UNIVERSIDADE NOVA DE LISBOA

FACULDADE DE CIÊNCIAS E TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA ELECTROTÉCNICA

GERAÇÃO DE CÓDIGO VHDL A PARTIR DE ESPECIFICAÇÕES IOPT PNML2VHDL

Por:

Paulo Luís Gonçalves Lima

Dissertação apresentada na Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa para obtenção de grau de Mestre em Engenharia Electrotécnica e de Computadores.

Orientador:

Prof. Doutor Luís Filipe dos Santos Gomes

Lisboa
2009

Agradecimentos

Ao Professor Dr. Luís Gomes por todo o apoio e suporte disponibilizados ao longo da realização deste trabalho, em particular aos seus conselhos que permitiram levar este trabalho a um bom porto e em tempo útil.

Aos meus pais, Domingos e Rosa, por todo o apoio, paciência e tempo disponibilizados ao longo dos dias em que as coisas pareciam complicadas e dificeis.

A todos os demais que neste pequeno período de tempo deram todo o apoio e disponibilidade para que este trabalho fosse realizado.

A todos o meu obrigado.

"Criar é afirmar no homem o sonho de divinização"

Vergílio Ferreira, in 'Carta ao Futuro'

Sumário

Com o aumento da complexidade no desenvolvimento de sistemas digitais existe uma necessidade cada vez maior de novas ferramentas que associadas a metodologias, criem mecanismos de abstracção tornando o desenvolvimento mais simples e menos demorado.

A utilização de Redes de Petri para a modelação de sistemas de eventos discretos e especificação de controladores digitais tem sido bastante discutida havendo no entanto uma falta de ferramentas no que respeita à implementação desses modelos quando se considera a geração automática de código.

Este trabalho tem como objectivo criar regras para a geração automática de código VHDL a serem usadas por uma nova ferramenta de ajuda à implementação de projectos de sistemas síncronos controlados a eventos discretos e baseados na utilização de Redes de Petri.

A ferramenta é baseada numa classe de Redes de Petri denominada *Input-Output Place-Transition Petri Net* (*IOPT*), representada através de *Petri Net Markup Language* (*PNML*). Esta classe IOPT é baseada nas redes lugar/transição e em conceitos bem estudados de Redes de Petri sincronizadas e interpretadas, permitindo a associação de sinais externos de entrada e eventos de entrada e saída a transições e a associação de sinais de saída externos a lugares e a eventos de saída.

Denominada por PNML2VHDL, a ferramenta permite gerar a partir de especificações IOPT uma descrição em VHDL da rede de forma a ser automaticamente implementada, por exemplo, numa FPGA.

A necessidade desta ferramenta surge após uma cuidada análise das ferramentas existentes na actualidade. A inexistência de tal ferramenta motivou a sua criação. Será, certamente, uma ferramenta da maior utilidade para projectistas e de forma particular para o projecto FORDESIGN.

A ênfase da dissertação está nas regras de tradução directa que permitem a tradução do comportamento de uma Rede de Petri descrita em IOPT para uma descrição em VHDL. Na

tradução usa-se o método de atribuir a cada característica do modelo em IOPT uma estrutura em VHDL predefinida a que se deu o nome de regra de tradução.

Após a definição das regras de tradução, a ferramenta é validada através da realização de diversos exemplos concretos e já bens estudados na literatura existente, como é o caso do parque de estacionamento.

Palavras Chave: PNML, VHDL, FORDESIGN, FPGA, IOPT, Redes de Petri

Abstract

With the increasing complexity in the development of digital systems there is a necessity to create new tools that associated with methodologies, creates mechanisms of abstraction to simplify the development.

The usage of Petri Nets for the modeling of discrete events systems and specification of digital controllers has been widely discussed. There is however a lack of tools in what concerns with the implementation of these models considering the automatic generation of code.

Thus, this work has the objective to create rules for the automatic generation of VHDL code based on the use of Petri nets models. These rules will be integrated in a new tool, considering the implementation of synchronous systems controlled by discrete events.

The tool is based on a class of nets called *Input-Output Place-Transition Petri Net* (IOPT), which is represented using the Petri Net Markup Language (PNML) format. This class of Petri Nets is based on the place/transition nets and is based on well-known concepts from synchronized and interpreted Petri Nets, allowing the association of external input signals and input and output events to the transitions and the association of external output signals for marking places and output events.

This tool, named as PNML2VHDL, intends to allow the generation of a description in VHDL from IOPT Petri Nets specifications amenable to be automatically deployed, for example, in a FPGA.

The necessity of this tool appears after an analysis of the current existing tools. The inexistence of such tool (in particular for class IOPT) was the motivation for its creation. It will be, certainly, a tool very effective to be used by designers, and in a particular way within FORDESIGN project.

The emphasis of this dissertation is in the definition of a set of rules allowing the direct translation of the behavior of a Petri Net described through a IOPT model into a description in

VHDL. Translation rules are using the method of associating to each characteristic of the IOPT model a specific structure predefined in VHDL.

After defining the translation rules, the tool is validated through a set of concrete examples, already widely used in existing literature, as it is the case of parking lot controllers.

Keys-Words: PNML, VHDL, FORDESIGN, FPGA, IOPT, Petri Net

Abreviaturas

ASIC	*Application-Specific Integrated Circuit*
CAD	*Computer-Aided Design*
CPLD	*Complex Programmable Logic Device*
CPN	*Coloured Petri Net*
EDA	*Electronic Design Automation*
ER	*Entity-Relationship Model*
FPGA	*Field Programmable Gate Array*
FSM	*Finite State Machine*
IEEE	*Institute of Electrical and Electronics Engineers*
IOPT	*Input-Output Place-Transition*
MTBF	*Mean Time Beteween Failure*
PNML	*Petri Net Markup Language*
PNTD	*Petri Net Type Definition Files*
RdP	*Redes de Petri*
RTL	*Register – Tranfer Level*
SIPN	*Signal Interpreted Petri Nets*
VHDL	*VHSIC Hardware Description Languagem*
VHSIC	*Very High-Speed Integrated Circuits*
XML	*Extensive Markup Language*

Índice de Matérias

Índice de Figuras

1. Introdução

Neste capítulo pretende-se dar a conhecer a motivação que levou à realização deste trabalho, os seus objectivos e a sua estrutura. Pretende-se também analisar o estado da arte das ferramentas disponíveis actualmente.

1.1. Motivação

A teoria de Redes de Petri tem vindo a ser cada vez mais aplicada nas mais diversas áreas de aplicação e projectos. O seu simples formalismo matemático aliado à flexibilidade da sua representação gráfica permitem modelar problemas complexos, permitindo a aplicação dos seus conceitos em diversas áreas do conhecimento e não se limitando às áreas de engenharia.

Através de uma pesquisa na Internet podem ser encontradas imensas aplicações que utilizam Redes de Petri. Na sua maioria permitem apenas a sua edição, simulação e análise, não tendo sido facilmente encontradas ferramentas dedicadas à implementação prática do modelo gerado pela ferramenta de edição.

Devido a esta lacuna de ferramentas destinadas à implementação, pretende-se contribuir com a implementação de um tradutor cujo objectivo é o de traduzir RdPs representadas pela classe IOPT [4] para VHDL, gerando uma descrição de um sistema/circuito síncrono.

1.2. Objectivos

Com esta dissertação pretende-se documentar um conjunto de regras que possibilitem a tradução das várias características de uma rede de Petri IOPT (*Input-Output Place-Transition*) para VHDL.

A classe de redes de Petri IOPT é uma extensão das redes lugar-transição [6]. O seu objectivo é permitir a construção de modelos que representam controladores, permitindo a modelação de sinais e eventos externos associados às transições e lugares.

Para a tradução é usada uma estratégia de implementação directa, ou seja, para cada característica de uma rede IOPT faz-se corresponder um conjunto de características em VHDL obtendo-se no final um circuito síncrono controlado a eventos discretos.

Após documentação das regras será desenvolvida uma ferramenta para que, em conjunto com um editor de redes (fora do âmbito deste trabalho), se possa automatizar a geração de controladores a partir de uma Rede de Petri (RdP) interpretada usando as regras descritas. O editor terá o objectivo de criar graficamente a rede, gravando-a em PNML quando terminada. De seguida o conversor irá usar o PNML gerado para aplicar as regras de tradução e obter as especificações VHDL da rede.

Para exemplo, considere-se uma dada Rede de Petri IOPT que modela o controlador para um parque de estacionamento. A descrição inicial do sistema é feita num modelo formal e de mais alto nível de abstracção, através de um editor de Redes de Petri IOPT, permitindo resolver ou facilitar problemas clássicos de projecto de sistemas digitais complexos, tais como a validação e verificação formal dos mesmos. De seguida é realizada a geração automática da sua descrição em VHDL através do tradutor proposto de forma a ser implementada numa plataforma como a Spartan-3 [32].

1.3. Estrutura da Dissertação

A dissertação está dividida em sete capítulos. Além desta introdução, onde se inclui uma análise das vantagens e desvantagens de ferramentas existentes na actualidade, serão enfatizados no capítulo II conceitos relacionados à teoria de Redes de Petri, em particular à classe IOPT. Serão também abordadas questões em relação à teoria e uso de PNML e da Linguagem de Descrição de Hardware VHDL (com foco nas vantagens e desvantagens). Cada um destes estará num sub-capítulo do capítulo II onde se irá detalhar as características, propriedades e conceitos.

No capítulo III são apresentadas as regras de tradução que serão utilizadas pela aplicação. Este capítulo está dividido em sub-capítulos onde se descreve a regra de tradução proposta para cada característica de uma rede IOPT.

No capítulo IV são apresentados exemplos de aplicação fazendo a conversão através do uso das regras de tradução apresentadas no capítulo III. A sua boa tradução pode ser verificada pelas diversas simulações apresentadas do comportamento do modelo após a tradução.

No capítulo V apresenta-se a aplicação desenvolvida, salientando-se as principais funcionalidades desta e a forma como poderá ser integrada no projecto FORDESIGN [4][20].

Finalmente, no capítulo VI apresentam-se as conclusões da dissertação e sugestões para trabalhos futuros com vista à continuidade deste projecto.

Além dos capítulos, o trabalho contém quatro anexos. O Anexo 1 pretende apresentar informação sobre a aplicação desenvolvida no âmbito desta dissertação. O Anexo 2 apresenta os possíveis avisos e erros que a aplicação devolve ao utilizador quando executada. No Anexo 3 e Anexo 4 é possível obter exemplos de redes traduzidas, em particular a rede do parque de estacionamento com uma entrada e uma saída e com duas entradas e uma saída, respectivamente.

1.4. Análise de ferramentas existentes

Através desta secção pretende-se dar a conhecer o estado da arte no que respeita a ferramentas que permitem a implementação de especificações de Redes de Petri.

Através de uma pesquisa na Internet, e em particular no repositório de ferramentas do site *World of Petri Nets* [34], verifica-se a inexistência de aplicações ou soluções com o objectivo de fazer a geração automática de código VHDL a partir de especificações de RdP. A informação encontrada é essencialmente sobre a forma como pode ser feita a tradução, como por exemplo em [27], [28] e [29] (apenas são referenciadas RdP seguras).

No que respeita a aplicações, foram encontradas diversas ferramentas que permitem a simulação da RdP através da geração de código C, C++ ou Java, mas muito poucas existem vocacionadas para a geração de código de controladores baseados em RdP's.

No que respeita à implementação, é possível distinguir dois grandes grupos de ferramentas:

- O primeiro grupo está vocacionado para a implementação de controladores assíncronos. Em [44] é referenciada a ferramenta Petrify que permite a tradução de RdP para hardware através de circuitos assíncronos. A ferramenta recebe como entrada uma RdP e informação sobre os sinais, produzindo de seguida a informação necessária para criar o controlador assíncrono [30].
- O segundo grupo pertence às ferramentas que fazem a implementação usando PLCs (Programmable Logic Controllers). Neste caso foram encontradas duas ferramentas:
    - SIPN-Editor [45], que permite a edição, visualização e tradução hierárquica de redes de Petri SIPN's, convertendo-a para instruções adequadas para proceder à implementação em PLC's [31]. Esta ferramenta foi desenvolvida exclusivamente para este fim;
    - *PetriLLD* [41], foi desenvolvida em Java com o objectivo de automatizar a implementação de controladores em PLCs baseados em RdP. Permite, a partir de modelos de redes elementares, a geração de linguagens IEC 61131-3 (incluindo LD, ST, and IL) capazes de serem usadas em PLC's de diversos fabricantes. É distribuída como ferramenta gratuita e permite a adição de novos *plugins* em Java para novos PLC's. Permite também a simulação da rede. A Figura 1.1 apresenta a interface gráfica da ferramenta.

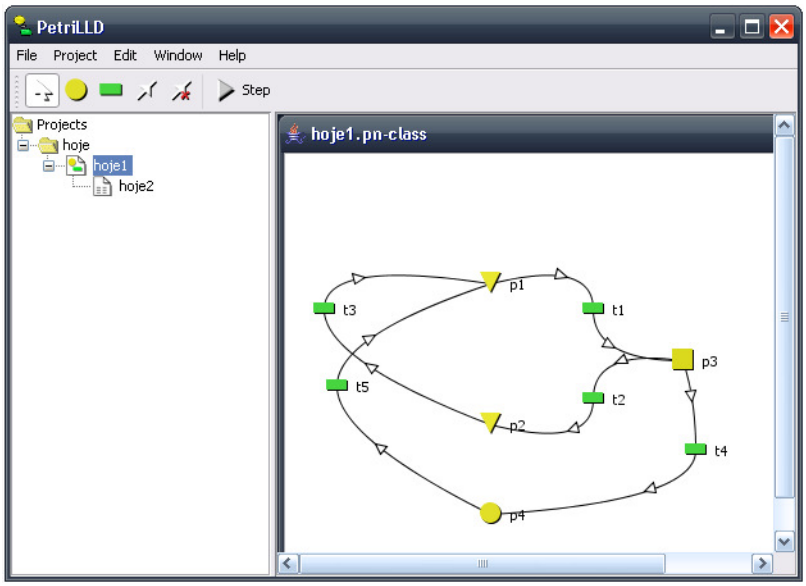

Figura 1.1: Interface gráfica da ferramenta PetriLLD

No contexto da edição, visualização e simulação destaca-se a ferramenta *Snoopy* [5] [42], desenvolvida pelo *Computer Science Department of Branenburg University of Technology Cottbus*. Dando continuidade ao trabalho iniciado, o projecto FORDESIGN [4][20] desenvolveu a versão *Snoopy-IOPT* [25] acrescentando a classe de redes IOPT. Esta versão permite fazer a exportação da rede para o formato PNML e está também prevista a interacção com diversas ferramentas. Na Figura 1.2 é possível ver a interface gráfica da ferramenta.

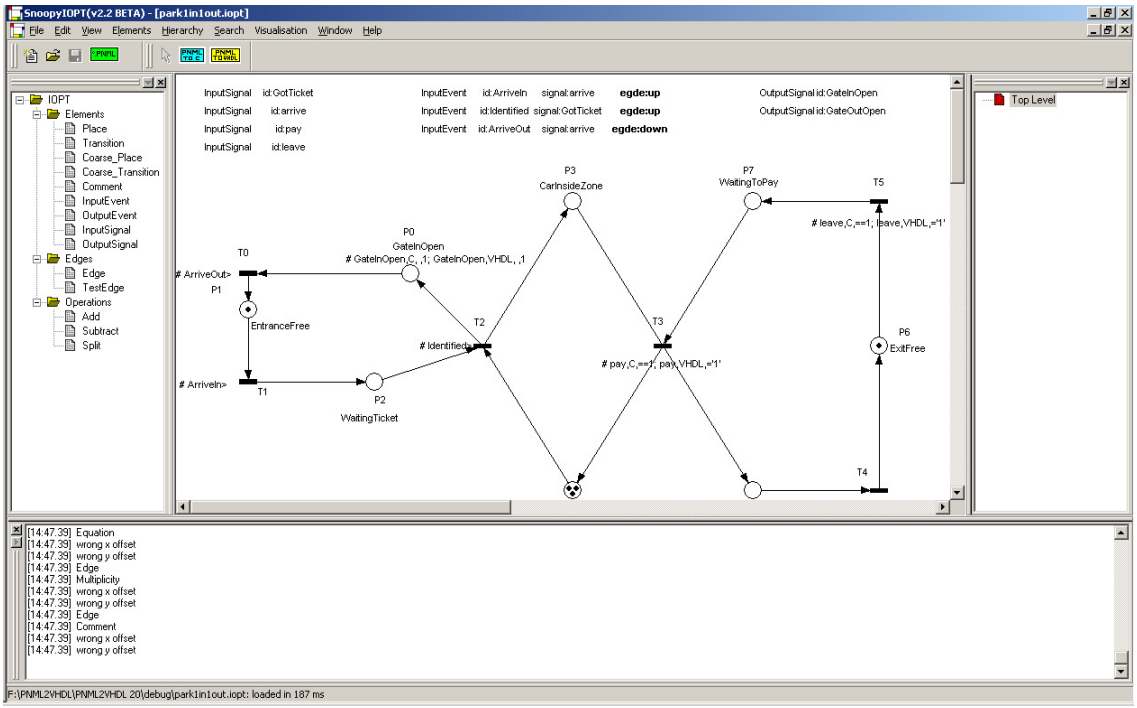

Figura 1.2: Interface gráfica da ferramenta Snoopy-IOPT

2. Referências teóricas

Este capítulo pretende fazer uma breve apresentação sobre as tecnologias utilizadas para representar as Redes de Petri e a linguagem VHDL para a qual será traduzida. São descritos alguns dos motivos que levaram ao seu uso como linguagem de descrição de hardware e são indicadas referências que apresentam os conceitos teóricos de redes de Petri e a forma como são descritas em PNML.

2.1. Redes de Petri

2.1.1. Introdução

As Redes de Petri (RdP) tiveram a sua origem no trabalho realizado por Carl Adam Petri na sua dissertação apresentada à Faculdade de Matemática e Física da Universidade Técnica de Darmstadt, na Alemanha em 1962. Com o seu trabalho, Carl Petri apresentou um tipo de grafo bipartido que continha estados associados com o intuito de estudar a comunicação entre autómatos [15]. Devido às suas características de modelação, as Redes de Petri foram tomando cada vez mais uma maior importância na modelação de sistemas, nomeadamente devido ao facto de permitirem a modelação de sincronização e concorrência entre processos, conflitos e partilha de recursos.

Uma das grandes vantagens do uso das RdP é o de possibilitarem uma representação para sistemas a eventos discretos permitindo a sua modelação, análise e simulação de uma forma prática e simples. Estes passos podem ser feitos através de ferramentas matemáticas ou então de uma forma gráfica, simplificando todo o processo para o desenvolvimento de sistemas a eventos discretos e permitindo desta forma uma visão da estrutura e comportamento do sistema. Em particular, a representação de eventos e condições, bem como as relações entre os dois, revelam-se como sendo características importantes de modelação para a utilização deste tipo de ferramenta para representar sistemas a eventos discretos. Segundo Peterson [16] (página 228), em cada estado do sistema podem verificar-se determinadas condições que possibilitam a ocorrência de eventos e que por sua vez podem ocasionar a mudança de estado do sistema.

Com o passar do tempo foram surgindo diversas variantes de Redes de Petri, cada uma apresentando diversas características que eram particularmente úteis para aplicações específicas, e donde se irá destacar neste capítulo a classe IOPT.

2.1.2. A estrutura da Rede – Lugares, Transições e Arcos

Uma Rede de Petri é constituída por dois grandes conjuntos de elementos: *lugares* e *transições*. Os lugares são normalmente representados por circunferências ou elipses, e as transições por quadrados ou rectângulos. Os lugares são ligados às transições e estas aos lugares através de arcos dirigidos. Para exemplo, na Figura 2.1, pode ser observado uma RdP.

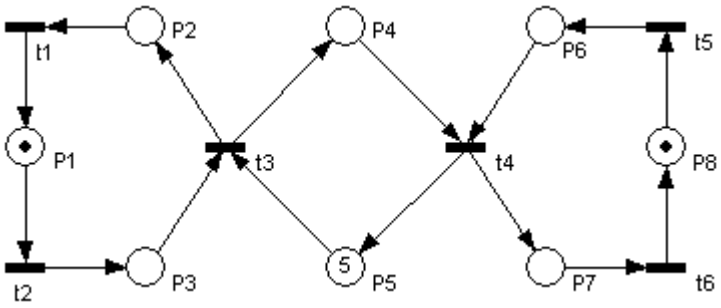

Figura 2.1: RdP que descreve um controlador de um parque de estacionamento

Considerando que as RdP são representações abstractas, os lugares e as transições podem ter interpretações distintas sendo no entanto, e de uma forma genérica, os lugares considerados como depósitos de recursos (representados por círculos pretos dentro dos lugares) e as transições como acções que manipulam esses recursos. Cada círculo tem o nome de *marca* (ou *token* em inglês).

A criação e/ou destruição de marcas é feita através de uma acção das transições, chamada de disparo, e que altera a marcação de um ou mais lugares fazendo evoluir a rede para um novo estado. Através dos arcos direccionados define-se quais são os lugares sobre os quais as transições vão actuar através da criação ou destruição de marcas. Não pode existir transições ligadas por arcos a outras transições, da mesma forma que os lugares apenas podem ser ligados a transições, não podendo estar ligados a outros lugares.

A quantidade de marcas presentes em cada lugar e que define o estado da rede num determinado instante é chamada de marcação [15].

Um arco com origem num lugar e fim numa transição (doravante designado por arco de entrada), indica que essa transição destrói, aquando do seu disparo, uma marca desse lugar. De forma simétrica, um arco com origem numa transição e fim num lugar (doravante designado por arco de saída), indica que essa transição cria, aquando do seu disparo, uma marca nesse lugar. Assim sendo, podemos pensar nos arcos como indicadores do sentido do movimento das marcas de um lugar para outro, atravessando a transição, permitindo associar uma animação gráfica ao disparo das transições. No entanto, formalmente o que se passa é que o disparo de uma transição faz, através de uma operação atómica, é destruir marcas nos seus lugares de entrada e criar marcas nos seus lugares de saída.

Uma transição diz-se *habilitada* quando pode disparar. Para que esse disparo aconteça é necessário que em cada lugar de entrada haja pelo menos uma marca (pré-condição).

É ainda possível associar pesos aos arcos, representativos da existência de múltiplos arcos entre os mesmos nós – Figura 2.2. Cada peso é representado através de um número inteiro, constituindo uma generalização do que foi dito sobre arcos anteriormente. Este tipo de rede é chamada como RdP generalizada (por oposição às RdP em que só existem arcos com peso unitário, designadas por RdP ordinárias).

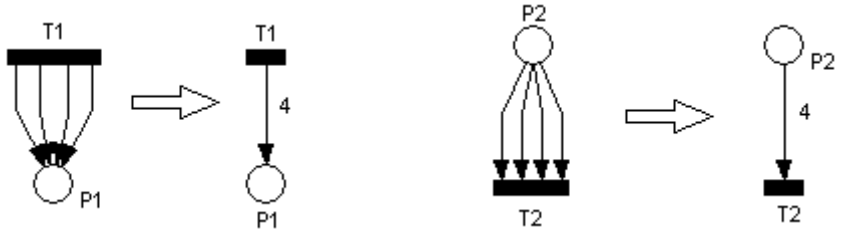

Figura 2.2: Pesos de arcos

Em [15] (página 543) a definição de Redes de Petri é feita considerando o conceito de pesos dos arcos e incluindo, desde o início, a sua marcação. Os conjuntos que definem os arcos, em vez de serem separados em arcos de entrada e arcos de saída, são definidos num único conjunto. De seguida mostra-se a forma como é definida uma RdP em [15] para a rede apresentada do controlador de um parque de estacionamento.

$R = (P, T, A, PA, M_0)$ onde:

$P = \{p_1, p_2, ..., p_m\}$	é um conjunto de lugares
$T = \{t_1, t_2, ..., t_n\}$	é um conjunto de transições
$P \cap T = \phi \wedge P \cup T \neq \phi$	os conjuntos P e T são disjuntos e não vazios
$A : (P \times T) \cup (T \times P)$	é um conjunto de arcos
$PA : A \rightarrow \mathrm{N}$	são os pesos dos arcos
$M_0 : L \rightarrow \mathrm{N}_0$	é a marcação inicial

Por exemplo, temos a seguinte representação da rede apresentada na Figura 2.1:

$$P = \{p_1, p_2, p_{3,} p_4, p_5, p_6, p_7, p_8\}$$
$$T = \{t_1, t_2, t_3, t_4, t_5, t_6\}$$
$$A = \begin{Bmatrix} (p_1, t_2), (p_2, t_1), (p_3, t_3), (p_4, t_3), (p_5, t_4), (p_6, t_4), (p_7, t_6), (p_8, t_5), \\ (t_1, p_1), (t_2, p_3), (t_3, p_2), (t_3, p_5), (t_4, p_4), (t_4, p_7), (t_6, p_8), (t_5, p_6) \end{Bmatrix}$$

$$\begin{array}{llll} PA((p_1, t_2)) = 1 & PA((p_5, t_4)) = 1 & PA((t_1, p_1)) = 1 & PA((t_4, p_4)) = 1 \\ PA((p_2, t_1)) = 1 & PA((p_6, t_4)) = 1 & PA((t_2, p_3)) = 1 & PA((t_4, p_7)) = 1 \\ PA((p_3, t_3)) = 1 & PA((p_7, t_6)) = 1 & PA((t_3, p_2)) = 1 & PA((t_6, p_8)) = 1 \\ PA((p_4, t_3)) = 1 & PA((p_8, t_5)) = 1 & PA((t_3, p_5)) = 1 & PA((t_5, p_6)) = 1 \end{array}$$

$$\begin{array}{llll} M_0(p_1) = 1 & M_0(p_3) = 0 & M_0(p_5) = 0 & M_0(p_7) = 0 \\ M_0(p_2) = 0 & M_0(p_4) = 5 & M_0(p_6) = 0 & M_0(p_8) = 1 \end{array}$$

Com esta definição de pesos nos arcos, uma transição apenas poderá ser habilitada se e só se todos os arcos de entrada da transição tiverem associados um peso menor ou igual que o número de marcas presentes nos respectivos lugares de origem. Formalmente:

$$(\forall (p,t) \in A) PA((p,t)) \leq M(p)$$

É ainda importante referir que, como as RdP não impõem uma ordem de ocorrência dos eventos, então se num determinado instante mais do que uma transição estiver habilitada a disparar, ela pode disparar. O estado seguinte (ou *passo*) é resultado do disparo de um qualquer conjunto de transições habilitadas.

2.1.3. Modelação com Redes de Petri

Sendo as Redes de Petri uma representação abstracta de um sistema, ao modelar situações concretas é necessário atribuir uma interpretação à rede de forma a obtermos uma representação da realidade e da forma como ela interage. Para a rede apresentada anteriormente, uma das possíveis interpretações é o facto de ela poder representar o controlador de um parque de estacionamento, interpretada da forma como se apresenta na Figura 2.3, e onde fica claro a associação de uma interpretação de cada nó do grafo.

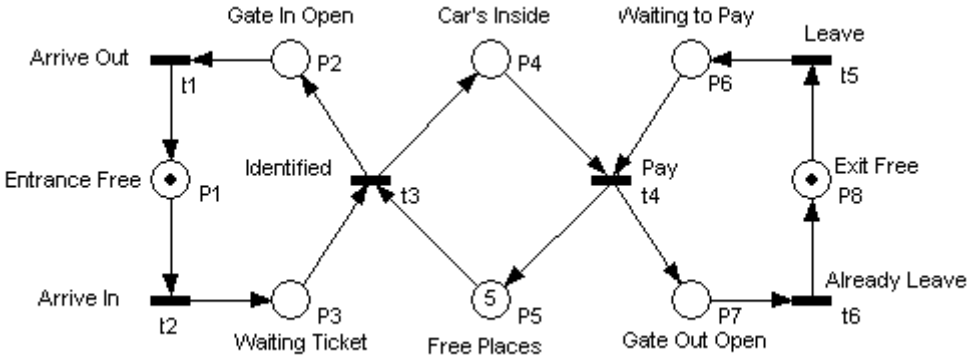

Figura 2.3: RdP interpretada

2.1.4. Propriedades das Redes de Petri

As propriedades das Redes de Petri podem ser classificadas como comportamentais ou estruturais em função de serem ou não dependentes da marcação inicial. Em [15] (páginas 547-569) e [19] pode ser obtido uma boa descrição das propriedades das Redes de Petri.

2.1.5. Redes de Petri não-autónomas

Uma Rede de Petri é considerada *autónoma* quando a sua evolução apenas depende de condicionalismos resultantes da sua estrutura e marcação. Assim, as RdP *não-autónomas* não dependem apenas delas para que haja evolução. Considere-se, por exemplo, o caso de RdP com temporizações associadas. Como o seu comportamento depende também de condições impostas pelas temporizações, então este tipo de redes são *não-autónomas*. Outro exemplo deste tipo de redes, é se considerarmos o disparo de uma ou mais transições dependente não apenas da marcação dos seus lugares de entrada mas também de uma condição externa

associada a estas, permitindo a sincronização da evolução da rede com eventos externos [7] (páginas 17 e 71-124). Estas RdP denominam-se *sincronizadas*, sendo uma das classes de RdP que serve de base a este trabalho.

2.1.6. A Classe IOPT (Input-Output Place Transition)

A classe de Redes de Petri *Input-Output Place-Transition* (*IOPT*) [4] permite ter uma definição de características que possibilitam a construção de modelos que representam controladores através da expansão das Redes Lugar-Transição [6].

É possível a modelação de sinais e eventos externos associados aos lugares e às transições. A cada lugar podem estar associadas expressões que representam condições e que permitem, quando a condição for verdadeira, a atribuição de valores aos sinais de saída em função de uma marcação específica. Por sua vez o disparo de uma transição pode estar dependente de eventos e sinais de entrada e pode gerar eventos de saída.

Permite também a especificação de um conjunto de prioridades entre as transições de forma a permitir a resolução de conflitos. As prioridades que são associadas às transições podem ser definidas através de árbitros específicos e que podem ser encontrados por exemplo em [11]. As transições envolvidas numa situação de conflito serão disparadas em função da sua prioridade em relação às outras transições envolvidas nesse mesmo conflito (contenda) ou, caso exista marcas suficientes para garantir os disparos, das transições prioritárias até que não existam mais marcas.

A classe IOPT permite ainda a atribuição de pesos aos arcos. Estes pesos deverão ser levados em consideração no momento do disparo da transição e na nova marcação de cada lugar após o disparo.

Nas próximas linhas é apresentada a definição formal da sintaxe e uma apresentação informal para a semântica respectiva de acordo com [4].

A. Definições básicas

Em seguida, apresenta-se a interface de sistema em relação às suas entradas e saídas, e a definição para a classe IOPT.

Definição 1 (interface de sistema): (de [4]) A interface de sistemas controlados através de redes IOPT é um tuplo *ICS* = (*IS, IE, OS, OE*) e que satisfaz os seguintes requisitos:

1) *IS é um conjunto finito de sinais de entrada*;
2) *IE é um conjunto finito de eventos de entrada*;
3) *OS é um conjunto finito de sinais de saída;*
4) *OE é um conjunto finito de eventos de saída;*
5) $IS \cap IE \cap OS \cap OE = \emptyset$.

Definição 2 (Estado das entradas do sistema): (de [4]) Dado um sistema controlado com uma interface *ICS* = (*IS, IE, OS, OE*) (Def. 1), um estado de entrada do sistema é definido pelo par *SIS* = (*ISB, IEB*) satisfazendo os seguintes requisitos:

1) *ISB* é um conjunto finito de sinais de entrada: $ISB \subseteq IS \times \mathrm{N}_0$
2) *IEB* é um conjunto finito de eventos de entrada: $IEB \subseteq IS \times B$

O conjunto de expressões booleanas é chamado *BE* e a função *Var(E)* retorna o conjunto das variáveis de uma dada expressão *E*.

Definição 3 (rede IOPT): (de [4]) Dado um controlador com uma interface *ICS* = (*IS, IE, OS, OE*) a rede IOPT é um tuplo *N* = (*P, T, A, TA, M, weight, weightTest, priority, isg, ie, oe, osc*) satisfazendo os seguintes requisitos:

1) *P* é um conjunto finito de lugares.
2) *T* é um conjunto finito de transições (separados de *P*).
3) *A* é um conjunto de arcos, tal que $A \subseteq ((P \times T) \cup (T \times P))$.
4) *TA* é um conjunto de arcos de teste, tal que $TA \subseteq (P \times T)$.
5) M é uma função de marcação: $M : P \to N_0$.
6) $weight : A \to N_0$.
7) $weightTest : TA \to N_0$.
8) *priority* é uma função parcial de inteiros não negativos aplicada a transições: $priority : T \to N_0$.

9) *isg* é um conjunto de expressões booleanas, associadas a sinais de entrada, e que são aplicadas como condições de habilitação a transições (onde todas as variáveis de entrada são sinais de entrada): $isg : T \rightarrow BE$, onde $\forall eb \in isg(T), Var(eb) \subseteq IS$.

10) *ie* é um conjunto de eventos de entrada, gerados a partir de sinais de entrada, e que representam condições de habilitação para transições: $ie : T \rightarrow IE$.

11) *oe* é um conjunto de eventos de saída, e com uma dependência de transições de forma a serem gerados: $oe : T \rightarrow OE$.

12) *osc* é um conjunto de sinais de saída, activados em função de uma série de condições associadas à marcação dos lugares (que permitem a activação do estado dos sinais de saída em função da marcação do lugar): $osc : P \rightarrow P(RULES)$, onde $RULES \subseteq (BESxOSxN_0), BES \subseteq BE$ e $\forall e \in BES(T), Var(e) \subseteq ML$ em que *ML* é o conjunto de identidades para cada marcação do lugar depois da execução de um passo: cada marcação de lugar tem um identificador associado, que é usado quando executado o código gerado.

Ainda de salientar os seguintes aspectos:

- O disparo de transições pode gerar eventos de saída (*oe*);
- O estado da activação de sinais de saída (*osc*) pode ser determinado pelas marcações de lugares;
- O disparo de cada transição pode depender de prioridades (*priority*), das condições que usam sinais externos (*isg*) e dos eventos de entrada (*ie*), bem como gerar eventos de saída (*oe*).

Sempre que uma transição é habilitada, e a condição externa associada é verdadeira (o evento da entrada e a condição do sinal de entrada são ambos verdadeiros), a transição é disparada. A evolução da rede depende também de instantes específicos no tempo chamados *tics* e que são definidos por um ciclo do relógio externo, tal como é comum para as redes não-autónomas. Um passo da execução é o período entre dois *tics*.

2.1.7. Contendas

2.1.7.1. Introdução

Uma *contenda* é um conjunto de transições que têm um ou vários conflitos estruturais e às quais estão associadas prioridades. Cada prioridade é única na contenda e definida por um número inteiro positivo que indica a ordem de disparo das transições pertencentes a uma contenda. A sua definição tem origem na classe *conflict* apresentada em [13] (página 673).

Por sua vez, uma transição que não se encontre em conflito constitui também uma contenda ou *contenda unitária*. O conjunto das contendas de uma RdP designa-se por CO e define-se como [24] (página 63):

$$CO = \left\{ C \mid \left(C \in 2^{T} \right) \wedge \left(\forall \mathrm{t1} \in C \ \exists \mathrm{t2} \in \left(C - \mathrm{t1} \right) : {}^{\bullet}\mathrm{t1} \cap {}^{\bullet}\mathrm{t2} \neq \phi \right) \right\}$$

Numa contenda é possível atribuir a cada transição um valor de prioridade. A ausência deste é equivalente à especificação de um valor 1. A prioridade entre transições apenas tem significado quando estas pertençam à mesma contenda.

2.1.7.2. Representação e disparo de contendas

Quando existe um lugar de entrada em comum para um certo número de transições, estas são agrupadas num mesmo objecto *conflict*. Este objecto representa assim uma contenda. Na Figura 2.4 apresenta-se uma Rede de Petri onde se mostram os agrupamentos de transições que representam contendas ou agrupamentos *conflict.*

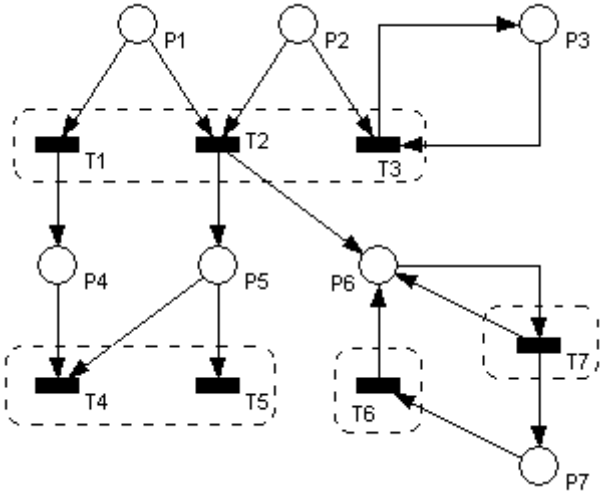

Figura 2.4: Exemplos de contendas

Baseado em [24], na Figura 2.4 existem quatro agrupamentos de transições em objectos *conflict* constituindo assim quatro contendas. Estes correspondem a conjuntos de transições que partilham directa (*T1* e *T2*) ou indirectamente (*T1* e *T3*) lugares de entrada. Mesmo quando não existe partilha de lugares de entrada considera-se um objecto com uma única transição:

- Uma constituída pelas transições *T1*, *T2* e *T3*;
- Uma unitária constituída pela transição *T7*;
- Uma constituída pelas transições *T4* e *T5*;
- E uma também unitária constituída pela transição *T6*.

Cada objecto *conflict* contém uma lista de referências para transições. Quando as transições da contenda têm prioridades associadas, essa lista encontra-se ordenada por ordem decrescente. Desta forma, e aquando do disparo da contenda, basta percorrer a sequência tentando disparar a primeira transição que se encontre apta. Caso não haja marcas suficientes para disparar uma transição, disparam as seguintes se o número de marcas o permitir, tendo sempre em consideração a sequência de disparo definida. Apesar de poderem existir várias sequências possíveis, apenas pode existir uma ordenação correcta fazendo corresponder uma única sequência de disparo.

Quando não existem prioridades associadas a ordem é irrelevante e a escolha da transição é feita aleatoriamente ou através de um outro método não equitativo.

2.2. PNML

2.2.1. Introdução

Devido ao crescente desenvolvimento de ferramentas que suportam Redes de Petri surgiu a necessidade de se conseguir exportá-las/importá-las garantindo a interoperabilidade entre ferramentas. Um dos problemas iniciais para se conseguir isso é a necessidade de existir um formato comum entre essas aplicações, sendo para além disso necessário garantir suporte para as diversas classes de Redes de Petri existentes considerando que cada uma é adequada para um certo fim e nenhuma pode ser colocada de lado. Para que se consiga tal objectivo é necessário existir um formato de transferência de informação entre aplicações universal que garanta essa transferência, guardando todos os dados relevantes à(s) classe(s).

Desta forma surge a *Petri Net Markup Language* (PNML) [23] que é um formato de representação de Redes de Petri baseado em XML (*Extensive Markup Language*) [36]. O objectivo da sua criação é o de tentar encontrar um padrão para representar as redes de Petri através da definição de um formato de arquivo que suporte todos os tipos de classes de redes de Petri.

Os princípios que levaram ao aparecimento do PNML podem ser encontrados em [23], dos quais se destacam os seguintes:

- O formato a adaptar deverá ser facilmente legível por pessoas e alterável através de um editor de texto convencional;
- Deve ser possível representar qualquer tipo de Rede de Petri;
- Deve fornecer tanta informação quanto possível da rede, mesmo que o seu tipo seja desconhecido.

Consegue-se assim ter um formato suficientemente genérico de forma a dar suporte para o *software* já existente e para o que virá a ser desenvolvido, garantindo a possibilidade de usar qualquer tipo de classe de Redes de Petri. As características específicas de cada classe de redes são definidas através de ficheiros "*Petri Net Type Definition Files*" (PNTD) contendo neles todas as informações importantes.

As inúmeras características existentes encontram-se definidas no PNML como anotações "*labels*". Da informação a guardar deve estar incluído todas as informações necessárias para o armazenamento da rede básica (transições, lugares, arcos e marcas). Poderá também armazenar informações referentes ao *layout* do sistema para apresentação gráfica através da definição de coordenadas *x* e *y* do plano cartesiano.

2.2.2. Representação de Redes de Petri usando PNML

Cada RdP num ficheiro PNML é constituída por objectos que representam a estrutura do grafo da rede. Para que um objecto possa ter significado, é adicionado um *label* e que tem a função de representar, por exemplo, o nome de um nó, a marcação inicial, condições de habilitação das transições, etc. No entanto, em função do tipo de rede utilizado (definido na PNTD), uma *label* pode ser de dois tipos:

- Anotações, que contém informação de texto, como por exemplo, a marcação inicial, nomes, pesos dos arcos, comentários,…
- Atributos, que contém informação sobre propriedades gráficas do objecto, por exemplo, atributos x e y para representar a rede.

Todos os objectos e anotações contêm informação gráfica. Um nó inclui informação sobre a sua posição, um arco contém uma lista de posições (que definem pontos intermediários do arco) e uma anotação tem a sua posição em relação aos seus objectos correspondentes. Pode existir informação adicional relacionada com o tamanho, cor e forma de nós e arcos, ou relacionada com cor, fonte e tamanho de fonte de *labels*.

Para algumas ferramentas, pode ser necessário armazenar também a informação específica da ferramenta. O seu formato depende da ferramenta e não é especificado pelo PNML. O PNML fornece um mecanismo para marcação clara da informação específica da ferramenta, contendo o nome e a versão da ferramenta. Além disso outras ferramentas podem facilmente ignorar e adicionar informação específica da ferramenta que nunca irá comprometer um ficheiro de RdP.

2.2.3. Representação de Redes de Petri IOPT usando PNML

De seguida pretende-se descrever resumidamente a gramática Relax NG [21] baseada na definição do tipo de redes de Petri lugar-transição [6]. Mais informação pode ser encontrada em [4] e o ficheiro completo está disponível em [35].

O ficheiro PNML que representa o modelo RdP do sistema deverá conter toda a informação associada à caracterização de entradas e saídas do controlador, bem como a caracterização do seu comportamento. Além da informação base (lugares, transições, e arcos), a gramática define ainda informação adicional sobre a rede, nomeadamente *NetIOInput*, *NetIOOutput*, *ConflictSet* e *SynchronySet*:

- *NetIOInput*: caracteriza toda a informação essencial sobre os sinais e eventos de entrada. Estão definidos actualmente dois tipos de sinais: *boolean* e *range*. O primeiro apenas suporta valores a verdadeiro (1) ou a falso (0) enquanto que os *range* permitem definir valores que podem variar num determinado intervalo, em função dos limites máximo e minimos definidos. Em relação aos eventos (*input*), estes são sempre do tipo *boolean* e têm que ter um *ID* de um sinal associado. É também necessário especificar o sentido do flanco (ascendente ou descendente) de forma a que em função da variação do sinal associado seja gerado o evento. No caso do sinal ser do tipo *range* é ainda possível especificar um nível de *threshold* para a geração do evento.
- *NetIOOutput*: contém toda a informação essencial sobre os sinais e eventos de saída. Da mesma forma que o ponto anterior, admite sinais do tipo *boolean* e do tipo *range*. Em relação aos eventos (*output*) regem-se essencialmente pelas mesmas regras do ponto anterior.
- *ConflictSet* permite identificar um conjunto de transições que estão envolvidas num conflito estrutural. É então possível definir um conjunto de prioridades para que a situação de conflito possa ser resolvida.
- *SynchronySet*, baseado no que é dito em [22], permite identificar um conjunto de transições interligadas através de um canal síncrono, provocando uma fusão para que estas se comportem como uma única transição (fora do âmbito deste trabalho).

Foram ainda adicionados novos atributos aos elementos principais do PNML (lugares, transições, e arcos). Em relação aos lugares, foram considerados os seguintes:

- *Bound*: define a marcação máxima alcançável num lugar, considerando uma determinada marcação inicial;
- *SignalOutputAction*: permite definir as dependências em termos dos sinais de saída do lugar, isto é, as saídas activadas pela marcação do lugar;
- *subNet*: permite representações hierárquicas, e que está fora do âmbito do trabalho apresentado neste documento.

Em relação às transições:

- *NetIOTransGuard*: permite a representação de condições da transição;
- *NetIOTransInput*: permite especificar as dependências da transição em relação aos eventos de entrada;
- *NetIOTransOutput*: permite especificar os eventos de saída que devem ser gerados quando existe um disparo da transição;
- *Priority*: permite a especificação de uma prioridade associada à transição dentro de um determinado *ConflictSet*;
- *subNet*: relacionado com as representações hierárquicas, não sendo usado no âmbito desta dissertação.

Em relação aos arcos:

- *PTArcInscription*: permite a especificação de pesos associados aos arcos;
- *ArcType*: permite definir o tipo de arco. Actualmente são considerados dois tipos de arcos: normal ou de teste. Nos arcos normais, em função do peso do arco, é destruído um número de marcas do lugar de saída e criado o mesmo número no lugar de entrada. Nos arcos de teste não existe destruição de marcas do lugar de saída;
- *subSource*: relacionado com as representações hierárquicas, não sendo usado no âmbito desta dissertação;
- *subTarget*: relacionado com as representações hierárquicas, não sendo usado no âmbito desta dissertação.

2.3. VHDL

Com esta secção pretende-se introduzir alguns conceitos sobre a linguagem de descrição de hardware VHDL, em particular o motivo pelo qual foi escolhida para a tradução de Redes de Petri. O capítulo está dividido em seis secções onde se apresentam os objectivos do seu uso, funcionalidades, vantagens e alguns conceitos teóricos.

Para um conhecimento mais aprofundado desta linguagem recomenda-se a referência [2].

2.3.1. Introdução

O VHDL (*VHSIC Hardware Description Language*) é uma linguagem de descrição de hardware tendo sido concebida para desenvolver circuitos integrados. Na realidade a linguagem VHDL permite descrever vários tipos de sistemas tais como uma rede de computadores, um circuito integrado ou simplesmente uma porta lógica.

A sua origem vem de um programa do Departamento de Defesa dos Estados Unidos da América em 1980. O objectivo do programa era o de promover uma norma para descrever a estrutura e função de Circuitos Integrados bem como facilitar a construção de sistemas digitais como resposta ao aumento da complexidade crescente associado ao seu desenvolvimento. Posteriormente a linguagem VHDL foi aperfeiçoada e tornou-se uma norma em 1987 aprovada pelo *Institute of Electrical and Electronics Engineers* (IEEE) nos EUA [43].

Existem duas grandes versões da norma emitidas pelo *IEEE*: uma de 1987, o IEEE Std 1076 – 1987 (chamada também de "VHDL 1987"), e a outra de 1993, o IEEE Std 1076 – 1993 (chamada também de "VHDL 1993"). Com a necessidade de definir um modelo para os valores lógicos possíveis surge também a norma IEEE Std. 1164-1993 e que juntamente com o IEEE Std. 1076-1993 são as normas mais utilizados actualmente. Nos trabalhos realizados no âmbito desta dissertação estas foram as normas consideradas.

Apesar de existirem outras linguagens de descrição de hardware (*Verilog*, *System C*, *SDL*, *Handel-C*,…), o VHDL apresenta-se como uma boa escolha pois é a mais reconhecida e utilizada mundialmente por empresas da área de engenharia (como por exemplo a Xilinx [32] e a Altera [33]) a par do *Verilog*.

2.3.2. Vantagens do uso do VHDL

As principais características presentes na linguagem VHDL que levaram à sua escolha para a tradução das RdP são as seguintes:

- Permitir uma descrição estruturada, pois um projecto pode ser composto por vários sub-projectos que podem ser interligados;
- Permitir a especificação de funções utilizando formas similares de linguagens de programação;
- Simulação de um projecto antes da fabricação de um circuito integrado (ASIC), ou da configuração de um circuito lógico programável (FPGA's).

Em relação ao uso de diagramas esquemáticos, esta apresenta as seguintes vantagens:

- Nos sistemas sequenciais, o detalhe da lógica de controlo é realizado pelas ferramentas de automação do projecto, o que evita a aplicação das técnicas manuais de implementação;
- O próprio objectivo do projecto fica mais claro em comparação com a representação por esquemáticos;
- O volume de documentação diminui, já que um código bem comentado em VHDL substitui com vantagens o esquemático e a descrição funcional do sistema;
- O projecto ganha portabilidade, podendo este ser compilado em qualquer ferramenta e para qualquer tecnologia. Por norma, é frequente o uso de FPGA's e CPLD's para produções iniciais ou de menores escalas em projectos e que posteriormente possam ser implementados em ASIC's. Todas as implementações podem usar o mesmo código VHDL;
- É independente da tecnologia de implementação;
- Descrição mais abstracta e focada cada vez mais no nível comportamental e menos na implementação física.

2.3.3. Objectivos do seu uso

O uso de uma linguagem de descrição de hardware permite uma eficiente documentação e modelação do sistema, garantindo uma precisão de excelência e portabilidade deste para diversas arquitecturas. Através da modelação pode-se ainda validar todo o desenvolvimento do projecto através da sua simulação sem custos elevados de prototipagem. De seguida apresentam-se os motivos para o seu uso:

- Permite ter projectos independentes da tecnologia tornando-os mais flexíveis devido ao facto de a maior parte das ferramentas reconhecerem a linguagem;
- Fornece uma maior facilidade de actualização de projectos;
- Permite explorar o projecto a um nível mais alto de abstracção;
- Através da simulação, verificar o comportamento do futuro sistema digital;
- Reduzir o tempo de projecto;
- Normalização, pois o VHDL é uma linguagem reconhecida pelo IEEE;
- Melhor legibilidade de um projecto. Possibilidade de particionar um projecto mais facilmente, desacoplando os seus blocos;
- Utilização de parâmetros que modificam a capacidade e desempenho de um projecto ou bloco;
- Redução de custos na fabricação de protótipos. Redução do tempo de inserção de um novo produto no mercado;
- Compativel com diversas ferramentas de CAD.

2.3.4. Estrutura da linguagem

Uma descrição VHDL é constituída por uma entidade (*entity*) e uma arquitectura (*architecture*). A entidade tem a função de descrever a interface do sistema enquanto que a arquitectura descreve o comportamento do sistema.

Numa descrição em VHDL cada módulo irá ter a sua entidade e arquitectura. As arquitecturas podem ser descritas a um nível comportamental, estrutural ou através de uma mistura de ambas de forma a definir as relações entre as entradas e as saídas. A comunicação com o exterior do módulo é realizada pelas portas definidas na entidade, onde se indica o tipo de

porta, a sua direcção (entrada, saída ou entrada/saída), o tamanho (barramento) e outras características relevantes [2].

O nível Comportamental permite representar a funcionalidade do sistema, sem preocupações de arquitectura ou de dependências temporais. De uma forma simples e abstracta permite descrever o que o sistema deve fazer sem ter qualquer tipo de ligação com a tecnologia que será usada na implementação. Esta é a forma mais flexível e poderosa de descrição.

O nível Estrutural permite por sua vez descrever o sistema através da sua estrutura, em termos da interligação de componentes. O circuito é construído através dos seus componentes e das ligações entre eles. Neste caso existe um aumento da complexidade na elaboração do modelo do sistema digital, podendo tornar-se pouco perceptível caso a complexidade do circuito seja média-alta.

A maioria dos sistemas digitais usa uma mistura das duas descrições, havendo um grande complemento entre elas, principalmente quando o circuito é de alguma complexidade.

2.3.5. Conclusão

Nesta secção foram apresentados os motivos que levaram ao uso de VHDL para descrever Redes de Petri. Foram abordadas algumas das vantagens do seu uso bem como alguns aspectos formais da linguagem que se mostram importantes no contexto deste trabalho.

O facto de esta linguagem permitir implementar sistemas digitais complexos e de forma simples e rápida torna-a bastante apelativa. Associado está o facto de os projectos poderem ser independentes da tecnologia utilizada, dando uma importante portabilidade ao projecto, e de ser possível a sua simulação antes da implementação.

3. Proposta de um conjunto de regras de tradução de redes IOPT para VHDL

Este capítulo pretende descrever as regras de tradução propostas que permitem passar de uma rede de Petri descrita através da classe IOPT em PNML para a linguagem de descrição de hardware VHDL. De forma a garantir portabilidade e uma adequada execução, optou-se pela implementação síncrona. São descritos os objectivos de cada regra e a forma como são obtidos os dados para a tradução a partir do PNML.

O capítulo encontra-se dividido em sete secções. Cada uma pretende apresentar as regras para cada característica da classe IOPT e são as seguintes: Sinais de Entrada, Eventos de Entrada, Arcos, Transições, Lugares, Eventos de Saída e Sinais de Saída.

3.1. Sinais de entrada

Esta secção pretende apresentar a regra de tradução proposta para Sinais de Entrada. A secção está dividida em duas sub-secções: a primeira apresenta a forma como são declarados os sinais na *entity* de um projecto em VHDL e a segunda sub-secção apresenta a forma como são sincronizados com o relógio global do sistema de forma a evitar situações de meta-estabilidade.

3.1.1. Tradução de sinais de entrada

A classe IOPT permite a representação de sinais externos de entrada que representam condições para habilitação de transições ou condições associadas a lugares. Estão definidos dois tipos de sinais de entrada:

- Sinais binários (*boolean*): sinais constituídos por um bit. A sua declaração em VHDL pode ser realizada por um sinal do tipo *std_logic*. De forma a uniformizar a representação de sinais, e considerando que declarar um sinal *std_logic* é equivalente a declarar um sinal *std_logic_vector (0 downto 0)* não havendo qualquer tipo de diferença na implementação em hardware tal como referido na

página 133 de [2], então a declaração na interface do ficheiro VHDL a gerar será feita através da seguinte regra onde *InputSignal* representa o nome do sinal:

```
InputSignal : in std_logic_vector(0 downto 0);
```

O seguinte excerto PNML mostra a forma como a situação anterior estará declarada em PNML:

```
<input>
  ...
   <signal id="InputSignal" type="boolean">
   </signal>
  ...
</input>
```

- Sinais com mais de um bit (*range*): São declarados em VHDL através de um vector *std_logic_vector* em que a sua dimensão depende do valor máximo e mínimo indicados em decimal pelo projectista. A declaração no ficheiro PNML tem o seguinte aspecto (onde *InputSignal* representa o nome do sinal, *MaxValue* e *MinValue* o valor máximo e mínimo respectivamente em decimal possível para o sinal indicados pelo projectista):

```
<input>
  ...
   <signal id="InputSignal" max="MaxValue" min="MinValue"
type="range">
   </signal>
  ...
</input>
```

É necessário considerar que *MaxValue* deverá sempre ser maior ou igual a *MinValue*. Assim, para encontrar o número de bits necessários para declarar o sinal de entrada na interface usa-se o seguinte método:

$$2^N - 1 \geq MaxValue \Leftrightarrow N \geq \frac{\log(MaxValue + 1)}{\log(2)}$$

Onde o número inteiro *N*, arredondado para cima, representa o número mínimo de bits necessários à declaração do sinal de entrada.

Surge assim a condição que será adicionada à interface do ficheiro VHDL para a representação de um sinal de entrada com mais de um bit:

```
InputSignal : in std_logic_vector((N-1) downto 0);
```

3.1.2. Sincronização dos sinais de entrada com o relógio global do sistema

Devido à possibilidade de ocorrência de situações de meta-estabilidade [3], todos os sinais de entrada passam por um registo de forma a anular os problemas associados à sua ocorrência.

Este motivo leva à necessidade de criar um número equivalente de sinais internos ao ficheiro VHDL e idênticos aos sinais de entrada. O nome desses novos sinais será constituído pelo nome original do sinal acrescentando-se a sigla "Sync" indicando que é o sinal sincronizado com o relógio global do sistema. Estes serão os sinais usados futuramente em cada instante a que se esteja a referir aos sinais de entrada.

De seguida é apresentada a estrutura completa da declaração e sincronização de *n* sinais de entrada representados por *InputSignal1*, *IinputSignal2*, …, *InputSignaln* (constituídos por *N1*, *N2*, …, *Nn* bits respectivamente) e que dão origem aos sinais *InputSignal1Sync*, *InputSignal2Sync*, …, *InputSignalnSync* respectivamente (constituídos por *N1*, *N2*, …, *Nn* bits respectivamente) após a passagem pelo registo:

```
entity NetName is
    Port    (

            CLK : in std_logic;
            RESET : in std_logic;


            -- Sinais de entrada assincronos
            InputSignal1 : in std_logic_vector((N1-1) downto 0);
            InputSignal2 : in std_logic_vector((N2-1) downto 0);
            ...
            InputSignaln : in std_logic_vector((Nn-1) downto 0);

            ...
            );
end NetName;

architecture Behavioral of NetName is

...

-- Declaração dos Sinais sincronizados
signal InputSignal1Sync : std_logic_vector((N1-1) downto 0);
```

```
signal InputSignal2Sync : std_logic_vector((N2-1) downto 0);
...
signal InputSignalnSync : std_logic_vector((Nn-1) downto 0);

...

begin

-- Registo
process (CLK)
begin
      if (CLK'event and CLK='1') then
            InputSignal1Sync <= InputSignal1;
            InputSignal2Sync <= InputSignal2;
            ...
            InputSignalnSync <= InputSignaln;
      end if;
end process;

...

end Behavioral;
```

3.2. Eventos de Entrada

Esta secção pretende apresentar a regra de tradução proposta para Eventos de Entrada. Está dividida em cinco sub-secções: a primeira apresenta uma introdução onde se pode encontrar informação comum para as regras que são apresentadas; a segunda sub-secção apresenta a regra proposta para tradução de eventos associados a sinais binários com flanco ascendente e descendente; a terceira sub-secção apresenta a regra para tradução de eventos associados a sinais binários com ciclos ascendente-descendente e descendente-ascendente; a quarta sub-secção apresenta a regra proposta para tradução de eventos associados a sinais não binários com flanco ascendente e descendente; finalmente, a quinta sub-secção apresenta a regra para tradução de eventos associados a sinais não binários com ciclos ascendente-descendente e descendente-ascendente.

3.2.1. Introdução

O código PNML que representa um evento numa rede IOPT pode ser observado de seguida. Cada evento deverá ter associado um flanco (*edge*), um nome (*Id*), um valor de referência (*level*) e o nome do sinal (*signal*) a que irá estar associado.Encontram-se representados de seguida os vários tipos de eventos possíveis, dependendo da dependência em termos de flanco.

```
<input>
  ...
   <event edge="up" id="EvInUp" level="Level" signal="InputSignal">
   </event>
   <event edge="down" id="EvInDown"
                                    level="Level" signal="InputSignal">
   </event>
   <event edge="up-down" id="EvInUpDown"
                                    level="Level" signal="InputSignal">
   </event>
   <event edge="down-up" id="EvInDownUp"
                                    level="Level" signal="InputSignal">
   </event>
  ...
</input>
```

No caso dos sinais binários, o valor associado ao *level* irá ser sempre igual a zero (pois para os sinais de entrada binários, o valor mínimo e máximo para o valor de referência é 0 e 1 respectivamente). Para todos os sinais não binários o valor de *level* (ou seja, *Level*) tem de estar compreendido entre os valores máximo e mínimo definidos (em decimal) para o sinal em causa.

Através do PNML apresentado observa-se que se pode representar um evento ascendente (*up*), um descendente (*down*), um ciclo ascendente-descendente (*Up-Down*) e um ciclo descendente-ascendente (*Down-Up*) do sinal *InputSignal* que estará representado internamente no ficheiro VHDL pelo sinal *InputSignalSync* sincronizado com o relógio do sistema (secção 3.1.2):

```
signal InputSignalSync : std_logic_vector(N-1 downto 0);
```

Em que N representa o número de bits do sinal em causa.

O nome a atribuir ao sinal que representa o evento será obtido através do seu *ID*, presente no ficheiro PNML. Todos os eventos gerados serão representados por sinais binários sendo portanto declarados como sinais do tipo *std_logic* (tipo *std_logic_vector* de dimensão um).

3.2.2. Eventos de flanco ascendente (*up*) e descendente (*down*) de sinais binários

A detecção de eventos de entrada ascendentes (*up*) e descendentes (*down*) de sinais de entrada binários é realizada considerando a mudança de estado dos sinais de entrada após a passagem pelo registo para sincronização com o relógio do sistema.

O esquema da Figura 3.1 mostra a implementação de um módulo de interface capaz de detectar acontecimentos de eventos ascendentes e/ou descendentes de sinais binários:

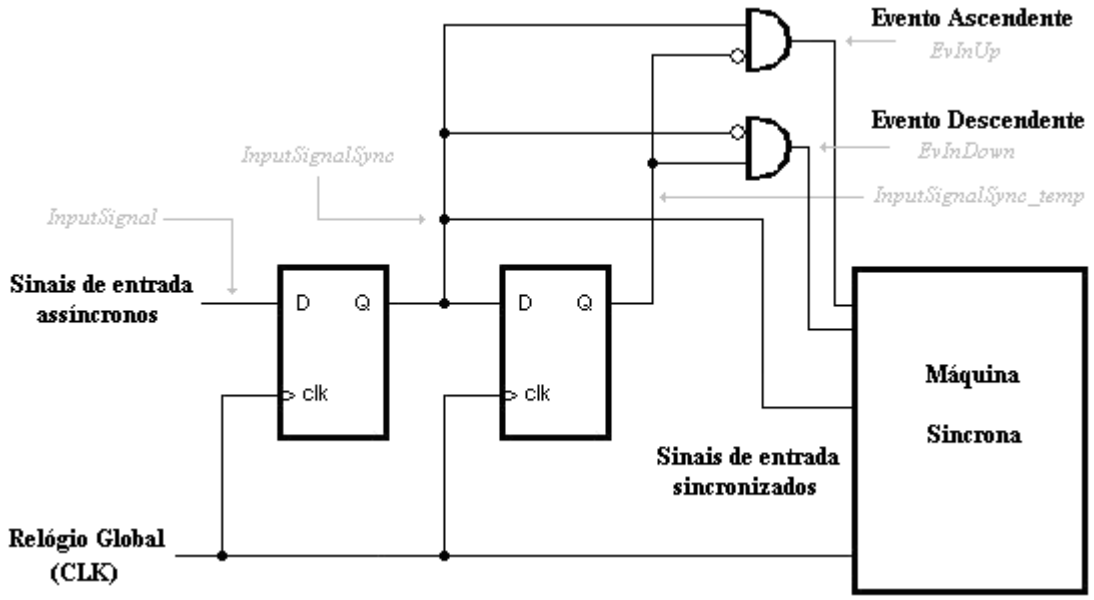

Figura 3.1: Circuito para geração de eventos *up* e *down* de sinais de um bit

Para a representação em VHDL do circuito apresentado irá ser necessário declarar internamente o sinal que vai representar o evento e um sinal temporário, *InputSignalSync_temp*, de forma a guardar o sinal de entrada *InputSignal* atrasado um ciclo de relógio (representado pelo nome do sinal ao qual se acrescenta "_temp"):

```
signal InputSignalSync_temp : std_logic_vector(0 downto 0);
signal EvInUp, EvInDown : std_logic;
```

Onde *EvInUp* e *EvInDown* representam respectivamente os eventos ascendente e descendente do sinal em causa.

Através da especificação em VHDL do circuito tem-se a seguinte regra para a detecção dos eventos de entrada ascendentes e descendentes de sinais binários:

Registo para guardar o sinal atrasado em um ciclo de clock

```
process (CLK)
begin
      if (CLK'event and CLK='1') then
            InputSignalSync_temp <= InputSignalSync;
      end if;
end process;
```

Geração dos eventos Up e Down

```
-- Flanco Ascendente
EvInUp <= '1' when (InputSignalSync="1" and InputSignalSync_temp="0")
            else '0';

-- Flanco Descendente
EvInDown <= '1' when
                  (InputSignalSync="0" and InputSignalSync_temp="1")
                  else '0';
```

3.2.3. Eventos constituídos por ciclos ascendente-descendente (*Up-Down*) e descendente-ascendente (*Down-Up*) de sinais binários

A identificação de acontecimentos de eventos constituídos por ciclos ascendente-descendente (*Up-Down*) e descendente-ascendente (*Down-Up*) associados a sinais binários serão obtidos a partir do código dos eventos ascendentes (*up*) e descendentes (*down*) de sinais binários.

Para a identificação deste tipo de eventos irá ser usada uma máquina de estados com dois estados cada.

No caso do evento constituído por um ciclo ascendente-descendente (*Up-Down*) a máquina de estados pode ser observada na Figura 3.2.

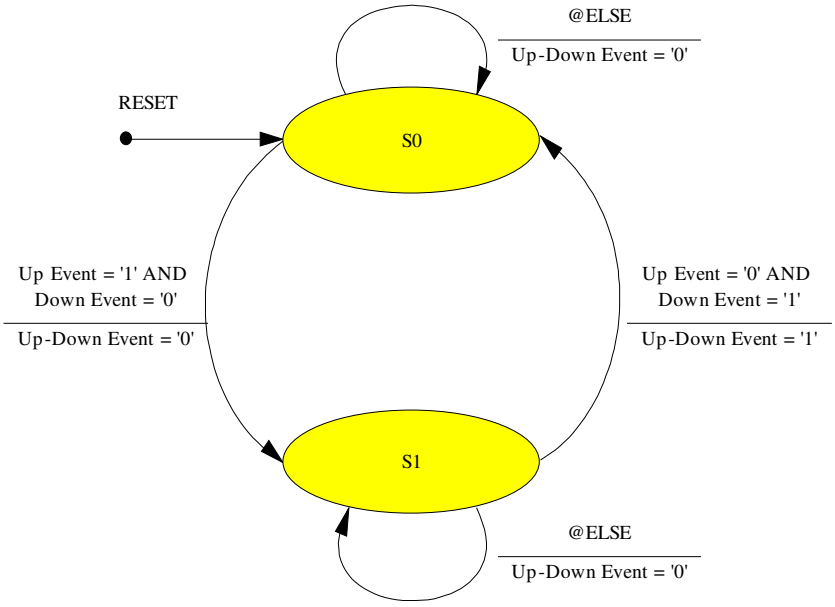

Figura 3.2: Máquina de estados para identificar eventos constituídos por um ciclo ascendente-descendente (*Up-Down*) de sinais binários

Para a detecção do evento constituído por um ciclo descendente-ascendente (*Down-Up*) a máquina de estados encontra-se na Figura 3.3.

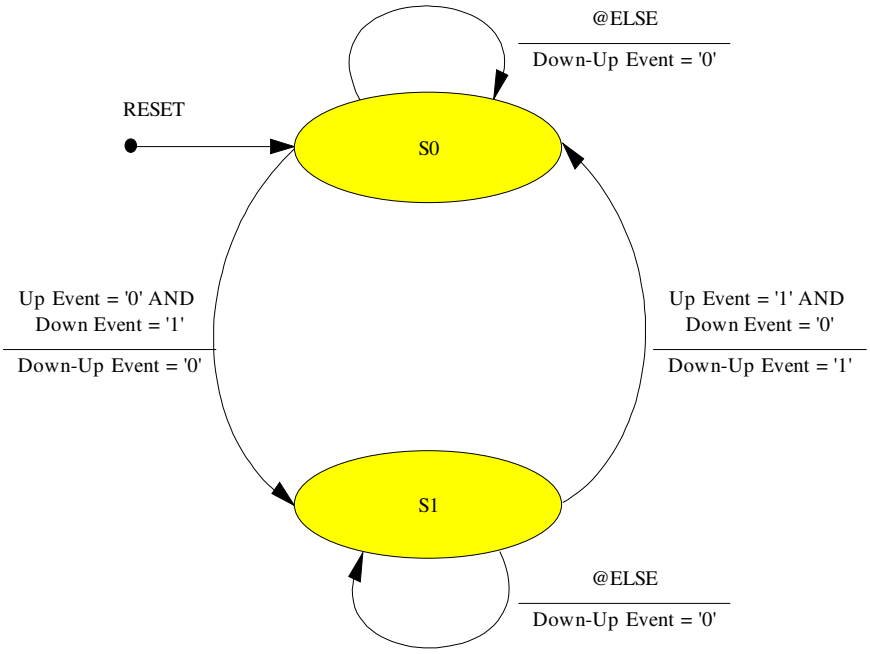

Figura 3.3: Máquina de estados para eventos constituidos por ciclos descendente-ascendente (*Down-Up*) de sinais binários

Para a detecção deste tipo de eventos em VHDL e para a constituição das máquinas de estado apresentadas é necessário declarar os seguintes sinais:

```
-- Ascendente-descendente (UP-DOWN):
signal EvInUpDown : std_logic;

-- Descendente-Ascendente (DOWN-UP):
signal EvInDownUp : std_logic;

-- Declaração dos estados da máquina de estados
signal StateUD : states;
signal StateDU : states;
```

Onde *EvInUpDown* e *EvInDownUp* representam os sinais dos eventos constituídos por ciclos ascendente-descendente (*Up-Down*) e descendente-ascendente (*Down-Up*), respectivamente. O sinal *StateUD* e *StateDU* representam os estados possíveis das máquinas de estados. Estes serão apenas dois e são declarados através da seguinte linha de código:

```
type states is (S0, S1);
```

A máquina de estados em VHDL, para o evento constituído por um ciclo ascendente-descendente (*Up-Down*), é apresentada de seguida (considerando [2], páginas 205 e 206):

Geração do evento constituído por um ciclo ascendente-descendente (Up-Down) de sinais binários:

```
process (CLK, RESET)
begin
if (RESET = '1') then StateUD <=S0;
elsif (CLK'event and CLK='1') then
    case StateUD is
       when S0 => if (EvInUp='1' and EvInDown='0')
                         then StateUD <= S1;
                  end if;
       when S1 => if (EvInUp='0' and EvInDown='1')
                         then StateUD <= S0;
                  end if;
       when others => StateUD <= S0;
     end case;
end if;
end process;


process (StateUD, EvInUp, EvInDown)
begin
    case StateUD is
       when S0 => EvInUpDown <='0';
       when S1 => if (EvInUp='0' and EvInDown='1')
                         then EvInUpDown <= '1';
                  else EvInUpDown <= '0';
                  end if;
       when others => EvInUpDown <= '0';
     end case;
end process;
```

O primeiro processo é responsável pela mudança de estado da máquina de estados, fazendo a sua avaliação em cada flanco ascendente do relógio. O estado inicial é *S0* e só irá ser alterado se acontecer um evento ascendente passando para o estado *S1*. De seguida, se acontecer um evento descendente, a máquina irá para o estado *S0* novamente. Durante esta ultima transição, de *S1* para *S0*, o segundo processo irá garantir que o evento de entrada é gerado colocando o sinal *EvInUpDown* com valor '1'. Caso haja outra combinação de acontecimentos que não seja um evento ascendente seguido de um descendente, o segundo processo garante que o sinal *EvInUpDown* esteja sempre com valor '0'.

Para o caso do evento constituído por um ciclo descendente-ascendente (*Down-Up*), o código VHDL da máquina de estados para o mesmo sinal *InputSignal* é apresentado de seguida [3]:

Geração do evento constituído por um ciclo descendente-ascendente (Down-Up) de sinais binários:

```
process (CLK, RESET)
begin
if (RESET = '1') then StateDU <=S0;
elsif (CLK'event and CLK='1') then
     case StateDU is
        when S0 => if (EvInUp='0' and EvInDown='1')
                           then StateDU <= S1;
                   end if;
        when S1 => if (EvInUp='1' and EvInDown='0')
                           then StateDU <= S0;
                   end if;
        when others => StateDU <= S0;
     end case;
end if;
end process;


process (StateDU, EvInUp, EvInDown)
begin
     case StateDU is
        when S0 => EvInDownUp <= '0';
        when S1 => if (EvInUp= '1' and EvInDown= '0')
                           then EvInDownUp <= '1';
                   else EvInDownUp <= '0';
                   end if;
        when others => EvInDownUp <= '0';
     end case;
end process;
```

Identicamente ao que foi descrito para o evento constituído por um ciclo ascendente-descendente (*Up-Down*), o primeiro processo garante a troca de estados quando surge um evento descendente seguido de um evento ascendente. O segundo processo, após acontecer esta combinação de acontecimentos, garante que o sinal *EvInDownUp* fica com valor '1', caso contrário estará com valor '0'.

3.2.4. Eventos ascendentes (*Up*) e descendentes (*Down*) de sinais com mais de um bit (*range*)

Para a geração deste tipo de eventos irá ser usada uma estrutura como a que é apresentada na Figura 3.4:

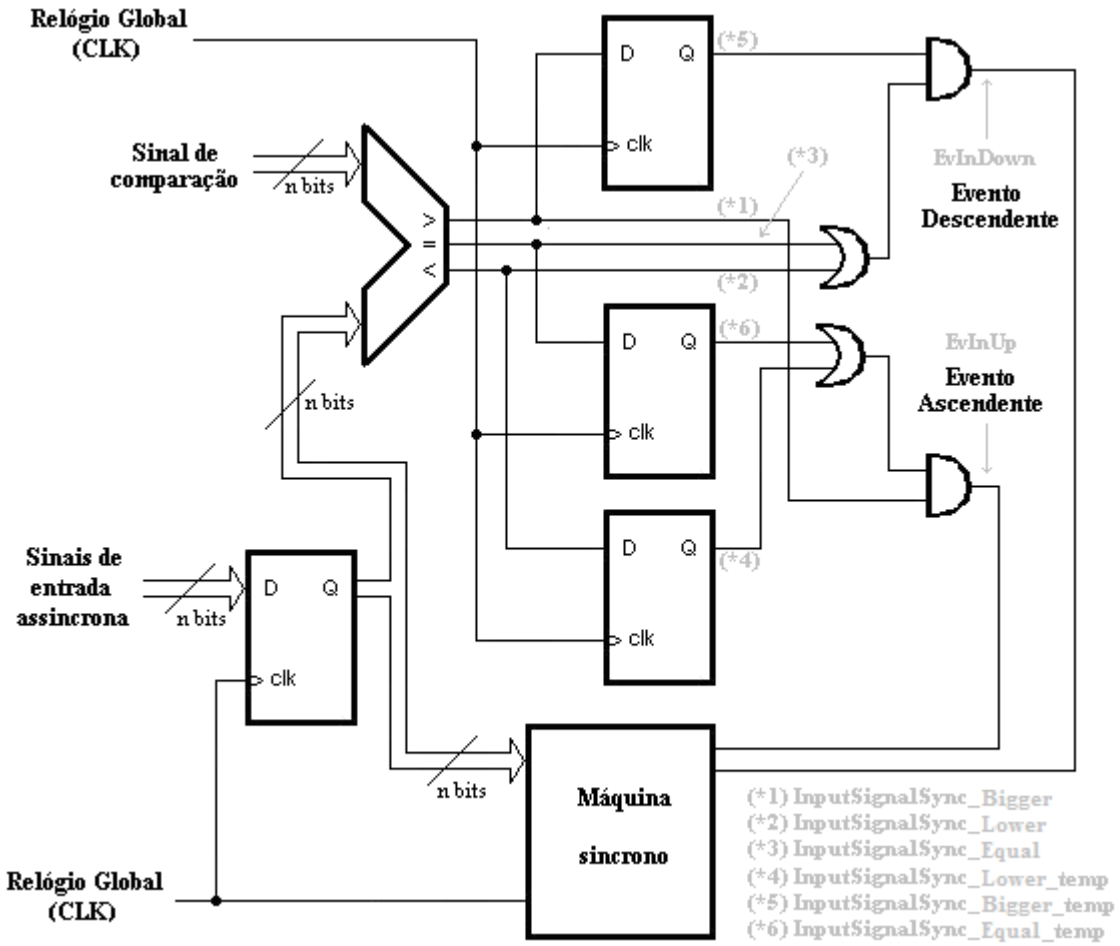

Figura 3.4: Circuito para a geração de eventos *up* e *down* para sinais com mais do que um bit

O sinal assíncrono externo vai passar pelo registo ficando síncrono com o sistema. É então comparado com o valor de referência *level* (valor de *threshold*), gerando três sinais que indicam se o valor actual é igual, menor ou maior ao valor de referência. Através de um registo, estes sinais são guardados (em função de se pretender um evento ascendente ou descendente) e usados posteriormente no próximo evento ascendente de relógio para comparar com o valor actual através de portas AND.

Do esquemático apresentado, é necessário declarar os seguintes sinais em VHDL para representar o evento e os sinais temporários necessários:

Sinais necessários para identificar eventos *Up* e *Down*:

```
signal EvInUp, EvInDown : std_logic;
signal InputSignalSync_Bigger : std_logic;
signal InputSignalSync_Lower : std_logic;
signal InputSignalSync_Equal : std_logic;
signal InputSignalSync_Lower_temp : std_logic;
signal InputSignalSync_Bigger_temp : std_logic;
signal InputSignalSync_Equal_temp : std_logic;
```

Após a declaração dos sinais anteriores, a criação do evento em função de se pretender identificar o flanco ascendente ou o flanco descendente, é realizado da seguinte forma (onde *Level* é obtido a partir do código PNML.):

```
-- Eventos Ascendentes e Descendentes
process (CLK)
begin
      if (CLK'event and CLK='1') then
            InputSignalSync_Lower_temp <= InputSignalSync_Lower;
            InputSignalSync_Bigger_temp <= InputSignalSync_Bigger;
            InputSignalSync_Equal_temp <= InputSignalSync_Equal;
      end if;
end process;

-- Comparador
InputSignalSync_Lower <= '1' when (InputSignalSync<"Level") else '0';
InputSignalSync_Bigger <= '1' when (InputSignalSync>"Level") else '0';
InputSignalSync_Equal <= '1' when (InputSignalSync="Level") else '0';
-- Fim do Comparador

EvInUp <= (InputSignalSync_Lower_temp or
            InputSignalSync_Equal_temp) and (InputSignalSync_Bigger);

EvInDown <= InputSignalSync_Bigger_temp and
            (InputSignalSync_Lower or InputSignalSync_Equal);
```

Onde *EvInUp* representa o evento ascendente e *EvInDown* o evento descendente do sinal *InputSignalSync* (sinal *InputSignal* já sincronizado com o relógio global do sistema).

3.2.5. Eventos constituídos por um ciclo ascendente-descendente (*Up-Down*) e descendente-ascendente (*Down-Up*) de sinais com mais de um bit (*range*)

Para identificar eventos deste tipo é necessário criar máquinas de estados em VHDL de forma a identificar a ocorrência de eventos ascendentes e descendentes e a sua sequência correcta.

Considere-se que *EvInUpDown* e *EvInDownUp* são, repectivamente, os eventos de cliclos ascendente-descendente (*Up-Down*) e descendente-ascendente (*Down-Up*) a gerar e que estão associados ao sinal *InputSignalSync* (sinal *InputSignal* sincronizado com o relógio global do sistema). É importante referir que *EvInUp* e *EvInDown* representam o evento ascendente e descendente, respectivamente, criados da forma já descrita anteriormente e que *InputSignalSyncTemp* representa o sinal *InputSignalSync* atrasado um ciclo de relógio de forma a perceber se houve ou não alteração do sinal.

Na Figura 3.5 apresenta-se o diagrama de estados para a identificação de eventos constituidos por um ciclo ascendente-descendente (*Up-Down*) de um sinal *InputSignal* com mais de um bit.

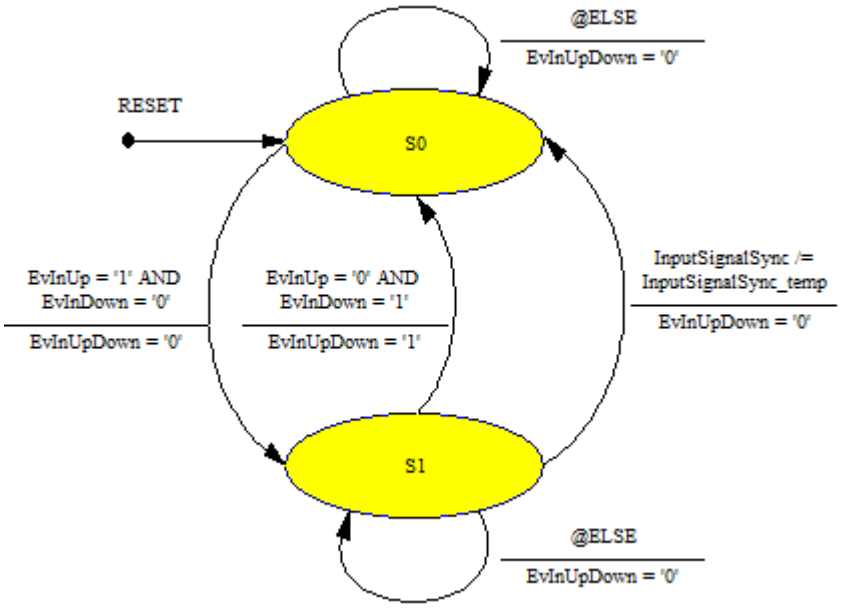

Figura 3.5: Máquina de estados para identificação de eventos constituídos por um ciclo ascendente-descendente (*Up-Down*) de sinais do tipo *range*

Na Figura 3.6 apresenta-se o diagrama de estados para a identificação de eventos constituidos por um ciclo descendente-ascendente (*Down-Up*) de um sinal *InputSignal* com mais de um bit.

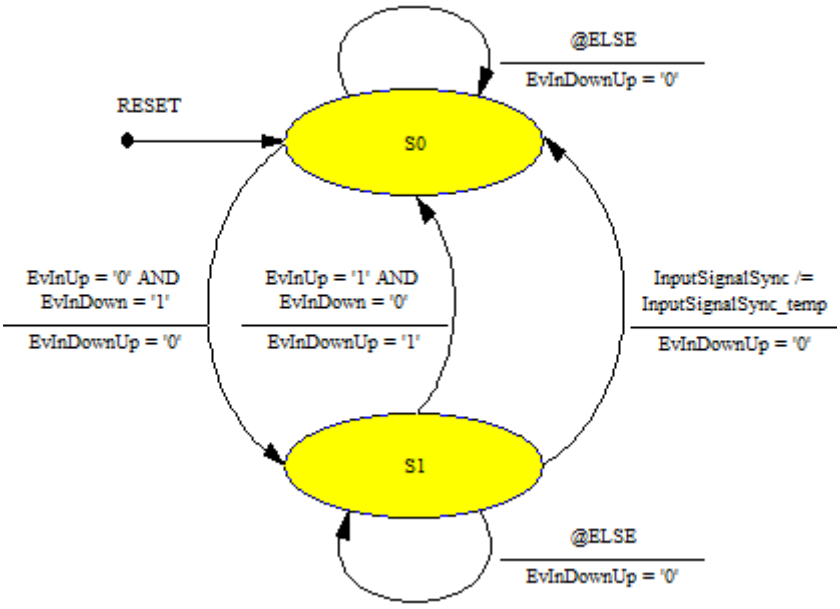

Figura 3.6: Máquina de estados para identificação de eventos de ciclos descendente-ascendente (*Down-Up*) de sinais do tipo *range*

Assim, para a geração do código VHDL das máquinas de estados descritas é necessário criar os seguintes sinais internos:

```
-- Sinal (com N bits) sincronizado com o relógio global do sistema
signal InputSignalSyncTemp : std_logic_vector((N-1) downto 0);

-- Sinal que representa o evento Up-Down a gerar
signal EvInUpDown : std_logic;

-- Sinal que representa o evento Down-Up a gerar
signal EvInDownUp : std_logic;

-- Representa os estados possíveis das máquinas de estados
type states is (S0, S1);

-- Atribuição dos estados para a máquina de estados do evento Up-Down
signal StateUD : states;

-- Atribuição dos estados para a máquina de estados do evento Down-Up
signal StateDU : states;


-- Registo para guardar a variação do sinal de entrada sincronizado
process (CLK)
begin
      if (CLK'event and CLK='1') then
            InputSignalSyncTemp <= InputSignalSync;
      end if;
end process;
```

A geração dos eventos de ciclos ascendente-descendente (*Up-Down*) e descendente-ascendente (*Down-Up*) é feita usando as seguintes máquinas de estados ([2], páginas 205 e 206):

```
-- Máquina de estados para a geração de eventos Up-Down
process (CLK, RESET)
begin
if (RESET = '1') then StateUD <=S0;
elsif (CLK'event and CLK='1') then
    case StateUD is
       when S0 => if (EvInUp='1' and EvInDown='0')
                        then StateUD <= S1;
                  end if;
       when S1 => if ((EvInUp='0' and EvInDown='1')
                        or (InputSignalSyncTemp /= InputSignalSync))
                        then StateUD <= S0;
                  end if;
       when others => StateUD <= S0;
    end case;
end if;
end process;


process (StateUD, EvInUp, EvInDown)
begin
    case StateUD is
       when S0 => EvInUpDown <='0';
       when S1 => if (EvInUp='0' and EvInDown='1')
                        then EvInUpDown <='1';
                  else EvInUpDown <='0';
                  end if;
       when others => EvInUpDown <='0';
    end case;
end process;


-- Máquina de estados para a geração de eventos Down-Up
process (CLK, RESET)
begin
if (RESET = '1') then StateDU <=S0;
elsif (CLK'event and CLK='1') then
    case StateDU is
       when S0 => if (EvInUp='0' and EvInDown='1')
                        then StateDU <= S1;
                  end if;
       when S1 => if ((EvInUp='1' and EvInDown='0')
                        or (InputSignalSyncTemp /= InputSignalSync))
                        then StateDU <= S0;
                  end if;
       when others => StateDU <= S0;
    end case;
end if;
end process;

process (StateDU, EvInUp, EvInDown)
begin
    case StateDU is
       when S0 => EvInDownUp <='0';
       when S1 => if (EvInUp='1' and EvInDown='0')
                        then EvInDownUp <='1';
```

```
                else EvInDownUp <='0';
                end if;
       when others => EvInDownUp <='0';
    end case;
end process;
```

3.3. Tradução de Arcos Normais e de Teste

O agrupamento de arcos com a mesma origem e o mesmo destino permite facilitar a representação gráfica de uma rede de Petri. Este agrupamento é realizado através da atribuição de um peso a cada arco que indica o número de arcos agrupados. A Figura 2.2 demonstra o conceito que se pretende descrever.

A descrição de um arco em PNML usando a classe IOPT é realizada da seguinte forma:

```
<pnml>
    ...
    <arc id="ID" source="IDsource" target="IDtarget">
      <type>Type</type>
      <inscription>
        <value>Value</value>
      </inscription>
    </arc>
    ...
</pnml>
```

Do excerto da representação PNML apresentado pode ser obtida a seguinte informação:

- *ID* do arco, único em toda a rede e que serve para o identificar;
- *Source*, que contém um *ID* (de origem) de um lugar ou de uma transição em função de o arco estar dirigido de um lugar para uma transição ou de uma transição para um lugar, respectivamente;
- *Target*, que contém um *ID* (de destino) de um lugar ou de uma transição em função de o arco estar dirigido de uma transição para um lugar ou de um lugar para uma transição, respectivamente;
- *Type*, que indica o tipo de arco, e que pode ser normal (*normal*) ou de teste (*test*);
- *Value*, que indica o valor do peso do arco.

A representação de pesos em VHDL irá ser realizada através da declaração de constantes inteiras associadas a cada arco (pois são valores fixos durante toda a execução da rede). O

nome destas constantes será iniciado pela letra *W* (de *weight*) no caso de arcos normais e por *Wt* no caso de arcos de teste, seguidos da origem e do destino de cada um deles.

Assim, a título de exemplo, a constante *WT1P1* indica o peso de um arco normal que tem origem na transição *T1* e destino o lugar *P1*. Analogamente, o peso *WtP2T2* representa o peso de um arco de teste com origem no lugar *P2* e destino a transição *T2*.

Considere-se a Figura 3.7 onde se apresentam dois arcos *normais*:

Figura 3.7: Arcos normais

A declaração em VHDL dos pesos destes dois arcos é realizada da seguinte forma:

```
constant WTiPn : integer := A;
constant WPiTn : integer := B;
```

Onde *A* e *B* representam valores inteiros positivos, *WTiPn* um arco com origem numa transição e destino um lugar e *WPiTn* um arco com origem num lugar e destino uma transição.

Na Figura 3.8 pode observar-se um arco de teste com origem num lugar e destino uma transição (os arcos de teste apenas podem ser direccionados de lugares para transições):

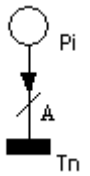

Figura 3.8: Arco de teste

Este tipo de arcos não destroem marcas de um lugar servindo essencialmente para verificar se há ou não um determinado número de marcas num determinado lugar (e que é indicado pelo seu peso) habilitando ou não a transição. Por este motivo não provocam conflitos podendo

apenas estar envolvidos em situações de outras transições que estejam em conflitos. A declaração em VHDL de pesos associados a arcos de teste é realizada da seguinte forma:

```
constant WtPiTn : integer := A;
```

Onde *A* representa um valor inteiro positivo e *WtPiTn* representa um arco de teste com origem num lugar e destino uma transição.

3.4. Tradução de transições

Esta secção apresenta a regra de tradução para transições. Está dividida em função das diversas condições que permitem condicionar o disparo de uma transição. A primeira sub-secção apresenta uma introdução onde se pode encontrar informação comum para as diversas componentes que são apresentadas; a segunda sub-secção apresenta a regra proposta para tradução de condições associadas a sinais de entrada; a terceira sub-secção apresenta a regra para tradução da componente de eventos de entrada; a quarta sub-secção apresenta a regra proposta para tradução das condições associadas à marcação de lugares.

3.4.1. Introdução à tradução de transições

Na Figura 3.9 pode ser observada uma transição IOPT e as suas condições de activação (sinais de entrada e eventos de entrada). Estão também representados os eventos de saída gerados quando a transição estiver apta.

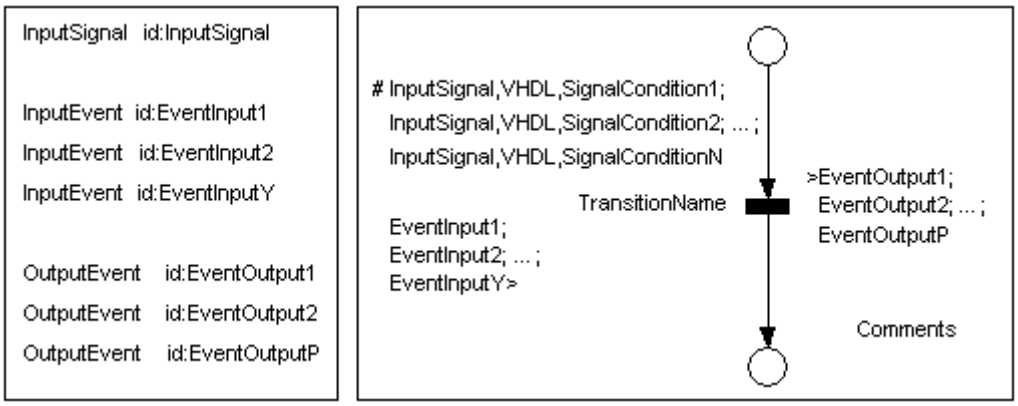

Figura 3.9: Representação de uma transição

Em PNML, a representação da transição é feita com recurso à seguinte estrutura:

```
<transition id="ID">
   <name>
      <text>TransitionName</text>
   </name>
   <comment>
      <text>Comments</text>
   </comment>
   <priority>Priority</priority>
   <signalInputGuards>
      <signalinputguard>
         <concreteSyntax language="VHDL">
            <text>(SignalCondition1)</text>
         </concreteSyntax>
      </signalinputguard>
      <signalinputguard>
         <concreteSyntax language="VHDL">
            <text>(SignalCondition2)</text>
         </concreteSyntax>
      </signalinputguard>
      ...
      <signalinputguard>
         <concreteSyntax language="VHDL">
            <text>(SignalConditionN)</text>
         </concreteSyntax>
      </signalinputguard>
   </signalInputGuards>
   <inputEvents>
      <event idRef="EventInput1"/>
      <event idRef="EventInput2"/>
      ...
      <event idRef="EventInputY"/>
   </inputEvents>
   <outputEvents>
      <event idRef="EventOutput1"/>
      <event idRef="EventOutput2"/>
      ...
      <event idRef="EventOutputP"/>
   </outputEvents>
</transition>
```

Observando a estrutura apresentada, é possível obter a seguinte informação do PNML relativa a cada transição:

- *ID*, número único que identifica a transição em toda a rede;
- *TransitionName*, nome dado pelo utilizador à transição;
- *Comments*, comentários introduzidos pelo utilizador;
- *Priority*, que indica a prioridade de disparo da transição dentro duma contenda;
- Condições de habilitação da transição no que respeita a sinais de entrada;
- Condições de habilitação da transição no que respeita a eventos de entrada;
- Eventos de saída gerados quando a transição disparar.

Em VHDL, uma transição pode ser representada por um sinal binário. Este sinal tem valor lógico '1' caso a transição esteja apta a disparar ou '0' caso a transição não esteja apta. Assim, a declaração do sinal será realizada da seguinte forma:

```
signal TransitionName_ID : std_logic;
```

Onde *TransitionName_ID* é o nome do sinal que representa a transição (concatenação entre o nome e o *ID* dado à transição).

A estrutura base para a sua tradução, usando lógica combinatória, irá ser a seguinte:

```
TransitionName_ID <= '1' when
            (Condition1 and Condition2 and ... and ConditionZ)
            else '0';
```

Onde *Condition1*, *Condition2*, …, *ConditionZ* representam as diversas condições que permitem determinar se uma transição está apta. A tradução das diversas condições possíveis é apresentada nos sub-capítulos seguintes e estão divididos em função da dependência dos sinais de entrada, dos eventos de entrada e finalmente da dependência da marcação. É também apresentada a regra de tradução que permite a resolução de conflitos.

São apenas consideradas as pré-condições de disparo e todas elas são aditivas, ou seja, a avaliação do conjunto de condições que habilitam a transição será feita através de portas lógicas AND. Assim, é necessário que todas as condições de uma transição estejam habilitadas (todas independentes umas das outras) para que a transição esteja apta a disparar. A implementação é realizada utilizando lógica combinatória em que a sua saída está com valor '1' caso as diversas contribuições das condições de habilitação estejam habilitadas, caso contrário a saída estará com valor lógico '0'.

3.4.2. Tradução de condições referentes a sinais de entrada

Considere-se a transição *TransitionName* que é habilitada em função de uma condição exterior *SignalCondition* quando o seu valor for igual a '1' (de forma a generalizar a implementação de sinais exteriores, todos os sinais de um bit são também declarados como *std_logic_vector*).

A avaliação da habilitação da transição é feita da seguinte forma:

```
TransitionName_ID <= '1' when (SignalCondition= "1") else '0';
```

Quando a condição exterior *SignalCondition* estiver a '1', então a transição está apta a disparar.

Considere-se agora que o evento externo *SignalCondition* representa um sinal de três bits e a transição está apta a disparar apenas quando esse sinal estiver com o valor igual a 4 ("100" binário). A declaração da avaliação da transição passa então a ser a seguinte:

```
TransitionName_ID <= '1' when (SignalCondition="100") else '0';
```

Caso haja várias condições de sinais externos, a sua avaliação faz-se através de portas AND e da contribuição individual de cada uma, como referido no final da secção 3.4.1.

3.4.3. Tradução de condições referentes a eventos de entrada

Considere-se a transição *TransitionName* e o sinal binário *EvInput* que representa um evento de entrada (habilitado quando *EvInput='1'*). A habilitação da transição far-se-á considerando a seguinte expressão:

```
TransitionName_ID <= '1' when (EvInput='1') else '0';
```

Caso haja mais que um evento de entrada a influenciar a habilitação de uma transição, estes devem ser avaliados através da introdução de portas AND como referido na sub-secção 3.4.1.

3.4.4. Tradução de transições envolvidas em contendas unitárias

Considere-se a Figura 3.10 onde se apresentam *N* lugares de entrada (*Pe1*, *Pe2*, …, *PeN*), em que cada um está ligado à transição *TransitionName* por um arco de entrada cujo peso é um valor inteiro. Os pesos são representos da forma descrita na secção 3.3.

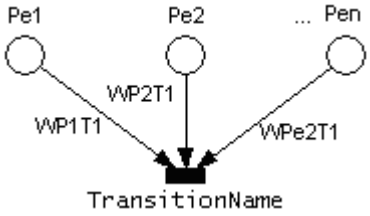

Figura 3.10: Influência de lugares de entrada na habilitação de transições

Surge assim a regra geral para habilitação de transições:

```
TransitionName_ID <= '1' when
            (Pe1>=WPe1T1 and Pe2>=WPe2T1 and … and Pen>=WPenT1)
            else '0';
```

No caso em que os lugares estão ligados à transição por arcos de teste a regra irá ser idêntica mudando apenas a forma como estes arcos são representados (secção 3.3), como pode ser observado para o exemplo da Figura 3.11:

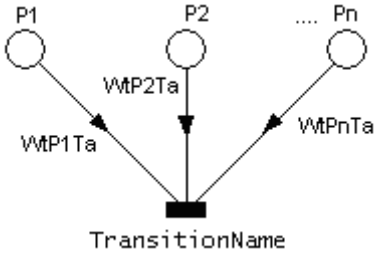

Figura 3.11: Influência de lugares de entrada ligados por arcos de teste

```
TransitionName_ID <= '1' when
      (P1>=WtP1Ta and P2>=WtP2Ta and ... and Pn>=WtPnTa)
      else '0';
```

Todas as condições de habilitação devem ser sujeitas a avaliação através de portas AND e têm de estar todas habilitadas para que a transição esteja habilitada.

Em ambas as situações anteriores, caso haja um valor de marcas maior ou igual ao peso associado a cada arco, a transição irá estar habilitada.

3.4.5. Tradução de transições envolvidas em contendas não unitárias

3.4.5.1. Tradução de lugares de entrada ligados por arcos normais

Com a possibilidade de existir situações de conflito é necessário definir um conjunto de prioridades no disparo das transições (secção 2.1.7.). Desta forma, em situações de contendas não unitárias, é necessário ter em consideração as respectivas prioridades de cada transição que influência a habilitação das restantes transições presentes na contenda.

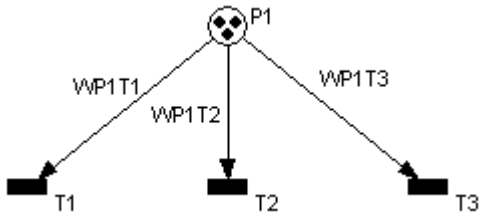

Figura 3.12: Introdução de proridades na habilitação de transições

Na Figura 3.12 pode ser observada uma contenda de transições constituída pelas transições *T1*, *T2* e *T3*. Considere-se que *T1* é prioritária sobre *T2* e que *T2* é prioritária sobre *T3*. Considere-se também que os pesos para cada arco são declarados, respectivamente, como se mostra de seguida:

```
constant WP1T1 : integer := 1;
constant WP1T2 : integer := 2;
constant WP1T3 : integer := 3;
```

Pela observação da Figura 3.12 sabe-se que o lugar *P1* tem três marcas disponíveis.

Considerando que existe uma condição exterior de habilitação das três transições (*a='1'*), disparando as três transições no mesmo instante, então a primeira transição a ficar habilitada será *T1* (pois é prioritária sobre as outras) retirando apenas uma marca das três disponíveis no lugar. Como a habilitação de *T2* retira duas marcas, então esta está também em condições de ser habilitada ficando o lugar sem marcas. Nesta situação não existe marcas suficientes para disparar *T3*, pelo que esta não será habilitada (seriam necessárias mais três marcas).

No entanto, se as condições de habilitação exteriores das três transições forem diferentes e apenas as transições *T2* e *T3* estejam habilitadas (não havendo habilitação da transição *T1*), então a transição que será disparada será apenas *T2* (como fica apenas uma marca no lugar, *T3*

não pode ser disparada). Só no caso de as condições exteriores não habilitarem *T1* e *T2* é que *T3* será habilitada (se a sua condição exterior estiver habilitada).

Só no caso do lugar *P1* possuir seis ou mais marcas é que todas as transições poderão estar habilitadas num mesmo instante.

É então necessário garantir que em cada lugar que habilita a transição existam marcas suficientes para fazer disparar as transições prioritárias. No caso do exemplo anterior as regras de disparo das transições são as seguintes considerando que *T1* é prioritária sobre *T2* e *T2* prioritária sobre *T3* e onde *a='1'* representa a condição exterior de habilitação das transições (neste caso, um evento de entrada):

```
T1 <= '1' when ((a='1') and (P1>=(WP1T1))) else '0';

T2 <= '1' when ((a='1') and (P1>=(WP1T2 +
             + conv_integer(T1)*WP1T1))) else '0';

T3 <= '1' when ((a='1') and (P1>=(WP1T3 +
             + conv_integer(T1)*WP1T1 +
             + conv_integer(T2)*WP1T2))) else '0';
```

T1 mantém-se inalterada em relação à regra apresentada para habilitação de transições (contenda unitária) pois é a transição com prioridade sobre as outras. No entanto *T2* só poderá disparar se depois de disparar *T1* houver marcas suficientes para esse disparo. Pode acontecer também que não haja marcas suficientes para disparar *T1* mas haja para disparar *T2*. O mesmo pensamento funciona para *T3* em relação a *T1* e *T2* (caso não haja marcas suficientes para disparar *T1*, estas podem ser suficientes para disparar *T2* e mesmo *T3* dependendo dos pesos associados aos seus arcos).

Ou seja, resumindo vem que:

```
Ti <= '1' when ((condições externas ='1') and
            (Pi>=(WPiTi + Σ(pesos das transições
                                prioritárias aptas a disparar)))
            else '0';
```

Para a regra geral, considere-se a rede apresentada na Figura 3.13, em que *T1* é prioritária sobre *T2*, *T2* prioritária sobre *T3*, *T3* prioritária sobre *T4*, e assim sucessivamente até *Tn*.

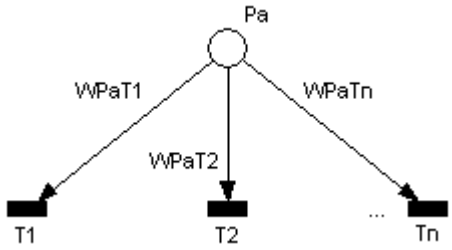

Figura 3.13: Influência de conflitos nas transições

A respectiva tradução será realizada da seguinte forma:

```
T1 <= '1' when (Pa>=(WPaT1)) else '0';
T2 <= '1' when (Pa>=(WPaT2 + conv_integer(T1)*WPaT1)) else '0';
...
Tn <= '1' when (Pa>=(WPaTn +
            + conv_integer(T1)*WPaT1 + conv_integer(T2)*WPaT2 +
            + ... + conv_integer(Tn)*WPaTn)) else '0';
```

Donde surge a seguinte regra para a habilitação de transições (de forma a simplificar a apresentação, fez-se corresponder *Ti* a *TransitionName_ID*):

```
Ti <= '1' when (
     ((Pe1>=(WPe1Ti [+ ... + conv_integer(Tn)*WPe1Tn])) and
      (Pe2>=(WPe2Ti [+ ... + conv_integer(Tn)*WPe2Tn])) and ... and
      (Pen>=(WPenTi [+ ... + conv_integer(Tn)*WPenTn]))))
     else '0';
```

Onde *n* representa a n-ésima prioridade da transição *Ti* no respectivo lugar.

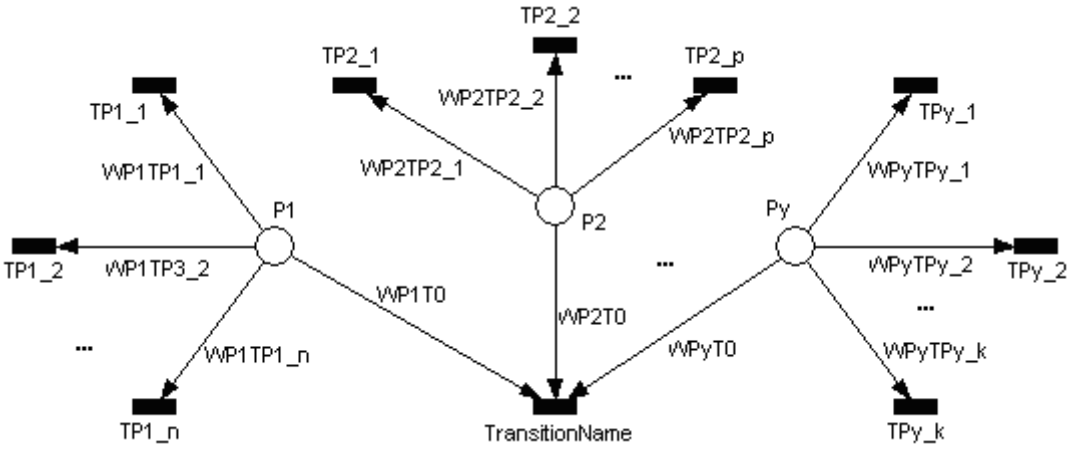

Figura 3.14: Influência de conflitos nas transições (vários lugares)

Considere-se agora a Figura 3.14 onde pode ser observada a transição *TransitionName*, dependente das prioridades das restantes transições da contenda, isto é, que é a transição menos prioritária. Considerando o que foi indicado para a tradução de uma transição dentro de uma contenda, vem a seguinte regra para a transição *TransitionName* e que permite traduzir contendas de transições:

```
WP1 = ∑(WP1TP1_1*conv_integer(TP1_1) +
        WP1TP1_2*conv_integer(TP1_2) + ... +
        WP1TP1_n*conv_integer(TP1_n))

WP2 = ∑(WP2TP2_1*conv_integer(TP2_1) +
        WP2TP2_2*conv_integer(TP2_2) + ... +
        WP2TP2_n*conv_integer(TP2_n))
…
WPy = ∑(WPyTPy_1*conv_integer(TPy_1) +
        WPyTPy_2*conv_integer(TPy_2) + ... +
        WPyTPy_n*conv_integer(TPy_n))

TransitionName <= '1' when (
            P1>=(WP1T0*conv_integer(T0) + WP1) and
            P2>=(WP2T0*conv_integer(T0) + WP2) and ... and
            Py>=(WPyT0*conv_integer(T0) + WPy))
            else '0';
```

Onde *WP1, WP2,…, WPy* representam o somatório dos pesos das transições aptas prioritárias em relação à transição *TransitionName* dos lugares *P1, P2,… , Py*, respectivamente.

3.4.5.2. Tradução de lugares de entrada ligados por arcos de teste

Em situações de contendas não unitárias em que existem diversos arcos de teste ligados de um lugar para várias transições, não existe necessidade de levar em consideração as prioridades associadas. O motivo de tal situação é o facto de um arco de teste não destruir marcas do lugar de origem, estando estas disponíveis para habilitação de uma transição (ou várias) ligada por arcos normais.

Considere-se a rede apresentada na Figura 3.15, em que *T1* é prioritária sobre *T2*, *T2* prioritária sobre *T3*, *T3* prioritária sobre *T4*, e assim sucessivamente até *Tn*. Todas as transições estão ligadas por arcos de teste.

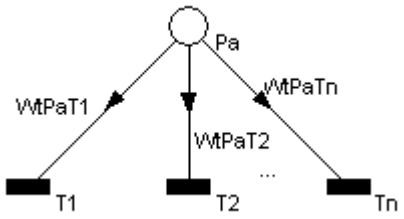

Figura 3.15: Influência de conflitos nas transições ligadas por arcos de teste

A tradução será realizada em função da seguinte regra:

```
T1 <= '1' when (Pa>=WtPaT1) else '0';
T2 <= '1' when (Pa>=WtPaT2) else '0';
...
Tn <= '1' when (Pa>=WtPaTn) else '0';
```

Donde surge a seguinte regra para a habilitação de transições ligadas por arcos de teste (de forma a simplificar a apresentação, fez-se corresponder *Ti* a *TransitionName_ID*):

```
Ti <= '1' when
      ((Pe1>=WPe1Ti) and (Pe2>=WPe2Ti) and ... and (Pen>=WPenTi))
       else '0';
```

Onde *n* representa a n-ésima prioridade da transição *Ti* no respectivo lugar.

Em caso de uma situação mista em que uma transição tem arcos de teste e arcos normais como arcos de entrada, a regra de tradução será baseada nas regras individuais apresentadas para cada tipo de arco, separadas por portas AND.

3.4.6. Tradução de condições passadas directamente do projectista

É possível escrever condições que influenciam a habilitação de transições directamente a partir do editor sem que estejam associados a elementos da rede de Petri. É necessário ter em conta que estas condições têm de ser interpretadas em VHDL, situação que o projectista tem de levar em consideração.

No seguinte exemplo são apresentados dois exemplos de condições que podem estar associadas a uma transição:

```
<transition id="ID">
  ...
   <signalInputGuards>
    ...
      <signalinputguard>
        <concreteSyntax language="C">
          <text>(leave==1)</text>
        </concreteSyntax>
      </signalinputguard>
      <signalinputguard>
        <concreteSyntax language="VHDL">
          <text>(leave='1')</text>
        </concreteSyntax>
      </signalinputguard>
    ...
  </signalInputGuards>
  ...
</transition>
```

Pode ser observado que ambas as condições têm associada uma linguagem. Uma condição destina-se a ser usada em linguagem C e outra, a que será considerada no âmbito deste trabalho, destina-se a linguagem VHDL.

Considere-se que *leave* representa um sinal de entrada, *leaveSync* o sinal de entrada após a sincronização com o relógio do sistema (secção 3.1.2) e que todos os sinais de entrada são representados por vectores de bits (secção 3.1.1) ficando, depois de alterada, a condição *leaveSync="1"*.

Assim, para a transição *TransitionName*, a sua avaliação assume a seguinte forma em VHDL:

```
TransitionName_ID <= '1' when (leaveSync="1") else '0';
```

Caso a condição não pertença a um elemento definido na rede, então a sua tradução irá ficar igual ao que foi indicado pelo projectista:

```
TransitionName_ID <= '1' when (leave="1") else '0';
```

É de salientar que neste caso o projectista deve ter em consideração se a condição representa um sinal binário ou se representa um sinal não binário pois esta será passada tal como está definida em PNML.

3.5. Tradução de lugares

Esta secção pretende propor uma regra de tradução para Lugares. Está dividida em três sub-secções: a primeira apresenta uma introdução com informação para as regras apresentadas; a segunda sub-secção apresenta a regra proposta para tradução de lugares ligados a transições por arcos normais; a terceira sub-secção apresenta a influência de transições ligadas por arcos de teste no cálculo do número de marcas em cada instante no lugar.

3.5.1. Introdução à tradução de lugares

A implementação de um lugar seguro pode ser realizada utilizando um elemento de memória (flip-flop) [1] pois o lugar irá conter no máximo uma marca. Quando a marca está presente no lugar o flip-flop irá ter o sinal lógico '1' na sua saída e '0' quando a marca não está presente no lugar.

No caso de o lugar a implementar ser um lugar k-limitado, a sua implementação é executada substituindo o elemento de memória por um contador de módulo *k+1*, incrementando ou decrementando sempre que as suas transições de entrada ou saída sejam disparadas [1].

De uma forma genérica, pode-se considerar que a implementação de lugares é feita considerando que o número de marcas de um determinado lugar é obtido pelo somatório do número de marcas existentes no lugar com o número de marcas que são criadas e o número de marcas que são destruídas desse lugar num determinado ciclo de relógio.

$$Núm\ Marcas\ \ Final = Núm\ Marcas\ Actual + \sum Marcas\ Que\ Entram - \sum Marcas\ Que\ Saem$$

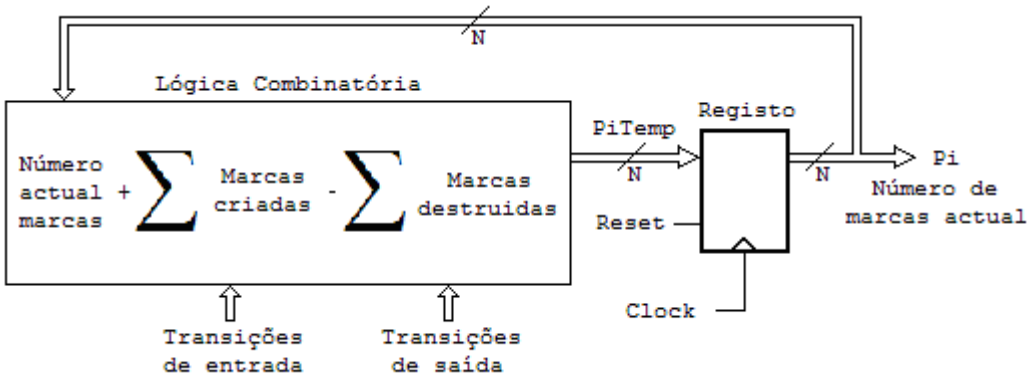

Figura 3.16: Estrutura para o cálculo do número de marcas dum lugar

A activação da influência de cada arco de entrada e saída é feita pela habilitação ou não das transições de origem e destino.

De forma a implementar a soma em VHDL é necessário usar em sequência a função *conv_integer* que converte um *std_logic* para um inteiro e a função *conv_std_logic_vector* que converte um inteiro para um *std_logic_vector* novamente. Nesta última função existe a necessidade de especificar o número de bits máximo do *std_logic_vector* gerado. Este número de bits será obtido através da característica *bound* presente no ficheiro PNML e que indica o número máximo de marcas que um determinado lugar terá durante a execução da rede de Petri. Estas conversões não ocupam qualquer tipo de recursos de hardware [2].

A representação do número de marcas de um lugar é feita através de um sinal temporário em VHDL que recebe o valor do número de marcas final após a execução do somatório já descrito (calculado através de lógica combinatória). O tamanho desse vector é obtido pela característica *bound* da classe IOPT. Após o flanco ascendente do relógio, este sinal passa por um registo e cujo sinal de saída representa o número de marcas naquele instante no lugar.

A característica *bound* é um valor decimal que indica o número máximo de marcas que um certo lugar irá ter durante a execução da rede. Esse valor deverá ser usado para obter o número de bits necessários para a declaração do sinal e também para indicar o número de bits necessários para a conversão de um valor inteiro para *std_logic_vector*. Através da seguinte condição é possível obter o valor de bits necessários:

$$2^N - 1 \geq bound \Leftrightarrow N \geq \frac{\log(bound + 1)}{\log(2)}$$

Após o arredondamento de *N* para o maior valor inteiro obtém-se o número mínimo de bits necessários para a declaração do vector que representa o lugar.

Para a execução da rede de Petri existe a necessidade de indicar o número de marcas de um lugar no arranque da execução. Para esse efeito implementa-se um sinal de *reset* que, além de atribuir o número de marcas iniciais a cada lugar, reinicia também a execução da rede.

Para exemplo considere-se o lugar *P1* cuja característica *bound* é 5, ou seja, irá ter no máximo cinco marcas durante a execução da rede.

```
<place id="ID">
  <name>
    <text>P1</text>
  </name>
  ...
  <bound>
    <text>5</text>
  </bound>
  ...
</place>
```

Então:

$$2^N - 1 \geq 5 \Leftrightarrow N \geq \frac{\log(5+1)}{\log(2)} \approx 2.5849$$

Fazendo o arredondamento para cima de *N* chega-se à conclusão que são necessários 3 bits para representar este lugar, sendo os sinais necessários declarado da seguinte forma em VHDL:

```
signal P1, P1Temp : std_logic_vector(2 downto 0);
```

Pode-se então concluir que a declaração em VHDL de um sinal que representa um lugar *Pi* tem a seguinte regra geral para lugares k-limitados:

```
signal Pi, PiTemp : std_logic_vector((N-1) downto 0);
```

E para lugares seguros (*N=1*, logo *N-1=0*):

```
signal Pi, PiTemp : std_logic_vector(0 downto 0);
```

O sinal *PiTemp* representa o somatório do número de marcas actual com o número de marcas que são criadas e com o número de marcas que são destruídas num determinado instante (o calculo é apresentado nos sub-capítulos seguintes). Após o flanco ascendente do relógio do sistema, o valor no sinal *PiTemp* é atribuído a *Pi*.

O processo em VHDL que representa o elemento de memória que irá actualizar o número de marcas no lugar em cada flanco ascendente do relogio é baseado no processo de flip-flop's do tipo D. Considerando *InitialPlaceValue* como o valor do número de marcas inicial num

determinado lugar (em binário) a ser atribuído quando existe um sinal de *Reset*, então a regra tem o seguinte aspecto:

```
PiTemp <= conv_std_logic_vector((conv_integer(Pi) +
                        + ∑(número de marcas que são criadas) +
                        + ∑(número de marcas que são destruidas)),N);

process (CLK, RESET) begin
      if (RESET = '1') then Pi <= "InitialPlaceValue";
      elsif (CLK'event and CLK='1') then
            Pi <= PiTemp;
      end if;
end process;
```

3.5.2. Tradução de lugares ligados a transições por arcos normais

Considere-se um lugar qualquer, *Pi*, com as seguintes características relevantes para a implementação:

- O número de marcas máximo no lugar durante a execução da rede é de *Z* marcas;
- Deverá ser iniciado, após *reset*, com *InitialPlaceValue* marcas;
- Tem *n* transições de entrada representados por *Te1, Te2,..., Ten* (logo existem *n* arcos de entrada);
- Tem *k* transições de saída, representados por *Ts1, Ts2,..., Tsk* (logo existem *k* arcos de saída);
- Todos os arcos têm um peso qualquer, um valor inteiro, e que são representados pela letra *W* seguido da origem e do destino do arco.

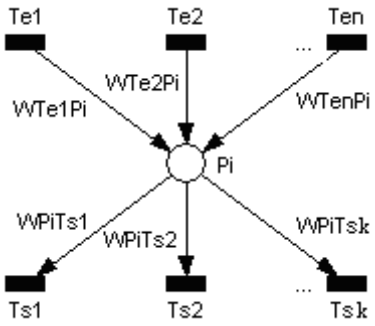

Figura 3.17: Influência de pesos dos arcos nos lugares e transições (regra geral)

Vem então que:

$$2^N - 1 \geq Z \Leftrightarrow N \geq \frac{\log(Z+1)}{\log(2)}$$

Considerando *BitsNumber* como sendo o valor obtido através do arredondamento de *N* para cima, tem-se o número de bits necessários para a conversão do inteiro para vector de bits. A regra geral para a tradução de lugares é então a seguinte:

```
PiTemp <= conv_std_logic_vector((conv_integer(Pi) +
                  + WTe1Pi*conv_integer(Te1) +
                  + WTe2Pi*conv_integer(Te2) + … +
                  + WTenPi*conv_integer(Ten) –
                  – WPiTs1*conv_integer(Ts1) –
                  – WPiTs2*conv_integer(Ts2) – … –
                  – WPiTsk*conv_integer(Tsk)), BitsNumber);

process (CLK, RESET) begin
      if (RESET = '1') then Pi <= "InitialPlaceValue";
      elsif (CLK'event and CLK='1') then
            Pi <= PiTemp;
      end if;
end process;
```

Caso haja uma transição habilitada vai ser adicionado ou retirado um número de marcas igual ao peso associado ao arco que liga esta ao lugar (pois quando a transição está habilitada tem-se *conv_integer(Transição)=1*). Assim, a primeira componente constituída por lógica combinatória é a responsável pelo cálculo do somatório do número de marcas que entram e saem com as actuais marcas no lugar. Após este cálculo, e no próximo flanco ascendente do relógio, o registo coloca na sua saída (representada pelo sinal *Pi*) o valor calculado de forma a representar o número actual de marcas no lugar respectivo.

3.5.3. Tradução de lugares ligados a transições por arcos de teste

Considere-se um lugar qualquer, *Pi*, com as seguintes características relevantes para a implementação:

- O número de marcas máximo no lugar durante a execução da rede é de *Z* marcas;
- Deverá ser iniciado, após *reset*, com *InitialPlaceValue* marcas;

- Tem *k* transições de saída, representados por *Ts1, Ts2,..., Tsk* (logo existem *k* arcos);
- Todos os arcos têm um peso qualquer, um valor inteiro, e que são representados pela letra *W* seguido da origem e do destino do arco;
- Os arcos são arcos de teste.

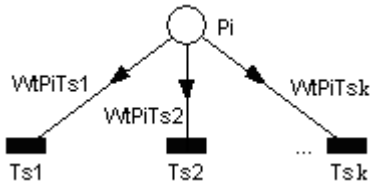

Figura 3.18: Influência de arcos de teste nos lugares

Os arcos de teste não retiram marcas do lugar. Neste sentido, durante a implementação, a influência da habilitação das transições ligadas por arcos de teste será ignorada e não será acrescentada à implementação.

Desta forma, tem-se então a seguinte tradução para lugares ligados com arcos de teste:

```
PiTemp <= conv_std_logic_vector((conv_integer(Pi)), BitsNumber);

process (CLK, RESET) begin
      if (RESET = '1') then Pi <= "InitialPlaceValue";
      elsif (CLK'event and CLK='1') then
            Pi <= PiTemp;
      end if;
end process;
```

Caso haja uma transição habilitada, não vai ser adicionado ou retirado marcas ao lugar.

Esta regra implica que em situações de transições ligadas por arcos normais e arcos de teste a um lugar, o cálculo do número de marcas do lugar irá basear-se na regra apresentada na secção 3.5.2.

3.6. Eventos de saída

A activação de eventos de saída será realizada em função do disparo das transições a que estão associados. É possível associar mais que uma transição a um evento. Neste caso

qualquer uma delas poderá provocar a activação do evento de saída, motivo pelo qual serão separadas através de condições OR na expressão que avalia a habilitação do evento.

De seguida pode ser observado a declaração de dois eventos de saída em PNML:

```
<pnml>
    ...
    <output>
    ...
      <event edge="up" id="EvOutUp"
                          level="Level" signal="OutputSignal">
      </event>
      <event edge="down" id="EvOutDown"
                          level="Level" signal="OutputSignal">
      </event>
    ...
    </output>
    ...
</pnml>
```

De acordo com as características da classe IOPT um evento de saída tem os seguintes atributos:

- Um *ID*, que permite identificá-lo e que será usado para representar o evento de saída em VHDL. Este *ID* é único na rede;
- *OutputSignal*, que indica um sinal de saída associado ao evento, e que irá ser influenciado em função da sua activação (situação dependente da associação do evento de saída a uma transição). Esta informação será usada no capítulo dos sinais de saída;
- Associar um flanco, ascendente (*up*) ou descendente (*down*), que irá indicar qual é a influência que o evento terá no sinal de saída anteriormente referido. Se ascendente implica que a influência será no sentido de aumentar o valor actual do sinal de saída (para '1' no caso de sinais de saída binários ou incrementar no caso dos sinais não binários em função do valor definido em *level*), enquanto descendente a influência será no sentido de diminuir o valor do sinal de saída (para '0' no caso de sinais binários ou fazer decrementos no caso dos sinais não binários em função do valor definido em *level*);
- Especificar um valor *level* que indica o valor a alterar no sinal de saída quando o evento de saída for activado. Se o flanco for ascendente o valor presente em *level* será adicionado ao actual valor do sinal, caso o flanco seja descendente será subtraído o valor presente em *level* ao actual valor do sinal de saída (com significado apenas para sinais não binários).

Os eventos de saída podem apenas assumir valores de verdadeiro (‘1’) ou falso (‘0’), podendo por este motivo ser representados através de sinais do tipo *std_logic* em VHDL.
Considere-se os dois eventos apresentados, *EvOutUp* e *EvOutDown*. Ambos estão associados a *n* transições representadas por *T1*, *T2*, …, *Tn*. O primeiro evento está associado a um flanco ascendente enquanto o segundo está associado a um flanco descendente. Os níveis de *threshold* estão representados como *Level* (inteiro positivo) para os eventos *EvOutUp* e *EvOutDown* (podendo assumir valores diferentes para cada evento).

```
OutputEvent id: EvOutUp Signal: OutputSignal Edge: up level: Level
OutputEvent id: EvOutDown Signal: OutputSignal Edge: down level: Level
```

A regra para a sua implementação será a seguinte:

```
entity NetName is
      Port ( -- General Signals
             ...

             -- Input Signals
             ...

             -- Output Events
             EvOutUp : out std_logic;
             EvOutDown : out std_logic;
             ...
             );
end NetName;

architecture Behavioral of NetName is

...

-- Transitions Signals
signal T1, T2, T3, … : std_logic;

-- Sinais internos do VHDL para representar os Eventos
signal EvOutUp_Temp : std_logic;
signal EvOutDown_Temp : std_logic;


begin

-- Condições de habilitação das Transições (ver secção respectiva)
...

-- Afectação dos Eventos
EvOutUp_Temp <= '1' when (T1='1' or T2='1' or … or Tn='1') else '0';
EvOutDown_Temp <= '1' when (T1='1' or T2='1' or … or Tn='1') else '0';

-- Atribuição do evento ao sinal exterior
EvOutUp <= EvOutUp_Temp;
EvOutDown <= EvOutDown_Temp;
...
end Behavioral;
```

Em suma, quando uma ou várias transições estiverem habilitadas, será gerado um evento de saída. Na regra apresentada pode ser constatado que não existe diferença entre a geração de um evento associado a um flanco ascendente ou descendente pois apenas irá ter consequências no sinal de saída associado (capítulo 3.7). O mesmo acontece com o atributo *Level*.

3.7. Sinais de saída

Esta secção pretende apresentar as regras de tradução propostas para Sinais de Saída. Encontra-se dividida em sete sub-secções: a primeira apresenta uma introdução onde existe informação comum para as regras seguintes; a segunda sub-secção apresenta a regra proposta para tradução de sinais de saída binários associados a lugares; a terceira sub-secção apresenta a regra para tradução de sinais de saída não binários associados a lugares; a quarta sub-secção apresenta a regra proposta para tradução de sinais de saída binários associados a eventos de saída; a quinta sub-secção apresenta a regra proposta para tradução de sinais de saída não binários associados a eventos de saída; a sexta sub-secção apresenta a regra proposta para tradução de sinais de saída binários associados a lugares e a eventos de saída; finalmente, a sétima sub-secção apresenta a regra para tradução de sinais de saída não binários associados a lugares e a eventos de saída.

3.7.1. Introdução

Os sinais de saída de uma rede IOPT são representados em PNML através da seguinte estrutura:

```
<pnml>
   ...
   <output>
      ...
      <signal id="Output1" type="boolean" value="0">
      </signal>
      <signal id="Output2" type="boolean" value="1">
      </signal>
      <signal id="Output3"
                        max="Max" min="Min" type="range" value="Value">
      /signal>
      ...
   </output>
   ...
</pnml>
```

Donde é possível obter as seguintes características:

```
OutputSignal1  id: Output1  type: Boolean  value: 0                (1)
OutputSignal2  id: Output2  type: Boolean  value: 1                (2)
OutputSignal3  id: Output3  type: Range
                            max="Max"  min="Min"  value: Value     (3)
```

Observa-se a existência de três sinais de saída:

- Dois binários (1 e 2), que apenas se diferenciam pelo valor de omissão do sinal de saída;
- Um não binário (3) onde estão definidos os seus valores máximo e mínimo possíveis (representado por *Max* e *Min* respectivamente) e em decimal.

Em ambos os casos é também possível obter o valor por omissão de cada sinal. Implicitamente as declarações dos sinais binários implicam que os valores mínimos e máximos possíveis sejam 0 e 1 respectivamente.

Considere-se os seguintes pressupostos associados aos sinais de saída:

- O sinal de saída *Output3* é constituído por um número de bits igual a *BitsNumber* (arredondamento por excesso de *N*) obtido em função do valor *Max* declarado (em decimal) pelo projectista:

$$2^{N-1} \geq Max \Leftrightarrow N \geq \frac{\log(Max+1)}{\log(2)}$$

- Seja *MaxValue* o valor, em binário, correspondente ao valor *Max* (em decimal) declarado pelo utilizador;
- Seja *MinValue* o valor, em binário, correspondente ao valor *Min* (em decimal) declarado pelo utilizador;
- *Value* é o valor por omissão do sinal de saída *Output3*, em decimal, estabelecido pelo projectista e que deverá estar entre os valores *Max* e *Min* declarados:

$$Min \leq Value \leq Max$$

- *ValueBin* é o valor de *Value* mas em binário;

- Para o caso de sinais binários associados a eventos ascendentes e descendentes, tendo como consequência a utilização de *set* e de *reset* simultaneo do sinal, estabelece-se que o *set* tem prioridade em relação ao *reset.*

Os sinais de saída podem ser influenciados de três formas distintas e que são:

- Sinais de saída associados a eventos de saída (que por sua vez estão associados a transições): são influênciados pelo disparo das transições e são implementados através do uso de elementos de memória. Esta opção corresponde à modelação tipicamente associada às máquinas de Mealy;
- Sinais de saída associados a lugares: situação implementada através de lógica combinatória, influenciando os sinais de saída através da execução se *set*'s ou *reset*'s de valores definidos pelo utilizador para um determinado sinal de saída. Esta opção corresponde à atitude típica das máquinas de Moore;
- Sinais de saída associados a lugares e eventos de saída: situação implementada recorrendo a elementos de memória e consiste numa influência mista dos dois casos apresentados anteriormente. É uma forma de utilizar as máquinas de Mealy e Moore simultaneamente, ambos associados à mesma variável, mas é uma situação que não se recomenda.

Para as últimas duas situações em que os sinais de saída estão associados a lugares, e de forma a apresentar as regras de tradução nas secções seguintes, considere-se um lugar, *Pa*, com *N* transições de entrada e *M* transições de saída representado na Figura 3.19:

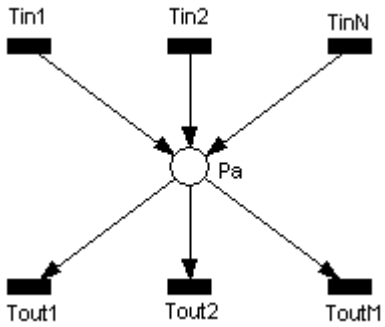

Figura 3.19: Influência de lugares e transições nos sinais de saída

Considere-se os seguintes pressupostos associados à rede da Figura 3.19:

- O lugar *Pa* tem vários arcos de entrada provenientes das transições $T_{in}1$, $T_{in}2$, …, $T_{in}N$;
- O lugar *Pa* tem vários arcos de saída ligados às transições $T_{out}1$, $T_{out}2$, …, $T_{out}M$;
- O sinal de saída depende da marcação do lugar *Pa*, estando o sinal de saída activo se o lugar estiver marcado (*marking>0*);
- A activação do sinal de saída, além da marcação do lugar, pode depender de uma condição fornecida pelo utilizador.

A implementação da regra de tradução para sinais de saída associados a lugares pode estar directamente dependente do número de marcas que está no(s) lugar(es). Esta dependência é especificada através de expressões que fazem referência a lugar(es) durante a fase de associação do sinal de saída ao lugar. As regras para especificar a expressão são as seguintes:

- Utilizar a palavra-chave *marking* para referir marcação do lugar actual. Por exemplo, para associar um sinal qualquer ao lugar actual coloca-se "*marking>0*" indicando que a expressão a ser avaliada corresponde a garantir que o número de marcas no lugar actual tem de ser maior que zero. Por omissão, e caso não exista uma expressão definida para o sinal de saída, esta será a opção seleccionada;
- Referirência à marcação de um lugar diferente do actual. Para tal pode ser usado a expressão genérica *marking(Lugar)>0*. Neste caso tá-se a definir que o número de marcas no lugar *Lugar* terá de ser maior que zero. O nome a utilizar para definir o lugar *Lugar* deverá ser igual ao nome dado ao lugar na sua definição e ao qual se pretende referir;
- É possível especificar uma condição diferente de "maior que zero", usando um operador relacional desde que seja válido em VHDL. Por exemplo, caso se pretenda que a expressão avalie se o lugar *P1* tem duas ou mais marcas, então a expressão a introduzir será:

marking(P1)>=2

- É possível escrever uma expressão que não use a palavra-chave *marking*, desde que seja válida em VHDL. Neste caso é da responsabilidade do projectista garantir que a expressão é válida. Os nomes usados para referir um determinado lugar deverão ser

idênticos aos usados para a sua declaração e é permitido o uso de condições *or* e *and* de forma a combinar várias expressões (desde que válidas em VHDL). Por exemplo, pretende-se uma expressão que garanta a situação anterior e também que haja marcas no lugar *P2*. Esta pode ser realizada em alternativa da seguinte forma:

(P1>=2 and marking(P2)>0)

Esta expressão implica que o lugar *P1* tenha duas ou mais marcas e que o número de marcas do lugar *P2* seja maior que zero.

Considera-se que as expressões/condições de habilitação podem ser mais que uma e são representadas nas regras dos sub-capítulos seguintes como *Expression1*, *Expression2*, …, *ExpressionY*.

3.7.2. Sinais de saída binários (boolean) associados a eventos de saída

Para sinais de saída binários associados a eventos de saída, considerem-se as seguintes situações possíveis:

- Quando o sinal de saída tem um valor por omissão de "0": neste caso os eventos de saída com flanco ascendente irão fazer um *set* ao sinal de saída, ficando este com valor final "1"; em relação aos eventos de saída com flanco descendente o seu valor final é igual ao valor pré-definido para o sinal, implicando que seja realizado um *reset* do sinal ficando este com valor "0";
- Quando o sinal de saída tem um valor por omissão de "1": É idêntico à situação anterior mas com o *set* e o *reset* complementados;
- Caso não existam eventos activos, o valor do sinal de saída deverá manter-se sem alterações.

Para encontrar uma regra de tradução considerem-se os sinais de saída binários apresentados na introdução deste capítulo. Considere-se também que:

- Existem *N* eventos com flancos ascendentes, representados por *EventUp1*, *EventUp2*, …, *EventUpN* e que todos estão associados aos sinais anteriores;

- Existem *M* eventos com flancos descendentes, representados por *EventDown1*, *EventDown2*, …, *EventDownM* e que todos estão associados aos sinais anteriores.

A tradução é feita da seguinte forma:

```
entity NetName is
      Port ( -- General Signals
            CLK : in std_logic;
            RESET : in std_logic;
            ...
            Output1 : out std_logic_vector(0 downto 0);
            Output2 : out std_logic_vector(0 downto 0);
            ...
            );
end NetName;

architecture Behavioral of NetName is

-- Declaração sinais dos eventos de saída
...
signal Output1_Temp: std_logic_vector(0 downto 0);
signal Output2_Temp: std_logic_vector(0 downto 0);
...

begin

-- Constituição dos eventos de saída, caso sejam necessários
-- (EventUp1, EventUp2, ..., EventUpN)

--(1) Caso o valor por omissão seja = "0" (Output1):
process (CLK, RESET) begin
      if (RESET='1') then Output1_temp <= "0";
      elsif (CLK'event and CLK='1') then
            if (EventUp1='1' or EventUp2='1' or ... or EventUpN='1')
                  then Output1_temp <= "1";
            elsif (EventDown1='1' or EventDown2='1' or ... or
                        EventDownM='1')
                  then Output1_temp <= "0";
            end if;
      end if;
end process;

--(2) Caso o valor por omissão seja = "1" (Output2):
process (CLK, RESET) begin
      if (RESET='1') then Output2_temp <= "1";
      elsif (CLK'event and CLK='1') then
            if (EventUp1='1' or EventUp2='1' or ... or EventUpN='1')
                  then Output2_temp <= "0";
            elsif (EventDown1='1' or EventDown2='1' or ... or
                        EventDownM='1')
                  then Output2_temp <= "1";
            end if;
      end if;
end process;

Output1 <= Output1_Temp;
Output2 <= Output2_Temp;

end Behavioral;
```

Caso o valor por omissão do sinal de saída seja "0" os eventos com flanco ascendente associados ao sinal de saída, implementado pelo primeiro registo apresentado no código anterior, vão colocar o valor do sinal de saída a "1". Por sua vez, os eventos com flanco descendente irão colocar o valor "0" no sinal de saída. Caso nenhum dos eventos estejam habilitados o valor actual presente no sinal de saída mantém-se.

Caso o valor por omissão do sinal de saída seja "1", implementado pelo segundo registo, os eventos com flanco ascendente associados ao sinal de saída colocam o valor de saída a "0". Os eventos com flanco descendente irão colocar o sinal de saída com valor "1". De forma idêntica ao anterior, caso nenhum dos eventos esteja habilitado, o valor do sinal mantém-se.

3.7.3. Sinais de saída com mais de um bit (range) associados a eventos de saída

Quando o sinal de saída é constituído por mais de um bit e está associado a eventos de saída é necessário considerar a influência de todos os eventos de saída activos e os seus *level's* associados. O resultado final a atribuir ao sinal é em função do flanco atribuído a cada evento e do somatório dos *level's* em função desse flanco.

Considere-se assim o sinal de saída não binário apresentado na introdução deste capítulo. Considerando *A* o conjunto de eventos de saída associados a flancos ascendentes e *B* o conjunto de eventos de saída associados a flancos descendentes, o valor final a atribuir ao sinal é obtido através da seguinte expressão:

$$Valor\ Final\ Sinal = Valor\ Actual\ do\ Sinal + \sum\left(Levels\ de\ A\right) - \sum\left(Levels\ de\ B\right)$$

É necessário garantir a saturação do sinal de saída, quer superiormente quer inferiormente, no caso de o somatório anterior ultrapassar o valor máximo e mínimo possível definido pelos valores máximo e mínimo do sinal de saída em causa. Por exemplo, caso o somatório apresentado anteriormente tenha resultado decimal 5 e o sinal apenas seja constituído por dois bits (sendo o seu valor máximo em decimal igual a 3), então o sinal deverá ficar com valor em decimal 3.

Considere-se assim o seguinte:

- Seja *Output3* o sinal de saída em causa, declarado como apresentado na introdução deste capitulo;
- Seja *MaxLevel* o valor do somatório dos *level's* dos eventos de saída com flanco ascendente associados ao sinal de saída em causa activos no instante de análise;
- Seja *MinLevel* o valor do somatório dos *level's* dos eventos de saída com flanco descendente associados ao sinal de saída em causa activos no instante de análise;
- Seja *MaxTotal* o valor correspondente ao somatório:

  MaxTotal=Max + *MaxLevel*;

- Seja *MinTotal* o valor correspondente ao somatório:

  MinTotal=Min - *MinLevel*;

- Sejam *LevelUp1, LevelUp2*, ..., *LevelUpN* os saltos ascendentes em decimal, dependentes de um até *N* eventos de saída com flanco ascendente e representados por *EventUp1, EventUp2, …, EventUpN* respectivamente;
- Sejam *LevelDown1*, *LevelDown2*, …, *LevelDownM* os saltos descendentes em decimal, dependentes de um até *M* eventos de saída com flanco descendente e representados por *EventDown1, EventDown2, …, EventDownM* respectivamente.

Assim, para representar a situação que foi referida, a regra de conversão é a seguinte:

```
entity NetName is
      Port ( -- General Signals
             ...
            Output3 : out std_logic_vector((BitsNumber-1) downto 0);
            ...
            );
end NetName;

architecture Behavioral of NetName is

-- Declaração sinais dos eventos de saída, caso sejam necessários
...

signal Output3_temp : std_logic_vector((BitsNumber-1) downto 0);

...
```

```
begin

-- Constituição dos eventos de saída associados a flancos ascendentes
-- (EventUp1, EventUp2, ..., EventUpN)

-- Constituição dos eventos de saída associados a flancos descendentes
-- (EventDown1, EventDown2, ..., EventDownM)

process (CLK, RESET)
variable Output3_int: integer range MinTotal to MaxTotal;

begin

      if (RESET='1') then Output3_temp <= "ValueBin";
      elsif (CLK'event and CLK='1') then
            Output3_int := conv_integer(unsigned(Output3_temp)) +
                  + conv_integer(EventUp1)*LevelUp1 +
                  + conv_integer(EventUp2)*LevelUp2 +
                  + ... +
                  + conv_integer(EventUpN)*LevelUpN -
                  - conv_integer(EventDown1)*LevelDown1 -
                  - conv_integer(EventDown2)*LevelDown2 -
                  - ... -
                  - conv_integer(EventDownM)*LevelDownM;


            if (Output3_int > Max) then Output3_temp <= "MaxValue";
            elsif (Output3_int < Min) then Output3_temp <= "MinValue";
            else Output3_temp <= conv_std_logic_vector(Output3_int,
      BitsNumber);

            end if;
      end if;
end process;

Output3 <= Output3_Temp;

end Behavioral;
```

Quando um dos eventos de saída com flanco ascendente está activo, este irá adicionar o seu contributo através do seu *Level* ao valor final do sinal. Por sua vez, quando um dos eventos de saída com flanco descendente está activo, este irá subtrair o seu contributo definido pelo *Level* do valor final do sinal. O somatório dos vários contributos é realizado num único ciclo de relógio sendo atribuído no final o valor obtido ao sinal de saída caso não seja ultrapassado o valor máximo e mínimo definidos. Caso estes limites sejam ultrapassados, então o valor a atribuir serão o respectivo valor máximo ou mínimo.

3.7.4. Sinais de saída binários (boolean) associados a lugares

Para sinais do tipo binário apenas poderão existir dois estados possíveis, logo a atribuição do valor ao sinal de saída irá depender do valor de omissão definido pelo para o sinal.

Caso o valor de omissão definido ser '0' então o sinal de saída irá ter o valor '1' quando uma (ou várias) das condições de habilitação se verificar. Caso o valor de omissão definido pelo projectista seja '1' então o valor a atribuir ao sinal de saída quando uma (ou várias) das condições for verdadeira é '0'.

Considerando os sinais binários definidos no inicio deste capítulo, então a regra é a seguinte em função dos seus valores de omissão:

```
entity NetName is
      Port ( -- General Signals
            ...
            Output1 : out std_logic_vector(0 downto 0);
            Output2 : out std_logic_vector(0 downto 0);
            ...
            );
end NetName;

architecture Behavioral of NetName is
...
signal Output1_Temp: std_logic_vector(0 downto 0);
signal Output2_Temp: std_logic_vector(0 downto 0);
...

begin

-- Caso o valor pré-definido do sinal for "0"
Output1_temp <= "1" when (Expression1 or Expression2 or ... or
                  ExpressionN) else "0";

-- Caso o valor pré-definido do sinal for "1"
Output2_temp <= "0" when (Expression1 or Expression2 or ... or
                  ExpressionN) else "1";

Output1 <= Output1_Temp;
Output2 <= Output2_Temp;

end Behavioral;
```

No primeiro caso, sinal *Output1*, o valor pré-definido para o sinal é "0". Apenas quando uma (ou várias) das condições de habilitação for verdadeira é que o sinal de saída irá ficar com o valor "1".

No segundo caso, sinal *Output2*, o valor pré-definido para o sinal é "1". Apenas quando uma (ou várias) das condições de habilitação for verdadeira é que o sinal de saída irá ficar com o valor "0".

Ambas as situações são implementadas por lógica combinatória.

3.7.5. Sinais de saída com mais de um bit (range) associados a lugares

Caso o sinal de saída seja constituído por mais de um bit e esteja associado a lugares, podem existir situações em que mais do que uma condição possa estar habilitada e com valores diferentes a atribuir ao sinal de saída. Neste caso considera-se que o valor final do sinal de saída é o maior valor dos valores possíveis em relação às condições habilitadas num determinado instante. Por exemplo, caso existam quatro condições possíveis em que os valores a atribuir ao sinal de saída são '5', '2', '3' e '1' respectivamente, e no caso de apenas as três últimas estarem habilitadas, então o sinal de saída irá ficar com o valor '3' pois é o maior valor entre os valores possíveis para atribuição ao sinal de saída. Esta estratégia permite resolver situações de conflitos em termos da especificação devido ao facto de não haver uma estratégia definida na classe IOPT para a sua resolução. Cabe ao projectista garantir, caso o pretenda, que tal situação não acontece.
Caso nenhuma das expressões esteja habilitada o valor a atribuir ao sinal de saída é o valor de omissão definido para o sinal de saída.

Considere-se o caso do sinal de saída não binário, *Output3*, apresentado na introdução deste capítulo. Considere-se os seguintes pressupostos:

- Os sinais *Sig1*, *Sig2*,…, *SigY* constituídos por um bit que vão estar activados em função da veracidade das expressões descritas anteriormente em que *Sig1* corresponde à *Expression1*, *Sig2* à *Expression2*, e assim sucessivamente até *Y*;
- Os valores *Value1*, *Value2*, …, *ValueY* (em decimal) correspondem aos valores a atribuir ao sinal se as expressões *Expression1*, *Expression2*, …, *ExpressionY* estiverem habilitadas, respectivamente;
- O número de vezes que é necessário chamar a função *Max* é igual ao número de expressões associadas ao sinal de saída menos um. Por exemplo, se houver cinco

expressões associadas ao sinal de saída, então irá ser necessário chamar quatro vezes a função *Max* tal como se mostra de seguida:

Max(Max(Max(Max(S1, S2), S3), S4), S5)

A regra a utilizar irá ser a que é apresentada de seguida:

```
entity NetName is
      Port ( -- General Signals
            ...
            Output3 : out  std_logic_vector((BitsNumber-1) downto 0);
            ...
            );
end NetName;

architecture Behavioral of NetName is

signal Output3_Temp : std_logic_vector((BitsNumber-1) downto 0);
signal Sig1, Sig2, Sig3, ..., SigN : std_logic;

function Max (a, b : in integer) return integer is
begin
      if a>b then return a;
      else return b;
      end if;
end Max;

begin

Sig1 <= '1' when (Expression1) else '0';
Sig2 <= '1' when (Expression2) else '0';
Sig3 <= '1' when (Expression3) else '0';
...
SigN <= '1' when (ExpressionY) else '0';

Output3_Temp <= conv_std_logic_vector(Max(Max(Max(...,
                        conv_integer(Sig1)*Value1,
                        conv_integer(Sig2)*Value2)
                        conv_integer(Sig3)*Value3),...
                        conv_integer(SigN)*ValueY),BitsNumber)
                        when
                        (Sig1='1' or Sig2='1' or ... or SigY='1')
                        else "ValueBin";

Output3 <= Output3_Temp;
end Behavioral;
```

Quando uma das expressões associadas aos lugares for verdadeira irá activar o respectivo sinal que a representa (*Sig1*, *Sig2*,…, *SigY*). É de seguida avaliado o máximo valor a atribuir ao sinal de saída (através da função *Max*) considerando as possibilidades em função dos sinais *Sig1*, *Sig2*,…, *SigY* e finalmente atribuído esse valor à saída.

Os valores *Value1*, *Value2*, …, *ValueY* devem ser valores entre o valor máximo e mínimo definido pelo projectista para o sinal de saída.

3.7.6. Sinais de saída binários (boolean) associados a eventos de saída e a lugares

A tradução de sinais de saída binários associados a eventos de saída e a lugares irá ser baseada nas regras de tradução de sinais binários associados a eventos de saída (secção 3.7.2) e sinais de saída binários associados a lugares (secção 3.7.4).

Desta forma, a tradução será realizada em função do valor de omissão do sinal:

- Caso o valor por omissão seja '1': é realizado um *Set* negado de forma a ficar com valor '0' quando uma ou várias condições associadas aos lugares está habilitada (desde que o valor indicado pelo projectista a atribuir ao sinal de saída pela condição seja '0') ou então quando houver um evento descendente. O *Reset*, também negado de forma a ficar com valor '1', irá ser realizado sempre que as condições de habilitação anteriores não se verifiquem ou haja um evento ascendente;
- Caso o valor por omissão seja '0': é realizado um *Set* a '1' quando uma ou várias condições associadas aos lugares esteja habilitada (e esse seja o o valor indicado pelo projectista a atribuir ao sinal), ou então quando houver um evento ascendente. O *Reset* a '0' irá ser realizado sempre que as condições de habilitação anteriores não se verifiquem ou haja um evento descendente.

Assim, o resultado a atribuir ao sinal de saída é resultado da soma de duas contribuições, uma associada à componente que associa o sinal de saída a lugares e outra componente que associa o sinal de saída a eventos de saída.

Considere-se assim os sinais binários apresentados na introdução deste capítulo. Considere-se também os seguintes pressupostos:

- Existem *N* eventos com flancos ascendentes, representados por *EventUp1*, *EventUp2*, …, *EventUpN* e que todos estão associados aos sinais apresentados anteriormente;
- Existem *M* eventos com flancos descendentes, representados por *EventDown1*, *EventDown2*, …, *EventDownM* e que todos estão associados aos sinais apresentados anteriormente;

- Considere-se as expressões *Expression1*, *Expression2*, …, *ExpressionY* que representam as condições de habilitação passadas pelo utilizador no momento em que associa um lugar ao sinal de saída;
- O *Set* terá prioridade sobre o *Reset* e não existe uma relação de prioridade entre as condições associadas aos lugares e as condições em relação aos eventos de saída associados ao sinal de saída (neste caso basta que uma delas esteja activa para o sinal de saída ficar activo).

As expressões *Expression* podem não se referir apenas a um lugar apesar de estar em associação a ele. Desta forma, a tradução será realizada em função das seguintes situações e da forma descrita para cada uma, sendo o resultado acrescentado na regra seguinte onde indica "Expression":

- Lugar: usa-se o sinal que representa o somatório do número de marcas presente no lugar antes de passar pelo registo (secção 3.5);
- Sinal de entrada: considera-se o sinal sincronizado do sinal de entrada em causa;
- Sinal de saída: considera-se o valor actualizado e actual do sinal em causa;
- Evento de entrada: considera-se o seu estado actual através do sinal que o representa;
- Evento de saída: considera-se o seu estado actual através do sinal que o representa;
- Outras situações, em que a expressão não é reconhecida como nenhum caso anterior, é passada na integra para o código VHDL.

A regra de tradução é a seguinte:

```
entity NetName is
      Port ( -- General Signals
            CLK : in std_logic;
            RESET : in std_logic;
            ...
            Output1 : out  std_logic_vector(0 downto 0);
            Output2 : out  std_logic_vector(0 downto 0);
            ...
            );
end NetName;


architecture Behavioral of NetName is

--(1) No caso do valor por omissão ser '0'
signal OutPut1_TempZev, OutPut1_TempZpl,
                        OutPut1_TempZplTemp : std_logic;
```

```
--(2) No caso do valor por omissão ser '1'
signal OutPut2_TempZev, OutPut1_TempZpl,
                        OutPut1_TempZplTemp : std_logic;


begin

-- Condições de habilitação das Transições
...

-- Constituição dos eventos
...

-- Constituição dos lugares
...


--(1) No caso do valor por omissão ser '0'
Output1_TempZplTemp <= '1' when
            (Expression1 or Expression2 or ... ExpressionY)
            else '0';

--(2) No caso do valor por omissão ser '1'
Output2_TempZplTemp <= '1' when
            (Expression1 or Expression2 or ... ExpressionY)
            else '0';

--(1) No caso do valor por omissão ser '0'
process (CLK) begin
      if (CLK'event and CLK='1') then
            Output1_TempZpl <= Output1_TempZplTemp;
      end if;
end process;

--(2) No caso do valor por omissão ser '1'
process (CLK) begin
      if (CLK'event and CLK='1') then
            Output2_TempZpl <= Output2_TempZplTemp;
      end if;
end process;


--(1) No caso do valor por omissão ser '0'
process (CLK, RESET) begin
      if (RESET='1') then OutPut1_TempZev <= '0';
      elsif (CLK'event and CLK='1') then
            if (EventUp1='1' or
                  EventUp2='1' or ... or
                  EventUpN='1')
                  then OutPut1_TempZev <= '1';
            elsif (EventDown1='1' or
                  EventDown2='1' or ... or
                  EventDownM='1')
                  then OutPut1_TempZev <= '0';
            end if;
      end if;
end process;


--(2) No caso do valor por omissão ser '1'
process (CLK, RESET) begin
      if (RESET='1') then OutPut2_TempZev <= '0';
      elsif (CLK'event and CLK='1') then
```

```
            if (EventDown1='1' or
                  EventDown2='1' or ... or
                  EventDownM='1')
                  then OutPut2_TempZev <= '1';
            elsif (EventUp1='1' or
                  EventUp2='1' or ... or
                  EventUpN='1')
                  then OutPut2_TempZev <= '0';
            end if;
      end if;
end process;


--(1) No caso do valor por omissão ser '0'
OutPut1 <= "1" when
                  (OutPut1_TempZev='1' or OutPut1_TempZpl='1')
                  else "0";


--(2) No caso do valor por omissão ser '1'
OutPut2 <= "0" when
                  (OutPut1_TempZev='1' or OutPut1_TempZpl='1')
                  else "1";
end Behavioral;
```

É importante referir que apenas uma das situações descritas na regra como (1) ou (2) é que é utilizada, escolhida em função do valor de omissão definido para o sinal de saída.

A cada *Expression* está associado um sinal "*Lugar_TempE*", usado na tradução de lugares para representar o estado que irá ser atribuido ao lugar após o flanco ascendente do relógio.

A titulo de exemplo, se considerarmos que: *Expression* é "marking(P1)>0" em que *P1* é um lugar limitado a três marcas no máximo, então a regra será declarada da seguinte forma (com valor de omissão igual a '0'):

```
-- Sinal de 2 bits para representar as marcas do lugar
signal Lugar_TempE1: std_logic_vector(1 downto 0);

...

Lugar_TempE1 <= Cálculo igual ao descrito para a tradução de lugares;
...

--(1) Caso o valor por omissão seja = "0":

Output1_TempZplTemp <= '1' when (P1_184>0) else '0';

process (CLK) begin
      if (CLK'event and CLK='1') then
            Output1_TempZpl <= Output1_TempZplTemp;
      end if;
end process;

process (CLK, RESET)
```

```
begin
      if (RESET='1') then Output1_TempZev <= "0";
      elsif (CLK'event and CLK='1') then
            if ((EventsUp='1'))  -- eventos ascendentes
                  then Output1_TempZev <= "1";
            elsif ((EventsDown='1'))  -- eventos descendentes
                  then Output1_TempZev <= "0";
            end if;
      end if;
end process;

Output1 <= "1" when (Output1_TempZev='1' or Output1_TempZpl='1') else "0";
```

No caso de uma ou várias das condições de activação do sinal de saída estarem activas, o sinal será actualizado tendo como principio que a activação (em relação ao valor de omissão) é prioritária sobre a desactivação. Caso nenhuma das condições esteja habilitada, o sinal de saída irá manter-se com o valor actual.

3.7.7. Sinais de saída com mais de um bit (range) associados a eventos de saída e a lugares

A tradução irá basear-se na regra abordada para os sinais de saída não binários associados a eventos de saída (secção 3.7.3) e a lugares (3.7.5), usando de forma semelhante o mesmo princípio.

O valor final a atribuir ao sinal de saída é em função de duas contribuições: uma contribuição em função da componente que associa o sinal de saída a lugares; e a contribuição da componente que associa o sinal de saída a eventos de saída. O valor final do sinal de saída é o valor máximo das duas contribuições.

No caso de haver mais que uma condição de lugares activa num instante, o valor a considerar será o valor máximo entre os valores possíveis nesse instante.

Considere-se assim o sinal de saída *Output3* apresentado na introdução desta secção, o qual está neste caso associado a eventos de saída e a lugares. Considere-se os seguintes pressupostos:

- Existem *N* eventos com flancos ascendentes, representados por *EventUp1*, *EventUp2*, …, *EventUpN* e com *Level's* igual a *LevelUp1*, *LevelUp2, …, LevelUpN* respectivamente, estando todos associados ao sinal *Output3*;
- Existem *M* eventos com flancos descendentes, representados por *EventDown1*, *EventDown2*, …, *EventDownM* e com *Level's* igual a *LevelDown1*, *LevelDown2, …, LevelDownM* respectivamente, estando todos associados ao sinal *Output3*;
- As expressões *Expression1*, *Expression2*, …, *ExpressionY* representam as condições de habilitação passadas pelo utilizador no momento em que associa um lugar ao sinal de saída (para mais informação sobre expressões pode ser consultada a introdução à tradução dos sinais de saída associados a lugares);
- Na associação de lugares ao sinal de saída foi definido *Value1*, *Value2*, …, *ValueY* como valores a atribuir ao sinal de saída para cada *Expression1*, *Expression2, …, ExpressionY* respectivamente;
- Os sinais *Sig1*, *Sig2*,…, *SigY* constituídos por um bit, serão activados em função da veracidade das expressões descritas anteriormente em que *Sig1* corresponde à *Expression1*, *Sig2* à *Expression2*, e assim sucessivamente até *Y*;
- Seja *MaxLevel* o valor do somatório dos *level's* dos eventos de saída com flanco ascendente e associados ao sinal de saída em causa;
- Seja *MinLevel* o valor do somatório dos *level's* dos eventos de saída com flanco descendente e associados ao sinal de saída em causa;
- Seja *MaxTotal* o valor correspondente ao somatório:

  MaxTotal=Max + *MaxLevel*;

- Seja *MinTotal* o valor correspondente ao somatório:

  MinTotal=Min - *MaxLevel*;

A regra de tradução é apresentada de seguida:

```
entity NetName is
      Port ( -- General Signals
            CLK : in std_logic;
            RESET : in std_logic;
            ...
            Output3 : out std_logic_vector((BitsNumber-1) downto 0);
            ...
            );
end NetName;
```

```
architecture Behavioral of NetName is


-- Declaração sinais dos eventos de saída, caso sejam necessários
...

signal Output3_tempZev : std_logic_vector((BitsNumber-1) downto 0);
signal Output3_tempZpl : std_logic_vector((BitsNumber-1) downto 0);
signal Output3_tempZplTemp : std_logic_vector((BitsNumber-1) downto 0);

...

-- Ver capítulo de tradução de lugares
signal Lugar_TempE1: std_logic_vector((L1-1) downto 0);
signal Lugar_TempE2: std_logic_vector((L2-1) downto 0);
...
signal Lugar_TempEN: std_logic_vector((LN-1) downto 0);

...

signal Sig1, Sig2, Sig3, ..., SigN : std_logic;

...


function Max (a, b : in integer) return integer is
begin
      if a>b then return a;
      else return b;
      end if;
end Max;


begin

-- Constituição dos eventos de saída de flanco ascendente
-- (EventUp1, EventUp2, ..., EventUpN)

-- Constituição dos eventos de saída de flanco descendente
-- (EventDown1, EventDown2, ..., EventDownM)

-- Cálculo dos valores das marcações dos lugares, em adianto ao clock
-- (ver caítulo de tradução de lugares)

-- Expression1
Lugar_TempE1 <= Cálculo das marcas no lugar;
-- Expression2
Lugar_TempE2 <= Cálculo das marcas no lugar;
...
-- ExpressionY
Lugar_TempEY <= Cálculo das marcas no lugar;

Sig1 <= '1' when (Expression1) else '0';
Sig2 <= '1' when (Expression2) else '0';
Sig3 <= '1' when (Expression3) else '0';
...
SigY <= '1' when (ExpressionY) else '0';


Output3_tempZplTemp  <= conv_std_logic_vector(Max(Max(Max(...,
            conv_integer(Sig1)*Value1,
            conv_integer(Sig2)*Value2),
            conv_integer(Sig3)*Value3),...
```

```
            conv_integer(SigY)*ValueY),BitsNumber)
            when (Sig1='1' or Sig2='1' or Sig3='1' or ... or SigY='1')
            else "ValueBin";


process (CLK) begin
      if (CLK'event and CLK='1') then
            Output3_tempZpl <= Output3_tempZplTemp;
      end if;
end process;


process (CLK, RESET)
variable Output3_int: integer range MinTotal to MaxTotal;
begin
      if (RESET='1') then Output3_tempZev <= "ValueBin";
      elsif (CLK'event and CLK='1') then

            Output3_int := conv_integer(unsigned(Output3_tempZev)) +
                  + conv_integer(EventUp1)*LevelUp1 +
                  + conv_integer(EventUp2)*LevelUp2 +
                  + ... +
                  + conv_integer(EventUpN)*LevelUpN -
                  - conv_integer(EventDown1)*LevelDown1 -
                  - conv_integer(EventDown2)*LevelDown2 -
                  - ... -
                  - conv_integer(EventDownM)*LevelDownM;

            if (Output3_int > Max)
                  then Output3_tempZev <= "MaxValue";
            elsif (Output3_int < Min)
                  then Output3_tempZev <= "MinValue";
            else Output3_tempZev <= conv_std_logic_vector(Output3_int,
            BitsNumber);
            end if;

      end if;
end process;

Output3 <= conv_std_logic_vector(Max(conv_integer(Output3_tempZev),
                  conv_integer(Output3_tempZpl)),BitsNumber);

end Behavioral;
```

A cada *Expression* irá estar associado um sinal, "*Lugar_TempE*", que representa o estado futuro do lugar calculado antes da ocorrência do flanco ascendente do relógio. Estes sinais são declarados como referido na secção de tradução dos lugares (secção 3.5). Os valores *L1*, *L2*, ..., *LN* representam o número de bits necessário para representar o número total de marcas que o lugar irá ter durante a execução da rede e será obtido através da característica *bound* de cada lugar (secção 3.5).

O sinal *Output3_tempZpl* irá representar o valor que deverá ser atribuído ao sinal de saída em função das condições activas associadas aos lugares, pelo que irá conter o valor máximo associado às condições activas naquele instante.

O sinal *Output3_tempZev* irá representar o valor do somatório dos pesos dos eventos de saída associados ao sinal de saída e que estão activos num determinado instante.

O sinal *Output3_temp* irá representar o valor a atribuir no próximo flanco ascendente do relógio ao sinal de saída e é constituído pelo valor máximo entre o sinal *Output3_tempZpl* e *Outpu3t_tempZev* num determinado instante.

Para o cálculo do valor a atribuir a *Output3_tempZev* é usado inteiro *Output3_int* e que irá conter o somatório dos pesos dos eventos de saída activos, desde que não ultrapasse os valores *MaxValue* e *MinValue* definidos pelo utilizador como limites máximos e mínimos respectivamente. Caso os limites sejam ultrapassados para o sinal de saída, este irá ficar com o valor máximo ou mínimo definidos, respectivamente.

A titulo de exemplo, se considerarmos que: *Expression* é "*marking(P1)>0*" em que *P1* é um lugar limitado a três marcas no máximo, então a regra será declarada da seguinte forma:

```
-- Ver declaração de lugares
signal Lugar_TempE1: std_logic_vector(1 downto 0); -- 2 bits
...

Lugar_TempE1 <= Cálculo descrito na secção 3.5;
...

Sig1 <= '1' when (Lugar_TempE1>0) else '0';

...

Output3_tempZplTemp  <= conv_std_logic_vector(
            conv_integer(Sig1)*Value1,BitsNumber)
            when (Sig1='1')
            else "ValueBin";

process (CLK) begin
      if (CLK'event and CLK='1') then
            Output3_tempZpl <= Output3_tempZplTemp;
      end if;
end process;

process (CLK, RESET)
variable Output3_int: integer range MinTotal to MaxTotal;
begin
      if (RESET='1') then Output3_tempZev <= "ValueBin";
      elsif (CLK'event and CLK='1') then
```

```
            Output3_int := conv_integer(unsigned(Output3_int)) +
                  + conv_integer(EventUp1)*LevelUp1 +
                  + conv_integer(EventUp2)*LevelUp2 +
                  + ... +
                  + conv_integer(EventUpN)*LevelUpN -
                  - conv_integer(EventDown1)*LevelDown1 -
                  - conv_integer(EventDown2)*LevelDown2 -
                  - ... -
                  - conv_integer(EventDownM)*LevelDownM;


            if (Output3_int > MaxValue)
                              then Output3_tempZev <= "MaxValB";
            elsif (Output3_int < MinValue)
                              then Output3_tempZev <= "MinValB";
            else Output3_temp <= conv_std_logic_vector(Output3_int,
            BitsNumber);
            end if;

      end if;
end process;

Output3 <= conv_std_logic_vector(Max(conv_integer(Output3_tempZev),
                conv_integer(Output3_tempZpl)),BitsNumber);
```

Por outro lado, as expressões *Expression* podem não se referir apenas a um lugar apesar de estar associado a ele. Desta forma, a tradução será realizada em função das seguintes situações e da forma descrita para cada uma, sendo o resultado acrescentado na regra onde está indicado "Expression":

- Lugar: usa-se o sinal que representa o somatório do número de marcas presente no lugar antes de passar pelo registo (secção 3.5);
- Sinal de entrada: considera-se o sinal sincronizado do sinal de entrada em causa;
- Sinal de saída: considera-se o valor actualizado e actual do sinal em causa;
- Evento de entrada: considera-se o seu estado actual através do sinal que o representa;
- Evento de saída: considera-se o seu estado actual através do sinal que o representa;
- Outras situações, em que a expressão não é reconhecida como nenhum caso anterior, é passada na integra para o código VHDL.

4. Exemplos

Este capítulo pretende apresentar exemplos da aplicabilidade das regras de tradução através da apresentação da tradução de redes de Petri bem estudadas na literatura existente. São explicados os métodos usados para a tradução e apresentam-se os resultados obtidos das simulações efectuadas através do ambiente Xilinx ISE e ModelSim [32] para verificar o correcto funcionamento da rede traduzida.

4.1. Parque de estacionamento 1-in 1-out

4.1.1. Descrição do modelo do parque de estacionamento 1-in 1-out

Pretende-se modelar através de uma rede de Petri o controlador de entradas e saídas de um parque de estacionamento com uma entrada e uma saída (Figura 4.1). A lotação deste parque é definida pelo número de marcas presentes no lugar *P4* (que representa o número de lugares vazios dentro do parque), limitada neste exemplo a três lugares de estacionamento. O lugar *P3* é o lugar complementar do lugar *P4* e representa o número de lugares ocupados dentro do parque de estacionamento.

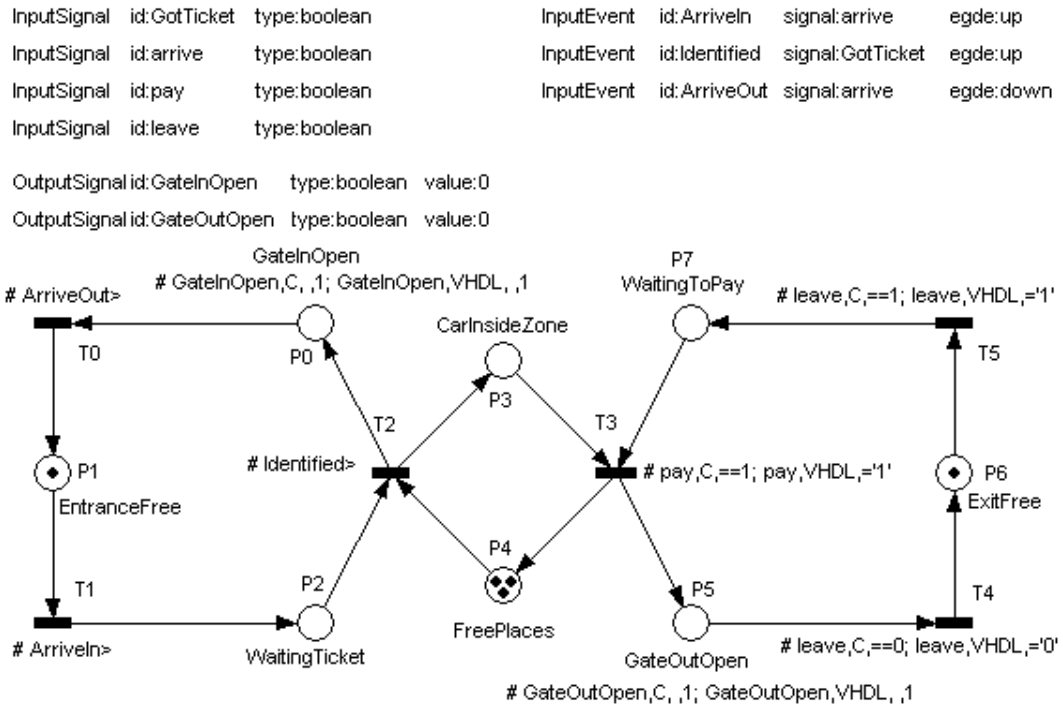

Figura 4.1: RdP de um controlador de entradas e saídas de um parque de estacionamento (uma entrada e uma saída)

Quando uma entrada está livre (situação modelada pelo lugar *P1*) e um carro chega à entrada, o sinal *arrive* fica com valor lógico '1'. Existe assim uma transição do sinal de '0' para '1' levando a que seja gerado um evento ascendente, *ArriveIn*, que vai indicar a chegada de um carro à entrada. Neste caso o lugar *P1* fica sem marcas e o lugar *P2* com uma marca indicando a existência de um carro na entrada em causa. Após o condutor obter um "*ticket*", o sinal *GotTicket* fica com valor '1' e é gerado um evento ascendente deste sinal chamado *Identified* que indica que o condutor pode entrar caso haja lugares disponíveis no parque. O lugar *P0* passa a ter uma marca e é aberta a cancela de entrada através de um sinal gerado *GateInOpen*. Após o sinal *arrive* ficar com valor lógico '0' é criado um evento descendente deste sinal, chamado *ArriveOut*, indicando que o carro já não está na entrada e fechando assim a cancela através da colocação do valor lógico '0' no sinal "*GateInOpen*". Fica assim a entrada disponível para outro carro.

Para a saída o funcionamento é idêntico. Quando a saída está livre o lugar *P6* está marcado. Quando um carro se apresenta na saída é colocado no sinal *leave* o valor lógico '1', ficando assim o controlador à espera que seja dada a indicação de que foi realizado o pagamento respectivo. Após o sinal *pay* estar com o valor '1', indicando que foi realizado o pagamento, a cancela de saída é aberta através do sinal "*GateOutOpen*" e é libertado um lugar. Quando o sinal *leave* está com valor lógico '0', significa que o carro já não se encontra na entrada e é fechada a cancela, ficando assim a saída à espera de um novo automobilista.

4.1.2. Tradução

4.1.2.1. Estrutura base do ficheiro VHDL

Inicialmente o ficheiro em VHDL tem uma estrutura base que irá ser preenchida por passos à medida que se progride na tradução. Essa estrutura pode ser observada de seguida e onde se inclui algumas referências, em comentários VHDL, a esses passos futuros.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity park1in1out is
    Port (
            -- General Signals
            CLK : in std_logic;
            RESET : in std_logic;
```

```
            -- Sinais de entrada assincronos (Inputs)
            -- Sinais de saída (Outputs)
            -- Eventos de saída
            );
end Name;

architecture Behavioral of park1in1out is

-- Constantes (pesos de arcos)
-- Sinais Internos
-- Sinais sincronizados de entrada
-- Sinais de eventos de entrada
-- Sinais representativos das transições
-- Sinais representativos dos lugares

begin

-- Transições
-- Processo do Registo de sincronização
-- Processos para os eventos de entrada
-- Eventos de entrada
-- Lugares
-- Eventos de saída

end Behavioral;
```

O nome da *entity* e da *architecture*, em negrito no exemplo apresentado, será obtido a partir do atributo *name* do elemento *net* presente no ficheiro PNML. Caso este atributo não exista ou esteja vazio o nome será obtido a partir do próprio nome do ficheiro a converter após verificar se o ficheiro aberto é válido, enviando um aviso ao utilizador.

```
<pnml>
    <net id="1" name="park1in1out" type="IOPN">
      <input/>
      <output/>
      ...
```

Neste exemplo o nome a atribuir à *entity* e à *architecture* é *park1in1out*. No caso de o nome conter espaços, estes são retirados.

Também se observa a existência de dois sinais de entrada, o relógio global do sistema e o sinal de *reset*. São sinais necessários e obrigatórios pois a tradução é baseada em sistemas síncronos.

A tradução completa da rede em causa pode ser consultada no Anexo 3.

4.1.2.2. Sinais de entrada

Os sinais de entrada (assíncronos) serão obtidos por leitura do ficheiro PNML através do elemento *input*, em que cada sub-elemento com o nome *signal* representa um sinal de *input* distinto tal como pode ser observado no seguinte excerto de código PNML da rede da Figura 4.1:

```
<input>
      ...
  <signal id="GotTicket" type="boolean">
  <signal id="arrive" type="boolean">
  <signal id="pay" type="boolean">
  <signal id="leave" type="boolean">
      ...
</input>
```

De seguida apresenta-se o interior de um sub-elemento *signal*.

```
  <signal id="GotTicket" type="boolean">
    <graphics>
      <position page="1" x="617" y="60"/>
    </graphics>
  </signal>
```

Verifica-se a existência no interior de cada declaração de cada sinal físico informação sobre aspectos gráficos que irá ser ignorada pela aplicação de tradução.

Em relação à rede apresentada é constituída por quatro sinais em que todos são identificados pelo seu *ID* e tipo de sinal.

Quando o sinal é do tipo binário, e considerando o que foi dito na secção 3.1.1, a tradução para VHDL dos sinais é feita da seguinte forma:

```
GotTicket : in std_logic_vector(0 downto 0);
arrive : in std_logic_vector(0 downto 0);
pay : in std_logic_vector(0 downto 0);
leave : in std_logic_vector(0 downto 0);
```

Caso não seja encontrado qualquer sinal no elemento *input*, o programa irá informar o utilizador com um aviso. Se houver sinais com o mesmo *ID* ou este não existir o utilizador é avisado com um erro.

4.1.2.3. Sinais de saída

Os sinais físicos de saída podem ser encontrados no elemento *output* do ficheiro PNML. Considere-se o seguinte excerto da rede apresentada:

```
<output>
  <signal id="GateInOpen" type="boolean" value="0">
  <signal id="GateOutOpen" type="boolean" value="0">
</output>
```

Existem dois sinais de saída, binários, que segundo o que foi dito na secção 3.7, ficam em VHDL da seguinte forma:

```
GateInOpen : out std_logic_vector(0 downto 0);
GateOutOpen : out std_logic_vector(0 downto 0);
```

Caso o elemento *output* esteja vazio, o programa irá mostrar um aviso ao utilizador. Caso não seja encontrado um *ID* válido, o utilizador será avisado através de um erro.

Dentro do sub-elemento *signal* pode existir ainda informação sobre aspectos gráficos destinados exclusivamente ao software de edição. Esta informação é irrelevante para a tradução e será ignorada.

```
<signal id="GateInOpen" type="boolean" value="0">
  <graphics>
    <position page="1" x="166" y="260"/>
  </graphics>
</event>
```

De seguida mostra-se a tradução que será efectuada para os sinais de saída de acordo com as regras apresentadas, onde *P0_189* representa o número de marcas presentes no lugar *P0* e *P5_259* representa o número de marcas presentes no lugar *P5* (o sinal que identifica o lugar é constituído pela concatenação do nome do sinal com o *ID* do lugar):

```
GateInOpen_Temp <= "1" when (P0_189>0) else "0";
GateInOpen <= GateInOpen_Temp;

GateOutOpen_Temp <= "1" when (P5_259>0) else "0";
GateOutOpen <= GateOutOpen_Temp;
```

4.1.2.4. Registo de Sincronização dos sinais de entrada

De seguida apresenta-se quatro sinais de entrada assíncronos que dão origem a quatro sinais síncronos. A declaração dos novos sinais é feita através das características dos sinais assíncronos sendo o tamanho e tipo de cada novo sinal síncrono idêntico ao seu sinal assíncrono correspondente (secção 3.1.2).

```
-- Declaração dos sinais
signal GotTicketSync, arriveSync : std_logic_vector(0 downto 0);
signal paySync, leaveSync : std_logic_vector(0 downto 0);

 ...

-- Registo
process (CLK) begin
 if (CLK'event and CLK='1') then
       GotTicketSync <= GotTicket;
       arriveSync <= arrive;
       paySync <= pay;
       leaveSync <= leave;
 end if;
end process;
```

Assim, qualquer alteração exterior dos sinais iniciais irá ser considerada com um atraso de até um ciclo de relógio no restante sistema através dos sinais síncronos.

4.1.2.5. Eventos de entrada

A implementação de eventos é feita utilizando os sinais de entrada já sincronizados.

Para a rede apresentada, os eventos presentes no ficheiro PNML são as seguintes:

```
 ...
<event edge="up" id="ArriveIn" signal="arrive">
<event edge="up" id="Identified" signal="GotTicket">
<event edge="down" id="ArriveOut" signal="arrive">
 ...
```

Pretende-se detectar eventos ascendentes (*up*) e descendentes (*down*) do sinal *arrive* e o evento ascendente do sinal *GotTicket*, ambos binários (*boolean*). Os sinais sincronizados correspondentes são *arriveSync* e *GotTicketSync* (secção 3.1.2).

De seguinda apresentam-se os sinais que têm de ser declarados no ficheiro VHDL para gerar cada evento. O nome do sinal que representa cada evento é obtido a partir do seu *ID* (*ArriveIn*, *Identified* e *ArriveOut*). É também necessário declarar sinais temporários para serem usados na geração dos eventos (secção 3.1.2). Assim, de forma a guardar o sinal de entrada atrasado um ciclo de relógio, são declarados os sinais *arriveSync_temp* e *GotTicketSync_temp* (o nome do sinal a declarar é obtido do nome do sinal síncrono ao qual se acrescenta "_temp").

```
-- Sinais temporários
signal arriveSync_temp : std_logic_vector(0 downto 0);
signal GotTicketSync_temp : std_logic_vector(0 downto 0);

-- Eventos
signal ArriveIn,  Identified : std_logic;  -- Ascendentes
signal ArriveOut : std_logic;              -- Descendentes
```

Todos os eventos gerados são representados por sinais binários pelo que todos serão declarados como sinais *std_logic*. Os sinais temporários têm de ser do mesmo tipo que os sinais a armazenar.

No caso de não existir informação sobre o flanco e *ID* do evento ou não exista informação sobre o sinal físico, o utilizador será avisado através de um erro ou avisos.

4.1.2.6. Constantes para representação dos pesos dos arcos de entrada normais

Os arcos declarados no ficheiro PNML têm de conter informação sobre o seu peso. Segundo o que foi referido em 3.3, estes pesos vão ser representados por constantes inteiras.

Considere-se o excerto de código do ficheiro PNML da rede apresentada em que se mostra um arco normal de peso um.

```
<arc id="319" source="189" target="287">
  <type>normal</type>
  <inscription>
    <value>1</value>
  </inscription>
</arc>
```

O peso correspondente a cada arco pode ser encontrado no sub-elemento *inscription* em *value*.

Caso um arco não tenha atribuído um peso será considerado o valor "1" por omissão e será emitido um aviso a informar o valor assumido, apesar de não haver forma de saber se está ou não correcto.

No ficheiro VDHL a constante que está relacionada com este arco irá aparecer com um nome em função da origem do arco e seu destino. Por exemplo, no caso anterior, o arco tem como origem (*source*) o lugar com *ID* "189" que representa o lugar *P0* e destino (*target*) a transição com *ID* "287" com o nome *T0*. Assim, a constante a ser declarada terá o seguinte aspecto (em que "W" vem de *weight*):

```
constant WP0_189T0_287 : integer := 1;
```

A interpretação da constante declarada pode assim ser traduzida como *o peso do arco com origem em P0 e destino T0 tem peso "1"*.

4.1.2.7. Declaração dos sinais internos do sistema VHDL

4.1.2.7.1. Sinais para representar o número de marcas nos lugares

Em seguida apresenta-se um excerto do código PNML de um lugar da rede em questão, limitado a três marcas e com o nome "P4":

```
<place id="231">
  <name>
    <text>P4</text>
  </name>
  <initialMarking>
    <text>3</text>
  </initialMarking>
  <comment>
    <text>FreePlaces</text>
  </comment>
  <bound>
    <text>3</text>
  </bound>
  <signalOutputActions/>
</place>
```

Em VHDL a declaração de um sinal interno para representar o número de marcas existente no lugar é feita através de um vector *std_logic_vector*. Este sinal terá um número de bits de forma a ser possível representar o número de marcas máximo possível que o lugar terá

durante toda a execução da rede (secção 3.5). Esse valor é obtido a partir do elemento *bound* e que, para a rede em questão, implica um máximo de três marcas para este lugar.
O nome a dar ao sinal será obtido a partir do elemento *name* e do seu *ID* ou caso este não exista, através do *ID* do lugar sendo emitido um aviso.

A sua representação em VHDL é feita da seguinte maneira:

```
signal P4_231 : std_logic_vector(1 downto 0);
```

Caso o valor presente em *bound* não seja válido ou não exista, será considerado por omissão que o lugar é seguro e é emitido um aviso.

Em relação aos comentários existentes no PNML de cada lugar, estes serão colocados também em comentários VHDL antes da declaração de cada lugar.

4.1.2.7.2. Sinais das transições

A representação de habilitações de transições é feita em VHDL por sinais binários (secção 3.4).

Considere-se o seguinte código em PNML que representa a transição *T3* da rede apresentada:

```
<transition id="391">
  <name>
    <text>T3</text>
  </name>
  <comment>
    <text></text>
  </comment>
  <priority>1</priority>
  ...
</transition>
```

A declaração em VHDL do sinal que representa a transição será:

```
signal T3_391 : std_logic;
```

O nome a dar ao sinal de cada transição será igual ao texto contido no elemento *name* seguido do *ID* indicado no ficheiro PNML. Caso o elemento *name* não exista o nome será substituído

pelo seu *ID* e o utilizador é avisado com um aviso. Caso não exista um nome ou um *ID* o utilizador será avisado com um erro.

4.1.2.8. Funções associadas às transições

Para obter as condições de avaliação de uma transição há que ter em conta o estado dos sinais e eventos exteriores de que esta depende, o número de marcas disponível nos lugares de origem, o peso dos arcos e a sua prioridade em relação a outras transições.

Considere-se a seguinte transição da rede em causa:

```
<transition id="391">
   <name>
     <text>T3</text>
   </name>
   <comment>
     <text></text>
   </comment>
   <priority>1</priority>
   <signalInputGuards>
     <signalinputguard>
       <concreteSyntax language="C">
         <text>(pay==1)</text>
       </concreteSyntax>
     </signalinputguard>
     <signalinputguard>
       <concreteSyntax language="VHDL">
         <text>(pay='1')</text>
       </concreteSyntax>
     </signalinputguard>
   </signalInputGuards>
 </transition>
```

Pela observação do código pode ser retirado as seguintes características desta transição:

```
Nome: T3
Prioridade: 1
Condições de habilitação (VHDL): (pay='1')
```

No que respeita à condição de habilitação, e como identifica um sinal de entrada, é necessário alterá-la para *paySync="1"* para que esteja de acordo com o que foi dito na secção 3.1.

É também necessário saber quais são os arcos de entrada da transição, obtendo os pesos associados a cada um deles e os lugares que influenciam a habilitação da transição (apenas é importante garantir as pré-condições de habilitação da transição).

Quando o *ID* da transição se encontra no sub-elemento *source* de um arco implica que estes sejam arcos de saída da transição. No caso de o *ID* da transição se encontrar no sub-elemento *target*, os arcos são de entrada. Neste caso pode observar-se que têm ambos peso igual a um.

Através do seu *ID* (*391*) obtêm-se os seguintes arcos de entrada:

- Arco com origem no lugar *P7* e destino a transição *T3*:

```
<arc id="519" source="245" target="391">
  <type>normal</type>
  <inscription>
    <value>1</value>
  </inscription>
</arc>
```

- Arco com origem no lugar *P3* e destino a transição *T3*:

```
<arc id="2991" source="217" target="391">
  <type>normal</type>
  <inscription>
    <value>1</value>
  </inscription>
</arc>
```

Percorrendo os arcos conclui-se que esta transição tem dois arcos de entrada, *P7* e *P3*.

Caso os lugares ou os arcos não existam (não seja encontrada uma correspondência com o *ID* em causa) o programa deverá informar o utilizador com um erro pois implica a existência de um erro grave na construção da rede.

Para a transição *T3* a declaração em VHDL é feita da seguinte forma:

```
T3_391 <= '1' when (paySync="1" and P7_245>=WP7_245T3_391 and
                  and P3_217>=WP3_217T3_391) else '0';
```

Onde *paySync="1"* é a condição passada pelo projectista depois de alterada para ficar de acordo com o que está definido na secção 3.1. Só será considerada se for exclusivamente escrita em linguagem VHDL (informação fornecida no sub-elemento *language*).

Apesar de a transição estar classificada como sendo de prioridade um, será feita uma avaliação da existência de conflitos. Caso seja encontrado algum conflito é apresentado um aviso ao utilizador.

4.1.2.9. Lugares

Na rede em causa pode ser encontrado o lugar *P0*. A sua representação em PNML pode ser observada de seguida:

```
<place id="189">
  <name>
    <text>P0</text>
  </name>
  <initialMarking>
    <text>0</text>
  </initialMarking>
  <comment>
    <text>GateInOpen</text>
  </comment>
  <bound>
    <text>1</text>
  </bound>
  <signalOutputActions>
    <signalOutputAction idRef="GateInOpen" value="1">
      <concreteSyntax language="C">
        <text> </text>
      </concreteSyntax>
    </signalOutputAction>
    <signalOutputAction idRef="GateInOpen" value="1">
      <concreteSyntax language="VHDL">
        <text> </text>
      </concreteSyntax>
    </signalOutputAction>
  </signalOutputActions>
</place>
```

Através do código apresentado pode ser verificado que o lugar não está marcado inicialmente. Através da característica *bound* conclui-se que o limite de marcas durante a execução da rede será de uma marca, sendo portanto um lugar seguro.

Procurando os arcos cujo *ID* de origem (*source*) é "189" obtém-se no destino (*target*) as transições de destino (ou seja, as que vão destruir marcas quando habilitadas). Se no entanto for realizada uma pesquisa pelos arcos cujo *ID* de destino é "189" obtém-se as transições de entrada que quando habilitadas criam marcas. Em concreto para o lugar *P0* o resultado é que a transição *T2* irá criar marcas em função do peso do arco que os une e transição *T0* destrói marcas também em função do peso do arco que os une.

O código VHDL para este lugar é o seguinte:

```
--P0: GateInOpen
P0_189_temp <= conv_std_logic_vector((conv_integer(P0_189) +
                   + WT2_351P0_189*conv_integer(T2_351)-
                   - WP0_189T0_287*conv_integer(T0_287)),1);

process (CLK, RESET)
begin
       if (RESET = '1') then P0_189 <= "0";
       elsif (CLK'event and CLK='1') then
             P0_189 <= P0_189_temp;
       end if;
end process;
```

O raciocínio para os restantes lugares é idêntico considerando o que foi dito em 3.5.

4.1.2.10. Sinais de saída

No código PNML do lugar apresentado no ponto anterior pode ser observado que o lugar irá gerar um sinal de saída, chamado *GateInOpen*, e que irá ter o valor '1' caso *P0* tenha marcas e '0' se não contiver marcas (valor de omissão do sinal de saída). Segundo o que foi descrito para sinais de saída binários associados a lugares, a sua tradução para VHDL irá ser a seguinte:

```
GateInOpen_Temp <= "1" when (P0_189>0) else "0";
```

Os restantes sinais de saída serão traduzidos de forma idêntica.

4.1.3. Simulação

4.1.3.1. Entrada de carros para o parque de estacionamento

Tomando o código VHDL gerado automaticamente a partir do ficheiro PNML, na Figura 4.2 pode ser observada a simulação efectuada para a entrada de um carro através da única entrada, correspondendo a um ciclo completo desde a chegada do carro à entrada do parque até o controlador estar à espera de outro pedido de entrada.

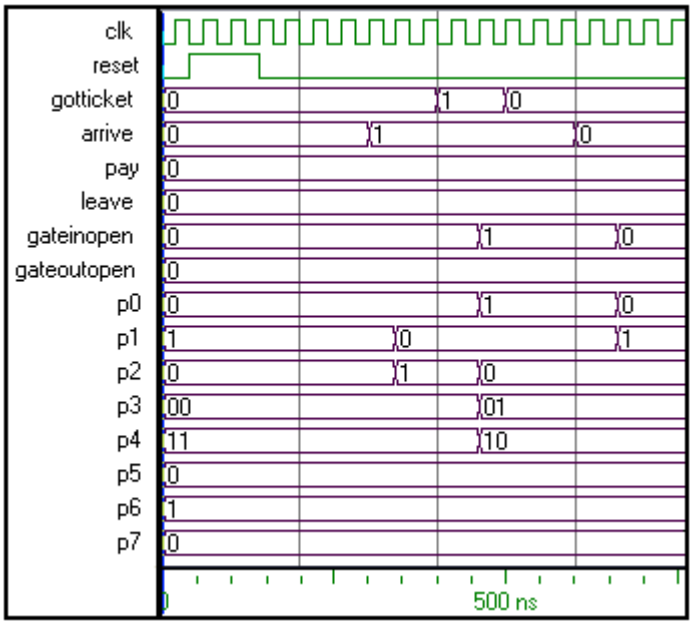

Figura 4.2: Simulação da entrada do parque de estacionamento 1 in 1 out

Tendo em conta o modelo apresentado na Figura 4.1, na Figura 4.2 observa-se a chegada de um carro à entrada, que vai activar o sinal *arrive*. A activação deste sinal gera um evento ascendente que indica a presença do carro na entrada. Nesta fase irá haver troca de marcas entre o lugar *P1* e *P2*, observável pelos sinais *P1* e *P2*. Assim que o condutor retira um "ticket", situação identificada pelo sinal *gotticket*, é gerado um evento *identified* (evento ascendente), habilitando a transição *T2*. Esta transição faz mudar de estado os lugares *P0* e *P1*, da mesma forma que os lugares *P3* e *P4* mudam para actualizar o número de carros presentes no parque (e o número de lugares vagos). Quando o lugar *P0* está marcado há a abertura da "gate" de entrada através do sinal de saída *gateinopen*. Após o sinal *arrive* passar a zero, indicando que o carro já não está na entrada, é gerado um evento descendente *arriveOut* que provoca a mudança de estado dos lugares *P0* e *P1* e consequentemente a mudança do sinal *gateinopen* para zero, fechando a "gate" de entrada.

A repetição deste ciclo até ao esgotamento do número de lugares no parque de estacionamento irá resultar, quando houver um pedido novo de entrada, à seguinte situação apresentada na Figura 4.3.

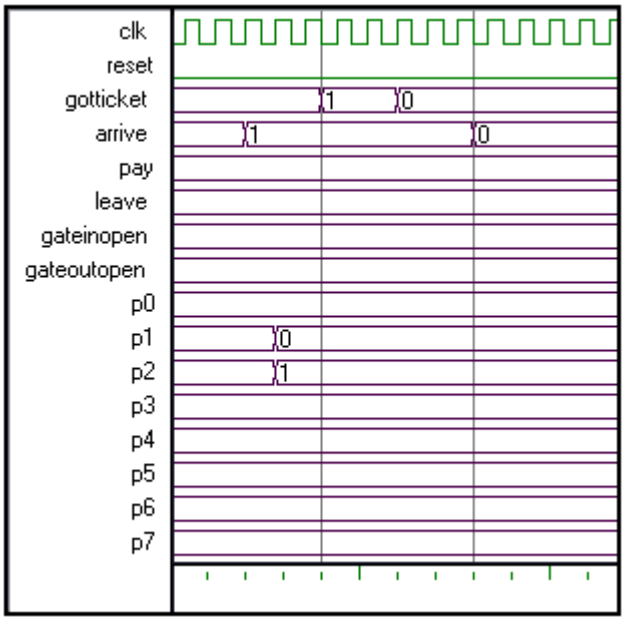

Figura 4.3: Simulação da entrada saturada do parque de estacionamento 1 in 1 out

Pode ser observado que devido à inexistência de marcas no lugar *P4* (ou seja, da inexistência de lugares disponíveis no parque), não há mudança de estado dos lugares *P0*, *P2*, *P3* e *P4* ficando assim a cancela de entrada fechada. Quando sair um carro do parque, libertando um lugar, o condutor já poderá entrar.

4.1.3.2. Saída de carros do parque de estacionamento

Na Figura 4.4 pode ser observada a simulação efectuada para a saída de um carro, correspondendo a um ciclo completo desde a chegada do carro à saída do parque até o controlador estar à espera de outro pedido de saída.

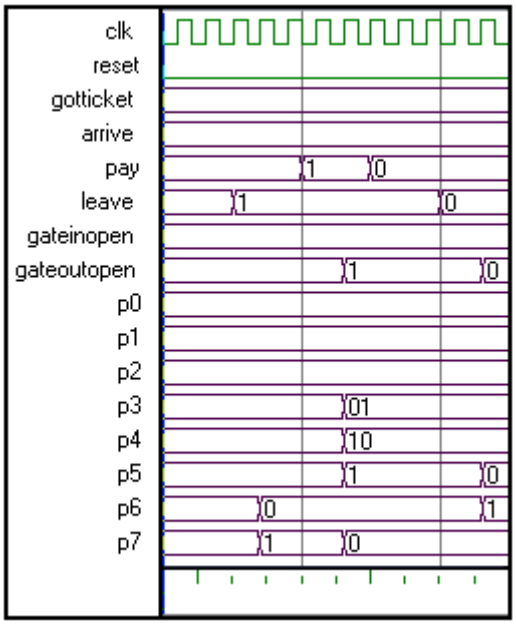

Figura 4.4: Simulação da saída do parque de estacionamento 1 in 1 out

Todo o processo começa com a chegada do carro à saída, representado pelo sinal *leave*, que provoca uma mudança de estado dos lugares *P6* e *P7*. Após a indicação, pelo sinal *pay*, que foi realizado o pagamento existe mudança de estado dos lugares *P5* e *P7* e dos lugares que representam o número de carros no interior do parque e de lugares livres (respectivamente, *P3* e *P4*). Há então a abertura da cancela de saída através da mudança de estado do sinal *gateoutopen*. Após o sinal *leave* passar a zero, indicando que o carro já não se encontra na saída, existe a mudança de estado dos lugares *P5* e *P6* fazendo com que o sinal da cancela de saída vá para zero e fechando esta. Neste estado o controlador está pronto para que outro carro entre na saída do parque.

Caso haja um pedido de saída do parque mas não haja registo que há um carro no seu interior, o controlador irá ficar à espera da entrada de um carro para abrir a cancela de saída. O processo é idêntico ao da entrada e pode ser observado na Figura 4.5 a sua simulação, ficando à espera da presença de marcas no lugar *P3* para que possa abrir a cancela de saída.

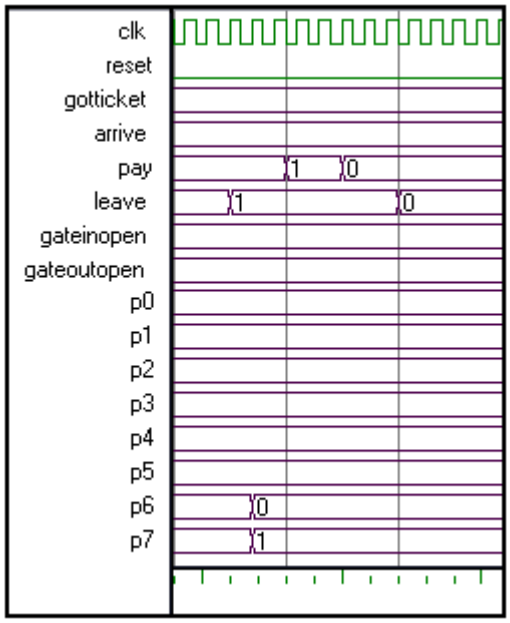

Figura 4.5: Simulação da saída saturada do parque de estacionamento 1 in 1 out

4.1.4. Considerações finais

Com esta secção pretendeu-se demonstrar através de um exemplo a aplicabilidade das regras de tradução apresentadas. Para tal usou-se o exemplo do parque de estacionamento. Após a sua tradução para VHDL, e através de simulações efectuadas no ambiente de simulação da Xilinx [32], é possível verificar o correcto funcionamento em VHDL do modelo do parque de estacionamento. Além das simulações apresentadas foram realizadas outras que se consideraram necessárias para validar a boa tradução, tendo-se verificado o seu correcto funcionamento em todos os testes realizados.

4.2. Parque de estacionamento 2-in 1-out

4.2.1. Descrição do modelo do parque de estacionamento 2-in 1-out

Idênticamente ao ponto anterior, este exemplo apresenta o controlador de um parque de estacionamento. Neste caso é constituído por duas entradas para o parque, moduladas pelos lugares *P8*, *P9* e *P10* e pelas transições *T6*, *T7* e *T8*.

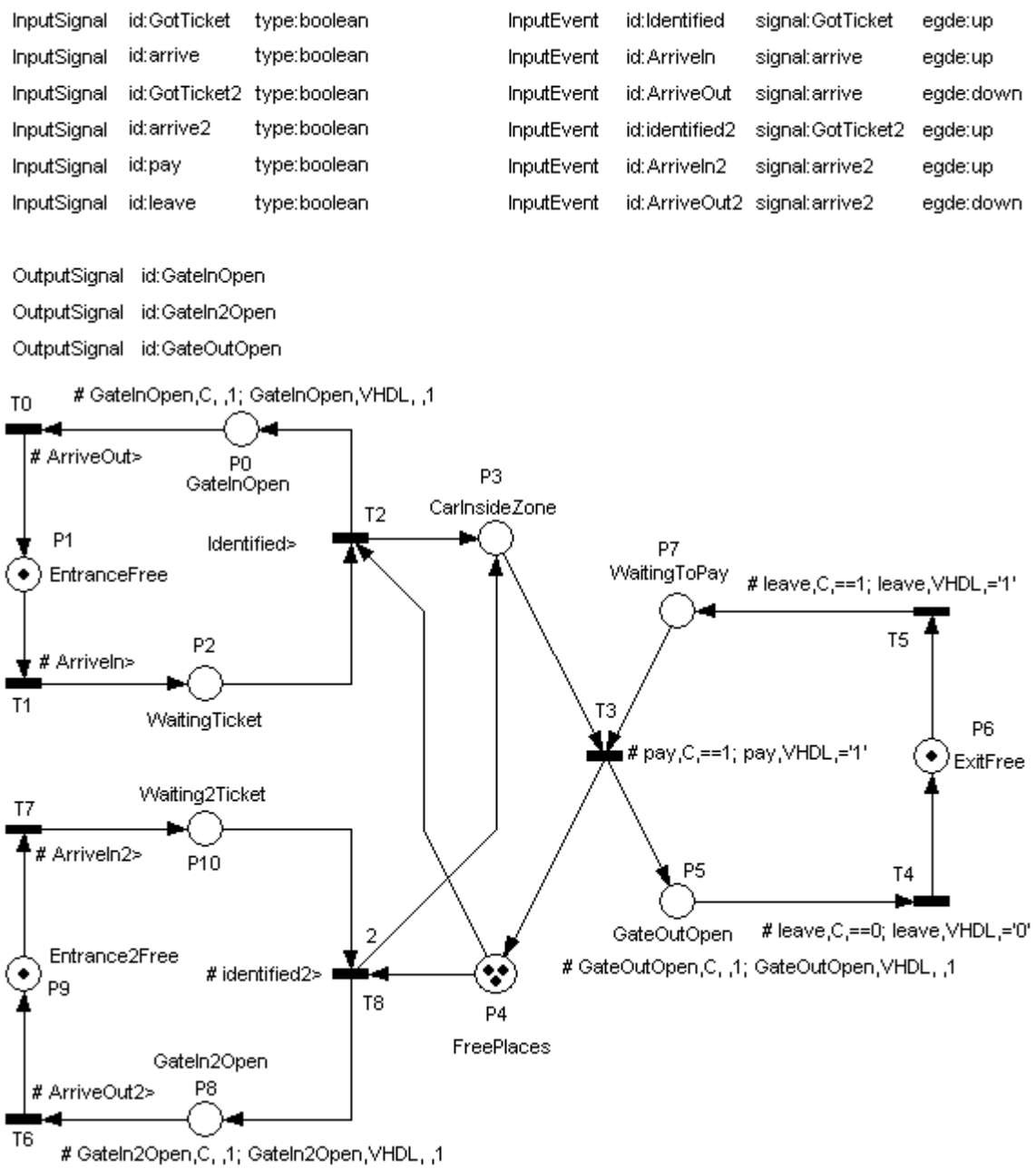

Figura 4.6: RdP de um controlador de entradas e saídas de um parque de estacionamento (duas entradas e uma saída)

Esta segunda entrada tem uma prioridade inferior em relação à primeira, significando que quando se apresentam dois condutores, um em cada entrada, e se o pedido for realizado no mesmo instante e existir apenas um lugar disponível, entra o condutor da primeira entrada ficando o segundo à espera. Neste caso, independentemente de o condutor da segunda entrada estar ou não á espera à mais tempo, um condutor que se apresente na primeira entrada terá sempre prioridade. Este comportamento é modelado através da utilização de prioridades associadas às transições *T2* (maior prioridade) e *T8* (menor prioridade).

No caso de existirem lugares de estacionamento suficientes (ou seja, marcas suficientes no lugar *P4* que indica o número de lugares vazios), os dois condutores podem entrar ao mesmo tempo.

O código completo gerado para a rede em causa está apresentado no Anexo 4.

4.2.2. Tradução

4.2.2.1. Estrutura base do ficheiro VHDL

O nome a atribuir à rede será obtido a partir do seguinte excerto de código (a negrito):

```
<pnml>
    <net id="1" name="park2in1out" type="IOPN">
      <input/>
      <output/>
      ...
```

Assim, neste caso, o nome a atribuir à *entity* e à *architecture* é "park2in1out" e a estrutura base pode ser obtida a partir do exemplo do parque de estacionamento com uma entrada e uma saída.

4.2.2.2. Sinais de entrada

Os sinais de entrada são obtidos a partir do seguinte excerto de código PNML:

```
<input>
      ...
  <signal id="GotTicket" type="boolean">
  <signal id="arrive" type="boolean">
  <signal id="GotTicket2" type="boolean">
  <signal id="arrive2" type="boolean">
  <signal id="pay" type="boolean">
  <signal id="leave" type="boolean">
      ...
</input>
```

A rede é constituída por seis sinais de entrada em que todos são identificados pelo seu *ID* e tipo de sinal. Assim, considerando o que foi dito na secção 3.1.1, a declaração em VHDL dos sinais é feita da seguinte forma:

```
GotTicket : in std_logic_vector(0 downto 0);
arrive : in std_logic_vector(0 downto 0);
GotTicket2 : in std_logic_vector(0 downto 0);
Arrive2 : in std_logic_vector(0 downto 0);
pay : in std_logic_vector(0 downto 0);
leave : in std_logic_vector(0 downto 0);
```

4.2.2.3. Sinais de saída

De seguida é apresentado o excerto do código PNML que representa os sinais de saída da rede de Petri em questão:

```
<output>
  <signal id="GateInOpen" type="boolean" value="0">
  <signal id="GateIn2Open" type="boolean" value="0">
  <signal id="GateOutOpen" type="boolean" value="0">
</output>
```

Existem três sinais de saída, que segundo a secção 3.7.1 quando traduzidos para VHDL dão origem aos seguintes sinais:

```
GateInOpen : out std_logic_vector(0 downto 0);
GateIn2Open : out std_logic_vector(0 downto 0);
GateOutOpen : out std_logic_vector(0 downto 0);
```

Caso o elemento *output* esteja vazio, o programa irá mostrar um aviso para o utilizador. Caso não seja encontrado um *ID* válido, o utilizador será avisado através de um erro.

4.2.2.4. Declaração dos sinais internos do sistema VHDL

4.2.2.4.1. Sinais síncronos de entrada criados a partir dos assíncronos (do mesmo tipo que os assíncronos)

Segundo a secção 3.1.2 ter-se-ão as seguintes declarações:

```
signal GotTicketSync,
       GotTicket2Sync : std_logic_vector(0 downto 0);
signal arriveSync, arrive2Sync : std_logic_vector(0 downto 0);
signal paySync, leaveSync : std_logic_vector(0 downto 0);
```

Não existem diferenças substanciais em relação ao exemplo anterior, tendo sido adicionados os sinais que foram adicionados para a segunda entrada.

4.2.2.5. Transições

Nesta sub-secção pretende-se dar destaque às transições *T2* e *T8* da rede em causa. Durante a execução da rede estas terão situações de conflito. Foi decidido para o presente exemplo que a transição *T2* irá ter prioridade sobre a transição *T8*, implicando que quando acontecer uma situação de conflito efectivo será a *T2* a ser disparada e não a *T8*.

A nível de implementação, e considerando o que foi referido na secção 3.4.5.1, o código VHDL a gerar para a transição *T2* é idêntico ao apresentado no exemplo anterior. No entanto para a transição *T8*, e como esta só será disparada se houver marcas suficientes no lugar de origem (*P4*) para disparar também a transição *T2* ou no caso de *T2* não ser disparada, a declaração em VHDL é feita da seguinte maneira:

```
T8_5424 <= '1' when
    ((identified2='1') and P10_4723>=(WP10_4723T8_5424) and
    P4_231>=(WP4_231T8_5424 + conv_integer(T2_351)*WP4_231T2_351))
    else '0';
```

Assim, verifica-se que para obter a condição de habilitação da transição *T8*, é necessário considerar o número de marcas necessárias para disparar a transição *T2*. Caso *T2* não dispare a sua influência na avaliação para o disparo de *T8* será nula, pois *conv_integer(T2)* é igual a zero.

4.2.3. Simulação

4.2.3.1. Entrada de carros para o parque de estacionamento

A rede deste exemplo contempla duas entradas de carros para o parque de estacionamento. A primeira entrada deste parque, considerando a inexistência de pedidos de entrada na entrada dois, é idêntica ao do parque com uma entrada, descrita em 4.1.3.1. Em relação à segunda entrada o seu funcionamento é também bastante idêntico havendo apenas alterações quando o número de lugares disponveis no interior do parque é apenas um e existe um pedido de

entrada no mesmo instante nas duas entradas. Neste caso a entrada que terá prioridade irá ser a primeira, ficando a segunda a aguardar que haja lugares disponíveis no interior do parque.

Na Figura 4.7 pode ser observada a simulação efectuada para a entrada de um carro na segunda entrada.

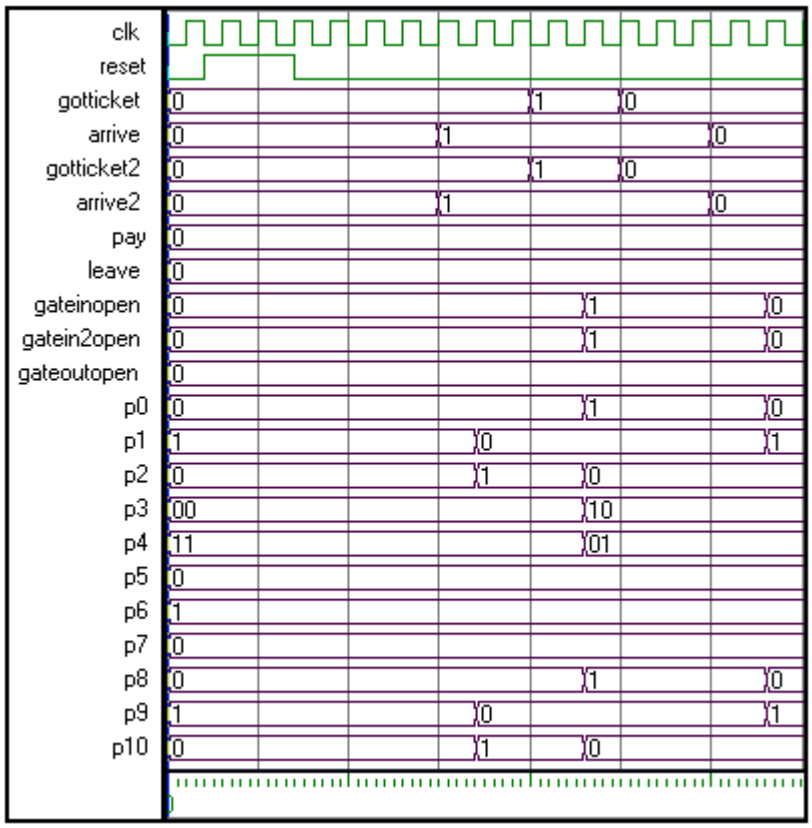

Figura 4.7: Entrada de um carro pela segunda entrada do parque de estacionamento

O seu funcionamento é idêntico ao apresentado no exemplo da secção 4.1.3.1.

Na Figura 4.8 observa-se a existência de um lugar de estacionamento livre no parque. Procede-se à simulação do comportamento do sistema quando existem dois pedidos de entrada, um em cada entrada e no mesmo instante.

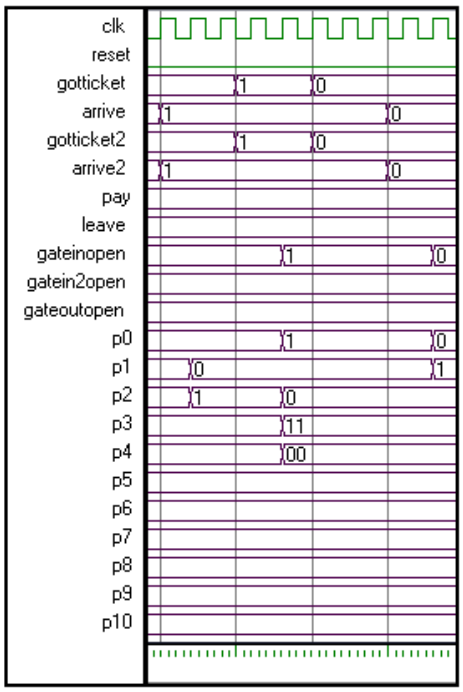

Figura 4.8: Pedido realizado no mesmo instante nas duas entradas (1 lugar vazio)

Como a primeira entrada é a prioritária, o pedido atendido foi o da primeira entrada (observável pela mudança de estado dos lugares *P0* e *P2*). O utilizador da segunda entrada irá ficar a aguardar a existência de um lugar disponível no parque desde que não haja mais pedidos de entrada na primeira entrada.

4.2.3.2. Considerações finais

Neste capítulo foram apresentadas as diferenças significativas em relação à rede anterior. Todos os passos omitidos são idênticos. Pretendeu-se assim dar relevância à resolução de conflitos.

Além das simulações apresentadas foram efectuadas outras com o objectivo de verificar o máximo possível de situações que poderiam acontecer. No fim verificou-se o bom funcionamento da tradução automática feita pelo tradutor implementado.

5. Caracterização da aplicação de tradução

Com este capítulo pretende-se dar a conhecer alguns dos detalhes da implementação da aplicação desenvolvida no âmbito desta dissertação para a geração de código VHDL a partir de especificações de RdP. Pretende-se apresentar a estrutura interna criada para a aplicação, em particular as classes desenvolvidas e a sua função, de forma a permitir uma percepção da forma como a aplicação funciona e como poderá ser integrada em novas aplicações que possam vir a ser desenvolvidas na mesma área de aplicação.

5.1. Introdução

A linguagem de programação utilizada para o desenvolvimento da aplicação foi C++. Esta apresenta-se como preferencial pois disponibiliza algumas classes, existentes na plataforma *.NET*, que podem ser usadas para o desenvolvimento da aplicação, permitindo ainda uma portabilidade para outros ambientes de desenvolvimento.

A classe utilizada para a leitura do *XML* é a presente na plataforma *.NET*. Desta forma, um dos requisitos para o funcionamento da aplicação é que esteja previamente instalado o *Microsoft .NET Framework 2.0* ou superior.

5.2. Análise do Projecto

5.1.1. Principais Funcionalidades do Sistema

O objectivo da aplicação é traduzir uma rede de Petri IOPT para uma linguagem de descrição de hardware VHDL, que simule e execute o seu comportamento. Por outro lado, através da classe IOPT, é possível atribuir sinais e eventos de entrada e saída que vão condicionar o comportamento e evolução da rede. A tradução é directa e realizada em função das regras já apresentadas em capítulo anterior.

Para atingir os objectivos foram implementadas algumas funcionalidades, nomeadamente a abertura de ficheiros PNML e a análise do seu conteúdo de forma a garantir que se encontra a informação necessária à conversão, havendo ainda diversos mecanismos para garantir que a

tradução se realize mesmo quando não há toda a informação. Nestes casos são apresentados avisos ou erros em função do grau de recuperação possível. Esses avisos e erros, além de apresentados na consola, irão ser registados num ficheiro de texto chamado "result.txt" para que seja criado um registo de fácil visualização e disponível após a aplicação terminar.

Do lado do código VHDL garante-se que, se a especificação da rede estiver bem construída, este irá ser construído convenientemente. Caso a rede não esteja bem construída, o código será gerado mas não haverá um controle das suas propriedades, sendo gerados avisos e mensagens de erros.

Foram criadas também uma série de opções para uso através da linha de comandos que permite controlar os sinais de entrada e saída na interface do ficheiro VHDL.

5.1.2. Limitações do Sistema

O tradutor não verifica as propriedades das redes de Petri nem do código VHDL gerado. O seu objectivo é garantir a tradução correcta em função das características apresentadas no ficheiro PNML que caracteriza a rede através da aplicação das regras descritas.

Nos dados passados directamente pelo utilizador, por exemplo nas expressões associadas com a activação/alteração dos sinais de saída em função da marcação de lugares, apenas se garante a passagem directa da condição, cabendo ao utilizador garantir que a condição está correcta para VHDL e sem erros.

Por outro lado, e caso a rede a traduzir seja complexa, pode originar que grandes linhas de código sejam geradas em VHDL devido às condições necessárias para garantir a boa tradução e o seu bom funcionamento. Este facto é minimizado pela divisão do código em várias linhas, pela optimização do código VHDL antes da implementação e pelas excelentes velocidades de processamento que são obtidas hoje no hardware.

5.1.3. Funcionalidades e interacção com o tradutor

Na Figura 5.1 é apresentado um diagrama onde está descrita as formas possíveis de interacção com a aplicação contruída:

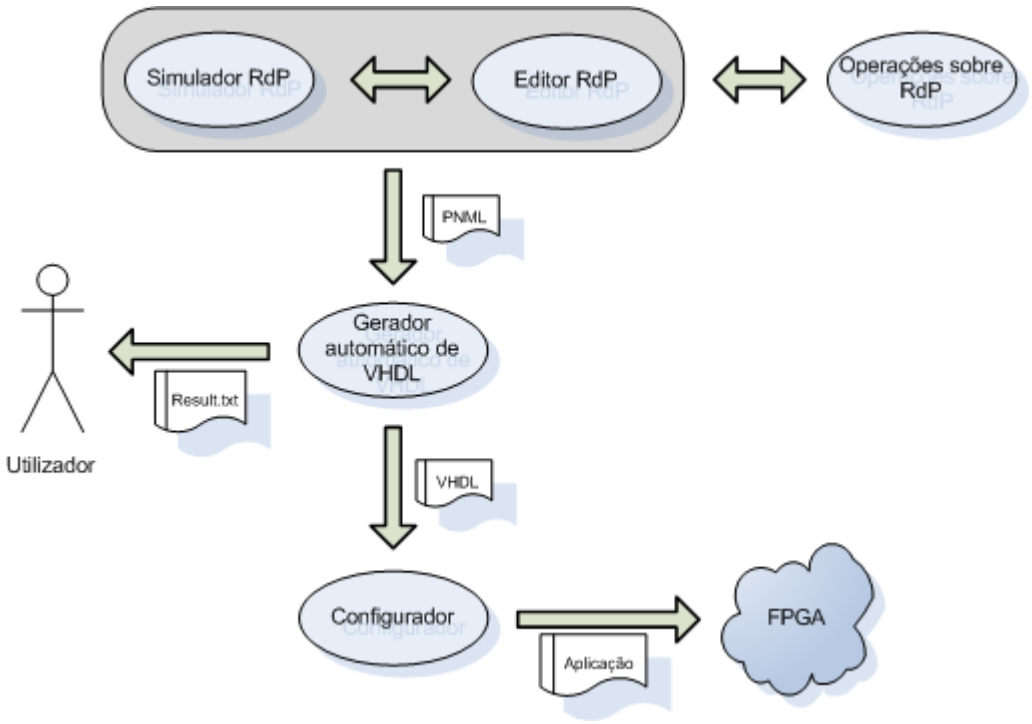

Figura 5.1: Diagrama de funcionalidade e de interacções do tradutor de VHDL

O utilizador poderá interagir com a aplicação directamente pela linha de comandos, ou em alternativa, através da integração com outras aplicações (como por exemplo, o editor de RdP IOPT). Os resultados obtidos poderão ser consultados pelo ficheiro VHDL gerado e pelo ficheiro com o *log* da tradução (*result.txt*).

5.2. Projecto do Sistema

Através de uma cuidada análise para a projecção do tradutor de redes IOPT descritas em PNML para VHDL concluíu-se pela necessidade de determinadas funcionalidades, nomeadamente:

1. Abrir e verificar as propriedades básicas de um ficheiro PNML, em particular para a classe IOPT;

2. Análise da informação lida do ficheiro PNML;
3. Tradução directa baseada em regras de conversão da rede para VHDL;
4. Criação de uma série de avisos e mensagens de erros que levasse o utilizador a descobrir problemas de projecto;
5. Criação de diversas opções para controlar os sinais de entrada e saída da interface do ficheiro VHDL gerado.

Por outro lado, para a criação do ficheiro de PNML, é necessário um editor gráfico de RdP (fora do âmbito desta dissertação) e que permita escrever as características de uma modelo da classe IOPT. Para este caso recomenda-se a utilização do editor *Snoopy* IOPT no âmbito do projecto *FORDESIGN* [4][20].

Na Figura 5.3 pode ser observado o diagrama de fluxo da aplicação construída. Há ainda que ter em consideração o seguinte:

- Para a leitura do ficheiro PNML foram usadas as classes disponíveis no ambiente .NET, nomeadamente *XmlDocument*, *XmlTextReader, XmlNode, XmlElement, XmlAttribute, XmlNamespaceManager, XmlParserContext* e *XPath*;
- A geração do código VHDL é feita sequencialmente para um ficheiro. Todos os dados que são relevantes naquele instante são, se necessário, calculados. Caso mais à frente voltem a ser necessários, serão novamente calculados. Esta repetição de funções não é relevante, pois não se verifica um atraso significativo na conversão;
- A leitura dos dados do ficheiro PNML é realizada para estruturas de dados (Figura 5.2). De seguida apresentam-se as respectivas estruturas e o motivo da sua existência:

    1. Vector *SignalsinputsVector*: Tem a função de conter todos os dados relativos aos sinais de entrada. Cada posição do vector representa um sinal de entrada constituído pela seguinte estrutura e informação:

        - Id: apontador do tipo *char* com a função de identificar o sinal de entrada através de um identificador único;

- Type: apontador do tipo *char* com a função de identificar o tipo de sinal de entrada (*boolean* para sinais binários ou *range* para sinais não binários);
- Max: representa um inteiro e tem a função de indicar o número máximo, em decimal, que este sinal de entrada pode ter;
- Min: representa um inteiro e tem a função de identificar o número mínimo, em decimal, que este sinal de entrada pode ter.

2. Vectores *EventsinputsVector* e *EventsinputsVectorOriginal*: Têm a função de conter todos os dados relativos aos eventos de entrada. A estrutura de cada posição destes dois vectores é constituída pelos seguintes campos:

    - *Edge*: apontador do tipo *char* com a função de identificar o flanco (*up*, *down*, *up-down* ou *down-up*) do evento de entrada;
    - *Id*: apontador do tipo *char* com a função de identificar o evento de entrada através de um identificador único;
    - *Level*: apontador do tipo *char* e tem a função de identificar o valor de referência (*threshold*) do sinal de entrada correspondente para que seja gerado o evento. Será essencialmente importante nos sinais não binários;
    - *Signal*: apontador do tipo *char* com a função de identificar o sinal de entrada que irá ser objecto de avaliação para a geração do evento. O sinal é representado pelo seu *Id*;

    A diferença entre ambos os vectores é que um deles será alterado durante a tradução, mantendo-se o outro com a informação inicial para controlo.

3. Vector *SignalsoutputsVector*: Tem a função de conter toda a informação relativa aos sinais de saída. A estrutura de cada posição deste vector é constituída pelos seguintes campos:

    - *Id*: apontador do tipo *char* com a função de identificar o sinal de saída através de um identificador único;

- *Type*: apontador do tipo *char* para identificar o tipo de sinal de saída (*boolean* para sinais binários ou *range* para sinais não binários);
- *Max*: representa um inteiro e tem a função de indicar o número máximo, em decimal, que este sinal de saída pode ter;
- *Min*: representa um inteiro e tem a função de identificar o número mínimo, em decimal, que este sinal de saída pode ter.
- *Value*: representa um inteiro e tem a função de identificar o valor que o sinal de saída terá por defeito quando houver uma situação de *reset* por exemplo.

4. Vector *EventsoutputsVector*: Tem a função de conter toda a informação necessária dos eventos de saída. A estrutura de cada posição deste vector é constituída pelos seguintes campos:

   - *Edge*: apontador do tipo *char* para identificar o flanco (*up*, *down*, *up-down* ou *down-up*) do evento de saída;
   - *Id*: apontador do tipo *char* com a função de identificar o evento de saída através de um identificador único;
   - *Level*: apontador do tipo *char* e tem a função de identificar o valor de referência (*threshold*) do sinal de saída correspondente para que seja gerado o evento. Será essencialmente importante nos sinais não binários;
   - *Signal*: apontador do tipo *char* para identificar o sinal de saída que irá ser objecto de alteração pela geração do evento. O sinal é representado pelo seu *Id*;

5. Vector *arcsVector*: Tem a função de conter todos os dados relativos aos arcos. A estrutura de cada posição deste vector é constituída pelos seguintes campos:

   - *Id*: apontador do tipo *char* com a função de identificar o arco através de um identificador único;
   - *Source*: apontador do tipo *char* e tem a função de identificar o lugar ou transição de origem do arco através de um *Id*;

- *Target*: apontador do tipo *char* com a função de identificar o lugar ou transição de destino do arco através de um *Id*;
- *Type*: apontador do tipo *char* com a função de identificar o tipo de arco (normal ou de teste);
- *Inscription*: representa um inteiro e tem a função de indicar qual é o peso do arco.

6. Vector *placesVector*: Tem a função de conter toda a informação disponível para a conversão dos lugares. A estrutura de cada posição deste vector é constituída pelos seguintes campos:

    - *Id*: apontador do tipo *char* com a função de identificar o lugar através de um identificador único;
    - *Name*: apontador do tipo *char* e tem a função de indicar qual o nome fornecido ao lugar;
    - *InitialMarking*: representa um inteiro e deverá indicar qual é a marcação inicial do lugar;
    - *Bound*: representa um inteiro e deverá indicar qual é o valor máximo possível que o lugar terá durante a execução da rede;
    - *Comment*: apontador do tipo *char* e irá conter comentários que o utilizador tenha colocado para este lugar.

7. Vector *SignalOutputActionPlaceVector*: Tem a função de conter informação sobre o conjunto de expressões que representam as condições associadas aos sinais de saída e os seus valores respectivos a atribuir ao sinal de saída caso estejam habilitadas. A estrutura de cada posição deste vector é constituída pelos seguintes campos:

    - *PlaceId*: apontador do tipo *char* e indica, pelo *Id*, qual o lugar que está associado a esta condição;
    - *IdRef*: apontador do tipo *char* com a informação sobre qual o sinal de saída que está associado à condição;
    - *Value*: representa um inteiro e indica o valor a atribuir ao sinal de saída indicado caso a expressão se verifique;

- *Language*: apontador do tipo *char* para informar em que linguagem foi escrita a condição. Para que seja considerada no âmbito deste trabalho deverá ser "VHDL" sendo todas as outras ignoradas;
- *Text*: apontador do tipo *char* e deverá conter a condição passada pelo projectista. Caso não contenha informação será considerado por omissão "*marking>0*".

8. Vector *transitionsVector*: Contém a informação necessária à tradução das transições. A estrutura de cada posição deste vector é constituída pelos seguintes campos:

    - *Id*: apontador do tipo *char* com a função de identificar a transição através de um identificador único;
    - *Name*: apontador do tipo *char* que fornece informação sobre o nome atribuído à transição pelo projectista;
    - *Comment*: apontador do tipo *char* que fornece os comentários que foram colocados pelo projectista para identificar esta transição;
    - *Priority*: representa um inteiro e dá-nos a informação, dentro de uma contenda, qual é a prioridade da transição aqui representada.

9. Vector *SignalInputGuardTransitionsVector*: Tem a função de conter todos os dados relativos às condições de habilitação das transições em relação aos sinais de entrada. A estrutura de cada posição deste vector é constituída pelos seguintes campos:

    - *IdTransition*: apontador do tipo *char* que indica qual é o *Id* da transição a que se destina esta condição de habilitação;
    - *Language*: apontador do tipo *char* que informa para que linguagem foi escrita a condição. Apenas será considerada no âmbito deste trabalho caso seja "VHDL";
    - *Text*: apontador do tipo *char* que indica qual é a condição que foi passada pelo projectista.

10. Vector *InputEventsTransitionsVector*: Tem a função de conter todos os dados relativos às condições de habilitação das transições no que respeita aos eventos de entrada. A estrutura de cada posição deste vector é constituída pelos seguintes campos:

    - *IdTransition*: apontador do tipo *char* que indica qual é o *Id* da transição a que se destina esta condição de habilitação;
    - *IdRef* apontador do tipo *char* com a informação sobre a qual evento de entrada está associada a condição.

11. Vector *OutputEventsTransitionsVector*: Tem a função de conter todos os dados relativos da associação dos eventos de saída às transições que os habilitam. A estrutura de cada posição deste vector é constituída pelos seguintes campos:

    - *IdTransition*: apontador do tipo *char* que indica qual é o *Id* da transição que habilita o evento de saída;
    - *IdRef* apontador do tipo *char* com a informação sobre qual evento de saída está associada a transição.

Na Figura 5.2 é apresentado o ER das tabelas onde pode ser observado a forma como todas as estruturas de dados anteriores se relacionam. A informação presente no PNML será colocada nestas estruturas de dados após a leitura do ficheiro.

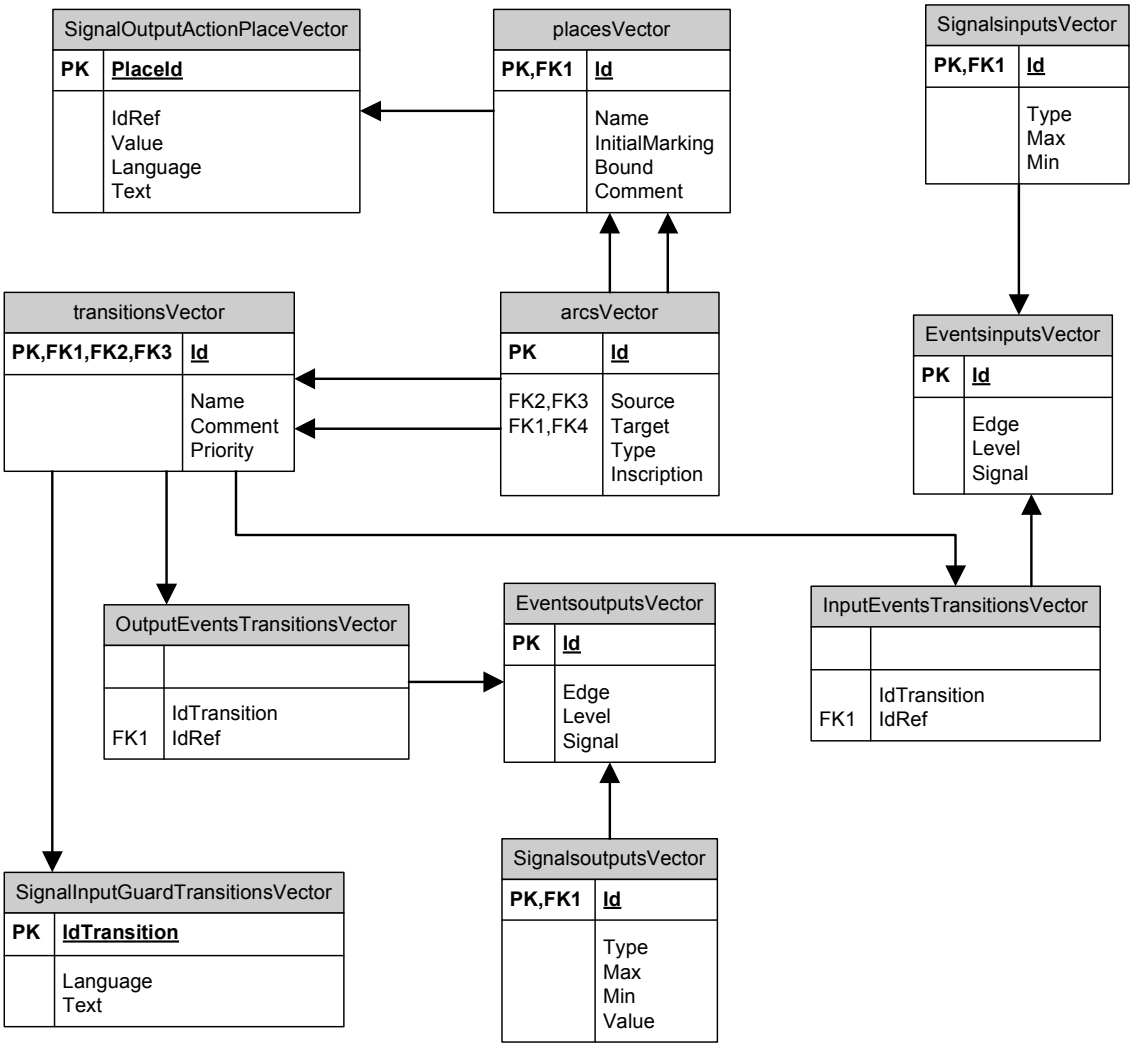

Figura 5.2: *Entity-Relationship Model* das tabelas

Por fim, na Figura 5.3 apresenta-se o fluxo definido para a construcção da aplicação de tradução.

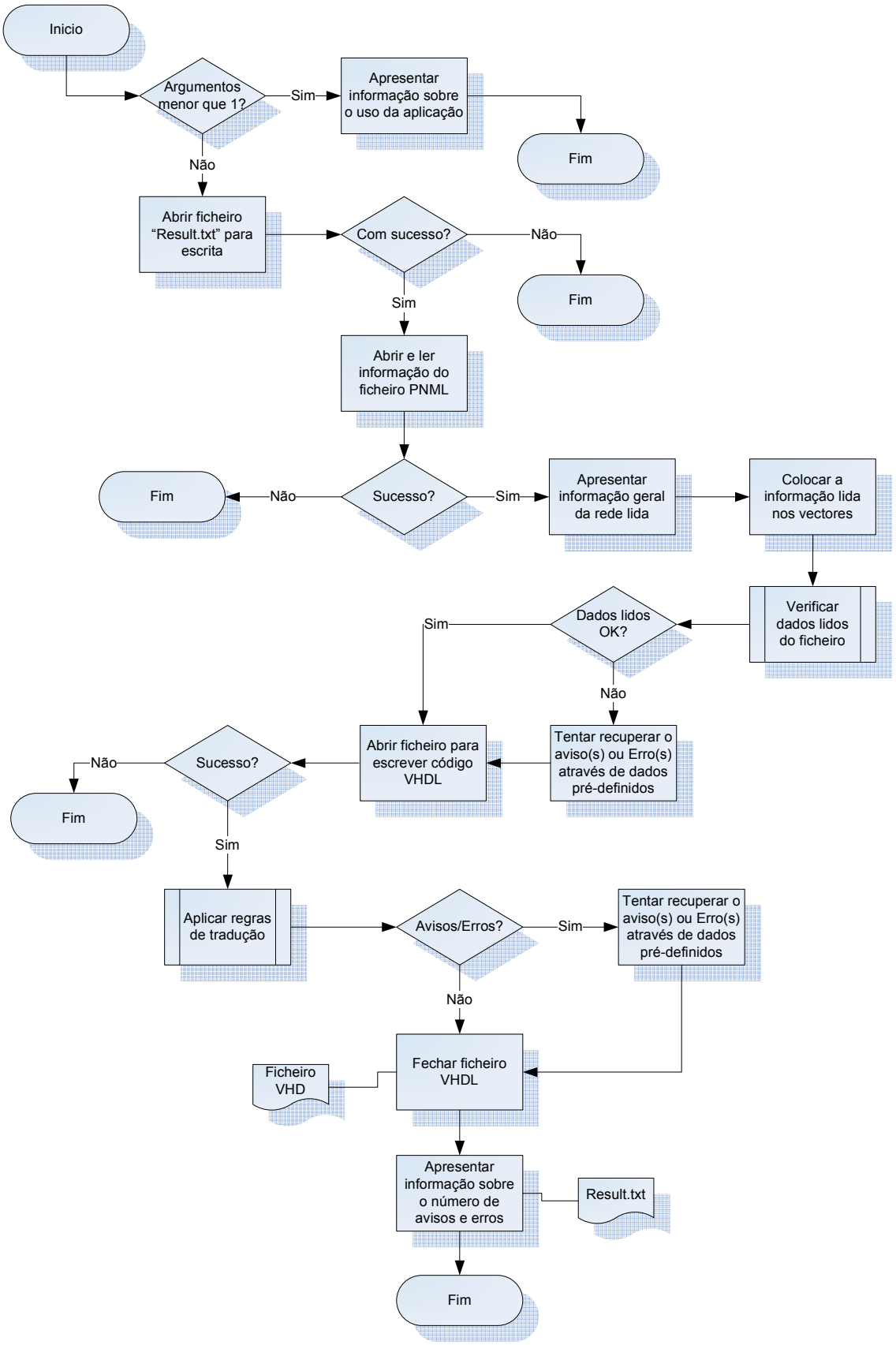

Figura 5.3: Fluxo de dados da aplicação de tradução

6. Conclusões

O objectivo deste trabalho foi criar regras de tradução que permitam a tradução de redes de Petri descritas através da classe IOPT para VHDL. Para cada característica da rede IOPT fez-se coincidir uma regra de tradução para VHDL, gerando no final de uma tradução um sistema síncrono.

A escolha de RdP baseou-se no facto de estas terem a capacidade de modelar sistemas com actividades paralelas e concorrentes, bem como de existirem técnicas de análise das suas propriedades, tornam-as especialmente adequadas para a modelação, análise e simulação de sistemas de eventos discretos. Por outro lado, a escolha da classe de redes de Petri IOPT foi baseada no facto de esta permitir a especificação de sinais e eventos, ambos de entrada e saída, bem como a resolução de conflitos entre as transições.

Considerando que o trabalho incide sobre controladores síncronos, houve a necessidade de introduzir um sinal de sincronização de forma a garantir que a análise das condições de habilitação das transições e a actualização do número de marcas dos lugares seja feito instantaneamente num determinado período de tempo.

O desenvolvimento do trabalho começou por verificar como poderia ser realizada a conversão das estruturas básicas de uma rede (lugares, transições e arcos). Após a validação das estruturas criadas documentou-se e passou-se à fase seguinte de criar também regras de tradução para os sinais e eventos de entrada e de saída, procedendo-se à sua posterior documentação.

Após a obtenção das regras de tradução, o foco central deste trabalho foi o de desenvolver um tradutor automático que fosse capaz de gerar código VHDL a partir de especificações de Redes de Petri escritas em PNML usando a classe IOPT. Após a implementação do tradutor, tendo em atenção a optimização do código VHDL construído (garantindo por exemplo que não existe repetição de sinais que já tenham as mesmas características, usando os já existentes), o tradutor foi testado e verificado com diversos casos de uso de forma a validar o código gerado. Assim, de acordo com os testes efectuados ao tradutor, pode ser afirmado que foi atingido o objectivo com sucesso.

No tradutor foram ainda implementados algumas condições de verificação do código gerado. O principal objectivo é garantir que existe sempre uma tradução razoável e funcional da descrição da rede, havendo diversos mecanismos (por exemplo, valores de omissão) para que o ficheiro VHDL gerado seja válido (apesar de haver o risco de, com a assumpção de valores de omissão, não corresponder à rede pretendida). Apenas em situações em que os dados lidos do ficheiro PNML são incorrectos ou não são suficientes é que o código VHDL criado pode não funcionar.

Foram também disponibilizadas diversas opções através da linha de comandos que permitem controlar os sinais de entrada e saída presentes na *entity* do ficheiro VHDL criado. Estas opções permitem uma maior agilidade da tradução.

No que respeita a trabalho futuro, é possível encontrar diversas opções interessantes que podem ser acrescentadas à aplicação em causa, nomeadamente:

- Optimização do código gerado, pois o código VHDL gerado através de ferramentas, quando comparado com o código gerado manualmente, pode ser optimizado (menor performance, maior e mais lento);
- Desenvolvimento de uma interface gráfica para que esta ferramenta possa ser usada de forma independente. A interface gráfica terá disponível todas as opções da linha de comandos, de forma a facilitar a interacção entre o utilizador e a aplicação;
- Exploração da possibilidade de a aplicação ser usada como aplicação de suporte para simuladores através do uso de hardware e a construcção do espaço de estados associado à rede.

A opção de tradução da RdP para um controlador assíncrono é também uma situação a ser explorada. Através de uma opção na aplicação, o utilizador poderá optar por uma implementação síncrona ou assincrona.

Finalmente, e após uma analise das intenções apresentadas na introdução, acreditamos que ao disponibilizarmos o tradutor *PNML2VHDL* estamos a dar um passo importante na direcção de modelação de circuitos digitais através de Redes de Petri.

Referências Bibliográficas

[1] Luís Gomes, "Redes de Petri e Sistemas Digitais", 1999, http://moodle.fct.unl.pt/file.php/978/docs/Parte_2/RdPSDv12.pdf

[2] Stefan Sjoholm, Lennart Lindh, "VHDL for Designers", Prentice Hall, 1997, ISBN 0-13-473414-9

[3] Jason Smith, "Metastability", March 2002, http://www.inf.ufrgs.br/~fglima/meta-estabilidade.pdf

[4] L. Gomes, J. Barros, A. Costa, and R. Nunes, "The Input-Output Place-Transition Petri Net Class and Associated Tools", in *Proceedings of the IEEE International Conference on Industrial Informatics (INDIN 2007)*. IEEE, July 2007, ISBN 978-1-4244-0851-1.

[5] D. Structures and S. D. B. U. of Technology Cottbus, "S N O O P Y ' s homepage", 2007, http://www-dssz.informatik.tu-cottbus.de/software/snoopy.html.

[6] W. Reisig, *Petri nets: an Introduction*. Springer-Verlag New York, Inc., 1985, ISBN 0387137238.

[7] R. David and H. Alla, *Petri Nets & Grafcet; Tools for Modelling Discrete Event Systems*. Prentice Hall International (UK) Ltd, 1992, ISBN 013327537X.

[8] M. Silva, *Las Redes de Petri: en la Automática y la Informática*. Madrid: Editorial AC, 1985.

[9] G. Frey and M. Minas, "Editing, Visualizing, and Implementing Signal Interpreted Petri Nets", in *Proceedings of the AWPN 2000, Koblenz*, Oct. 2000, pp. 57–62.

[10] H.-M. Hanisch and A. Lüder, "A Signal Extension for Petri Nets and its Use in Controller Design", *Fundamenta Informaticae*, vol. 41, no. 4, pp. 415–431, 2000.

[11] L. Gomes, “On Conflict Resolution in Petri Nets Models Through Model Structuring and Composition”, in *3rd IEEE International Conference on Industrial Informatics (INDIN 2005).*, Aug. 2005.

[12] R. Pais, J. Barros, and L. Gomes, “A Tool for Tailored Code Generation from Petri Net Models”, in *10th IEEE Conference on Emerging Technologies and Factory Automation (ETFA 2005)*, Sep. 2005.

[13] Mhamed Itmi, “Designing Generalized Petri Nets”, s.d..

[14] René David, “Modeling of Dynamic Systems by Petri Nets”, ECC 91 European Control Conference, Grenoble, France, Julho 1991.

[15] Tadao Murata, “Petri Nets: Properties, Analysis and Applications”, *Proceedings of the IEEE*, Vol. 77, No. 4, Abril 1989.

[16] James L. Peterson, “Petri Nets”, *Computing Surveys*, Vol. 9, No. 3, Setembro 1977.

[17] Robert Sedgewick, *Algorithms*, segunda edição, Addison-Wesley Publishing Company, 1988.

[18] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, *Introduction to Algorithms*, second printing, MIT Press, 1990, ISBN 0262531968.

[19] Luís Filipe dos Santos Gomes, *Especificação de sistemas digitais*, trabalho de síntese, submetido para a prestação de provas de aptidão pedagógica e capacidade científica, Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, Novembro 1991.

[20] L. Gomes, J. P. Barros, A. Costa, R. Pais, and F. Moutinho, “Towards Usage of Formal Methods within Embedded Systems Co-Design”, in *Proceedings of the 2005 IEEE Conference on Emerging Technologies and Factory Automation (ETFA 2005).* IEEE, September 2005.

[21] “RELAX NG Specification – Committee Specification 3 December 2001”, http://relaxng.org/spec-20011203.html, 2001.

[22] S. Christensen and N. D. Hansen, “Coloured Petri Nets Extended with Channels for Synchronous Communication”, *Daimi PB-390*, 1992, also in: Valette, R.: Lecture Notes in Computer Science, Vol. 815; Application and Theory of Petri Nets 1994, Proc. 15th International Conference, Zaragoza, Spain, pages 159-178. Springer-Verlag, 1994.

[23] Jungel M., Kindler E., Weber M., “The Petri Net Markup Language”, HU Berlin: Institut für Informatik (03 mar. 2003), http://www.informatik.hu-berlin.de/top/pnml/download/PNML_LNCS.pdf

[24] J. P. Barros, “CpPNeTS: uma Classe de Redes de Petri de Alto-nível”, Tese de Mestrado, Departamento de Informática, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, 1996, http://www.estig.ipbeja.pt/~jpb/BarrosTeseDeMestrado.pdf

[25] R. Nunes, L. Gomes, J. P. Barros, “A graphical editor for the input-output place-transition petri net class”, Emerging Technologies and Factory Automation, 2007, Univ. Nova de Lisboa, Lisbon, ISBN 978-1-4244-0825-2, http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?isnumber=4416743&arnumber=4416858&count=223&index=114

[26] José Alberto Baptista Tomé, “Controlo de Sistemas Digitais”, Editorial Presença – Colecção Informática e Computadores, 1989

[27] J. M. Fernandes, M. Adamski, A. J. Proença: VHDL Generation from hierarchical Petri Net Specifications of Parallel Controllers, IEE Proceedings: Computer and Digital Techniques (1997)

[28] E. Soto, M. Pereira, Implementing a Petri net Specification in a FPGA using VHDL, Em M. Adamski, M. Wegrzyn, Proceedings of the International Workshop on Discrete-Event System Design – DESDes01 (2001)

[29] N. Marranghello, J. Mirkowski, K. Bilinski, Synthesis of synchronous Digital Systems Specified by Petri Nets, Em Hardware Design and Petri Nets, Kluwer Acabdemic Publishers (2000)

[30] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, A. Yakovlev, Petrify: A Tool for Manipulating Concorrent Specifications and Synthesis os Asynchronous Controllers, IEICE Transactions on Information and Systems E80-D (1997)

[31] G. Frey, M. Minas, Editing, Visualizing, and Implementing Signal Interpreted Petri nets, Em Proceedings of the AWPN 2000, Koblenz (2000)

Ligações de Internet

[32] Xilinx Website:
Site: http://www.xilinx.com

[33] Altera Website:
Site: http://www.altera.com

[34] Petri Net World Website (Hamburgo):
Site: http://www.informatik.uni-hamburg.de/TGI/PetriNets

[35] http://www.uninova.pt/gres/iopt/iopt.pntd

[36] W3C – World Wide Web Cconsortium, Extensible Markup Language (XML),
Site: http://www.w3.org/XML

[37] Aplicação ARP:
Site: http://www.ppgia.pucpr.br/~maziero/diversos/petri/arp.html

[38] Aplicação JARP – Petri Net Analyzer:
Site: http://jarp.sourceforge.net

[39] Aplicação HPSim:
Site: http://www.winpesim.de

[40] Aplicação PIPE – Platform Independent Petri net Editor 2:
Site: http://pipe2.sourceforge.net

[41] Aplicação Petri-LLD:
Site: http://petrilld.sourceforge.net

[42] Aplicação Snoopy:
Site: http://www-dssz.informatik.tu-cottbus.de

[43] Institute of Electrical and Electronics Engineers (IEEE):
Site: http://www.ieee.org

[44] Aplicação Petrify:
Site: http://www.lsi.upc.es/~jordicf/petrify/home.html

[45] Aplicação SIPN-Editor:
Site: http://www.eit.uni-kl.de/litz/ENGLISH/software/SIPNEditor.htm

Anexos

Anexo 1

Manual de utilização da aplicação

A utilização da aplicação desenvolvida é intuitiva. A tradução pode ser feita através da linha de comandos ou através de uma outra aplicação. O utilizador deverá colocar o nome do ficheiro executável (PNML2VHDL) e à frente o nome do ficheiro a converter, podendo colocar mais que um ficheiro. Por exemplo:

```
PNML2VHDL park1in1out.pnml park2in1out.pnml
```

Estão disponíveis um conjunto de opções, todas facultativas, de forma a controlar a interface do ficheiro VHDL a gerar. Essas opções são as seguintes:

- **trn**: Adiciona as transições que vão disparar no próximo ciclo de relógio;
- **trp**: Adiciona as transições que dispararam no último ciclo de relógio;
- **sim**: Adiciona uma condição exterior às transições de forma a condicionar o seu disparo;
- **pla**: Adiciona a marcação de lugares (estado actual);
- **pln**: Adiciona a marcação de lugares que irá estar presente após acontecer o próximo ciclo de relógio;
- **sis**: Adiciona o estado dos sinais de entrada sincronizados;
- **evo**: Adiciona os eventos de saída (estado actual);
- **evi**: Substitui os eventos criados internamente (estado actual) por eventos passados pelo utilizador;
- **eis**: Adiciona o estado dos eventos de entrada no próximo ciclo de relógio.

Para remoção de referências na *entity* do código gerado estão também disponíveis as seguintes opções:

- **sir**: Permite remover da *entity* um sinal de entrada que não seja utilizado no restante código VHDL;
- **sor**: Permite remover da *entity* um sinal de saída que não seja utilizado no restante código VHDL;

Estão também disponíveis três opções de controlo de directorias, todas elas facultativas:

- **dir**: Define a directoria de origem do PNML e de destino do VHDL (se activa, as próximas duas não serão consideradas);
- **dirfr**: Directoria de origem do ficheiro PNML;
- **dirto**: Directoria de destino do ficheiro VHDL.

No caso de os parâmetros não estarem correctos irá aparecer a seguinte informação no ecrã a explicar como deverá ser chamada a aplicação:

```
PNML2VHDL Translator v1.2 Beta

Number of arguments invalid.

Usage:
PNML2VHDL PNMLFile1.pnml [PNMLFile2.pnml PNMLFile3.pnml ...] [Options]

[Options] are:
-trn: It adds the transitions that fire in the next clock.
-trp: It adds the transitions that fired in the last clock.
-sim: It adds a condition to transitions that fired in the next clock.
-pla: It adds the marking of places (current marking).
-pln: It adds the marking of places after the next clock.
-sis: It adds the input signals state synchronized with clock.
-sir: It removes the input signals from entity.
-sor: It removes the output signals from entity.
-evo: It adds the output events (current state).
-evi: Input events come from outside of VHDL file (current state).
-eis: It adds the input events state in the next clock.

-dir: It defines the PNML directory (from) and VHDL directory (to).
        If this option is used, the next two doesn't matter.
-dirfr: PNML directory (from).
-dirto: VHDL directory (to).


Additional information about PNML2VHDL can be found at
http://www.uninova.pt/fordesign/PNML2VHDL.htm

Or contacting:
        Paulo Lima (pnml2vhdl@yahoo.com)
        Luis Gomes (lugo@fct.unl.pt)

Suggestions and/or comments are welcome.

Press any key to continue . . .
```

A aplicação pode mostrar informação sobre avisos (*warnings*) ou erros que possam existir na informação lida do ficheiro ou durante a tradução.

Neste contexto, sempre que não esteja disponível a informação necessária para a conversão ou haja necessidade de supor algum valor por omissão, a aplicação irá avisar o utilizador através de um aviso na consola e no ficheiro de resultados ("Result.txt"). Todos os valores assumidos serão informados no aviso.

Os erros serão também mostrados na consola e no ficheiro de resultados. Estes aparecem essencialmente nos casos em que a informação em falta é essencial à tradução.

Em ambos os casos a aplicação irá continuar a conversão. No caso de haver avisos, garante-se que a tradução efectuada é válida, não garantindo no entanto o funcionamento desejado (pois assumem-se valores por omissão). No caso de haver erros, o funcionamento da rede não é garantido.

No Anexo 2 pode ser observada a lista completa de avisos e erros disponíveis e a sua respectiva interpretação.

O tradutor foi desenvolvido com o objectivo de ser flexível, tentando criar sempre o código VHDL mesmo que haja incoerências na rede lida do ficheiro PNML.

Detalhe das opções da aplicação

Para modificação dos sinais presentes na entity do ficheiro VHDL gerado estão disponíveis as seguintes opções:

- **trn**: Adiciona, na *entity* do ficheiro VHDL gerado, sinais que permitem verificar o estado das transições que vão disparar no próximo ciclo de relógio. De seguida é possível verificar a negrito o código VHDL produzido quando esta opção é utilizada. A forma como é realizada a tradução das diversas condições para cada transição pode ser consultada na secção 3.4.

```
entity NetName is
    Port (
            -- General Signals
            CLK : in std_logic;
            RESET : in std_logic;

            T1 : out std_logic;
            T2 : out std_logic;
```

```
            ...
            Tn : out std_logic;
            ...
            );
end NetName;

architecture Behavioral of NetName is

-- Sinais internos das transições
signal T1_Id, T2_ Id, ..., Tn_ Id : std_logic;

...

begin

-- Transições
T1_Id <= '1' when (Condições) else '0';
T2_Id <= '1' when (Condições) else '0';
...
Tn_Id <= '1' when (Condições) else '0';


T1 <= T1_Id;
T2 <= T2_Id;
...
Tn <= Tn_Id;

end Behavioral;
```

- **trp**: Adiciona, na *entity* do ficheiro VHDL gerado, sinais que permitem verificar qual foi o estado das transições no ciclo de relógio anterior. De seguida é possível verificar a negrito o código VHDL produzido quando esta opção é utilizada. A forma como é realizada a tradução das diversas condições para cada transição pode ser consultada na secção 3.4.

```
entity NetName is
    Port (
            -- General Signals
            CLK : in std_logic;
            RESET : in std_logic;

            T1Past : out std_logic;
            T2Past : out std_logic;
            ...
            TnPast : out std_logic;
            ...
            );
end NetName;

architecture Behavioral of NetName is

signal T1_Id, T2_Id, ..., Tn_Id : std_logic;

...
```

```
begin

-- Transições
T1_Id <= '1' when (Condições) else '0';
T2_Id <= '1' when (Condições) else '0';
...
Tn_Id <= '1' when (Condições) else '0';

process (CLK)
begin
      if (CLK'event and CLK='1') then
            T1Past <= T1_Id;
            T2Past <= T2_Id;
            ...
            TnPast <= T2_Id;
      end if;
end process;

end Behavioral;
```

- **sim**: Adiciona, na *entity* do ficheiro VHDL gerado, sinais de entrada que representam condições exteriores que permitem condicionar o disparo de transições. De seguida é possível verificar a negrito o código VHDL produzido quando esta opção é utilizada. A forma como é realizada a tradução das diversas condições para cada transição pode ser consultada na secção 3.4.

```
entity NetName is
    Port (
            -- General Signals
            CLK : in std_logic;
            RESET : in std_logic;

            T1_Sim : in std_logic;
            T2_Sim : in std_logic;
            ...
            Tn_Sim : in std_logic;
            ...
            );
end NetName;

architecture Behavioral of NetName is

signal T1_Id, T2_Id, ..., Tn_Id : std_logic;

...

begin

-- Transições
T1_Id <= '1' when (Condições and T1_Sim='1') else '0';
T2_Id <= '1' when (Condições and T2_Sim='1') else '0';
...
Tn_Id <= '1' when (Condições and Tn_Sim='1') else '0';

end Behavioral;
```

- **pla**: Adiciona, na *entity* do ficheiro VHDL gerado, sinais que permitem verificar a marcação actual de cada lugar presente na rede traduzida. De seguida é possível verificar a negrito o código VHDL produzido quando esta opção é utilizada. A forma como é realizado o cálculo do número de marcas em cada lugar pode ser consultada na secção 3.5.

```
entity NetName is
    Port (
            -- General Signals
            CLK : in std_logic;
            RESET : in std_logic;

            P1 : out std_logic_vector(N1 downto 0);
            P2 : out std_logic_vector(N2 downto 0);
            ...
            Pn : out std_logic_vector(Nn downto 0);
            ...
            );
end NetName;

architecture Behavioral of NetName is

signal P1_Id, P1_IdTemp : std_logic_vector(N1 downto 0);
signal P2_Id, P2_IdTemp : std_logic_vector(N2 downto 0);
...
signal Pn_Id, Pn_IdTemp : std_logic_vector(Nn downto 0);

...

begin

P1_IdTemp <= “Cálculo do número de marcas”;
process (CLK, RESET) begin
      if (RESET = '1') then P1_Id <= “Marcas Iniciais”;
      elsif (CLK'event and CLK='1') then
            P1_Id <= P1_IdTemp;
      end if;
end process;

P2_IdTemp <= “Cálculo do número de marcas”;
process (CLK, RESET) begin
      if (RESET = '1') then P2_Id <= “Marcas Iniciais”;
      elsif (CLK'event and CLK='1') then
            P2_Id <= P2_IdTemp;
      end if;
end process;

...

Pn_IdTemp <= “Cálculo do número de marcas”;
process (CLK, RESET) begin
      if (RESET = '1') then Pn_Id <= “Marcas Iniciais”;
      elsif (CLK'event and CLK='1') then
            Pn_Id <= Pn_IdTemp;
      end if;
end process;
```

```
P1 <= P1_Id;
P2 <= P2_Id;
...
Pn <= Pn_Id;

end Behavioral;
```

- **pln**: Adiciona, na *entity* do ficheiro VHDL gerado, sinais que permitem verificar a marcação que irá estar em cada lugar após o próximo ciclo ascendente do relógio. De seguida é possível verificar a negrito o código VHDL produzido quando esta opção é utilizada. A forma como é realizado o cálculo do número de marcas em cada lugar pode ser consultada na secção 3.5.

```
entity NetName is
    Port (
            -- General Signals
            CLK : in std_logic;
            RESET : in std_logic;

            P1Next : out std_logic_vector(N1 downto 0);
            P2Next : out std_logic_vector(N2 downto 0);
            ...
            PnNext : out std_logic_vector(Nn downto 0);
            ...
            );
end NetName;

architecture Behavioral of NetName is

signal P1_Id, P1_IdTemp : std_logic_vector(N1 downto 0);
signal P2_Id, P2_IdTemp : std_logic_vector(N2 downto 0);
...
signal Pn_Id, Pn_IdTemp : std_logic_vector(Nn downto 0);

...

begin


P1_IdTemp <= “Cálculo do número de marcas”;
process (CLK, RESET) begin
      if (RESET = '1') then P1_Id <= “Marcas Iniciais”;
      elsif (CLK'event and CLK='1') then
            P1_Id <= P1_IdTemp;
      end if;
end process;

P2_IdTemp <= “Cálculo do número de marcas”;
process (CLK, RESET) begin
      if (RESET = '1') then P2_Id <= “Marcas Iniciais”;
      elsif (CLK'event and CLK='1') then
            P2_Id <= P2_IdTemp;
      end if;
end process;

...
```

```
Pn_IdTemp <= “Cálculo do número de marcas”;
process (CLK, RESET) begin
      if (RESET = '1') then Pn_Id <= “Marcas Iniciais”;
      elsif (CLK'event and CLK='1') then
            Pn_Id <= Pn_IdTemp;
      end if;
end process;


P1Next <= P1_IdTemp;
P2Next <= P2_IdTemp;
...
PnNext <= Pn_IdTemp;

end Behavioral;
```

- **sis**: Adiciona, na *entity* do ficheiro VHDL gerado, sinais que permitem verificar o estado dos sinais de entrada após passar pelo registo de sincronização. De seguida é possível verificar a negrito o código VHDL produzido quando a opção é utilizada. Pode ser obtida mais informação na secção 3.1.

```
entity NetName is
    Port (
            -- General Signals
            CLK : in std_logic;
            RESET : in std_logic;

            InputSig1 : in std_logic_vector(N1 downto 0);
            InputSig2 : in std_logic_vector(N2 downto 0);
            ...
            InputSign : in std_logic_vector(Nn downto 0);


            InputSig1SyncOut : out std_logic_vector(N1 downto 0);
            InputSig2SyncOut : out std_logic_vector(N2 downto 0);
            ...
            InputSignSyncOut : out std_logic_vector(Nn downto 0);

            ...
            );
end NetName;

architecture Behavioral of NetName is

signal InputSig1Sync : std_logic_vector(N1 downto 0);
signal InputSig2Sync : std_logic_vector(N2 downto 0);
...
signal InputSignSync : std_logic_vector(Nn downto 0);

...

begin

process (CLK) begin
      if (CLK'event and CLK='1') then
            InputSig1Sync <= InputSig1;
            InputSig2Sync <= InputSig2;
            ...
```

```
            InputSignSync <= InputSign;
      end if;
end process;


InputSig1SyncOut <= InputSig1Sync;
InputSig2SyncOut <= InputSig2Sync;
...
InputSignSyncOut <= InputSignSync;

end Behavioral;
```

- **evo**: Adiciona, na *entity* do ficheiro VHDL gerado, sinais que permitem verificar o estado dos eventos de saída (estado actual). De seguida é possível verificar a negrito o código VHDL produzido quando esta opção é utilizada. A forma como é gerado o evento de saída pode ser consultada na secção 3.6

```
entity NetName is
    Port (

            -- General Signals
            CLK : in std_logic;
            RESET : in std_logic;

            EventOut1 : out std_logic;
            EventOut2 : out std_logic;
            ...
            EventOutn : out std_logic;
            ...

            );
end NetName;

architecture Behavioral of NetName is

signal EventOut1_Temp : std_logic;
signal EventOut2_Temp : std_logic;
...
signal EventOutn_Temp : std_logic;

begin

EventOut1_Temp <= '1' when (Condições) else '0';
EventOut2_Temp <= '1' when (Condições) else '0';
...
EventOutn_Temp <= '1' when (Condições) else '0';


EventOut1 <= EventOut1_Temp;
EventOut2 <= EventOut2_Temp;
...
EventOutn <= EventOutn_Temp;

end Behavioral;
```

- **evi**: Esta opção permite substituir os eventos de entrada que são, por omissão, criados internamente (estado actual) em função dos sinais de entrada respectivos por condições externas. Fica assim o utilizador responsável por indicar quando os eventos devem estar activos. De seguida é possível verificar a negrito o código VHDL produzido quando esta opção é utilizada

```
entity NetName is
    Port (
            -- General Signals
            CLK : in std_logic;
            RESET : in std_logic;

            EventIn1 : out std_logic;
            EventIn2 : out std_logic;
            ...
            EventInn : out std_logic;

            );
end NetName;

architecture Behavioral of NetName is


begin

--Utilização dos eventos

end Behavioral;
```

- **eis**: Adiciona, na *entity* do ficheiro VHDL gerado, sinais que permitem verificar o estado dos eventos de entrada e que vão influenciar o sistema no próximo ciclo ascendente do relógio. De seguida é possível verificar a negrito o código VHDL produzido quando esta opção é utilizada. A forma como são criados os eventos de entrada podem ser consultada na secção 3.2.

```
entity NetName is
    Port (
            -- General Signals
            CLK : in std_logic;
            RESET : in std_logic;

            EventIn1Out : out std_logic;
            EventIn2Out : out std_logic;
            ...
            EventInnOut : out std_logic;
            ...
            );
end NetName;

architecture Behavioral of NetName is

signal EventIn1 : std_logic;
```

```
signal EventIn2 : std_logic;
...
signal EventInn : std_logic;
begin

EventIn1 <= '1' when (Condições) else '0';
EventIn2 <= '1' when (Condições) else '0';
...
EventInn <= '1' when (Condições) else '0';


EventIn1Out <= EventIn1;
EventIn1Out <= EventIn1;
...
EventIn1Out <= EventIn1;

end Behavioral;
```

Para remoção de referências da *entity* do código gerado estão também disponíveis as seguintes opções:

- **sir**: Permite remover sinais de entrada da *entity* que não sejam utilizados no restante código VHDL. A sua utilidade baseia-se no facto de caso estes sinais não sejam utilizados em nenhuma condição do código, estes podem ser retirados e libertar recursos, bem como permitir uma optimização do código gerado. Considere-se os sinais de entrada *InputSig1*, *InputSig2*, ..., *InputSign*. De seguida é possível verificar a negrito o código VHDL produzido quando esta opção é utilizada, assumindo que se pretende retirar o sinal *InputSig2* do código gerado (*-sir InputSig2*).

```
entity NetName is
    Port (
            -- General Signals
            CLK : in std_logic;
            RESET : in std_logic;

            InputSig1 : in std_logic_vector(N1 downto 0);
            -- InputSig2
            InputSig3 : in std_logic_vector(N3 downto 0);
            ...
            InputSign : in std_logic_vector(Nn downto 0);


            );
end NetName;

architecture Behavioral of NetName is

signal InputSig1Sync : std_logic_vector(N1 downto 0);

signal InputSig3Sync : std_logic_vector(N3 downto 0);
...
signal InputSignSync : std_logic_vector(Nn downto 0);
```

```
begin

process (CLK) begin
      if (CLK'event and CLK='1') then
            InputSig1Sync <= InputSig1;

            InputSig3Sync <= InputSig3;
            ...
            InputSignSync <= InputSign;
      end if;
end process;

end Behavioral;
```

- **sor**: Permite remover sinais de saída que não sejam utilizados no restante código VHDL. A sua utilidade baseia-se no facto de caso estes sinais não sejam utilizados em nenhuma condição do código, estes podem ser retirados e libertar recursos, bem como permitir uma optimização do código gerado. Considere-se os sinais de entrada *OutputSig1*, *OutputSig2*, ..., *OutputSign*. De seguida é possível verificar a negrito o código VHDL produzido quando esta opção é utilizada, assumindo que se pretende retirar o sinal *OutputSig2* do código gerado. Na secção 3.7 pode ser obtido mais informação sobre a forma como os sinais de saída são gerados.

```
entity NetName is
    Port (
            -- General Signals
            CLK : in std_logic;
            RESET : in std_logic;

            ...

            );
end Name;

architecture Behavioral of NetName is

signal OutputSig1 : std_logic_vector(N1 downto 0);

signal OutputSig3 : std_logic_vector(N3 downto 0);
...
signal OutputSign : std_logic_vector(Nn downto 0);

begin

--Criação do código para a geração dos sinais de saída
--Todos os sinais são gerados com a excepção do sinal OutputSig2

end Behavioral;
```

Estão também disponíveis três opções de controlo de directorias, todas elas facultativas:

- **dir**: Define a directoria de origem do PNML e de destino do VHDL (se activa, as próximas duas não serão consideradas);
- **dirfr**: Directoria de origem do ficheiro PNML;
- **dirto**: Directoria de destino do ficheiro VHDL.

Anexo 2

Avisos e Erros

<u>Avisos que podem surgir na análise da informação lida do ficheiro:</u>

```
Warning 01.01: Missing information of an Input Signal (attribute MAX):
      ID: ID      Max: Max    Min: Min    Type: Type
```

Interpretação: Avisa que existe um sinal de entrada em que o valor máximo de bits não está definido (está vazio). Neste caso assume-se que *Max=Min*, desde que *Min* seja válido, caso contrário assume-se que é um sinal binário.

```
Warning 01.02: Missing information of an Input Signal (attribute MIN):
      ID: ID      Max: Max    Min: Min    Type: Type
      Assuming Min = 0.
```

Interpretação: Avisa que existe um sinal de entrada em que o valor mínimo de bits não está definido (está vazio). Neste caso assume-se como valor por omissão *Min=0*.

```
Warning 01.03: Missing information of an Input Signal (attribute TYPE):
      ID: ID      Max: Max    Min: Min    Type: Type
      Assuming Boolean type.
```

Interpretação: Avisa que existe um sinal de entrada em que o tipo de sinal não está definido (está vazio). Irá ser assumido que o sinal é do tipo binário.

```
Warning 01.04: Missing information of an Output Signal (attribute MAX):
      ID: ID      Max: Max    Min: Min    Type: Type  Value: Value
```

Interpretação: Avisa que existe um sinal de saída cujo valor máximo de bits não está definido (está vazio). Neste caso assume-se que *Max=Min*, desde que *Min* seja válido. Caso contrário assume-se que o sinal é binário.

```
Warning 01.05: Missing information of an Output Signal (attribute MIN):
      ID: ID      Max: Max    Min: Min    Type: Type  Value: Value
      Assumming Min = 0.
```

Interpretação: Avisa que existe um sinal de saída em que o valor mínimo de bits não está definido (está vazio). Neste caso assume-se como valor por defeito *Min=0*.

```
Warning 01.06: Missing information of an Output Signal (attribute TYPE):
      ID: ID      Max: Max    Min: Min    Type: Type  Value: Value
      Assumming Boolean.
```

Interpretação: Avisa que existe um sinal de saída em que o tipo de sinal não está definido (está vazio). Irá ser considerado que o sinal é binário.

```
Warning 01.07: Missing information of an Output Signal (attribute VALUE):
      ID: ID      Max: Max    Min: Min    Type: Type  Value: Value
      Assuming Value = 0.
```

Interpretação: Avisa que existe um sinal de saída em que o seu valor por omissão (em situação de *RESET*) não está definido (está vazio). Neste caso assume-se um valor por defeito, pelo que passa a ser *Value=0*.

```
Warning 01.08: Missing information of an Input Event associated to a signal
of type range (attribute LEVEL):
      ID: ID      Edge: Edge  Level: Level      Signal: Input Signal
      Assuming Level = 0.
```

Interpretação: Avisa que existe um evento de entrada associado a um sinal do tipo *Range* (com mais de um bit) em que o seu atributo *Level* não está definido (está vazio). Neste caso assume-se um valor por omissão, pelo que passa a ser *Level = 0*.

```
Warning 01.09: Missing information of an Input Event (attribute LEVEL)
associated with signal of type range (internal error):
      ID: ID      Edge: Edge  Level: Level      Signal: Input Signal
      Assuming Level = 0.
```

Interpretação: Avisa que existe um evento de entrada associado a um sinal do tipo *Range* (com mais de um bit) em que o seu atributo *Level* não está definido (está vazio). Neste caso assume-se um valor por omissão, pelo que passa a ser *Level = 0*.

```
Warning 01.10: Missing information of an Output Event associated with
signal of type range (attribute LEVEL):
      ID: ID      Edge: Edge  Level: Level      Signal: Output Signal
      Assuming Level = 0.
```

Interpretação: Avisa que existe um evento de saída associado a um sinal do tipo *Range* (com mais de um bit) em que o seu atributo *Level* não está definido (está vazio). Neste caso assume-se um valor por omissão, pelo que passa a ser *Level = 0*.

```
Warning 01.11: Missing information from an Arc (attribute INSCRIPTION):
      ID: ID      Inscription: Inscription
      Source: Source    Target: Target    Type: Type
      Assuming Inscription = 1.
```

Interpretação: Avisa que existe um arco em que o seu peso não está definido (está vazio). Neste caso assume-se o valor por omissão *Inscription = 1*.

```
Warning 01.12: Missing information from an Arc (attribute TYPE):
      ID: ID      Inscription: Inscription
      Source: Source    Target: Target    Type: Type
      Assuming Type = normal.
```

Interpretação: Avisa que existe um arco em que não se define que tipo de arco é (está vazio). Neste caso assume-se um tipo por omissão, pelo que passa a ser *Type = normal*.

```
Warning 01.13: Missing information of an Place (attribute BOUND):
      ID: ID      Bound: Bound      Comment: Comment
      InitialMarking: InitialMarking      Name: Name
      Assuming Bound = 1 (safe place).
```

Interpretação: Avisa que existe um lugar em que o seu número de marcas máximo não está definido. Neste caso assume-se o valor por omissão *Bound = 1* (lugar seguro).

```
Warning 01.14: Missing information of an Place (attribute INITIALMARKING):
      ID: ID      Bound: Bound      Comment: Comment
      InitialMarking: InitialMarking      Name: Name
      Assuming InitialMarking = 0.
```

Interpretação: Avisa que existe um lugar em que o seu número de marcas inicial não está definido. Neste caso assume-se o valor por omissão *InitialMarking = 0*.

```
Warning 01.15: Missing information of an Place (attribute NAME):
      ID: ID      Bound: Bound      Comment: Comment
      InitialMarking: InitialMarking      Name: Name
      Assuming Name = Id.
```

Interpretação: Avisa que existe um lugar em que o seu nome não está definido (está vazio). Neste caso assume-se um valor por omissão, pelo que passa a ser *Name = ID*.

```
Warning 01.16: Missing information of an SOAP (attribute TEXT):
      IdRef: IdRef      Language: Language      PlaceId: PlaceId
      Text: Text        Value: Value
      Assuming marking > 0.
```

Interpretação: Avisa que não existe (está vazio) uma condição (*Signal Output Action*) de activação de um sinal de saída associado a lugares ou lugares e eventos de saída. Neste caso assume-se por omissão que tem de haver marcas no lugar para que o sinal de saída esteja activo - *Text = "marking>0"*.

```
Warning 01.17: Missing information of an Transition (attribute NAME):
      ID: ID            Comment: Comment
      Name: Name        Priority: Priority
      Assuming Name = ID.
```

Interpretação: Avisa que existe uma transição cujo seu nome não está definido (está vazio). Neste caso assume-se um valor por omissão *Name = ID*.

```
Warning 01.18: Missing information of an Transition (attribute PRIORITY):
      ID: ID            Comment: Comment
      Name: Name        Priority: Priority
      Assuming Priority = 1.
```

Interpretação: Avisa que existe uma transição cuja prioridade não está definida (está vazio). Neste caso assume-se um valor por omissão, pelo que *Priority = 1*.

```
Warning 01.19: Do not exist any Arc (empty vector).
```

Interpretação: Indica que não foram encontrados arcos na rede.

```
Warning 01.20: Do not exist any Place (empty vector).
```

Interpretação: Indica que não foram encontrados lugares na rede.

```
Warning 01.21: Do not exist any Transition (empty vector).
```

Interpretação: Indica que não foram encontradas transições na rede.

```
Warning 01.22: Do not exist any Input Signals (empty vector).
```

Interpretação: Indica que não foram encontrados sinais de entrada na rede.

```
Warning 01.23: Do not exist any Output Signals (empty vector).
```

Interpretação: Indica que não foram encontrados sinais de saída na rede.

```
Warning 01.24: It was found a space at the name of an signal associated to
an Original Name. Assuming New Name.
```

Interpretação: Indica que foi encontrado um espaço no nome de alguma transição ou lugar, ou então num dos *Id's* de um sinal ou evento de entrada ou saída. Será apresentada informação detalhada sobre o tipo de sinal/evento e o nome/Id em causa.

```
Warning 01.25: Structural conflict found:
      Transition Name1 (with priority …) and
      Transition Name2 (with priority …)
```

Interpretação: Informa que encontrou duas transições em conflito estrutural, ou seja, ambas têm a mesma prioridade e o mesmo lugar de origem. Informação detalhada é apresentada sobre as transições e as suas prioridades associadas.

Avisos que podem surgir durante a fase de tradução da rede de Petri:

```
Warning 02.01: OET transition NOT found by ID:
      IDRef: IDRef      Transition: Transition
```

Interpretação: Avisa que não encontra a transição, cujo *ID* da transição é informado, a que se refere a condição referida por *IDRef*.

```
Warning 02.02: An unknown symbol was found on the first position of SIGT:
      Symbol: Symbol    Transition: TransitionID
```

Interpretação: Avisa que foi encontrado um simbolo desconhecido na primeira posição numa das condições de activação (*Signal Input Guard*) da transição referida através do *TransitionID*.

```
Warning 02.03: Unknown symbol found at SIGuard on transition
TransitionName/TransitionID: Ingnoring symbol.
```

Interpretação: Avisa que foi encontrado um simbolo desconhecido numa das condições de activação (*Signal Input Guard*) da transição referida através do *TransitionName* ou *TransitionID*. O simbolo será ignorado.

```
Warning 02.04: Missing information of SIGT (Transition). Condition added:
Condition
```

Interpretação: Avisa que nem toda a informação necessária para a definição da condição de activação da transição (*Signal Input Guard*) foi encontrada. Informa depois qual a condição que foi considerada.

```
Warning 02.05: Edge NOT recognized – OET/Output Event:
      IdRef: IdRef           Transition ID: TransitionID
      Event ID: EventID      Edge: Edge
      Level: Level           Signal: Signal
```

Interpretação: Avisa que o flanco do evento de saída não foi reconhecido.

```
Warning 02.06: Output Event NOT found in vector OET:
      IdRef: IdRef           Transition ID: TransitionID
```

Interpretação: Avisa que o evento de saída ao qual se refere um *IdRef* não foi encontrado como evento de saída.

```
Warning 02.07: Transition NOT found: TransitionName
```

Interpretação: A transição apresentada não foi encontrada.

```
Warning 02.08: (Registry) OET transition NOT found by ID:
      IDRef: IDRef            Transition: Transition
```

Interpretação: A informação necessária para declarar o registo de sincronização não está completa no OET.

```
Warning 02.09: Value > 1 at Output Signal Signal ID. Assuming Value=1.
```

Interpretação: Avisa que o valor presente na condição de activação de um sinal de saída (*Signal Output Action*) associado a lugares ou a lugares e eventos de saída não está correcto. Assume *1* e informa qual o sinal de saída associado.

```
Warning 02.10: Nothing found at SignalOutputAction ID.
      Assuming marking>0.
```

Interpretação: A condição de activação de um sinal de saída (*Signal Output Action ID*) associado a lugares ou a lugares e eventos de saída está vazia. Assume-se que a marcação do lugar tem de ser maior (condição de omissão).

```
Warning 02.11: Unknown Symbol Found at Signal Output Action on PLACE Place:
      Ignoring (position of symbol): Expression      Signal: Signal
```

Interpretação: Informa que encontrou um simbolo desconhecido na condição de activação de um sinal de saída (*Signal Output Action*) do lugar apresentado. É indicado a posição desse símbolo e a respectiva expressão.

```
Warning 02.12: Place not found: Place
```

Interpretação: Não foi encontrado o lugar representado na expressão da condição de activação de um sinal de saída (*Signal Output Action*).

```
Warning 02.13: Signal OutPut Action not right. Assuming Value > 1.
```

Interpretação: Avisa que o valor de uma das expressões da condição de activação de um sinal de saída (*Signal Output Action*) não está correcto. Assume *Value > 1*.

```
Warning 02.14: Symbol (Symbol) found at SOAP on Place Place.
```

Interpretação: O símbolo indicado foi encontrado no Signal Output Action Place relativo ao lugar indicado.

```
Warning 02.15: Edge unknown:
      EventID: EventID  EventEdge: EventEdge
```

Interpretação: Problema encontrado com um flanco.

```
Warning 02.16: Type of signal associated with event out was not found:
      Signal: Signal     Type: Type
```

Interpretação: O tipo de sinal não foi encontrado quando se procurava pelo evento de saída.

```
Warning 02.17: No condition found. Assuming marking>0.
```

Interpretação: Não foi encontrada uma condição de avaliação nas condições que definem a activação de um sinal de saída associados a lugares ou a lugares e eventos de saída (*Signal Output Action*), pelo que assume-se *marking>0*.

```
Warning 02.18: Value to attribute to signal is bigger than maximum value
declared for the signal (signal).
```

Interpretação: Informa que durante a conversão do valor passado num dos *SOAP* é maior do que o valor máximo atribuido pelo projectista para o sinal.

```
Warning 02.19: The first character in the Name of the Net is not a letter:
NetName
```

Interpretação: Informa que durante a conversão foi observado que o nome atribuído à rede contém um caractér não válido na primeira posição (esta deverá ser obrigatoriamente uma letra).

```
Warning 02.20: Value to attribute to output signal (output signal)
associated to the place (place) is equal to the value of omission of the
output signal (value).
```

Interpretação: Indica que o valor que irá ser atribuído ao sinal de saída quando a condição associada ao lugar informado for verdadeira é igual ao valor de omissão do sinal de saída (não haverá assim alteração no valor do sinal).

```
Warning 02.21: Value pre-defined of signal isn't correct:
               Signal: Signal      Value: Value
```

Interpretação: O valor pré-definido para o sinal não está correcto.

Erros que podem surgir na análise da informação lida do ficheiro:

```
Error 01.01: Missing information of an Input Signal (attribute ID):
      ID: ID      Max: Max    Min: Min    Type: Type
```

Interpretação: Foi encontrado um sinal de entrada com um *ID* vazio.

```
Error 01.02: Bad definition of an Input Signal (MIN>MAX):
      ID: ID      Max: Max    Min: Min    Type: Type
```

Interpretação: Foi encontrado um sinal de entrada com um valor mínimo maior que o seu valor máximo.

```
Error 01.03: Missing information of an Output Signal (attribute ID):
      ID: ID      Max: Max    Min: Min    Type: Type  Value: Value
```

Interpretação: Foi encontrado um sinal de saída com um *ID* vazio.

```
Error 01.04: Bad definition of an Output Signal (MIN>MAX):
      ID: ID      Max: Max    Min: Min    Type: Type
```

Interpretação: Foi encontrado um sinal de saída com um valor mínimo maior que o seu valor máximo.

```
Error 01.05: Missing information of an Input Event (attribute SIGNAL):
      ID: ID      Edge: Edge        Level: Level      Signal: Signal
```

Interpretação: Foi encontrado um evento de entrada sem sinal associado (está vazio).

```
Error 01.06: Missing information of an Input Event (attribute ID):
      ID: ID      Edge: Edge        Level: Level      Signal: Signal
```

Foi encontrado um evento de entrada sem *ID* (está vazio).

```
Error 01.07: Missing information of an Input Event (attribute EDGE):
      ID: ID      Edge: Edge        Level: Level      Signal: Signal
```

Interpretação: Foi encontrado um evento de entrada sem ter definido correctamente o flanco (está vazio).

```
Error 01.08: Input event without an input Signal associated:
      ID: ID      Edge: Edge        Level: Level      Signal: Signal
```

Interpretação: O sinal associado ao evento de entrada não foi encontrado.

```
Error 01.09: Missing information of an Input Event (attribute SIGNAL)
(Original vector):
      ID: ID      Edge: Edge        Level: Level      Signal: Signal
```

Interpretação: Foi encontrado um evento de entrada sem sinal associado (está vazio) (vector *Original*).

```
Error 01.10: Missing information of an Input Event (attribute ID) (Original
vector):
      ID: ID      Edge: Edge        Level: Level      Signal: Signal
```

Interpretação: Foi encontrado um evento de entrada sem *ID* (está vazio) (vector *Original*).

```
Error 01.11: Missing information of an Input Event (attribute EDGE)
(Original vector):
      ID: ID      Edge: Edge        Level: Level      Signal: Signal
```

Interpretação: Foi encontrado um evento de entrada sem ter definido correctamente o flanco (está vazio) (vector *Original*).

```
Error 01.12: Event has no Signal associated:
      ID: ID      Edge: Edge        Level: Level      Signal: Signal
```

Interpretação: Não foi encontrado o sinal associado ao evento de entrada apresentado.

```
Error 01.13: The name of an Output Signal associated to an Output Event is
empty:
      ID: ID      Edge: Edge        Level: Level      Signal: Signal
```

Interpretação: O atributo que define o nome do sinal de saída associado ao evento de saída está vazio.

```
Error 01.14: Missing information of an Output Event (attribute ID):
      ID: ID      Edge: Edge        Level: Level      Signal: Signal
```

Interpretação: Foi encontrado um evento de saída sem *ID* (está vazio).

```
Error 01.15: Missing information of an Output Event (attribute EDGE):
      ID: ID      Edge: Edge        Level: Level      Signal: Signal
```

Interpretação: Foi encontrado um evento de saída sem ter definido correctamente o flanco (está vazio).

```
Error 01.16: Output event without a Output Signal associated:
      ID: ID      Edge: Edge        Level: Level      Signal: Signal
```

Interpretação: O sinal associado ao evento de saída não foi encontrado.

```
Error 01.17: Missing information of an Arc (attribute ID):
      ID: ID            Inscription: Inscription
      Source: Source    Target: Target          Type: Type
```

Interpretação: Foi encontrado um arco sem indicação do seu *ID* (está vazio).

```
Error 01.18: Missing information of an Arc (attribute SOURCE):
      ID: ID            Inscription: Inscription
      Source: Source    Target: Target          Type: Type
```

Interpretação: Foi encontrado um arco sem indicação da sua origem (está vazio).

```
Error 01.19: Missing information of an Arc (attribute TARGET):
      ID: ID            Inscription: Inscription
      Source: Source    Target: Target          Type: Type
```

Interpretação: Foi encontrado um arco sem indicação do seu destino (está vazio).

```
Error 01.20: Missing information of an Place (attribute ID):
      ID: ID            Bound: Bound            Comment: Comment
      InitialMarking: InitialMarking            Name: Name
```

Interpretação: Foi encontrado um lugar sem indicação do seu *ID* (está vazio).

```
Error 01.21: Missing information of an Place (NAME and ID):
      ID: ID            Bound: Bound            Comment: Comment
      InitialMarking: InitialMarking            Name: Name
```

Interpretação: Foi encontrado um lugar sem indicação do seu nome e do seu *ID* (estão vazios). Não é possível substituir.

```
Error 01.22: Missing information of an SOAP (attribute IDREF):
      IdRef: IdRef            Language: Language
      PlaceId: PlaceId        Text: Text        Value: Value
```

Interpretação: Foi encontrado uma condição de activação de um sinal de saída associado a lugares ou a lugares e eventos de saída (*Signal Output Action*) sem indicação do seu *IdRef* (está vazio).

```
Error 01.23: Missing information of an SOAP (attribute LANGUAGE):
      IdRef: IdRef            Language: Language
      PlaceId: PlaceId        Text: Text        Value: Value
```

Interpretação: Foi encontrado uma condição de activação de um sinal de saída associado a lugares ou a lugares e eventos de saída (*Signal Output Action*) sem indicação da linguagem a que se refere (está vazio).

```
Error 01.24: Missing information of an SOAP (attribute PLACEID):
      IdRef: IdRef            Language: Language
      PlaceId: PlaceId        Text: Text        Value: Value
```

Interpretação: Foi encontrado uma condição de activação de um sinal de saída associado a lugares ou a lugares e eventos de saída (*Signal Output Action*) sem indicação do *ID* do lugar a que se refere (está vazio).

```
Error 01.25: Missing information of an SOAP (attribute VALUE):
      IdRef: IdRef            Language: Language
      PlaceId: PlaceId        Text: Text        Value: Value
```

Interpretação: Foi encontrado uma condição de activação de um sinal de saída associado a lugares ou a lugares e eventos de saída (*Signal Output Action*) sem indicação do valor a atribuir quando activo (está vazio).

```
Error 01.26: Missing information of an Transition (attribute ID):
      ID: ID            Comment: Comment
      Name: Name        Priority: Priority
```

Interpretação: Foi encontrada uma transição sem indicação do seu *ID* (está vazio).

```
Error 01.27: Missing information of an Transition (NAME and ID):
      ID: ID            Comment: Comment
      Name: Name        Priority: Priority
```

Interpretação: Foi encontrada uma transição sem indicação do seu nome e do seu *ID* (ambos estão vazios).

```
Error 01.28: Missing information of an SIGT (attribute IDTRANSITION):
      IdRef: IdRef      Language: Language      Text: Text
```

Interpretação: Foi encontrada uma condição de habilitação de uma transição (*Signal Input Guard*) sem indicação do *ID* da transição (está vazio).

```
Error 01.29: Missing information of an SIGT (attribute LANGUAGE):
      IdRef: IdRef      Language: Language      Text: Text
```

Interpretação: Foi encontrada uma condição de habilitação de uma transição (*Signal Input Guard*) sem indicação da linguagem a que se destina (está vazio).

```
Error 01.30: Missing information of an SIGT (attribute TEXT):
      IdRef: IdRef      Language: Language      Text: Text
```

Interpretação: Foi encontrada uma condição de habilitação de uma transição (*Signal Input Guard*) sem indicação da sua condição de habilitação (está vazio).

```
Error 01.31: Missing information of an IET (attribute IDREF):
      IdRef: IdRef            IdTransition: IdTransition
```

Interpretação: Foi encontrada uma associação de um evento de entrada a uma transição (*IET-Input Event Transition*) sem indicação do *IdRef* do evento de entrada (está vazio).

```
Error 01.32: Missing information of an IET (attribute IDTRANSITION):
      IdRef: IdRef            IdTransition: IdTransition
```

Interpretação: Foi encontrada uma associação de um evento de entrada a uma transição (*IET-Input Event Transition*) sem indicação do *IdTransition* da transição (está vazio).

```
Error 01.33: Missing information of an OET (attribute IDREF):
      IdRef: IdRef            IdTransition: IdTransition
```

Interpretação: Foi encontrada uma associação de um evento de saída a uma transição (*OET-Output Event Transition*) sem indicação do *IdRef* do evento de saída (está vazio).

```
Error 01.34: Missing information of an OET (attribute IDTRANSITION):
      IdRef: IdRef            IdTransition: IdTransition
```

Interpretação: Foi encontrada uma associação de um evento de saída a uma transição (*OET-Output Event Transition*) sem indicação do *IdTransition* da transição (está vazio).

Erros que podem surgir durante a fase de tradução da rede de Petri:

```
Error 02.01: Can't open file to write VHDL.
```

Interpretação: Não foi possivel abrir para escrita o ficheiro destino VHD.

```
Error 02.02: Input Signal Type definition: Input Signal.
```

Interpretação: Erro na definição do tipo de sinal de entrada indicado.

```
Error 02.03: Output Signal Type definition: Output Signal.
```

Interpretação: Erro na definição do tipo de sinal de saída indicado.

```
Error 02.04: Bad definition of signal:
      Signal: Signal    Type: Type  Max: Max    Min: Min
```

Interpretação: Erro na definição dos sinais.

```
Error 02.05: Type of Event Edge not recognize at Event Event.
```

Interpretação: O flanco do evento representado pelo seu *ID* não foi reconhecido.

```
Error 02.06: Temporary Output Signal Type Definition: Output Signal.
```

Interpretação: Erro na definição do sinal de saída representado pelo seu *ID*.

```
Error 02.07: Source ID from arc not found: arch ID
```

Interpretação: Indica que não foi encontrado o *ID* de origem do arco cujo *ID* é mostrado.

```
Error 02.08: Target ID from arc not found: arch ID
```

Interpretação: Indica que não foi encontrado o *ID* de destino do arco cujo *ID* é mostrado.

```
Error 02.09: Source ID from arc not found: arch ID
```

Interpretação: Indica que não foi encontrado o *ID* de origem do arco cujo *ID* é mostrado (situação diferente da anterior).

```
Error 02.10: Target ID from arc not found: arch ID
```

Interpretação: Indica que não foi encontrado o *ID* de destino do arco cujo *ID* é mostrado (situação diferente da anterior).

```
Error 02.11: Copy of arcs vector didn't complete correctly (diferent
sizes).
```

Interpretação: Indica que houve um erro no processo de replicação do vector de arcos, o tamanho final dos dois não é idêntico havendo assim um erro na cópia.

```
Error 02.12: SIGT Signal NOT Found at Signal Input's Vector.
```

Interpretação: Indica que o sinal presente na condição do *Signal Input Guard* não foi encontrado entre os sinais de entrada. O nome a procurar é indicado.

```
Error 02.13: Type of Event Edge not recognize at Event Event.
```

Interpretação: O flanco do evento representado pelo seu *ID* não foi reconhecido.

```
Error 02.14: Events not found or bad declaration of an Input Event.
```

Interpretação: Não foi encontrado ou contém erro a declaração de algum evento de entrada.

```
Error 03.01: Can't Write Log file Result.txt.
```

Interpretação: Informa que não foi possivel abrir para escrita o ficheiro de resultados *Result.txt* (pode estar com protecção de escrita/permissões apenas de leitura).

Avisos que podem surgir na análise das opções indicadas pelo utilizador:

```
Warning 03.01: Please verify user options: ( after ).
```

Durante a análise das opções que o utilizador inseriu na linha de comandos, existe primeiro um fechar de parêntesis e depois é que este é aberto – "(" aparece depois de ")".

```
Warning 03.02: Option not understood – ( not found, ) found.
```

Interpretação: Durante a análise das opções que o utilizador inseriu na linha de comandos, existe um fechar de parêntesis mas não foi encontrado um abrir de parêntesis.

```
Warning 03.03: Symbol of a new option found but not expected.
```

Interpretação: Foi encontrado o simbolo "-" indicando que é uma nova opção mas não era expectável pois a opção anterior não estava terminada.

```
Warning 03.04: Options error.
```

Interpretação: Encontrado um erro nas opções definidas pelo utilizador.

```
Warning 03.05: Path on –dir/–dirfr/–dirto option not defined.
```

Interpretação: Uma das opções –dir/-dirfr/dirto foi activada, mas não existe um *path* definido. Para tal deverá ser introduzido o *path* da seguinte forma: -dir(*path*).

Anexo 3

Parque de estacionamento (1 in 1 out)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity park1in1out is
      Port ( -- General Signals
            CLK : in std_logic;
            RESET : in std_logic;

            GotTicket : in std_logic_vector(0 downto 0);
            arrive : in std_logic_vector(0 downto 0);
            pay : in std_logic_vector(0 downto 0);
            leave : in std_logic_vector(0 downto 0);

            GateInOpen : out  std_logic_vector(0 downto 0);
            GateOutOpen : out  std_logic_vector(0 downto 0)
            );
end park1in1out;

architecture Behavioral of park1in1out is

signal GotTicketSync : std_logic_vector(0 downto 0);
signal arriveSync : std_logic_vector(0 downto 0);
signal paySync : std_logic_vector(0 downto 0);
signal leaveSync : std_logic_vector(0 downto 0);
signal arriveSync_temp : std_logic_vector(0 downto 0);
signal ArriveIn : std_logic;
signal GotTicketSync_temp : std_logic_vector(0 downto 0);
signal Identified : std_logic;
signal ArriveOut : std_logic;
signal GateInOpen_Temp : std_logic_vector(0 downto 0);
signal GateOutOpen_Temp : std_logic_vector(0 downto 0);

constant WP0_189T0_287, WT0_287P1_175, WP1_175T1_303,
      WT1_303P2_203, WP2_203T2_351, WT2_351P0_189,
      WT3_391P5_259, WP5_259T4_431, WT4_431P6_273,
      WP6_273T5_463, WT5_463P7_245, WP7_245T3_391,
      WP4_231T2_351, WT2_351P3_217, WP3_217T3_391,
      WT3_391P4_231 : integer := 1;

signal T0_287, T1_303, T2_351, T3_391, T4_431, T5_463 : std_logic;

signal P1_175, P1_175Temp : std_logic_vector(0 downto 0);
signal P0_189, P0_189Temp : std_logic_vector(0 downto 0);
signal P2_203, P2_203Temp : std_logic_vector(0 downto 0);
signal P3_217, P3_217Temp : std_logic_vector(1 downto 0);
signal P4_231, P4_231Temp : std_logic_vector(1 downto 0);
signal P7_245, P7_245Temp : std_logic_vector(0 downto 0);
signal P5_259, P5_259Temp : std_logic_vector(0 downto 0);
```

```
signal P6_273, P6_273Temp : std_logic_vector(0 downto 0);

begin

T0_287 <= '1' when
            ((ArriveOut='1') and P0_189>=WP0_189T0_287)
            else '0';
T1_303 <= '1' when
            ((ArriveIn='1') and P1_175>=WP1_175T1_303)
            else '0';
T2_351 <= '1' when
            ((Identified='1') and P2_203>=WP2_203T2_351
            and P4_231>=WP4_231T2_351)
            else '0';
T3_391 <= '1' when
            (paySync="1" and P7_245>=WP7_245T3_391
            and P3_217>=WP3_217T3_391)
            else '0';
T4_431 <= '1' when (leaveSync="0" and P5_259>=WP5_259T4_431) else '0';
T5_463 <= '1' when (leaveSync="1" and P6_273>=WP6_273T5_463) else '0';

process (CLK) begin
      if (CLK'event and CLK='1') then
            GotTicketSync <= GotTicket;
            arriveSync <= arrive;
            paySync <= pay;
            leaveSync <= leave;
            arriveSync_temp <= arriveSync;
            GotTicketSync_temp <= GotTicketSync;
      end if;
end process;

GateInOpen_Temp <= "1" when (P0_189>0) else "0";
GateOutOpen_Temp <= "1" when (P5_259>0) else "0";

ArriveIn <= '1' when
            (arriveSync="1" and arriveSync_temp="0") else '0';
Identified <= '1' when
            (GotTicketSync="1" and GotTicketSync_temp="0") else '0';
ArriveOut <= '1' when
            (arriveSync="0" and arriveSync_temp="1") else '0';

--EntranceFree
P1_175Temp <= conv_std_logic_vector((conv_integer(P1_175)
            + WT0_287P1_175*conv_integer(T0_287)
            - WP1_175T1_303*conv_integer(T1_303)),1);

process (CLK, RESET) begin
      if (RESET = '1') then P1_175 <= "1";
      elsif (CLK'event and CLK='1') then
            P1_175 <= P1_175Temp;
      end if;
end process;

--GateInOpen
P0_189Temp <= conv_std_logic_vector((conv_integer(P0_189)
            + WT2_351P0_189*conv_integer(T2_351)
            - WP0_189T0_287*conv_integer(T0_287)),1);

process (CLK, RESET) begin
      if (RESET = '1') then P0_189 <= "0";
      elsif (CLK'event and CLK='1') then
            P0_189 <= P0_189Temp;
```

```
      end if;
end process;

--WaitingTicket
P2_203Temp <= conv_std_logic_vector((conv_integer(P2_203)
            + WT1_303P2_203*conv_integer(T1_303)
            - WP2_203T2_351*conv_integer(T2_351)),1);

process (CLK, RESET) begin
      if (RESET = '1') then P2_203 <= "0";
      elsif (CLK'event and CLK='1') then
            P2_203 <= P2_203Temp;
      end if;
end process;

--CarInsideZone
P3_217Temp <= conv_std_logic_vector((conv_integer(P3_217)
            + WT2_351P3_217*conv_integer(T2_351)
            - WP3_217T3_391*conv_integer(T3_391)),2);

process (CLK, RESET) begin
      if (RESET = '1') then P3_217 <= "00";
      elsif (CLK'event and CLK='1') then
            P3_217 <= P3_217Temp;
      end if;
end process;

--FreePlaces
P4_231Temp <= conv_std_logic_vector((conv_integer(P4_231)
            + WT3_391P4_231*conv_integer(T3_391)
            - WP4_231T2_351*conv_integer(T2_351)),2);

process (CLK, RESET) begin
      if (RESET = '1') then P4_231 <= "11";
      elsif (CLK'event and CLK='1') then
            P4_231 <= P4_231Temp;
      end if;
end process;

--WaitingToPay
P7_245Temp <= conv_std_logic_vector((conv_integer(P7_245)
            + WT5_463P7_245*conv_integer(T5_463)
            - WP7_245T3_391*conv_integer(T3_391)),1);

process (CLK, RESET) begin
      if (RESET = '1') then P7_245 <= "0";
      elsif (CLK'event and CLK='1') then
            P7_245 <= P7_245Temp;
      end if;
end process;

--GateOutOpen
P5_259Temp <= conv_std_logic_vector((conv_integer(P5_259)
            + WT3_391P5_259*conv_integer(T3_391)
            - WP5_259T4_431*conv_integer(T4_431)),1);


process (CLK, RESET) begin
      if (RESET = '1') then P5_259 <= "0";
      elsif (CLK'event and CLK='1') then
            P5_259 <= P5_259Temp;
      end if;
end process;
```

```
--ExitFree
P6_273Temp <= conv_std_logic_vector((conv_integer(P6_273)
            + WT4_431P6_273*conv_integer(T4_431)
            - WP6_273T5_463*conv_integer(T5_463)),1);

process (CLK, RESET) begin
      if (RESET = '1') then P6_273 <= "1";
      elsif (CLK'event and CLK='1') then
            P6_273 <= P6_273Temp;
      end if;
end process;

GateInOpen <= GateInOpen_Temp;
GateOutOpen <= GateOutOpen_Temp;

end Behavioral;
```

Anexo 4

Parque de estacionamento (2 in 1 out)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity park2in1out is
      Port ( -- General Signals
            CLK : in std_logic;
            RESET : in std_logic;

            GotTicket : in std_logic_vector(0 downto 0);
            arrive : in std_logic_vector(0 downto 0);
            pay : in std_logic_vector(0 downto 0);
            leave : in std_logic_vector(0 downto 0);
            GotTicket2 : in std_logic_vector(0 downto 0);
            arrive2 : in std_logic_vector(0 downto 0);

            GateInOpen : out  std_logic_vector(0 downto 0);
            GateOutOpen : out  std_logic_vector(0 downto 0);
            GateIn2Open : out  std_logic_vector(0 downto 0)
            );
end park2in1out;

architecture Behavioral of park2in1out is

signal GotTicketSync : std_logic_vector(0 downto 0);
signal arriveSync : std_logic_vector(0 downto 0);
signal paySync : std_logic_vector(0 downto 0);
signal leaveSync : std_logic_vector(0 downto 0);
signal GotTicket2Sync : std_logic_vector(0 downto 0);
signal arrive2Sync : std_logic_vector(0 downto 0);
signal arriveSync_temp : std_logic_vector(0 downto 0);
signal ArriveIn : std_logic;
signal GotTicketSync_temp : std_logic_vector(0 downto 0);
signal Identified : std_logic;
signal ArriveOut : std_logic;
signal GotTicket2Sync_temp : std_logic_vector(0 downto 0);
signal identified2 : std_logic;
signal arrive2Sync_temp : std_logic_vector(0 downto 0);
signal ArriveIn2 : std_logic;
signal ArriveOut2 : std_logic;
signal GateInOpen_Temp : std_logic_vector(0 downto 0);
signal GateOutOpen_Temp : std_logic_vector(0 downto 0);
signal GateIn2Open_Temp : std_logic_vector(0 downto 0);

constant WP0_189T0_287, WT0_287P1_175, WP1_175T1_303,
      WT1_303P2_203, WT3_391P5_259, WP5_259T4_431,
      WT4_431P6_273, WP6_273T5_463, WT5_463P7_245,
      WP7_245T3_391, WP3_217T3_391, WT3_391P4_231,
      WP8_4751T6_4781, WT6_4781P9_4737, WP9_4737T7_4765,
      WT7_4765P10_4723, WP2_203T2_351, WT2_351P0_189,
```

```
      WT8_5424P8_4751, WP10_4723T8_5424, WT2_351P3_217,
      WP4_231T8_5424, WP4_231T2_351, WT8_5424P3_217 : integer := 1;

signal T0_287, T1_303, T2_351, T3_391, T4_431, T5_463,
      T7_4765, T6_4781, T8_5424 : std_logic;

signal P1_175, P1_175Temp : std_logic_vector(0 downto 0);
signal P0_189, P0_189Temp : std_logic_vector(0 downto 0);
signal P2_203, P2_203Temp : std_logic_vector(0 downto 0);
signal P3_217, P3_217Temp : std_logic_vector(1 downto 0);
signal P4_231, P4_231Temp : std_logic_vector(1 downto 0);
signal P7_245, P7_245Temp : std_logic_vector(0 downto 0);
signal P5_259, P5_259Temp : std_logic_vector(0 downto 0);
signal P6_273, P6_273Temp : std_logic_vector(0 downto 0);
signal P10_4723, P10_4723Temp : std_logic_vector(0 downto 0);
signal P9_4737, P9_4737Temp : std_logic_vector(0 downto 0);
signal P8_4751, P8_4751Temp : std_logic_vector(0 downto 0);

begin

T0_287 <= '1' when
      ((ArriveOut='1') and P0_189>=WP0_189T0_287)
      else '0';
T1_303 <= '1' when
      ((ArriveIn='1') and P1_175>=WP1_175T1_303)
      else '0';
T2_351 <= '1' when
      ((Identified='1') and P2_203>=WP2_203T2_351
      and P4_231>=WP4_231T2_351)
      else '0';
T3_391 <= '1' when (paySync="1" and P7_245>=WP7_245T3_391
      and P3_217>=WP3_217T3_391)
      else '0';
T4_431 <= '1' when
      (leaveSync="0" and P5_259>=WP5_259T4_431)
      else '0';
T5_463 <= '1' when
      (leaveSync="1" and P6_273>=WP6_273T5_463)
      else '0';
T7_4765 <= '1' when
      ((ArriveIn2='1') and P9_4737>=WP9_4737T7_4765)
      else '0';
T6_4781 <= '1' when
      ((ArriveOut2='1') and P8_4751>=WP8_4751T6_4781)
      else '0';
T8_5424 <= '1' when
      ((identified2='1') and P10_4723>=(WP10_4723T8_5424) and
      P4_231>=(WP4_231T8_5424 + conv_integer(T2_351)*WP4_231T2_351))
      else '0';

process (CLK) begin
      if (CLK'event and CLK='1') then
            GotTicketSync <= GotTicket;
            arriveSync <= arrive;
            paySync <= pay;
            leaveSync <= leave;
            GotTicket2Sync <= GotTicket2;
            arrive2Sync <= arrive2;
            arriveSync_temp <= arriveSync;
            GotTicketSync_temp <= GotTicketSync;
            GotTicket2Sync_temp <= GotTicket2Sync;
            arrive2Sync_temp <= arrive2Sync;
      end if;
```

```
end process;

GateInOpen_Temp <= "1" when (P0_189>0) else "0";
GateOutOpen_Temp <= "1" when (P5_259>0) else "0";
GateIn2Open_Temp <= "1" when (P8_4751>0) else "0";

ArriveIn <= '1' when
      (arriveSync="1" and arriveSync_temp="0")
      else '0';
Identified <= '1' when
      (GotTicketSync="1" and GotTicketSync_temp="0")
      else '0';
ArriveOut <= '1' when
      (arriveSync="0" and arriveSync_temp="1")
      else '0';
identified2 <= '1' when
      (GotTicket2Sync="1" and GotTicket2Sync_temp="0")
      else '0';
ArriveIn2 <= '1' when
      (arrive2Sync="1" and arrive2Sync_temp="0")
      else '0';
ArriveOut2 <= '1' when
      (arrive2Sync="0" and arrive2Sync_temp="1")
      else '0';

--EntranceFree
P1_175Temp <= conv_std_logic_vector((conv_integer(P1_175)
      + WT0_287P1_175*conv_integer(T0_287)
      - WP1_175T1_303*conv_integer(T1_303)),1);

process (CLK, RESET) begin
      if (RESET = '1') then P1_175 <= "1";
      elsif (CLK'event and CLK='1') then
            P1_175 <= P1_175Temp;
      end if;
end process;

--GateInOpen
P0_189Temp <= conv_std_logic_vector((conv_integer(P0_189)
      + WT2_351P0_189*conv_integer(T2_351)
      - WP0_189T0_287*conv_integer(T0_287)),1);

process (CLK, RESET) begin
      if (RESET = '1') then P0_189 <= "0";
      elsif (CLK'event and CLK='1') then
            P0_189 <= P0_189Temp;
      end if;
end process;

--WaitingTicket
P2_203Temp <= conv_std_logic_vector((conv_integer(P2_203)
      + WT1_303P2_203*conv_integer(T1_303)
      - WP2_203T2_351*conv_integer(T2_351)),1);

process (CLK, RESET) begin
      if (RESET = '1') then P2_203 <= "0";
      elsif (CLK'event and CLK='1') then
            P2_203 <= P2_203Temp;
      end if;
end process;

--CarInsideZone
P3_217Temp <= conv_std_logic_vector((conv_integer(P3_217)
```

```
      + WT2_351P3_217*conv_integer(T2_351)
      + WT8_5424P3_217*conv_integer(T8_5424)
      - WP3_217T3_391*conv_integer(T3_391)),2);

process (CLK, RESET) begin
      if (RESET = '1') then P3_217 <= "00";
      elsif (CLK'event and CLK='1') then
            P3_217 <= P3_217Temp;
      end if;
end process;

--FreePlaces
P4_231Temp <= conv_std_logic_vector((conv_integer(P4_231)
      + WT3_391P4_231*conv_integer(T3_391)
      - WP4_231T8_5424*conv_integer(T8_5424)
      - WP4_231T2_351*conv_integer(T2_351)),2);

process (CLK, RESET) begin
      if (RESET = '1') then P4_231 <= "11";
      elsif (CLK'event and CLK='1') then
            P4_231 <= P4_231Temp;
      end if;
end process;

--WaitingToPay
P7_245Temp <= conv_std_logic_vector((conv_integer(P7_245)
      + WT5_463P7_245*conv_integer(T5_463)
      - WP7_245T3_391*conv_integer(T3_391)),1);

process (CLK, RESET) begin
      if (RESET = '1') then P7_245 <= "0";
      elsif (CLK'event and CLK='1') then
            P7_245 <= P7_245Temp;
      end if;
end process;

--GateOutOpen
P5_259Temp <= conv_std_logic_vector((conv_integer(P5_259)
      + WT3_391P5_259*conv_integer(T3_391)
      - WP5_259T4_431*conv_integer(T4_431)),1);

process (CLK, RESET) begin
      if (RESET = '1') then P5_259 <= "0";
      elsif (CLK'event and CLK='1') then
            P5_259 <= P5_259Temp;
      end if;
end process;

--ExitFree
P6_273Temp <= conv_std_logic_vector((conv_integer(P6_273)
      + WT4_431P6_273*conv_integer(T4_431)
      - WP6_273T5_463*conv_integer(T5_463)),1);

process (CLK, RESET) begin
      if (RESET = '1') then P6_273 <= "1";
      elsif (CLK'event and CLK='1') then
            P6_273 <= P6_273Temp;
      end if;
end process;

--Waiting2Ticket
P10_4723Temp <= conv_std_logic_vector((conv_integer(P10_4723)
      + WT7_4765P10_4723*conv_integer(T7_4765)
```

```
      - WP10_4723T8_5424*conv_integer(T8_5424)),1);

process (CLK, RESET) begin
      if (RESET = '1') then P10_4723 <= "0";
      elsif (CLK'event and CLK='1') then
            P10_4723 <= P10_4723Temp;
      end if;
end process;

--Entrance2Free
P9_4737Temp <= conv_std_logic_vector((conv_integer(P9_4737)
      + WT6_4781P9_4737*conv_integer(T6_4781)
      - WP9_4737T7_4765*conv_integer(T7_4765)),1);

process (CLK, RESET) begin
      if (RESET = '1') then P9_4737 <= "1";
      elsif (CLK'event and CLK='1') then
            P9_4737 <= P9_4737Temp;
      end if;
end process;

--GateIn2Open
P8_4751Temp <= conv_std_logic_vector((conv_integer(P8_4751)
      + WT8_5424P8_4751*conv_integer(T8_5424)
      - WP8_4751T6_4781*conv_integer(T6_4781)),1);

process (CLK, RESET) begin
      if (RESET = '1') then P8_4751 <= "0";
      elsif (CLK'event and CLK='1') then
            P8_4751 <= P8_4751Temp;
      end if;
end process;

GateInOpen <= GateInOpen_Temp;
GateOutOpen <= GateOutOpen_Temp;
GateIn2Open <= GateIn2Open_Temp;

end Behavioral;
```