semelhante à descrita na seção 6.1.3, pois possui o mesmo funcionamento, a diferença está na representação do seu diagrama de estados, ou seja, o tipo de máquina utilizada, nesta subseção utilizou se a máquina de Moore ao invés da Mealy. Na figura 6.58 é apresentada a máquina de Moore para FSM com duas entradas e uma saída.

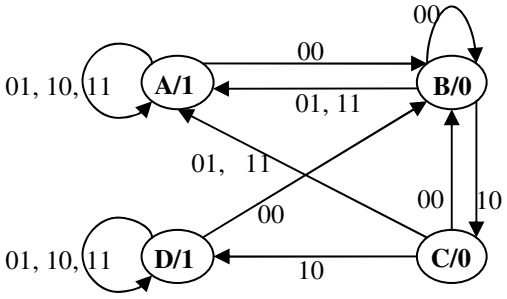


Figura 6.58: Diagrama de estados representado em máquina de Moore para FSM com duas entradas e uma saída.

Na figura 6.59 é ilustrada a modelagem em RdP Lugar/Transição para a FSM mostrada na figura 6.58. Nota-se no modelo da RdP que os lugares A/S1, B/S0, C/S0 e D/S1

correspondem aos lugares A, B, C e D da máquina de Moore, que possuem como saída os valores lógicos 1, 0, 0 e 1. As entradas da máquina são modeladas pelos lugares E0, E1, E2 e E3, que representam os valores binários 00, 01, 10 e 11.

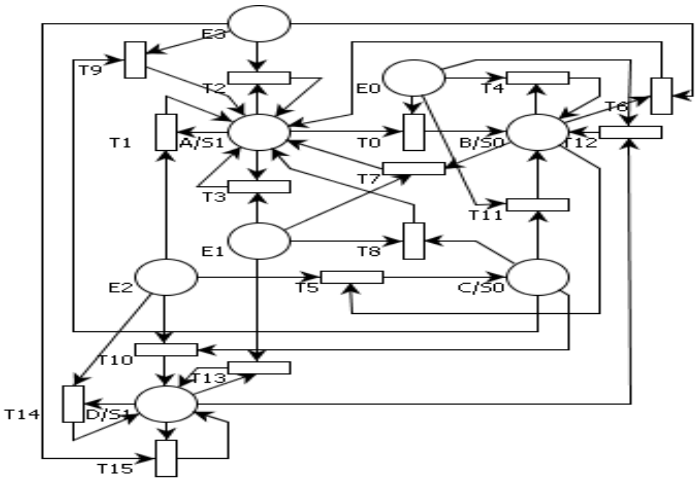


Figura 6.59: RdP Lugar/Transição para a FSM com duas entradas e uma saída, representada em máquinas de Moore.

O arquivo gerado pelo ambiente PIPE, relativo ao modelo da RdP apresentado na figura 6.59, é utilizado como entrada pelo programa PIPE2TAB5M, este por sua vez gera a tabela de transição de estados da RdP que modela a FSM com duas entradas e uma saída.

A figura 6.60 mostra o arquivo gerado pelo programa PIPE2TAB5M. Nota-se que os dois elementos de memórias mencionados na figura 6.60 são respectivamente do tipo D para o 1º e 2º *flip-flops*.

Especifica-se na quarta linha a quantidade de entradas e saídas do AMI. Têm-se dois portos de entrada e um porto de saída.

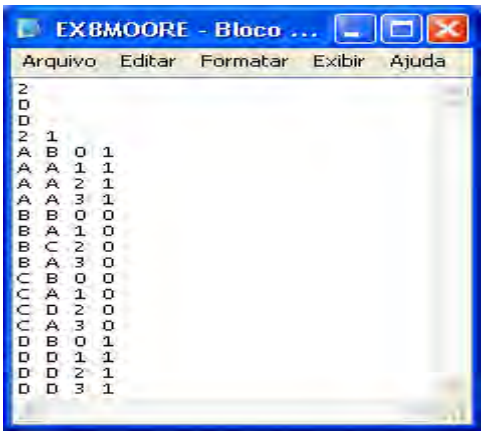


Figura 6.60: Arquivo gerado pelo programa PIPE2TAB5M para a FSM com duas entradas e uma saída.

O programa AGPS utiliza o arquivo gerado pelo PIPE2TAB5M, como arquivo de entrada. O arquivo de saída criado pelo AGPS é mostrada na figura 6.61. Observa-se neste que os estados A, B, C e D receberam as alocações 2, 3, 1 e 0, respectivamente.

Figura 6.61: Arquivo gerado pelo programa AGPS.

O arquivo gerado pelo AGPS é utilizado como arquivo de entrada pelo programa TABELA. Na figura 6.62 é apresentada a descrição obtida pelo TABELA para a FSM apresenta nesta subseção. Observa-se que as funções combinacionais D1, D0 e Z0 possuem o custo final da sua implementação igual a 9, 7 e 1. O custo total para a implementação das três funções é igual a 17. O custo total de todas as funções é obtido através da soma do custo final de cada função.

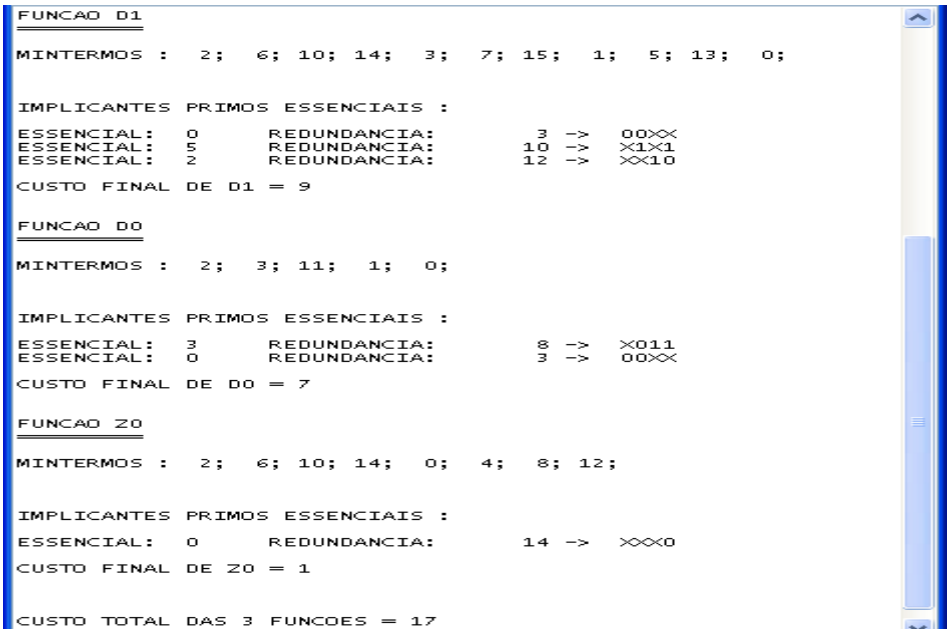
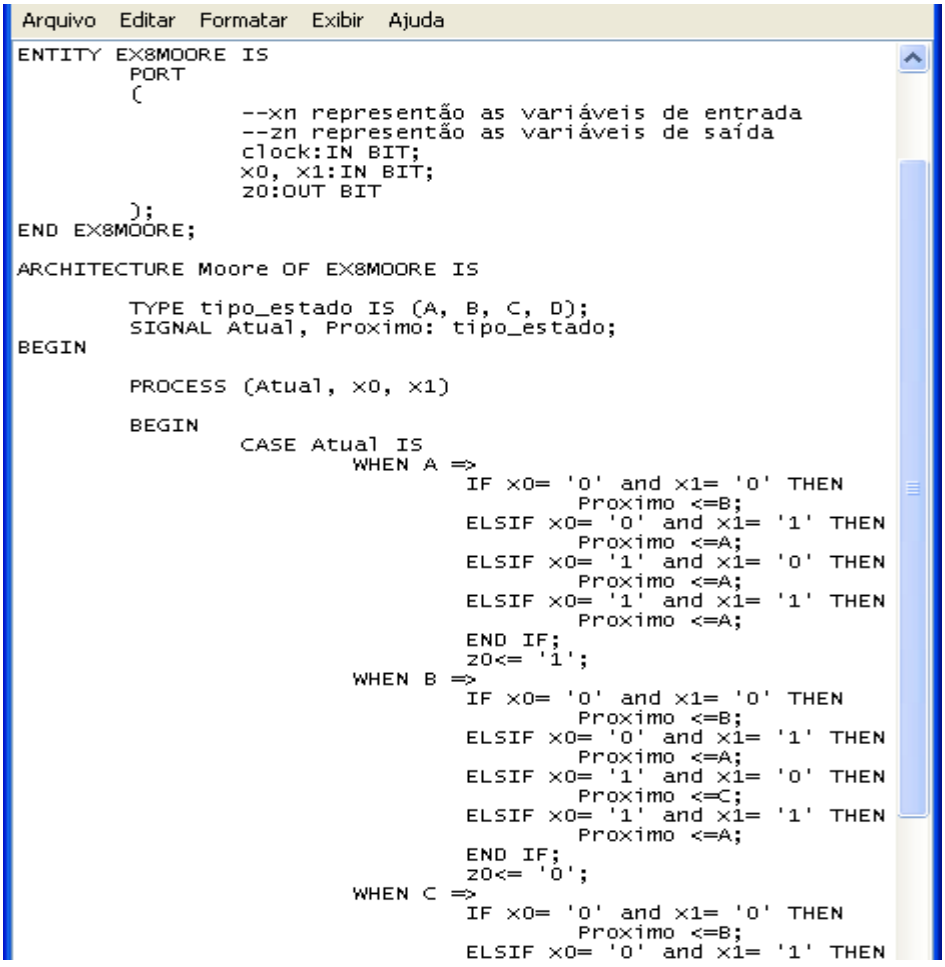


Figura 6.62: Descrição gerada pelo programa Tabela para FSM com duas entradas e uma saída

Por fim utiliza-se o programa PIPE2VHDL5M para obter a descrição comportamental na linguagem VHDL da FSM com duas entradas e uma saída modelada em RdP Lugar/Transição. A figura 6.63 exibe a descrição comportamental obtida pelo programa. Nota-se que os portos de entrada (x0 e x1), saída (z0) e o sinal de sincronismo (*clock*) são definidos dentro da entidade (*ENTITY*). Os estados da máquina (A, B, C e D) são representados dentro da arquitetura como um tipo de variável definida "tipo_estado". Dentro da estrutura *Case* é definida a mudança de estados e saídas da máquina descrita. Nota-se na descrição que para cada estado é atribuída uma única saída (z0), isto ocorre, pois às saídas nas máquinas de Moore dependem apenas do estado atual.



Arquivo Editar Formatar Exibir Ajuda

```
ENTITY EX8MOORE IS
        PORT
        (
                --xn representão as variáveis de entrada
                --zn representão as variáveis de saída
                clock:IN BIT;
                x0, x1:IN BIT;
                z0:OUT BIT
        );
END EX8MOORE;

ARCHITECTURE Moore OF EX8MOORE IS

        TYPE tipo_estado IS (A, B, C, D);
        SIGNAL Atual, Proximo: tipo_estado;
BEGIN

        PROCESS (Atual, x0, x1)

        BEGIN
                CASE Atual IS
                        WHEN A =>
                                IF x0= '0' and x1= '0' THEN
                                        Proximo <=B;
                                ELSIF x0= '0' and x1= '1' THEN
                                        Proximo <=A;
                                ELSIF x0= '1' and x1= '0' THEN
                                        Proximo <=A;
                                ELSIF x0= '1' and x1= '1' THEN
                                        Proximo <=A;
                                END IF;
                                z0<= '1';
                        WHEN B =>
                                IF x0= '0' and x1= '0' THEN
                                        Proximo <=B;
                                ELSIF x0= '0' and x1= '1' THEN
                                        Proximo <=A;
                                ELSIF x0= '1' and x1= '0' THEN
                                        Proximo <=C;
                                ELSIF x0= '1' and x1= '1' THEN
                                        Proximo <=A;
                                END IF;
                                z0<= '0';
                        WHEN C =>
                                IF x0= '0' and x1= '0' THEN
                                        Proximo <=B;
                                ELSIF x0= '0' and x1= '1' THEN
```

Figura 6.63: Descrição comportamental para máquina de Moore gerada pelo programa PIPE2VHDL5M.

O resultado da simulação realizada no ambiente Quatus II 6.0 para a descrição comportamental obtida pelo programa PIPE2VHDL5M está ilustrado na figura 6.64.

Observa-se nesta figura na transição ocorrida no instante 50 ns, a máquina transita do estado “a” para o estado "b". No mesmo instante pode-se notar que a saída permaneceu em nível lógico "1" e o *clock* assumiu nível lógico "1". Já na transição seguinte, ocorrida no instante 70 ns, nota-se que a máquina transita do estado “b” para o estado "c". No mesmo instante observa-se que a saída permaneceu em nível lógico "0" e o *clock* assumiu nível lógico "1". No instante 90 ns, nota-se que a máquina transita do estado “c” para o estado "d". No mesmo instante observa-se que a saída está em nível lógico "0". Dessa forma pode-se constatar que a simulação está de acordo com o comportamento da descrição apresentada na figura 6.64, considerando-se o atraso de cerca de 2 ns para a mudança dos estados e das saídas.

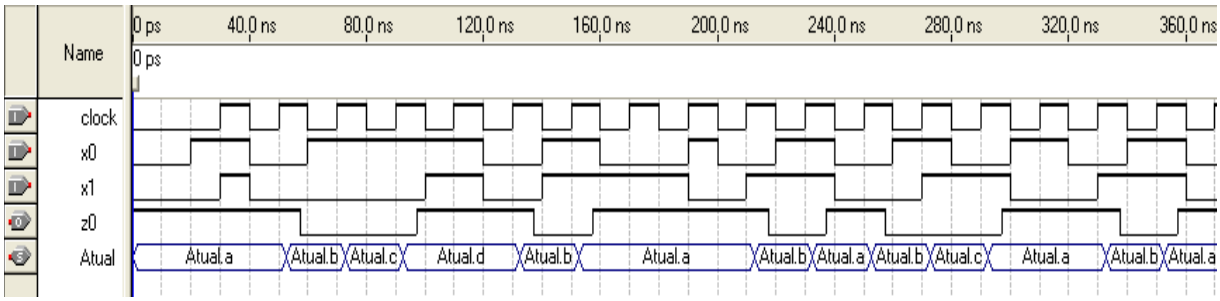


Figura 6.64: Simulação da descrição comportamental da máquina de Moore, para FSM com duas entradas e uma saída.

CONCLUSÕES

Neste trabalho apresentou-se duas metodologias, denominadas 4M e 5M, utilizadas para modelar máquina de estados finitos em RdP Lugar/Transição e quatro ferramentas de síntese denominadas PIPE2TAB4M, PIPE2VHDL4M, PIPE2TAB5M e PIPE2VHDL5M, que convertem o modelo da rede de Petri em uma descrição VHDL correspondente a máquina modelada.

As metodologias desenvolvidas auxiliam o projeto de sistemas digitais, possibilitando a modelagem de FSM do tipo Mealy ou Moore em alto nível de abstração. Este tipo de abstração facilita a visualização e o entendimento do funcionamento do sistema. Sendo assim a simulação da RdP contribui na visualização do funcionamento dos sistemas modelados, permitindo a identificação de um possível erro de projeto.

Aplicando se as metodologias 4M e 5M para a modelagem da mesma FSM em RdP, observou-se que a quantidade de elementos gráficos utilizados pela metodologia de modelagem 5M é inferior em relação ao modelo da RdP obtido pela metodologia 4M, chegando a ter uma redução de cerca de 20% dos elementos gráficos.

As ferramentas PIPE2TAB4M e PIPE2VHDL4M foram desenvolvidas para analisar os modelos da RdP obtidos pela metodologia de modelagem 4M, gerando a tabela de transição de estados e o código VHDL da FSM modela em RdP, enquanto que as PIPE2TAB5M e PIPE2VHDL5M consideram os modelos da RdP obtidos pela metodologia 5M.

Procurou-se modelar a mesma FSM utilizando-se as duas metodologias com o propósito de comparar as saídas resultantes de cada uma. Observou-se que as tabelas geradas pelas ferramentas PIPE2TAB4M e PIPE2TAB5M eram iguais para a mesma FSM modelada em RdP por metodologias distintas, os códigos VHDL gerados pelas PIPE2VHDL4M e

PIPE2VHDL5M também eram semelhantes para as metodologias distintas, que modelava a mesma FSM.

As metodologias estabelecidas e as ferramentas desenvolvidas foram validadas aplicando-se os testes do espaço amostral, assim verificou-se seus resultados práticos, simulando-os nos ambientes Quartus II 6.0 e MaxPlus II 10.2, ambos da Altera.

Procurou-se simular os projetos nos dois ambientes, para analisar as diferenças que poderiam ocorrer em cada simulação. Notou-se que o atraso na mudança das variáveis de estados e saídas eram maior na simulação no Quartus II do que no Max+Plus II e que o ambiente Quartus II não suportava a simulação do código VHDL gerado pelas ferramentas para FSM com várias entradas e saídas, sendo que a compilação ocorria sem nenhum problema. O atraso observado durante a simulação no Quartus II era superior cerca de 0,4 ns em relação ao do Max+Plus II, sendo assim este atraso não é significativo para desconsiderar a validade dos resultados obtidos.

Em trabalhos futuros podem-se desenvolver metodologias e ferramentas que trabalhem com o projeto de máquinas assíncronas. A integração das ferramentas em um único ambiente de projeto e o uso de outras extensões de RdP nas metodologias de modelagens desenvolvidas também pode ser considerado.

REFERÊNCIAS

[1] CHANG, N.; KWON, W. H.; PARK, J. FPGA – based implementation of synchronous Petri nets. **Industrial Electronics, Control, and Instrumentation, 1996. Proceedings of the 1996 IEEE IECON 22nd International Conference on,** Raleigh, v. 1, p.934-939, Aug. 1996. Disponível em: <http://ieeexplore.ieee.org/servlet/opac?punumber=4195 >. Acessado em: 29 Jan. 2006.

[2] FERNANDES, J. M; PROENÇA, A. J. Redes de Petri na especificação e validação de controladores paralelos. In: CONFERÊNCIA NACIONAL DO COLÉGIO DE ENGENHARIA ELETROTÉCNICA, 1, 1994, Lisboa. **Encontro...** Lisboa: Universidade do Minho, 1994. p. 113-118.

[3] OLIVERIA, W.L.A.; MARRANGHELLO, N.; DAMIANI, F. Modeling a processor with a Petri net extension for digital systems, In: DESIGN, ANALYSIS AND SIMULATION OF DISTRIBUTED SYSTEMS, 2004, Washington. **Proceedings...**. San Diego: Simulation Councils, 2004. p. 208-215.

[4] MIRKOVSKI, J.; YAKOVLEV, A. A Petri net model for embedded systems. In: WORKSHOP ON DESIGN AND DIAGNOSTICS OF ELECTRONIC CIRCUITS AND SYSTEMS, 2, 1998, Poland. **Proceedings...** Poland: Szczyrk, 1998. p. 313-321.

[5] MACIEL, P. R. M.; BARROS, E. N. S.; ROSENSTIEL, W. A Petri net model for hardware/software codesign, **Design Automation for Embedded Systems Journal**, Boston, v. 4, n.4, p. 243-310, 1999.

[6] OLIVEIRA FILHO, J. A.; LIMA, M. E.; MACIEL, P. R. Petri Net Based Interface Analysis for Fast IP-Core Integration, In: FIRST ACM & IEEE INTERNATIONAL CONFERENCE ON FORMAL METHODS AND MODELS FOR CODESIGN, 1, 2003, Mont Saint-Michel. **Formal Methods and Models for Codesign.** Mont Saint-Michel: IEEE Press, Jun. 2003. p.34 – 42.

[7] MARRANGHELLO, N.; MIRKOWSKI, J. Experiments on using colored Petri nets as an internal model for hardware/software codesign of embedded systems. In: INTERNATIONAL CONFERENCE ON MICROELECTRONICS AND PACKAGING, 1998, Curitiba. **Proceedings...**. 1998. v. 1, p. 63-70.

[8] YAKOVLEV, A.; GOMES, L.; LAVAGNO, L. **Hardware design and Petri nets.** New York**:** Kluwer Academic Publishers, 2000, p. 129-150.

[9] WEBER, M.; KINDLER, E. (Ed). **The Petri Net Markup Language**. Berlin: Humboldt – Universitat zu Belin, 2006. p. 1-21 Disponível em: <http://xml.coverpages.org/WeberPNML200204.pdf>. Acesso em: 14 set. 2006.

[10] Gajski, D.; Kuhn, R. **New VLSI tools**. **IEEE Computer Magazine,** New Work, v.16, n. 12, Dec. 1983. p.11-14.

[11] HADY, F.T.; AYLOR, J.H.; WILLIAMS, R.D.; WAXMAN, R. Uninterpreted modeling using the VHSIC hardware description language (VHDL)**,** In: CONFERENCE ON COMPUTER-AIDED DESIGN , 1989, Santa Clara. **Digest of technical papers ot the ieee international**... Los Alamitos: IEEE Comput. Soc. Press, 1989. p. 172-175.

[12] FERNANDES, J. M.; ADAMSKI, M.; PROENÇA A. J. VHDL generation from hierarchical Petri net specifications of parallel controllers. **IEE Proc. Comp. Digit. Tech**, Minho, v. 144, n. 2, p. 127-137. Mar. 1997.

[13] GREINER, A.; PÊCHEUX, F. Alliance: a complete set of CAD tools for teaching digital VLSI Design. In: EUROCHIP, 1992. **Workshop on VLSI Design Training.** Toledo: [s.n.], 1992. p.230-237.

[14] MACHADO R. J.; FERNADES J. M.; PROENÇA A. J. Redes de Petri e VHDL na prototipagem rápida de sistemas digitais. **Anais da engenharia e tecnologia eletrotécnica, ordem dos engenheiros**, Minho, v.2, n.4, p.1-4, Jul. 1997.

[15] SHANG, D.; BURNS, F.; KOELMANS, A.; YAKOVLEV, A.; XIA, F. **Asynchronous system synthesis based on direct mapping using VHDL and Petri nets. Computers and Digital Techniques, IEE Proceedings**, v. 151, p. 209 – 220, May 2004.

[16] ABELLARD, A.; KHELIFA, M. M. B.; BOUCHOUICHA, M. A Petri net modelling of an adaptive learning control applied to an electric wheelchair. In: COMPUTATIONAL INTELLIGENCE IN ROBOTICS AND AUTOMATION, 2005, Raleigh. **Proceedings...** Espoo: IEEE, 2005. p. 397 – 402.

[17] FERNADES, J. M. L. **Redes de Petri e VHDL na especificação de controladores paralelos.** 1994. 113f. Dissertação (Mestrado) - Departamento de Informática, Universidade do Minho, Minho, 1994.

[18] CARDOSO, J.; VALETTE R. **Redes de Petri**. Florianópolis: Universidade Federal de São Carlos, 1997. p. 36-39.

[19] RIASCOS, L. A. M. **Metodologia para detecção e tratamento de falhas em sistemas de manufatura através de redes de Petri.** 2002. 164f. Tese (Doutorado) - Escola Politécnica, Universidade de São Paulo, São Paulo, 2002.

[20] GOMES, L. **Redes de Petri e sistemas digitais:** uma introdução. Portugal: Faculdade de Ciência e Tecnologia, Departamento de Engenharia Eletrotécnica, Universidade Nova de Lisboa, 1999. p.48.

[21] PETERSON, J. Petri nets. **Computing surveys**. New York, v. 9, n. 3, p. 223-252, Set.1977.

[22] MURATA, T. Petri nets: properties, analysis and applications. **Proceedings of the IEEE**, v.77, n.4, p. 541-580, Abr. 1989

[23] JÚNIOR, V. C. V. **Definição de um modelo de sistema de workflow baseado em rede de Petri em plataforma web**. 2004. 71f. Monografia (Bacharel em Ciência da Computação) - Centro de Ciências e Tecnologia, Universidade Estadual do Ceará, Fortaleza, 2004.

[24] DAVID, R.; ALLA, H. Petri nets for modeling of dynamic system. **A Survey Automatic,** v. 30, n. 2, p. 175-202, 1994.

[25] MARRANGHELLO, N. **Apostila redes de Petri:** conceitos e aplicações. São José do Rio Preto: UNESP/IBILCE, 2005. p. 33. Disponível em: <ww.dcce.ibilce.unesp.br/~norian/cursos/mds/ApostilaRdP-CA.pdf> Acesso em: 11 Nov. 2005.

[26] JENSEN, K.; ROZENBERG, G. **High-level Peri nets:** theory and application. Berlin: Springer-Verlag, 1991. 724 p.

[27] THIAGARAJAN, P. S.; ROZENBERG, G.; FERNÁNDEZ, C. Elementary net systems - fundamentals**.** In BRAUER W.; REISIG W.; ROZENBERG, G. (Eds.) **Lecture Notes in Computer Science.** Berlin: Springer-Verlag, 1987. v. 254, p. 25-115.

[28] ROZENBERG, G., ENGELFRIET, J. Elementary Net Systems. In: REISIG, W.; ROZENBERG, G. (Eds.) **Lecture Notes in Computer Science.** Berlin: Springer-Verlag, 1998. v.1491, p.12-12.

[29] RAMCHANDANI C. **Analysis of asynchronous concurrent systems by timed Petri nets**. 1973. 225 f. Tese (Doutorado em Filosofia). Massachusetts Institute of Technology, Cambridge, 1973.

[30] PIPE. **Web site PIPE**. Disponível em: **<**http://pipe2.sourceforge.net/docs.html> Acesso em: 20 out. 2005.

[31] MORAES, F.; CALAZANS, N. **VHDL**. Lisboa: Instituto Superior Técnico – Universidade Técnica de Lisboa, 2001. 189 p. Disponível em: <http://medusa.inesc.pt/~las/pde01/vhdl_printer.pdf> Acesso em: 25 ago. 2006.

[32] MOURELLE, L. **Controle de processos com computador**. Rio de Janeiro: Faculdade de Engenharia – Universidade do Estado do Rio de Janeiro, 2006. 28 p. Disponível em: <http://www.eng.uerj.br/~ldmm/controle%20de%20processos/introducao%20a%20vhdl.ppt>. Acesso em: 9 set. 2006.

[33] GIACOMINI, R. **Apostila básica de VHDL**. São Paulo: Centro Universitário da FEI, 2006, 16 p. Disponível em: <http://www.fei.edu.br/eletrica/disciplinas_diurno/Sist_Dig_3_D/Apostila_VHDL.pdf> Acesso em: 10 set. 2006.

[34] HÜSEMANN, R. **Apostila de VHDL.** Porto Alegre: Departamento de Energia Elétrica da Universidade Federal do Rio Grande do Sul. 2001. 49 p. Disponível em

<http://www.ene.unb.br/~juliana/cursos/sistdigital1/vhdl_ronhuse.pdf> Acesso em: 10 set. 2006.

[35] ALTERA, C. **Max+plus II - Programmable Logic Development System.** San Jose: ALTERA, 1996. 112 p.

[36] PERRY, D. L. **VHDL.** New York: McGraw-Hill, 1993. 337 p.

[37] SCAICO, A. **Projeto de sistemas digitais na atualidade**. Campina Grande: Ed. UFCG, 2000, p. 61. Disponível em: <http://www.dee.ufcg.edu.br/~elmar/scaico.pdf>. Acesso em: 5 jan. 2006

[38] SANTOS, C. E. S. **Algoritmo genético emprego na alocação de estados em máquinas Markovianas.** 2005. 102f. Dissertação (Mestrado em Automação) - Faculdade de Engenharia de Ilha Solteira, Universidade Estadual Paulista, Ilha Solteira, 2005.

[39] SILVA, A. C. R. **Contribuição à minimização e simulação de circuitos lógicos.** 1989. 138f. Dissertação (Mestrado em Engenharia Elétrica) - Faculdade de Engenharia Elétrica, Universidade Estadual de Campinas, Campinas, 1989.

[40] TANCREDO, L. O. **TAB2VHDL:** um ambiente de síntese lógica para máquinas de estados finitos. 2002. 122f. Dissertação (Mestrado em Automação) - Faculdade de Engenharia de Ilha Solteira, Universidade Estadual Paulista, Ilha Solteira, 2002.

[41] MELO, J. J. L; SOBREIRA P. L. **Petri Net**. Pernambuco: Ed. UFPE, 1999. Disponível em: <www.jameson.hpg.ig.com.br/>. Acesso em: 25 out. 2006.

[42] ALTERA. **ALTERA web site.** Disponível em: <https://www.altera.com>. Acesso em: 15 nov. 2006.

APÊNDICE

Código Fonte do Programa PIPE2TAB4M

```
# UNESP Ilha Solteira
#Programa: PIPE2TAB4M
#Objetivo: Este programa tem por objetivo transformar a descrição PNML da RdP que modela uma FSM para uma tabela de transição de estados.
#Programadora: Giorjety Licorini Dias.
# -*- coding: cp1252 -*-
x=raw_input ("Digite o endereço do arquivo de entrada \n--por ex. 'Diretório:\Pasta\Nome do arquivo.extenção'\n")
arquivo= open (x,'r')
y=raw_input ("Digite o endereço do arquivo de saída \n--por ex. 'Diretório:\Pasta\Nome do arquivo. extenção'\n")
arquivocopia= open (y,'w+')
arquivo.seek(0)
b1=arquivo.read()
b=b1.splitlines()
c1=4 # Variável utilizada no while
QEs=0 # Variável responsável por armazenar a quantidade de estados da RdP
QE=0 # Variável responsável por armazenar a quantidade de entradas da RdP
QS=0  # Variável responsável por armazenar a quantidade de saídas da RdP
saida=0
while "Fim" not in b[c1-1]:
    if "nome do lugar = E" in b[c1]:
        p1=b[c1].rfind ('E')
        p2=b[c1].rfind ('')
        parametro=b[c1][p1+1:p2]
        if parametro <> '':
            QE=QE+1
            QEs=QEs-1
    elif "nome do lugar = S" in b[c1]:
        p1=b[c1].rfind ('S')
        p2=b[c1].rfind ('')
        parametro=b[c1][p1+1:p2]
        parametro=int(parametro)
        if parametro > saida:
            saida=parametro
        if parametro <> '':
            QEs=QEs-1
    QEs=QEs+1
    c1=c1+3
entrada=QE
QS=saida+1
saida=QS
import math # importando o módulo matemático que provê a maioria das funções matemáticas mais familiares
QE = math.log(QE)/math.log(2.0)
if QE > int (QE):
    QE= int (QE)+1
else:
    QE= int (QE)
QS = math.log(QS)/math.log(2.0)
if QS > int (QS):
    QS= int (QS)+1
else:
    QS= int (QS)
QFF= math.log(QEs)/math.log(2.0)# Cálculo para a quantidade de Flip Flops
if QFF > int (QFF):
    QFF= int (QFF)+1
else:
    QFF= int (QFF)
linha=QEs*entrada
coluna=3
QEs = str (QEs)
QFF = str (QFF)
```

```
QE = str (QE)
QS = str (QS)
arquivocopia.write (QFF)
QFF = int (QFF)
i=1
while i <= QFF:
    y=str(i)
    x=raw_input ("Digite o tipo do flip flop " + y + "\n")
    arquivocopia.write ("\n" + x)
    i=i+1
arquivocopia.write ("\n" + QE + ' ' + QS)
c2=c1+3
while "Arco" not in b [c2]:
    c2=c2+2
c1=4 # Variável utilizada no while
matriz=[[]]
while "Fim" not in b[c1-1]:
    p1=b[c1].rfind ('=')
    p2=b[c1].rfind ('')
    if "E" not in b[c1] or ("0" not in b[c1] and "1" not in b[c1] and "2" not in b[c1] and "3" not in b[c1] and "4" not in
b[c1] and "5" not in b[c1] and "6" not in b[c1] and "7" not in b[c1] and "8" not in b[c1] and "9" not in b[c1]):
        Atual=b[c1][p1+2:p2]
        c1=c1-1
        p1=b[c1].rfind ('P')
        p2=b[c1].rfind ('')
        Atual1=b[c1][p1:p2-1]
        c1=c1+1
        c3=c2
        while "Arco de T" not in b[c3]:
            p1=b[c3].find ('P')
            p2=b[c3].rfind ('p')
            lugar=b[c3][p1:p2-1]
            if lugar == Atual1:
                p1=b[c3].rfind ('T')
                p2=b[c3].rfind ('')
                transicao=b[c3][p1:p2-1]
                c4=c2
                while "Arco de T" not in b[c4]:
                    c4=c4+5
                c5=c4
                while "Fim da RdP" not in b[c5+1]:
                    p1=b[c5].find ('T')
                    p2=b[c5].find ('p')
                    transicao1=b[c5][p1:p2-1]
                    if transicao1 == transicao:
                        p1=b[c5].rfind ('P')
                        p2=b[c5].rfind ('')
                        lugar1=b[c5][p1:p2-1]
                        c6=3
                        while "Fim" not in b[c6]:
                            p1=b[c6].rfind ('P')
                            p2=b[c6].rfind ('')
                            lugar2=b[c6][p1:p2-1]
                            if lugar2 == lugar1 and ("S" not in b[c6+1] or ("0" not in b[c6+1] and "1" not in b[c6+1]and "2" not
in b[c6+1]and "3" not in b[c6+1]and "4" not in b[c6+1]and "5" not in b[c6+1]and "6" not in b[c6+1]and "7" not in
b[c6+1]and "8" not in b[c6+1]and "9" not in b[c6+1])):
                                c6=c6+1
                                p1=b[c6].rfind ('=')
                                p2=b[c6].rfind ('')
                                Proximo=b[c6][p1+2:p2]
                                c6=c6-1
                            if lugar2 == lugar1 and "S" in b[c6+1]:
                                c6=c6+1
                                p1=b[c6].rfind ('=')
                                p2=b[c6].rfind ('')
                                Saida=b[c6][p1+3:p2]
                                c6=c6-1
```

```
                    c6=c6+3
                c5=c5+5
            c5=c2
            n=0
            while "Arco de T" not in b[c5]:
                p1=b[c5].rfind ('T')
                p2=b[c5].rfind ('')
                transicao1=b[c5][p1:p2-1]
                p1=b[c5].find ('P')
                p2=b[c5].rfind ('p')
                lugar1=b[c5][p1:p2-1]
                if transicao1 == transicao and lugar1<>Atual1:
                    c6=3
                    while "Fim" not in b[c6] and n==0:
                        p1=b[c6].rfind ('P')
                        p2=b[c6].rfind ('')
                        lugar2=b[c6][p1:p2-1]
                        if lugar2 == lugar1:
                            c6=c6+1
                            p1=b[c6].rfind ('=')
                            p2=b[c6].rfind ('')
                            Entrada=b[c6][p1+3:p2]
                            c6=c6-1
                        c6=c6+3
                c5=c5+5
            matriz=matriz+[[Atual,Proximo,Entrada,Saida]]
        c3=c3+5
    c1=c1+3
l=1
l1=1+entrada
i=1+entrada
while l<(linha-entrada):
    while l1<=(linha-(entrada-1)):
        if matriz[l][0]>matriz[l1][0]:
            l2=0
            while l2< entrada:
                c=0
                while c<= coluna:
                    elemento=matriz[l+l2][c]
                    matriz[l+l2][c]=matriz[l1+l2][c]
                    matriz[l1+l2][c]=elemento
                    c=c+1
                l2=l2+1
        l1=l1+entrada
    l=l+entrada
    l1=i+entrada
    i=i+entrada
l=1
linha1=entrada
while l<=linha:
    l1=1
    while l+l1<=linha1:
        if matriz[l][2]>matriz[l1+l][2]:
            i=0
            while i<=coluna:
                elemento=matriz[l][i]
                matriz[l][i]=matriz[l1+l][i]
                matriz[l1+l][i]=elemento
                i=i+1
        l1=l1+1
    l=l+1
    if l==linha1:
        linha1=linha1+entrada
        l=l+1
l=1
if ('0'or '1' or '2'or '3' or '4'or '5' or '6'or '7' or '8'or '9') in matriz[1][0]:
    while l<=linha:
```

```
        p1=matriz[l][0].find ('')
        p2=matriz[l][0].rfind ('')
        elemento=matriz[l][0][p1+1:p2]
        caractere=matriz[l][0][p1:p2-1]
        matriz[l][0]=int (elemento)
        l=l+1
    l=1
    l1=1+entrada
    i=1+entrada
    while l<=(linha-entrada):
        while l1<= (linha-(entrada-1)):
            if matriz[l][0]>matriz[l1][0]:
                l2=0
                while l2< entrada:
                    c=0
                    while c<= coluna:
                        elemento=matriz[l+l2][c]
                        matriz[l+l2][c]=matriz[l1+l2][c]
                        matriz[l1+l2][c]=elemento
                        c=c+1
                    l2=l2+1
            l1=l1+entrada
        l=l+entrada
        l1=i+entrada
        i=i+entrada
    l=1
    while l<=linha:
        matriz [l][0]=str (matriz[l][0])
        matriz[l][0]=caractere+matriz[l][0]
        l=l+1
l=1
while l<=linha:
    arquivocopia.write ("\n")
    l1=0
    while l1<=coluna:
        if l1==0:
            arquivocopia.write (matriz[l][l1])
        else:
            arquivocopia.write (" " + matriz[l][l1])
        l1=l1+1
    l=l+1
arquivo.close ()
arquivocopia.close()
```

Código Fonte do Programa PIPE2VHDL4M

```
# UNESP Ilha Solteira
#Programa: PIPE2VHDL4M
#Objetivo: Este programa tem por objetivo transformar a descrição de uma máquina de estados finitos em Rede de Petri para o seu modelo comportamental descrito em VHDL.
#Programadora: Giorjety Licorini Dias.
### -*- coding: cp1252 -*-
x=raw_input ("Digite o endereço do arquivo de entrada \n--por ex. 'Diretório:\Pasta\Nome do arquivo.extenção'\n")
arquivo= open (x,'r')
y=raw_input ("Digite o endereço do arquivo de saída \n--por ex. 'Diretório:\Pasta\Nome do arquivo. extenção'\n")
arquivocopia= open (y,'w')
b4=y.rfind ('\\')
b5=y.rfind ('.')
arquivo.seek(0)
entidade= y [b4+1:b5]
```

```
arquivocopia.write ("-- PROJETO DE TESE DE MESTRADO - Ferramenta para a Integração de Redes de Petri e
VHDL na Síntese de Sistemas Digitais")
arquivocopia.write ("\n-- A ferramenta tem por objetivo transformar a descrição de uma máquina de estados
finitos em Rede de Petri para o seu modelo comportamental descrito em VHDL")
arquivocopia.write ("\n\n--Programa: PIPE2VHDL4M ")
arquivocopia.write ("\n--Programadora: Giorjety Licorini Dias")
arquivocopia.write ("\n--Versão: 1.2 de 20 de outubro de 2006")
arquivocopia.write ("\n\nENTITY "+ entidade +" IS\n")
arquivo.seek(0)
b1=arquivo.read()
b=b1.splitlines()
c1=4 # Variável utilizada no while
QEs=0 # Variável responsável por armazenar a quantidade de estados da RdP
QE=0 # Variável responsável por armazenar a quantidade de entradas da RdP
QS=0  # Variável responsável por armazenar a quantidade de saídas da RdP
saida=0
while "Fim" not in b[c1-1]:# Contabiliza a quantidade de Entradas e Estados da FSM
   if "nome do lugar = E" in b[c1]:
      p1=b[c1].rfind ('E')
      p2=b[c1].rfind (")
      parametro=b[c1][p1+1:p2]
      if parametro <> ":
         QE=QE+1
         QEs=QEs-1
   elif "nome do lugar = S" in b[c1]:
      p1=b[c1].rfind ('S')
      p2=b[c1].rfind (")
      parametro=b[c1][p1+1:p2]
      parametro=int(parametro)
      if parametro > saida:
         saida=parametro
      if parametro <> ":
         QEs=QEs-1
   QEs=QEs+1
   c1=c1+3
entrada=QE
QS=saida+1
saida=QS
import math # importando o módulo matemático que provê a maioria das funções matemáticas mais familiares
QE = math.log(QE)/math.log(2.0)
if QE > int (QE):
   QE= int (QE)+1
else:
   QE= int (QE)
QS = math.log(QS)/math.log(2.0)
if QS > int (QS):
   QS= int (QS)+1
else:
   QS= int (QS)
arquivocopia.write ("\tPORT\n\t(\n\t\t--xn representão as variáveis de entrada")
arquivocopia.write ("\n\t\t--zn representão as variáveis de saída")
arquivocopia.write ("\n\t\tclock:IN BIT;")
arquivocopia.write ("\n\t\tx0")
e=1
while e<QE:
   arquivocopia.write (", x"+ str(e))
   e=e+1
arquivocopia.write (":IN BIT;")
arquivocopia.write ("\n\t\tz0")
s=1
while s<QS:
   arquivocopia.write (", z"+ str(s))
   s=s+1
arquivocopia.write (":OUT BIT \n\t);")
arquivocopia.write ("\nEND "+ entidade +";\n")
arq=raw_input ("Digite o nome da Arquitetura que vc deseja para o código VHDL\n")
arquivocopia.write ("\nARCHITECTURE "+ arq +" OF " + entidade +" IS\n")
```

```
arquivocopia.write ("\n\tTYPE tipo_estado IS (")
linha=QEs*entrada
coluna=3

c2=c1+3
while "Arco" not in b [c2]:
   c2=c2+2
matriz=[[]]
c1=4 # Variável utilizada no while
while "Fim" not in b[c1-1]:
   p1=b[c1].rfind ('=')
   p2=b[c1].rfind (")
   if "E" not in b[c1] or ("0" not in b[c1] and "1" not in b[c1] and "2" not in b[c1] and "3" not in b[c1] and "4" not in
b[c1] and "5" not in b[c1] and "6" not in b[c1] and "7" not in b[c1] and "8" not in b[c1] and "9" not in b[c1]):
      Atual=b[c1][p1+2:p2]
      c1=c1-1
      p1=b[c1].rfind ('P')
      p2=b[c1].rfind (")
      Atual1=b[c1][p1:p2-1]
      c1=c1+1
      c3=c2
      while "Arco de T" not in b[c3]:
         p1=b[c3].find ('P')
         p2=b[c3].rfind ('p')
         lugar=b[c3][p1:p2-1]
         if lugar == Atual1:
            p1=b[c3].rfind ('T')
            p2=b[c3].rfind (")
            transicao=b[c3][p1:p2-1]
            c4=c2
            while "Arco de T" not in b[c4]:
               c4=c4+5
            c5=c4
            while "Fim da RdP" not in b[c5+1]:
               p1=b[c5].find ('T')
               p2=b[c5].find ('p')
               transicao1=b[c5][p1:p2-1]
               if transicao1 == transicao:
                  p1=b[c5].rfind ('P')
                  p2=b[c5].rfind (")
                  lugar1=b[c5][p1:p2-1]
                  c6=3
                  while "Fim" not in b[c6]:
                     p1=b[c6].rfind ('P')
                     p2=b[c6].rfind (")
                     lugar2=b[c6][p1:p2-1]
                     if lugar2 == lugar1 and ("S" not in b[c6+1] or ("0" not in b[c6+1] and "1" not in b[c6+1]and "2" not
in b[c6+1]and "3" not in b[c6+1]and "4" not in b[c6+1]and "5" not in b[c6+1]and "6" not in b[c6+1]and "7" not in
b[c6+1]and "8" not in b[c6+1]and "9" not in b[c6+1])):
                           c6=c6+1
                           p1=b[c6].rfind ('=')
                           p2=b[c6].rfind (")
                           Proximo=b[c6][p1+2:p2]
                           c6=c6-1
                     if lugar2 == lugar1 and "S" in b[c6+1]:
                           c6=c6+1
                           p1=b[c6].rfind ('=')
                           p2=b[c6].rfind (")
                           Saida=b[c6][p1+3:p2]
                           c6=c6-1
                     c6=c6+3
               c5=c5+5
            c5=c2
            n=0
            while "Arco de T" not in b[c5]:
               p1=b[c5].rfind ('T')
               p2=b[c5].rfind (")
```

```
                transicao1=b[c5][p1:p2-1]
                p1=b[c5].find ('P')
                p2=b[c5].rfind ('p')
                lugar1=b[c5][p1:p2-1]
                if transicao1 == transicao and lugar1<>Atual1:
                    c6=3
                    while "Fim" not in b[c6] and n==0:
                        p1=b[c6].rfind ('P')
                        p2=b[c6].rfind (")
                        lugar2=b[c6][p1:p2-1]
                        if lugar2 == lugar1:
                            c6=c6+1
                            p1=b[c6].rfind ('=')
                            p2=b[c6].rfind (")
                            Entrada=b[c6][p1+3:p2]
                            c6=c6-1
                        c6=c6+3
                c5=c5+5
            matriz=matriz+[[Atual,Proximo,Entrada,Saida]]
        c3=c3+5
    c1=c1+3
l=1
l1=1+entrada
i=1+entrada
while l<(linha-entrada):
    while l1<=(linha-(entrada-1)):
        if matriz[l][0]>matriz[l1][0]:
            l2=0
            while l2< entrada:
                c=0
                while c<= coluna:
                    elemento=matriz[l+l2][c]
                    matriz[l+l2][c]=matriz[l1+l2][c]
                    matriz[l1+l2][c]=elemento
                    c=c+1
                l2=l2+1
        l1=l1+entrada
    l=l+entrada
    l1=i+entrada
    i=i+entrada
l=1
linha1=entrada
while l<=linha:
    l1=1
    while l+l1<=linha1:
        if matriz[l][2]>matriz[l1+l][2]:
            i=0
            while i<=coluna:
                elemento=matriz[l][i]
                matriz[l][i]=matriz[l1+l][i]
                matriz[l1+l][i]=elemento
                i=i+1
        l1=l1+1
    l=l+1
    if l==linha1:
        linha1=linha1+entrada
        l=l+1
l=1
if ('0'or '1' or '2'or '3' or '4'or '5' or '6'or '7' or '8'or '9') in matriz[1][0]:
    while l<=linha:
        p1=matriz[l][0].find (")
        p2=matriz[l][0].rfind (")
        elemento=matriz[l][0][p1+1:p2]
        caractere=matriz[l][0][p1:p2-1]
        matriz[l][0]=int (elemento)
        l=l+1
    l=1
```

```
    l1=1+entrada
    i=1+entrada
    while l<=(linha-entrada):
        while l1<= (linha-(entrada-1)):
            if matriz[l][0]>matriz[l1][0]:
                l2=0
                while l2< entrada:
                    c=0
                    while c<= coluna:
                        elemento=matriz[l+l2][c]
                        matriz[l+l2][c]=matriz[l1+l2][c]
                        matriz[l1+l2][c]=elemento
                        c=c+1
                    l2=l2+1
            l1=l1+entrada
        l=l+entrada
        l1=i+entrada
        i=i+entrada
    l=1
    while l<=linha:
        matriz [l][0]=str (matriz[l][0])
        matriz[l][0]=caractere+matriz[l][0]
        l=l+1

l=1
while l<=linha-(entrada-1):
    if l==1:
            arquivocopia.write (matriz[l][0])
    else:
            arquivocopia.write (", " + matriz[l][0])
    l=l+entrada

arquivocopia.write (");\n\tSIGNAL Atual, Proximo: tipo_estado;\nBEGIN")
arquivocopia.write ("\n\n\tPROCESS (Atual, x0")
e=1
while e<QE:
    arquivocopia.write (", x"+ str(e))
    e=e+1
arquivocopia.write (")\n\n\tBEGIN")
arquivocopia.write ("\n\t\tCASE Atual IS")
l=1
while l<=linha:
    arquivocopia.write ("\n\t\t\tWHEN " + matriz[l][0] + " =>")
    l1=0
    while l1<entrada:
        decimal= int(matriz[l+l1][2])
        posicao=0
        binario=[0]
        while posicao<QE:
            binario= binario + [decimal%2]
            decimal=decimal/2
            posicao=posicao+1
        posicao=QE
        while posicao > 0:
            if l1==0:
                if (QE-posicao)==0:
                    arquivocopia.write ("\n\t\t\t\tIF x" + str(QE-posicao)+"= '" + str(binario[posicao]))
                else:
                    arquivocopia.write ("' and x"+ str(QE-posicao)+"= '" + str(binario[posicao]))
            elif l1<entrada-1:
                if (QE-posicao)==0:
                    arquivocopia.write ("\n\t\t\t\tELSIF x" + str(QE-posicao)+"= '" + str(binario[posicao]))
                else:
                    arquivocopia.write ("' and x"+ str(QE-posicao)+"= '" + str(binario[posicao]))
            posicao=posicao-1
        if l1!= entrada-1:
            arquivocopia.write ("' THEN")
```

```
        else:
            arquivocopia.write("\n\t\t\t\tELSE")
        decimal= int(matriz[l+l1][3])
        posicao=0
        binario=[0]
        while posicao<QS:
            binario= binario + [decimal%2]
            decimal=decimal/2
            posicao=posicao+1
        posicao1=QS
        while posicao1>0:
            arquivocopia.write ("\n\t\t\t\t\t z" + str(QS-posicao1)+"<= '" + str(binario[posicao1]) +"';")
            posicao1=posicao1-1
        arquivocopia.write ("\n\t\t\t\tProximo <=" + matriz[l+l1][1] + ";")
        l1=l1+1
    l=l+entrada
    if l<linha:
        arquivocopia.write ("\n\t\t\t\tEND IF;")
arquivocopia.write ("\n\t\t\t\tEND IF;")
arquivocopia.write ("\n\t\tEND CASE;\n\tEND PROCESS;")
arquivocopia.write("\n\n\tPROCESS\n\tBEGIN")
arquivocopia.write("\n\t\tWAIT UNTIL clock'Event AND clock = '1';")
arquivocopia.write("\n\t\t\tAtual <= Proximo;\n\tEND PROCESS;")
arquivocopia.write ("\n\nEND "+ arq +";")
arquivo.close ()
arquivocopia.close()
```

Código Fonte do Programa PIPE2TAB5M

```
# UNESP Ilha Solteira
#Programa: PIPE2TAB5M
#Objetivo: Este programa tem por objetivo transformar a descrição PNML da RdP que modela uma FSM para uma tabela de transição de estados.
#Programadora: Giorjety Licorini Dias.
# -*- coding: cp1252 -*-
x=raw_input ("Digite o endereço do arquivo de entrada \n--por ex. 'Diretório:\Pasta\Nome do arquivo.extenção'\n")
arquivo= open (x,'r')
y=raw_input ("Digite o endereço do arquivo de saída \n--por ex. 'Diretório:\Pasta\Nome do arquivo. extenção'\n")
arquivocopia= open (y,'w+')
arquivo.seek(0)
b1=arquivo.read()
b=b1.splitlines()
c1=4 # Variável utilizada no while
QEs=0 # Variável responsável por armazenar a quantidade de estados da RdP
QE=0 # Variável responsável por armazenar a quantidade de entradas da RdP
QS=0  # Variável responsável por armazenar a quantidade de saídas da RdP
while "Fim" not in b[c1-1]:
    if "nome do lugar = E" in b[c1]:
        p1=b[c1].rfind ('E')
        p2=b[c1].rfind (")
        parametro=b[c1][p1+1:p2]
        if parametro <> '':
            QE=QE+1
            QEs=QEs-1
    QEs=QEs+1
    c1=c1+3
c1=4
entrada=QE
saida=0
while "Fim" not in b[c1-1]:
    if "/S" in b[c1]:
```

```
        Maquina="Moore"
        p1=b[c1].rfind ('S')
        p2=b[c1].rfind ('')
        parametro=b[c1][p1+1:p2]
        parametro=int(parametro)
        if parametro > saida:
            saida=parametro
    c1=c1+3
c2=c1+3
c8=c2
while "Arco" not in b [c2]:
    if "/S" in b[c2]:
        Maquina="Mealy"
        p1=b[c2].rfind ('S')
        p2=b[c2].rfind ('')
        parametro=b[c2][p1+1:p2]
        parametro=int(parametro)
        if parametro > saida:
            saida=parametro
    c2=c2+2
QS=saida+1
import math # importando o módulo matemático que provê a maioria das funções matemáticas mais familiares
QE = math.log(QE)/math.log(2.0)
if QE > int (QE):
    QE= int (QE)+1
else:
    QE= int (QE)
QS = math.log(QS)/math.log(2.0)
if QS > int (QS):
    QS= int (QS)+1
else:
    QS= int (QS)
QFF= math.log(QEs)/math.log(2.0)# Cálculo para a quantidade de Flip Flops
if QFF > int (QFF):
    QFF= int (QFF)+1
else:
    QFF= int (QFF)
linha=QEs*entrada
coluna=3
QEs = str (QEs)
QFF = str (QFF)
QE = str (QE)
QS = str (QS)
arquivocopia.write (QFF)
QFF = int (QFF)
i=1
while i <= QFF:
    x=raw_input ("Digite o tipo do flip flop \n")
    y=str(i)
    arquivocopia.write ("\n" + x)
    i=i+1
arquivocopia.write ("\n" + QE + ' ' + QS)
c1=4
matriz=[[]]
if Maquina == "Moore":
    while "Fim" not in b[c1-1]:
        p1=b[c1].rfind ('=')
        p2=b[c1].rfind ('/')
        if "E" not in b [c1]:
            Atual=b[c1][p1+2:p2]
            p1=b[c1].rfind ('/')
            p2=b[c1].rfind ('')
            Saida=b[c1][p1+2:p2]
            c1=c1-1
            p1=b[c1].rfind ('P')
            p2=b[c1].rfind ('')
            Atual1=b[c1][p1:p2-1]
```

```
            c1=c1+1
            c3=c2
            while "Arco de T" not in b[c3]:
                p1=b[c3].find ('P')
                p2=b[c3].rfind ('p')
                lugar=b[c3][p1:p2-1]
                if lugar == Atual1:
                    p1=b[c3].rfind ('T')
                    p2=b[c3].rfind ('')
                    transicao=b[c3][p1:p2-1]
                    c4=c2
                    while "Arco de T" not in b[c4]:
                        c4=c4+5
                    c5=c4
                    while "Fim da RdP" not in b[c5+1]:
                        p1=b[c5].find ('T')
                        p2=b[c5].find ('p')
                        transicao1=b[c5][p1:p2-1]
                        if transicao1 == transicao:
                            p1=b[c5].rfind ('P')
                            p2=b[c5].rfind ('')
                            lugar1=b[c5][p1:p2-1]
                            c6=3
                            while "Fim" not in b[c6]:
                                p1=b[c6].rfind ('P')
                                p2=b[c6].rfind ('')
                                lugar2=b[c6][p1:p2-1]
                                if lugar2 == lugar1:
                                    c6=c6+1
                                    p1=b[c6].rfind ('=')
                                    p2=b[c6].rfind ('/')
                                    Proximo=b[c6][p1+2:p2]
                                    c6=c6-1
                                c6=c6+3
                        c5=c5+5
                    c5=c2
                    while "Arco de T" not in b[c5]:
                        p1=b[c5].rfind ('T')
                        p2=b[c5].rfind ('')
                        transicao1=b[c5][p1:p2-1]
                        p1=b[c5].find ('P')
                        p2=b[c5].rfind ('p')
                        lugar1=b[c5][p1:p2-1]
                        if transicao1 == transicao and lugar1<>Atual1:
                            c6=3
                            while "Fim" not in b[c6]:
                                p1=b[c6].rfind ('P')
                                p2=b[c6].rfind ('')
                                lugar2=b[c6][p1:p2-1]
                                if lugar2 == lugar1:
                                    c6=c6+1
                                    p1=b[c6].rfind ('=')
                                    p2=b[c6].rfind ('')
                                    Entrada=b[c6][p1+3:p2]
                                    c6=c6-1
                                c6=c6+3
                        c5=c5+5
                    matriz=matriz+[[Atual,Proximo,Entrada,Saida]]
                c3=c3+5
        c1=c1+3
c1=4
if Maquina == "Mealy":
    while "Fim" not in b[c1-1]:
        p1=b[c1].rfind ('=')
        p2=b[c1].rfind ('')
        if "E" not in b [c1]:
            Atual=b[c1][p1+2:p2]
```

```
c1=c1-1
p1=b[c1].rfind ('P')
p2=b[c1].rfind ('')
Atual1=b[c1][p1:p2-1]
c1=c1+1
c3=c2
while "Arco de T" not in b[c3]:
    p1=b[c3].find ('P')
    p2=b[c3].rfind ('p')
    lugar=b[c3][p1:p2-1]
    if lugar == Atual1:
        p1=b[c3].rfind ('T')
        p2=b[c3].rfind ('')
        transicao=b[c3][p1:p2-1]
        c4=c2
        while "Arco de T" not in b[c4]:
            c4=c4+5
        c5=c4
        while "Fim da RdP" not in b[c5+1]:
            p1=b[c5].find ('T')
            p2=b[c5].find ('p')
            transicao1=b[c5][p1:p2-1]
            if transicao1 == transicao:
                p1=b[c5].rfind ('P')
                p2=b[c5].rfind ('')
                lugar1=b[c5][p1:p2-1]
                c6=3
                while "Fim" not in b[c6]:
                    p1=b[c6].rfind ('P')
                    p2=b[c6].rfind ('')
                    lugar2=b[c6][p1:p2-1]
                    if lugar2 == lugar1:
                        c6=c6+1
                        p1=b[c6].rfind ('=')
                        p2=b[c6].rfind ('')
                        Proximo=b[c6][p1+2:p2]
                        c6=c6-1
                    c6=c6+3
            c5=c5+5
        c5=c2
        n=0
        while "Arco de T" not in b[c5]:
            p1=b[c5].rfind ('T')
            p2=b[c5].rfind ('')
            transicao1=b[c5][p1:p2-1]
            p1=b[c5].find ('P')
            p2=b[c5].rfind ('p')
            lugar1=b[c5][p1:p2-1]
            if transicao1 == transicao and lugar1<>Atual1:
                c6=3
                while "Fim" not in b[c6] and n==0:
                    p1=b[c6].rfind ('P')
                    p2=b[c6].rfind ('')
                    lugar2=b[c6][p1:p2-1]
                    if lugar2 == lugar1:
                        c6=c6+1
                        p1=b[c6].rfind ('=')
                        p2=b[c6].rfind ('')
                        Entrada=b[c6][p1+3:p2]
                        c6=c6-1
                        n=1
                        c7=c8
                        while "Arco" not in b[c7]:
                            c7=c7-1
                            p1=b[c7].rfind ('T')
                            p2=b[c7].rfind ('')
                            transicao2=b[c7][p1:p2-1]
```

```
                                c7=c7+1
                                if transicao2 == transicao:
                                    p1=b[c7].rfind ('/')
                                    p2=b[c7].rfind (")
                                    Saida=b[c7][p1+2:p2]
                                    n=1
                                c7=c7+2
                            c6=c6+3
                        c5=c5+5
                    matriz=matriz+[[Atual,Proximo,Entrada,Saida]]
                c3=c3+5
        c1=c1+3
l=1
l1=1+entrada
i=1+entrada
while l<(linha-entrada):
    while l1<=(linha-(entrada-1)):
        if matriz[l][0]>matriz[l1][0]:
            l2=0
            while l2< entrada:
                c=0
                while c<= coluna:
                    elemento=matriz[l+l2][c]
                    matriz[l+l2][c]=matriz[l1+l2][c]
                    matriz[l1+l2][c]=elemento
                    c=c+1
                l2=l2+1
        l1=l1+entrada
    l=l+entrada
    l1=i+entrada
    i=i+entrada
l=1
linha1=entrada
while l<=linha:
    l1=1
    while l+l1<=linha1:
        if matriz[l][2]>matriz[l1+l][2]:
            i=0
            while i<=coluna:
                elemento=matriz[l][i]
                matriz[l][i]=matriz[l1+l][i]
                matriz[l1+l][i]=elemento
                i=i+1
        l1=l1+1
    l=l+1
    if l==linha1:
        linha1=linha1+entrada
        l=l+1
l=1
if ('0'or '1' or '2'or '3' or '4'or '5' or '6'or '7' or '8'or '9') in matriz[1][0]:
    while l<=linha:
        p1=matriz[l][0].find (")
        p2=matriz[l][0].rfind (")
        elemento=matriz[l][0][p1+1:p2]
        caractere=matriz[l][0][p1:p2-1]
        matriz[l][0]=int (elemento)
        l=l+1
    l=1
    l1=1+entrada
    i=1+entrada
    while l<=(linha-entrada):
        while l1<= (linha-(entrada-1)):
            if matriz[l][0]>matriz[l1][0]:
                l2=0
                while l2< entrada:
                    c=0
                    while c<= coluna:
```

```
                    elemento=matriz[l+l2][c]
                    matriz[l+l2][c]=matriz[l1+l2][c]
                    matriz[l1+l2][c]=elemento
                    c=c+1
                l2=l2+1
            l1=l1+entrada
        l=l+entrada
        l1=i+entrada
        i=i+entrada
    l=1
    while l<=linha:
        matriz [l][0]=str (matriz[l][0])
        matriz[l][0]=caractere+matriz[l][0]
        l=l+1
l=1
while l<=linha:
    arquivocopia.write ("\n")
    l1=0
    while l1<=coluna:
        if l1==0:
            arquivocopia.write (matriz[l][l1])
        else:
            arquivocopia.write (" " + matriz[l][l1])
        l1=l1+1
    l=l+1
arquivo.close ()
arquivocopia.close()
```

Código Fonte do Programa PIPE2VHDL5M

```
# UNESP Ilha Solteira
#Programa: PIPE2VHDL5M
#Objetivo: Este programa tem por objetivo transformar a descrição de uma máquina de estados finitos em Rede de Petri para o seu modelo comportamental descrito em VHDL.
#Programadora: Giorjety Licorini Dias.
# DISSERTAÇÃO DE MESTRADO
### -*- coding: cp1252 -*-
x=raw_input ("Digite o endereço do arquivo de entrada \n--por ex. 'Diretório:\Pasta\Nome do arquivo.extenção'\n")
arquivo= open (x,'r')
y=raw_input ("Digite o endereço do arquivo de saída \n--por ex. 'Diretório:\Pasta\Nome do arquivo. extenção'\n")
arquivocopia= open (y,'w')
b4=y.rfind ('\\')
b5=y.rfind ('.')
arquivo.seek(0)
entidade= y [b4+1:b5]
arquivocopia.write ("-- PROJETO DE TESE DE MESTRADO - Ferramenta para a Integração de Redes de Petri e VHDL na Síntese de Sistemas Digitais")
arquivocopia.write ("\n-- A ferramenta tem por objetivo transformar a descrição de uma máquina de estados finitos em Rede de Petri para o seu modelo comportamental descrito em VHDL")
arquivocopia.write ("\n\n--Programa: PIPE2VHDL5M ")
arquivocopia.write ("\n--Programadora: Giorjety Licorini Dias")
arquivocopia.write ("\n--Versão: 1.0 de 26 de agosto de 2006")
arquivocopia.write ("\n\nENTITY "+ entidade +" IS\n")
arquivo.seek(0)
b1=arquivo.read()
b=b1.splitlines()
c1=4 # Variável utilizada no while
QEs=0 # Variável responsável por armazenar a quantidade de estados da RdP
QE=0 # Variável responsável por armazenar a quantidade de entradas da RdP
QS=0  # Variável responsável por armazenar a quantidade de saídas da RdP
while "Fim" not in b[c1-1]: # Contabiliza a quantidade de Entradas e Estados da FSM
```

```
    if "nome do lugar = E" in b[c1]:
        p1=b[c1].rfind ('E')
        p2=b[c1].rfind ('')
        parametro=b[c1][p1+1:p2]
        if parametro <> '':
            QE=QE+1
            QEs=QEs-1
    QEs=QEs+1
    c1=c1+3
c1=4
entrada=QE
saida=0
while "Fim" not in b[c1-1]:
    if "/S" in b[c1]:
        Maquina="Moore"
        p1=b[c1].rfind ('S')
        p2=b[c1].rfind ('')
        parametro=b[c1][p1+1:p2]
        parametro=int(parametro)
        if parametro > saida:
            saida=parametro
    c1=c1+3
c2=c1+3
c8=c2
while "Arco" not in b [c2]:
    if "/S" in b[c2]:
        Maquina="Mealy"
        p1=b[c2].rfind ('S')
        p2=b[c2].rfind ('')
        parametro=b[c2][p1+1:p2]
        parametro=int(parametro)
        if parametro > saida:
            saida=parametro
    c2=c2+2
QS=saida+1
import math # importando o módulo matemático que provê a maioria das funções matemáticas mais familiares
QE = math.log(QE)/math.log(2.0)
if QE > int (QE):
    QE= int (QE)+1
else:
    QE= int (QE)
QS = math.log(QS)/math.log(2.0)
if QS > int (QS):
    QS= int (QS)+1
else:
    QS= int (QS)
arquivocopia.write ("\tPORT\n\t(\n\t\t--xn representão as variáveis de entrada")
arquivocopia.write ("\n\t\t--zn representão as variáveis de saída")
arquivocopia.write ("\n\t\tclock:IN BIT;")
arquivocopia.write ("\n\t\tx0")
e=1
while e<QE:
    arquivocopia.write (", x"+ str(e))
    e=e+1
arquivocopia.write (":IN BIT;")
arquivocopia.write ("\n\t\tz0")
s=1
while s<QS:
    arquivocopia.write (", z"+ str(s))
    s=s+1
arquivocopia.write (":OUT BIT \n\t);")
arquivocopia.write ("\nEND "+ entidade +";\n")
arq=raw_input ("Digite o nome da Arquitetura que vc deseja para o código VHDL\n")
arquivocopia.write ("\nARCHITECTURE "+ arq +" OF " + entidade +" IS\n")
arquivocopia.write ("\n\tTYPE tipo_estado IS (")
linha=QEs*entrada
coluna=3
```

```
c1=4
matriz=[[]]
if Maquina == "Moore":
    while "Fim" not in b[c1-1]:
        p1=b[c1].rfind ('=')
        p2=b[c1].rfind ('/')
        if "E" not in b [c1]:
            Atual=b[c1][p1+2:p2]
            p1=b[c1].rfind ('/')
            p2=b[c1].rfind ('')
            Saida=b[c1][p1+2:p2]
            c1=c1-1
            p1=b[c1].rfind ('P')
            p2=b[c1].rfind ('')
            Atual1=b[c1][p1:p2-1]
            c1=c1+1
            c3=c2
            while "Arco de T" not in b[c3]:
                p1=b[c3].find ('P')
                p2=b[c3].rfind ('p')
                lugar=b[c3][p1:p2-1]
                if lugar == Atual1:
                    p1=b[c3].rfind ('T')
                    p2=b[c3].rfind ('')
                    transicao=b[c3][p1:p2-1]
                    c4=c2
                    while "Arco de T" not in b[c4]:
                        c4=c4+5
                    c5=c4
                    while "Fim da RdP" not in b[c5+1]:
                        p1=b[c5].find ('T')
                        p2=b[c5].find ('p')
                        transicao1=b[c5][p1:p2-1]
                        if transicao1 == transicao:
                            p1=b[c5].rfind ('P')
                            p2=b[c5].rfind ('')
                            lugar1=b[c5][p1:p2-1]
                            c6=3
                            while "Fim" not in b[c6]:
                                p1=b[c6].rfind ('P')
                                p2=b[c6].rfind ('')
                                lugar2=b[c6][p1:p2-1]
                                if lugar2 == lugar1:
                                    c6=c6+1
                                    p1=b[c6].rfind ('=')
                                    p2=b[c6].rfind ('/')
                                    Proximo=b[c6][p1+2:p2]
                                    c6=c6-1
                                c6=c6+3
                        c5=c5+5
                    c5=c2
                    while "Arco de T" not in b[c5]:
                        p1=b[c5].rfind ('T')
                        p2=b[c5].rfind ('')
                        transicao1=b[c5][p1:p2-1]
                        p1=b[c5].find ('P')
                        p2=b[c5].rfind ('p')
                        lugar1=b[c5][p1:p2-1]
                        if transicao1 == transicao and lugar1<>Atual1:
                            c6=3
                            while "Fim" not in b[c6]:
                                p1=b[c6].rfind ('P')
                                p2=b[c6].rfind ('')
                                lugar2=b[c6][p1:p2-1]
                                if lugar2 == lugar1:
                                    c6=c6+1
                                    p1=b[c6].rfind ('=')
```

```
                        p2=b[c6].rfind ('')
                        Entrada=b[c6][p1+3:p2]
                        c6=c6-1
                    c6=c6+3
                c5=c5+5
            matriz=matriz+[[Atual,Proximo,Entrada,Saida]]
        c3=c3+5
    c1=c1+3

c1=4
if Maquina == "Mealy":
    while "Fim" not in b[c1-1]:
        p1=b[c1].rfind ('=')
        p2=b[c1].rfind ('')
        if "E" not in b [c1]:
            Atual=b[c1][p1+2:p2]
            c1=c1-1
            p1=b[c1].rfind ('P')
            p2=b[c1].rfind ('')
            Atual1=b[c1][p1:p2-1]
            c1=c1+1
            c3=c2
            while "Arco de T" not in b[c3]:
                p1=b[c3].find ('P')
                p2=b[c3].rfind ('p')
                lugar=b[c3][p1:p2-1]
                if lugar == Atual1:
                    p1=b[c3].rfind ('T')
                    p2=b[c3].rfind ('')
                    transicao=b[c3][p1:p2-1]
                    c4=c2
                    while "Arco de T" not in b[c4]:
                        c4=c4+5
                    c5=c4
                    while "Fim da RdP" not in b[c5+1]:
                        p1=b[c5].find ('T')
                        p2=b[c5].find ('p')
                        transicao1=b[c5][p1:p2-1]
                        if transicao1 == transicao:
                            p1=b[c5].rfind ('P')
                            p2=b[c5].rfind ('')
                            lugar1=b[c5][p1:p2-1]
                            c6=3
                            while "Fim" not in b[c6]:
                                p1=b[c6].rfind ('P')
                                p2=b[c6].rfind ('')
                                lugar2=b[c6][p1:p2-1]
                                if lugar2 == lugar1:
                                    c6=c6+1
                                    p1=b[c6].rfind ('=')
                                    p2=b[c6].rfind ('')
                                    Proximo=b[c6][p1+2:p2]
                                    c6=c6-1
                                c6=c6+3
                        c5=c5+5
                    c5=c2
                    n=0
                    while "Arco de T" not in b[c5]:
                        p1=b[c5].rfind ('T')
                        p2=b[c5].rfind ('')
                        transicao1=b[c5][p1:p2-1]
                        p1=b[c5].find ('P')
                        p2=b[c5].rfind ('p')
                        lugar1=b[c5][p1:p2-1]
                        if transicao1 == transicao and lugar1<>Atual1:
                            c6=3
                            while "Fim" not in b[c6] and n==0:
```

```
                        p1=b[c6].rfind ('P')
                        p2=b[c6].rfind ('')
                        lugar2=b[c6][p1:p2-1]
                        if lugar2 == lugar1:
                            c6=c6+1
                            p1=b[c6].rfind ('=')
                            p2=b[c6].rfind ('')
                            Entrada=b[c6][p1+3:p2]
                            c6=c6-1
                            n=1
                            c7=c8
                            while "Arco" not in b[c7]:
                                c7=c7-1
                                p1=b[c7].rfind ('T')
                                p2=b[c7].rfind ('')
                                transicao2=b[c7][p1:p2-1]
                                c7=c7+1
                                if transicao2 == transicao:
                                    p1=b[c7].rfind ('/')
                                    p2=b[c7].rfind ('')
                                    Saida=b[c7][p1+2:p2]
                                    n=1
                                c7=c7+2
                        c6=c6+3
                c5=c5+5
            matriz=matriz+[[Atual,Proximo,Entrada,Saida]]
        c3=c3+5
    c1=c1+3
l=1
l1=1+entrada
i=1+entrada
while l<(linha-entrada):
    while l1<=(linha-(entrada-1)):
        if matriz[l][0]>matriz[l1][0]:
            l2=0
            while l2< entrada:
                c=0
                while c<= coluna:
                    elemento=matriz[l+l2][c]
                    matriz[l+l2][c]=matriz[l1+l2][c]
                    matriz[l1+l2][c]=elemento
                    c=c+1
                l2=l2+1
        l1=l1+entrada
    l=l+entrada
    l1=i+entrada
    i=i+entrada
l=1
linha1=entrada
while l<=linha:
    l1=1
    while l+l1<=linha1:
        if matriz[l][2]>matriz[l1+l][2]:
            i=0
            while i<=coluna:
                elemento=matriz[l][i]
                matriz[l][i]=matriz[l1+l][i]
                matriz[l1+l][i]=elemento
                i=i+1
        l1=l1+1
    l=l+1
    if l==linha1:
        linha1=linha1+entrada
        l=l+1
l=1
if ('0'or '1' or '2'or '3' or '4'or '5' or '6'or '7' or '8'or '9') in matriz[1][0]:
    while l<=linha:
```

```
        p1=matriz[l][0].find (')
        p2=matriz[l][0].rfind (')
        elemento=matriz[l][0][p1+1:p2]
        caractere=matriz[l][0][p1:p2-1]
        matriz[l][0]=int (elemento)
        l=l+1
    l=1
    l1=1+entrada
    i=1+entrada
    while l<=(linha-entrada):
        while l1<= (linha-(entrada-1)):
            if matriz[l][0]>matriz[l1][0]:
                l2=0
                while l2< entrada:
                    c=0
                    while c<= coluna:
                        elemento=matriz[l+l2][c]
                        matriz[l+l2][c]=matriz[l1+l2][c]
                        matriz[l1+l2][c]=elemento
                        c=c+1
                    l2=l2+1
            l1=l1+entrada
        l=l+entrada
        l1=i+entrada
        i=i+entrada
    l=1
    while l<=linha:
        matriz [l][0]=str (matriz[l][0])
        matriz[l][0]=caractere+matriz[l][0]
        l=l+1
l=1
while l<=linha-(entrada-1):
    if l==1:
            arquivocopia.write (matriz[l][0])
    else:
            arquivocopia.write (", " + matriz[l][0])
    l=l+entrada

arquivocopia.write (");\n\tSIGNAL Atual, Proximo: tipo_estado;\nBEGIN")
arquivocopia.write ("\n\n\tPROCESS (Atual, x0")
e=1
while e<QE:
    arquivocopia.write (", x"+ str(e))
    e=e+1
arquivocopia.write (")\n\n\tBEGIN")
arquivocopia.write ("\n\t\tCASE Atual IS")
n=1
if Maquina=="Moore":
    while n<=linha:
        arquivocopia.write ("\n\t\t\tWHEN " + matriz[n][0] + " =>")
        n1=0
        while n1<entrada:
            decimal= int(matriz[n+n1][2])
            posicao=0
            binario=[0]
            while posicao<QE:
                binario= binario + [decimal%2]
                decimal=decimal/2
                posicao=posicao+1
            posicao=QE
            while posicao>0:
                if n1==0:
                    if (QE-posicao)==0:
                        arquivocopia.write ("\n\t\t\t\tIF x" + str(QE-posicao)+"= '" + str(binario[posicao]))
                    else:
                        arquivocopia.write ("' and x"+ str(QE-posicao)+"= '" + str(binario[posicao]))
                elif n1<entrada:
```

```
                if (QE-posicao)==0:
                    arquivocopia.write ("\n\t\t\t\tELSIF x" + str(QE-posicao)+"= '" + str(binario[posicao]))
                else:
                    arquivocopia.write ("' and x"+ str(QE-posicao)+"= '" + str(binario[posicao]))
            posicao=posicao-1
        if n1!= entrada:
            arquivocopia.write ("' THEN")
        #else:
            #arquivocopia.write ("\n\t\t\t\tELSE")
        decimal= int(matriz[n+n1][3])
        posicao=0
        binario=[0]
        while posicao<QS:
            binario= binario + [decimal%2]
            decimal=decimal/2
            posicao=posicao+1
        arquivocopia.write ("\n\t\t\t\t\tProximo <=" + matriz[n+n1][1] + ";")
        n1=n1+1
    n=n+entrada
    if n<=linha+1:
        arquivocopia.write ("\n\t\t\t\tEND IF;")
    posicao1=QS
    while posicao1>0:
        arquivocopia.write ("\n\t\t\t\tz" + str(QS-posicao1)+"<= '" + str(binario[posicao1]) +"';")
        posicao1=posicao1-1
n=1
if Maquina=="Mealy":
    while n<=linha:
        arquivocopia.write ("\n\t\t\tWHEN " + matriz[n][0] + " =>")
        n1=0
        while n1<entrada:
            decimal= int(matriz[n+n1][2])
            posicao=0
            binario=[0]
            while posicao<QE:
                binario= binario + [decimal%2]
                decimal=decimal/2
                posicao=posicao+1
            posicao=QE
            while posicao > 0:
                if n1==0:
                    if (QE-posicao)==0:
                        arquivocopia.write ("\n\t\t\t\tIF x" + str(QE-posicao)+"= '" + str(binario[posicao]))
                    else:
                        arquivocopia.write ("' and x"+ str(QE-posicao)+"= '" + str(binario[posicao]))
                elif n1<entrada-1:
                    if (QE-posicao)==0:
                        arquivocopia.write ("\n\t\t\t\tELSIF x" + str(QE-posicao)+"= '" + str(binario[posicao]))
                    else:
                        arquivocopia.write ("' and x"+ str(QE-posicao)+"= '" + str(binario[posicao]))
                posicao=posicao-1
            if n1!= entrada-1:
                arquivocopia.write ("' THEN")
            else:
                arquivocopia.write ("\n\t\t\t\tELSE")
            decimal= int(matriz[n+n1][3])
            posicao=0
            binario=[0]
            while posicao<QS:
                binario= binario + [decimal%2]
                decimal=decimal/2
                posicao=posicao+1
            posicao1=QS
            while posicao1>0:
                arquivocopia.write ("\n\t\t\t\t\t z" + str(QS-posicao1)+"<= '" + str(binario[posicao1]) +"';")
                posicao1=posicao1-1
            arquivocopia.write ("\n\t\t\t\t\tProximo <=" + matriz[n+n1][1] + ";")
```

```
            n1=n1+1
        n=n+entrada
        if n<linha:
            arquivocopia.write ("\n\t\t\t\tEND IF;")
    arquivocopia.write ("\n\t\t\t\tEND IF;")
arquivocopia.write ("\n\t\tEND CASE;\n\tEND PROCESS;")
arquivocopia.write("\n\n\tPROCESS\n\tBEGIN")
arquivocopia.write("\n\t\tWAIT UNTIL clock'Event AND clock = '1';")
arquivocopia.write("\n\t\t\tAtual <= Proximo;\n\tEND PROCESS;")
arquivocopia.write ("\n\nEND "+ arq +";")
arquivo.close ()
arquivocopia.close()
```