Universidade Federal de Pernambuco

Centro de Informática

Pós-Graduação em Ciência da Computação

InterfPISH – Uma ferramenta para geração automática de interfaces em *hardware/software Co-design*

por

Cristiano Coêlho de Araújo

cca2@cin.ufpe.br

Dissertação submetida para o Centro de Informática da Universidade Federal de Pernambuco, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Dissertação orientada por

Professora Doutora Edna Natividade da Silva Barros

Professora Adjunta do Centro de Informática da

Universidade Federal de Pernambuco.

Recife, fevereiro de 2001

Agradecimentos

Durante o tempo em que durou este mestrado, e olha que foi bastante tempo algumas pessoas foram muito importantes direta ou indiretamente para o cumprimento de mais esta etapa em minha vida.

Gostaria inicialmente de agradecer a oportunidade de entrar no departamento de informática àminha orientadora Professora Edna Barros, embora não imaginasse ainda o quanto sofreria depois. Brincadeiras a parte, o ambiente encontrado foi muito positivo e pesou para que optasse por fazer o mestrado.

Também agradeço aos professores Manoel Eusébio e Sérgio Cavalcante pela confiança e ensinamentos de que fui depositário.

Não posso esquecer o amigo Marcus Vinícius pelas noites viradas enquanto o ajudava na implementação da sua tese e das farras, é claro.

Há uma lista enorme de pessoas que gostaria de agradecer, Fred, Abel, Raquel que entraram e se não sofreram ainda passarão por todo este processo de prévias de defesa de tese.

À Veronica pelo apoio e carinho dado em todos os momentos fossem eles os melhores ou os mais difíceis. Ela que é uma das minhas mais importantes conquistas dentre as muitas que obtive durante este mestrado.

Finalmente quero agradecer a toda a minha família que suportou um maluco que virava noites no departamento e acordava às 10:00hs da manhã. Mal sabem eles que é apenas o começo de uma nova fase na minha vida.

A todos que contribuíram direta ou indiretamente meu muito obrigado.

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

Hardware/Software Co-design é uma metodologia utilizada para o desenvolvimento de sistemas digitais onde parte da funcionalidade do sistema será implementada em *software* e executada em processadores e outra parte implementada diretamente em componentes *hardware* específicos (ASIC´s, FPGA[1]´s). Esta metodologia permite que uma mesma especificação de um sistema digital possa ser implementada com diversas configurações de processos em hardware e software distintas e em diversas arquiteturas alvo diferentes. Um dos grandes problemas em Co-design é a geração da comunicação entre os diversos processos que estão sendo executados em software e aqueles sendo executados em hardware, particularmente a geração da interface entre os processadores e os componentes de hardware. A geração da interface é uma das tarefas mais sujeitas a erros e que atrasam o desenvolvimento do projeto de um sistema digital, devido ao fato de que para cada nova arquitetura alvo escolhida uma nova interface deve ser implementada, o mesmo acontecendo quando uma nova configuração de processos em hardware ou software é definida. Um segundo problema relacionado ao *Co-design* diz respeito a síntese das partes em *hardware* e *software* do sistema, ou *co-síntese*. Deve-se, a partir de uma única representação do sistema, implementar partes usando tecnologia distintas.

Neste trabalho de dissertação são estudados dois problemas inerentes ao *Co-design*: a co-síntese das partes do sistema em *hardware* e *software* e a geração automática da interface entre estas partes dentro do contexto do sistema PISH de *Co-design.* Como resultado, foi desenvolvida a ferramenta InterfPISH que é capaz de realizar a co-síntese do sistema digital assim como gerar automaticamente a comunicação entre os diversos processos em hardware e software e também a interface entre os componentes de hardware e processadores que compõem a arquitetura do sistema. Isto, permite o projetista se ater à funcionalidade do sistema e explorar de maneira mais

[1] FPGA (Field Programmable Gate Array) são dispositivos digitais que podem ser configurados pelo usuário.

eficiente as diversas opções de arquitetura àsua disposição.

Palavras-chaves: *hw/sw Co-design*, co-síntese, prototipação, geração de interfaces.

Abstract

Hardware/Software Co-design is a methodology used for the design of heterogeneous digital systems where part of the system functionality is implemented as software running on processors and the other part is implemented directly as hardware components (ASIC's, FPGA's). This methodology allows a single specification of a digital system to be implemented in several different configurations of hardware and software processes running on different target architectures. One of the big problems in Co-design is the generation of the communication among the processes that are executing in a software processor and the ones implemented in hardware, particularly, the interface generation between the processor and the hardware components. Interface generation is one of the most error prone tasks that delays the design of heterogeneous digital systems, due to the fact that for each new target architecture a new interface must be constructed. The same happens when a new hardware/software configuration is defined without changing the target architecture. A second problem related to hardware/software Co-design is synthesis of the hardware and software parts of the system, known as co-synthesis. From a single representation of the system the hardware and software parts that, which are very distinct in nature must be implemented.

In this work two problems of Co-design are addressed: co-synthesis and automatic interface generation. This is done in the context of the PISH Co-design system. As a result, the InterfPISH tool has been developed. This tool is able to make the co-synthesis of the hardware and software parts of a digital system, as well as the automatic generation of the interface between hardware and software. This allows the system designer to more efficiently explore the different architecture and configuration solutions available.

Key words: *hw/sw Co-design*, co-synthesis, prototyping, interface generation.

Capítulo 1: Introdução

1.1 MOTIVAÇÃO

O surgimento de novas técnicas de fabricação de circuitos CMOS permitiram a construção de componentes digitais a um custo relativamente baixo e que podem implementar sistemas de elevada complexidade. Devido a estas novas técnicas **[referência]** a aplicação de sistemas digitais em equipamentos eletrônicos teve um crescimento muito alto na última década. Equipamentos tais como brinquedos eletrônicos, eletrodomésticos, equipamentos médicos, equipamentos industriais, dispositivos controladores, sistemas para automóveis, aviões, etc. passaram a ter sistemas digitais embarcados como uma importante componente dos mesmos. A introdução de tais sistemas teve impacto na funcionalidade, bem como na redução do custo final do produto. O mercado consumidor de sistemas embarcados possui algumas características bem interessantes: extremamente competitivo, é um mercado heterogêneo, exige produtos específicos e está em constante evolução. A competitividade é marcante devido ao grande número de empresas que desenvolvem equipamentos que utilizam sistemas digitais embarcados; isto exige o lançamento constante de novos produtos em janelas de mercado cada vez menores, o que obriga o desenvolvimento destes sistemas em um tempo cada vez menor. A heterogeneidade do mercado consumidor implica que os sistemas embarcados devem atender a requisitos bastante diferentes entre si: um equipamento médico possui especificações bem diferentes de um brinquedo, por exemplo. Além de possuírem funcionalidades diferentes, as aplicações muitas vezes impõem restrições temporais aos sistemas embarcados. Desta maneira o projetista de sistemas embarcados deve lidar com projetos de natureza bastante distinta que atendam a requisitos temporais restritos e diferentes. Além disto, restrições de tempo real exigem dos sistemas digitais embarcados alto desempenho que muitas vezes não podem ser obtidos pela utilização de componentes de uso genérico como processadores de propósito geral. Os produtos que utilizam sistemas embarcados estão evoluindo cada vez mais, o que faz com que o sistema embarcado também deva ser flexível para permitir a rápida evolução dos equipamentos que o utilizam. Finalmente,

o uso dos sistemas embarcados tem tornado o custo da eletrônica embarcada um fator importante no custo total dos equipamentos atualmente desenvolvidos. Como resultado, reduções no custo de desenvolvimento e produção de sistemas embarcados provocam uma redução razoável no preço final dos equipamentos que apresentam eletrônica embarcada.

Fazendo uma síntese do cenário acima, o projetista de sistemas digitais embarcados precisa desenvolver sistemas que sejam ao mesmo tempo confiáveis, apresentem alto desempenho, sejam flexíveis, apresentem baixo custo e tudo isso em um tempo de projeto cada vez menor. Para permitir que o projetista de sistemas digitais embarcados desenvolva projetos com tais características, ambientes de projeto, como aquele mostrado na Figura 1. Este ambiente consiste em uma metodologia de projeto conjunta para as partes de um sistema digital em *hardware* e *software.* Este tipo de ambiente, em geral, deve analisar diferentes alternativas de projeto combinando componentes de *hardware* e de *software* e deve incluir uma metodologia de projeto bem definida que garanta a integração desses componentes de natureza distinta. Mais importante ainda, como um dos principais objetivos do projetista de sistemas digitais é desenvolver projetos no menor tempo possível, esta metodologia deve ser automatizada, incluindo ferramentas de CAD nas diversas etapas de projeto.

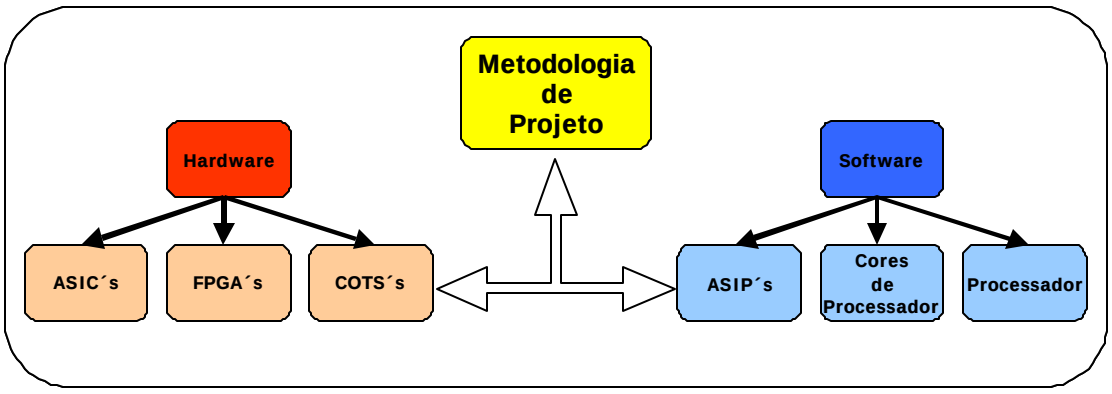

Figura 1: Metodologia de *Co-design*

Para tentar solucionar estes problemas surgiram as técnicas de *hardware/software Co-design* para o projeto de sistemas digitais embarcados. Estas técnicas associam a utilização de componentes de *hardware* tais como ASIC's

(*Application Specific Integrated Circuit*) e FPGA's (*Field Programmable Gate Arrays*), componentes discretos e componentes de *software*, tais como processadores, núcleos de processadores e ASIP's (*Application Specific Instruction Set Processor*) a uma metodologia de projeto que integra ambos componentes em uma única solução. A Figura 2 mostra uma metodologia genérica de *Co-design.* Uma das principais vantagens de se utilizar as técnicas de *Co-design* consiste em aliar o alto desempenho das aplicações desenvolvidas em *hardware* à flexibilidade e baixo custo das aplicações em *software* dentro de um ambiente integrado onde uma metodologia de projeto é aplicada visando garantir um rápido desenvolvimento do projeto de um sistema digital.

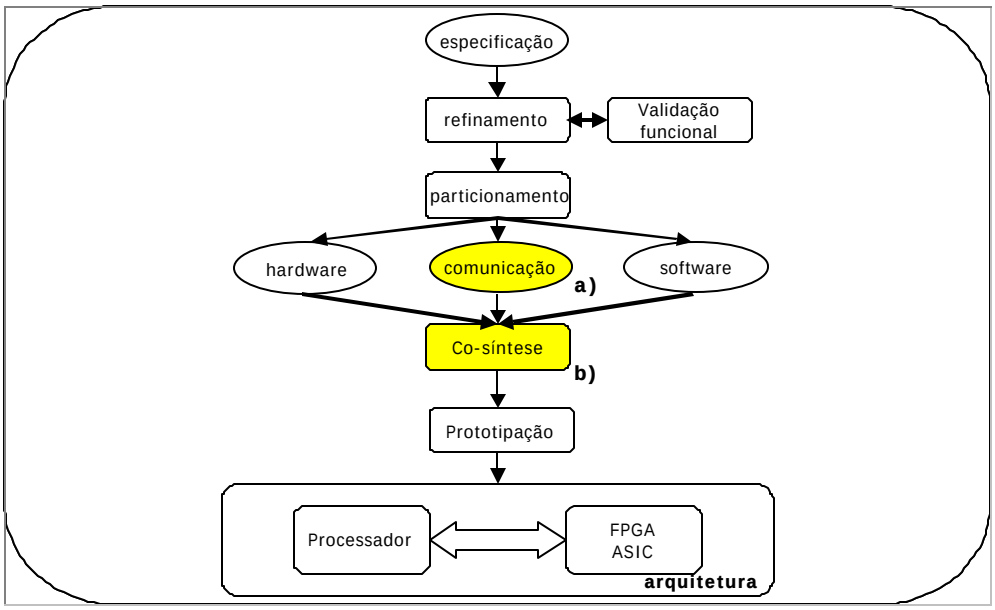

Figura 2: Metodologia de *hardware/software Co-design*

Como pode ser observado na Figura 2 a metodologia de *Co-design* é bastante ampla, abrangendo desde a especificação inicial do sistema, passando pela etapa de particionamento que define que partes do sistema serão implementadas em *hardware* ou em *software*. Nesta etapa surge um novo componente no sistema, a comunicação entre as partições (Figura 2a). Na etapa de co-síntese (Figura 2b) os subsistemas de *hardware* e *software* são sintetizados. A síntese é distinta para os dois subsistemas, para o subsistema de *software* devem ser gerados códigos executáveis para processadores específicos utilizados na arquitetura e para o subsistema de *hardware* devem ser

gerados, através de ferramentas de síntese, códigos para configuração de componentes de *hardware*.

Finalmente, na etapa de prototipação é gerado um protótipo a partir dos processadores e componentes de *hardware* que formam a arquitetura alvo do sistema digital embarcado. Esta etapa permite validar requisitos não funcionais do sistema tais como: desempenho, área ocupada, consumo de potência, custo de implementação, etc.

1.2 CO-SÍNTESE E GERAÇÃO AUTOMÁTICA DE INTERFACE

Duas etapas são importantes para que a metodologia de *Co-design* atinja seu objetivo de permitir ao projetista do sistema digital embarcado implementar rapidamente o seu projeto: a co-síntese e a geração automática da interface entre os subsistemas de *hardware* e *software*.

Uma vez particionado o sistema digital e definido os subsistemas de *hardware* e *software,* é necessária a descrição destes subsistemas em linguagens específicas que permitam a compilação para processador e a síntese para componentes de *hardware*. A Figura 3 mostra, de uma maneira geral, o que acontece entre o particionamento e a prototipação do sistema digital. Após a definição de que partes do sistema serão implementadas em *hardware* e quais serão implementadas em *software*. Estas descrições devem refletir o sistema particionado, garantindo a correta funcionalidade especificada originalmente. Isto implica que a tradução da descrição particionada deve ser feita sem erros o que pode ser uma tarefa bastante complicada quando se trata de sistemas digitais de porte razoável, porque erros podem ser introduzidos durante a tradução das descrições. Outro problema importante reside na natureza dos subsistemas que irão compor o sistema digital. Tal fato implica que o projetista ou projetistas do sistema digital devem ser especialistas em áreas diferentes, lidar com ferramentas diferentes e conhecer linguagens de descrição de natureza diversa.

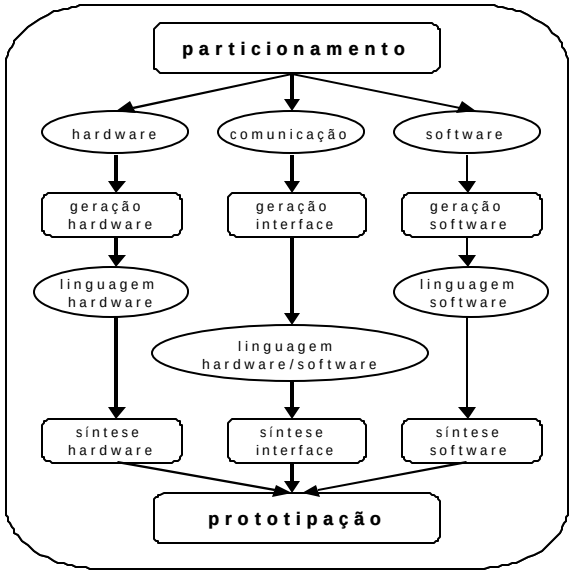

Figura 3: Geração de código e interface

1.2.1 Síntese de Interface

Um terceiro fato complicador também relacionado à natureza heterogênea dos sistemas digitais é o novo elemento que surge devido ao particionamento do sistema digital: a comunicação entre as partes de *hardware* e *software*, como pode ser visto na Figura 4. Torna-se necessária agora a inserção de um novo elemento no projeto do sistema digital: a interface entre os componentes de *hardware* e *software.* Além de ser um novo elemento, implicando em mais um projeto e implementação, a interface deve ser implementada tanto em *hardware* como em *software* e deve ser capaz de implementar a comunicação entre estes dois subsistemas. Agora, além de se preocupar em gerar código para os subsistemas de *hardware* e *software*, o projetista do sistema deve também se preocupar com a interação entre estas duas partes. Esta tarefa é muito tediosa, primeiro porque não acrescenta novas funcionalidades ao sistema, segundo ela envolve detalhes como protocolos de comunicação, temporização, sincronismo, etc. e finalmente porque a sua validação é muito trabalhosa envolvendo mecanismos de validação, tais como simulação, que devem ser realizados utilizando-se ao mesmo tempo ferramentas diferentes que precisam se comunicar entre si. Para piorar o cenário, a cada novo particionamento uma nova interface deve ser gerada, o que pode aumentar o tempo

de projeto caso várias partições devam ser testadas antes da escolha final do sistema a ser implementado.

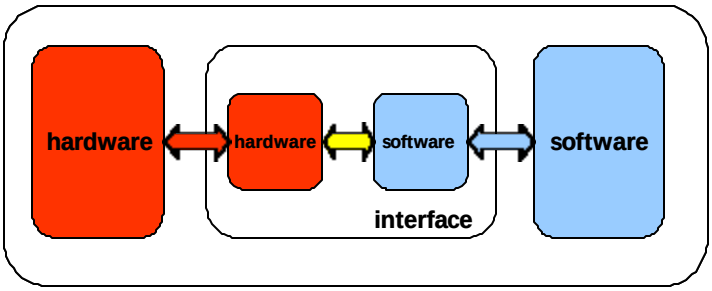

Figura 4: Interface *hardware/software*

1.2.2 Síntese de Hardware

Síntese de *hardware* implica na transformação de uma descrição de um sistema digital de um domínio para outro. O diagrama da Figura 5 (Diagrama de Gajski) **[referência]** mostra os três domínios em que um sistema digital pode ser descrito: comportamental, estrutural e físico. Dentro de cada um desses domínios a descrição pode ser feita em vários níveis de abstração. Caminhando de dentro para fora do diagrama, vai-se do nível mais baixo de abstração até o nível mais alto. Uma ferramenta de síntese de *hardware* é capaz de realizar a transformação de descrições que estão no domínio estrutural ou comportamental para gerar uma descrição no domínio físico.

O domínio comportamental descreve um circuito digital com relação à sua funcionalidade. Quanto mais alto o nível de uma descrição no domínio comportamental mais a mesma se parece com uma descrição de um *software* de alto nível. Exemplos de descrição comportamental podem ser máquinas de estado finito, equações booleanas, funções de transferência, etc.

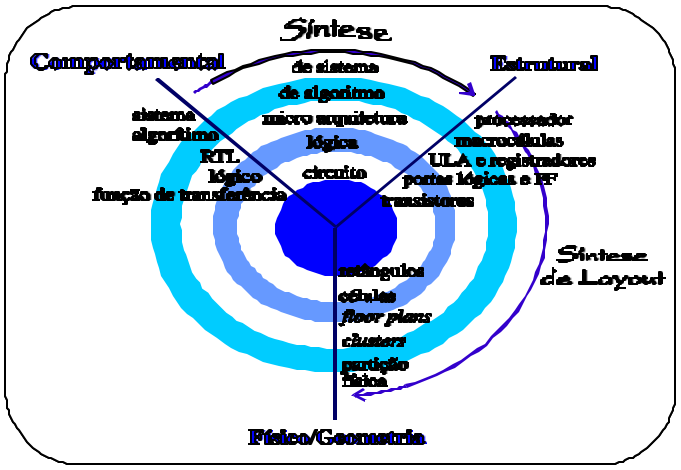

Figura 5: Diagrama de Gajski

No domínio estrutural o circuito digital é descrito pelo seu conjunto de componentes e como estes estão conectados. Neste tipo de descrição não há mais a noção de comportamento e sim de estrutura. Também existem vários níveis de abstração no domínio estrutural, podendo um circuito ser descrito através de portas lógicas, nível baixo de descrição até níveis de sistema onde na descrição do circuito podem aparecer componentes complexos como processadores, memória, subsistemas, etc.

Finalmente o último domínio é o físico. Este é o domínio de implementação do circuito digital. Descrições físicas podem variar desde arquivos de configuração para FPGA's até arquivos EDIF que permitem a confecção de circuitos integrados por uma Silicon Foundry[2].

Linguagens de descrição de *hardware* tais como VHDL (VHSIC[3] Hardware Description Language) [46] são capazes de descrever circuitos digitais nos domínios comportamental e estrutural.

1.3 ESTADO DA ARTE

A geração de comunicação entre subsistemas de *hardware* e *software* vem sendo

[2] *Silicon Foundry* é uma fábrica capaz de confeccionar circuitos integrados.

[3] VHSIC – Very High Speed Integrated Circuit

estudada por vários grupos de pesquisa em diversos trabalhos. Estes trabalhos procuram enfocar aspectos diferentes da geração de comunicação tais como: geração de comunicação para co-simulação[7], geração de *device drivers* para arquiteturas compostas de processadores e dispositivos[16], geração de comunicação para arquiteturas discretas[32], geração de comunicação para arquiteturas heterogêneas de sistemas em único *chip* (*System on a Chip*, SoC) [3] e geração de comunicação entre processos [31].

A geração de comunicação para sistemas embarcados é um problema que vem sendo tratado em diversos trabalhos. Alguns focam a geração de comunicação para arquiteturas heterogêneas discretas tais como geração de interface entre microcontroladores e dispositivos periféricos [37][43], outros abordam síntese de comunicação para sistemas distribuídos com tempo real [43][45][18][43], e outros focalizam a síntese de *device drivers* para controle de dispositivos em *hardware/software Co-design* [16]. Outros trabalhos focam em geração de comunicação em arquiteturas heterogêneas para sistemas implementados em um único *chip (SoC Design)* onde a integração de subsistemas que muitas vezes operam com *clocks* em freqüências diferentes é importante.

A geração de comunicação para co-simulação se destaca das demais formas devido ao seu objetivo que não é a geração de comunicação para implementação do sistema digital. Seu objetivo é permitir que simuladores dos subsistemas de *hardware* e *software* sejam capazes de simular o sistema digital como um todo de maneira sincronizada. A co-simulação pode ser utilizada quando se deseja validar a execução de um programa, normalmente em linguagem C, em um processador juntamente com a simulação de componentes de *hardware* descritos em uma linguagem de descrição de *hardware* (HDL) tal como VHDL. Este tipo de simulação é muito importante para a validação do sistema digital e pode ser muito complexa dada a necessidade de se trabalhar com ferramentas de simulação de fabricantes distintos, que normalmente não possuem mecanismos eficientes de comunicação entre si ou que apenas podem funcionar em conjunto de forma restrita. Exemplos de trabalhos em co-simulação podem ser encontrados em [7][8].

1.4 OBJETIVOS

O objetivo deste trabalho consiste na implementação de uma ferramenta capaz de realizar a co-síntese de um sistema digital particionado no contexto do sistema PISH de *Co-design*. Embora alguns trabalhos interessantes estejam sendo desenvolvidos atualmente tais como: Hasis, Cosyma, Symphony, que serão vistos com mais detalhes em seção posterior, estes não se aplicam às necessidades do sistema PISH de Co-design. Uma visão geral da técnica proposta pode ser vista na Figura 6.

No topo da figura tem-se uma descrição de sistema digital onde são definidas partes do sistema a serem implementadas em *hardware*, em *software* e uma parte de controle que gerencia a comunicação entre os módulos em *hardware* e em *software*. Esta descrição é feita em uma linguagem de alto, occam, nível capaz de especificar concorrência e comunicação. A primeira etapa do trabalho consiste em gerar uma representação interna do sistema acima descrito que independa da linguagem de especificação. Nesta etapa devem ser modelados o fluxo de controle e o fluxo de dados do sistema digital.

A segunda etapa consiste na representação do controle e do fluxo de dados como objetos. Desta maneira torna-se mais fácil realizar alterações em ambos.

Em seguida é realizada a extração de *Threads* do sistema digital. Um *Thread* é capaz de realizar apenas processamento sequencial. O sistema digital é composto de vários *Threads* trabalhando concorrentemente.

Posteriormente é realizada a inserção de E\S no sistema digital. Nesta etapa são utilizadas bibliotecas contendo os componentes de E\S disponíveis para a escolha do usuário.

A etapa seguinte consiste na geração de código que possa ser compilado por compiladores de *software* e sintetizado por ferramentas de *síntese* de *hardware*. O código gerado deve ser compatível com a arquitetura alvo escolhida pelo projetista, devendo levar em consideração o processador a ser utilizado, bem como a topologia de conexão entre os componentes que implementarão as partes em *hardware* e *software* do sistema.

Para implementação do sistema digital devem ser gerados códigos para os *Threads* concorrentes que realizam o processamento do sistema, bem como para a

comunicação entre os *Threads*. Também deve ser gerada automaticamente a interface entre as partes do sistema em *hardware* e as partes do sistema em *software*. Finalmente devem ser gerados mecanismos que permitam a interação do sistema digital com o mundo exterior.

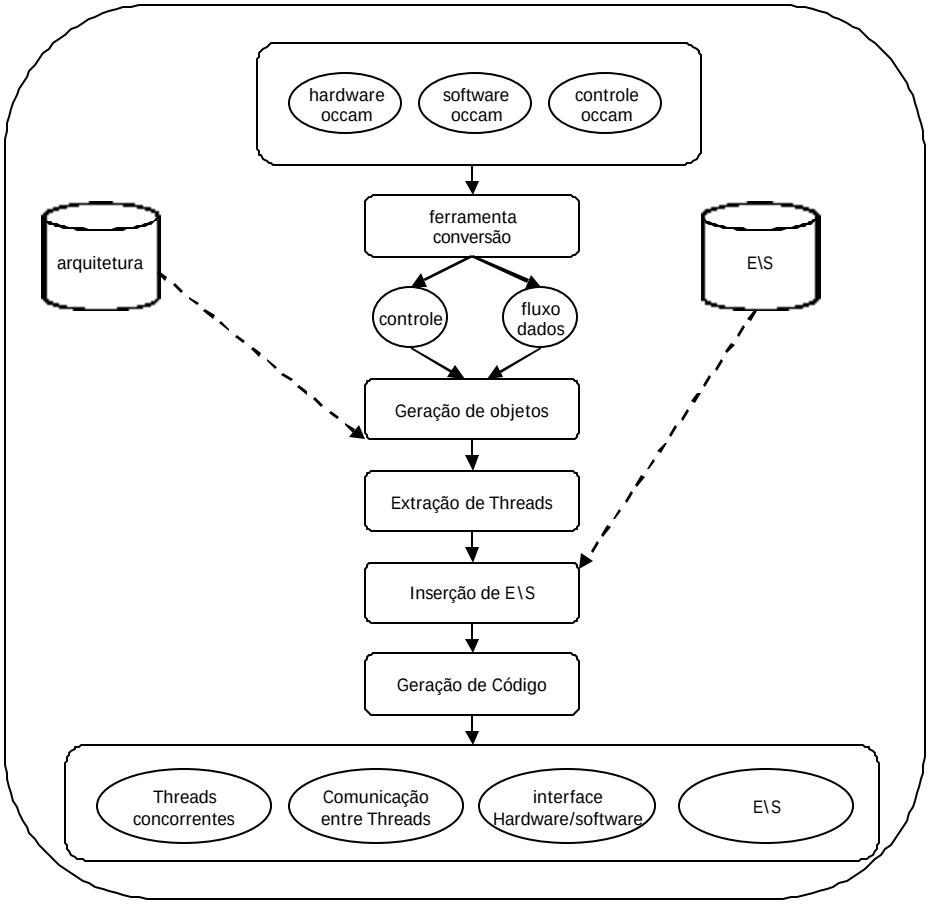

Figura 6: Visão geral do trabalho

1.5 ESTRUTURA DA DISSERTAÇÃO

No capítulo 2 são apresentados brevemente os trabalhos relacionados, os principais ambientes de *Co-design* existentes e suas características. No capítulo 3 é dada uma visão do sistema PISH de *Co-design* no qual este trabalho de dissertação está inserido, são abordados os módulos do sistema PISH e o contexto deste trabalho. No capítulo 4 é descrita uma proposta de ambiente de co-síntese para o sistema PISH. No

capítulo 5 é mostrada a ferramenta, resultado desta dissertação, que implementa a proposta apresentada no capítulo 4. Um estudo de caso que exemplifica o uso da ferramenta é discutido no capítulo 6. Finalmente, as conclusões e os resultados são mostrados no capítulo 7.

Capítulo 2: Ambientes de *Co-design*: O Estado da Arte

Neste capítulo são abordados os principais sistemas de *Co-design* atualmente existentes e suas principais características.

2.1 HASIS

Hasis foi desenvolvido no *Computer Engineering and Communication Networks Lab* (TIK) na Suíça. Este trabalho procura mapear programas que expressam fluxo de dados em arquiteturas heterogêneas compostas de componentes de *hardware* e *software*. Foi desenvolvida a ferramenta HASIS (*Hardware Software Interface Synthesis*) que é capaz de gerar automaticamente a interface entre componentes de *hardware* e *software* [31].

O tipo de aplicação característica abordada neste trabalho são aplicações de processamento digital de sinais, onde a ênfase é dada não ao controle da aplicação mas às transformações nos dados. Estas aplicações são normalmente expressas por fluxos de dados, onde um fluxo de dados pode ser visto como um grafo orientado no qual os nós representam computações e os arcos representam transferência de dados entre computações[23]. A Figura 7 mostra, de maneira simplificada, o mapeamento de um sistema digital em uma arquitetura formada por componentes de *software* (processador) e *hardware* (FPGA´s). Neste exemplo tem-se uma fonte de dados que são processados por um filtro digital e o resultado é mostrado em um display. Na representação em fluxo de dados a fonte, filtro e o destino são representados como nós. Neste caso a fonte é representada pelo nó s0, o filtro pelo nó fir e o destino pelo nó si. Para a implementação os nós são mapeados em componentes de *software*, como a CPU na figura, ou de *hardware*. No exemplo o filtro é mapeado em um componente FPGA. Os arcos são mapeados em barramentos, como o arco B que é mapeado no barramento B1.

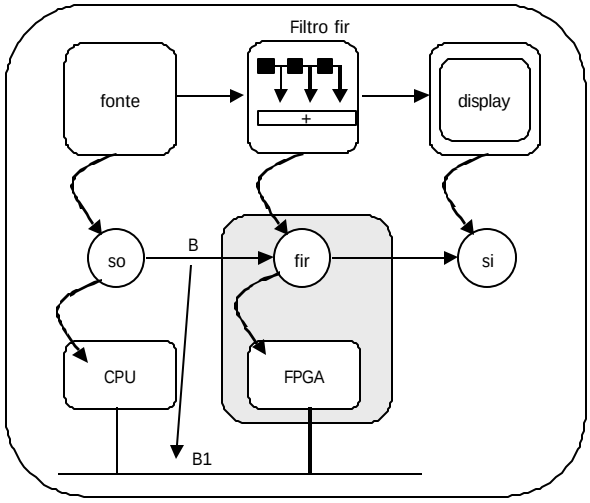

Figura 7: Mapeamento no Hasis

As restrições existentes neste trabalho estão relacionadas ao domínio de aplicação que é restrito à implementação de aplicações de fluxo de dados, especificamente às aplicações SDF (*Synchronous Dataflow Graph*). Isto restringe sua utilização em sistemas embarcados onde a ênfase é dada às aplicações que representam controle. Uma característica interessante deste sistema é que o mesmo vem sendo utilizado para aplicações em sistemas reconfiguráveis[32][32]. Estes sistemas são compostos de componentes de *hardware* que podem ser reconfigurados dinamicamente. Os sistemas de reconfiguração dinâmica permitem que um único componente de *hardware* possa implementar diversos circuitos digitais diferentes, o que implica num menor custo de implementação.

2.2 COSYMA

O sistema COSYMA (*Cosynthesis of Embedded Micro Architectures*) é uma plataforma para exploração do processo de co-síntese [21]. Esta plataforma é desenvolvida na Universidade Técnica de Braunschweig na Alemanha. É uma ferramenta que utiliza especificação homogênea, onde uma única linguagem descreve todas as partes do sistema digital e tem uma arquitetura alvo fixa.

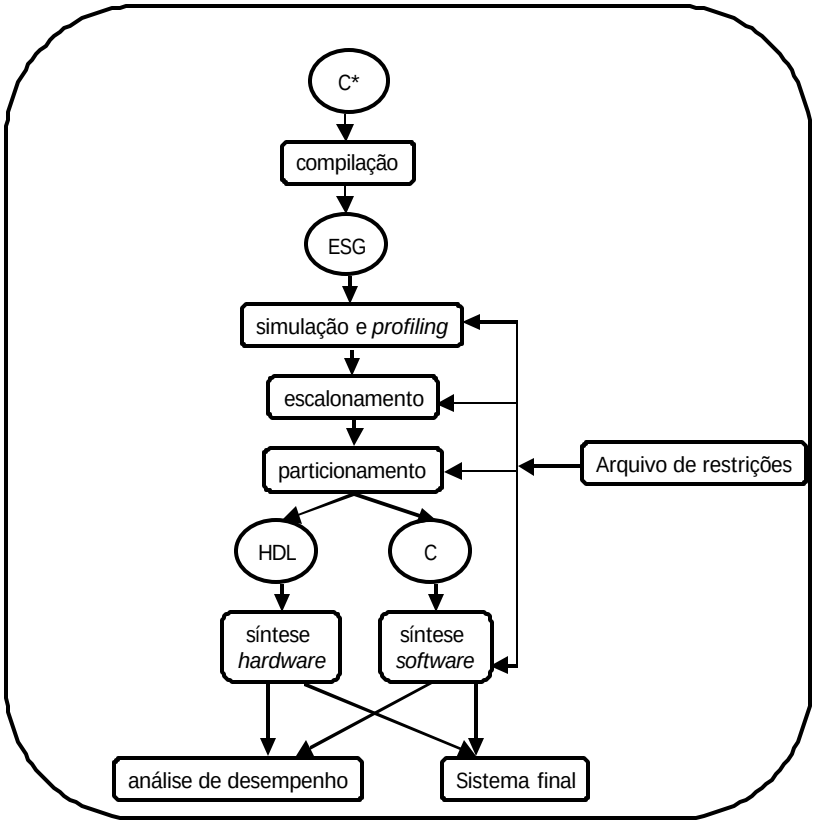

Figura 8: Fluxo de Projeto do COSYMA

A arquitetura alvo do COSYMA é fixa e composta de um processador RISC, memória e um co-processador em *hardware*. Este co-processador é gerado automaticamente pela ferramenta. A Figura 9 mostra o exemplo da arquitetura alvo do COSYMA, onde o processador, memória e co-processador compartilham um barramento comum. No co-processador são mapeados os segmentos de código que foram transferidos de *software* para *hardware.*

O fluxo de projeto do COSYMA é mostrado na Figura 8. Inicialmente é feita uma descrição do sistema digital utilizando-se uma extensão da linguagem C, C*, que permite descrever paralelismo. Esta especificação é convertida para um formato interno que utiliza uma variação de CDFG (*Control Dataflow Graph*) [1] chamada ESG (*Extended Syntax Graph*). São então realizadas simulações de modo a se obter um perfil (*profiling*) que são utilizadas pela etapa de particionamento, juntamente com as restrições definidas pelo usuário. A etapa de escalonamento é executada caso haja concorrência entre

processos.

O particionamento é realizado movendo-se iterativamente blocos de *software* para *hardware* até que as restrições de desempenho sejam atendidas. Neste ponto tem-se a implementação final do sistema digital.

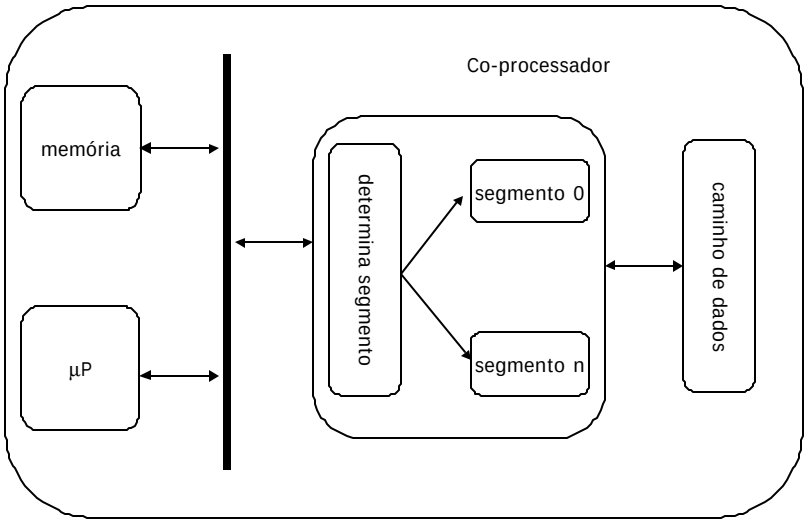

Figura 9: Arquitetura COSYMA

As principais restrições do COSYMA são sua arquitetura fixa, o que restringe o espaço de projeto disponível para o projetista, e a necessidade de dados de simulação para a realização do particionamento. Embora a utilização de dados de simulação permita uma implementação mais eficiente, muitas vezes não se pode simular o sistema de forma a se obter todos os dados necessários o que implica em alto custo computacional e atrasos na implementação do sistema.

2.3 SYMPHONY

Este projeto aborda a implementação de dispositivos SoC (*System on a Chip*) onde podem ser combinados componentes de *hardware* e *software* em uma arquitetura heterogênea dentro de um único chip. A solução apresentada pelo projeto SYMPHONY utiliza uma combinação de arquitetura escalável, ferramentas de CAD e bibliotecas parametrizadas de componentes[3].

A abordagem utiliza uma arquitetura em camadas como mostrada na Figura 10.

Na camada mais baixa, a camada física, o sistema digital é visto como composto de módulos de *hardware* e *software* conectados por interfaces bem definidas ao nível de circuito. Sobre a camada física é construída uma camada abstrata de comunicação e execução. Nesta camada são utilizadas bibliotecas (*packages)* em VHDL que abstraem os detalhes da comunicação em *hardware* e núcleos de sistema operacional (*kernels)* com funções de comunicação de alto nível que também abstraem os detalhes de comunicação em *software*. Finalmente, na última camada estão os códigos em C e VHDL das aplicações a serem implementadas na arquitetura heterogênea.

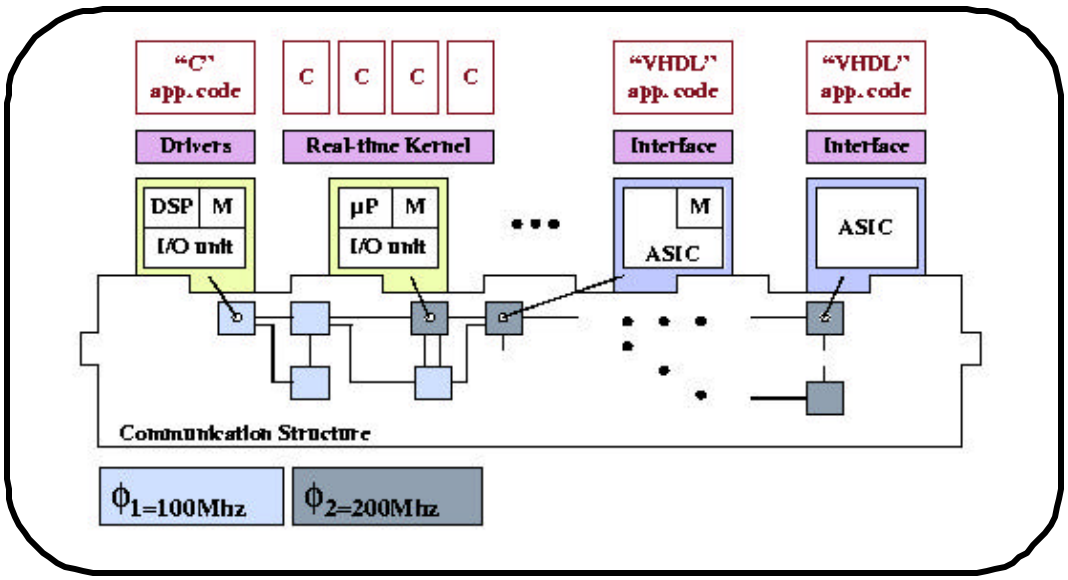

Figura 10:Arquitetura em camadas do Symphony

Na Figura 11 é mostrada a camada física no SYMPHONY. Nesta camada tanto os componentes de *hardware* quanto os processadores (componentes de *software*) são vistos como unidades de processamento e são conectados via componentes específicos de comunicação ou canais. Neste modelo tanto um processador quanto um componente de *hardware* possuem a mesma interface do ponto de vista de comunicação. Em destaque na figura está uma das unidades de processamento onde existe um núcleo (*core*) de um processador, memória e uma unidade de controle de entrada e saída (I/O UNIT). Esta unidade realiza a interface com os componentes de comunicação. A comunicação é realizada via a utilização de canais que implementam comunicação síncrona entre as unidades de processamento.

Deve-se observar que neste modelo cada unidade de processamento tem sua própria memória independente das demais unidades de processamento. Este tipo de arquitetura se aplica a projetos onde as tarefas estão separadas em várias unidades de processamento diferentes, como é o caso de sistemas implementados em SoC. Grande parte dos projetos que envolvem *hardware/software Co-design* **[referência]** utilizam uma arquitetura mais simples onde vários processos são executados em um único elemento de *hardware* ou de *software*.

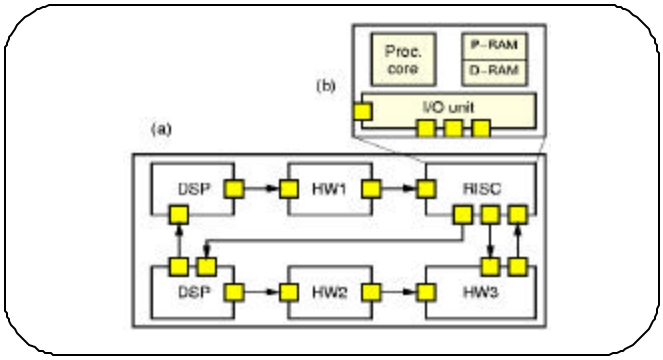

Figura 11: Camada física de sistema sintetizado via SYMPHONY

Capítulo 3: Sistema PISH de *Co-design*

Neste capítulo é apresentado o sistema PISH [11] (Projeto Integrado de *Software* e *Hardware*) de *Co-design* ou simplesmente PISH. Este sistema vem sendo desenvolvido pelo Grupo de Engenharia da Computação do Centro de Informática da Universidade Federal de Pernambuco [48].

O PISH é um sistema de *Co-design* que utiliza uma abordagem neutra para a especificação de sistemas digitais. Isto é, o sistema a ser implementado é inicialmente descrito sem que se saiba a priori que partes do sistema serão implementadas em *hardware* e que partes serão implementadas em *software*. Esta abordagem permite ao projetista abstrair os detalhes de como o sistema será realmente implementado, dando maior flexibilidade para a exploração do espaço de soluções possíveis.

Uma segunda característica do PISH é que o mesmo utiliza uma única linguagem de especificação para a descrição do sistema a ser implementado, occam [20]. Com isto o projetista não precisa lidar com várias linguagens para a especificação do sistema. Outra vantagem da utilização de apenas uma linguagem de especificação, é que não é necessária a criação de um ambiente de integração que permita que descrições feitas em diferentes linguagens interajam entre si. Embora a utilização de uma única linguagem seja uma característica interessante, a utilização de várias linguagens permite ao projetista escolher, para determinada parte do circuito, uma linguagem que se adeqüe melhor ao comportamento que se deseja implementar.

A terceira característica do PISH é a utilização de técnicas de verificação formal para a realização do particionamento do sistema inicialmente proposto. Esta característica garante formalmente que as alterações realizadas na descrição inicial do sistema digital pela inclusão de comunicação são corretas. Isto é muito interessante porque em sistemas de grande porte não se consegue simular todas as entradas do sistema e suas respectivas respostas. Garantindo-se formalmente que o sistema está corretamente particionado, uma grande economia de tempo na verificação do sistema a ser implementado é permitida.

Como resultado da etapa de particionamento tem-se a definição de que partes do

sistema digital serão implementadas em *hardware* e que partes serão implementas em *software*. Mas neste ponto ainda não está definida a arquitetura final onde o sistema será implementado. No estágio atual de desenvolvimento do sistema PISH não existem ferramentas que gerem, a partir do resultado do particionamento, código para a co-síntese das partes do sistema a serem implementadas em *hardware* e *software*. Isto implica que uma vez tendo o resultado do particionamento, o projetista deve realizar manualmente a descrição de cada uma das partes do sistema.

Finalmente, a última etapa do sistema PISH consiste na implementação do sistema. Nesta etapa uma arquitetura composta de um processador e FPGA's ou ASIC é utilizada para a implementação do protótipo do sistema digital.

Este capítulo está estruturado da seguinte maneira. Inicialmente será dada uma visão geral do PISH, onde serão descritos seus módulos, desde a especificação inicial do sistema digital que se deseja implementar, passando pela etapa de particionamento que define que partes do sistema ficarão em *hardware* e que partes serão implementadas em *software* até a sua implementação. Neste ponto será definido o papel deste trabalho dentro do contexto do projeto PISH. A seguir a linguagem occam, utilizada para especificação inicial do sistema digital, será descrita. As vantagens da linguagem e sua influência neste trabalho serão exploradas. Depois será explicada a estratégia de particionamento utilizada no PISH.

3.1 VISÃO GERAL

Na Figura 12 pode-se ver o fluxo de projeto utilizado pelo sistema PISH. Este é basicamente dividido em 3 (três) partes: (a) especificação e particionamento; (b) geração de *hardware* e *software* sintetizáveis; e (c) prototipação. A primeira das etapas, especificação e particionamento, é responsável por gerar, a partir de uma descrição inicial do sistema digital, uma nova descrição que é semanticamente equivalente à descrição original, mas que está particionada em processos a serem implementados em *hardware* e em *software*.

A segunda etapa, geração de *hardware* e *software* sintetizáveis, recebe como entrada o sistema já particionado e gera código tanto para ser executado em um processador (parte em *software*), quanto ser sintetizado em um ASIC ou FPGA (parte em

hardware). Nesta etapa o sistema a ser implementado é composto de duas partes de natureza distinta, *hardware* e *software*, que podem se comunicar. Isto implica que, além da geração de código das partes anteriormente descritas, deve ser gerada automaticamente a interface entre estas duas partes. A geração de interface entre componentes de *hardware* e *software* é uma tarefa muito sujeita a erros, quando executada de forma manual. Além disto, para cada nova configuração de componentes de *hardware* e *software* uma nova interface deve ser gerada. No estágio atual, a geração da interface não é realizada pelo sistema PISH, devendo ser feita manualmente pelo projetista do sistema digital.

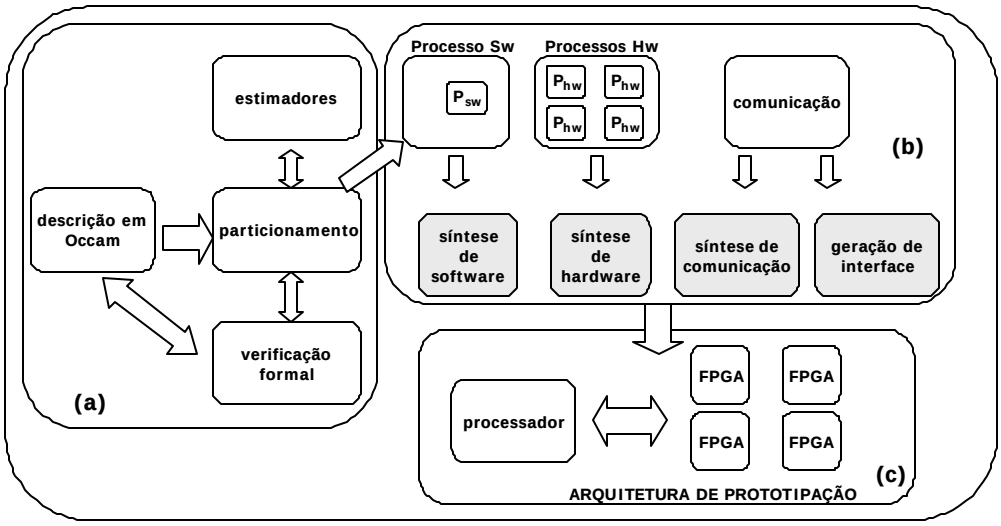

Figura 12: Fluxo de projeto do sistema PISH

A última etapa, prototipação, é a geração do protótipo real do sistema digital. É nesta etapa que os códigos gerados anteriormente são sintetizados para uma arquitetura composta de um processador e vários FPGA's. Neste ponto o sistema digital pode ser testado em condições reais de funcionamento.

3.1.1 Descrição e Particionamento

A primeira das etapas, especificação e particionamento, é mostrada na Figura 12a. Nesta etapa é fornecida como entrada para o sistema uma descrição textual do sistema digital na linguagem occam. Embora possua esta característica de neutralidade,

o projetista do sistema digital pode determinar que alguns trechos da descrição inicial sejam forçosamente implementados em *hardware* ou *software.* O sistema digital descrito é composto por processos que podem ser seqüenciais ou concorrentes. Na Figura 13 temos um modelo de uma descrição no sistema PISH. Pode-se ver os processos em seqüência (processos P1 e P2), e processos concorrentes (processos P0 e Pn e P0 e P3). Processos em seqüência podem transferir dados por variáveis compartilhadas. Já os processos concorrentes se comunicam de maneira síncrona via canais de comunicação, Figura 13.

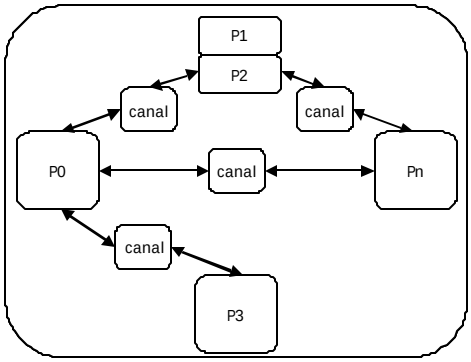

Figura 13: Sistema PISH – processos

Esta flexibilidade de se descrever processos em seqüência e concorrentes permite ao projetista explicitar paralelismo entre os diversos módulos que compõem o sistema a ser implementado. Não existe compartilhamento de variáveis entre os processos concorrentes, sendo toda a transferência de informações entre os processos concorrentes baseada em passagem de mensagens. A comunicação entre os diversos processos, passagem de mensagens, ocorre baseada na semântica de CSP [9]. Isto quer dizer que os processos se comunicam de maneira síncrona e bloqueante através de canais de comunicação.

Como mencionado, processos concorrentes não compartilham variáveis, toda a troca de dados entre estes processos ocorre através de canais de comunicação. A Figura 14 mostra um exemplo de comunicação por canal entre dois processos P0 e P1. Um canal conecta apenas dois processos, sendo a comunicação unidirecional. No exemplo da Figura 14 o canal conecta apenas os processos P0 e P1 e a direção da comunicação se dá de P0 para P1. Outra característica importante da comunicação entre processos no

PISH é que a mesma é síncrona bloqueante. Isto quer dizer que ambos os processos devem estar prontos simultaneamente para que a comunicação seja efetivada. Caso um dos processos esteja pronto antes do outro, este deve esperar até que o segundo processo seja também capaz de realizar a comunicação.

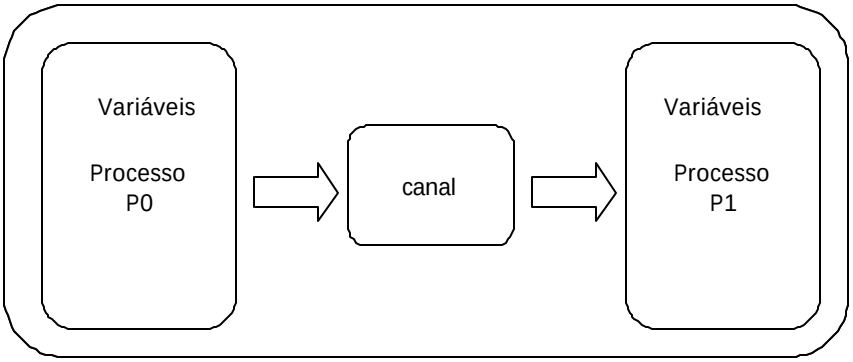

Figura 14: Comunicação entre processos

A partir da especificação inicial tem-se a fase de particionamento (Figura 12a) do sistema digital. A etapa de particionamento consiste na definição de quais módulos da descrição inicial serão implementados em *software* e quais módulos serão implementados em *hardware*.

Embora o sistema digital resultante seja uma nova descrição em occam, este possui a mesma semântica da descrição original. Pode-se garantir este fato porque, devido à semântica de occam ser formal, são utilizadas transformações algébricas sobre o programa original que não alteram sua semântica. O particionamento é implementado pela aplicação destas transformações sobre a descrição original[11].

3.1.2 Geração de Hardware e Software Sintetizáveis

Como saída do particionamento temos uma nova descrição do sistema digital que representa os processos a serem implementados em *hardware, software*, processos de controle e comunicação. Como foi dito, esta especificação do sistema particionado continua sendo uma descrição em occam, o que implica que esta descrição particionada não pode ser implementada diretamente.

O sistema PISH não sintetiza o *hardware* e *software* diretamente. A estratégia

utilizada é a geração de código que possa ser sintetizado por outras ferramentas, tanto comerciais quanto acadêmicas. Por serem linguagens de grande aceitação, foram escolhidas a linguagem C para geração da parte de *software* e a linguagem VHDL para geração de código em *hardware*. A tradução de occam para C e VHDL não é feita diretamente. Primeiro é feita a tradução para uma representação em formato intermediário do sistema particionado e a partir deste formato interno são gerados os códigos em C e VHDL

3.1.3 Prototipação

A etapa de prototipação consiste na geração de um protótipo real do sistema digital em uma arquitetura composta de elementos de *hardware* e *software*. No PISH foi desenvolvida a plataforma CHAMELEON II [6] para a implementação de protótipos. Esta plataforma é mostrada na Figura 15 e consiste de um microcontrolador da família 8051, memória e placas expansíveis contendo quatro FPGA's. Toda a interface com o usuário é implementada através do *host*, permitindo que o usuário selecione o sistema digital a ser implementado e particione os componentes de *hardware* nos diversos FPGA's da plataforma [6]. O microcontrolador (Figura 15b) tem três funções: servir de interface com o *host*, gerenciar a plataforma e executar a parte em *software* do sistema particionado. Na memória EPROM (Figura 15d) está armazenado o código de um *kernel* que gerencia a configuração dos FPGA's (os FPGA's são configurados com a parte em *hardware* do sistema particionado). A parte em *software* do sistema particionado é armazenada na memória RAM (Figura 15e) da plataforma. As placas com os FPGA's (Figura 15c) são expansíveis, podendo ser adicionadas mais placas com o objetivo de se aumentar a capacidade de implementação de *hardware* da plataforma **[referência para a plataforma CHAMELEON]**.

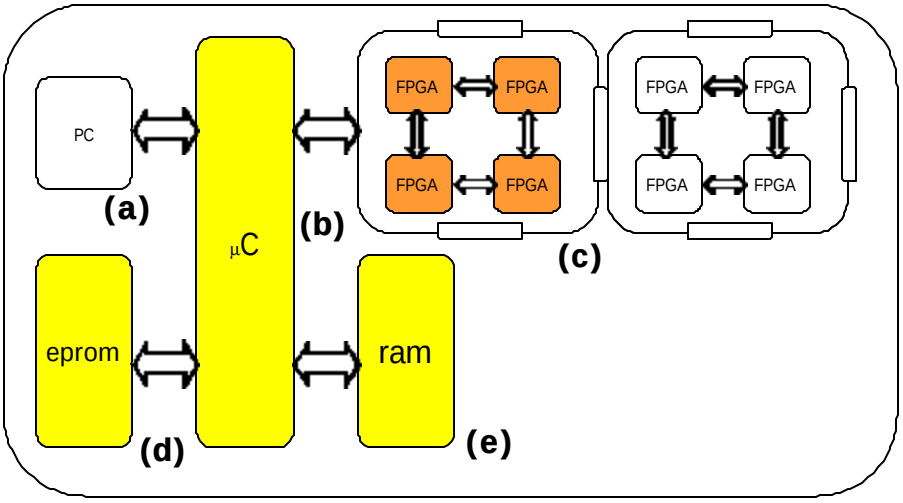

Figura 15: Plataforma CHAMELEON

3.2 OCCAM

Nesta seção será introduzida brevemente a linguagem de programação occam [1]. Serão vistas algumas características da linguagem, suas principais propriedades, vantagens e desvantagens de sua utilização como linguagem de especificação de sistemas digitais. É importante salientar que embora este trabalho procure se abstrair da linguagem de especificação utilizada para a descrição do sistema digital, há uma forte influência desta no desenvolvimento do mesmo. Especialmente no tocante à comunicação entre processos.

3.2.1 Introdução à Occam

Occam [1] é uma linguagem de programação baseada na semântica de CSP [9]. É uma linguagem que foi projetada para escrever algoritmos concorrentes, permitindo ao programador especificar processos concorrentes a partir de construtores simples. O fato de occam ser baseada na semântica de CSP implica que a comunicação entre os processos concorrentes especificados pelo programador utiliza o modelo de comunicação síncrona bloqueante.

Um programa em occam é composto por uma coleção de processos. Processos podem ser classificados em processos primitivos e processos combinados. Os tipos de

processos em occam são mostrados na Figura 16. Processos primitivos representam a execução de uma ação, de uma comunicação, de uma continuação ou de uma parada. Já processos combinados permitem descrever seqüências de processos, concorrência entre processos e determinar o fluxo do programa em occam.

3.2.2 Processos

Os processos primitivos de occam, mencionados anteriormente, podem ser vistos na Figura 16a e são: atribuição, entrada, saída, SKIP e STOP. O processo de atribuição associa o valor de uma expressão a uma variável. O processo de entrada realiza uma operação de recebimento durante uma comunicação por canal, também realizando associação de valor de expressão à variável. O processo de saída representa o envio do valor de uma ou mais expressões durante uma operação de comunicação. Embora a operação de atribuição e uma operação de comunicação associem valor de expressão à variável, a comunicação é utilizada quando processos concorrentes precisam enviar/receber valores. Não é possível realizar uma operação de atribuição de um valor gerado por um processo a uma variável pertencente a outro processo que seja concorrente com o primeiro, um processo apenas pode receber o valor de uma expressão oriunda de outro processo concorrente via uma operação de comunicação. SKIP é um processo que não realiza nenhuma ação, mas permite a continuação do programa. Já o processo de STOP representa *deadlock.*

Combinadores de processos são vistos na Figura 16b e podem ser: seqüência, paralelo, condicional, laço, alternação. O combinador de seqüência (SEQ) permite combinar vários processos para serem executados seqüencialmente. Já o combinador paralelo (PAR) permite construir processos concorrentes. O condicional (IF) permite que se tome decisões baseadas em condições booleanas. O combinador de laço (WHILE) permite que se repita a execução de um trecho do programa enquanto uma condição for verdadeira. O combinador de alternação (ALT) permite a execução não determinística de processos.

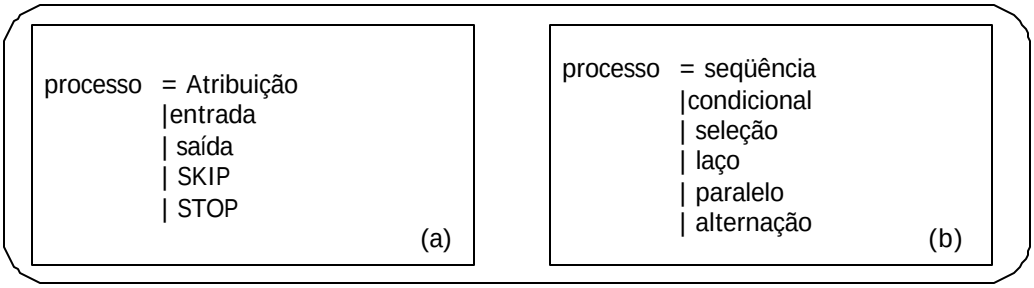

Figura 16: Processos em occam

3.2.3 Declaração e Escopo de Variáveis

A Figura 18 mostra exemplos de declarações de variáveis em occam. Em uma mesma declaração podem ser definidas várias variáveis que sejam do mesmo tipo. Isto é feito definindo-se o tipo da variável e colocando-se separados por vírgula 1 ou mais nomes das variáveis ({1, nome}). No exemplo da Na Figura 17 é mostrado um trecho de programa em occam e o escopo das declarações de variáveis. São declaradas as variáveis “max” e “min” e definidos seus escopos. O escopo de uma variável em occam vai da sua declaração até que o nível de indentação do programa retorne ao nível da declaração. Isto é mostrado pelas chaves identificando os escopos de “min” e “max” na figura. Logo após as declarações das variáveis vem um construtor PAR seguido de dois construtores SEQ em um nível de indentação a mais. Isto implica que os processos combinados definidos pelos construtores SEQ são concorrentes e mesmo assim enxergam as declarações de “max” e “min”, pois estes processos estão na região de escopo de ambas as declarações. Neste trecho de programa o primeiro processo envia o valor 10 através do canal de comunicação “c” (c ! 10). O segundo processo atribui à variável “max” o valor 20 e depois recebe o valor 10, também pelo canal de comunicação “c” e o armazena na variável “min” (c ? min). Após isto são feitas comparações com o valor de “max” utilizando-se o construtor “IF”. Este exemplo mostra a utilização de canais de comunicação, onde valores são transferidos entre processos concorrentes.

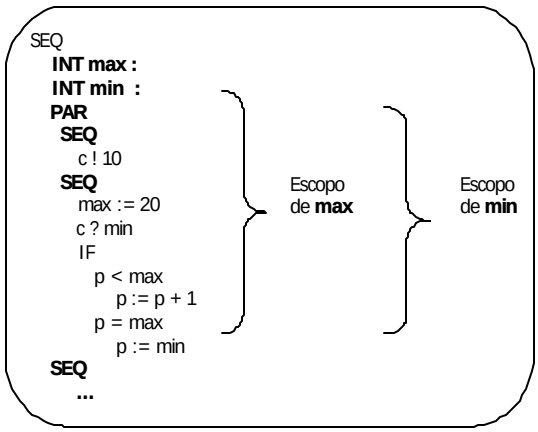

Figura 17: Escopo de variáveis em occam

Declaração = tipo de dado {$_1$, nome} :

INT A, B :
BOOL C, D:

Figura 18: Declaração de variáveis em occam

Na Tabela 1 são mostrados os tipos básicos suportados pela linguagem occam, juntamente com uma descrição de cada tipo.

BOOL	Valores booleanos : TRUE e FALSE
BYTE	Valores inteiros entre 0 e 255
INT	Valores inteiros com sinal, representados em complemento 2, utilizando o tamanho de palavra mais eficiente da implementação
INT16	Valores inteiros com sinal, representados em complemento 2, implementados com 16 bits
INT32	Valores inteiros com sinal, representados em complemento 2, implementados com 32 bits
INT64	Valores inteiros com sinal, representados em complemento 2, implementados com 64 bits
REAL32	Números em ponto flutuante armazenados utilizando-se 1 bit de sinal (0 para valores positivos, 1 para valores negativos) , 8 bits para o expoente e 23 bits para a mantissa. Padrão ANSI/IEEE 754-1985
REAL64	Números em ponto flutuante armazenados utilizando-se 1 bit de sinal (0 para valores positivos, 1 para valores negativos) , 11 bits para o expoente e 52 bits para a mantissa. Padrão ANSI/IEEE 754-1985

Tabela 1: Tipos básicos de dados em occam

3.2.4 Comunicação

Comunicação em occam é utilizada para troca de informações entre dois processos concorrentes. A comunicação entre processos em occam é baseada na semântica de CSP, sendo a comunicação síncrona. Isto significa que ambos os processos devem estar prontos simultaneamente para que a comunicação se concretize.

A comunicação em occam é implementada por canais de comunicação. Estes canais, assim como as variáveis, devem ser declarados e seu escopo é determinado da mesma maneira que o escopo das declarações de variáveis. Os canais de comunicação permitem a comunicação unidirecional entre dois processos. Outra característica importante dos canais em occam é que os mesmos sempre transferem de acordo com o tipo de dado especificado na sua declaração. Em occam existe ainda o construtor PROTOCOL, que define o conjunto de tipos de dados que são transferidos por um canal de comunicação. Na Figura 19 são mostradas as definições em occam de protocolo e um exemplo. A primeira linha da figura mostra a definição de protocolo em função de protocolo simples e protocolo seqüencial. Protocolo simples é representado por um único tipo de dados (2ª definição da figura). Já o protocolo seqüencial é composto por um ou mais protocolos simples (3ª definição). Isto é representado pela expressão "{1; protocolo simples}". A última linha da figura mostra um exemplo onde é definido o protocolo TESTE composto dos tipos INT, BOOL e INT. Isto significa que um canal que utilize este protocolo sempre irá transferir um valor do tipo INT, seguido de um valor do tipo BOOL e outro do tipo INT novamente.

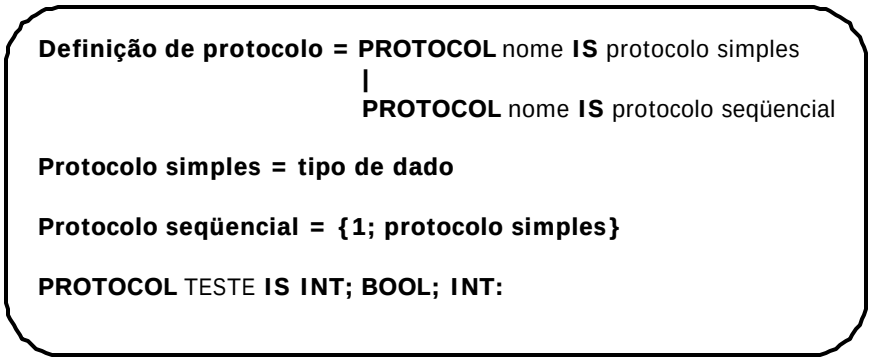

Figura 19: Protocolo em occam

A Figura 20 mostra a declaração de canais em occam. A primeira linha da figura mostra o tipo do canal definido em função do protocolo. A segunda linha define a declaração de canal como sendo um tipo de canal seguido de um ou mais nomes ({1, nome}). Finalmente na última linha é dado o exemplo da declaração dos canais “ch0” e “ch1” que utilizam o protocolo TESTE mostrado na Figura 19.

```
Tipo de canal = CHAN OF protocol

Declaração de canal = tipo de canal {1, nome} :

CHAN OF TESTE ch0, ch1 :
```

Figura 20: Declaração de canal em occam

3.3 ESTRATÉGIA DE PARTICIONAMENTO

O objetivo do particionamento é determinar, de forma automática, que elementos do sistema digital devem ser implementados em *hardware* e quais devem ser implementados em *software*. A estratégia de particionamento utilizada no PISH é mostrada na Figura 21. Ela é composta de quatro fases: *splitting*, *classification*, *clustering* e *joining*. Seu objetivo é receber como entrada uma especificação inicial do sistema digital em occam e retornar uma especificação semanticamente equivalente do mesmo sistema particionada em componentes a serem implementados em *hardware* e *software*. Na etapa de particionamento são utilizadas leis e regras algébricas na transformação da especificação original, as quais preservam a semântica da especificação [28][28][29]. Como resultado do particionamento tem-se uma nova especificação também em occam composta por processos a serem implementados em *hardware* e *software*. Em PISH o problema do particionamento pode ser dividido em duas etapas: transformação da especificação original com preservação semântica; e seleção, utilizando-se estimadores de custo de quais elementos devem ser implementados em *hardware* e quais em *software*.

A Figura 21 mostra as 4 fases que compõem a etapa de particionamento do

sistema digital. A primeira fase do particionamento é o *spliting*. Esta fase recebe como entrada a especificação original em occam do sistema digital e fornece como saída uma nova especificação também em occam formada por processos simples concorrentes. A etapa seguinte, *classification* e *clustering*, é responsável por definir alternativas de implementação para cada um dos processos originais e como estes serão agrupados baseados em custo de implementação. Finalmente a etapa de *joinning* fornece como resultado uma descrição final também em occam do sistema particionado. As etapas de *spliting* e *joinning* realizam transformações algébricas que não alteram a semântica da descrição inicial do sistema digital. Significando que nestas etapas são geradas novas descrições, também em occam, que possuem a mesma semântica da etapa anterior. Cada uma das etapas do particionamento é descrita com mais detalhes nas seções seguintes.

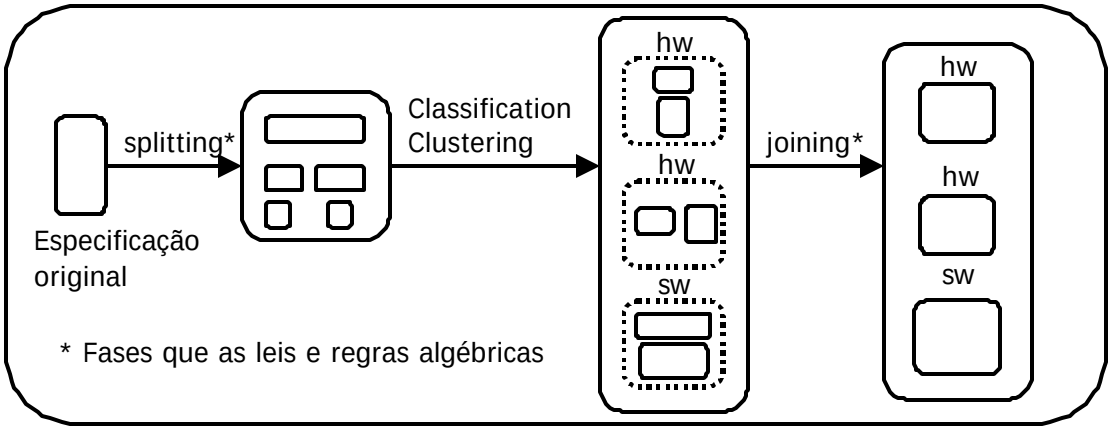

Figura 21: Visão geral do particionamento

3.3.1 Etapa de Splitting

Na etapa de *splitting* a especificação original é transformada em uma forma normal com o formato mostrado na Figura 22. Neste formato tem-se as declarações de canais, um combinador paralelo e diversos processos P_1, P_2, ..., P_m em paralelo. Cada um dos processos P_1, P_2, ..., P_m é um processo indivisível chamado de processo simples [24]. Durante a etapa de *spliting* todos os processos são quebrados recursivamente até que a descrição fique na forma normal. Vale ressaltar que a semântica da descrição original é preservada. Na Figura 23 é mostrado um exemplo de uma transformação de

uma especificação original para a forma normal. Nesta figura tem-se uma especificação original de um sistema em occam.

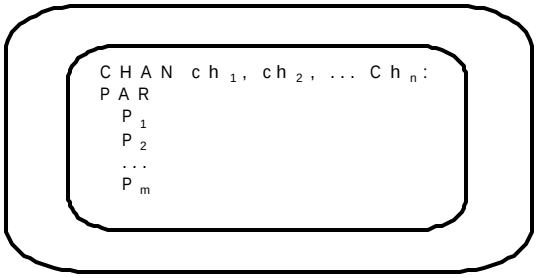

Figura 22: Forma normal

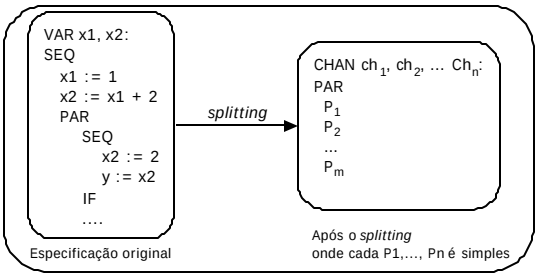

Figura 23: Fase de *splitting*

3.3.2 Etapa de Classification

A etapa *classification* é responsável por definir, para cada processo original, alternativas de implementação com relação ao grau de paralelismo com que cada processo original pode ser implementado. Nesta etapa é definido se um processo será implementado de forma completamente seqüencial, se partes do processo podem ser implementadas em paralelo ou se o mesmo pode ser quebrado me vários processos que podem ser executados em paralelo. As alternativas de implementação podem ser seqüencialmente, parcialmente paralelo e paralelo. Para cada processo original um conjunto de alternativas é definido, mas apenas uma dessas alternativas é escolhida para a fase de *clustering*. Essa escolha pode ser feita pelo projetista ou de forma automática.

3.3.3 Etapa de Clustering

A etapa de *clustering* tem como objetivo agrupar os processos originais em *clusters* e determinar através de uma função de custo quais *clusters* serão implementados em *hardware* e quais em *software*. O algoritmo utilizado para realização desta etapa é um algoritmo baseado em *clustering* hierárquico multi-estágio. Este algoritmo utiliza os dados contidos em uma matriz de distâncias (Figura 24a) que contem os graus de proximidade entre todos os pares de processos originais. O grau de proximidade é calculado utilizando-se métricas tais como: comunicação entre os processos, grau de paralelismo, possibilidade de implementação em *pipeline*, multiplicidade de atribuições e o resultado da fase de *classification*. Utilizando-se a

matriz, é construída a árvore de *clustering* (Figura 24b) onde os processos mais próximos são agrupados dois a dois a cada iteração do algoritmo. Os *clusters* são definidos do corte na árvore em determinado nível. Para cada nível da árvore é calculado um custo de implementação sendo a linha de corte da árvore definida pelo nível de menor custo. Na árvore da figura verifica-se que o menor corte (c = 15,3) define os *clusters* de *hardware* (P1, P2, P3 e P4) e os de *software* (P5). O resultado da etapa de *clustering* é uma nova descrição em occam (Figura 24c).

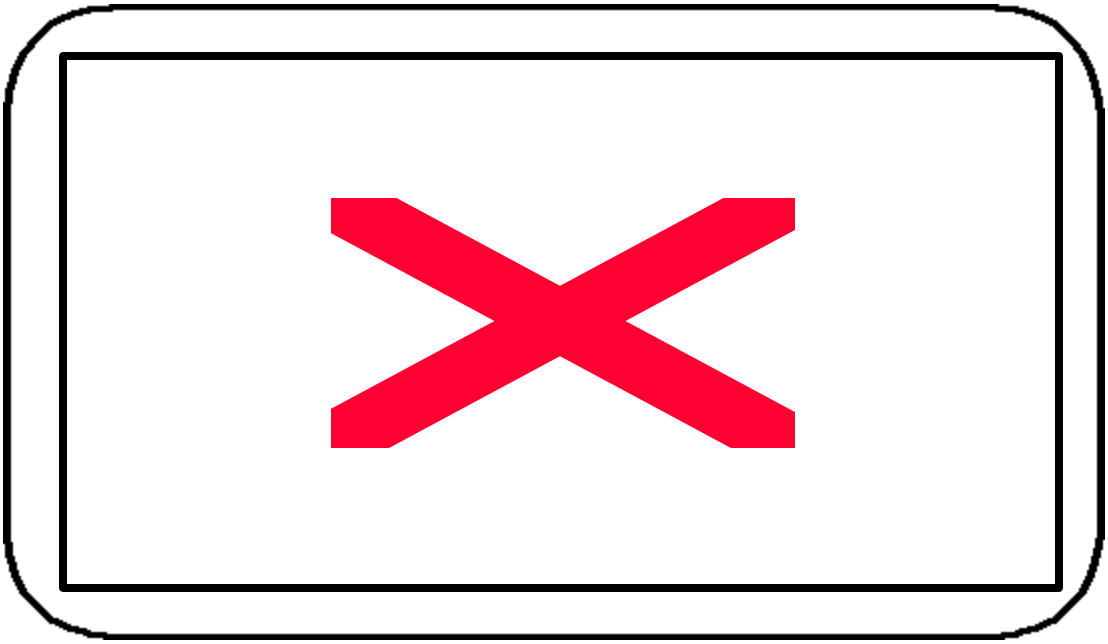

Figura 24: Etapa de *clustering*

3.3.4 Joining

A fase de Joining [24] é responsável por combinar os processos originais alterados pela fases de *splitting* e *clustering,* e fornecer uma descrição em occam representando o sistema final particionado. Nesta etapa procura-se eliminar os processos controladores introduzidos nas etapas anteriores e deixar os componentes (processos em *hardware* ou *software*) do sistema na forma mostrada na Figura 25. Nesta forma cada processo, que pode ser implementado tanto em *hardware* como *software*, é representado por uma comunicação onde são recebidos dados de outros processos ou do mundo exterior via comunicação (ch1 ? x). Q representa a combinação dos processos

resultantes na etapa de *clustering*. Vale lembrar que ao final da etapa de *clustering* tem-se como resultado *clusters* formados por vários processos simples. Estes são combinados em um único processo Q. x representa as variáveis utilizadas e atribuídas pelo processo Q, enquanto x´representa apenas as variáveis atribuídas por Q. Na prática isto implica que o processo Q retorna os valores das variáveis atualizados.

```
VAR x: SEQ (ch1 ? x, Q, ch2 ! x´)

Onde x = USED(Q) U ASS(Q), x´= ASS(Q)
```

Figura 25: Forma ideal de processos após o Joining

Na Figura 26 mostra o formato da descrição do sistema em occam antes e depois da etapa de *joining*. Como se pode ver, o resultado do *clustering* é composto de vários conjuntos de processos em paralelo. Como exemplo pode-se ver o conjunto de processos agrupados em F que vão de P1 a Pi. Também destaca-se o número de processos controladores que é maior que 1. Após a etapa de *joining* estes processos são agrupados. Olhando novamente os processos em *software*, vê-se que os mesmo foram agrupados em um único processo P. Ao final do *joining* os processos controladores são agrupados em um único processo controlador, como mostra a Figura 26

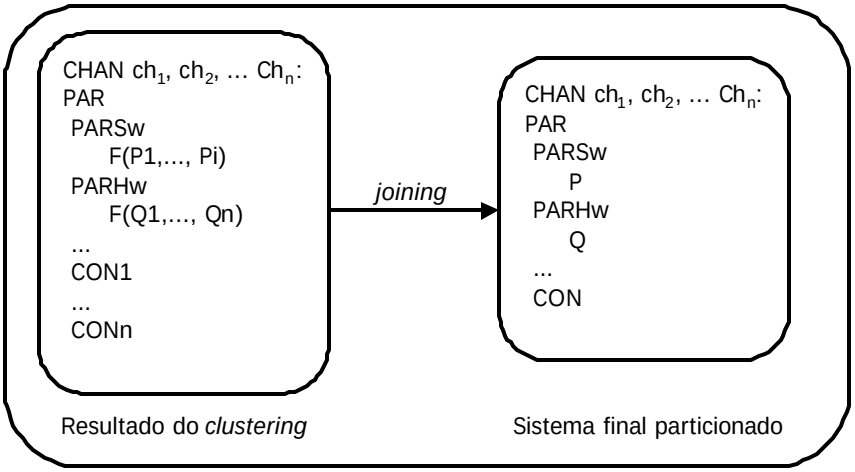

Figura 26: A fase de Joining

3.4 SISTEMA PARTICIONADO

O sistema PISH é capaz de realizar o particionamento automático a partir da especificação inicial do sistema digital feita em occam. O particionamento é realizado através da aplicação de regras de transformação que permitem modificações na especificação inicial e da utilização de estimadores que permitem fazer uma estimativa do custo de implementação de partes do sistema em *hardware* ou *software.* Como resultado da etapa de particionamento é gerado um conjunto de processos concorrentes que compõem o sistema digital final. Pode-se observar na Figura 27 o resultado do particionamento em *hardware* e *software* resultante da execução de PISH. O sistema particionado possui algumas características importantes a serem utilizadas pelo ambiente de co-síntese proposto: são especificados quais processos serão implementados em *hardware* e quais em *software*, é introduzido um novo processo de controle, são especificados os tipos de protocolo dos canais de comunicação utilizados. O sistema particionado também é descrito em occam.

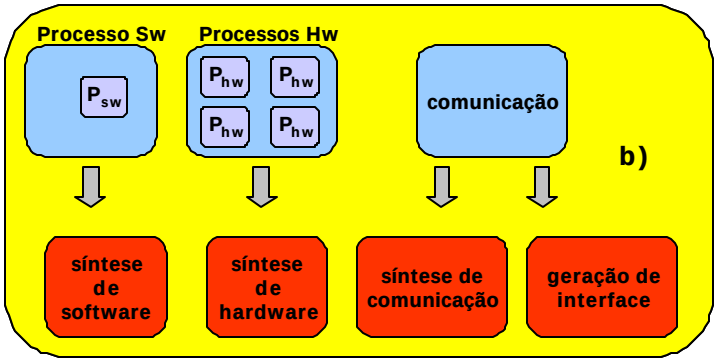

Figura 27: Saída do particionamento

Uma tarefa relevante de um particionador é permitir a introdução de um novo processo de controle. Este processo é introduzido para garantir a sincronização dos diversos processos concorrentes que compõem o sistema digital. Ele não realiza operações lógicas e aritméticas, realizando apenas comunicação com o objetivo de sincronizar os processos concorrentes. Toda comunicação entre os processos em *software* e os processos em *hardware* é feita através do processo de controle. Também passam pelo processo de controle das comunicações entre processos que estão

somente em *hardware*.

A próxima tarefa do sistema de particionamento é identificar o protocolo dos canais, possibilitando assim que seja implementada a comunicação entre os processos concorrentes. Estes canais representam comunicação unidirecional entre dois processos, podendo os mesmo serem ambos em *hardware*, ambos em *software* ou um em *hardware* e o outro em *software*. Cada canal sempre transfere todos os tipos de dados especificados no seu protocolo.

A última tarefa do sistema de particionamento é produzir uma nova descrição em occam. Esta nova descrição permite a utilização de ferramentas já existentes para a geração de uma representação em Rede de Petri do sistema particionado. Na Figura 28 é mostrada a saída do particionamento como uma descrição em occam onde são mostrados os processos a serem implementados em *hardware* e em *software,* e o processo de controle.

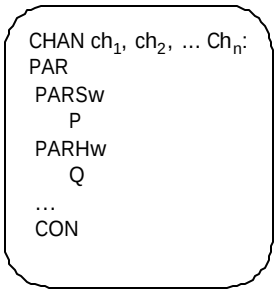

Figura 28: Sistema particionado em occam

Capítulo 4: Proposta de um Ambiente de Co-síntese

Neste capítulo é descrita a proposta de um ambiente de co-síntese que permita a geração de código sintetizável para a implementação de sistemas digitais. Tal ambiente de co-síntese procura preencher a lacuna existente dentro do sistema PISH de *Co-design*, conforme visto na seção 3.1.2. Outra preocupação deste ambiente é torná-lo independente da linguagem occam, utilizada tanto para especificação quanto para o particionamento do sistema digital. Serão abordados os problemas inerentes à co-síntese de sistemas digitais: a modelagem de processos, a comunicação entre processos, a geração automática da interface entre componentes em *hardware* e *software* e finalmente a comunicação do sistema digital com o mundo exterior.

4.1 DESCRIÇÃO DO AMBIENTE

Na Figura 29 é mostrado o fluxo geral utilizado para implementação do ambiente de co-síntese. Na figura as áreas claras representam atividades já realizadas dentro do PISH, sendo as áreas destacadas desenvolvidas neste trabalho.

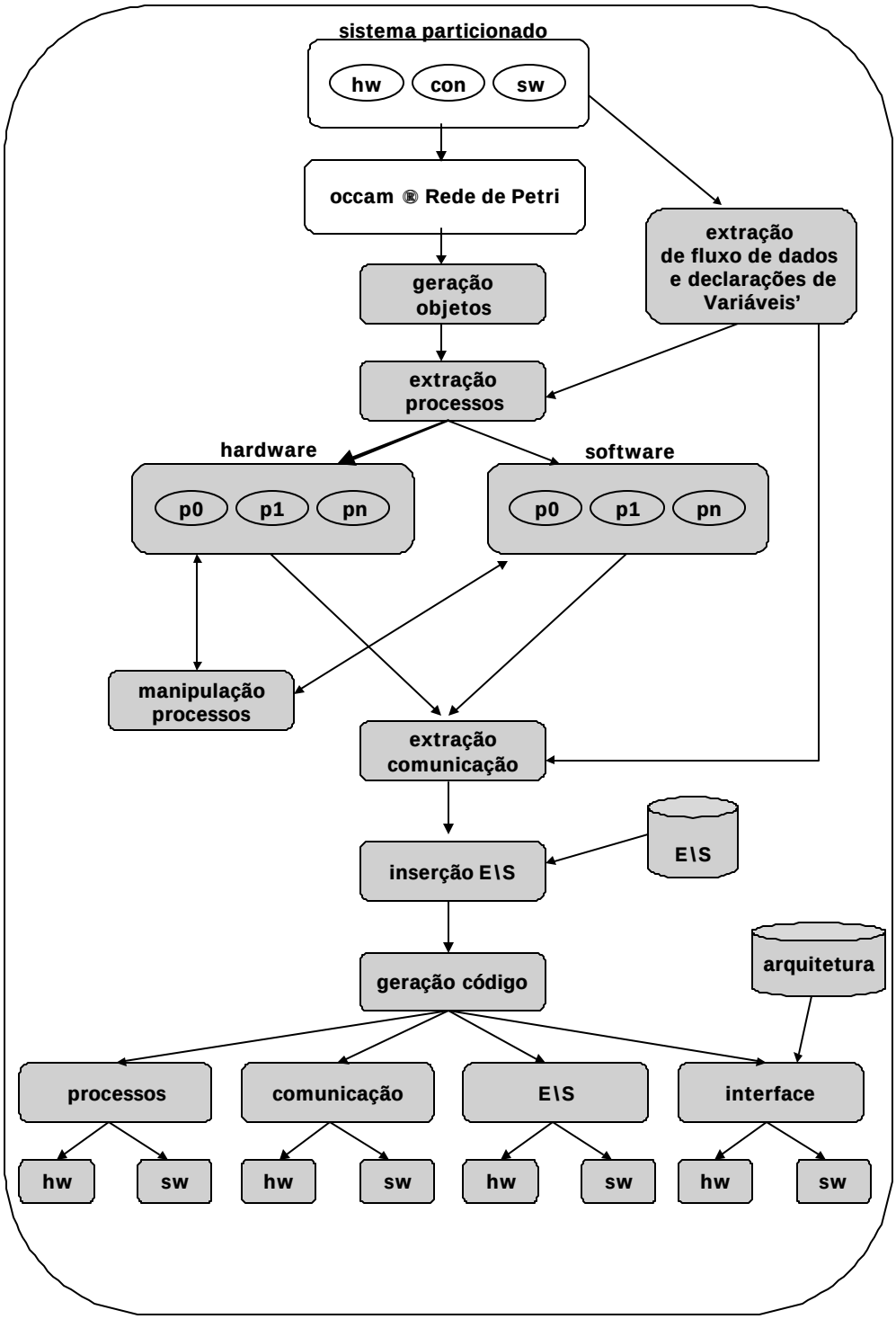

Figura 29: Ambiente de co-síntese

Como visto na seção 3.3 o sistema PISH realiza o particionamento automático de uma especificação original em occam. Este particionamento fornece como resultado uma nova especificação, semanticamente equivalente à especificação original, representando o sistema particionado. Na etapa de particionamento a especificação em occam é transformada em uma descrição em Redes de Petri, que é utilizada para o cálculo das métricas usadas durante o *clustering* do particionamento. Desta maneira já existe no PISH ferramentas capazes de gerar uma representação em Redes de Petri a partir de uma especificação em occam. Esta representação é implementada como um série de arquivos texto representando a rede de controle do sistema digital, os tipos das transições, grau de paralelismo, etc.

A primeira etapa deste trabalho consiste na geração de uma representação orientada a objetos da Rede de Petri do sistema digital. Desta maneira fica mais fácil a manipulação dos elementos das Redes de Petri.

Na etapa seguinte são extraídos processos seqüenciais da rede. A estes processos são associados os fluxos de dados extraídos da especificação do sistema particionado. Este fluxo de dados é convertido para um formato independente da linguagem de especificação e associado aos processos seqüenciais. Além do fluxo de dados também são associadas as declarações de variáveis utilizadas por cada processo seqüencial. Estas declarações também são retiradas da especificação do sistema particionado e transformadas em objetos num formato independente da linguagem de especificação.

Embora o PISH realize o particionamento automático, o ambiente de co-síntese permite que o particionamento seja alterado manualmente de maneira simples. Na etapa de manipulação de processos o usuário pode mover processos seqüenciais de *hardware* para *software* e vice-versa.

O ambiente de co-síntese identifica a comunicação entre os processos, determinando os canais de comunicação e os seus respectivos processos remetentes e destinatários. Também nesta etapa são identificados os canais que se comunicam com o mundo exterior (entrada e saída do sistema). A comunicação extraída é utilizada posteriormente para a geração da comunicação entre processos em *hardware*, em *software* e entre um processo em *hardware* e outro em *software*. A comunicação é

extraída das declarações de canais da especificação do sistema particionado e também transformada para formato interno independente da especificação.

Na etapa seguinte, inserção de E/S, são gerados processos de entrada e saída para permitir a comunicação do sistema com o mundo exterior, principalmente com dispositivos de entrada e saída. Estes processos de E/S estão armazenados em uma biblioteca da arquitetura alvo e têm características semelhantes aos processos extraídos da Rede de Petri que possibilitam o uso dos mesmos canais de comunicação utilizados pelos processos seqüenciais.

A última etapa, geração de código, consiste na geração de código compilável, para as partes do sistema digital a serem implementadas em *software,* e código sintetizável, para as partes do sistema digital a serem implementadas em *hardware*. Devem ser gerados códigos em *software* e *hardware* para os processos do sistema, para a comunicação entre processos de mesma natureza (ambos em *hardware* ou em *software*), geração de código para a interface quando os processos são de natureza diferente e finalmente código de E/S, onde são implementados os processos de comunicação com dispositivos de entrada e saída.

4.2 INTRODUÇÃO ÀS REDES DE PETRI

Redes de Petri permitem a representação gráfica de sistema concorrentes. São uma técnica consolidada para a representação de sistemas concorrentes [41]. Uma rede de Petri é formada por *lugares, transições, arcos e marca*.

Arcos são elementos direcionados e ligam sempre um lugar a uma transição ou uma transição a um lugar. Por exemplo, o lugar de saída de uma transição é o lugar cujo arco sai da transição para este lugar.

Dois lugares não podem estar conectados entre si, assim como duas transições também não podem. Um arco sempre liga uma transição a um lugar ou um lugar a uma transição. Uma Rede de Petri simples é mostrada na Figura 30. Nela podem ser identificados os elementos que compõem a rede. Os lugares p0 e p1 estão conectados a transição t0 via arcos.

Marca ou *token* significa uma informação atribuída a um lugar. A existência de

uma marca em todos os lugares de entrada de uma transição permitem o seu disparo, resultando na eliminação das marcas dos lugares de entrada da transição e o surgimento de marcas nos lugares de saída da transição.

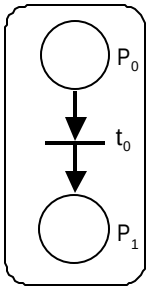

Figura 30: Elementos da Rede de Petri

Na Figura 31 pode ser visto o disparo de uma transição. A existência de uma marca no lugar inicial da rede permite o disparo da tranisção t0. Quando o disparo acontece, a marca existente no lugar inicial desaparece e surgem marcas nos três lugares de saída da transição.

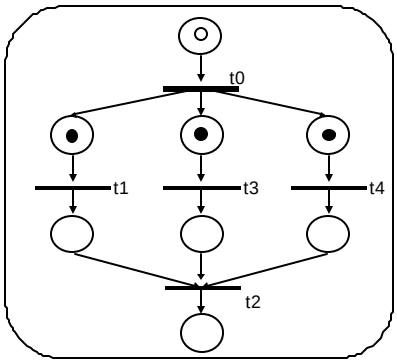

Figura 31: Disparo de uma transição

4.3 REPRESENTAÇÃO DE OCCAM EM REDES DE PETRI

Nesta seção é descrita a representação de uma descrição occam em Redes de Petri [38]. Embora o sistema particionado seja descrito como um programa em occam, o ambiente de co-síntese proposto utiliza como formato de entrada uma descrição em Redes de Petri do sistema digital. Também é mostrado o formato da representação de entrada da ferramenta de co-síntese. Esta representação pode ser dividida em duas

partes, a representação do fluxo de controle do programa e a do fluxo de dados. O fluxo de controle é representado pela estrutura da Rede de Petri, isto pela ligação de lugares e transições via arcos. O fluxo de dados está associado às transições da rede. A representação completa do programa em occam é feita pela combinação do fluxo de controle e do fluxo de dados.

Como visto na seção 3.2.2, um programa em occam é constituído de processos primitivos e combinadores de processos. Nas seções seguintes será descrita a representação em Redes de Petri dos processos primitivos e depois dos combinadores de processos.

4.3.1 Fluxo de Dados

O fluxo de dados representa transformações e a transferência de dados dentro do programa. No ambiente de co-síntese o fluxo de dados está associado às transições da Rede de Petri, permitindo uma associação com o fluxo de controle. A Figura 32 mostra um exemplo de fluxo de dados representando uma operação utilizando as variáveis A, B e X, que é associada à transição t0. Quando um *token* passar do lugar p0 para o lugar p1, a operação associada à transição t0, e representada pelo fluxo de dados, é executada.

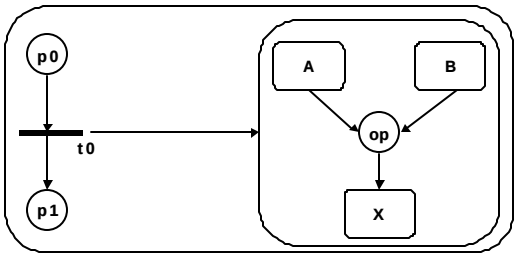

Figura 32: Fluxo de dados

O fluxo de dados associado às transições pode representar uma expressão, uma atribuição ou uma comunicação, como será mostrado nas próximas seções.

4.3.1.1 Expressão

Na Figura 33 é mostrado o fluxo de dados de uma expressão. As linhas tracejadas indicam que o elemento ligado a elas é opcional. Uma expressão é definida como sendo

uma das opções: constante, variável, operador unário seguido de expressão ou operador binário ligado a duas expressões.

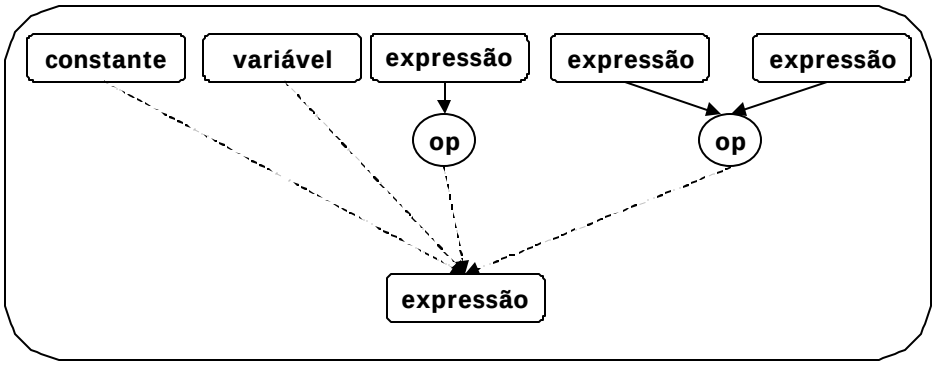

Figura 33: Fluxo de dados de uma expressão

4.3.1.2 Atribuição

A atribuição consiste na associação do valor de uma expressão a uma variável, como é mostrado na Figura 34.

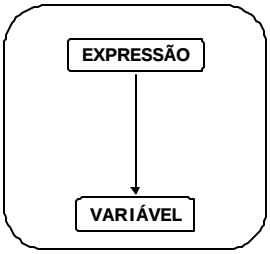

Figura 34: Fluxo de dados de atribuição

Atribuições associam valores de expressões a variáveis dentro de um processo. Não é possível fazer uma atribuição de uma expressão pertencente a um processo a uma variável pertencente a outro processo concorrente.

4.3.1.3 Comunicação

O fluxo de dados de uma comunicação é composto de dois elementos: envio e recebimento. No envio (Figura 35a) o valor da expressão é atribuído ao canal de comunicação, enquanto na operação de recebimento o valor que está associado ao canal é atribuído à variável. Pode-se observar que a junção da operação de envio com a

operação de recebimento equivale a uma operação de atribuição. A diferença reside no fato de que uma operação de atribuição é feita entre processos concorrentes. Tal fato será melhor explicado em seção subseqüente.

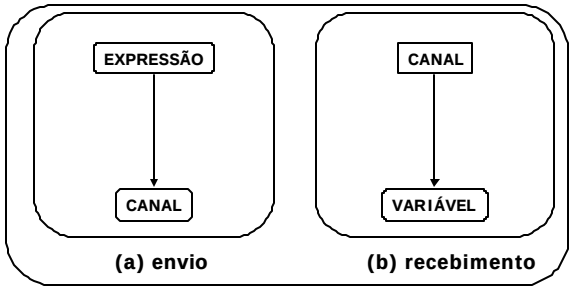

Figura 35: Fluxo de dados da comunicação

4.3.2 Representação de Processos Primitivos

Nas subseções anteriores foi vista a representação do fluxo de dados com suas três variações. Nesta seção e nas seguintes será abordada a descrição do fluxo de controle como Rede de Petri. Como visto anteriormente os processos primitivos são: atribuição, recebimento, envio, SKIP e STOP. Estes processos modelam uma ação, ação nula ou *deadlock.* Na Figura 16a são mostrados os processos primitivos em occam. Um processo primitivo pode ser uma operação de atribuição, uma operação de comunicação de entrada, uma operação de comunicação de saída, uma operação de SKIP que não realiza nenhuma ação e uma operação de STOP que representa *deadlock*.

4.3.2.1 Atribuição

A atribuição consiste na associação do valor de uma expressão a uma variável. A sub-rede que representa a atribuição é mostrada na Figura 36. A operação de atribuição está associada à transição t0. Quando a marca passa de p0 para p1 a operação de atribuição associada a t0 é executada.

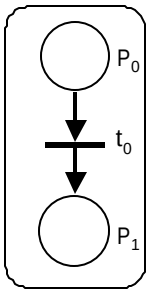

Figura 36: Atribuição em Redes de Petri

Uma transição que realiza operação de atribuição é rotulada como transição lógico/aritmética.

4.3.2.2 Comunicação

A comunicação é um dos elementos principais da linguagem occam. Esta permite que valores sejam trocados entre processos concorrentes. Como visto na seção 3.2.4 processos em occam se comunicam via canais de comunicação. Transições que contém comunicação associada ao fluxo de dados são rotuladas como transições do tipo comunicação. Na Figura 37a tem-se o exemplo de um trecho de programa em occam onde aparecem os trechos de código “CH ! 10” e “CH ? X” indicando o envio do valor 10 pelo canal CH e o recebimento deste valor pela variável X através do mesmo canal. A operação de comunicação é representada pela sub-rede da Figura 37b onde deve existir um *token* nos lugares p0 e p2 para que a comunicação se concretize, disparando a transição t0. Esta configuração de rede garante a comunicação síncrona pois se existir apenas um *token* em p0 ou apenas em p2 a transição t0 não é disparada. Na Figura 37c tem-se o fluxo de dados associado à transição t0. O valor 10 é associado ao canal CH e o valor do canal CH é transferido para a variável X.

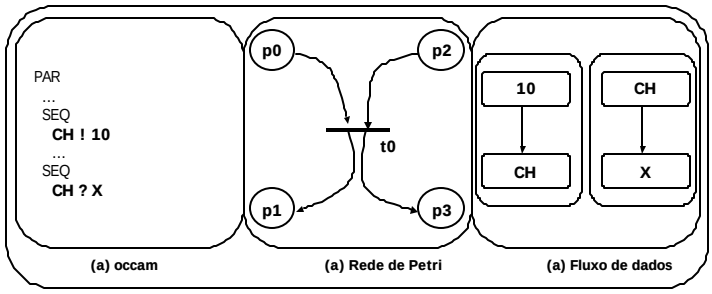

Figura 37: Rede de Petri da comunicação

4.3.2.3 SKIP e STOP

Os dois últimos processos primitivos em occam são o processo de SKIP e STOP. O processo SKIP não realiza nenhuma ação observável e termina. Isto é representado por uma transição sem ação associada como mostrado na Figura 38a. Esta transição entre p0 e p1 não realiza nenhuma ação e permite o disparo subseqüente da transição ti. O processo de STOP, ao contrário, não realiza nenhuma ação e nunca termina. É um exemplo de *deadlock* como visto na Figura 38b. O disparo da transição ts não permite o disparo de mais nenhuma transição, uma vez que não há transição posterior ao lugar p1.

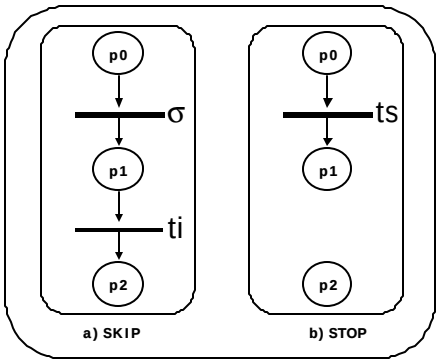

Figura 38: Processos de SKIP e STOP

4.3.3 Representação de Combinadores de Processos

Programas desenvolvidos em occam são construídos pela combinação de processos primitivos. Nesta seção são apresentadas as redes que modelam os combinadores de processos em occam.

4.3.3.1 Seqüência

Processos combinados como seqüência são executados um após o outro. A Figura 39a mostra um trecho de programa em occam onde os processos primitivos "X := A + 10" e "Y := 5" são combinados pelo combinador seqüencial SEQ. A rede que representa este trecho de código é mostrada na Figura 39b, onde o lugar p0 e a transição t0 estão associados ao primeiro processo e o lugar p1 juntamente com a transição t1 estão associados ao segundo processo primitivo.

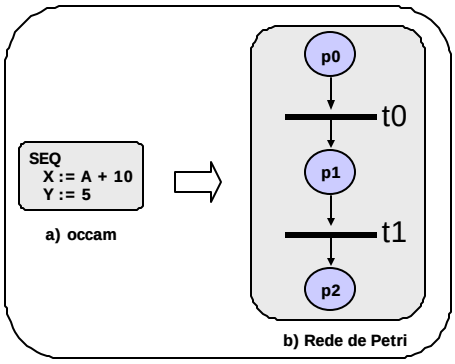

Figura 39: Combinador seqüencial

4.3.3.2 Condicional

O combinador condicional combina processos guardados por expressões booleanas. A rede que representa o combinador condicional é mostrada no exemplo da Figura 40. Quando existe uma marca no lugar p0 apenas uma das transições t0, t2 ou t4 pode ser disparada quando sua condição de guarda, representada pela expressão entre parênteses, é avaliada como verdadeira. Devido à construção da Rede de Petri as expressões são avaliadas da direita pra a esquerda. Desta maneira a transição t4 que contém uma condição que é sempre verdadeira (TRUE) só é disparada se nenhuma das condições relacionadas ás transições t0, t2 forem verdadeiras. O trecho de programa em occam (Figura 40a) mostra um combinador condicional IF onde existem três expressões de guarda: x < y, x = y e TRUE. Após cada expressão de guarda um processo primitivo é executado. Na representação em Rede de Petri da Figura 40b tem-se três transições posteriores ao lugar p0, este lugar representa o combinador condicional. Para cada transição t0, t2 e t4 é associado um fluxo de dados de uma expressão booleana

representando cada uma das condições de guarda. As transições t0, t2 e t4 são transições de início de condicional.

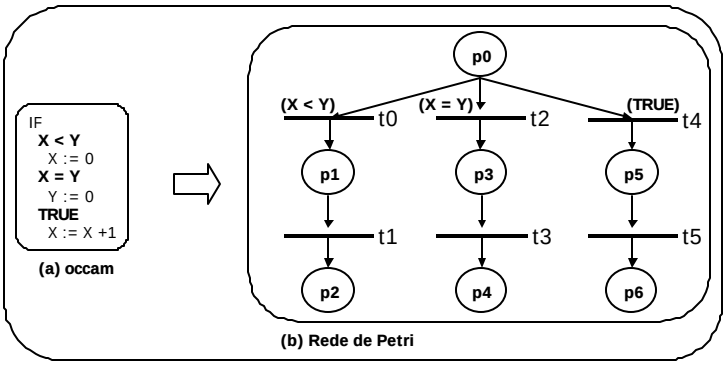

Figura 40: Combinador condicional

4.3.3.3 Laço

O combinador de laço tem uma representação em Rede de Petri semelhante ao combinador condicional. Existem duas diferenças básicas, a primeira é que a representação do combinador de laço possui apenas duas transições contendo expressões booleanas de guarda. A primeira transição está associada à condição de guarda do laço e a segunda transição à condição complementar à condição de guarda do laço. A segunda diferença reside no fato de existir uma transição para o início do laço, como visto na Figura 41.

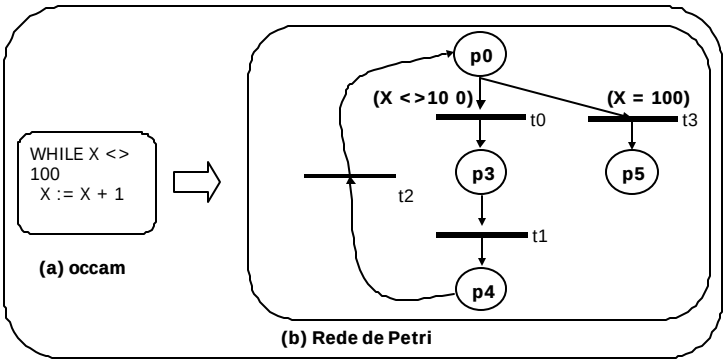

Figura 41: Combinador de laço

No programa em occam (Figura 41a) temos o processo “X := X + 1” que deve ser

executado enquanto a condição de guarda “X <> 100” for satisfeita. Este combinador é representado pela rede da Figura 41b a transição t0 está associada à expressão de guarda “X <> 100” e a transição t2, está associada à condição contrária. A transição t1 está associada à operação de atribuição “X := X + 1”. Finalmente a transição t2 não realiza nenhuma operação e retorna para o início do laço representado pelo lugar p0. As transições t0 e t3 são transições de início de laço e a transição t2 é uma transição de final de laço.

4.3.3.4 Paralelismo

O combinador paralelo permite descrever processos concorrentes em occam. É importante realçar que o término da execução do combinador paralelo só ocorre quando todos os processos disparados são finalizados. A rede que representa o combinador paralelo possui uma transição com vários lugares posteriores, tantos quanto forem os processos concorrentes. Cada um destes lugares representa o início de um processo concorrente. Todas as sub-redes que representam os processos concorrentes devem finalizar em uma transição comum que indica o fim da representação do combinador de paralelismo em Rede de Petri. No exemplo da Figura 42 a descrição em occam descreve a execução de três processos primitivos em paralelo. O combinador inicia no lugar p0 que possui a transição posterior t0. Quando a transição t0 é disparada é colocado um marca em cada um dos lugares p1, p3 e p5. Isto possibilita o disparo das três transições t1, t3 e t4 que estão respectivamente associados com os fluxo de dados das operações de atribuição mostradas na descrição em occam da Figura 42a. A transição t0 é uma transição de início de paralelismo, enquanto a transição t2 é uma transição de final de paralelismo.

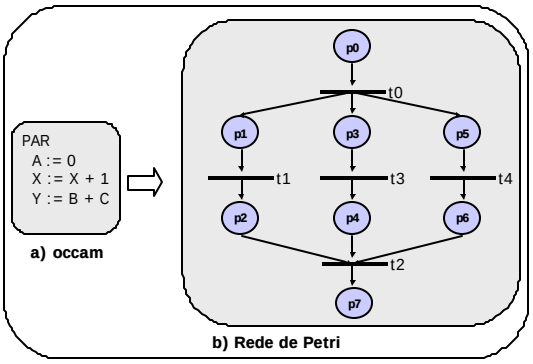

Figura 42: Combinador paralelo

4.4 GERAÇÃO DE OBJETOS

A razão principal para a escolha de Redes de Petri e não occam como base para geração de código sintetizável tem por objetivo permitir a utilização de um formato interno independente da linguagem de especificação utilizada. Assim, torna-se mais fácil a alteração da linguagem de especificação. A primeira etapa deste trabalho foi a criação de uma ferramenta capaz de gerar uma representação orientada a objetos da Rede de Petri, a qual foi obtida a partir da descrição em occam do sistema particionado. Rede de Petri, lugar e transição foram modelados como objetos. Também com o objetivo de se obter uma ferramenta independente da linguagem de especificação são gerados tipos independentes. Vale ressaltar que neste trabalho as Rede de Petri não são utilizadas para análise, mas sim como modelo interno de representação do sistema digital. Desta maneira, a análise de propriedades não é realizada neste trabalho.

4.4.1 Geração de Objetos a partir da Rede de Petri

O sistema PISH gera, a partir de uma descrição em occam, arquivos texto representando a Rede de Petri. Este formato não é muito útil, porque não é de fácil manipulação. Para solucionar este problema foi primeiro desenvolvido um modelo orientado a objetos da Rede de Petri, assim pode-se realizar alterações nesta rede de maneira mais fácil.

O objeto lugar consiste em um conjunto de transições posteriores e anteriores. Este também possui um identificador único e pertence a uma única sub-rede. Uma sub-

rede consiste em um conjunto de lugares e transições que são disparadas em seqüência.

A transição possui um conjunto de lugares de entrada e de saída. Os lugares anteriores identificam o(s) processo(s) aos quais a transição pertence. Uma característica importante da transição é seu tipo. Os tipos de transição são: lógico/aritmética, comunicação, parada, transição, *fork*, *join*, início de laço, fim de laço, início de condicional, fim de condicional, parada e transição de continuação. O tipo de transição será importante para a fase de extração de processos concorrente e geração de código.

O objeto transição também possui um fluxo de dados associado. Este fluxo de dados pode incluir: conjunto de operações lógico/aritméticas, condicionais, ações de comunicação e também é representado como objeto. Este fluxo de dados é associado à transição com o objetivo de se ter uma representação completa do sistema digital, uma vez que a Rede de Petri gerada inicialmente representa apenas o fluxo de controle. Na etapa de extração de processos o fluxo de dados é associado às transições.

Tipo	Descrição da Transição
AL	Transição lógico/aritmética
COM	Transição de comunicação
B_PAR	Transição de *fork*
E_PAR	Transição de *join*
B_LOOP	Transição de início de laço
D_LOOP	Transição de final de laço
B_IF	Transição de início de condicional
D_IF	Transição de final de condicional
SKIP	Transição de continuação
STOP	Transição de parada

Tabela 2: Tipos de transição

4.4.2 Geração de Tipos

No caso deste sistema que utiliza três linguagens distintas no projeto de um sistema digital, uma para especificação, uma para implementação em *hardware* e uma para implementação em *software*, deve existir uma preocupação com a compatibilidade

entre os tipos utilizados nas três linguagens. Assim como foi feito com o fluxo de dados e com o fluxo de controle, há necessidade de se derivar tipos para a implementação em *hardware* e *software* não diretamente dos tipos da linguagem de especificação, garantindo assim a independência da ferramenta. Neste trabalho são definidos tipos intermediários ou tipos do sistema. Como mostrado na Figura 43, os tipos do sistema são derivados dos tipos da linguagem de especificação e os tipos de *hardware* e *software* são derivados a partir destes. Assim, garante-se compatibilidade entre os tipos em *hardware* e em *software* e a independência da linguagem de especificação. A compatibilidade entre os tipos de *hardware* e *software* deve ser garantida com relação ao tamanho dos tipos e da sua implementação, por exemplo *little endian* ou *big endian*. Deve-se notar que estes fatores, tamanho e implementação, são definidos pela arquitetura, especificamente pelo processador utilizado. Em uma implementação em *hardware* estes parâmetros podem ser alterados ao contrário de uma implementação em *software* onde estas características são determinadas pelos parâmetros do processador utilizado.

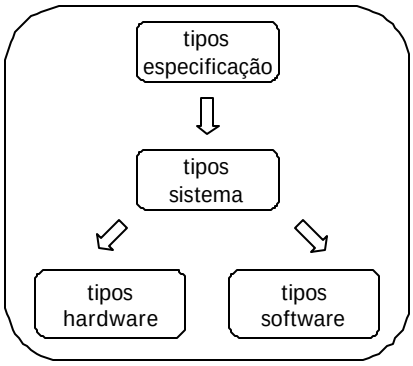

Figura 43: Tipos do sistema

Exemplos de tipos de sistema podem ser vistos na Tabela 3 e podem ser divididos em tipos básicos ou tipos compostos. Tipos básicos são aqueles que não derivam de nenhum outro tipo e compostos são derivados de tipos já existentes. São utilizados dois tipos compostos: ARRAY e COMPOSIÇÃO. A diferença básica entre os dois é que o tipo ARRAY é formado por vários elementos do mesmo tipo, ao contrário do tipo COMPOSIÇÃO que pode ser composto de elementos de tipos diferentes.

Tipo	Descrição
INT	Valor com sinal com 8 bits
BYTE	Valor sem sinal de tamanho 8 bits
INT16	Valor com sinal 16 bits
BOOL	Tipo booleano
COMPOSIÇÃO	Composição de tipos básicos e/ou compostos
ARRAY	*Array* composto de um dos tipos acima

Tabela 3: Tipos de sistema

Em *hardware* são utilizados além dos tipos de sistema o vetor de bits. Assim são necessárias funções de conversão de tipos de sistema para vetor de bits e de vetor de bits para tipos do sistema.

Um segundo caso particular relacionado aos tipos em *hardware* é o caso em que os tipos em *hardware* e *software* possuem tamanhos naturalmente diferentes. Este é um caso típico do tipo booleano que em *software* possui, normalmente, o tamanho da palavra do processador. No caso do *hardware* este tipo é normalmente implementado como um único bit. Desta forma, em implementações de *hardware* o custo em área seria bem maior caso o tipo booleano em *hardware* possuísse o mesmo tamanho que seu equivalente em *software*. Assim, faz-se necessário a conversão do tamanho do tipo quando se envia ou recebe tipos de tamanhos diferentes do *hardware* para o *software* e vice-versa. As funções de conversão de tipos em *hardware* são mostradas na Figura 44.

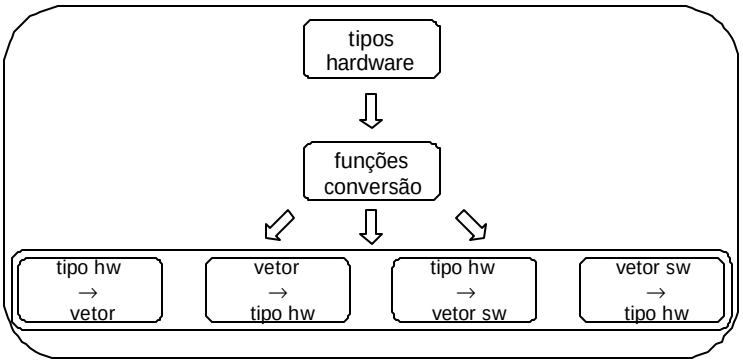

Figura 44: Tipos em *hardware*

4.5 EXTRAÇÃO DE PROCESSOS

Nesta seção é definido o modelo de processos usado na etapa de co-síntese e representado em Rede de Petri. Neste modelo, um processo é capaz de executar apenas tarefas seqüenciais. A concorrência na rede é expressa pelos conceitos de processo pai e filho, onde um processo pai dispara processos filhos que trabalham concorrentemente. Em seguida é vista a extração dos processos de uma Rede de Petri, como é feita a associação do fluxo de dados aos processos. Além do fluxo de dados também são associadas aos processos as declarações de variáveis utilizadas por cada processo.

4.5.1 Transição de Substituição e Página

É uma super transição que representa uma Rede de Petri. A transição de substituição permite representar hierarquia em uma Rede de Petri [26]. A Figura 45a mostra uma Rede de Petri sem hierarquia. Esta rede pode ser representada hierarquicamente como se vê na Figura 45b onde a sub-rede formada pelos lugares {p2, p3, p4, p5, p6, p7} e transições {t2, t4 e t5} é substituída pela transição de substituição TS. A transição de substituição tem o lugar p1 como lugar de entrada e p9 como lugar de saída.

A rede representada pela transição de substituição é chamada de página. A página da rede representada por TS é mostrada na Figura 45c. Note que na página existem os lugares p1 e p9 que são chamados de *port places* que representam os lugares de entrada e saída da página.

Uma página possui lugares especiais chamados *socket places* que servem para que a mesma se conecte à Rede de Petri que a contém. Os *socket places* da página e os *port places* da transição de substituição estão relacionados, de tal maneira que se existe uma marca no *port place* também existe um *marca* no *socket place* correspondente.

No contexto deste trabalho, transição pai é uma transição de substituição cujo *socket place* de entrada possui como transição de saída uma transição do tipo início de paralelismo e como lugar final um *socket place* que possui como transições de entrada transições do tipo final de paralelismo. Na Figura 45b a transição TS é um exemplo de transição pai, pois a transição de saída do *port place* p1 é do tipo início de paralelismo e a transição de entrada do *port place* p9 é do tipo final de paralelismo.

Uma operação de inserção de transição pai ocorre quando se extrai as páginas da rede original que podem ser substituídas por transições pai. Note que a sub-rede da página repete os lugares p1 e p8 (*port places*) e engloba todos os lugares e transições entre estes lugares. A Figura 45b mostra o resultado de uma operação de inserção de transição pai onde é extraída a página (Figura 45c).

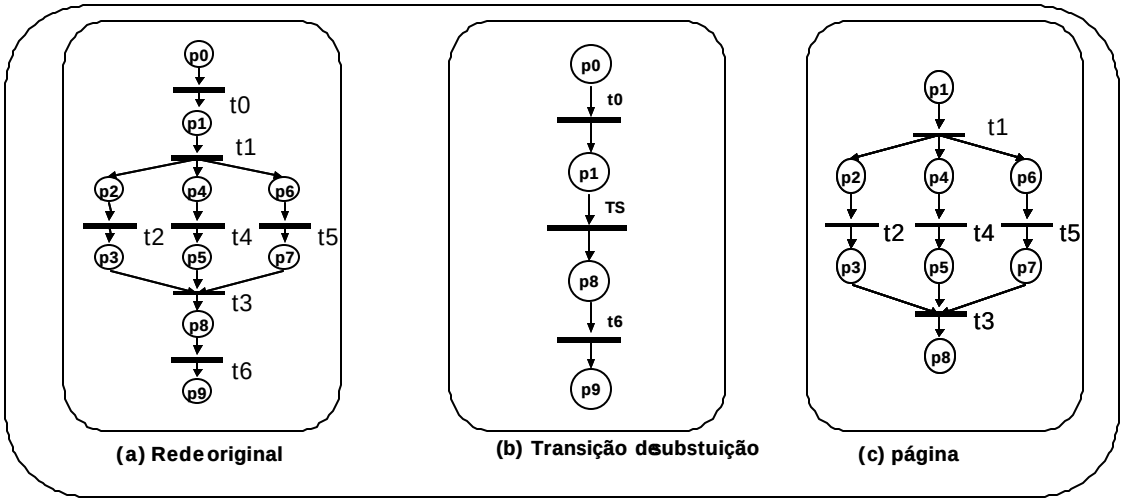

Figura 45: Transição de substituição

4.5.2 Definição de Processo

No contexto deste trabalho a definição de processo será diferente da definição de processo em occam. Um processo é definido como uma Rede de Petri que possui as

algumas características especiais. A primeira característica é que seu lugar inicial não possui transição de entrada ou possui como transição de entrada uma transição do tipo início de paralelismo. A segunda característica de um processo é que seu lugar final não possui transição de saída ou possui uma transição do tipo final de paralelismo. A próxima característica do processo é que este não pode conter lugares que possuam transições de saída tipo início de paralelismo e transições de entrada do tipo final de paralelismo. A última característica de um processo é que o mesmo pode conter transições pai. Na Figura 46 a sub-rede composta de todos os lugares não pode ser considerada um processo porque o lugar p0 possui uma transição de saída do tipo início de paralelismo. Além disso, o lugar p19 possui como transição de entrada uma transição do tipo final de paralelismo. As sub-redes que possuem os lugares p1, p7, p13 e p20 e lugares finais p6, p12, p18 e p21 respectivamente são processos. Estas têm lugares iniciais que possuem como ponto de entrada uma transição do tipo início de paralelismo e lugares finais que possuem como ponto de entrada uma transição do tipo final de paralelismo. Vale ressaltar que o último processo que tem como lugar inicial p20 possui a transição pai TS.

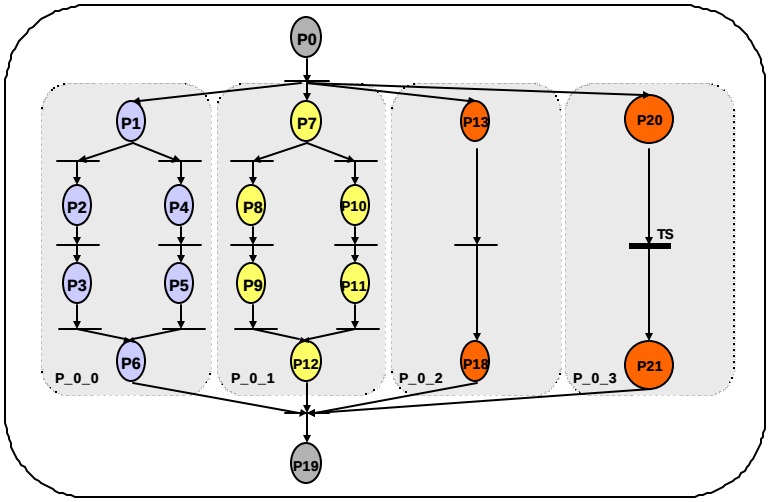

Figura 46: Processos

Um processo é considerado filho quando está contido em uma das páginas associadas a transições pai de um outro processo. Este último é considerado processo pai. É importante ressaltar que uma página pode conter vários processos (Figura 46) isto implica que um processo pai pode ter vários processos filhos. Já uma transição pai

pertence a apenas um processo, isto implica que um processo filho possui apenas um processo pai.

4.5.3 Inserção de Fluxo de Dados

O fluxo de dados é representado também como objeto, existindo basicamente três tipos de objetos sendo definidos: atribuição, expressão e comunicação. O fluxo de dados é extraído da especificação original e associado à transição correspondente. Assim como o controle, o fluxo de dados também fica independente da especificação original em occam. Conforme descrito na seção 4.3.1 uma atribuição consiste em associar o valor de uma expressão a uma variável. Uma expressão pode ser uma expressão lógica ou aritmética e uma comunicação pode ser o envio de uma expressão ou o recebimento de um valor a ser associado a uma variável.

Exemplos de objetos representando fluxos de dados associados às transições na Rede de Petri podem ser vistos na Figura 47. Temos duas expressões booleanas associadas às transições posteriores ao lugar P1 (A = 0 e true), uma transição aritmética (x ← 10) associada à transição posterior ao lugar P2 e duas comunicações associadas à transição posterior aos lugares P4 e P7. É importante ressaltar que transições de comunicação sempre possuem uma comunicação de envio e uma comunicação de recebimento. Este tipo de transição é posterior e anterior a lugares que pertencem a processos diferentes.

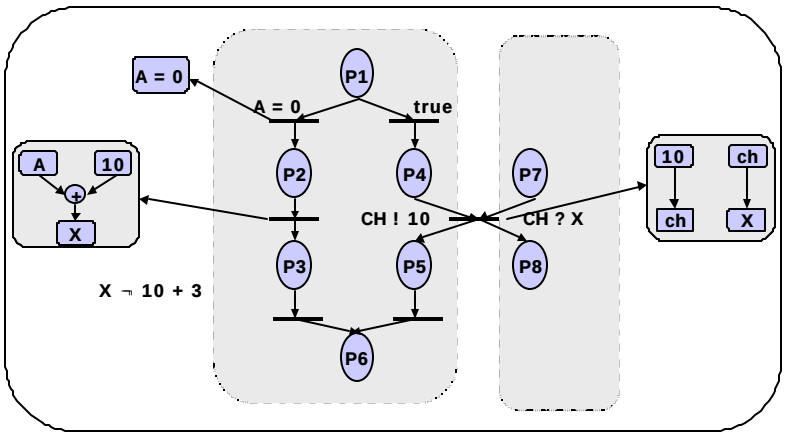

Figura 47: Fluxo de dados na Rede de Petri

Na seção seguinte será visto com mais detalhes o modelo de processo e como o mesmo pode ser representado na máquina de estados finitos.

4.5.4 Associação de Declarações aos Processos

Linguagens de especificação podem definir vários escopos para suas variáveis, incluindo o escopo global. No ambiente de *Co-design* proposto, cada processo trabalha com seu conjunto de variáveis localmente. Desta maneira deve-se identificar o escopo das declarações de variáveis da especificação do sistema particionado e associá-las aos respectivos processos. A Figura 48 mostra a associação de declarações de variáveis na especificação inicial aos processos seqüenciais. Deve-se observar que uma mesma declaração de variável pode estar associada a mais de um processo. Como exemplo tem-se a primeira declaração de variável da Figura 48 (INT A, B:), a qual está associada a todos os processos. Também pode acontecer de uma declaração estar associada a um único processo, como é mostrado no segundo caso em destaque da mesma figura (INT C:).

No primeiro caso o escopo da declaração inicial abrange vários processos. Neste caso são definidas declarações locais da mesma variável para cada processo. Podem ocorrer dois casos de utilização de uma mesma variável simultaneamente: utilização por processo pai e processo filho e utilização simultânea por vários processos filhos. O primeiro caso é explicado mais detalhadamente na seção 4.7.2.

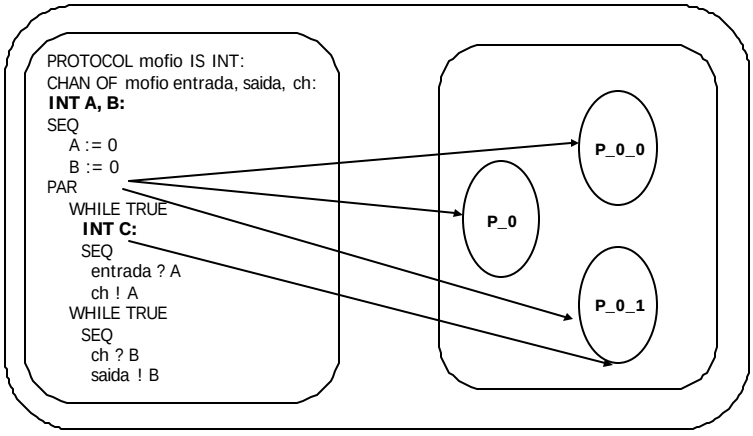

Figura 48: Associação de declarações a processos

Um dos atributos de um processo é sua natureza: *hardware* ou *software*. Embora o PISH já retorne o sistema particionado, o ambiente de co-síntese permite que se manipule a natureza dos processos. Alterando-se este atributo é possível modificar o particionamento original feito pelo PISH. Esta característica é interessante pois permite ao projetista do sistema utilizar sua experiência para realizar alterações manuais no particionamento do sistema digital.

4.6 INSERÇÃO DE E/S

Mecanismos de entrada e saída possibilitam ao sistema digital o recebimento e envio de dados de/para o mundo exterior. O acesso ao mundo exterior é feito via dispositivos de entrada/saída ou E/S. Estes dispositivos de E/S utilizam protocolos dedicados que devem ser manipulados pelo sistema digital. Embora possuam protocolos dedicados, os consumidores de dados fornecidos por dispositivos de entrada são os processos concorrentes que compõem o sistema digital. Da mesma maneira, os dados fornecidos aos dispositivos de saída também são produzidos pelos processos concorrentes. Como visto antes, processos concorrentes utilizam o mecanismo de comunicação por canal quando a comunicação não é realizada de processo pai para processo filho. Cada processo recebe ou envia, por canal, tipos específicos de dados quando realiza uma comunicação.

O modelo de processos visto na seção 4.5 não suporta o controle de dispositivos de entrada/saída, apenas o controle de canais de comunicação. Desta maneira é introduzido o modelo de processo de E/S. Este modelo pode ser visto na Figura 49, onde é mostrado um exemplo de processo E/S de saída, e na Figura 50, de um processo de entrada. O processo de E/S é responsável pelo controle do dispositivo de entrada e saída. Este é capaz de acionar de maneira correta os sinais de controle do dispositivo, enviando ou recebendo dados do dispositivo de E/S. Assim como os processos convencionais, também são capazes de enviar ou receber dados via canais de comunicação permitindo assim que um processo do sistema digital se comunique com dispositivos de E/S de forma transparente. Cada dispositivo de entrada/saída possui um protocolo de controle diferente dos demais. Isto implica que uma característica particular do processo de E/S é que deve existir um processo diferente para cada dispositivo de

E/S existente na arquitetura alvo do sistema digital. Um processo de E/S capaz de controlar um *display* de 7 segmentos não pode ser utilizado para controle de um modem por exemplo. Novos processos de E/S devem ser construídos e adicionados às arquiteturas alvo já existentes para o sistema digital.

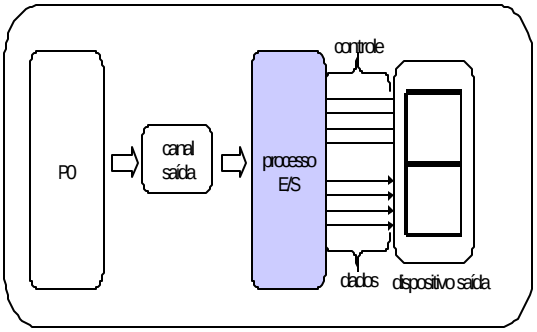

Figura 49: Processo de saída

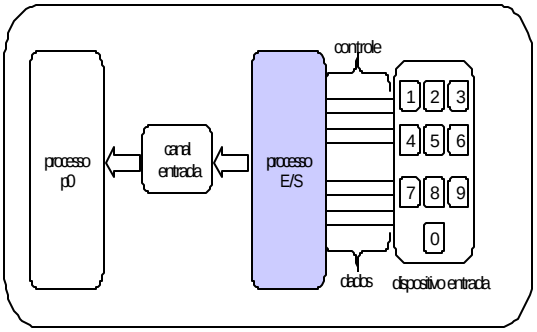

Figura 50: Processo de entrada

A Figura 51 mostra, utilizando como exemplo um processo de entrada, uma outra característica dos processos de E/S. Estes se comunicam com apenas um processo concorrente utilizando um único canal de comunicação. Ao contrário dos processos do sistema que podem se comunicar com vários processos utilizando-se diferentes canais de comunicação. Um processo de E/S é composto de três partes: controle de dispositivo, composição de tipos e controle de canal. O controle de dispositivos é responsável pelo acionamento dos sinais de controle do dispositivo de E/S e pela leitura/escrita de uma palavra de dado do dispositivo. A palavra de dado do dispositivo não deve ser confundida com a palavra da arquitetura vista anteriormente. Cada dispositivo em particular possui seu próprio tipo de palavra de dado com tamanho específico. Esta parte é responsável pela comunicação de baixo nível com o dispositivo de entrada/saída. A segunda parte, composição de tipos, é responsável pela composição dos tipos de dados a serem transferidos pelo canal de comunicação. O canal de entrada da Figura 51 é capaz de realizar a transferência do protocolo de dados composto de tipo 0, tipo 1, tipo n, sendo estes tipos aceitos pelo bloco de composição de tipos do processo E/S. Finalmente o bloco de controle de canal deve controlar o canal de entrada/saída ligado ao processo de E/S. Este controle é feito de forma igual àquela realizada pelos processos concorrentes do sistema digital.

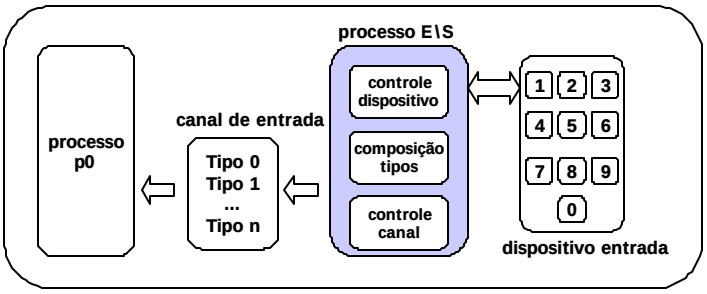

Figura 51: Processo E/S

A última característica dos processos de E/S é a sua natureza. Um processo de E/S pode ser implementado tanto em *hardware* quanto em *software*. Isto significa que um dispositivo de saída pode ser controlado tanto pelo processador quanto pelos componentes de *hardware*. A existência de processos E/S de natureza distinta para controle de um mesmo dispositivo dá ao projetista do sistema digital flexibilidade para escolher a melhor alternativa para seu projeto. Deve-se destacar que o mecanismo de processo de E/S separa de maneira clara a aplicação, composta dos processos concorrentes do sistema, do acesso aos dispositivos de entrada e saída. Este fato, juntamente com a utilização da interface *hardware/software* permite várias combinações diferentes de processos concorrentes e processos de E/S. Por exemplo, um sistema digital pode ser composto de todos os seus processos concorrentes em *software* enquanto seus processos de E/S são implementados em *hardware*. Esta configuração é bastante comum uma vez que os processadores possuem recursos limitados de E/S, sendo necessária a utilização de componentes de *hardware* para acesso a dispositivos de entrada/saída.

4.7 GERAÇÃO DE CÓDIGO

Nesta seção são apresentados os modelos utilizados para a geração de código do sistema digital. É visto o modelo do sistema digital como um todo e depois seus componentes. O primeiro de seus componentes é o modelo de processo baseado em máquinas de estado finito. Máquinas de estado finito são facilmente implementadas tanto em linguagens de *software* quanto em linguagens de descrição de *hardware*. Também é descrito como são representadas máquinas de estado finito tanto em *hardware* quanto em *software.* No caso específico de geração de código para *hardware* foi criada uma

extensão da linguagem VHDL, denominada VHDL*, que permite a implementação de máquinas de estado finito que podem se comunicar via construtores de comunicação síncrona. Conjuntamente foi desenvolvida uma estratégia para geração de código VHDL padrão a partir de VHDL*. Em seguida a comunicação entre processos é descrita pelo modelo de comunicação. Em seqüência o modelo de interface *hardware/software* é apresentado, cuja finalidade é permitir que processos em *hardware* se comuniquem com processos em *software* de forma transparente. Finalmente o mecanismo de entrada e saída responsável pela comunicação do sistema com o mundo exterior será discutido.

4.7.1 Processos como FSM's

Como visto anteriormente o sistema digital é modelado como um conjunto de processos concorrentes, sendo cada processo capaz de realizar tarefas em seqüência. Outra característica necessária aos processos é que os mesmos devem ser capazes de ativar outros processos de forma concorrente. Tal característica permite que embora um processo possa apenas realizar tarefas em seqüência, vários processos trabalhando em paralelo fazem com que o sistema digital como um todo realize tarefas concorrentes. Além de ativar outros processos, um processo deve ser capaz de informar àquele que o ativou que terminou de executar suas tarefas. Deve-se ressaltar que a comunicação utilizada pelo sistema PISH é baseada em CSP sendo, portanto, síncrona. Isto implica que processos devem ser capazes de realizar este tipo de comunicação.

A forma adotada para a representação de processos é a máquina de estados finitos (*FSM*)[4]. Uma máquina de estados finitos **[referência]** é composta por um conjunto de estados e seu funcionamento é ditado por mudanças entre transições de estados. Estas mudanças de estado representam as ações realizadas dentro de um determinado estado e podem ou não ser guardadas por condições. Tarefas concorrentes não podem ser representadas por uma *FSM* simples, apenas tarefas seqüenciais, uma vez que a seqüência de ações é determinada pela seqüência de estados percorrida. A Figura 52 ilustra uma seqüência de estados com as ações executadas pela máquina de estados finitos.

[4] *Finite State Machine*

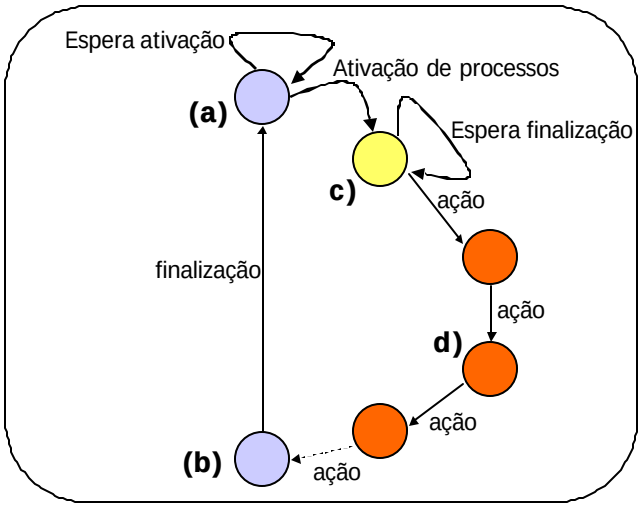

Figura 52: Modelo de processo

No modelo de processo adotado a máquina de estados possui algumas características especiais. A primeira delas é a existência de estados de inicialização (Figura 52a) e finalização (Figura 52b). Quando o processo está no estado de inicialização este espera até que o seu sinal de ativação seja disparado por um processo pai. Quando no estado de finalização o processo indica a seu processo pai, através do sinal de finalização, que terminou sua execução.

Outro tipo de estado especial do processo pode ser visto na Figura 52c. Este estado (ativação de processo) permite que este processo dispare processos filhos concorrentes. Não deve haver mudança de estado enquanto todos os filhos ativados não terminarem sua execução através de seus sinais de finalização, como visto no parágrafo anterior.

Finalmente os estados de ação são responsáveis pelas ações realizadas pelo processo. Ações podem ser classificadas em: operação lógica, operação aritmética, comunicação, decisão, ação nula e ação de parada. Uma ação nula não realiza nenhuma operação, sendo representada simplesmente por uma mudança de estado. Já uma ação de parada é representada pela não realização de operações e também por não haver mudança de estado. Isto implica que se um processo chega a um estado que contenha uma ação de parada o mesmo não executará mais nenhuma atividade, ficando preso ao estado que realiza a ação de parada. Uma ação de decisão permite que a seqüência de

estados seja alterada dependendo de condições. Estas ações podem ser separadas em ações de decisão condicional e de laço. As decisões condicionais possibilitam a escolha de uma entre várias seqüências de estados e pode ser vista como equivalente a um construtor *if then else.* A ação de decisão de laço permite que uma determinada seqüência de estados seja executada várias vezes. Isto é equivalente ao construtor *while*.

Como dito anteriormente para a geração de código dos processos em *hardware* foi desenvolvida a linguagem VHDL* contendo construtores de comunicação de comunicação síncrona. Estes construtores são de envio e recebimento de dados. Estes construtores permitem que máquinas de estado se comuniquem de maneira síncrona. Este comportamento é semelhante aos construtores de envio e recebimento de occam. Uma máquina de estados em VHDL* que possua um construtor de comunicação em um de seus estados, só sai deste estado quando a comunicação tiver sido completada. A estratégia utilizada para conversão de VHDL* em VHDL baseia-se na inserção de novos estados que representam as ações realizadas antes e depois da comunicação. Isto é visto na Figura 53 onde no estado s0 existe a comunicação C ! B em VHDL*. Estão destacadas as ações realizadas antes da comunicação (B := A * 20) e as ações realizadas após a comunicação (B := B + 1). No diagrama do lado direito da mesma figura estão os estados originais s0 e s1 e os novos estados adicionados c0 e c1. O estado s0 original é alterado. Neste passam a se realizar as ações anteriores à comunicação. O estado c0 é responsável pela realização da operação de comunicação propriamente dita e o estado c1 realiza as ações após a comunicação.

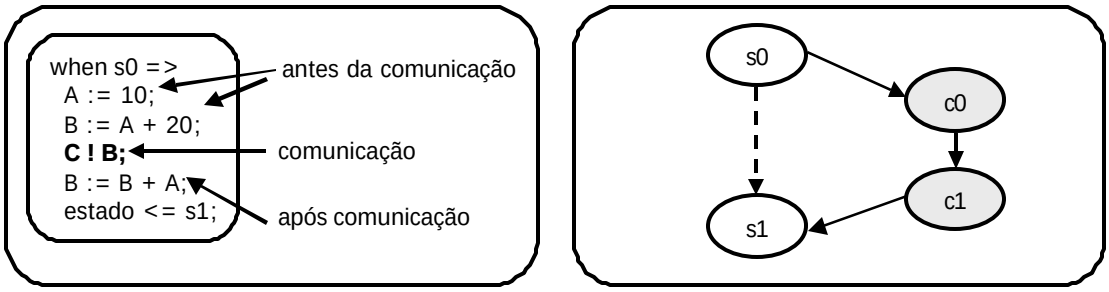

Figura 53: Tradução VHDL* para VHDL

Ferramentas foram desenvolvidas que transformam uma descrição VHDL* em VHDL padrão.

4.7.2 Modelos de Comunicação

Dois modelos de comunicação são utilizados para a implementação do sistema digital: comunicação direta e comunicação síncrona por canal. A comunicação direta é utilizada em duas ocasiões, quando da ativação por um processo pai de um processo filho e quando um processo filho informa a seu pai da sua finalização.

No primeiro caso, ativação do processo filho, o processo pai deve transferir para o processo filho os valores das variáveis compartilhadas entre os dois processos. Na Figura 54 um processo pai ativa um processo filho. O conjunto {Vp0, ..., Vpn} representa as variáveis utilizadas pelo processo pai e {Vf0,..., Vfm} o conjunto de variáveis utilizadas pelo processo filho. Caso alguma das variáveis pertencentes ao conjunto de variáveis do processo pai também pertença ao conjunto de variáveis do processo filho seu valor deve ser transferido para o processo via o bloco de ativação da Figura 54. O bloco de ativação também é responsável por levar o sinal de ativação do processo pai para o processo filho.

Função oposta possui o bloco de finalização mostrado na Figura 55. Este é responsável por levar o sinal de finalização do processo filho para o processo pai. Assim como, retornar os valores atualizados das variáveis compartilhadas entre os dois processos.

Este mecanismo de atualização de variáveis compartilhadas é necessário porque uma vez que o processo pai dispara processos filhos, este interrompe sua execução até que todos os processos filhos ativados tenham finalizado suas execuções. Desta maneira, um processo pai pode delegar a seus filhos operações sobre suas variáveis e receber depois os valores atualizados pelas operações realizadas. A segunda forma de comunicação utilizada por processos é a comunicação síncrona por canais. Este caso ocorre entre processos filhos onde não há possibilidade de compartilhamento de variáveis. Este modelo de comunicação implementa o comportamento de processos paralelos em occam que não podem compartilhar variáveis [20].

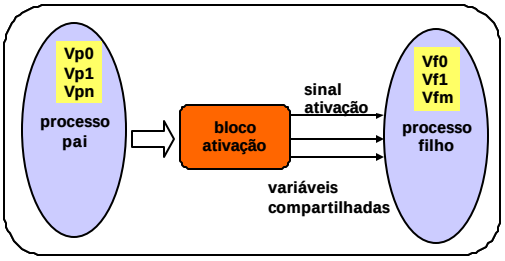

Figura 54: Ativação de processo

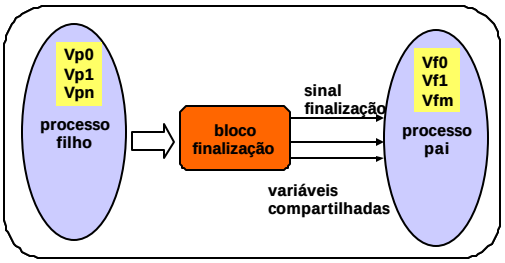

Figura 55: Finalização de processo

Na Figura 56 é mostrado o esquema da comunicação por canais entre dois processos p0 e p1 utilizando o canal c0. Neste modelo cada processo tem uma interface definida com o canal. Tanto o processo remetente (p0) quanto o processo destinatário (p1) devem ativar o canal para que a comunicação se efetue. Isto é feito via os respectivos sinais *ativa*. Devido ao fato da comunicação ser síncrona ambos os processos devem esperar até que os sinais de pronto sejam ativados indicando que a comunicação foi completada. No modelo proposto os canais de comunicação deixam os detalhes da transferência de dados transparentes para os processos que estão realizando a comunicação.

Os canais de comunicação são responsáveis pela transferência de um protocolo, que consiste em um conjunto fixo de tipos de dados. Por exemplo, se o canal c0 da Figura 56 possui um protocolo capaz de transferir um valor inteiro e um valor booleano entre os processos p0 e p1, ele sempre transferirá um valor inteiro e um valor booleano entre estes dois processos, não podendo transferir apenas um valor booleano ou um valor inteiro.

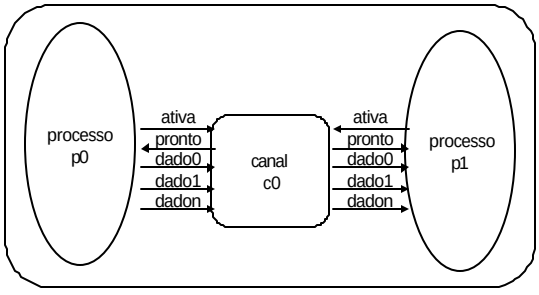

Figura 56: Comunicação por canais

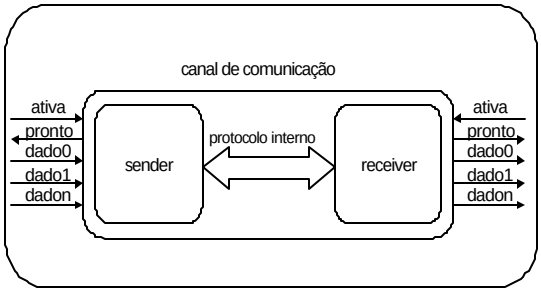

Figura 57: Canal de comunicação

O canal é composto de duas partes: *sender* e *receiver.* A parte *sender* realiza a

interface com o processo remetente enquanto a parte *receiver* realiza a interface com o processo destinatário. Este esquema pode ser visto na Figura 57 onde também é destacada a existência de um protocolo interno entre as duas partes componentes do canal. Esta abordagem permite a utilização de diversos componentes de *sender* e *receiver* que podem estar armazenados em uma biblioteca, o que garante mais flexibilidade.

4.7.3 Modelo de Interface

Os modelos de comunicação mostrados na seção anterior são independentes da natureza dos processos que estão se comunicando. Não importa se os processos estão ambos em *hardware*, *software* ou se um deles está em *hardware* e outro em *software*. A comunicação se realiza pela utilização de blocos de ativação, finalização ou canal, dependendo do tipo de comunicação realizada. Se ambos os processos estão em *hardware* o canal pode ser implementado como um componente em *hardware.* No segundo caso, processos em *software*, o canal pode ser implementado como funções e estrutura. Quando os processos são de natureza distinta é necessária a utilização de uma interface entre os componentes de *hardware* e *software* do sistema

O modelo de interface proposto é mostrado na Figura 58. Como pode ser verificado, a interface é simétrica com relação a sua parte em *hardware* e *software*. O modelo de interface adotado é baseado em camadas, sendo formado por três camadas *prcs_unit* (Figura 58a), *comm_unit* (Figura 58b) e *io_unit* (Figura 58c). A finalidade da subdivisão em camadas é permitir que novas arquiteturas alvo sejam acrescentadas à ferramenta da maneira mais simples possível.

No contexto deste trabalho palavra é a unidade básica de informação transferida entre os componentes de *hardware* e *software* do sistema digital. A camada mais externa da interface, *prcs_unit*, tem como função implementar as partes dos componentes de comunicação, canal, bloco de ativação e bloco de finalização, como componentes em *hardware* ou *software*. Esta camada é formada por elementos ligados diretamente aos processos em execução, aqui chamados de elementos *prcs*. Os elementos *prcs* possuem três funções: conversão de palavras em tipos de dados, conversão de tipos de dados em palavras e implementação do componente de comunicação correspondente em *hardware*

ou *software*. Um exemplo de como a *prcs_unit* está ligada aos processos é mostrado na Figura 59. Nesta figura são mostrados os quatro tipos de elementos que podem ser elementos *prcs.* O primeiro deles é o componente de *sender* que está ligado a um processo remetente. Em seguida vem o componente de *receiver*, ligado a um processo destinatário. Como visto anteriormente estes componentes são utilizados para a comunicação síncrona por canal. O terceiro tipo de componente é o bloco de ativação e por último o bloco de finalização. Vale ressaltar que os elementos de *prcs* estão sempre em pares, havendo um elemento na *prcs_unit* em *hardware* e seu equivalente na *prcs_unit* em *software*.

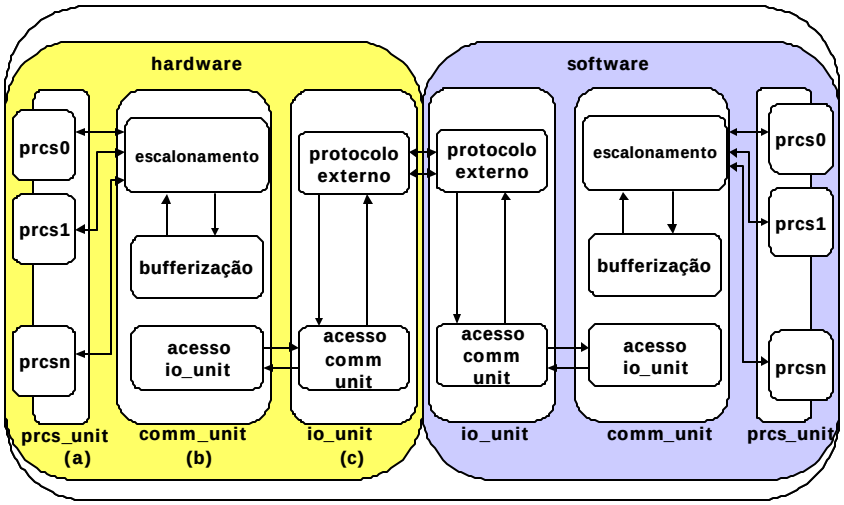

Figura 58: Interface *hardware/software*

Os tipos de dados transferidos utilizando-se a interface terão sempre tamanho múltiplo de uma palavra, sendo compostos de uma ou mais palavras. A função de conversão de tipos de dados em palavras é realizada pelos elementos *prcs* de *sender* e bloco de ativação. Estes elementos *prcs* devem receber palavras da camada intermediária da interface, a *comm_unit*, e convertê-las para os tipos de dados que estão sendo transferidos pelo canal ou pelos tipos de dados das variáveis compartilhadas pelos processos sendo enviadas pelo bloco de ativação. A função de conversão de tipos de dados em palavras é realizada pelos elementos de *receiver* e bloco de finalização. Estes elementos recebem os dados vindos dos processos e devem convertê-los em palavras para que possam ser enviados pela interface.

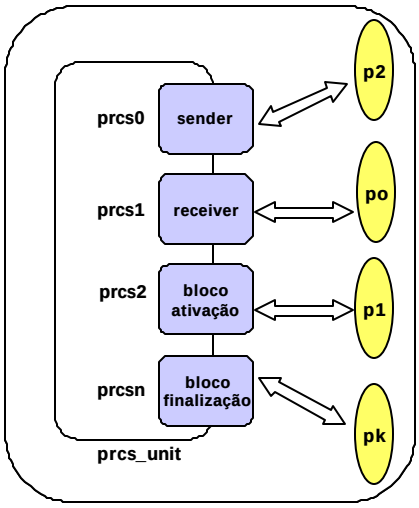

Figura 59: Camada *Prcs-unit*

A camada intermediária da interface, *comm_unit* (Figura 58b), serve como ponte de ligação entre a camada inferior, *io_unit*, e a *prcs_unit* vista anteriormente. Esta possui três funções: acesso à camada *io_unit*, bufferização e escalonamento. O acesso à camada *io_unit* é responsável pelo envio/recebimento de palavras para/de *io_unit*. Este tem como objetivo permitir que diferentes implementações da camada *io_unit* possam ser acessadas pela camada *comm_unit* e também possam acessar a camada *comm_unit* via acesso pré-determinado. A função de bufferização tem como objetivo permitir que palavras vindas de/para a camada *io_unit* sejam armazenadas em um *buffer* aumentando o desempenho da interface uma vez que os elementos da camada *prcs_unit* não precisarão esperar enquanto houver espaço dentro do *buffer* da camada *comm_unit*. A segunda função da camada *comm_unit* é realizar o escalonamento da interface. Este é um recurso limitado da arquitetura compartilhado pelos diversos processos concorrentes do sistema digital. Desta maneira é papel da camada *comm_unit* identificar que elemento da camada *prcs_unit* está envolvido na comunicação e dar a este elemento o controle da interface.

A camada mais interna da interface, a camada *io_unit* (Figura 58c), é responsável pela conexão do processador ao componente físico de *hardware*. Esta camada é a única camada da interface que depende da arquitetura onde o sistema digital será implementado. Esta arquitetura pode variar desde a utilização de componentes diferentes

tais como modelos distintos de processador até a maneira como processador e componentes de *hardware* estão conectados, ex FPGA. Como exemplo de diferentes arquiteturas pode-se ter uma arquitetura onde o processador está ligado ao componente de *hardware* via interface serial e uma outra arquitetura onde o mesmo processador está ligado ao mesmo componente de *hardware* via entrada e saída mapeada em memória.

Devido a esta flexibilidade de permitir a implementação de várias arquiteturas para o sistema digital a camada *io_unit* não é fixa como as camadas vistas anteriormente. Embora camadas *io_unit* diferentes possam implementar variadas formas de interconexão entre processador e componentes de *hardware*, estas possuem uma interface predefinida com a camada intermediária, a camada *comm_unit*.

Capítulo 5: Ferramenta InterfPiSH

Neste capítulo é apresentada a ferramenta InterfPISH, implementada no contexto desta dissertação. InterfPISH permite a co-síntese de um sistema digital incluindo a geração automática de interface entre processos implementados em *hardware* e *software*. Esta ferramenta implementa os modelos apresentados no capítulo 4. A primeira seção deste capítulo apresenta a estrutura da ferramenta dando uma visão geral de como a mesma está organizada. Ao longo das seções seguintes são vistos os diversos recursos e funcionalidades suportados pela InterfPISH tais como: gerenciamento de projeto, extração dos processos em *hardware* e em *software*, geração de código VHDL para os processos em *hardware*, geração de código C para os processos em *software,* seleção de processos de entrada e saída em *hardware* e *software* e seleção da arquitetura alvo, e gerador de modelos para inserção de novos processos de entrada e saída para uma determinada arquitetura.

A ferramenta InterfPISH tem como função permitir ao usuário projetista de sistemas digitais implementar de maneira simples e intuitiva um projeto dentro do sistema de *Co-design* PISH. Esta ferramenta foi desenvolvida para trabalhar em conjunto com ferramentas comerciais ou acadêmicas de síntese de *hardware* e compilação de *software*, não sendo seu papel realizar a síntese ou compilação propriamente, mas gerar código que pode ser sintetizado por ferramentas acadêmicas ou comerciais.

As linguagens escolhidas para geração de código são C, para as partes do sistema a serem implementadas em *software,* e VHDL, para as partes em *hardware*. Estas linguagens foram escolhidas devido a existência de uma grande quantidade de ferramentas acadêmicas e comerciais disponíveis que as utilizam. Vale ressaltar que no caso específico de VHDL há restrições quanto às ferramentas disponíveis devido a não existência de uma ferramenta que a suporte completamente. Grande parte das ferramentas de síntese de *hardware* baseadas em VHDL aceitem apenas um subconjunto da linguagem, que nem sempre é o mesmo para todas as ferramentas. Estas restrições são mais acentuadas em ferramentas acadêmicas, onde o subconjunto aceito é normalmente menor.

A ferramenta desenvolvida é capaz de gerar código padrão C e código padrão VHDL[5] aceito por algumas ferramentas comerciais.

5.1 ESTRUTURA DA FERRAMENTA INTERFPISH

Nesta seção é vista a estrutura da ferramenta InterfPISH. Na Figura 60 tem-se uma visão geral da ferramenta com seus blocos de funcionalidade: gerenciamento de projeto, geração de processos, manipulação de processos, seleção de E/S e geração de código.

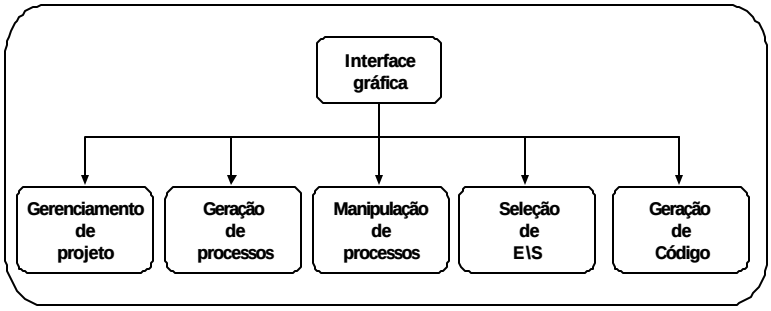

Figura 60: Visão geral da ferramenta InterfPISH

A ferramenta possui uma interface gráfica simplificada composta de menus, botões e uma janela de *log* onde são informados erros e dados resultantes de etapas realizadas durante a execução do programa. A interface gráfica pode ser vista na Figura 61. InterfPISH foi implementada utilizando-se a ferramenta visual de desenvolvimento JBuilder3.5[6] da Inprise Corporation. JBuilder3.5 **[referência]** aceita a linguagem Java e no caso específico foi utilizado o JDK1.2.2[7] **[referência]**.

Nas próximas seções deste capítulo serão vistos com mais detalhes cada um dos

---

[5] VHDL, normalmente aceito por ferramentas comerciais, Foundation® da Xilinx e MaxPlusII® da Altera.

[6] Versões de avaliação do JBuilder podem ser trazidas do *site* da Inprise (www.inprise.com). Estas versões possuem restrições de alguns recursos mas permitem a criação de programas completos.

[7] JDK é a sigla de *Java Development Kit* que é um ambiente de desenvolvimento gratuito fornecido pela SUN® Microsystems. Existem várias versões, sendo o JDK1.3 a mais nova.

blocos que compõem a ferramenta InterfPISH.

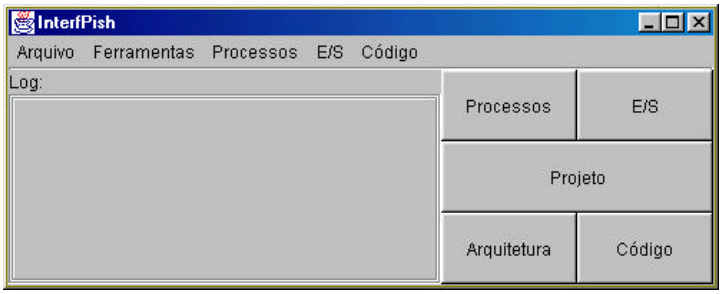

Figura 61: Ferramenta InterfPISH

5.2 GERENCIAMENTO DE PROJETO

A função do módulo de gerenciamento de projeto consiste em permitir ao usuário a criação de uma estrutura de diretórios para o projeto do sistema digital a ser implementado. Esta estrutura é importante tanto para facilitar o gerenciamento do projeto pelo usuário como para facilitar a utilização das ferramentas de síntese de *hardware* e os compiladores para síntese de *software*. Normalmente estas ferramentas já trabalham utilizando diretórios de projeto. Outra facilidade dada ao usuário consiste no armazenamento de informações anteriores do projeto em desenvolvimento. Isto faz com que o usuário da ferramenta não precise refazer passos que serão repetidos a cada nova utilização da ferramenta dentro do mesmo projeto. A ferramenta InterfPISH possibilita ao usuário especificar um nome particular para o seu projeto e definir o diretório base onde o projeto será implementado. Quando estas operações são realizadas, InterfPISH cria a árvore de diretórios mostrada na Figura 62. Basicamente são criados três diretórios e um arquivo de projeto. O primeiro diretório, com o nome do projeto (Figura 62b), é criado logo abaixo do diretório base. Neste diretório fica o arquivo de projeto contendo a informações tais como arquitetura utilizada, processos de E/S atualmente utilizados, configuração atual do sistema indicando que processos concorrentes estão em *hardware* e quais estão em *software*.

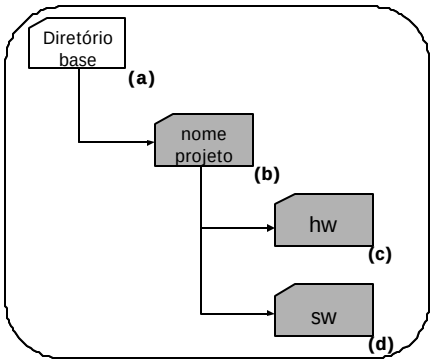

Figura 62: Estrutura de diretórios

Estas informações são úteis porque evitam que o projetista precise repetir todos os passos realizados num primeiro uso da ferramenta neste projeto. Os outros dois diretórios são criados em baixo do diretório de projeto, e são um diretório de projeto de *hardware,* diretório "hw" (Figura 62c), e outro de projeto de *software*, diretório "sw" (Figura 62d) que contém os arquivos em C. Na Figura 63 tem-se a janela de projeto da ferramenta InterfPISH onde é dado o nome do projeto e seu diretório base.

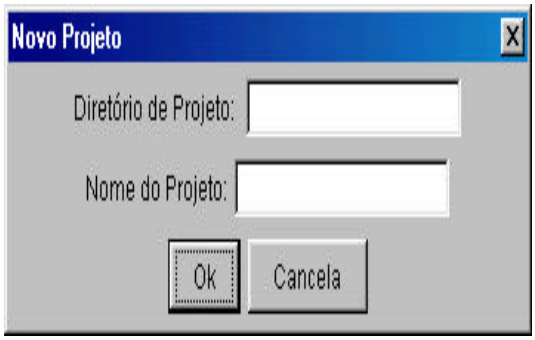

Figura 63: Novo projeto

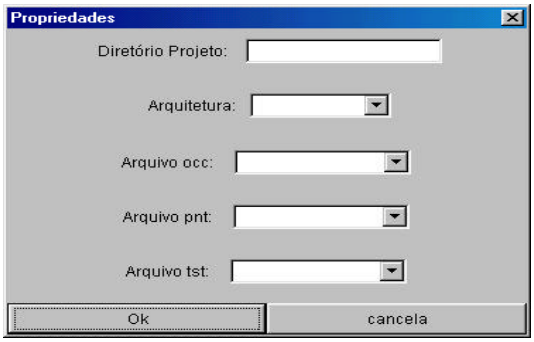

Figura 64: Propriedades de projeto

A janela de propriedades (Figura 64) é utilizada quando o usuário deseja trabalhar em um projeto já existente. Esta permite que o usuário altere a arquitetura alvo e os arquivos de entrada da ferramenta. Estas informações são armazenadas no arquivo de projeto de tal maneira que não precisam ser re-introduzidas por ocasião de uma nova utilização deste projeto. Os arquivos de entrada utilizados pela ferramenta são o arquivo occam original (arquivo occ), o arquivo contendo a estrutura da Rede de Petri (arquivo pnt) e o arquivo contendo o tipo das transições contidas na Rede de Petri. Estes arquivos

são fornecidos pelo usuário e deve estar no diretório de projeto criado anteriormente.

5.3 GERAÇÃO DE PROCESSOS

O bloco de geração de processos é responsável por extrair dos arquivos de entrada os processos concorrentes que compõem o sistema digital. A Rede de Petri representando o sistema está contida no arquivo pnt em formato texto. O bloco de geração de processos extrai esta informação da rede gerando objetos em Java que representam a Rede de Petri original. Esta função da ferramenta InterfPISH permite a extração e manipulação de processos. Como visto anteriormente (seção 4.5) os processos concorrentes são definidos como máquinas de estado finito que realizam apenas tarefas seqüenciais, sendo o sistema digital composto de um ou mais processos concorrentes.

5.3.1 Extração de Lugares Transições e Inserção de Fluxo de Dados

A Figura 65 mostra a extração de lugares que é realizada em cima do arquivo .pnt que consiste na Rede de Petri representando o fluxo de controle do sistema digital. Como pode ser observado é gerado um vetor de lugares contendo todos os lugares da Rede de Petri original. Na Figura 66 é mostrado o fluxo para extração de transições que é bastante semelhante ao fluxo para extração de lugares. A diferença básica entre os dois pode ser observada na última etapa da extração de transições que consiste na associação do fluxo de dados aos objetos transição contidos no vetor de transições. Assim como o vetor de lugares, o vetor de transições contém todas as transições existentes na Rede de Petri original.

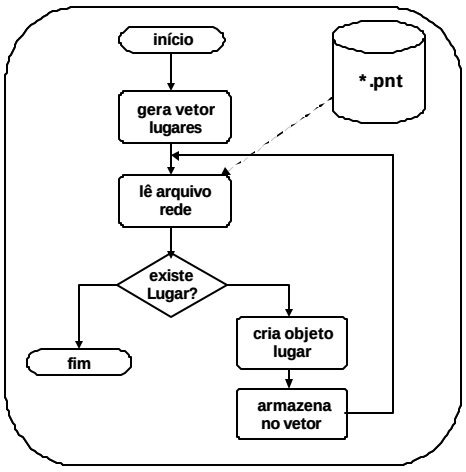

Figura 65: Extração de lugares

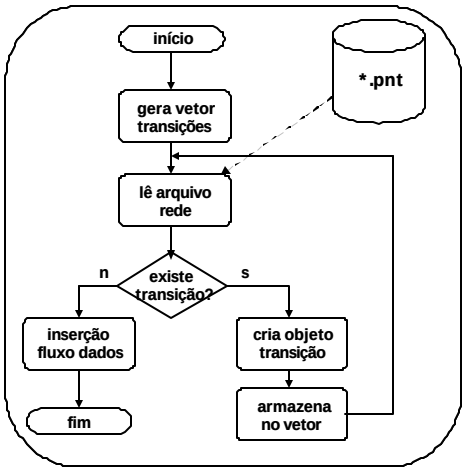

Figura 66: Extração de transições

A inserção do fluxo de dados associa a cada transição do vetor de transições as ações ou decisões a serem realizadas nesta transição específica. Na implementação atual da ferramenta InterfPISH o fluxo de dados é extraído diretamente da especificação em occam e assim como as declarações de canais e variáveis são convertidas para um formato interno neutro representando as operações. Isto torna a ferramenta independente da linguagem de especificação escolhida. Embora tente ser independente da linguagem de especificação occam, o fato de se extrair o fluxo de dados diretamente da especificação original é uma limitação atual da ferramenta. Assim como o fluxo de controle é extraído de uma Rede de Petri o mesmo deveria acontecer com o fluxo de dados. Para solução deste problema, trabalhos vêm sendo desenvolvidos [17][41] que permitirão a representação do fluxo de dados também como Rede de Petri, tornando a ferramenta totalmente independente da linguagem de especificação occam. O fluxo de dados é extraído de um arquivo .exp, como pode ser verificado na Figura 67, um *parser* foi desenvolvido para extrair expressões em occam e transformá-las em objetos. Atualmente o *parser* extrai as expressões da especificação em occam.

Na Figura 68 pode ser visto um exemplo de arquivo de expressões (.exp). Neste arquivo estão relacionadas as expressões às respectivas transições. No arquivo também pode se ver uma particularidade das transições de comunicação que podem ter mais de uma expressão associada. No exemplo da figura pode se verificar as expressões de envio da variável B e recebimento na variável A pelo canal ch relacionados à transição 3.

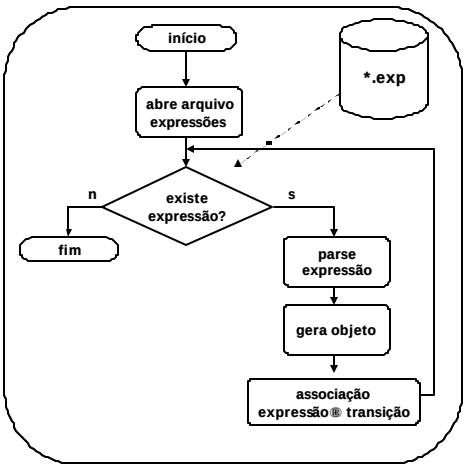

Figura 67: Extração de fluxo de dados

```
TR     EXPR
0:  A := 0
2:  A := 0
3:  ch ? A
3:  ch ! B
5:  B := 2
@
```

Figura 68: Exemplo de arquivo .exp

5.4 SELEÇÃO DE ENTRADA E SAÍDA

A janela de seleção de E/S (Figura 69) permite ao usuário associar um processo de E/S a um canal de comunicação de E/S. Quando esta janela é aberta automaticamente todos os canais de comunicação de E/S são listados no *combo box* de canais de E/S (Figura 69a) e identificada a direção da comunicação (Figura 69b) como entrada ou saída do sistema digital. Os processos de E/S disponíveis para o protocolo transferido pelo canal de comunicação de E/S são listados nos *list box* de processos de E/S em *hardware* (Figura 69d) e em *software* (Figura 69e). Também é mostrada uma descrição do processo de E/S na janela de descrição (Figura 69f). Nesta janela são descritas propriedades do processo de E/S tais como tamanho do dado, tipo de dispositivo de E/S controlado, etc. A lista de associações entre canais e processos de E/S (Figura 69c) mostra as associações de canais e processos selecionadas pelo usuário. Por último estão disponíveis para o usuário botões (Figura 69g) que permitem cancelar a seleção e fechar a janela (Cancela), fechar a janela aceitando as seleções feitas (Ok), associar um canal a um processo (Associa) e desfazer uma associação canal/processo (Desassocia).

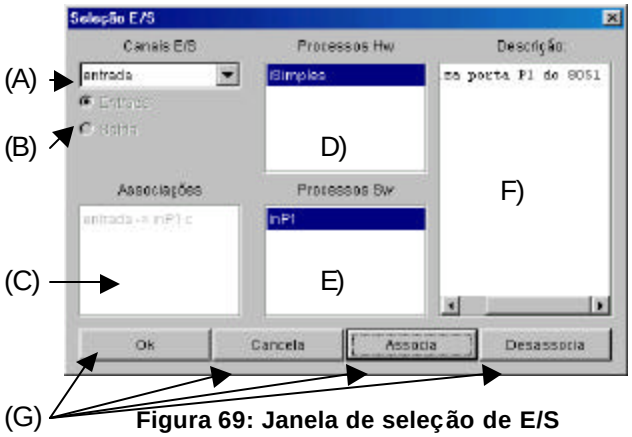

Figura 69: Janela de seleção de E/S

5.5 MANIPULAÇÃO DE PROCESSOS

A ferramenta permite ao usuário alterar manualmente o particionamento do sistema digital. Através da janela de manipulação de processos (Figura 70) o usuário pode identificar os processos existentes no sistema e sua hierarquia. A hierarquia, relação pai/filho, é identificada por e meio dos nomes dos processos. Onde o nome de um processo é composto do nome do seu processo pai seguido de um identificador numérico. Na janela de manipulação de processos são listados os processos em *hardware* (Figura 70a) e *software* (Figura 70b). O usuário pode selecionar que todo o sistema seja implementado em *hardware* ou *software* através dos botões de seleção total (Figura 70d) e também pode escolher quais processos serão implementados em *hardware* ou *software* através dos botões de seleção parcial (Figura 70d). Na implementação atual da ferramenta se o usuário move um processo de *hardware* para *software* e vice-versa todos os seus processos filhos são movimentados também.

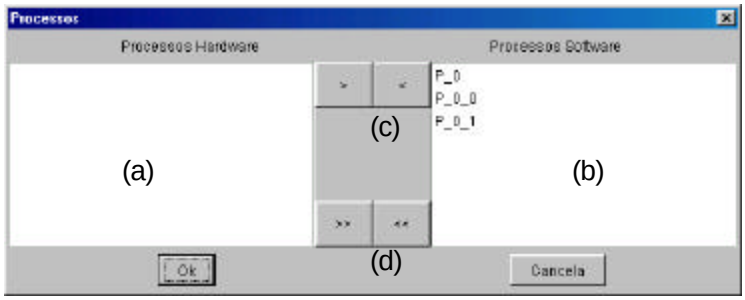

Figura 70: Janela de manipulação de processos

5.6 GERAÇÃO DE CÓDIGO

A ferramenta gera código C para as partes do sistema a serem implementadas em *software* e código VHDL para as partes do sistema a serem implementadas em *hardware*. São gerados arquivos de tipos que contem as definições de tipos e funções de conversão utilizadas por cada parte do sistema. São gerados arquivos em C e VHDL que representam as quatro partes componentes do sistema: processos, comunicação, interface, e E/S.

5.6.1 Geração de Arquivos de Tipo em Hardware e Software

Esta etapa consiste na implementação dos tipos de *hardware* e *software* a partir dos tipos do sistema. São utilizados recursos de definição de tipos das linguagens de implementação C e VHDL. Em C é utilizada a diretiva "#define" para a definição dos tipos de sistema. Vale salientar que a implementação do tipo de sistema em *software* depende do processador sendo utilizado. Isto acontece porque a implementação dos tipos em C é dependente do compilador/processador. Por exemplo o tipo "int" pode ser implementado com tamanho 16 bits em um microcontrolador ou 32 em um Pentium. Esta informação deve ser fornecida via arquivo de configuração da arquitetura alvo de modo que diferentes tipos possam ser aceitos. Em VHDL é utilizado o comando "subtype" que permite definir novos tipos a partir de tipos já existentes. Na Tabela 4 são mostrados os tipos de *hardware* e *software* definidos a partir dos tipos de sistema[8].

Na implementação é gerado um arquivo de cabeçalho "tipos.h" que contém todas as definições de tipos utilizadas pela ferramenta. Também é gerado o arquivo "tipos.vhd" contendo as definições de tipo para o *hardware*, juntamente com as funções de conversão de/para vetor mencionadas na seção 4.4.2. Todo o código gerado é dependente dos tipos definidos nos arquivos acima. Isto facilita a implementação da ferramenta uma vez que basta alterar os arquivos de tipos em C e VHDL.

[8] Os tipos de *software* tem como base a família de microcontroladores 8051.

Tipo do Sistema	Tipo em C	Tipo em VHDL
INT	signed char	integer range -128 to 127
BYTE	unsigned char	Integer range 0 to 255
INT16	signed int	Integer range -32768 to 32767
BOOL	unsigned char	Boolean
Tipo composto	Struct	record[9]
ARRAY	tipo array em C	array of <type>

Tabela 4: Tipos de *hardware* e *software*

5.6.2 Geração de Código VHDL para Processos em Hardware

Os processos são implementados como máquinas de estado finito em VHDL [46] e C. Exemplos de máquinas de FSM em VHDL e C são mostrados na Figura 71 e Figura 72 respectivamente. Em VHDL a FSM é implementada como um processo. Cada "when" dentro do "case" representa um estado e as mudanças de estado são representadas pela construção "P_0_state <= novo estado". Em C a máquina de estado é implementada de maneira semelhante por uma função ("void P_0()") cada estado é representado por um construtor "case" e as mudanças de estado pela atribuição de um valor de novo estado à variável P_0_state.

[9] Em algumas ferramentas de síntese de VHDL o tipo record não suporta um *array* como um de seus elementos.

```
P_0: process (reset, clk)
   variable filhos_ativados : boolean;
 begin
   if (reset = '1') then
     filhos_ativados := false;
     P_0_state <= P_0_s_inicio;
   elsif (clk = '1' and clk'event) then
     case P_0_state is
      when P_0_s_inicio =>
      P_0_state <= P_0_s0;
      when P_0_s0=>
        if (not filhos_ativados) then
          ativa_P_0_0 <= TRUE;
        end if;
        if (filhos_ativados) then
          if (finalizado_P_0_0) then
            filhos_ativados := false;
            P_0_state <= P_0_s4;
          end if;
        end if;
      when P_0_s4=>
     end case;
   end if;
 end process;end arc_P_0;
```

Figura 71: FSM em VHDL

```
void P_0 () {
  static unsigned int P_0_state = P_0_s_inicio;
  switch (P_0_state) {
    case (P_0_s_inicio) : {
     P_0_state = P_0_s0;
    }break;
    case (P_0_s0) : {
        ativa_P_0_0();
        if (finalizado_P_0_0()) {
          P_0_state = P_0_s4;
        }
    }break;
    case (P_0_s4) : {

    }break;
  }
}
```

Figura 72: FSM em C

Capítulo 6: Estudo de Caso

Este capítulo mostra como exemplo a implementação do comutador ATM proposto por [24], e cujo particionamento é descrito em [24].

Vale ressaltar que o comutador utilizado como estudo de caso neste trabalho é o mesmo utilizado para provar o funcionamento da ferramenta de particionamento ParTS, sendo descrito em [24]. Na seção 6.1 é feita uma descrição do comutador ATM e seu funcionamento. A seção 6.2 mostra a extração dos *Threads* que compõem o comutador. Na seção seguinte, 6.3, são gerados os objetos que representam o fluxo de dados e fluxo de controle do comutador e na seção 6.4 é explicada a inserção de *Threads* de E/S que permitem o acesso ao mundo exterior e finalmente na seção 6.6 são mostrados a geração de código e resultados.

6.1 DESCRIÇÃO DO COMUTADOR ATM

A utilização da rede telefônica analógica permite serviços de voz e o tráfego de dados a uma baixa velocidade. Esta baixa capacidade de transmissão de informação aliada à saturação da rede telefônica analógica fizeram com que as companhias telefônicas juntamente com o CCITT[10] propusessem a substituição da rede analógica por uma nova rede totalmente digital que ofereça diversos serviços de comunicação integrados [2]. A partir desta proposta inicial surgiu o padrão B-ISDN (*Broadband Integrated Services Digital Network)* que prevê uma rede digital de alta velocidade e grande largura de banda [2].

A implementação da rede ISDN é feita utilizando-se uma rede ATM (*Assinchronous Transfer Mode*). Esta rede transmite informação digital em pacotes, chamados células, cujo tamanho é fixo e igual a 53 bytes. Uma rede ATM é formada por comutadores ATM, sendo a função destes receber células por linhas de entrada e encaminhá-las corretamente por linhas de saída. Um comutador ATM genérico pode ser

[10] CCITT (*Comité Consultatif International de Teléphonique e Telegraphique*) – Comitê Consultivo Internacional de Telegrafia e Telefonia.

visto na Figura 73, onde pacotes chegam pelas linhas de entrada e são enviados pelas linhas de saída.

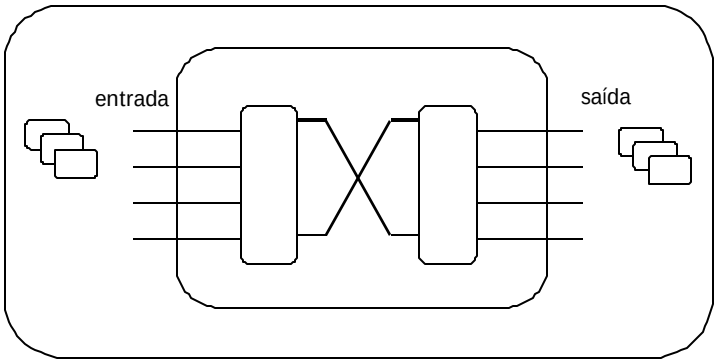

Figura 73: Comutador ATM

A comunicação em uma rede ATM é orientada à conexão. Isto implica que antes que sejam enviados dados da fonte ao destino, estes devem negociar uma conexão. O estabelecimento de uma conexão implica na definição de um caminho fixo entre fonte e destino e na determinação da qualidade da conexão. Para cada conexão estabelecida é associado um identificador único que é o registro da conexão. Cada comutador da rede possui uma tabela de rotas e nesta tabela são registradas as conexões das quais o comutador faz parte. Na Figura 74 é mostrada uma rede ATM onde a linha tracejada representa uma conexão e são identificadas as tabelas de rotas existentes em cada comutador.

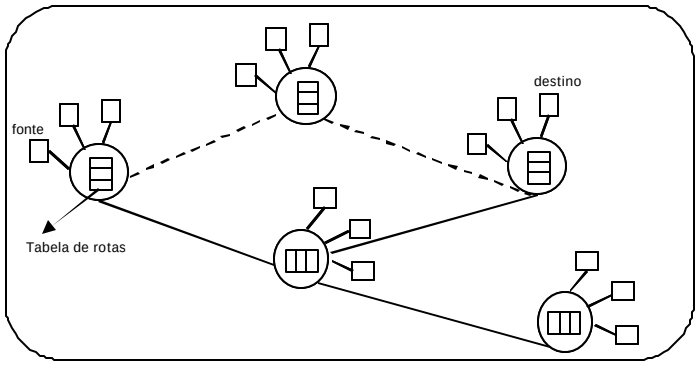

Figura 74: Conexão em Rede ATM

A conexão entre fonte e destino numa rede ATM é hierárquica, possuindo dois

níveis. O primeiro nível é chamado de caminho virtual e é dividido em vários canais virtuais. Desta maneira uma conexão ATM é identificada através dos identificadores de caminho e canal virtuais.

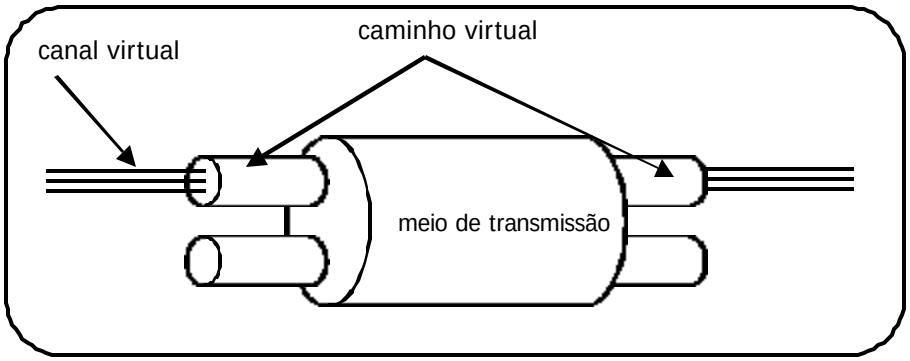

Figura 75: Caminho e canal virtuais

Como dito anteriormente, uma rede ATM transmite informação em células de tamanho 53 bytes. Estes bytes são organizados em campos agrupados em dois conjuntos: cabeçalho e *payload*. O primeiro conjunto, cabeçalho, é formado por 6 campos que ocupam 5 bytes e possuem informações a respeito dos dados sendo transmitidos. O *payload* é composto de 48 bytes e contém os dados sendo transmitidos. Os campos do cabeçalho são mostrados a seguir.

GFC (*General Flow Control*) – Definido para controle do fluxo de dados, mas não é utilizado atualmente.

VPI (*Virtual Path Identifier*)**-** Identifica o caminho virtual da célula.

VCI (*Virtual Channel Identifier*) – Identifica o canal virtual da célula.

PT (*Payload Type*) – Classifica o conteúdo do *payload* como dado do usuário ou dado de gerenciamento da rede.

CLP (*Cell Loss Priority*) – Este campo identifica a prioridade da célula, podendo a mesma ser de alta prioridade (CLP = 0) ou baixa prioridade (CLP = 1). Este campo tem tamanho de 1 bit.

HEC (*Header Error Check*) – Permite verificar a integridade dos dados do cabeçalho.

A tabela de rotas do comutador ATM contém informações sobre o destino da célula e a qualidade da conexão. A chave para acesso à tabela de rotas é composta pelos identificadores de caminho virtual e canal virtual. Ao rotear uma célula recebida por uma linha de entrada, o comutador substitui os identificadores de caminho e canal virtuais (VPI e VCI) por novos identificadores e envia a célula por uma das linhas de saída.

O comutador ATM deve decidir se uma célula dever ser enviada ou não. Uma célula pode ser descartada caso a taxa de chegada de células esteja muito acima da taxa combinada durante a negociação de uma conexão. São utilizados algoritmos de policiamento que permitem definir a transmissão ou não de uma célula. O procedimento de descartar uma célula tem como objetivo evitar o congestionamento da rede ATM.

O comutador proposto em [24] utiliza o algoritmo do balde furado. Este algoritmo faz analogia com um balde onde existe um pequeno furo na sua parte de baixo. Este balde pode ir enchendo através de um fluxo de água inconstante e é esvaziado através de um fluxo de água constante devido ao furo. Analogamente o comutador pode receber células com uma taxa de chegada variável mas as transmite com uma taxa de saída fixa. Quando o balde está cheio ocorre o transbordamento do mesmo, analogamente o comutador elimina as células que chegam quando o mesmo está cheio.

O algoritmo de policiamento do comutador utilizado como estudo de caso é o algoritmo de referência proposto pelo ITU, chamado de GCRA (*Generic Cell Rate Algorithm*) e baseado no balde furado. No GCRA o balde é representado por um contador de tempo. O algoritmo GRCA pode ser visto na Figura 76.

A variável X representa o valor do contador em um determinado instante. O tempo de chegada da última célula válida é armazenado na variável LCT (*Last Conformance Time*). A variável I representa o intervalo de tempo ideal entre a chegada de duas células consecutivas, enquanto L representa o intervalo mínimo aceitável entre a chegada de duas células consecutivas. O valor de $t_a(k)$ representa o tempo de chegada da célula atual. O tamanho do balde é dado por (I + L) . A cada unidade de tempo o balde é esvaziado em I e a cada célula aceita o balde é incrementado em I. Quando uma célula é aceita os valores de X e LCT são atualizados.

Os valores de intervalo ideal entre chegada de células consecutivas (I) e intervalo mínimo entre chegada de células são determinados durante a negociação de uma

conexão entre a fonte e o destino, ficando armazenados na tabela de rotas do comutador ATM.

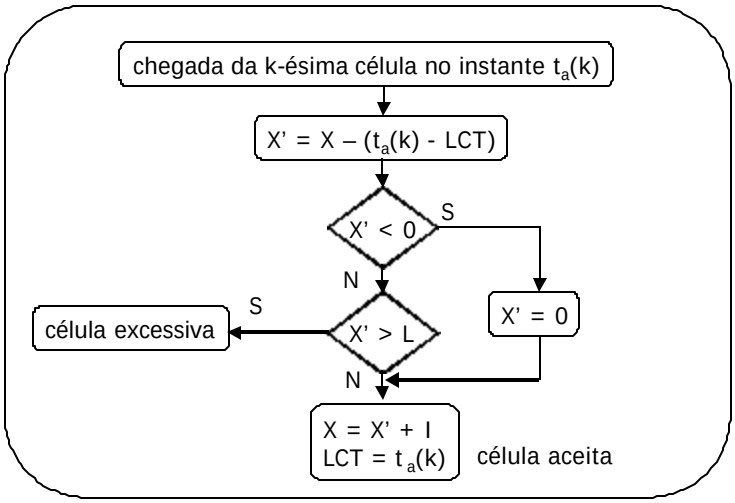

Figura 76: GCRA

A Figura 77 mostra a arquitetura do comutador ATM proposta em [24]. Esta é dividida em 3 módulos: sistema hospedeiro, tabela de rotas e comutador. O módulo sistema hospedeiro é responsável por realizar a negociação e estabelecimento de conexões. Este módulo atualiza na tabela as novas conexões, rotas, parâmetros de policiamento e estatísticas. O módulo comutador recebe uma célula por uma das linhas de entrada e acessa a tabela para calcular o policiamento e decidir se a célula deve ser aceita ou não. Caso a célula seja aceita deve ser feita a atualização do cabeçalho da célula e a atualização dos dados da tabela, com os dados do policiamento e estatísticas. A saída do comutador é composta pelo número da linha de saída, a qualidade de serviço da conexão e a nova célula com cabeçalho atualizado.

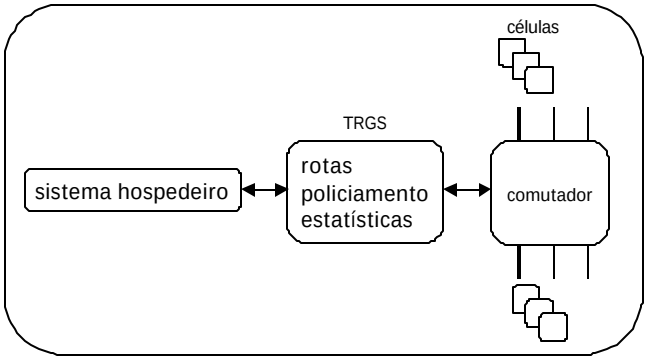

Figura 77: Arquitetura do comutador

O módulo comutador realiza 4 tipos de policiamento: pico alta, média alta, pico baixa e média baixa. Os policiamento de pico monitoram a conexão com relação à taxa máxima estabelecida durante a negociação da conexão enquanto os policiamentos de média observam a conexão com relação à média de chegada das células. Os termos "alta" e "baixa" referem-se à prioridade da célula (CLP = 0 para alta e CLP = 1 para baixa). O fluxo de policiamento é mostrado na Figura 78. Inicialmente verifica-se a prioridade da célula. Se esta for de alta prioridade é submetida ao policiamento de pico alta, caso aprovada é submetida ao policiamento de média alta. Se for aprovada em ambos os policiamentos a célula é aceita. Caso seja reprovada em algum dos policiamentos de alta, é submetida ao policiamento de baixa respectivo. Quando reprovada em um policiamento de baixa a célula é rejeitada. Se aprovada em ambos os policiamentos de baixa é verificada a prioridade da célula, se ela for uma célula de alta prioridade então tem sua prioridade rebaixada, uma vez que foi rejeitada nos policiamentos de alta. Uma célula de baixa prioridade é submetida inicialmente ao policiamento de pico baixa e caso aprovada é submetida ao policiamento de média baixa. Se aprovada em ambos a célula é aceita e caso seja reprovada em algum é eliminada.

Todos os policiamentos são baseados no algoritmo da Figura 76. O que os diferencia são os parâmetros X, LCT, I e L que possuem valores diferentes para cada um dos policiamentos executados.

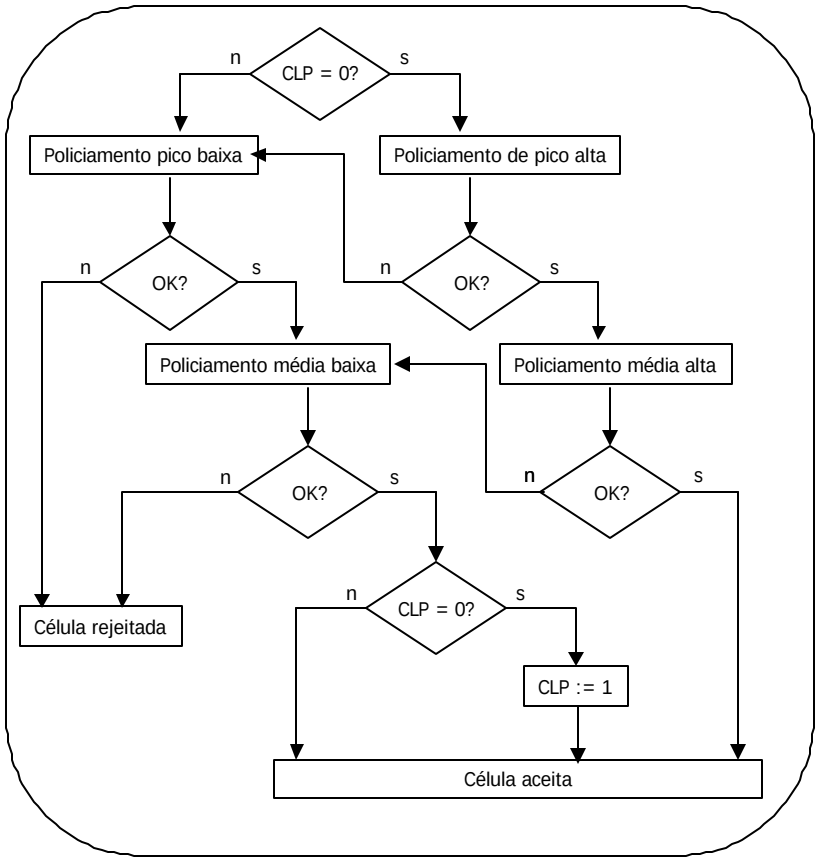

Figura 78: Fluxo de policiamento

6.2 COMUTADOR EM OCCAM

Nesta seção é mostrada a descrição em occam do comutador. Esta descrição representa o comutador após a etapa de *splitting*. A descrição do comutador (Figura 79) é composta das declarações de protocolo e canal seguidas de um construtor PAR. Isto indica que os processos de leitura de dados da célula, policiamento, aceitação, envio da célula, atualização da tabela e processos de controle são executados em paralelo. No processo de policiamento, os 4 tipos de policiamento são executados concorrentemente. Como está destacado na figura os processos que implementam o particionamento devem ser implementados em *hardware*, enquanto todos os outros processos devem ser implementados em *software*. Existem também 4 processos controladores que gerenciam os outros processos e a comunicação.

O comutador ATM possui seis canais de E/S que permitem que o mesmo acesse a tabela de rotas e receba e envie células. Os canais de comunicação de E/S são mostrados na Figura 80a. O canal chCell é de entrada e recebe uma célula. O canal ch1ReadTable, de saída, é utilizado para enviar os identificadores de caminho e canal virtual para ler os dados da tabela referentes á esta célula. O canal ch2ReadTable é utilizado para se receber os dados da tabela relacionados ao caminho e canal virtuais fornecidos via o canal ch1ReadTable. Estes dados são utilizados para se calcular os policiamentos. O canal chRouteTable de entrada é utilizado para receber da tabela os novos caminho e canal virtuais que irão atualizar o cabeçalho da célula a ser enviada. Os dois últimos canais de E/S, chOut e chWTable são utilizados para se enviar a célula com o cabeçalho atualizado e para atualizar os parâmetros da tabela respectivamente. Além dos canais de E/S existem mais 16 canais (Figura 80b) são utilizados para a comunicação entre os processos do comutador e os processos controladores. Cada canal transfere um conjunto de tipos especificado nas declarações de protocolo de canal. Na Figura 81 é mostrado o conjunto de tipos utilizados pelo canal chCell. Este canal transfere 8 expressões do tipo INT.

```
Declarações de Protocolo
Declarações de canais
PAR
  Processo de Leitura de Dados da Célula
  Processo de Policiamentos
    PAR
      Policiamento Pico Alta Hw
      Policiamento Media Alta Hw
      Policiamento Pico Baixa Hw
      Policiamento Media Baixa Hw
  Processo aceitação da célula
  Processo envio da Célula
  Processo de atualização da tabela
  Controlador0
  Controlador1
  Controlador2
  Controlador3
```

Figura 79: Comutador em occam

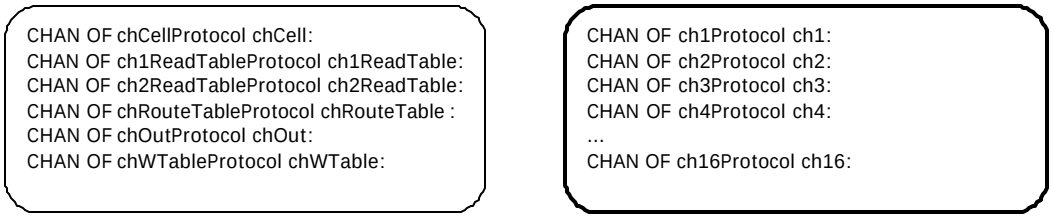

Figura 80: Canais

```
PROTOCOL chCellProtocol IS INT, INT, INT, INT, INT, INT, INT, INT:
CHAN OF chCellProtocol chCell:
```

Figura 81: Tipos enviados pelo canal chCell

6.3 EXTRAÇÃO DE DECLARAÇÕES E FLUXO DE DADOS

A Rede de Petri gerada a partir da descrição em occam do comutador é composta de 156 lugares e 154 transições, os arquivos utilizados contendo a Rede de Petri são mostrados no apêndice. Os arquivos representando a Rede de Petri e a descrição em occam são fornecidos para a ferramenta. A partir do arquivo que contem a descrição em occam a ferramenta gera dois arquivos modo texto. O primeiro “declaracoes.dec”, associa as declarações de canal como globais, e separa as declarações de variáveis indicando a que *Thread* a declaração pertence. Um trecho do arquivo de declarações é mostrado na Figura 82. As declarações de protocolo de canal e de canal são selecionadas como globais, estando entre [GLOBAL] e [END GLOBAL]. O *Thread* P_0 não possui nenhuma declaração de variável associada e o *Thread* P_0_0 possui três declarações de variáveis associadas.

Também é gerado o arquivo “expressoes.exp” que associa a cada transição uma expressão extraída da descrição original em occam. Um trecho do arquivo “expressoes.exp” é mostrado na Figura 83. Cada condição, ação de envio, ação de recebimento, operação lógico/aritmética da descrição em occam é associada a uma transição da Rede de Petri.

```
[GLOBAL]
  PROTOCOL chCellProtocol IS INT, INT, INT, INT, INT, INT, INT, INT:
  PROTOCOL ch1ReadTableProtocol IS INT, INT:
  ...
  PROTOCOL ch16Protocol IS INT:
  CHAN OF chCellProtocol chCell:
  CHAN OF ch1ReadTableProtocol ch1ReadTable:
...
  CHAN OF ch15Protocol ch15:
  CHAN OF ch16Protocol ch16:
[END GLOBAL]
[P_0]
  [P_0_0]
      INT GFC,VPI,VCI,PT,CLP,HEC, Payload,arrivalTime ,QoS,RCdP,RCdM,RC,AC:
      [3] INT newVPI,newVCI,portNo:
      [4] INT X,L,I,LCT:
  [END P_0_0]
  ...
[END P_0]
```

Figura 82: Arquivo de declarações

```
TR      EXPR
..
2:      chCell ? GFC;VPI;VCI;PT;CLP;HEC;Payload;arrivalTime
3:      ch1ReadTable ! VPI;VCI
90:       send := TRUE
91:       newRCdP := RCdP
92:       newRCdM := RCdM
93:       newRC := RC
94:       newAC := AC + 1
113:      ((NOT (ok[2])) AND ((CLP = 1) OR ((CLP = 0) AND (NOT (ok[0])))))
114:      send := FALSE
115:      newRCdP := RCdP + 1
116:      newRCdM := RCdM
117:      newRC := RC + 1
118:      newAC := AC
119:      newCLP := CLP
121:      TRUE
...
@
```

Figura 83: Arquivo de expressões

6.4 EXTRAÇÃO DE THREADS

Nesta etapa são extraídos os *Threads* que compõe o sistema digital. Para cada processo concorrente da descrição em occam é gerado um *Thread*. Como resultado são gerados 14 *Threads*. A Tabela 5 mostra os *Threads* gerados e seus respectivos número de lugares e transições. A última coluna da tabela mostra a natureza dos respectivos *Threads*, ou seja *hardware* ou *software*. A escolha de que *threads* devem ser

implementados em *hardware* ou *software* é determinada na etapa de particionamento do sistema PISH explicada na seção 3.3.

Thread	ID	Qtd. Lugares	Qtd. Transições	Natureza
0	P_0	2	2	sw
1	P_0_0	7	7	sw
2	P_0_1	4	4	sw
3	P_0_1_0	15	17	hw
4	P_0_1_1	15	17	hw
5	P_0_1_2	15	17	hw
6	P_0_1_3	15	17	hw
7	P_0_2	46	51	sw
8	P_0_3	8	9	sw
9	P_0_4	4	4	sw
10	P_0_5	7	7	sw
11	P_0_6	7	7	sw
12	P_0_7	7	7	sw
13	P_0_8	5	5	sw

Tabela 5: *Threads*

6.5 INSERÇÃO DE E/S

Nesta etapa o usuário associa a cada canal de E/S um *Thread* de E/S armazenado em uma biblioteca. A Tabela 6 mostra as associações de canais de E/S a *Threads* de E/S. A primeira coluna identifica o canal e tem o mesmo nome da declaração de canal na descrição em occam. A segunda coluna identifica a direção da comunicação, se entrada ou saída. A terceira coluna identifica os tipos de dados transferidos pelo canal de E/S, o número indica quantos dados daquele tipo são enviados. A penúltima coluna

identifica o processo de E/S associado ao canal. Este é o nome do arquivo que descreve o *Thread* em C ou VHDL* na biblioteca. Finalmente a última coluna identifica a natureza do *Thread* de E/S, se o mesmo é implementado em *software* ou *hardware*.

Canal E/S	Direção	Tipos	Thread E/S	Natureza
ChCell	entrada	8 INT	I8Int	Hw
Ch2ReadTable	entrada	21 INT	I21Int	Hw
ChRouteTable	entrada	9 INT	I9Int	Hw
Ch1ReadTable	Saída	2 INT	oDuplo	Hw
ChOut	Saída	15 INT	o15Int	Hw
ChWTable	Saída	12 INT	o12Int	Hw

Tabela 6: Associação canal E/S a *thread* E/S

6.6 GERAÇÃO DE CÓDIGO

Esta seção descreve a geração dos arquivos de implementação em *hardware* e *software* para os *Threads*, comunicação, E/S e interface.

6.6.1 Código VHDL e VHDL*

A Tabela 7 mostra os arquivos gerados para os *Threads* em *hardware*. São gerados 8 arquivos. Inicialmente são gerados 4 arquivos em VHDL*, sendo estes posteriormente convertidos para VHDL padrão.

Thread	VHDL*	Qtd. Linhas	VHDL	Qtd. Linhas
P_0_1_0	P_0_1_0.vhx	183	P_0_1_0.vhd	185
P_0_1_1	P_0_1_1.vhx	183	P_0_1_1.vhd	185
P_0_1_2	P_0_1_2.vhx	183	P_0_1_2.vhd	185
P_0_1_3	P_0_1_3.vhx	183	P_0_1_3.vhd	185

Tabela 7: Arquivos para *threads* em *hardware*

Os arquivos mostrados na Tabela 8 correspondem aos *Threads* de E/S. Estes estão armazenados em biblioteca com arquivos VHDL*, sendo convertidos para VHDL padrão.

Thread E/S	VHDL*	Qtd. Linhas	VHDL	Qtd. Linhas
I8Int	I8Int.vhx	77	I8Int.vhd	138
I21Int	I21Int.vhx	116	I21Int.vhc	216
I9Int	I9Int.vhx	80	I9Int.vhd	144
ODuplo	ODuplo.vhx	65	ODuplo.vhd	109
O15Int	O15Int.vhx	104	O15Int.vhd	187
O12Int	O12Int.vhx	95	O12Int.vhd	169

Tabela 8: Arquivos *threads* de E/S em *hardware*

No exemplo do comutador ATM não existe comunicação entre os *Threads* em *hardware*. Assim não são gerados arquivos de comunicação em *hardware*.

A Tabela 9 mostra os arquivos gerados para a camada mais externa da interface entre *hardware* e *software*, relativos aos canais de comunicação. No caso do comutador existe comunicação por canal entre os *Threads* em *software* e os *Threads* de E/S em *hardware*. Como existem seis canais de E/S e todos utilizam a interface são gerados 6 arquivos, e como os canais de E/S transferem dados de tamanhos diferentes são gerados arquivos de tamanho diferentes. A Tabela 10 mostra os *prcs's* utilizados para que o *Thread* P_0_1 em *software* possa ativar os 4 *Threads* em *hardware.* Como os *Threads* em *hardware* são iguais, todos os *prcs's* são iguais. Na Tabela 11 estão os arquivos dos *prcs's* que possibilitam aos *Threads* em *hardware* informar ao *Thread* em *software* que os ativou de suas respectivas finalizações.

Na Tabela 12 são mostrados os arquivos que implementam as três camadas da interface em *hardware.* O arquivo io_unit.vhd representa a camada mais interna da interface, comm_unit.vhd a camada intermediária e finalmente o arquivo prcs_unit.vhd utiliza como componentes os arquivos de *prcs's* vistos anteriormente.

Arquivo	Qtd. Linhas
prcs0.vhd	142
prcs1.vhd	104
prcs2.vhd	233
prcs3.vhd	149
prcs4.vhd	247
prcs5.vhd	214

Tabela 9: *Prcs's* de comunicação

Arquivo	Qtd. Linhas
Prcs6.vhd	236
Prcs7.vhd	236
Prcs8.vhd	236
Prcs9.vhd	236

Tabela 10: *Prcs's* de ativação de *thread*

Arquivo	Qtd. Linhas
prcs10.vhd	233
prcs11.vhd	233
prcs12.vhd	233
prcs13.vhd	233

Tabela 11: *Prcs's* de finalização de *threads*

Arquivo	Qtd. Linhas
io_unit.vhd	86
comm_unit.vhd	920
prcs_unit.vhd	1183

Tabela 12: Arquivos da interface em *hardware*

6.6.2 Código C

A implementação dos componentes do sistema digital em *software* é mais simples que em *hardware*. São gerados pares de arquivos ".c" e ".h" para cada parte do sistema digital. A exceção fica para o arquivo, atm_protocolo.c que contem a função "main" do programa em C. Este não possui o arquivo de cabeçalho correspondente. Vale observar que devido a não existência de *Threads* de E/S em *software* não são gerados arquivos para E/S. O arquivo "processos.c" contém os *Threads* em *software* do sistema. O arquivo "comunicacao.c" contém as estruturas e funções de comunicação utilizadas para a comunicação entre os *Threads*. Finalmente "io_unit.c", "comm_unit.c" e "prcs_unit.c" implementam as camadas da interface em *software*.

Arquivo C	Qtd. Linhas	Arquivo H	Qtd. Linhas
atm_protocolo.c	58	-	-
processos.c	573	processos.h	11
comunicacao.c	1997	comunicacao.h	791
e_s.c	-	-	-
io_unit.c	40	io_unit.h	2
comm_unit.c	230	comm_unit.h	17
prcs_unit.c	808	prcs_unit.h	182

Tabela 13: Arquivos do comutador ATM em *software*

6.7 ANÁLISE DOS RESULTADOS

Uma característica da especificação do comutador ATM é a existência de várias ações de comunicação. São utilizados 22 canais de comunicação. Outra característica da especificação está relacionada ao número de dados transferidos pelos canais de comunicação. Existem canais que transferem 31 palavras de 8 bits. Deve ser observado também que o processamento realizado no exemplo é bastante simples.

A grande quantidade de comunicação demonstra claramente a utilidade de uma ferramenta que gere a interface entre componentes de *hardware* e *software*. De forma automática. Estas características de muita comunicação aliada a pouco processamento implica num custo relativo muito alto da interface com relação à parte de processamento do sistema, representada pelos *Threads*.

Devido ao tamanho do código gerado não foi possível sua implementação. Este trabalho é uma opção a ser realizada futuramente.

Capítulo 7: Conclusão

Este capítulo de conclusão está dividido em 2 seções. Na primeira seção são enfatizadas as contribuições deste trabalho e finalmente na segunda seção são propostos trabalhos futuros.

7.1 CONTRIBUIÇÕES

A principal contribuição deste trabalho foi preencher a lacuna do sistema PISH, onde a implementação dos sistemas particionados automaticamente era realizada de forma manual.

O sistema PISH realiza o particionamento automático de uma especificação em occam e permite que o projetista determine manualmente, na especificação inicial, partes do sistema para serem implementadas em *hardware* ou *software*. Isto implica que, se o projetista quiser alterar o resultado do particionamento automático, só pode fazê-lo alterando a especificação inicial. A ferramenta proposta facilita a obtenção de diferentes particionamentos pois permite que o projetista selecione *Threads* para implementação em *hardware* ou *software* com um simples toque do *mouse.* Isto permite ao projetista o ajuste fino do particionamento, de modo a conseguir um melhor resultado.

Ainda dentro do ambiente acadêmico, esta ferramenta é interessante por permitir aos alunos verificar várias soluções diferentes para uma única especificação de um sistema digital que eles possam estar implementando. Isto é particularmente útil devido à existência de disciplinas relacionadas à área de *Co-design* nos cursos de graduação e pós-graduação em Ciência da Computação na UFPE.

A ferramenta separa claramente *Threads*, E\S, comunicação e interface. Isto possibilita uma rápida análise de cada uma das partes do sistema digital. Além disso, permite que modificações sejam feitas em apenas uma das partes da ferramenta, sem que seja necessário alterar outras partes.

Novas arquiteturas podem ser facilmente adicionadas à ferramenta. Devido ao seu modelo em camadas, o projetista precisa se preocupar apenas com o projeto das io_units em *hardware* e *software*. Isto também é facilitado pelo modelo simétrico da

interface que torna a implementação em *hardware* semelhante à implementação em *software*.

Para a implementação do código VHDL para os componentes em *hardware* do sistema digital foi utilizada uma extensão desta linguagem, chamada VHDL*. Embora seja utilizada na ferramenta de interface, ela também pode ser utilizada separadamente para a descrição de sistemas em hardware que utilizem comunicação síncrona. A vantagem pode ser vista no tamanho do código dos *Threads* de E\S, onde o código em VHDL* chega a ser metade do tamanho do código em VHDL.

A utilização de *Threads* de E\S permite a implementação simplificada, embora de forma manual, de novos recursos de E\S de uma determinada arquitetura.

7.2 TRABALHOS FUTUROS

Como trabalhos futuros pode-se destacar a geração automática de *Threads* de E\S onde, dadas as características dos dispositivos de E\S, o código de seus *Threads* seria gerado automaticamente. O mesmo pode ser realizado para a camada de io_unit, tornando assim mais rápida a adição de novas arquiteturas. Também relacionado a E\S está a implementação simplificada de *Threads* de E\S bidirecionais, uma vez que atualmente os *Threads* de E\S são unidirecionais.

Uma outra alternativa para a implementação do *software* seria a substituição de sua implementação como máquina de estados finitos por processos controlados por uma sistema operacional de tempo real (RTOS).

Atualmente não são feitas otimizações na Rede de Petri, o que resulta em um sistema maior do que poderia ser. Otimizações podem ser feitas eliminando-se transições que não realizam nenhuma ação, implicando apenas em mudança de estado na implementação. Outro tipo de otimização mais interessante é a substituição de várias ações em transições diferentes por uma única transição que realize várias ações em sequência. Isto aumentaria o desempenho da aplicação devido a redução no custo de mudança de estado.

Atualmente apenas o modelo de comunicação síncrona é implementado. É interessante a implementação de outros modelos de comunicação tais como

compartilhamento de variáveis.

Referências

[1] A. Orailoglu and D. D. Gajski *Flow graph representation*. In Proceedings of the 23rd ACM/IEEE Design Automation Conference, pp 503-509, Las Vegas, NV, June 1986, IEEE Computer Society Press.

[2] A. S. Tanenbaum. *Computer Networks* Prentice Hall, Inc. Third edition, 1996

[3] B. Lin, S. Vercauteren, H. De Man, *Constructing Application-Specific Heterogeneous Embedded Architectures for Custom HW/SW Applications*, In Proceeding of the ACM/IEEE Design Automation Conference, June 1996.

[4] B. Lin, S. Vercauteren, H. De Man, *Embedded Architecture Co-Synthesis and System Integration*, In Proceedings of the International Workshop on Hardware/Software Codesign, March 1996.

[5] C. Araújo, D. Silva, E. Barros, M. Lima and P. Maciel *Co-Synthesis and Prototyping Supporting the Design of Reconfigurable Systems*, In the Reconfigurable Computing Workshop, Marília, SP, 2000.

[6] C. Araújo, D. Silva, E. Barros, M. E. de Lima and P. Maciel. *Co-Synthesis and Prototyping Supporting the Design of Reconfigurable Systems*. In Proceedings of the Workshop em Computação Reconfigurável CORE-2000, PP. 54-66, Marília, Brasil, Agosto, 2000.

[7] C. Liem, F. Nacabal, C. Valderrama, P. Paulin, A.A. Jerraya. *Cosimulation and Software Compilation Methodologies for the System-on-a-Chip in Multimedia*. IEEE Design &Test of computers, special issue on Design, Test & ECAD in Europe, vol. 14 (2), pp. 16-25, April-June 1997.

[8] C.A. Valderrama, A. Changuel, P.V. Vijaya-Raghavan, M. Abid, T. Ben Ismail, A.A. Jerraya. *A Unified Model for Co-simulation and Co-synthesis of Mixed Hardware/Software Systems*. In Proceedings of the European Design and Test Conference (EDAC-ETC-EUROASIC'95), Paris, France, 6-9 March 1995.

[9] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[10] E. A. Lee and D. G. Messerschmitt. *Synchronous Dataflow.* In Proceedings of IEEE, 75(9):1235-1245, 1987.

[11] E. Barros and A. Sampaio,. Towards *Probably Correct Hardware/ Software Partitioning Using Occam*. In Proceedings of the Third International Workshop on Hardware/Software Codesign, (1994) 210-217, IEEE Press.

[12] E. Barros, M. Correia, T. Weber, A. Sampaio, J. Queiroz, M. Lima, N. Calazans e R. Jacobi. *Projeto Integrado de Software/Hardware*, Março de 1994.

[13] E. Barros, C. Araujo *An Approach for Interface Generation in the PISH Co-design System*. In Proceedings of the Brazilian Symposium on Integrated Circuits and System Design – SBCCI – Natal, 1999.

[14] E. Barros, C. Araujo. *Automatic Interface Generation among VHDL Processes in Hardware/Software Co-Design*. In Proceedings of the Forum on Design Languages - FDL'99 – Lyon, 1999.

[15] E. Robert *An Object Oriented Petri Net Approach to Embedded System Design*, TIK-Schrifenreihe, No.16, vdf Verlag 8092, Nov, 1996, ISBN 3-7281-2416-8

[16] E. Walkup and G. Borriello. *Automatic Synthesis of Device Drivers for Hardware/Software Co-design*. In techinical Report 94-06-04, Department of Computer Science and Enginnering, University of Washignton, Seatle, WA 98195, USA.

[17] F. Cruz, P. Maciel and E. Barros *Using Petri Nets for Data-Dependency Analysis* In the Proceedings of the International Conference of Systems, SMC Society,Man and Cybernetics, pp 2938-3003, Nashville, TN, USA, October 2000.

[18] F. Vahid and L. Tauro. *An object-oriented communication library for hardware-software Co-design*. In Proceedings of the 5th International Workshop on Hardware/Software Codesign, March 1997.

[19] G. Jones. *Programming in Occam*. C.A.R. Hoare Series Editor, Prentice-Hall International Series in Computer Science, 1987.

[20] Geof Barret *Occam 3 Reference Manual*, INMOS, 1992.

[21] *Hardware/Software Co-Design: Principles and Practice* edited by J. Staunstrup and W. Wolf, Kluwer Academic Publishers, 1997, ISBN 0792380134.

[22] http://www.ida.ing.tu-bs.de/projects/cosyma/home.e.shtml

[23] J. B. Dennis. *Data Flow Supercomputers*. IEEE Computer Magazine, pp 48-56, 1980.

[24] J. A. G. Lima. *Um Controlador Microprogramado para Comutadores ATM*. Tese de Doutorado, Universidade Federal da Paraiba, Brasil, 1999.

[25] J. Yioda *ParTS – Uma Ferramenta de Suporte ao Particionamento Hardware/Software,* Universidade Federal de Pernambuco, Recife, 1999. Dissertação Mestrado.

[26] K. Jurdsen “An Introduction to the Practical Use of Coloured Petri Nets”. In Wolfgang Reisig and Grzegorz Rozenberg, editors, volume 1492 of Lectures on Petri Nets II: Applications, pages 237-292. Springer, 1998. ISBN 3-540-65307-4.

[27] K. Van Rompaey, D. Verkest, I. Bolsens, and Hugo De Man, *CoWare - A Design Environment for Heterogeneous Hardware/Software systems.* In Proceedings of the European Design Automation Conference (EURO-DAC), September 1996.

[28] L. Silva. *An Algebraic Approach to Hardware/Software Partitioning*. Tese de Doutorado, Universidade Federal de Pernambuco, 2000.

[29] L. Silva, A. Sampaio e E. Barros. *A Normal Form Reduction Strategy for Hardware/Software Partitioning*. In Jonh Fitzgerald, Clif B. Jones and Peter Lucas, editors, FME’97: Industrial Applications and Strengthened Foundations of Formal Methods (Proc. 4th Intl. Symposium of Formal Methods Europe, Graz, Austria, September 1997), volume 1313 of Lecture Notes in Computer Science, pages 624-643. Springer Verlag, September 1997. ISBN 3-540-63533-5.

[30] L. Silva, A. Sampaio, E. Barros e J. Iyoda. *An Algebraic Approach to Combining Processes in a Hardware\Software Partitioning Environment*. Lecture Notes in Computer Science, 1548:308-324, 1998.

[31] M. Eisenring and J. Teich, *Domain-Specific Interface Generation from Dataflow Specification*. In Proceedings of the Sixth International Workshop on Hardware/Software Co-design CODES98, pp 43-47, March 15-18, 1998, Seattle, Washington.

[32] M. Eisenring, J. Teich and L. Thiele *Rapid Prototyping of Dataflow Programs on Hardware\Software Architectures*. In Proceedings of the 31th Annual Hawaii International Conference on System Sciences, Vol. VII, pp 187-196, Kona, Hawaii, January 1998.

[33] M.Eisenring, M. Platzner and L. Thiele *Communication Synthesis for Reconfigurable Embedded Systems*. In Proceedings of the 9th International Workshop on Field-Programmable Logic and Applications, FPL´99, Lecture Notes on Computer Science, 1673, Aug.30, 1999, pp 205-214.

[34] M. Santos, E. Barros and C. Araujo. *An FPGA Based Implementation of an Intravenous Infusion Controller System.* In Proceedings of the ASAP'97 – IEEE International Conference on Application Specific Systems, Architectures and Processors. Zurique, Suiça, Julho 1997, IEEE.

[35] M. Santos, E. Barros and C. Araujo *IVICS. An Intravenous Infusion Controller System.* Anais da XII Conferência da Sociedade Brasileira de Microeletrônica. Caxambu, Brasil, Agosto de 1997.

[36] P. Chou, R. B. Ortega, G. Borriello, *Interface Co-Synthesis Techniques for Embedded Systems*. In Proceedings of the IEEE/ACM International Conference on Computer-Aided Design,San Jose, CA, November, 1995. pp.280-287.

[37] P. Chou, R. Ortega, G. Borriello, *Synthesis of the Hardware/Software Interface in Microcontroller-Based Systems.* In Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, Santa Clara, CA, November 1992, pp.488-495.

[38] P. Maciel, E. Barros, W. Rosenstiel.*A Petri Net Model for Hardware/Software Codesign*. Design Automation for Embeddedd Systems Journal, Kluwer Academic Publishers.1999.

[39] P. Maciel, E. Barros, W. Rosenstiel.*A Petri Net Approach for Estimating Hardware Area Considering Clock Period*. IEEE 15th International Conference on CAD/CAM Robotics & Factories of the Future, Àguas de Lindóia, Brazil.August, 1999.

[40] P. Maciel, E. Barros, W. Rosenstiel. *Estimating Functional Unit Number in PISH Codesign System by Using Petri Nets*. In the Proceedings of the IEEE 12th Symposium on Integrated Circuits and System Design, Natal, Brazil, October, 1999.

[41] P. Maciel, F. Cruz and E. Barros *Estimates Based on Petri Nets for Hardware/Software Co-design* In the Proceedings of the 4th Portuguese Conference on Automatic Control, pp 546-551, APCA, IFAC, Guimarães, Portugal, Outubro 2000.

[42] P. Maciel, R. Lins e P. Cunha *Introdução às Redes de Petri e Aplicações*. X Escola de Computação, Campinas-SP, Brasil, julho 1996.

[43] R. B. Ortega and G. Borriello. *Communication Synthesis for Distributed Embedded Systems*. In Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, San Jose, CA, November 1998.

[44] R. B. Ortega, G. Borriello. *Communication Synthesis for Embedded Systems with Global Considerations*. In Proc. CODES/CACHE, Braunschweig, Germany, March 24-26, 1997, pp. 69--73.

[45] R. B. Ortega L. Lavagno and G. Borriello *Models and Methods for hw/sw intellectual property interfacing*. In 1998 NATO ASI on System Level Synthesis.

[46] S. Mazor. *A guide to VHDL* 2nd Edition, ISBN 0-7932-9387-2

[47] T. Murata. *Petri Nets: Properties, Analysis and Applications*. In Proceedings of the IEEE, 1989.

[48] www.cin.ufpe.br/~pish