UMA METODOLOGIA PARA IMPLEMENTAÇÃO DE PROTOCOLOS DE COMUNICAÇÃO UTILIZANDO HARDWARE/SOFTWARE CODESIGN

Ricardo Neimo Belem Lima

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Aprovada por:

Prof. Aloysio de Castro Pinto Pedroza, Dr.Ing.

Prof. Antônio Carneiro de Mesquita Filho, Dr. d'Etat

Prof. José Ferreira de Rezende, Dr.

Prof. Luci Pirmez, D. Sc.

Prof. Marco Aurélio Cavalcanti Pacheco, Dr.

RIO DE JANEIRO, RJ - BRASIL

MARÇO DE 2004

LIMA, RICARDO NEIMO BELEM

Uma Metodologia para Implementação de Protocolos de Comunicação Utilizando Hardware/Software Codesign [Rio de Janeiro] 2004

XV, 126 p. 29,7 cm (COPPE/UFRJ, D.Sc., Engenharia Elétrica, 2004)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Protocolos de Comunicação
2. HW/SW Codesign
3. Implementações em HW

I. COPPE/UFRJ II. Título (série)

Agradecimentos

Ao professor Aloysio de Castro Pinto Pedroza, pela atenção e incentivo dedicados à mim desde a fase em que estive na Iniciação Científica, passando pelo Mestrado e principalmente agora no Doutorado. Ele contribuiu de forma decisiva para a realização deste trabalho.

Ao professor Antônio Carneiro de Mesquita Filho por estar sempre disposto a esclarecer qualquer dúvida e apontar soluções para problemas que pareciam intransponíveis. Devido a eles, hoje em dia tenho um novo conceito para a palavra Orientadores.

Aos professores Luci Pirmez, José Rezende e Marco Aurélio Pacheco por participarem da banca e contribuirem para o desenvolvimento deste trabalho.

Aos meus amigos de Grupo de Teleinformática e Automação (GTA) pelo auxílio nas tarefas disciplinares e pelo excelente convívio extra-classe.

A todo o pessoal do Laboratório de Projetos de Circuitos (LPC) pela infra-estrutura cedida em termos de recursos computacionais, suporte técnico e acima de tudo pelo companheirismo e troca de idéias.

À minha família, pois sem seu apoio não teria chegado ao final desta fase.

A todos, que direta ou indiretamente tenham me incentivado durante a conclusão deste curso.

Um agradecimento especial aos meus amigos João Marcelo, Salomão, Castanon, Márcio Miranda, Rubi, Fábio Antônio, Édson Granja, Fábio Dutra, Marcelo Azambuja, Renato Dutra e a todos os demais.

Este trabalho foi realizado com recursos da UFRJ, FUJB, CNPq e CAPES.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

UMA METODOLOGIA PARA IMPLEMENTAÇÃO DE PROTOCOLOS DE COMUNICAÇÃO UTILIZANDO HARDWARE/SOFTWARE CODESIGN

Ricardo Neimo Belem Lima

Março/2004

Orientador: Aloysio de Castro Pinto Pedroza

Programa: Engenharia Elétrica

O objetivo deste trabalho é apresentar uma metodologia de *Hardware/Software Codesign* aplicada à síntese de protocolos de comunicação. O projeto se inicia com a especificação dos protocolos através de um modelo de Máquina de Estados (MEF), onde cada transição de estado representa uma operação ou um conjunto de operações a serem avaliadas. O modelo de MEF adotado neste trabalho foi adaptado de modo a incorporar características específicas de implementações em *hardware*. Este modelo permite, além de descrever as funcionalidades do protocolo, se preocupar de antemão com as futuras implementações em *hardware*. Um conjunto de valores de atraso e custo de projeto são extraídos, fornecendo assim, subsídios que auxiliem o projetista no processo de escolha da melhor partição da especificação. O ciclo de projeto se encerra com a integração dos módulos após a etapa de partição do sistema. A etapa de integração se torna bastante simplificada pois o método utilizado para descrever o protocolo em alto nível já se encarregou de especificar várias características inerentes à comunicação entre entidades de *hardware*. Implementações utilizando a ferramenta Synopsys para síntese de circuitos ASIC com biblioteca de células padrão e implementações com a ferramenta ALTERA para prototipagem rápida em PLD são consideradas.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

A METHODOLOGY FOR PROTOCOL IMPLEMENTATION USING HARDWARE/SOFTWARE CODESIGN

Ricardo Neimo Belem Lima

March/2004

Advisor: Aloysio de Castro Pinto Pedroza

Department: Electrical Engineering

A Hardware/Software Codesign Methodology applied to the synthesis of communication protocols is presented in this work. Iniatially, the designer specifies the protocol by a state diagram, where each state transition is a set of clauses and expressions to be evaluated. The state diagram used in this work was modified to capture hardware implementations specific characteristics. At the end of synthesis process, delay and area measures for each transition are obtained. These measures are used to evaluate if the cost and delay of the chosen partition are within the bounds calculated. Implementations using Synopsys tool to the synthesis of ASIC circuits with a standard cell library for a specific technology and using ALTERA tool to the fast prototyping in Programmable Logic Devices (PLD) are considered.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

NOS últimos anos, as técnicas de projeto de circuitos integrados tiveram uma grande evolução. Ao mesmo tempo ocorreu uma evolução sem precedentes nas arquiteturas a base de processadores (RISC - *Reduced Instruction Set Computer*) e elementos programáveis (FPGA - *Field Programmable Gate Array*, PLD - *Programmable Logic Devices*). Atualmente, devido ao surgimento de novas e complexas aplicações, um dos sistemas mais estudados são os protocolos de comunicação. As atuais redes de computadores de alta velocidade e alto desempenho impuseram modificações no projeto destes protocolos. O uso da otimização em *software* na elaboração de protocolos para as atuais redes nem sempre permite operações a altas taxas de velocidade.

Um fator importante para o bom desempenho dos protocolos de alta velocidade é a utilização integrada de *hardware* e *software* durante sua implementação. Neste sentido, algumas operações do sistema, normalmente as menos críticas em termos de velocidade, podem ser implementadas em *software*, enquanto que aquelas que necessitam de uma maior velocidade devem ser implementadas em *hardware*. Devido à crescente demanda por protocolos com altas vazões, as soluções que utilizam *hardware* são cada vez mais investigadas.

O princípio do *Codesign* [1, 2, 3, 4, 5] é a realização de um projeto cooperativo baseado em dois ambientes de projeto específicos, *hardware* e *software*. A verificação

e a simulação de todo o sistema são realizadas em qualquer etapa do projeto e este tipo de concepção conduz a um aumento da produtividade na indústria. Para fazer face ao aumento da complexidade dos sistemas eletrônicos, e para responder aos critérios de desempenho esperados, é conveniente realizar o projeto destas aplicações em um nível de abstração elevado.

Aumentando o nível de abstração ao nível de sistema, o projetista pode manipular aplicações mais complexas, com um domínio total das tecnologias. O projetista pode escolher a tecnologia que melhor implementa cada parte do seu sistema, levando em consideração os critérios de desempenho e custo. A primeira etapa no projeto de sistemas é a especificação de sua funcionalidade, que é realizada normalmente, através da utilização de um modelo. O objetivo principal é descrever a funcionalidade de um sistema em um nível de detalhemento suficiente que permita prever o seu comportamento completo. Diferentes modelos são requisitados em diferentes domínios de aplicação. Este trabalho opta por utilizar um modelo mais apropriado à especificação de protocolos de comunicação.

À medida que se avança no detalhamento da especificação de um dado sistema, torna-se necessária a aplicação de técnicas que auxiliem o projetista a tomar decisões relativas à partição *hardware/software* das operações do protocolo. Apesar dos tempos de execução em *hardware* serem, normalmente, muito menores do que na implementação em *software*, o custo deste tipo de solução é maior. Portanto, deve-se estabelecer um compromisso entre o desempenho desejado e o custo de implementação no momento de se decidir qual parte do protocolo deve ser implementada em *hardware* e qual parte deve ser implementada em *software*. Esta etapa é denominada de *particionamento* e sua decisão tem impacto direto no desempenho do sistema de comunicação do qual o protocolo faz parte.

O projetista ainda dispõe de poucos recursos para auxiliá-lo no refinamento da especificação, no desenvolvimento do projeto e na partição *hardware/software* de um dado protocolo [1]. Normalmente, a experiência do projetista aliada a técnicas informais constituem a base para se definir a melhor partição, o que limita a exploração do espaço de soluções. A tarefa do projetista exige métodos e meios que

permitam valorar e analisar os diferentes tipos de arquiteturas do sistema afim de atender melhor às restrições de desempenho e custo e garantir uma reação rápida a uma mudança da tecnologia de implementação. Neste caso, é útil fornecer ao projetista subsídios que auxiliem-no na escolha da melhor partição.

A avaliação de uma dada partição é realizada através do uso de parâmetros. Alguns dos parâmetros comumente utilizados são: custo econômico, desempenho, consumo, área de silício, tamanho da memória, linhas de código ou custo da comunicação. Os parâmetros utilizados devem ser combinados numa função de custo única, denominada *função objetivo*. Esta função fornece uma medida da qualidade da partição em questão e através de sua avaliação pode-se comparar duas partições e selecionar aquela que satisfaz melhor as especificações. Este trabalho se concentra na obtenção dos parâmetros de custo e atraso de implementações auxiliando assim o projetista a avaliar uma dada partição.

A integração do sistema particionado se dá através da prototipagem, que é a etapa responsável em gerar o código executável para cada domínio de projeto. Cada subsistema é traduzido separadamente, e ao final tem-se uma arquitetura heterogênea representada pelos códigos da linguagem de descrição de *hardware* e de *software*. Normalmente, esta etapa consiste da síntese de *hardware*, da síntese de *software* e da síntese da comunicação.

Muitos grupos de pesquisa em renomadas universidades estão desenvolvendo ambientes de projeto baseados na metodologia de *Codesign*. Dentre eles podemos citar: COSMOS [6], SpecSyn [7, 8], Ptolomey [9], LYCOS [10], Chinook [11], Olympus [12], AKKA e BEKKA [13, 14] e PISH [15]. Estes ambientes de *Codesign* diferem na linguagem de especificação do sistema, no método de particionamento, na arquitetura alvo e nos métodos de validação utilizados. Os sistemas-alvo destes projetos são processadores de sinais digitais, processadores de televisão interativos, controladores de tempo-real, comutadores de rede e produtos de telecomunicações. Nenhum destes ambientes focaliza o seu projeto para o caso específico de protocolos de comunicação.

1.1 Objetivo do Trabalho

A utilização conjunta de *hardware* e *software* na concepção de protocolos de comunicação assume uma maior importância devido à crescente demanda por redes de alto desempenho e por aplicações que exigem altas taxas de processamento. O objetivo deste trabalho é apresentar uma metodologia de *Hardware/Software Codesign* aplicada à síntese de protocolos de comunicação.

Na metodologia desenvolvida neste trabalho, os protocolos são especificados, inicialmente, por um modelo de Máquina de Estados (MEF), onde cada transição de estado representa uma operação ou um conjunto de operações a serem avaliadas. A implementação de sistemas em *hardware* não é uma tarefa trivial. Várias características de sistemas facilmente descritas em *software* podem se tornar impossíveis de serem sintetizadas em *hardware*. Por esta razão, outro objetivo deste trabalho é propor um modelo baseado em MEF adaptado de modo a incorporar características específicas de implementação em *hardware*. A idéia é adotar estratégias, baseado em estudos passados, para facilitar as implementações em HW de sistemas.

Este modelo visa facilitar as implementações em *hardware* do protocolo, bem como mostrar ao projetista, ainda em um nível de abstração elevado, as restrições que estas implementações podem impor. Outra vantagem que a adoção deste modelo implica é facilitar a tarefa de síntese de *hardware*, uma vez que o modelo de descrição do protocolo em alto nível já previu alguns detalhes típicos de implementações de circuitos integrados.

Este modelo serve como base para a descrição das especificações de todas as transições do protocolo na linguagem VHDL (*Very High-Speed Integrated Circuits Hardware Description Language*) [16, 17]. Estas especificações são utilizadas como entrada para as ferramentas Synopsys [18] e ALTERA [19] para a síntese de *hardware*. Estas ferramentas fornecem informações de custo de *hardware* e medidas de atraso a partir da análise dos circuitos sintetizados. Estas mesmas transições são também implementadas na linguagem C, de modo a se obterem medidas de desempenho e custo de programação para uma implementação em *software*. Todos estes

valores, tanto da síntese de *hardware* quanto da implementação em *software*, são utilizados como parâmetros para a otimização de uma função objetivo no processo de particionamento. Este processo é encarregado de, efetivamente, indicar quais partes serão implementadas em *hardware* e quais partes serão implementadas em *software* utilizando para tal um algoritmo genético [20, 21] e uma ferramenta de análise de desempenho Tangram-II [22][23]. A desvantagem de um possível aumento no tempo de projeto para se calcular tais parâmetros é compensada pela melhor qualidade dessas medidas em relação aos resultados obtidos por estimadores de *hardware* e de *software*.

Ao final do processo de otimização da função objetivo, são apontadas quais transições serão implementadas em *hardware* ASIC (*Application Specific Integrated Circuits*), em *hardware* PLD ou em *software*. A partir deste resultado, este trabalho efetua o processo de integração do sistema, preocupando-se principalmente com a síntese da comunicação. Este processo, por sua vez, será menos trabalhoso, pois muitos detalhes inerentes à comunicação entre entidades de *hardware* já foram descritos no modelo de alto nível.

1.2 Roteiro do Trabalho

O capítulo 2 apresenta a metodologia de *Codesign* mostrando as principais características desta técnica, bem como suas vantagens em relação às abordagens tradicionais. Um detalhamento das principais etapas de desenvolvimento de um projeto utilizando esta técnica é abordado. Também são apresentadas as principais características dos protocolos de comunicação.

O capítulo 3 apresenta um resumo dos principais ambientes de *Codesign* no mundo e outros trabalhos relacionados com o tema. São apresentadas as principais diferenças entre ambientes desenvolvidos por diversos grupos de pesquisa ressaltando as características mais importantes de cada um.

O capítulo 4 apresenta a metodologia proposta para a síntese de protocolos de comunicação utilizando *HW/SW Codesign* e apresenta as ferramentas utilizadas para

a síntese de *hardware* (Synopsys e ALTERA). É apresentado um detalhamento da metodologia onde são abordados, inclusive, todas as decisões de projeto adotadas.

No capítulo 5 são apresentados e analisados os resultados obtidos utilizando-se a metodologia proposta no projeto de três protocolos usados como estudos de caso. São apresentadas as diversas medidas de custo e desempenho de implementação das transições dos protocolos em *hardware* e *software*. São apresentadas quais partes deverão ser implementadas em *hardware* e quais partes deverão ser implementadas em *software*, ou seja, o resultado final do processo de particionamento, bem como as etapas de refinamento e integração do sistema.

No capítulo 6 são apresentadas as conclusões e contribuições do trabalho desenvolvido, as suas vantagens e desvantagens em relação aos métodos conhecidos e as sugestões para trabalhos futuros.

Capítulo 2

HW/SW Codesign de Protocolos de Comunicação

2.1 Introdução

A I nteração dos caminhos de desenvolvimento de projetos de *hardware* e *software* é a idéia principal do *Codesign*. Esta interação ocorre de diferentes modos e em diferentes níveis. O *Codesign* oferece ao projetista um conjunto de métodos e ferramentas para a prototipagem rápida e para a avaliação do desenvolvimento de sistemas complexos. A análise das decisões de projeto é útil para permitir a otimização da arquitetura do sistema em função das restrições impostas quando da especificação do sistema. A concepção conjunta está ligada aos aspectos técnicos e à organização do trabalho. Esta organização engloba a utilização conjunta de várias ferramentas de projeto e a união de vários aspectos relevantes a todos os níveis do ciclo de desenvolvimento.

A utilização da metodologia de *Codesign* no projeto de protocolos de comunicação consiste na idéia chave deste trabalho. Com o intuito de se atender aos requisitos de velocidade das atuais redes de alto desempenho, tornou-se necessário recorrer à utilização integrada do *hardware* e do *software* durante a implementação de tais protocolos. A implementação de determinadas funções de um protocolo em *hardware*

visa a obtenção de subsistemas de alta velocidade.

A seção 2.2 apresenta os conceitos referentes ao assunto de *Codesign* mostrando as principais características desta técnica, bem como suas vantagens em relação às abordagens tradicionais. A seção 2.3 apresenta um detalhamento das principais etapas de desenvolvimento de um projeto utilizando a técnica de *Codesign*. A seção 2.4 apresenta as principais características de protocolos de comunicação.

2.2 Principais Conceitos de Codesign

Tradicionalmente, o desenvolvimento de sistemas em *hardware/software* de diversos tipos era realizado com pouca ou nenhuma interação entre os ambientes de projeto. As decisões de particionamento, responsáveis em determinar que partes do sistema seriam implementadas em determinado ambiente, eram fixas e tomadas no início do ciclo de desenvolvimento. Os projetos de *hardware* e *software* eram encaminhados separadamente e permaneciam independentes até o final, quando ocorria a integração do sistema.

Geralmente, o *hardware* a ser sintetizado era especificado sem nenhum conhecimento das necessidades computacionais do projeto de *software*, como por exemplo, a velocidade de operação do processador e sua capacidade de memória. O desenvolvimento do projeto de *software* também não influenciava o projeto de *hardware* e, o mais grave, não era capaz de inserir possíveis mudanças durante a fase de projeto. Somente durante a fase final de desenvolvimento, os dois ambientes eram integrados e testados como um todo.

O uso de tal metodologia acarreta algumas conseqüências desagradáveis. Como a integração dos sistemas só ocorre no último estágio do processo, a detecção de possíveis problemas pode exigir modificações no projeto de *hardware*, no projeto de *software* ou em ambos. A grande maioria dos ASIC's funcionam corretamente de acordo com suas especificações lógicas, porém 50% falham quando interagem com os demais componentes do sistema [24]. Estes problemas se resumem a erros na especificação do circuito, problemas de comunicação ou defeitos elétricos relacionados

à construção da placa. A propagação de erros na descrição a etapas posteriores conduz a um aumento substancial no tempo de projeto. O retorno a etapas anteriores, necessário à correção de erros, acarreta uma duplicação de esforços e geralmente deve ser realizado em um tempo mínimo, devido à proximidade dos prazos ocasionando novos problemas, e acarretando também um aumento significativo do custo do projeto.

O surgimento de certas tecnologias provocou mudanças na metodologia de projeto de sistemas em *hardware/software*. O aperfeiçoamento de ferramentas automáticas de projeto eletrônico, tais como ferramentas de síntese de alto nível, permite que sistemas possam ser implementados de um modo mais rápido. O desenvolvimento de circuitos integrados para aplicações específicas (ASIC) provê alto desempenho na implementação de um projeto. FPGAs, PLDs e processadores são componentes reprogramáveis e podem ser utilizados em uma rápida prototipagem do sistema, apesar de não apresentarem um desempenho tão elevado.

No domínio de *software*, o uso de Técnicas de Descrição Formal (TDF) facilita a modelagem e validação do sistema. A utlização de algoritmos complexos de particionamento e o emprego de modelos para a avaliação de diferentes alternativas de partição *hardware/software* vêm a facilitar a tarefa do projetista.

A utilização em conjunto de todos estes artifícios, tanto no domínio de *hardware* quanto no domínio de *software*, facilita o desenvolvimento de uma metodologia cooperativa mais flexível e mais unificada. Esta metodologia é denominada *Hardware/Software Codesign*. A interação dos caminhos de desenvolvimento de projetos de *hardware* e *software* é a idéia principal do *Codesign*. Esta interação entre os projetos, antes rotulados de distintos, ocorre de diferentes modos e em diferentes níveis do ciclo de desenvolvimento.

A experiência mostra que 80% dos problemas de integração podem ser evitados com o desenvolvimento conjunto do projeto de *hardware* e de *software* [24]. Desta forma, a especificação inicial antes de ser particionada entre as equipes é validada, de modo a se ter certeza da correta funcionalidade do sistema especificado. As equipes trabalham em paralelo e a cada etapa de projeto realizam a integração e o teste das

partes. Esta constante tarefa de testes consome um certo tempo no projeto, mas impede a propagação dos erros às etapas posteriores. A integração final é realizada sem maiores dificuldades e o tempo total do projeto diminui, uma vez que não é mais necessário o retorno às etapas anteriores do projeto.

A figura 2.1 ilustra uma abordagem de *Codesign* comumente empregada. O processo se inicia com uma representação do sistema, a nível funcional, independente de ambos os domínios de projeto, *hardware* e *software*. Algumas representações do sistema são realizadas utilizando-se Máquinas de Estado Finitas (MEF) e processos concorrentes. Em alguns ambientes de pesquisa, o sistema é descrito usando-se uma linguagem de programação, que é então compilada numa representação interna, tais como descrições de fluxo de dados e de controle.

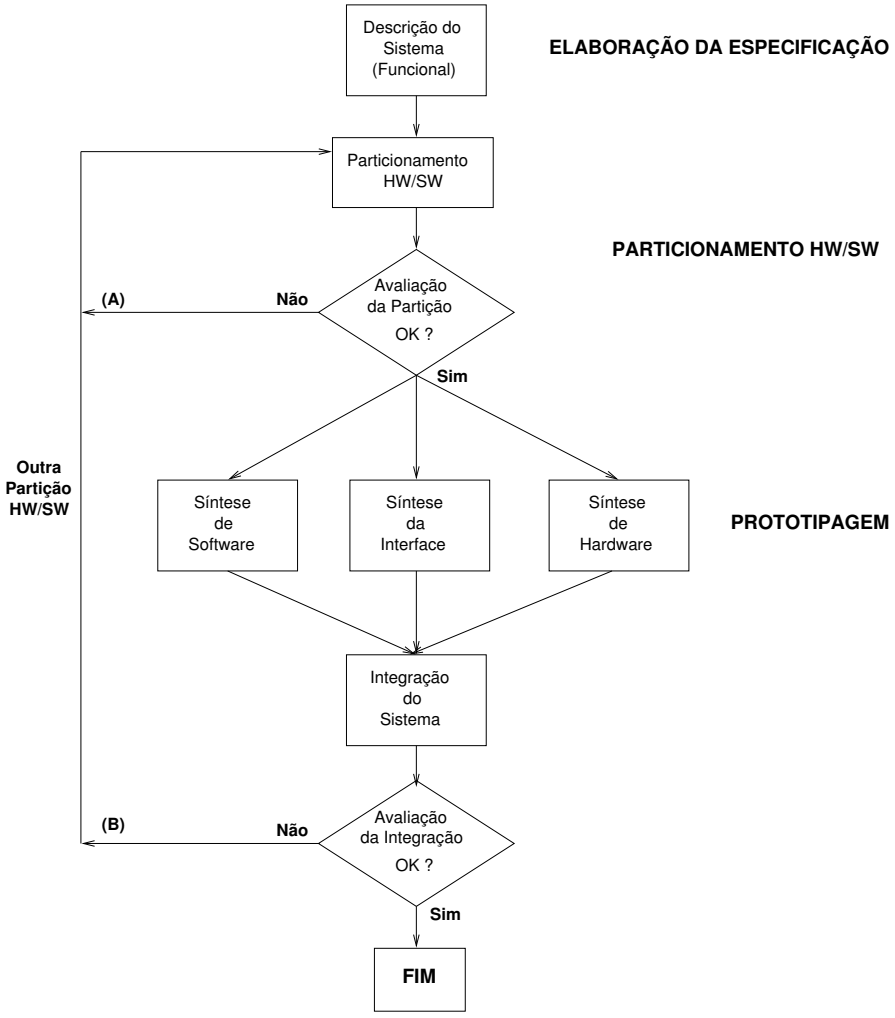

Figura 2.1: Abordagem hardware/software codesign.

O processo de particionamento determina quais funções serão implementadas em *hardware* e quais serão implementadas em *software*. Este processo é realizado numa descrição interna ou na própria descrição do sistema, podendo ser manual ou automático. A qualidade da decisão de particionamento pode ser avaliada em dois diferentes pontos, designados por (a) e (b) na figura 2.1.

Após o processo de particionamento, ocorre a etapa de prototipagem, que consiste em se efetuar a síntese do *hardware*, a síntese do *software* e a síntese da comunicação entre os dois domínios. Por fim, ocorre a integração do sistema, ou seja, a integração dos componentes implementados em cada domínio específico. Nesta fase, uma nova avaliação do projeto é executada de modo a validar o projeto como um todo.

Diversos grupos de pesquisa adotam, basicamente, a metodologia apresentada na figura 2.1 como solução para a implementação *hardware/software* de diversos sistemas. A próxima seção mostra detalhadamente as principais etapas do ciclo de desenvolvimento de um projeto *Codesign*.

2.3 Etapas de Desenvolvimento

O processo de concepção conjunta se adapta perfeitamente às etapas do ciclo de desenvolvimento de um sistema eletrônico. Este processo é formado por um conjunto de tarefas que conduzem à produção de um primeiro protótipo físico, a partir de uma descrição inicial do sistema. A metodologia de *Codesign* caracteriza-se normalmente por três etapas básicas: representação do sistema, particionamento *HW/SW* e protipagem.

2.3.1 Representação do Sistema

A primeira etapa no projeto de sistemas eletrônicos é a especificação da sua funcionalidade. A descrição do comportamento, ou funcionalidade, do sistema pode ser melhor entendida através da utilização de *Modelos*. Um modelo representa um conjunto de objetos e a interação entre eles. O modelo de uma máquina de estados, por exemplo, abrange um conjunto de estados e transições entre estados. O modelo algorítmico, em contrapartida, consiste em um conjunto de parâmetros que são executados segundo uma seqüência de controle. O objetivo principal é como descrever a funcionalidade de um sistema em um nível de detalhe suficiente que permita prever o seu comportamento completo.

Diferentes modelos são requisitados em diferentes domínios. Por exemplo, projetistas modelariam sistemas de tempo real e sistemas de banco de dados de maneiras diferentes. O primeiro seria modelado através de um comportamento temporal, enquanto que o segundo focalizaria a organização de dados. Uma vez que o projetista definiu o modelo apropriado para especificar a funcionalidade do sistema, torna-se possível descrever em detalhes exatamente como o sistema irá trabalhar. Neste ponto, o processo do projeto do sistema ainda não está completo, devido ao fato do modelo não descrever exatamente como o sistema será implementado.

A próxima etapa é a implementação da funcionalidade do sistema descrito através do modelo escolhido utilizando um conjunto de componentes físicos. O detalhamento de como o modelo será implementado, especificando o número e o tipo dos componentes bem como as interconexões entre eles, é melhor visualizado através da definição de uma *Arquitetura*. O processo de projeto de um sistema *hardware/software* pode ser definido como um conjunto de tarefas que transformam um modelo em uma arquitetura.

No início do processo, somente a funcionalidade do sistema é conhecida. Com o avanço do projeto, uma arquitetura iniciará a emergir, com detalhes sendo adicionados a cada passo do processo. A tecnologia de implementação e o domínio do projeto possuem grande influência na escolha da arquitetura. Os projetistas devem

considerar várias alternativas diferentes de implementação antes do processo de projeto ser finalizado. Um resumo dos principais modelos e das principais arquiteturas utilizadas nas diversas metodologias de projeto é apresentado a seguir.

Modelos

Os projetistas de sistemas utilizam vários modelos diferentes nas várias metodologias de projeto *hardware/software*. Cada modelo é mais apropriado para descrever um conjunto de aplicações específicas. Em geral, os modelos podem ser classificados em cinco categorias distintas.

Um **modelo orientado a estado** representa o sistema como sendo um conjunto de estados e transições que são ativadas por eventos externos. Este modelo é mais apropriado para descrever sistemas de controle, onde o comportamento temporal do sistema é o aspecto mais importante do projeto. Máquinas de Estados Finitos (MEF), Redes de Petri e Máquinas de Estados Finitos Concorrentes e Hierárquicos (MEFCH) são exemplos deste modelo.

Um **modelo orientado à atividade** representa um sistema como um conjunto de atividades descritas através de dados ou dependências de execução. Este modelo é mais apropriado para descrever sistemas transformacionais, onde os dados sofrem um conjunto de transformações. Grafo de Fluxo de Dados (GFD) e Grafo de Fluxo de Controle (GFC) são exemplos deste modelo.

Um sistema descrito através de módulos físicos e interconexões utiliza um **modelo orientado à estrutura**. Um diagrama em blocos é um exemplo deste modelo. Estes modelos se concentram principalmente na composição física do sistema. Diagramas de Conectividade de Componentes (DCC) é um exemplo deste modelo.

A representação de um sistema como um conjunto de dados relacionados através de seus atributos, classes, etc., é feita com auxílio de um **modelo orientado aos dados**. Este modelo, por exemplo um diagrama entidade-relacionamento, é mais apropriado pra descrever sistemas de informação, como banco de dados.

A utilização de um **modelo heterogêneo** integra várias características dos mo-

delos apresentados e pode ser útil para representação de visões diferentes de um sistema complexo. Grafo de Fluxo de Dados e Controle (GFDC) e Linguagens de Programação são exemplos deste modelo.

Arquiteturas

Diversos modelos capazes de descrever a funcionalidade de um sistema foram apresentados. A especificação de uma arquitetura visa completar a descrição do modelo, especificando como o modelo será realmente implementado. O objetivo de uma arquitetura é descrever o número de componentes, o tipo e a conexão entre eles. As diversas arquiteturas podem ser classificadas em três classes distintas: **arquiteturas específicas à aplicação**, **mono-processador** e **multi-processador**.

Nas **arquiteturas específicas à aplicação** o projetista inicia o projeto com uma aplicação, constrói um processador programável dedicado e traduz a aplicação para o código a ser executado neste processador. Neste esquema, o particionamento inclui o projeto do conjunto de instruções. A função custo está geralmente relacionada à área, velocidade de execução e consumo de potência.

No projeto de um **mono-processador**, a arquitetura alvo é um processador atuando como controlador mestre e um conjunto de aceleradores em *hardware* atuando como co-processadores. Neste esquema, dois tipos de particionamento tem sido desenvolvidos: particionamento orientado ao *software* e particionamento orientado ao *hardware*. Muito dos trabalhos publicados nesta área adotam este esquema. Estes geralmente utilizam uma função simples de custo relacionada à área e custo do processador para o *software* e desempenho para o *hardware*.

No projeto de um **multi-processador**, o *Codesign* consiste em mapear um conjunto de processos comunicantes em um conjunto de processadores interconectados. Este esquema engloba a decomposição comportamental, a alocação de processadores e a síntese da comunicação. Muitos dos métodos de particionamento existentes restringem a função custo a parâmetros como restrições de tempo real e custo.

Esta classificação é independente da tecnologia. Cada um dos modelos pode

ser realizado sobre um circuito integrado único, sobre uma placa ou várias placas. De fato, as novas famílias de FPGA permitem a integração de microprocessadores, RAM (*Random Acess Memory*) e DSP (*Digital Signal Processor*) sobre um mesmo sistema, formando um sistema em um só circuito. O algoritmo de particionamento e as etapas de síntese dependem da escolha do modelo de especificação do sistema e da arquitetura alvo.

Tradicionalmente, a descrição da funcionalidade de novos sistemas no nível conceitual faz uso de uma linguagem natural. Entretanto, os sistemas têm se tornado cada vez mais complexos, requisitando novas metodologias para sua conceitualização. Os projetistas necessitam descrever visões conceituais do sistema. Uma das alternativas usadas é o uso de especificações executáveis, que são capazes de descrever a funcionalidade do sistema em uma forma simulável e executável. Uma vez escolhido o melhor modelo para representar o sistema a ser projetado e escolhida a melhor arquitetura de implementação, diversos ambientes de projeto *Codesign* utilizam linguagens de programação para a especificação destes sistemas. Um breve comentário sobre linguagens de especificação e sobre representações internas é apresentado a seguir.

Linguagens de Especificação

Um sistema pode ser descrito em diferentes níveis de abstração, cada qual possuindo características particulares adaptadas aos domínios de aplicação. Por exemplo, para descrever um sistema no nível lógico, os projetistas podem especificar a estrutura do sistema com uma ferramenta de esquemáticos. Alternativamente, no nível de sistema, uma linguagem de descrição de *hardware* permite ao projetista descrever a funcionalidade de cada componente sem especificar detalhes estruturais. Em um nível conceitual, é possível descrever o conjunto de funcionalidades do sistema sem qualquer indicação dos componentes.

A grande vantagem da utilização de uma especificação executável é permitir ao projetista verificar a correção da funcionalidade através da simulação da especificação. A especificação inicial de um sistema em linguagem natural não permite a

verificação de sua funcionalidade até que o projeto esteja a um nível avançado o suficiente para se obter uma descrição simulável. Conseqüentemente, alguns problemas associados com a integração do sistema podem ser minimizados no início do ciclo de desenvolvimento.

Muitos grupos de pesquisa utilizam linguagens de especificação como ponto de partida num projeto de *Codesign*. Dentre as linguagens mais utilizadas estão: VHDL, Verilog, Statecharts, SDL (*Specification and Description Language*), SpecCharts e C.

Representação Interna

Diversos ambientes de pesquisa em *Codesign* iniciam suas especificações ao nível de sistema através da utilização de uma linguagem de programação [7, 6, 10]. Nestes ambientes essas especificações são então compiladas em uma representação interna. Esta representação interna é vista como um fator importante no desenvolvimento do projeto. Antes de se realizar o particionamento, é interessante ter uma descrição que possa ser implementada tanto em *hardware* quanto em *software*. Tal descrição permite que o sistema seja visto independentemente do domínio escolhido. Após o particionamento, uma representação unificada é importante nos estágios de refinamento e avaliação da decisão escolhida.

Representações baseadas em grafos, tais como grafos de fluxo de dados e redes de Petri, MEF e processos concorrentes são escolhas que podem ser utilizadas como representações internas. Grafos possuem a habilidade natural em modelar fluxos de dados e de controle e também concorrência entre processos. MEFs podem ser utilizadas para expressar o comportamento do sistema além de permitir uma verificação formal da funcionalidade do particionamento e da síntese.

2.3.2 Particionamento HW/SW

As abordagens de particionamento *hardware/software* podem ser classificadas de diferentes modos. Estes modos incluem o tipo de representação usada como ponto de

partida para o particionamento, o algoritmo utilizado neste processo, a quantidade de automação, a granularidade das funções que estão sendo consideradas e o estágio do processo de projeto no qual o particionamento é efetuado.

A especificação inicial utilizada como ponto de partida no processo de particionamento pode ser uma especificação em *hardware*, em *software* ou uma descrição mais geral independente do domínio. O algoritmo de particionamento determina que operações do sistema serão implementadas em *hardware* e que operações serão implementadas em *software*. O grau de automação utilizado no particionamento vai do manual ao automático. A granularidade dos objetos define o menor objeto funcional indivisível utilizado durante o particionamento, por exemplo, processos, subrotinas, *loops*, operações a nível aritmético ou expressões booleanas. Quanto maior a granularidade, menos objetos particionados e mais fácil o processo de particionamento. O processo de particionamento de um sistema termina quando as partições respeitam as restrições impostas pelo projetista e têm uma boa relação custo/desempenho. Alguns comentários sobre algoritmos de particionamento e sobre a avaliação das partições são apresentados a seguir.

Algoritmos de Particionamento

O processo de particionamento pode ser interativo ou automático. A **técnica interativa** disponibiliza ao usuário um ambiente do tipo caixa de ferramentas, contendo um conjunto de primitivas de transformação e particionamento. As primitivas permitem executar operações básicas sobre a descrição sem alterar a funcionalidade original. O critério de escolha das primitivas a aplicar é deixado aos cuidados do usuário. Esta técnica é muito utilizada devido ao fato de não haver métodos de estimação universais, ou seja, métodos que trabalhem sobre diferentes modelos, arquiteturas e domínios de aplicação. A escolha de critérios de partição e pesos desses critérios numa função custo geral pode variar dependendo do domínio de aplicação, assim como da tecnologia utilizada.

As técnicas de **particionamento automático** utilizam uma função custo para avaliar as partições. As abordagens automatizadas geralmente partem de uma des-

crição inicial e gradualmente migram para uma implementação ou em *hardware* ou em *software* que satisfaça aos objetivos especificados. Normalmente, os sistemas automáticos são destinados a um simples modelo de arquitetura e a um domínio de aplicação definido. Alguns algoritmos básicos utilizados são: *clustering*, *group migration*, *simulated annealing* e algoritmos genéticos. Os atuais ambientes de *Codesign* que utilizam a técnica de particionamento automático implementam um desses algoritmos com algumas mudanças para a adaptação ao domínio de aplicação destinado.

Avaliação da Partição

O objetivo principal do processo de particionamento é agrupar o conjunto de tarefas que serão implementadas em *hardware* e *software* independentemente. A etapa de avaliação da partição visa justamente examinar a qualidade das partições determinadas pelo algoritmo de particionamento e, conseqüentemente, comparar diversas alternativas de implementação. De modo a facilitar a avaliação das partições, são utilizadas algumas métricas ou parâmetros combinados numa função de custo única, denominada *função objetivo*.

O principal problema a ser resolvido consiste em encontrar um compromisso entre os diversos parâmetros escolhidos para se avaliar uma determinada partição, tais como, a área e desempenho para cada partição. Neste ponto, muitos grupos de pesquisa utilizam resultados de estimações tanto de *hardware* quanto de *software*, de modo a se avaliar melhor as diferentes alternativas de implementação. Em contrapartida, alguns grupos utilizam ferramentas de síntese para se obter métricas de custo na implementação em *hardware* e compiladores para as partes de *software*. A utilização de estimadores diminui o tempo de projeto, mas em compensação, os valores obtidos pelas ferramentas são de melhor qualidade quando comparadas aos resultados dos estimadores.

Tanto as partições quanto a arquitetura alvo são refinadas nesta etapa. Normalmente, as partições que fazem parte do caminho crítico da aplicação, e que necessitam de um tempo curto de resposta, são realizadas em *hardware* e as demais

partições realizadas em *software* com intuito de reduzir o custo da implementação.

2.3.3 Prototipagem

A prototipagem é a etapa que gera o código executável para cada domínio de projeto. As descrições produzidas são simuláveis e podem servir à síntese. Cada sub-sistema é traduzido separadamente, e ao final tem-se uma arquitetura heterogênea representada pelo código na linguagem C para *software* e pelo código na linguagem VHDL para *hardware*. Os códigos correspondentes aos processos de comunicação são normalmente retirados de uma biblioteca contendo protocolos de realização de comunicação.

O protótipo gerado pode ser simulado, tendo como entrada os mesmos estímulos utilizados durante a verificação da descrição de entrada. Esta é uma forma de verificar, por simulação, se as etapas de síntese ao nível sistema não provocaram mudanças no comportamento inicial. E ainda mais, estas descrições executáveis (C e VHDL) permitem a estimação mais precisa de certas características de desempenho do sistema. Comentários a respeito da prototipagem do *hardware*, do *software* e da interface de comunicação são apresentados a seguir.

Síntese do Hardware

Uma vez definidas, pelo algoritmo de particionamento, quais partes do sistema serão implementadas em *hardware* ocorre a implementação destes sub-sistemas em circuito dedicado (ASIC) ou lógica programável (FPGA ou PLD). Esta etapa é denominada de síntese de *hardware* e incorpora outras técnicas como Síntese de Alto Nível (*High-Level Synthesis*), Síntese da Máquina de Estados (*Finite State Machines Synthesis*), Síntese Lógica e mapeamento na tecnologia adotada.

A utilização de ASICs visa obter circuitos com alto desempenho, porém mais custosos. O uso de FPGAs ou PLDs acarreta na obtenção de circuitos com menor desempenho, em compensação, sua implementação é imediata e existe a possibilidade de se reprogramar um sistema sem necessidade de uma nova rodada de fabricação,

como no caso de ASICs.

Síntese do Software

As partes do sistema definidas pelo algoritmo de particionamento para serem implementadas em *software* são realizadas nesta etapa. Normalmente, uma descrição a nível de sistema utiliza características mais complexas do que as encontradas em linguagens de programação tradicionais. A etapa de síntese de *software* é encarregada de converter um descrição complexa, baseada geralmente em MEF ou grafos, em um programa numa linguagem de alto nível, como por exemplo, a linguagem C.

Nesta etapa são definidos os processos concorrentes e realiza-se um escalonamento, ou seja, define-se quais processadores executam cada um dos processos. Deve-se considerar aspectos como minimização do tempo que o processador espera por eventos externos, restrições de tempo de execução para cada processo, seqüência de execução de cada processo e garantia de que todos poderão ser executados pelo menos uma vez.

Síntese da Comunicação

A síntese da comunicação é a etapa do processo de *Codesign* responsável pela modelagem e implementação do esquema de comunicação utilizado pela linguagem de especificação. Esta etapa implementa a troca de informações entre os diversos módulos da descrição, ou seja, a comunicação entre os processadores da arquitetura. O modelo de comunicação deve ser geral o suficiente para acomodar diferentes esquemas de comunicação, como, passagem de mensagens, variáveis compartilhadas, filas FIFO (*First In First Out*), etc.

Normalmente, a síntese da comunicação é dividida em seleção do protocolo e síntese da interface. A **seleção do protocolo** consiste em selecionar dentre os canais existentes em uma biblioteca aquele que possa executar a comunicação entre as unidades existentes. Esta comunicação pode realizar-se na forma de protocolos contendo funções de resolução de conflitos de acessos ou na forma de sinais de troca

diretamente entre as unidades. Com o canal definido, a etapa seguinte, de **síntese da interface**, tem como objetivo distribuir os elementos necessários à realização do canal no conjunto do sistema. Os serviços que realizarão a comunicação são ligados às unidades, o controlador de comunicação aparece explicitamente na descrição e as linhas de sinais são criadas.

2.4 Características dos Protocolos de Comunicação

Uma rede de computadores pode ser classificada como um conjunto de camadas hierárquicas, cada uma sendo construída utilizando as funções e serviços oferecidos pelas camadas inferiores. Cada camada (ou nível) deve ser pensada como um programa ou processo, implementado por *hardware* ou *software*, que se comunica com o processo correspondente na outra máquina. As regras e convenções usadas nesta conversação são chamadas de protocolos [25].

O desempenho de uma rede está fortemente relacionado com o protocolo utilizado e com o tempo de processamento, em cada nó, dos pacotes que transitam pela rede. Esta situação se torna mais crítica em redes de alta velocidade. Dobrar a velocidade da rede, porém, não tem qualquer efeito prático, pois o gargalo está nos *hosts* [26]. Portanto, no projeto de um protocolo, devem ser considerados alguns aspectos que influenciam diretamente o seu desempenho e, conseqüentemente, o desempenho da rede como um todo.

Nesta seção são apresentados os mecanismos normalmente utilizados em diferentes protocolos, tais como, sinalização, estabelecimento e término de conexão, informações de controle, formato de pacotes, confirmação de recepção de dados, controle de fluxo e tratamento de erros. Paralelamente à apresentação desses mecanismos, será descrita a adequação desses mesmos mecanismos aos protocolos de altas velocidades.

2.4.1 Sinalização

A sinalização é a troca de informações entre os pares de entidades com o propósito de gerência de uma conexão. Ela é utilizada para estabelecer e terminar as conexões e para trocar parâmetros do sistema de comunicação. A sinalização pode ser "in-band" (informações de controle e dados são multiplexados na mesma associação) ou "out-of-band" (informações de controle e dados são transmitidas em associações diferentes).

Em um sistema de comunicação de alto desempenho é mais interessante a utilização do esquema "out-of-band" devido à redução do processamento de dados, pois as entidades não necessitam analisar todos os pacotes para identificar quando está presente a informação de sinalização [27].

2.4.2 Estabelecimento e Término de Conexão

As fases de estabelecimento e término de conexão podem ser atendidas através do esquema implícito (conexão estabelecida com a chegada da primeira PDU) ou do esquema explícito ("hand-shake", troca de mensagens). Em um sistema de comunicação de alto desempenho é mais interessante a utilização do esquema de "hand-shake" quando a sinalização é do tipo "out-of-band" e o esquema implícito quando a sinalização é do tipo "in-band" [27].

2.4.3 Informações de Controle

As informações de controle são necessárias para sincronizar o estado do transmissor com o do receptor como, por exemplo: para o suporte das funções de gerência de conexão, confirmação de unidade de dados, controle de fluxo e tratamento de erros.

A fim de obter um alto desempenho na comunicação, todas as informações de estado relevantes devem ser incluídas em uma mesma mensagem de sinalização e armazenadas em pacotes separados dos pacotes de dados. Essas informações devem ser trocadas periodicamente, independente de outros eventos relacionados com a

conexão.

2.4.4 Formato de Pacotes

O tamanho e a ordem do pacote tem um impacto significativo na velocidade de construção e análise destes pacotes. Os campos de controle de tamanhos fixos são preferíveis aos de tamanhos variáveis, pois estes últimos exigem maior processamento. As informações utilizadas para identificar os pacotes são melhor localizadas no cabeçalho facilitando a decodificação. Já o *checksum* deve ser inserido após os dados, permitindo assim que ele possa ser calculado em paralelo com a transmissão e recepção de dados. Este esquema é basicamente o adotado para a implementação de protocolos de alta velocidade, em particular, para implementação das Redes Digitais de Serviços Integrados (RDSI-BL) através do uso do modo de transferência assíncrono (ATM - *Asynchronous Transfer Mode*) [28].

2.4.5 Confirmação de Recepção de Dados

A confirmação é a indicação de sucesso na recepção de dados pelo usuário. Quando um nó recebe uma confirmação, ele atualiza as informações de estado e descarta os pacotes de dados retidos para possíveis retransmissões. O esquema considerado ideal para subsistemas de alto desempenho consiste em se gerar confirmações assincronamente pelo receptor, isto é, independente das ações geradas pelo transmissor.

2.4.6 Controle de Fluxo

O controle de fluxo assume uma enorme importância em redes de alta velocidade, pois a maior fonte de pacotes perdidos nestas redes é a sobrecarga de um nó ocasionado pelo congestionamento de determinado enlace. A taxa de transmissão de dados deve ser limitada pela taxa de dados suportada pela rede e pela taxa com que a entidade receptora possa receber, processar e enviar os dados para seu usuário.

Como nas atuais redes de alta velocidade a taxa de dados suportada pela rede é elevada, o ponto crítico passou a ser a taxa de processamento dos protocolos.

O fluxo de dados do transmissor pode ser controlado através de dois métodos: método da janela (receptor especifica a quantidade máxima de dados (janela) que o transmissor pode enviar) e método do controle de taxa (uso de temporizadores). Nas redes baseadas em ATM, o esquema de controle de taxa é o mais apropriado para o mecanismo de controle [27]. O transmissor deve conhecer a taxa na qual os dados podem ser transmitidos e o tamanho da rajada, de forma a especificar a quantidade máxima de dados que podem ser enviados em uma rajada de pacotes.

2.4.7 Tratamento de Erros

Um protocolo deve ser capaz de detectar erros, reportá-los e corrigi-los. A detecção de erros é executada através do número de seqüência, do tamanho e do *checksum*. O número de seqüência é utilizado para detectar dados perdidos, a entrega fora de ordem e a proteção contra dados duplicados. O campo relativo ao tamanho dos dados é encarregado de verificar a entrega completa dos dados. O *checksum* protege contra a modificação de dados dentro dos nós de roteamento.

A notificação de erros é usada para explicitamente informar ao transmissor sobre os erros detectados pelo receptor. Isto pode ser feito através de uma confirmação negativa (Nack), identificando o ponto no qual os dados são perdidos ou através de um *reject* seletivo, indicando todos os dados perdidos pelo receptor.

2.5 Características Relevantes para um Processador de Protocolos em Hardware

Nos parágrafos anteriores foram apresentados mecanismos utilizados em diferentes protocolos e como adequar estes mecanismos aos protocolos de alta velocidade. Mas, nem sempre, a otimização em *software* permite um ganho de desempenho

compatível com as necessidades das atuais redes de Gigabits. Implementar determinadas tarefas destes protocolos em *hardware* pode levar a um considerável aumento de desempenho.

Virtanen [29] estuda as características de alguns protocolos utilizados atualmente com intuito de conceber um processador de protocolo geral. Este processador consiste de um microprocessador programável com uma arquitetura previamente definida através da análise destas características relevantes. Este trabalho conclui que devam existir determinados blocos funcionais específicos na arquitetura deste processador, de modo a torná-lo o mais geral possível.

Um bloco encarregado do cálculo do *checksum* é imprescindível na arquitetura proposta. De certo, muitos dos protocolos existentes utilizam tal expediente para assegurar que os dados chegarão de modo correto ao destino. Uma memória extremamente rápida é necessária com intuito de armazenar os dados que estão chegando de modo a evitar congestionamento e possíveis descartes de dados. Um barramento de dados veloz também é necessário, uma vez que dados devem ser movidos rapidamente de uma parte da memória para outra ou para outra camada ou para a saída do sistema. A análise dos cabeçalhos dos pacotes deve ser efetuada rapidamente de modo a não atrasar o processamento dos dados. Esta análise pode ser efetuada comparando-se os campos dos cabeçalhos com possíveis valores na memória e uma decisão deve então ser tomada.

Pode-se concluir que existem certas características que estão presentes em vários protocolos, talvez com pequenas diferenças. Virtanen propõe um método geral de implementação em *hardware* baseado na análise destas características. O trabalho aqui apresentado aborda este conceito de uma maneira diferente. A metodologia aqui proposta visa identificar quais pontos do protocolo são críticos em termos de velocidade de processamento. Estas funções devem ser encaminhadas para implementação em *hardware*, enquanto que o restante do protocolo pode ser implementado em *software*. Deste modo, leva-se em consideração somente as funcionalidades identificadas como "gargalo" do processamento e, por conseguinte, a variável custo também é levada em consideração.

Outro ponto a ser ressaltado, diz respeito a característica do processador proposto por Virtanen. Obviamente que, para um protótipo comercial, é mais interessante que este processador seja o mais "geral" possível, podendo, assim, ser utilizado no projeto de quase todos os protocolos atuais. Na metodologia aqui desenvolvida, os projetos são únicos e específicos para cada protocolo em particular. Em contrapartida, esses resultados são otimizados em relação às características e peculiaridades de cada projeto.

2.6 Comentários

A utilização da metodologia de *Codesign* no projeto de protocolos de comunicação é a idéia chave deste trabalho. Neste capítulo foram apresentados os conceitos básicos desta metodologia, suas principais características, bem como suas vantagens e desvantagens em relação às abordagens tradicionais. Foi apresentado um detalhamento das principais etapas de desenvolvimento de um projeto utilizando a técnica de *Codesign*. A escolha de uma descrição inicial adequada ao domínio de aplicação, bem como um resumo sobre os diversos modelos, arquiteturas, linguagens de especificação e representações internas que podem descrever estes sistemas foi comentada. Foi ressaltada a importância da correta escolha de uma partição *hardware/software* e, apresentados comentários sobre algoritmos de particionamento e sobre a avaliação das partições escolhidas. A etapa de prototipagem foi apresentada ressaltando a importância de diversos aspectos quando da síntese de *hardware*, *software* e da interface entre os domínios. As principais características de diversos protocolos de comunicação também foram apresentadas e comentadas.

O próximo capítulo apresenta um resumo dos principais ambientes de projeto *Codesign* desenvolvidos por grupos de pesquisa em todo mundo. Outros trabalhos relacionados com o tema também são apresentados. O objetivo principal do capítulo é discutir as principais características desses ambientes e ressaltar as diferenças existentes.

Capítulo 3

Principais Ambientes de Codesign

3.1 Trabalhos Relacionados

NOS últimos anos, diversos grupos de pesquisa em renomadas universidades desenvolveram ambientes de HW/SW Codesign. Estes sistemas são baseados em diferentes metodologias de projeto e concentram esforços em diferentes aspectos do problema de Codesign. SpecSyn [7], Cosyma [30] e Vulcan II [31] são ferramentas de Codesign que realizam particionamento funcional de uma dada especificação em componentes de HW e SW, tais como ASIC's (*Application Specific Integrated Circuits*) e processadores. Estes ambientes diferem no nível de granularidade do processo de particionamento (processos versus blocos de declarações) e diferem também na sua abordagem (particionamento iniciado com tudo em SW versus iniciado com tudo em HW).

Outros sistemas, tais como Ptolemy [9] e Chinook [11] abordam diferentes aspectos do problema. Ptolemy provê um ambiente para especificação, simulação e prototipagem de aplicações DSP (*Digital Signal Processing*) e se concentra na cosimulação HW/SW e na geração de códigos ao invés de se preocupar com particionamento e estimação. Chinook se detém no processo de síntese de interfaces de HW e SW e, como Ptolemy, exige que o usuário especifique manualmente a partição HW/SW.

O ambiente LYCOS [10] utiliza VHDL ou C como linguagem de especificação de entrada para o projeto de seus sistemas. Em contrapartida, COSMOS [6] possui a vantagem de permitir a construção da especificação inicial utilizando diversas linguagens, tais como SDL, LOTOS, Estelle, StateCharts e CSP. Ambos os sistemas, porém, convertem suas especificações de entrada em um formato intermediário que será utilizado pelas diferentes ferramentas do projeto. O projeto PISH [15] é outro que utiliza um formato intermediário que serve como entrada para o algoritmo de particionamento. O modelo adotado como representação interna neste ambiente é o modelo de Rede de Petri.

Fischer [32] ressalta a importância do uso combinado do HW e do SW para se alcançar um alto desempenho de sistemas distribuídos, como sistemas multimídias. É apresentado um ambiente de desenvolvimento para o suporte das etapas de projeto e implementação. Porém, quanto ao desenvolvimento da parte destinada ao HW, a única etapa suportada é a simulação. A etapa de síntese de HW não é abordada, ficando para análise posterior.

Hidalgo [33] propõe uma metodologia para a partição HW/SW baseada no uso de algoritmos genéticos. A divisão dos blocos funcionais em HW e SW é baseada na avaliação de uma função objetivo. Esta função utiliza uma tabela previamente estabelecida para os valores dos desempenhos, não calculando esses parâmetros durante o processo de busca da melhor partição, como é realizado neste trabalho.

Diversos ambientes fazem uso de estimadores tanto de HW quanto de SW com o objetivo de calcular atrasos e custos de diversas implementações sem a necessidade de sintetizá-las. A utilização de estimadores diminui o tempo de projeto. Somente após o processo de particionamento, onde efetivamente, são designadas que partes serão implementadas em HW e SW, há a síntese dessas partes. Em contrapartida, o uso de estimadores pode levar a resultados não tão precisos em termos de atraso e custo.

Existem diversos ambientes de *Codesign* que abordam diferentes aspectos do projeto. Estes ambientes podem diferir na linguagem de especificação, no método de particionamento, na arquitetura alvo e nos métodos de validação utilizados. Podem

diferir também em relação a classe de projetos suportada, tais como DSP, sistemas em tempo real, automotivos ou de telecomunicações e outros.

O presente trabalho concentra-se em propor uma metodologia visando o projeto de protocolos de comunicação adotando para isso a técnica de Codesign. O estabelecimento de um critério objetivo que auxilie o projetista na escolha de uma determinada partição HW/SW que atenda aos requisitos impostos e a obtenção dos parâmetros de custo e atraso de implementações em HW, durante o ciclo de projeto, consistem nos objetivos desta metodologia. Para tal, esta metodologia utiliza, como representação inicial, um modelo de máquina de estados finitos. O método de particionamento é automático e baseado na otimização de desempenho do sistema. A etapa de síntese de HW é efetuada, em contrapartida ao uso de estimadores de HW, com a finalidade de se obter medidas mais precisas.

3.2 Características dos Principais Ambientes de Codesign

Muitos grupos de pesquisa em renomadas universidades estão desenvolvendo ambientes de projeto baseados na metodologia de *Codesign*, diferindo basicamente na linguagem de especificação utilizada, no método de particionamento, na arquitetura alvo e nos métodos de validação utilizados. A tabela 3.1 ilustra as principais diferenças entre alguns grupos de pesquisa ressaltando as características mais importantes de cada um. As seções seguintes abordam de uma forma mais detalhada os tópicos principais num projeto *Codesign* de alguns renomados grupos de pesquisa e também apresenta alguns trabalhos relacionados com o tema.

3.2.1 COSMOS

COSMOS consiste de um ambiente de concepção conjunta de *hardware* e *software* desenvolvido no Laboratório TIMA em Grenoble-França [6]. A primeira etapa de projeto consiste na construção na especificação executável utilizando diversas lin-

	Linguagem de especificação	**Método de particionamento**	**Arquitetura alvo**
COSMOS	Multi-formalismo	Semi-automático	Multi-processador
SpecSyn	SpecCharts ou VHDL	Manual	Multi-processador
Ptolomey	FDS	Manual	Multi-processador
LYCOS	VHDL ou C	Manual	Mono-processador
BEKKA	Multi-formalismo	Manual ou Automático	Multi-processador
PISH	CSP e Lotos	Automático	Multi-processador

Tabela 3.1: Comparação de Ambientes de Codesign.

guagens, como, SDL, Estelle, LOTOS, StateCharts, Esterel, CSP e OCCAM. Este ambiente permite a descrição do sistema a partir de diversas linguagens, pois utiliza um formato intermediário denominado SOLAR [34], o qual é utilizado pelas diferentes ferramentas de projeto.

Este ambiente utiliza uma técnica semi-automática de particionamento, onde partindo de uma descrição funcional, obtém-se um conjunto de funções a serem realizadas em *software* e um conjunto de funções a serem realizadas em *hardware*. A parte de *hardware* pode utilizar componentes programáveis FPGAs ou circuitos dedicados ASICs. Por outro lado, o *software* utiliza processadores de uso geral.

O número de partições geradas representa o número de processadores da arquitetura alvo. Fica a cargo do usuário encontrar um bom compromisso entre as restrições de custo e desempenho. O usuário testa diferentes alternativas sempre guiado por resultados de estimações. Estas estimações são obtidas utilizando-se a técnica de *profiling*, onde o desempenho e a área são calculados para o conjunto de funções implementadas em processadores da arquitetura.

3.2.2 SpecSyn

SpecSyn é um ambiente de projeto a nível de sistema desenvolvido na Univ. de Irvine. Este ambiente basea-se no paradigma especificação-exploração-refinamento (*specify-explore-refine (SER)*) [1, 7] que permite obter descrições sintetizáveis a partir de especificações de sistemas escritas nas linguagens SpecCharts ou VHDL. O projeto se inicia com a especificação da funcionalidade do sistema e suas restrições. Estas especificações são convertidas numa representação interna similar a um grafo de chamadas denominado SLIF (*Specification-Level Intermediate Format*). Esta representação serve como base para operação de outras ferramentas do ambiente.

O próximo passo consiste no processo de exploração arquitetural. Exploração arquitetural é definida como a tarefa de se encontrar um conjunto de arquiteturas potenciais que satisfaçam as restrições exigidas. Este passo é dividido nas seguintes tarefas: alocação, particionamento, transformação e estimativa.

Alocação é a tarefa de se adicionar componentes ao projeto. Este passo é efetuado pelo Alocador SpecSyn que se utiliza de um número de componentes padrão armazenados numa biblioteca e caracterizados por um conjunto de restrições. O processo de particionamento é realizado manualmente e o projetista pode realocar objetos controlando assim o peso de várias métricas numa função custo total. A estimativa da qualidade do projeto é necessária para se determinar se um particular projeto particionado satisfaz as restrições especificadas. Transformações são possíveis de modo a se obter projetos particionados mais acurados. A parte de refinamento consiste na geração de uma nova especificação para cada componente do sistema após a etapa de exploração. O processo se encerra com a implementação dos componentes. As partes em *software* necessitam de compilação e as partes em *hardware* exigem síntese comportamental e RTL.

3.2.3 LYCOS

LYCOS [10] é um sistema de *hardware/software codesign* desenvolvido junto à Universidade da Dinamarca. O projeto se inicia a partir da especificação do sistema

em VHDL ou C. Esta especificação é traduzida para grafo de fluxo de controle e de dados. A operação de *profiling* é realizada tendo este modelo como base, ou seja, a descrição é analisada e informações ligadas à freqüência de execução de cada módulo são obtidas. Os módulos que são executados com maior freqüência têm preferência na implementação em *hardware*.

Antes da operação de particionamento, o projetista seleciona a arquitetura alvo. A arquitetura alvo é composta por um simples processador para o *software* e um simples componente de *hardware* (ASIC, FPGA, etc.) conectados por um canal de comunicação. Mesmo sendo limitado, este tipo de arquitetura é útil em muitas áreas, como processamento de sinais digitais, sistemas integrados e etc. Esta é a arquitetura normalmente encontrada nos sistemas de *hardware/software* automáticos.

O processo de particionamento é realizado tendo como objetivo a redução do tempo de execução do sistema (desempenho) e considerando restrições de área para o *hardware* (custo). Este processo é manual e depende da experiência do projetista. A decisão de onde alocar um bloco em particular é baseada em resultados de estimações de *hardware* e de *software*. Estas estimações podem dar-se no domínio físico, realizando as implementações e verificando os resultados, ou no domínio de modelo, menos preciso mas com cálculos bem mais rápidos. Na exploração de várias possíveis soluções, devem ser repetidas inúmeras vezes as operações de particionamento e estimação. Desta forma, a velocidade do processo de estimação é crítica, necessitando de técnicas e heurísticas eficientes.

3.2.4 AKKA e BEKKA

O Laboratório de Projeto de Sistemas Eletrônicos (*Electronic System Design Laboratory (ESDlab)*) do Instituo Real de Tecnologia de Estocolmo (*Royal Institute of Technology in Stockholm*) na Suécia desenvolveu um sistema de *HW/SW Codesign* denominado AKKA [13].

Este ambiente aceita especificações de sistemas escritos em C/C++. O componente principal nesta estrutura é o GNU C Compiler. Este compilador possui dois

propósitos principais: análise de desempenho e geração de código.

A análise de desempenho realizada pelo ambiente AKKA se baseia na técnica de *profiling*. AKKA realiza dois tipos de *profiling*: o *execution profiling*, que determina os pontos de gargalo na execução do programa e o *data transfer profiling*, que determina os pontos de gargalo na fase de transferência de mensagens. A execução destas tarefas resulta numa coletânea de informações armazenada numa base de dados comum.

A descrição inicial do sistema é também analisada por um estimador de *hardware*. Um estimador baseado em rede neural, chamado *Nestimator*, é utilizado e gera como saída o número de ciclos de *clock* e o número de portas lógicas necessárias para se implementar a descrição. Estas informações são armazenadas em conjunto com os dados da análise de desempenho na base de dados comum. Estes passos podem ser melhor visualizados na figura 3.1.

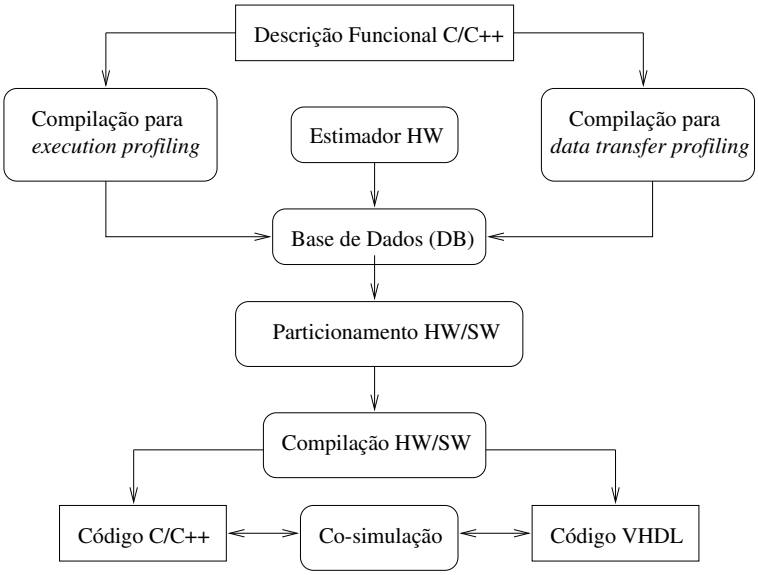

Figura 3.1: Fluxo de projeto no ambiente AKKA.

Embora o particionamento do sistema possa ser feito manualmente com auxílio de uma ferramenta gráfica, AKKA também permite que o particionamento seja totalmente automático. Este ambiente inclui um algoritmo de particionamento baseado no algoritmo *Knapsack stuffing*. O algoritmo utiliza como parâmetros o tempo de execução do *software*, a estimativa do *hardware*, os parâmetros da arquitetura

alvo e as informações da base de dados comum para encontrar uma partição que melhor atenda a restrição custo/desempenho.

As partes escolhidas para implementação em *hardware* são implementadas em VHDL e o *software* e o *hardware* gerados podem ser co-simulados antes e depois da etapa de síntese, aqui denominados, co-simulação comportamental e co-simulação a nível RTL (*Register Transfer Level*).

A característica principal dos sistemas de telecomunicações é que seu comportamento pode oscilar entre partes de controle e de comunicação. Isto leva a uma divisão natural na partição do projeto em termos de partes de controle e partes de dados, onde cada parte tem seu próprio fluxo de projeto independente.

Tendo em vista estes sistemas, este laboratório desenvolveu um ambiente sucessor do AKKA. Um ambiente de projeto heterogêneo denominado BEKKA [14]. Este ambiente aceita especificações a nível de sistemas utilizando-se as linguagens MATLAB para modelagem de aplicações dominadas por fluxo de dados e SDL para modelagem do fluxo de controle. Especificações em C++ e VHDL também são aceitas.

Existem dois tipos de particionamento: o particionamento a nível de sistema e o particionamento de *HW/SW*. O particionamento a nível de sistema divide a funcionalidade da especificação em partes de dados e de controle. Esta etapa é implementada manualmente. Uma vez que o sistema tenha sido dividido em partes de dados e de controle, ocorre o particionamento *HW/SW* em cada um dos fluxos de projeto. Esta etapa pode ser implementada automaticamente ou manualmente, dependendo do projetista. A localização de gargalos na computação e na comunicação dos sub-sistemas através da utilização da análise de desempenho é efetuada através da técnica de *profiling*.

3.2.5 PISH

A Universidade Federal de Pernambuco (UFPE) em conjunto com a Univ. do Rio Grande do Sul (UFGRS) e a faculdade PUC-RS desenvolvem um sistema cujo obje-

tivo é suportar o projeto de sistemas digitais complexos considerando arquiteturas heterogêneas compostas por componentes programáveis de propósito geral (denominados no projeto componentes de *software*) e componentes de aplicação específica (ou componentes de *hardware*).

Este ambiente, denominado de PISH (Projeto Integrado Software-Hardware), utiliza as linguagens CSP e LOTOS para a descrição de sistemas. Essas descrições são traduzidas para um modelo de representação interna que servirá de entrada para o algoritmo de particionamento. O modelo utilizado como representação interna é o modelo de Rede de Petri. Esse modelo é usado para análises quantitativas e qualitativas que permitem associar um conjunto de implementações possíveis para cada processo. Nesta fase, alguns métodos são aplicados para se calcular o balanço de carga, custo da comunicação, estimativa de área e estimativa de unidades funcionais [15, 35, 36].

O algoritmo de particionamento utilizado é o algoritmo de agrupamento (*clustering*) [37]. O processo de particionamento leva em consideração uma função de custo com os parâmetros de atraso, minimização da área e custo da comunicação. A síntese do *hardware* consiste na implementação automática dos módulos de *hardware* resultante do particionamento. A linguagem de descrição dos módulos de *hardware* utilizada é VHDL. A implementação é feita em FPGAs utilizando-se a tecnologia ALTERA, devido a flexibilidade existente nesta tecnologia, ao baixo custo de implementação e por ser uma excelente opção na confeção de protótipos.

3.3 Trabalhos Relacionados com Protocolos de Comunicação

S. Fischer, J. Wytrebowick e S. Budkowski desenvolveram um ambiente de *HW/SW Codesign* voltado especificamente para protocolos de comunicação. O componente principal desta estrutura é o ambiente EDT (*Estelle Development Toolset*) [38] propício para especifição e simulação em Estelle.

O projeto se inicia a partir de uma especificação de um sistema baseada em Estelle. O Tradutor Estelle do EDT gera uma forma intermediária que serve de entrada para as outras ferramentas do ambiente. Com o auxílio do Simulador Estelle, a especificação é analisada e validada.

O processo de particionamento é realizado pela ferramenta *Distributed Specification Generator* [39] com o auxílio do Avaliador de Métricas que analisa a complexidade e a qualidade da especificação em Estelle. As partes selecionadas para implementação em *software* são geradas utilizando-se o Gerador de Código Estelle-C e as partes selecionadas para implementação em *hardware* são geradas utilizando-se o Tradutor Estelle-to-VHDL [32, 40].

Os códigos gerados necessitam ser refinados de modo a incluir todas as subrotinas relevantes à comunicação entre o *software* e o *hardware*. Quanto ao código VHDL, este está apto para simulação, mas necessita ser retrabalhado de modo a obedecer as restrições da ferramenta de síntese escolhida.

A. Wenban, J. O'Leary e G. Brown desenvolveram um processo de *codesign* de protocolos de comunicação [41]. A especificação do comportamento dos protocolos é realizada através da linguagem Promela.

As partes que serão implementadas em *hardware* e *software* são obtidas a partir da utilização de compiladores de *hardware* e *software*. O compilador de *software* traduz especificações escritas em Promela para a linguagem C++ e o compilador de *hardware* produz uma descrição do circuito em VHDL a partir da linguagem Promela. A criação das instâncias responsáveis pela comunicação entre o *hardware* e o *software* é realizada manualmente.

3.4 Comentários

A utilização da metodologia de *Codesign* no projeto de sistemas é um campo ainda recente. Neste capítulo foram apresentados diversos ambientes de *Codesign* desenvolvidos em renomadas universidades e grupos de pesquisa. Estes ambientes

diferem basicamente no tipo de especificação inicial, no método de particionamento, na arquitetura alvo e no método de validação utilizado. Estes ambientes diferem também no domínio de aplicação escolhido. Outros trabalhos relacionados com o tema também foram apresentados.

O presente trabalho concentra-se em propor uma metodologia visando a síntese de protocolos de comunicação adotando para isso a técnica de Codesign. A obtenção dos parâmetros de custo e atraso de implementações em HW, durante o ciclo de projeto, consiste em um dos objetivos desta metodologia. Para tal, esta metodologia utiliza, como representação inicial, um modelo de máquina de estados finitos. Adaptações em relação a este modelo são propostas com intuito de facilitar as futuras implementações em *hardware*. O método de particionamento é automático e baseado na otimização de desempenho do sistema. A etapa de síntese de HW é efetuada, em contrapartida ao uso de estimadores de HW, com a finalidade de se obter medidas mais precisas.

O próximo capítulo apresenta a metodologia para a síntese de protocolos de comunicação utilizando *HW/SW Codesign*. Um detalhamento da metodologia é apresentado, bem como, todas as decisões de projeto adotadas visando sua implementação.

Capítulo 4

Metodologia de HW/SW Codesign de Protocolos de Comunicação

4.1 Introdução

A Utilização da metodologia de *Codesign* no projeto de protocolos de comunicação tem assumido maior importância devido à crescente demanda por redes com alto desempenho e por aplicações que exigem altas taxas de processamento. Com o intuito de se atender aos requisitos de velocidade das atuais redes, tornou-se necessário recorrer à utilização de implementações em *hardware*.

A metodologia apresentada neste trabalho, congrega uma série de técnicas em um único processo visando a síntese de protocolos utilizando partes em *hardware* e partes em *software*. O ponto de partida consiste na adoção de um projeto cooperativo, *HW/SW Codesign*, baseado em dois ambientes de projeto específicos, onde a possibilidade de verificação e simulação de todo o sistema é encontrada em qualquer etapa do projeto.

O problema principal consiste em se determinar que partes do protocolo serão implementadas em *hardware* e que partes serão implementadas em *software*. De modo a se comparar diferentes partições do sistema, uma função objetivo é definida utilizando-se como parâmetros o desempenho do protocolo e o seu custo de imple-

mentação. Para se obter tais parâmetros, este trabalho utiliza duas ferramentas: Synopsys e ALTERA para a síntese de *hardware* em ASIC com uma biblioteca de células padrão e em PLD através de lógica programável.

A síntese do *hardware* é realizada utilizando-se a linguagem VHDL, que é uma linguagem padrão para a descrição de sistemas eletrônicos digitais. A descrição comportamental em VHDL do protocolo é utilizada como entrada para as ferramentas Synopsys e ALTERA de onde são obtidas as medidas de atraso e área dos circuitos sintetizados.

A seção 4.2 descreve as ferramentas utilizadas neste trabalho. A seção 4.3 apresenta a metodologia aplicada à síntese de protocolos utilizando *HW/SW Codesign*. Essa seção mostra a metodologia detalhadamente, abordando inclusive, as decisões de projeto adotadas.

4.2 Ferramentas de Apoio

Para a obtenção das medidas de atraso e custo que orientam a escolha da melhor partição *HW/SW* do protocolo são utilizadas duas ferramentas: o Synopsys para a obtenção do custo do *hardware* e medidas de atraso nas implementações em ASIC e o ALTERA para a obtenção das medidas de atraso nas implementações em PLD.

O projeto de circuitos integrados se inicia a partir de uma descrição comportamental do sistema na linguagem VHDL. Implementações em ASIC e PLD são sintetizadas de modo a se obter uma variedade maior de opções de projeto. O Synopsys é um dos ambientes mais conceituados e amplamente utilizados. Ele produz como resultado um circuito lógico do qual são extraídas informações referentes ao custo de implementação e medidas de atraso. O ALTERA é um ambiente de prototipagem rápida capaz de produzir um circuito integrado que realiza funções complexas através de lógica reprogramável. Deste circuito são extraídas medidas de atraso.

4.2.1 Synopsys

A implementação de sistemas em *hardware* tomou um grande impulso devido à adoção de uma linguagem única capaz de descrever sistemas digitais nos diversos domínios de representação. Dentre estas linguagens VHDL é uma das mais adotadas tanto pelos centros de pesquisa quanto pela indústria. Isto permitiu a construção de ambientes de projeto de *hardware* integrados, onde as etapas intermediárias do processo de síntese são implementadas automaticamente.

Um circuito integrado VLSI (*Very Large Scale Integrated*) pode ser descrito a partir de um dos três domínios de representação: comportamental, estrutural e físico. Cada domínio pode ser hierarquicamente dividido em alguns níveis de abstração, o que permite que um projeto de circuito integrado possa ser classificado pelo seu domínio e pelo seu nível. Estes domínios são ilustrados de um modo mais claro através do diagrama-Y de Gajski [42].

A ferramenta Synopsys permite uma série de ações a partir da utilização de diversas de suas sub-ferramentas. A síntese de circuitos pode ser efetuada a partir da descrição de sistemas em qualquer um dos três domínios de representação. Normalmente, um projeto de circuitos se inicia a nível comportamental, pois este domínio permite a descrição de um projeto num nível de abstração mais elevado. Estas descrições podem ser efetuadas nas linguagens VHDL e Verilog.

Neste projeto restringimos a utilização desta ferramenta à síntese lógica de circuitos a partir de descrições comportamentais em VHDL. Como dito anteriormente, estas descrições possuem um nível de abstração mais elevado e por isso são mais familiares a projetistas de protocolos de comunicação.

Synopsys realiza a síntese lógica do sistema utilizando métodos de alocação e roteamento de células e otimização de recursos. Estas tarefas são "transparentes" para o projetista que não precisa especificar a arquitetura exata de um projeto, podendo então explorar diferentes implementações possíveis objetivando uma arquitetura ótima. O projetista não precisa nem criar unidades funcionais mais complexas como somadores e multiplicadores.

A utilização desta ferramenta oferece, também, redução no tempo de especificação e simulações mais rápidas (devido à utilização de descrições num nível mais abstrato), exploração arquitetural (possibilidade de alteração na estrutura da descrição comportamental) e alta qualidade dos resultados (devido à automatização dos métodos de síntese).

4.2.2 ALTERA

Os Dispositivos de Lógica Programável (PLDs) são circuitos integrados digitais configuráveis utilizados para implementar funções lógicas sob medida. As PLDs podem implementar qualquer expressão booleana ou elementos de armazenamento com estruturas lógicas padrão cuja função é definida através de programação.

As PLDs são consideradas como uma alternativa aos projetos discretos e aos dispositivos customizados ou semi-customizados, tais como ASICs. Com a utilização de tecnologias na faixa de submicron na implementação de sistemas digitais, os fabricantes de PLDs vêm sendo capazes de oferecer dispositivos com maior densidade de integração, melhor desempenho e menor custo por função do que os dispositivos discretos.

A prototipagem é útil na estimativa de eficiência em área e velocidade do *hardware* sintetizado sendo uma alternativa econômica para implementações sujeitas a modificações ao longo do tempo sem que isto implique em uma nova rodada de fabricação do circuito integrado a cada nova versão do projeto. As desvantagens deste tipo de implementação se referem às limitações de velocidade e desempenho deste tipo de dispositivos.

Os projetistas geralmente precisam avaliar algumas exigências de projeto, tais como, velocidade, custo, tempo de desenvolvimento, prototipagem e tempo de simulação, tempo de fabricação e possibilidade de modificações futuras.

Comparando a eficácia em atender as exigências de projeto utilizando PLDs ou ASICs, tem-se que em termos de velocidade de circuito sintetizado, ASICs são muito mais rápidos do que PLDs. Isto ocorre pois o projeto de ASICs possui um maior grau

de liberdade na alocação e roteamento de células. Em contrapartida, em termos de custo, a utilização de PLDs é muito mais barata do que ASICs. O projeto em ASIC somente conseguirá alcançar um baixo custo para um alto volume de produção.

Dependendo da sofisticação do sistema em questão, o tempo de desenvolvimento de um projeto ASIC é bem maior que um projeto PLD, pois este somente depende de programação. Portanto, a prototipagem utilizando PLDs é mais rápida do que o desenvolvimento de ASICs. Por fim, uma das grandes vantagens do uso de PLDs é a possibilidade de modificações futuras, pois seu princípio fundamental é a reprogramação.

4.3 A Metodologia Proposta

A metodologia para a síntese de protocolos em *hardware* e *software*, apresentada neste trabalho congrega a técnica de *Codesign* com as ferramentas de apoio descritas na seção 4.2 e fornece subsídios que auxiliam o projetista na escolha da melhor partição. Estes subsídios são representados por medidas de atraso e custo das implementações em *hardware* e *software* que servirão como parâmetros para a otimização de uma função objetivo.

A figura 4.1 apresenta o diagrama em blocos da metodologia proposta. Inicialmente, o projetista especifica o protocolo através de um modelo de máquinas de estados adaptado às características de *hardware*, onde cada transição de estado representa uma operação ou um conjunto de operações a serem avaliadas. Este modelo serve como base para a descrição do protocolo na linguagem VHDL, para implementação em *hardware*, e na linguagem C, para implementação em *software*. A partir destas implementações, são obtidas as medidas de atraso e custo para ambos os domínios. Neste ponto, cria-se uma tabela com os valores de atraso e custo associado a cada transição de estado do protocolo em *hardware* (ASIC e PLD) e *software*.

A segunda etapa consiste em se determinar quais transições serão implementadas em *hardware* e quais serão implementadas em *software*. Este trabalho optou por utilizar a estratégia de particionamento proposta por Miranda em [43]. Essa estra-

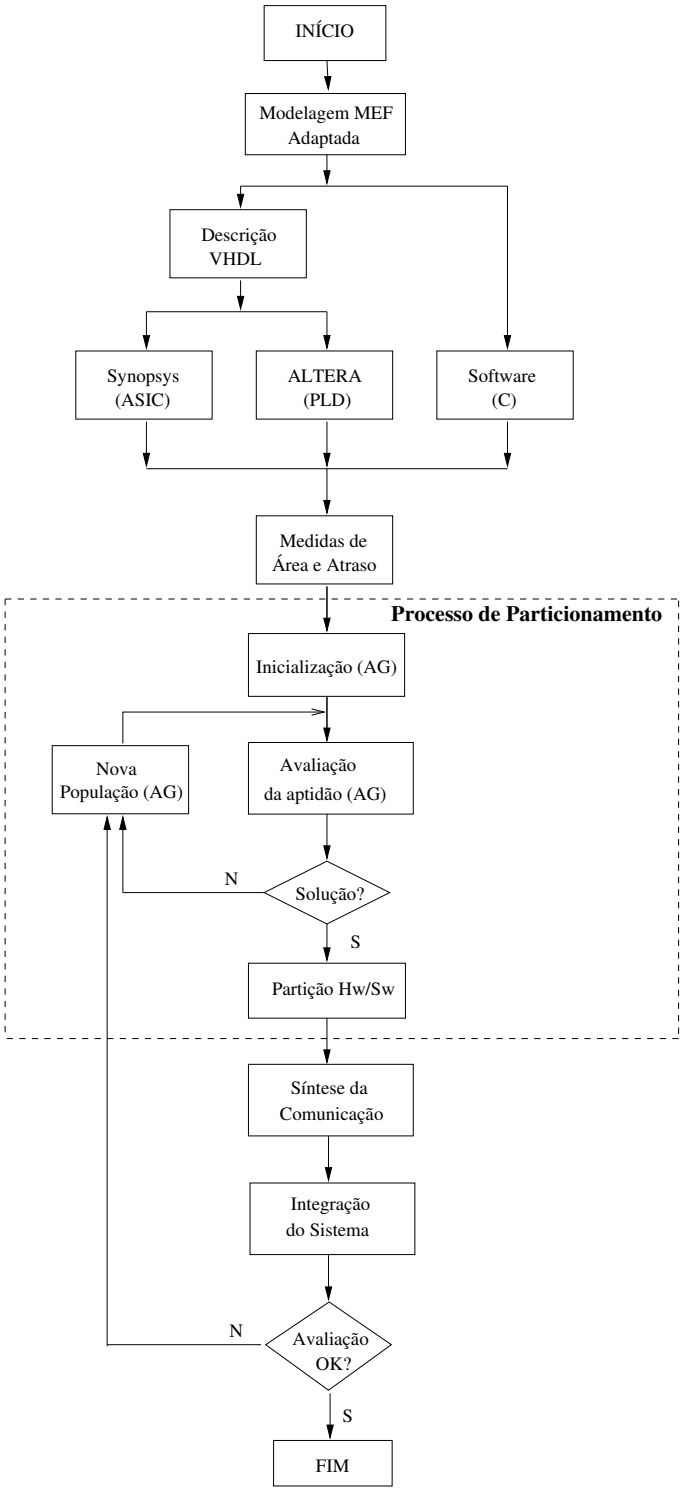

Figura 4.1: Diagrama de blocos da metodologia proposta.

tégia é realizada com o auxílio de um algoritmo genético, da ferramenta de análise de desempenho Tangram-II e das medidas de atraso e custo fornecidas pelo presente trabalho. O algoritmo genético é responsável pela otimização da função objetivo. O

AG selecionará diferentes valores de atraso para as diferentes implementações possíveis e os fornecerá para o Tangram-II calcular o desempenho do protocolo. Com o valor do desempenho para um específico conjunto possível de implementação e com o valor de custo associado a este conjunto, acontece o cálculo da função objetivo. Este processo se repete até que a função objetivo atinja um valor satisfatório, obtendo assim, uma possível partição *hardware/software* que atenda aos requisitos impostos pelo projetista.

A próxima etapa consiste em se efetuar a síntese da comunicação. Esta etapa é responsável em se implementar o esquema de troca de informações entre os diversos módulos de *hardware* e de *software* definidos pelo processo de particionamento. Após esta etapa ocorre, finalmente, a integração do sistema e sua avaliação final. As seções seguintes apresentam com detalhes os passos seguidos no diagrama em blocos da metodologia e apresenta também as decisões de projeto adotadas.

4.3.1 Representação do Sistema

A primeira etapa na implementação da metodologia proposta consiste na especificação da funcionalidade do protocolo através de um modelo escolhido. A descrição da funcionalidade do protocolo pode ser melhor entendida através da utilização deste modelo, conforme discutido no capítulo 2. Diferentes tipos de modelos são requisitados em diferentes domínios de aplicação. Como protocolos de comunicação são sistemas, freqüentemente, dominados por controle, o modelo de Máquinas de Estado Finitos (MEF) foi escolhido. Este modelo representa o sistema como sendo um conjunto de estados e transições que são ativadas por eventos externos.

A implementação em *hardware* de sistemas não é uma tarefa simples. Tendo este detalhe em mente, foram feitas algumas adaptações ao modelo de máquinas de estado tradicionalmente utilizado. Foram inseridos alguns detalhes que são essenciais quando se está modelando sistemas em VHDL. A utilização de sinais de sincronização e a preocupação com o tratamento de dados foram aspectos levados em consideração na modelagem do protocolo.

A especificação de uma arquitetura visa completar a descrição do modelo, especificando como o modelo será realmente implementado. O objetivo de uma arquitetura é descrever o número de componentes, o tipo e a conexão entre eles. A arquiteura alvo escolhida foi a multi-processador. Na metodologia proposta, após o processo de particionamento, as partes designadas para *hardware* serão implementadas em um ou mais circuitos dedicados (ASICs) ou circuitos de lógica programável (PLD) e as partes designadas para *software* serão implementadas em um ou mais processadores para o *software* dedicados.

Este trabalho optou por não utilizar linguagens de programação na especificação inicial do protocolo. Apesar de ter ciência que a grande vantagem da utilização de especificações executáveis é permitir ao projetista verificar a correção da funcionalidade através do processo de simulação, a metodologia proposta optou por utilizar o modelo de MEF adaptado. Esta opção se deveu à três motivações principais:

- o modelo de MEF é um modelo que pode ser testado, excluindo assim possíveis preocupações em se carregar erros iniciais de especificação a etapas posteriores do projeto;
- este tipo de modelagem se compara à utilização de uma representação interna na descrição inicial do sistema. Assim, insere-se uma possibilidade de se "acoplar" diversos módulos de representação inicial baseados em diversas linguagens de programação. Estes módulos somente terão a preocupação de gerar especificações em MEF a partir da utilização da linguagem de programação da preferência do projetista;
- este modelo permite que o sistema seja visto independentemente de uma implementação em *hardawre* ou de uma implementação em *software*. Isto é um fator importante nas etapas de particionamento e refinamento.

4.3.2 Modelo de MEF Adaptado à Síntese

Um dos objetivos deste trabalho é incorporar, em um nível de abstração o mais elevado possível, características específicas de implementações de sistemas em *hardware*. Esta caracterização visa facilitar as futuras descrições em *hardware* e também poder avaliar as implementações de uma forma mais realista, mesmo estando num nível de abstração elevado.

Logo, existe a necessidade de se obter um modelo capaz de descrever as funcionalidades dos protocolos de comunicação e, ao mesmo tempo, capaz de descrever também as características de implementações em *hardware*. Com este intuito, é apresentado o modelo de MEF Adaptado à Síntese de protocolos. Este modelo parte do modelo básico de MEF, que é amplamente utilizado por projetistas de protocolos, onde o comportamento do protocolo é descrito por um conjunto de estados e transições.

Este modelo tem a preocupação de descrever, em um alto nível de abstração, algumas características das especificações em *hardware*. Estas características foram analisadas a partir de descrições VHDL de diferentes sistemas [44, 45, 46]. Estas alterações no modelo de MEF original têm como objetivo proporcionar uma modelagem do protocolo mais preocupada com o processo de síntese de *hardware*. A seguir, são mostradas e comentadas as principais contribuições deste modelo.

Definição do Modelo

O Modelo de Máquina de Estados pode ser descrito por um conjunto de estados {S} e transições {T}. O conjunto de transições engloba um conjunto de ações de entrada {I}, um conjunto de ações de saída {O} e um conjunto de ações internas {Int}. A união das ações de entrada e as ações de saída são designadas como ações externas, visíveis ao ambiente. O comportamento de um determinado protocolo, pode então, ser expresso através desses conjuntos.

Conforme dito anteriormente, VHDL é uma linguagem padrão para descrição de sistemas digitais, mas que não foi concebida com a facilidade de expressar uma

máquina de estados. Portanto, o presente trabalho não se preocupa com o conjunto de estados {S} do modelo e sim, com o conjunto de transições {T}. Parâmetros de todas as transições $\{t_i\}$ do protocolo servirão de base para o processo de particionamento que definirá quais transições serão implementadas em *hardware* e quais serão implementadas em *software*.

Como um dos objetivos deste trabalho é descrever, neste modelo de alto nível, algumas características das especificações em *hardware*, o modelo de MEF Adaptado se concentra, basicamente, nos conjuntos de ações de entrada {I}, de ações de saída {O} e de ações internas {Int} das respecticas transições $\{t_i\}$. As adaptações ao modelo se resumem às restrições que devem ser consideradas para cada um destes conjuntos.

O tópico **Inserção de Sinais** reflete as considerações em relação às ações de entrada {I} e às ações de saída {O}. O tópico **Restrições às Descrições VHDL** comenta algumas estratégias adotadas em relação às ações internas {Int}. Ambos os tópicos visam facilitar a futura tentativa de implementação em *hardware*.

Inserção de Sinais

Uma especificação VHDL, utilizada para síntese de sistemas digitais, é composta por entidades que se comunicam com outras entidades, basicamente, através de sinais. Pode-se agrupar estes sinais em três categorias principais: essenciais ao funcionamento de um sistema digital, essenciais na sincronização da troca de mensagens e utilizados na manutenção da seqüencialidade do funcionamento dos circuitos. A idéia básica, então, é poder descrever estes sinais característicos de implementações em *hardware* a partir de um alto nível de abstração. Logo, estes sinais devem ser parte integrante do modelo proposto.

Os primeiros sinais levados em consideração dizem respeito ao funcionamento de qualquer sistema digital. Sinais como *start*, *reset* e *clock* são encontrados em diversos circuitos digitais e muitas vezes não são considerados quando trabalha-se num nível elevado de abstração. Como, um dos objetivos principais deste trabalho

é facilitar a tarefa de se projetar circuitos em *hardware*, existe a clara necessidade de se preocupar com tais sinais ainda na fase inicial de modelagem do protocolo e suas transições.

No processo de troca de mensagens, uma das partes mais importantes é a sincronização da comunicação. Partindo de diversas observações, os sinais de sincronização responsáveis pela comunicação entre diferentes entidades foram incluídos no modelo em alto nível. Normalmente, em especificações de protocolos em um nível elevado de abstração, as trocas de mensagens são realizadas através de unidades de dados (*Protocol Data Unit* - PDU) passadas diretamente entre entidades.

Estas unidades de dados podem representar a mensagem como um todo ou diferentes mensagens de controle como mensagem de reconhecimento ou de erro. No modelo adotado neste trabalho, essas unidades de dados estarão presentes, mas antes de haver troca de PDUs, haverá a sinalização através dos sinais de sincronização.

A idéia do trabalho não é implementar o mecanismo de controle de estados, principalmente, porque a linguagem VHDL não foi projetada com suporte à implementações de máquina de estados. Na metodologia proposta, todas as transições serão implementadas através de um processo em *software* ou um circuito em *hardware*. Obviamente, estes processos ou circuitos precisam de um controle que indique quando devem iniciar seu funcionamento e quando já efetuaram seu processamento. A implementação da máquina de estados propriamente dita, se dará então, através da seqüência de funcionamento das transições, seqüência esta indicada por sinais de controle. Estes sinais de começo e fim de cada transição seriam encarregados, então, de "mudar de estado".

Classificação das Transições

Uma das principais características dos sistemas de telecomunicações é que seu comportamento pode oscilar entre partes de controle e de comunicação. Isto leva a uma divisão natural na partição do projeto em termos de partes de controle e partes de dados, onde cada parte tem seu próprio fluxo de projeto independente.

Tendo em vista esta característica dos sistemas de comunicação, este modelo optou por classificar as diversas transições em dois tipos: *controle* e *operação sobre dados*. De antemão, as transições de *controle* são mais simples e não necessitam de cuidados extras durante sua implementação. Em contrapartida, as transições de *operação sobre dados* merecem uma consideração especial.

Normalmente, o primeiro passo na síntese de circuitos consiste na divisão da especificação em dois blocos: a estrutura lógica e a estrutura de armazenamento de dados. A estrutura de armazenamento de dados é geralmente implementada a partir de uma memória externa. Esta característica deve ser levada em consideração durante a implementação das transições classificadas como *operação sobre dados*. Todas as estruturas que armazenam dados, como *arrays* e *buffers*, devem ser implementadas separadamente através de uma memória externa [47].

Esta classificação também servirá posteriormente quando houver a necessidade, durante o processo de refinamento da partição, de se dividir uma determinada transição ou então, de se agrupar determinadas transições.

Restrições às Descrições VHDL

VHDL é uma linguagem que possui uma variedade de estruturas e funções que podem ser simuladas, mas não podem ser sintetizadas. O uso de determinadas restrições tem como objetivo permitir que todas as transições possam ser sintetizadas. Estas restrições são definidas levando em consideração os ambientes Synopsys e ALTERA. Possivelmente, outras ferramentas de síntese tenham outros tipos de restrições que impeçam a obtenção de circuitos sintetizáveis. A seguir, estão enumerados os cuidados que devem ser tomados quando se estiver descrevendo uma transição em VHDL:

- evitar o uso da instrução "*wait until ...*". Sempre que possível utilizar a construção "*if..then..elsif..endif*";
- não utilizar a instrução *block*;

- cada arquitetura deve conter uma única instrução *process* dependente somente do sinal de *clock*;
- não utilizar as instruções *function* e *procedure* do VHDL;
- não utilizar os seguintes tipos em VHDL: *physical*, *floating point*, *enumeration* e *record*;
- não utilizar a instrução *generic*;
- e quanto ao tipo *array*, quando usado como variável interna para armazenamento de dados, deverá ser substituído por uma memória externa.

Implementação de uma Transição Genérica

O objetivo desta seção é apresentar de forma didática como será a implementação em VHDL de uma transição genérica. Como já discutido, uma especificação VHDL é composta por entidades. Estas entidades se comunicam através de sinais e seu comportamento é visualizado através de portas de entrada e saída.

Primeiramente, são definidas as portas de entrada e saída de cada entidade responsáveis pela comunicação externa. Basicamente, estas portas são as próprias PDUs das especificações em alto nível do protocolo, acrescidas dos sinais de sincronização associados a estas PDUs. Outros sinais acrescentados, através de portas, são os sinais de *start*, *clock* e *reset*. Estes sinais são obrigatórios e devem estar presentes em todas as descrições das transições em VHDL.

O comportamento da transição é descrito pela estrutura *architecture*. Esta descrição conterá uma única instrução concorrente definida pela instrução *process*. Este processo será sensível às variações do sinal de *clock*, ou seja, a cada transição deste sinal o processo será executado. Dentro do processo, o comportamento, propriamente dito, é implementado através da construção "*if..then..elsif..endif*". Os sinais de sincronização são testados por cada instrução "*if*" e dependendo da situação apresentada ocorre uma operação específica.

O trecho abaixo ilustra a estrutura responsável pela implementação de uma transição em VHDL. Este trecho apresenta a implementação da transição Recebe ACK.

```
ENTITY PRIMEIRA_TRANS3 IS
  port (start: in std_logic;
        clk: in std_logic;
        reset: in std_logic;
        possoenviartrans: in std_logic;
        podemandarusuario: out std_logic;
        msg_usuario_cod: in std_logic;
        valor_janela: in std_logic_vector (9 downto 0);
        msg_entidade_cod: out std_logic;
        novo_valor_janela: out std_logic_vector (9 downto 0));
END PRIMEIRA_TRANS3;

ARCHITECTURE PRIMEIRA_TRANS3_BODY OF PRIMEIRA_TRANS3 IS

BEGIN

  PROCESS (clk)

   BEGIN

     IF clk'event and clk = '1' then
         IF (reset = '1') THEN
             Conjunto de Ações
         ELSIF (start = '1') THEN
             Conjunto de Ações
         ELSIF (msg_usuario_cod = '1') THEN
             Recebe valor_janela
         ELSIF (possoenviartrans = '1') THEN
             Envia novo_valor_janela
```

```
        END IF;
    END IF;
  END PROCESS;

END PRIMEIRA_TRANS3_BODY;
```

Pode-se observar a presença dos sinais de *start*, *clock* e *reset*, bem como a presença dos sinais de sincronização. Neste caso, os sinais de sincronização são: *possoenviar-trans*, *podemandarusuario*, *msg-usuario-cod* e *msg-entidade-cod*. Observa-se no corpo da descrição a presença de um único processo sensível ao sinal de *clock*. Observa-se também a presença da construção "*if..then..elsif..endif*", onde as ações são realizadas caso ocorra um determinado valor para os sinais de *reset* (zera-se todas as variáveis e sinais de saída), de *start* (envio de um aviso de pronto), de *msg-usuario-cod* (recebimento do valor da janela de congestionamento) e de *possoenviartrans* (pode enviar novo valor para a variável de janela de congestionamento).

4.3.3 Síntese em Hardware e Software

Neste trabalho, a especificação do protocolo em máquina de estados serve como referência para a descrição em VHDL e C de todas as transições de estado. No caso específico do *hardware*, cada transição de estado, contendo uma ou mais operações do protocolo, é associada a uma *entidade* em VHDL. Cada *entidade* é sintetizada, de modo a se obter as medidas de atraso e área referentes às transições de estado quando implementadas em *HW*. A ferramenta Synopsys sintetiza um circuito lógico a partir da descrição comportamental de cada transição em VHDL. Esta implementação é realizada utilizando-se uma biblioteca de células padrão (*standard cells*) numa dada tecnologia. As etapas de alocação e roteamento de células e otimização são automáticas e se tornam "transparentes" para o projetista. A ferramenta ALTERA implementa as funções lógicas de cada transição a partir de suas descrições em VHDL.

Para o caso específico do *software*, as mesmas transições de estado são imple-

mentadas na linguagem C. Cada transição corresponde a um módulo independente. Estas descrições são compiladas através de um compilador C e são obtidas as medidas de atraso para cada transição implementada. As medidas de custo referentes às implementações em *software* são obtidas através do tempo gasto para se descrever cada uma das transições. Esta medida é calculada associando-se um valor financeiro por homem/hora de trabalho. Para a implementação em PLD, tem-se além do mesmo valor de homem/hora de trabalho, um acréscimo referente à utilização da placa de PLD. Para a implementação em ASIC, tem-se o valor de homem/hora de trabalho somado ao custo da área do circuito sintetizado.

Ao final do processo, obtém-se uma tabela contendo todas as medidas de atraso e custo de implementação para cada uma das transições do protocolo. Estas medidas serão utilizadas para avaliar a qualidade de cada partição determinada pelo processo de otimização (AG + função objetivo).

4.3.4 Particionamento HW/SW

A etapa de particionamento é responsável em se determinar quais operações serão implementadas em *hardware* e quais operações serão implementadas em *software*. O processo de particionamento consiste, praticamente, no tipo de algoritmo de particionamento escolhido e na avaliação da partição. Este processo termina quando as partições encontradas respeitam as restrições impostas pelo projetista e têm uma boa relação custo/desempenho.

O algoritmo de particionamento implementado em [43] é um algoritmo genético simples que otimiza uma função objetivo baseada em parâmetros de desempenho e custo das implementações das transições do protocolo. Esta otimização parte da análise da tabela de medidas de atraso e custo para todas as possíveis implementações das transições do protocolo (ASIC, PLD e *software*) fornecidas pelo presente trabalho.

O AG irá gerar um conjunto de possíveis implementações das transições, ou seja, um conjunto de possíveis indivíduos. Este conjunto de indivíduos representa

uma população e seu tamanho é definido pelo projetista. Para cada indivíduo da população é calculada sua aptidão através da função objetivo definida. A otimização desta função consiste em selecionar os melhores indivíduos (aqueles que fornecem os melhores resultados para a função) e gerar uma nova população, que espera-se encontre melhores resultados do que a população anterior. A geração da nova população é determinada pelos processos de seleção, cruzamento e mutação dos indivíduos. O ponto de parada do algoritmo pode ser o número de gerações ou a taxa de variação do valor da função objetivo.

A seguir, são descritas as diversas etapas que fazem parte do processo de particionamento [48]. Inicialmente, descreve-se o modelo que será utilizado pela ferramenta de análise de desempenho Tangram-II. Esta ferramenta fornecerá valores que serão utilizados no cálculo da função objetivo. Esta função fornece uma medida da qualidade de uma determinada solução e através de sua avaliação pode-se selecionar aquela que satisfaz melhor os requisitos de desempenho. Posteriormente, baseado em um critério de partição, determina-se que partes serão designadas para implementação em *hardware* e que partes serão designadas para implementação em *software*.

Descrição Inicial e Verificação Formal

O protocolo inicialmente descrito por um modelo de máquina de estados, passa a ser, neste estágio, descrito por redes de Petri estocásticas (*Stochastic Petri Nets* - **SPNs**), que são uma extensão do modelo de rede de Petri tradicional. SPN é utilizado por ser mais adequado na construção do modelo na linguagem da ferramenta de análise de desempenho Tangram-II. A opção pelas SPNs para descrever o comportamento do protocolo neste estágio, se deve também, à disponibilidade de ferramentas para a verificação formal da especificação e à sua estreita ligação ao modelo de máquina de estados finitos.

Nas SPNs, o tempo de disparo de uma transição, após a mesma ser habilitada, possui distribuição exponencial com taxa λ_i, onde λ_i é a taxa associada à transição i do modelo. Em Molloy [49] é mostrado que as SPNs são equivalentes a uma cadeia

de Markov a parâmetro contínuo finita (CMTC) [50], onde cada *marca* da rede de Petri corresponde a um estado da cadeia e as taxas da SPN correspondem às taxas de transição entre os estados da cadeia.

Uma vez descrito o comportamento do protocolo, verifica-se a correção dessa descrição através da aplicação da ferramenta **ARP** (Analisador de Redes de Petri), que realiza uma análise sintática e semântica. A análise sintática verifica se a rede é *viva*, *limitada* e *reiniciável*. A análise semântica é realizada através da verificação de propriedades específicas, como *invariantes de transição* e *invariantes de lugar*. Dessa forma, pode-se verificar se a descrição do protocolo está livre de *deadlocks* e *livelocks*.

Construção do Modelo

Uma vez verificada a correção da descrição inicial através da ferramenta ARP, é construído um modelo na linguagem da ferramenta de análise de desempenho Tangram-II, utilizando-se uma interface gráfica, o TGIF (Tangram Graphical Interface) [51]. Cada transição da rede de Petri é associada a um *evento* e cada *lugar* é representado por uma *variável de estado* no modelo do Tangram-II. Os parâmetros do modelo construído no Tangram-II são as taxas dos *eventos*. Como cada transição da SPN, quando habilitada, possui um tempo de disparo com distribuição exponencial, os *eventos* também têm distribuição exponencial e, portanto, cada modelo tem uma Cadeia de Markov a Tempo Contínuo (CMTC) associada, gerada pela ferramenta Tangram-II. Deve ser observado que a cadeia de Markov gerada é uma cadeia **literal** que somente é resolvida quando são atribuídos valores numéricos aos parâmetros, uma vez que os métodos de solução disponíveis na ferramenta Tangram-II não são literais.

Construção da Função Objetivo

A *função objetivo* ou função de erro do problema de partição HW/SW é dada pela diferença entre os valores das medidas de desempenho desejadas, especificadas pelo

projetista, e os valores das medidas de desempenho obtidas para um dado conjunto de parâmetros. Como os parâmetros especificados normalmente "competem entre si", é útil atribuir-se pesos a cada um dos parâmetros que compõem a função objetivo para equalizar as possíveis diferenças de sensibilidade entre eles. Essa função fornece uma medida da qualidade de uma determinada solução e através de sua avaliação pode-se selecionar aquela que satisfaz melhor os requisitos de desempenho.

A função objetivo utilizada neste trabalho é definida a partir de medidas de desempenho do protocolo, tais como vazão, retardo, probabilidade de perda, entre outras, obtidas a partir do Tangram-II. A forma geral da função objetivo é a seguinte:

$$\varepsilon(\lambda) = \sum_{i=1}^{n} w_i * |f_{d_i}(\lambda) - f_{o_i}(\lambda)| \quad R^n \rightarrow R^1 \tag{4.1}$$

onde:

$\lambda = (\lambda_1, ..., \lambda_n)$ - é o vetor de parâmetros da otimização;

f_{d_i} - é a i-ésima medida de desempenho desejada;

f_{o_i} - é a i-ésima medida de desempenho obtida;

w_i - são os fatores de ponderação ou normalização;

As medidas de desempenho utilizadas dependem dos requisitos de qualidade de serviço desejados.O problema de otimização consiste em determinar λ^* tal que:

$$\varepsilon(\lambda^*) = \min_{\lambda} \varepsilon(\lambda) \tag{4.2}$$

Otimização da Função Objetivo

O problema de otimização inicial consiste em determinar as taxas de transição de estado da CMTC, isto é, as taxas dos *eventos* do modelo, que satisfazem restrições de desempenho impostas pelo projetista. Com base nessas restrições, o melhor

conjunto de parâmetros é determinado pelo algoritmo genético. A determinação dessas taxas fornece subsídios para orientar a escolha de um critério de partição.

A cadeia de Markov gerada pelo Tangram-II só pode ser resolvida quando os parâmetros forem substituídos por valores numéricos. Esses valores são fornecidos por um método de otimização baseado em AGs que, inicialmente, gera conjuntos de valores para os parâmetros de forma aleatória, porém obedecendo certas restrições de projeto. Após a substituição dos parâmetros pelos valores numéricos, a cadeia de Markov é resolvida através de algum dos métodos de solução analítica que a ferramenta Tangram-II oferece. Dessa forma são calculadas as probabilidades em estado estacionário de cada um dos estados da CMTC. Essas probabilidades são utilizadas para o cálculo de medidas de interesse relacionadas com o desempenho do protocolo. Essas medidas são os *valores obtidos*.

A especificação a ser atendida pode ser dada, por exemplo, por uma curva relativa a uma determinada medida de desempenho. Essa curva fornece os *valores desejados* da função objetivo. De posse dos valores desejados e dos valores obtidos pode-se calcular o valor da função objetivo para cada conjunto de taxas gerado pelo AG. Cada conjunto é denominado um *indivíduo* e o número de indivíduos corresponde ao tamanho da *população* gerada pelo AG. O projetista também especifica o número de pontos da curva que ele deseja aproximar. O valor da função objetivo, dado pela soma dos erros entre os valores desejados e os valores obtidos, fornece uma medida da *aptidão* do indivíduo para resolver o problema.

O processo de otimização é iniciado calculando-se a *aptidão* de cada *indivíduo* da população e verificando se algum desses *indivíduos* representa a solução do problema. Caso a solução não tenha sido encontrada, realiza-se uma seleção dos *indivíduos*, com maior probabilidade para os *indivíduos* mais aptos, faz-se o cruzamento de cada um com seu *parceiro* e gera-se uma nova população. O Tangram-II é realimentado com a nova *geração* de indivíduos e repete-se o processo até que se encontre um conjunto de taxas que atenda a um erro máximo especificado pelo projetista ou se alcance um determinado número de gerações.

O Critério de Partição

Para a automatização do particionamento é fundamental estabelecer um critério que relacione as taxas calculadas pelo AG com uma determinada partição HW/SW. Quanto maior a taxa de um determinado evento, maior o número de vezes, por segundo, que um determinado conjunto de operações do protocolo é executado. Este fato poderia levar à conclusão (precipitada) de que essas operações são as que devem ser implementadas em HW. No entanto deve-se considerar, a partir dos resultados obtidos resolvendo-se a cadeia de Markov associada, as probabilidades em estado estacionário de cada estado do sistema. Ou seja, mesmo que um evento tenha uma alta taxa, a probabilidade do sistema estar num estado em que o mesmo está habilitado pode ser muito baixa.

A partir das considerações realizadas acima foi estabelecido como critério de partição HW/SW dos eventos, onde cada evento representa um conjunto de operações do protocolo, o produto $\lambda_i * \pi$, onde:

λ_i - taxa do evento i determinada pelo AG.

π - soma das probabilidades em estado estacionário dos estados nos quais o evento i está habilitado.

É como se cada evento tivesse uma "vazão" associada a ele. Desta forma, as operações do protocolo relacionadas com os eventos de maior "vazão" são implementadas em HW e aquelas associadas aos eventos de menor "vazão" são implementadas em SW.

A metodologia de otimização e o critério apresentados fornecem subsídios ao projetista para uma escolha objetiva da melhor partição, mantendo a análise em alto nível, sem a necessidade de um conhecimento detalhado no nível de circuitos. Conseqüentemente, o projetista pode tomar uma decisão consistente baseada em dados concretos e não apenas em sua experiência.

4.3.5 Síntese da Comunicação e Integração do Sistema

Após o processo de particionamento, onde haverá a determinação de quais partes do protocolo serão implementadas em *hardware* e quais partes serão implementadas em *software*, ocorre a síntese da comunicação. Esta etapa é responsável por implementar o esquema de troca de informações entre os diversos módulos de *hardware* e *software*. Esta troca de informações pode ser realizada através de sinais de controle e sincronização no caso dos módulos de *hardware* e através de *sockets* no caso dos módulos de *software*.

Na integração das partes designadas para cada domínio são examinadas as necessidades de inserção de funcionalidades inerentes à comunicação. Lembrando que, o próprio modelo utilizado para se descrever o protocolo inicialmente já trata de algumas dessas necessidades, como sinais de sincronização em *hardware* e certos cuidados com o tratamento de dados. Sendo assim, esta etapa se torna um pouco menos trabalhosa.

4.4 Comentários

Este capítulo apresentou em detalhes a metodologia proposta com o objetivo de implementar protocolos com partes em *hardware* e partes em *software*. Esta metodologia basea-se na utilização da técnica de *Codesign*. Foram apresentados detalhes referentes à todas as etapas de desenvolvimento do ciclo de projeto. O modelo utilizado para a descrição do protocolo, a arquitetura alvo adotada, as etapas de síntese de *hardware* e de *software*, os parâmetros que serão utilizados pelo processo de particionamento e a etapa de síntese da comunicação também foram abordados.

Este capítulo apresentou também as principais características das ferramentas Synopsys e ALTERA utilizadas para obter as medidas dos parâmetros de custo da implementação e atraso das diferentes transições do protocolo. Estas medidas servirão como um subsídio ao projetista de modo a auxiliá-lo na etapa de particionamento.

O próximo capítulo apresenta os resultados obtidos quando da utilização da metodologia proposta na implementação de três protocolos que possuem diferentes características. Foram implementados os protocolos TCP "Congestion Avoidance", o protocolo WATM para controle de *handoff* em redes sem fio e o protocolo ARM que realiza *multicast* confiável. Neste capítulo são abordadas todas as etapas de desenvolvimento do projeto destes protocolos.

Capítulo 5

Resultados

5.1 Introdução

NESTE capítulo são apresentados e discutidos os resultados obtidos a partir da aplicação da metodologia proposta neste trabalho em protocolos com diferentes características. Com a finalidade de ajudar na compreensão dos resultados são apresentados gráficos e tabelas, onde são consideradas algumas observações pertinentes ao desenvolvimento dos projetos.

Em cada uma das implementações serão detalhadas as funcionalidades dos protocolos e como essas funcionalidades foram descritas através da utilização do modelo de MEF adaptado. Serão apresentadas as medidas de área e atraso obtidas para as diversas transições dos protocolos e como estas medidas foram obtidas. Serão abordados os respectivos processos de particionamento, ilustrando e detalhando suas diversas etapas que conduzem a escolha de uma determinada partição HW/SW que atenda aos requisitos de projeto. Por fim, são mostradas as etapas de refinamento, onde possíveis modificações no projeto podem ser realizadas, e a etapa de integração do sistema.

As funcionalidades dos protocolos TCP (seção 5.2) e ARM (seção 5.4) são especificadas através do modelo de MEF Adaptado para as necessidades da síntese de *hardware*. Este modelo descreve o comportamento geral do protocolo através de um

conjunto de estados e transições. Cada transição corresponde a uma operação ou um conjunto de operações. Todas as possíveis transições de estado serão implementadas em ASIC, PLD e *software* de modo a se obter um conjunto de medidas de atraso e custo. Este conjunto de medidas será utilizado no processo de particionamento, mais especificamente na avaliação da partição, no cálculo da função objetivo.

Este modelo servirá como base para as descrições das transições em VHDL e C. As descrições das transições em VHDL serão sintetizadas nas ferramentas Synopsys e ALTERA com o objetivo de se obter medidas de atraso e custo em ASIC e PLD. As descrições em C serão compiladas obtendo-se assim, as medidas de atraso relativas a cada uma das transições. De posse dessas medidas, o processo de avaliação das diferentes partições propostas pelo algoritmo genético pode ser realizado de forma mais acurada.

As descrições VHDL para cada uma das transições foram utilizadas pelo Synopsys com o objetivo de sintetizar circuitos ASICs. Cada uma das especificações foi implementada utilizando uma biblioteca de células padrão (*standard cells*) ES2 com tecnologia 0,7μm. O valor máximo para o *clock* foi obtido a partir da análise do caminho crítico do circuito sintetizado. As medidas de atraso foram calculadas através da simulação das transições. Os valores de área são fornecidos automaticamente pelo Synopsys.

As mesmas descrições VHDL foram utilizadas pelo ALTERA com o objetivo de sintetizar circuitos PLDs. Estas implementações foram realizadas utilizando ALTERA Max+PlusII (*Multiple Array Matrix Programmable Logic User System*) [55]. Durante a compilação, o compilador do ambiente ALTERA ficou no modo *auto* para a escolha do componente mais adequado ao projeto. O valor máximo para o *clock* e as medidas de atraso foram obtidas simulando-se o comportamento das transições. Note que uma vez escolhido, o componente ALTERA possui uma área fixa independente do projeto, ou seja, não tem sentido falar em medidas de área no projeto ALTERA.

A seção 5.2 aborda o protocolo "TCP Congestion Avoidance", a partir da modelagem do seu mecanismo de controle de congestionamento. A implementação de

um protocolo bastante difundido é o objetivo desta seção. Na seção 5.3 é modelado um protocolo de controle de *handoff* aplicado a uma rede sem fio baseado na tecnologia ATM (WATM) com estações base que controlam o roteamento à medida que as unidades móveis trocam de células. Nesse exemplo pretende-se analisar a metodologia quando aplicada ao projeto de um protocolo dominado por fluxo de controle. A seção 5.4 apresenta a modelagem de um protocolo *multicast* confiável que realiza processamento nos nós (*nós ativos*). Nesse exemplo, pretende-se analisar a metodologia no caso de um protocolo com um alto grau de processamento nos nós e dominado por fluxo de dados.

5.2 TCP "Congestion Avoidance"

5.2.1 Descrição das Funcionalidades do Protocolo

Nesta seção, a metodologia proposta é aplicada na implementação de um protocolo bastante conhecido, o protocolo TCP [52, 53]. Neste exemplo é analisado o desempenho do TCP quando em fase de controle de congestionamento, o "TCP Congestion Avoidance", utilizando-se o modelo matemático proposto em Mathis [54]. Este protocolo efetua a transmissão de dados entre uma fonte e um receptor baseado em regras de conversação bem definidas.

O transmissor se encarrega de efetuar o envio de dados com o auxílio de uma variável denominada *janela de congestionamento*. Esta variável será responsável por determinar a quantidade de *bytes* que o transmissor poderá enviar sem que isto acarrete sobrecarga na rede. A variável *janela de congestionamento* é incrementada de uma parcela constante para cada sinal de ACK recebido pelo emissor, ou seja, a cada mensagem corretamente transmitida, a fonte pode enviar um número maior de informações. Esta variável é dividida pela metade quando é detectado um congestionamento, ou seja, quando ocorre uma perda de pacote. Assim, o transmissor fica obrigado a enviar um número menor de *bytes* na próxima vez.

O modelo proposto por Mathis é simplificado e supõe que somente o mecanismo

"Congestion Avoidance" influencia o desempenho do TCP. Uma das suposições feitas considera o tráfego moderado, ou seja, considera que existem poucas perdas de pacotes, o que permite ao TCP se recuperar destas perdas sem a necessidade de *timeout*. Outra suposição é que não existem retransmissões por *timeout*.

A transmissão de dados considera que a janela do receptor possui um tamanho "suficientemente grande", ou seja, a janela de congestionamento é controlada pela janela do emissor e o emissor tem sempre dados a enviar. Nesta transmissão, a ocorrência de múltiplas perdas dentro de um *round trip time* (RTT) indica sinal de congestionamento e o RTT é constante porque a banda passante é considerada suficiente para que não ocorra a formação de filas.

O receptor envia um ACK para cada *segmento* de dados (certa quantidade de dados em *bytes* de tamanho fixo) recebido. Ocorre a perda de pacotes com uma probabilidade constante p, o que significa que aproximadamente $1/p$ pacotes são enviados corretamente, seguidos da ocorrência de uma perda. Detalhes referentes à recuperação e retransmissão dos pacotes são desprezados, embora a recuperação de perdas seja completada dentro de um RTT. Por fim, as conexões são consideradas longas o suficiente para que o algoritmo de "Congestion Avoidance" alcance estado estacionário.

A próxima seção apresenta a modelagem do protocolo de forma a atender as funcionalidades apresentadas nesta seção e, visando principalmente, o processo de síntese de *hardware*.

5.2.2 Modelagem do Protocolo

A figura 5.1 ilustra a modelagem das funcionalidades do protocolo através do modelo de MEF adaptado às características da síntese de *hardware*. Observa-se que o modelo consiste de três estados (Pronto (S1), Espera_ACK (S2) e Processa_MSG (S3)) e seis transições (envia_msg, recebe_ack, recebe_perda, recebe_msg, envia_ack e envia_perda). Ressalta-se na figura, a inclusão dos sinais de sincronização importantes na descrição destas transições em VHDL durante o processo de síntese

de *hardware*. Ressalta-se também, a classificação das transições em transições de controle e transições de operações de dados.

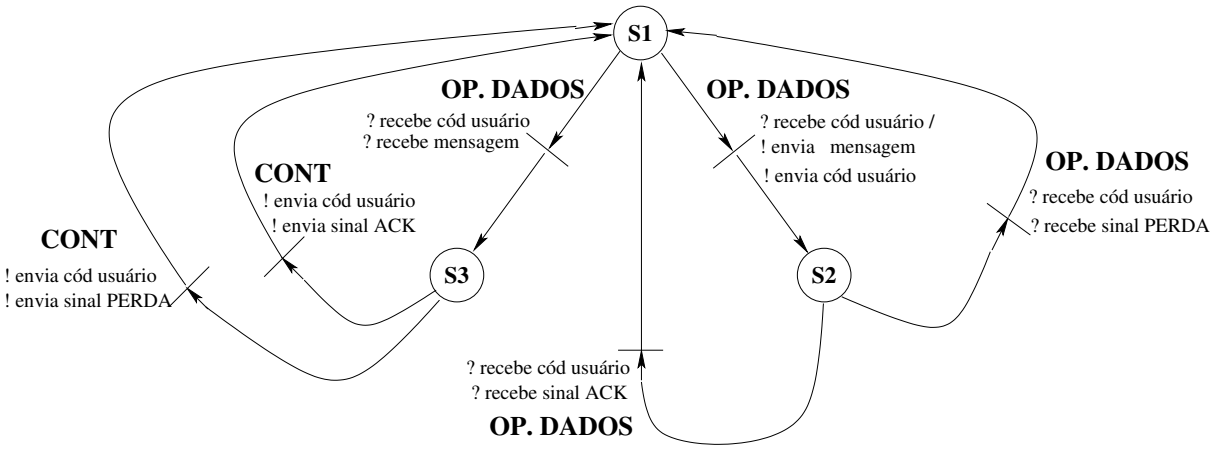

Figura 5.1: Modelo adotado para descrever as funcionalidades do protocolo.

A próxima seção apresenta em detalhes as descrições VHDL de cada uma das transições do protocolo modelado nesta seção. Cada uma das transições será comentada levando em consideração as características da linguagem VHDL, pois ressalta-se que o processo de síntese de *hardware* é muito mais complexo do que a síntese de *software* a partir da linguagem C.

5.2.3 Detalhamento das Transições para Descrições em Hardware

O modelo de MEF, conforme já discutido, é um dos mais apropriados para descrever o comportamento de protocolos, além de ser uma ferramenta familiar ao projetista. Na seção anterior, foram apresentadas algumas adaptações a este modelo, visando permitir a inserção de certas características de *hardware* na descrição em alto nível do sistema. A finalidade principal desta seção é apresentar um detalhamento das transições do protocolo sob o ponto de vista da síntese de *hardware*. Baseado no modelo descrito na figura 5.1, cada transição é detalhada para posterior especificação em VHDL.

Transição de Envio de MSG

A figura 5.2 trata a transição como uma "caixa preta" e ilustra com detalhes todos os sinais necessários para se atingir o comportamento desejado. A transição responsável pelo envio de mensagem inicialmente interage com o usuário da aplicação recebendo um sinal indicando o desejo de se enviar dados. Este sinal será utilizado como entrada para a caixa preta e a partir daí os sinais de sincronização atuarão de modo a receber a mensagem do usuário e armazenar na transição.

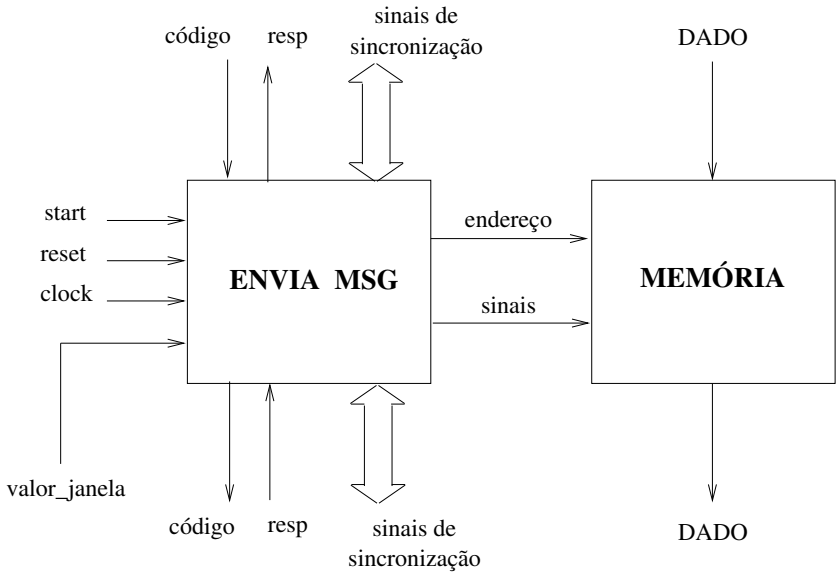

Figura 5.2: Detalhamento da transição envia mensagem.

Conforme discutido anteriormente, em um projeto de circuitos integrados existe a separação da parte lógica e da parte encarregada de armazenar dados. Este fato pode ser melhor visualizado na figura 5.2. O dado oriundo do usuário é armazenado na memória para posterior envio. Paralelamente ao armazenamento dos dados vindos do usuário, a transição já envia um sinal à outra entidade avisando que existem dados a transmitir. Quando a outra entidade retorna um sinal positivo, ocorre então a transmissão de dados regida pelos sinais de sincronização. Esta transmissão é efetuada com o valor da janela de congestionamento ajustado pelo sistema.

Transição de Recebimento de ACK

A figura 5.3 ilustra com detalhes a transição responsável pelo recebimento do sinal de ACK. Quando a transição recebe um aviso de ACK é porque a mensagem foi enviada corretamente e recebida pela outra entidade corretamente. Neste caso, a transição pode atualizar o valor da janela de congestionamento para que o emissor utilize um valor maior nas futuras transmissões. Esta atualização é efetuada somando-se ao valor atual uma constante previamente determinada. Esta operação será realizada internamente e a transição devolverá o novo valor da janela de congestionamento.

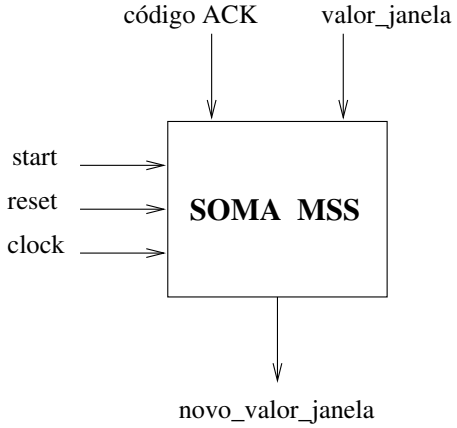

Figura 5.3: Detalhamento da transição recebimento de ACK.

Transição de Recebimento de sinal de PERDA

A figura 5.4 ilustra com detalhes a transição responsável pelo recebimento do sinal de PERDA. Quando a transição recebe um aviso de PERDA é porque a mensagem não foi recebida pela outra entidade ou foi recebida com erros ou existem sinais de congestionamento na rede. Neste caso, a transição diminuirá o valor da janela de congestionamento para que não haja sobrecarga na rede nas próximas transmissões. Esta atualização é efetuada dividindo-se o valor atual da janela por dois. Esta operação será realizada internamente e a transição devolverá o novo valor da janela de congestionamento.

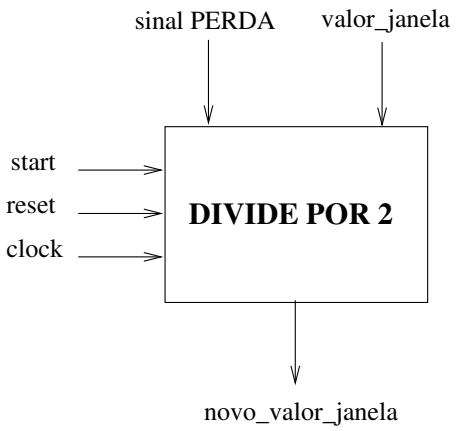

Figura 5.4: Detalhamento da transição recebimento de sinal de PERDA.

Transição de Recebimento de MSG

A figura 5.5 ilustra com detalhes a transição responsável pelo recebimento de mensagens. A transição inicialmente recebe um sinal da entidade transmissora indicando que existem dados a serem enviados. Quando a transição estiver pronta para receber os dados ela envia um sinal de resposta. A partir daí os sinais de sincronização atuarão de modo a receber a mensagem enviada pela outra entidade e armazenar na transição.

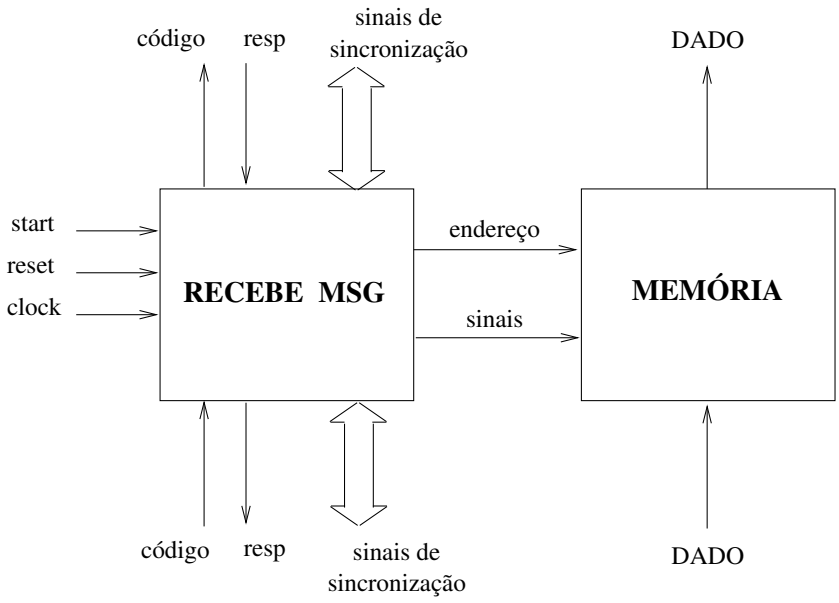

Figura 5.5: Detalhamento da transição recebe mensagem.

O dado oriundo do transmissor é armazenado na memória para posterior envio ao usuário. Paralelamente ao armazenamento dos dados vindos do transmissor, a transição já envia um sinal ao usuário da aplicação avisando que existem dados recebidos e armazenados. Estes dados serão repassados ao usuário quando ele sinalizar a transição.

Transições de Envio de ACK e Envio de sinal de PERDA

A figura 5.6 ilustra com detalhes as transições responsáveis pelo envio de ACK e envio de sinal de PERDA respectivamente. Estas transições foram classificadas como transições de controle no modelo adotado na seção anterior. Este fato se justifica pois essas transições não efetuam nenhuma operação sobre dados, por isso elas são inclusive mais simples. Posteriormente, na etapa de refinamento, elas poderão ser incorporadas por outras transições ou aglutinadas numa transição única dependendo dos resultados do processo de particionamento.

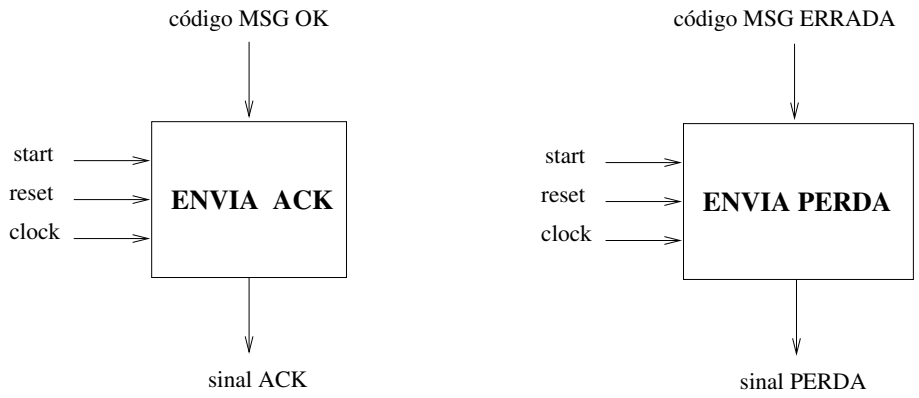

Figura 5.6: Detalhamento das transições envio de ACK e sinal de PERDA.

5.2.4 Síntese de Hardware e Síntese de Software

Após a etapa de detalhamento das diversas transições do protocolo, ocorre a etapa de síntese de *hardware* e *software*. Cada transição do modelo é descrita em VHDL e C visando sua implementação. Após estas implementações, obtem-se uma tabela contendo os diferentes valores de atraso/desempenho e custo para cada tran-

sição do protocolo. Estes valores serão utilizados pelo processo de particionamento para a avaliação da qualidade de diferentes alternativas de particionamento.

Uma das vantagens da utilização do modelo descrito anteriormente é que uma grande parte da comunicação existente entre as diversas entidades já foi levada em consideração. Isto é importante para a descrição em VHDL das entidades e, posteriormente, será importante para a etapa da síntese da comunicação. Na etapa de síntese da comunicação, a tarefa de incluir as operações necessárias para se interconectar as diversas entidades será facilitada. Como uma grande parte destas operações já foi considerada, é necessário somente se ater a alguns detalhes de implementação, principalmente da comunicação entre o *hardware* e o *software*.

A tabela 5.1 apresenta todos os valores obtidos para as implementações em *hardware* das diversas transições do protocolo utilizando a ferramenta Synopsys. Nesta tabela, as transições Envia MSG e Recebe MSG foram implementadas sem a parte referente à memória, ou seja, os valores de atraso e área se referem somente à parte de controle. Essas transições são bem parecidas do ponto de vista do número de sinais e forma de tratamento dessas sinalizações, por isso as medidas de área e *clock* são próximas. Analisando a tabela, ve-se que as transições Envia ACK e Envia PERDA são as que apresentam os menores valores de área e *clock*, o que já era de se esperar pelo detalhamento do funcionamento das mesmas na seção anterior. As transições Recebe ACK e Recebe PERDA possuem valores maiores de *clock* e área quando comparadas com as transições Envia ACK e Envia PERDA por terem que realizar operações sobre a variável de janela de congestionamento.

A tabela 5.2 apresenta as medidas de atraso e área para as transições Envia MSG e Recebe MSG em diferentes formas de implementação. Primeiramente, implementou-se a memória de forma direta por instanciação de módulos de memória previamente descritos nas bibliotecas do Synopsys. Esses valores correspondem às transições com "memória sintetizada". Neste caso, a ferramenta sintetiza a memória a partir de um conjunto de registradores, não otimizando assim sua estrutura, e por conseguinte, obtendo o maior *clock* e a maior área de todas as implementações. A segunda implementação utiliza uma memória previamente sintetizada e alocada na

Transição	Clock máximo	Atraso total	Frequência de operação	Área mm^2
Envia MSG	7,3 ns	43,8 ns	22,8 MHz	0,202
Recebe MSG	7,3 ns	43,8 ns	22,8 MHz	0,202
Envia ACK	4,7 ns	4,7 ns	212,7 MHz	0,017
Recebe ACK	5,8 ns	5,8 ns	172,4 MHz	0,148
Envia PERDA	4,7 ns	4,7 ns	212,7 MHz	0,016
Recebe PERDA	5,6 ns	5,6 ns	178,6 MHz	0,096

Tabela 5.1: Medidas de atraso e área para cada transição do protocolo em ASIC.

biblioteca de células do Synopsys. A esta implementação chamou-se "transição com memória da biblioteca". As medidas de atraso e área totais foram calculadas, de forma aproximada, ao juntar-se a memória à parte de controle já sintetizada. Conforme já mostrado, a parte de controle corresponde às implementações das transições sem memória, ilustrada nas duas últimas linhas da tabela 5.2.

A partir desta tabela, pode-se concluir que ao utilizar a ferramenta Synopsys para a síntese de circuitos lógicos de transições que exijam armazenamento de dados é mais vantajoso primeiramente sintetizar a parte de controle desta transição isoladamente. Em seguida, obtem-se a memória de tamanho desejado da biblioteca de células da ferramenta e une-se as duas partes.

A tabela 5.3 apresenta todos os valores obtidos para as implementações em *hardware* das diversas transições do protocolo utilizando a ferramenta ALTERA.

As transições com maior *clock* e atraso são as que manipulam mensagens através de acesso à memória. Novamente, as transições Envia ACK e Envia PERDA são as mais simples, por conseguinte as mais rápidas. A transição Recebe ACK manipula a variável janela de congestinamento utilizando um somador, por isso seu *clock* é maior. Já a transição Recebe PERDA realiza uma divisão na variável janela de congestinamento, e o baixo valor de *clock* se deve ao esquema adotado para implementar esta operação, provavelmente através de um *shift-register*.

Transição	Clock máximo	Atraso total	Frequência de operação	Área mm^2
Envia MSG (mem. sintetizada)	10 ns	60 ns	16,7 MHz	8,751
Recebe MSG (mem. sintetizada)	10 ns	60 ns	16,7 MHz	8,751
Envia MSG (mem. da biblioteca)	8,5 ns	51 ns	19,7 MHz	4,510
Recebe MSG (mem. da biblioteca)	8,5 ns	51 ns	19,7 MHz	4,510
Envia MSG (parte controle)	7,3 ns	43,8 ns	22,8 MHz	0,202
Recebe MSG (parte controle)	7,3 ns	43,8 ns	22,8 MHz	0,202

Tabela 5.2: Medidas de atraso e área das transições que tratam mensagens em ASIC.

Transição	Clock máximo	Atraso total	Frequência de operação
Envia MSG	50 ns	300 ns	3,3 MHz
Recebe MSG	50 ns	300 ns	3,3 MHz
Envia ACK	7 ns	7 ns	142,8 MHz
Recebe ACK	17 ns	17 ns	58,8 MHz
Envia PERDA	7 ns	7 ns	142,8 MHz
Recebe PERDA	7 ns	7 ns	142,8 MHz

Tabela 5.3: Medidas de atraso para cada transição do protocolo em PLD.

Os resultados destas tabelas mostram uma nítida vantagem em relação à velocidade de operação do projeto utilizando *standard cells*, em relação à implementação utilizando PLDs. Por outro lado deve ser notado que a implementação em PLDs é imediata e o sistema pode ser reprogramado sem a necessidade de uma nova rodada de fabricação, como no caso das *standard cells*.

Um dos objetivos principais deste trabalho consiste em se buscar um método eficiente de descrever protocolos em VHDL visando sua síntese, além disso, buscar inserir em um nível mais abstrato de projeto os detalhes pertinentes às características específicas de *hardware*. Por outro lado, é preciso fornecer ao processo de particionamento dados de atraso/desempenho e custo de implementações em *software*. A tabela 5.4 apresenta valores arbitrados para o desempenho das transições do protocolo TCP em *software*. Os valores das implementações em ASIC e PLD foram tomados como base de cálculo. Vale a pena ressaltar que os valores de desempenho de implementações em *software* variam de acordo com o ambiente utilizado.

Transição	**Atraso total**	**Frequência de operação**
Envia MSG	1000 ns	1 MHz
Recebe MSG	1000 ns	1 MHz
Envia ACK	40 ns	25 MHz
Recebe ACK	60 ns	16,7 MHz
Envia PERDA	40 ns	25 MHz
Recebe PERDA	40 ns	25 MHz

Tabela 5.4: Medidas de atraso para cada transição do protocolo em *software*.

A tabela 5.5 apresenta os valores referentes ao custo de implementação em SW de cada transição do protocolo TCP. O custo de *software* foi ignorado a partir da suposição que existe memória disponível o suficiente em nosso sistema para acomodar todas as descrições. O custo das implementações em PLD se referem ao custo das placas de desenvolvimento. O custo das implementações em ASIC se referem à área de cada transição associada a um custo arbitrado de fabricação.

Transição	Custo PLD	Custo ASIC
Envia MSG	100	500
Recebe MSG	100	500
Envia ACK	75	200
Recebe ACK	75	350
Envia PERDA	75	200
Recebe PERDA	75	300

Tabela 5.5: Medidas de custo para cada transição do protocolo em *software*.

As placas usadas para implementar as transições Envia MSG e Recebe MSG possuem uma maior capacidade de integração, justificando seu maior custo. O custo das implementações em ASIC levaram em consideração a área de cada transição, quanto maior a área, maior seu custo. Arbitrou-se um custo de fabricação, pois este custo pode variar bastante dependendo da quantidade a ser produzida.

Esses dados serão encaminhados ao processo de particionamento e ajudarão nos cálculos das aptidões das diversas possibilidades de partições. As partes que serão implementadas em *software* e as partes que serão implementadas em *hardware* serão arbitradas a partir da análise deste conjunto de medidas mostrado.

5.2.5 Particionamento HW/SW

O processo de particionamento determina efetivamente quais partes do protocolo serão implementadas em *hardware* e quais partes serão implementadas em *software*. A primeira etapa deste processo consiste em se modelar as funcionalidades do protocolo no modelo aceito pelo Tangram-II. O modelo especificado na figura 5.1 é adaptado ao modelo do Tangram-II somente acrescentando-se um estado de PERDA e uma transição associada. Esta alteração foi necessária pois a análise de desempenho deste protocolo utiliza a probabilidade de perda de mensagens como parâmetro para se calcular o desempenho total do protocolo. A figura 5.7 ilustra o

modelo utilizado como base para a construção do modelo inserido no Tangram-II.

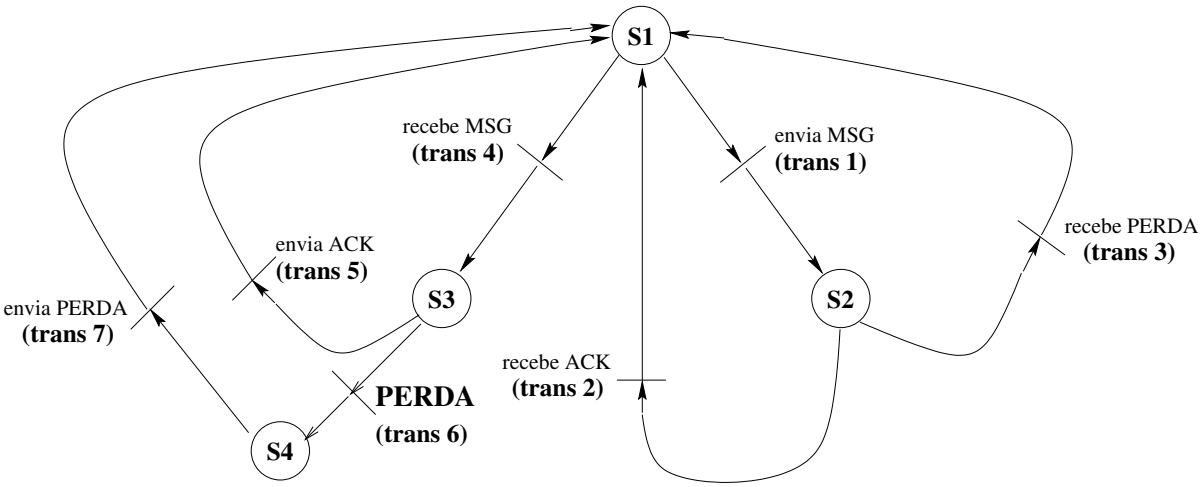

Figura 5.7: Modelo utilizado pela ferramenta de análise de desempenho.

Após a construção do modelo do protocolo na ferramenta Tangram-II, utiliza-se o modelo matemático e as suposições propostas em Mathis [54] para aproximar a curva *cwnd versus RTT* por um gráfico com a forma de um *dente de serra* como é mostrado na figura 5.8, onde *cwnd* é o tamanho da janela de congestionamento do emissor em número de pacotes e *RTT* é o *round trip time*.

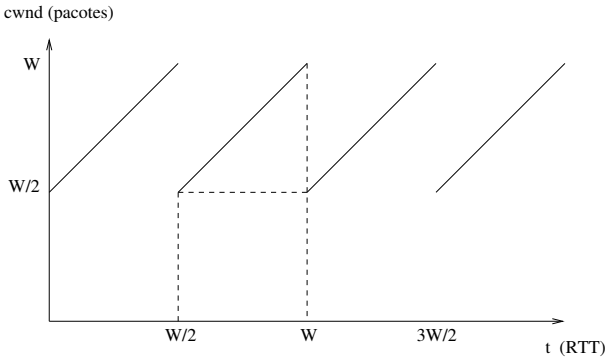

Figura 5.8: cwnd versus RTT com perda periódica.

Analisando-se a figura 5.8 pode-se determinar [54] o tamanho da janela de congestionamento, em número de pacotes, em função da probabilidade de perda, a saber:

$$BW * RTT/MSS = C/\sqrt{p} \tag{5.1}$$

Onde BW é a banda passante, MSS é o tamanho máximo de segmento, C é uma constante de proporcionalidade e p é a probabilidade de perda. A determinação dos parâmetros do protocolo é realizada pelo AG aproximando a curva $(BW * RTT)/MSS$ *versus perda*, traçada utilizando-se a equação 5.1, com $C = 1$, que pode ser visualizada na figura 5.9. A expressão $BW * RTT/MSS$ é uma **estimativa** do tamanho médio da janela de congestionamento [54]. Deve ser observado que o gráfico está em escala logarítmica.

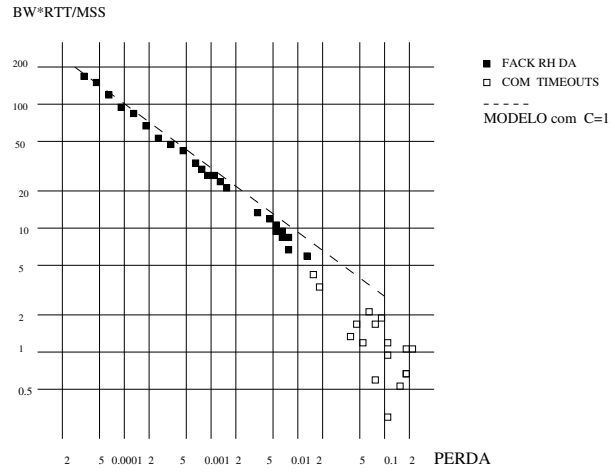

Figura 5.9: Curva Janela versus *perda* para o modelo matemático dado pela equação 5.1.

Foram utilizados 10 pontos para aproximar a curva do modelo e se obter os parâmetros desejados. Foram realizados experimentos com uma população de 100 indivíduos e utilizou-se como critério de parada do algoritmo genético o limite de 10 gerações. A taxa do evento PERDA (λ_6) foi variada (abscissa da curva) e a função objetivo foi calculada baseada na diferença entre o tamanho da janela obtido e o tamanho da janela desejado.

Obteve-se um erro mínimo para a função objetivo (considerando apenas o desempenho) igual a aproximadamente 1,7% para cada um dos pontos escolhidos. Os resultados obtidos são apresentados na tabela 5.6. O pequeno erro no valor das

janelas, fornecido pelo AG, se deve ao valor da constante C, com perdas aleatórias seu valor é um pouco menor do que 1 [54].

Evento	Taxa (1/seg.)
Envia_msg (λ_1)	1078,67
Recebe_ack (λ_2)	803,90
Recebe_perda (λ_3)	46,17
Recebe_msg (λ_4)	1
Envia_ack (λ_5)	984,02
Envia_perda (λ_7)	116,41

Tabela 5.6: Taxas para os eventos do TCP "Congestion Avoidance" obtidas pelo AG.

Observe que, pelo modelo, em média $1/p$ pacotes são enviados antes da ocorrência de uma perda. Utilizando-se as taxas da tabela 5.6, aplica-se o critério de partição descrito na seção 4.3.4 e calcula-se o produto $\lambda_i * \pi$ para cada evento do protocolo. Os resultados obtidos são apresentados na tabela 5.7.

Evento	valor
$\lambda_1 * \pi_1$	1,76
$\lambda_2 * \pi_2$	1,04
$\lambda_3 * \pi_3$	0,013
$\lambda_4 * \pi_4$	0,99
$\lambda_5 * \pi_5$	1,71
$\lambda_7 * \pi_7$	0,23

Tabela 5.7: Valores obtidos para o produto $\lambda_i * \pi_j$.

De acordo com os resultados acima, as operações do protocolo correspondentes aos eventos Envia_msg (λ_1) e Envia_ack (λ_5) devem ser implementadas em HW enquanto que aquelas correspondentes aos eventos Recebe_perda (λ_3) e Envia_perda (λ_7) devem ser implementadas em SW.

Com relação aos eventos Recebe_ack (λ_2) e Recebe_msg (λ_4), como os valores

são próximos e estão praticamente na média, deve-se analisar o modelo. Observa-se que os eventos Envia_msg, Envia_ack, Recebe_msg e Recebe_ack são aqueles que correspondem à transmissão de pacotes e ao recebimento de ACK's quando não ocorrem perdas na transmissão. É como se esses eventos correspondessem ao *caminho crítico do protocolo*, isto é, os eventos pelos quais o protocolo passa o maior número de vezes quando em estado estacionário. Logo, os eventos que obtiveram valores indefinidos, devem ser implementados em HW por pertencerem ao caminho crítico do protocolo.

Portanto, de acordo com os cálculos da ferramenta de análise de desempenho e do AG e utilizando o critério de partição demonstrado, as operações do protocolo correspondentes aos eventos Envia_msg, Recebe_ack, Envia_ack e Recebe_msg devem ser implementadas em HW enquanto que aquelas correspondentes aos eventos Recebe_perda e Envia_perda devem ser implementadas em SW.

Esta é a primeira partição que será avaliada em conjunto com as medidas de atraso e custo de área fornecidas pelo Synopsys e pelo ALTERA. Caso o custo do HW esteja dentro das especificações e as taxas encontradas possam ser atendidas pelos circuitos sintetizados, uma solução foi encontrada. Caso contrário, uma nova população é gerada para que uma nova otimização seja realizada.

5.2.6 Avaliação da Partição Escolhida

Tendo em vista os resultados fornecidos pelas ferramentas Synopsys e ALTERA mostrados nas tabelas contidas na seção 5.2.4, a partição proposta é avaliada. Levando-se em conta, por exemplo, um custo de área especificado pelo projetista de 2 mm^2, pode-se observar que as transições Envia_msg e Recebe_msg extrapolam os requisitos de área desejados. Sugere-se, então, que estas transições sejam implementadas em PLD. Apesar das medidas de atraso da implementação em PLD serem maiores do que em *standard cells*, elas atendem às taxas calculadas pelo AG. As transições Envia_ack e Recebe_ack serão implementadas em *standard cells*, pois as medidas de área atendem ao exigido pelo projetista e as medidas de atraso atendem

às taxas calculadas pelo AG. Por fim, as transições Envia_perda e Recebe_perda serão implementadas em *software*. Esta sugestão de implementação pode ser melhor avaliada utilizando-se novamente a ferramenta Tangram-II, conforme mostrada na próxima seção.

5.2.7 Análise do Desempenho da Partição Escolhida

A análise do desempenho da partição é obtida utilizando-se as medidas fornecidas pelas ferramentas Synopsys e ALTERA. A tabela 5.8 mostra a soma dos erros obtidos para cada ponto da curva usada na otimização para as diversas possibilidades de implementação do protocolo. Pôde-se verificar que não existe um ganho de desempenho quando as transições são implementadas em HW. Somente o erro em relação à curva utilizada para a otimização diminui à medida que a velocidade de processamento aumenta. Isso se deve ao fato do modelo utilizado supor que a janela de congestionamento é completamente dependente da probabilidade de perda de pacotes no meio (vide Eq. 5.1).

Implementação	**ERRO**
SW	0,000360
HW (SC)	0,000019
HW (PLD)	0,000104
HW/SW (SC/SW)	0,000036
HW/SW (PLD/SW)	0,000117

Tabela 5.8: Erros obtidos para cada implementação.

Os resultados obtidos estão coerentes com os resultados esperados para cada tipo de implementação: o erro maior para a implementação em SW, o menor para a implementação em *standard cells* e valores intermediários para as partições HW/SW e para a implementação em PLD. Sob as suposições realizadas pelo modelo, a equação 5.1 pode ser vista como um limite superior para o desempenho do protocolo, ou seja:

$$BW * RTT/MSS < C/\sqrt{p} \tag{5.2}$$

O **ERRO** obtido, mostrado na tabela 5.8 é calculado em relação a esse limite. Pode-se observar que os resultados obtidos, ou seja, os valores dos erros obtidos para cada tipo de implementação, estão coerentes com os resultados esperados: a implementação somente em SW fornece o maior erro e a implementação somente em *standard cells* fornece o menor erro. As implementações com PLDs e as partições PLD/SW e SC/SW fornecem valores intermediários.

5.2.8 Refinamento

A etapa de refinamento é responsável em analisar o resultado gerado pelo processo de particionamento e identificar, se possível, pontos que podem ser melhorados. O resultado proposto pelo algoritmo de particionamento pode ser visualizado na figura 5.10. Nesta figura, estão presentes todas as transições do protocolo e apontadas como elas serão implementadas, em ASIC, PLD ou *software*.

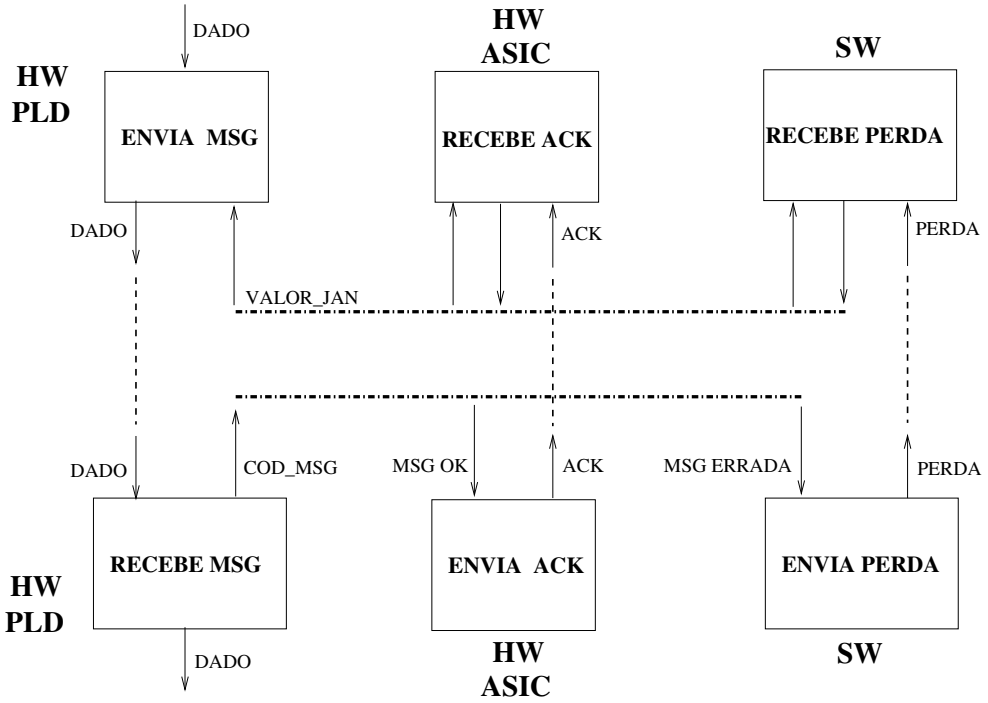

Figura 5.10: Particionamento proposto.

Um aspecto que deve ser levado em consideração nesta etapa de refinamento é a minimização da comunicação entre os diferentes módulos. Esta comunicação, quando excessiva, pode acarretar uma diminuição do desempenho geral do sistema. Portanto, um fator importante consiste em se identificar "pontos críticos" de comunicação entre estes sub-sistemas. Analisando a figura 5.10 juntamente com os dados da tabela 5.1 e relembrando do detalhamento das transições apresentadas na seção 5.2.3 podemos notar que as transições 5 e 7 (envia ACK e envia PERDA) são muito simples e podem ser "englobadas" pela transição 4 (recebe MSG).

Na verdade, a inserção das funcionalidades das transições 5 e 7 na transição 4 não acarretará sobrecarga excessiva. A transição 4 somente precisará analisar o pacote que receber e checar se está correto ou não, e então enviar um sinal de ACK ou PERDA. Isto pode ser feito através da análise de um número de seqüência por exemplo.

A nova disposição dos módulos nos domínios de *hardware* e *software* é ilustrado na figura 5.11. Nota-se que a transição 4 (recebe MSG) possui agora mais dois sinais de saída, respectivamente, sinal de ACK e sinal de PERDA. Esta nova disposição minimizou a comunicação entre os diferentes módulos. É interessante notar que, justamente as transições classificadas como transições *de controle* no modelo de MEF adaptado foram "reprojetadas".

5.2.9 Integração do Sistema

A etapa de integração do sistema, que no atual estágio do projeto consiste basicamente na síntese da comunicação, será responsável em implementar o esquema de troca de informações entre os diversos módulos de *hardware* e de *software*.

Esta etapa se torna bastante simplificada devido ao esquema adotado nesta metodologia. A forma como o protocolo foi modelado inicialmente contribuiu para este fato. A inserção de sinais de controle e de sincronização na modelagem em alto nível permitiu que, na descrição das transições em VHDL, muitos dos detalhes pertinentes à comunicação já pudessem ser implementados.

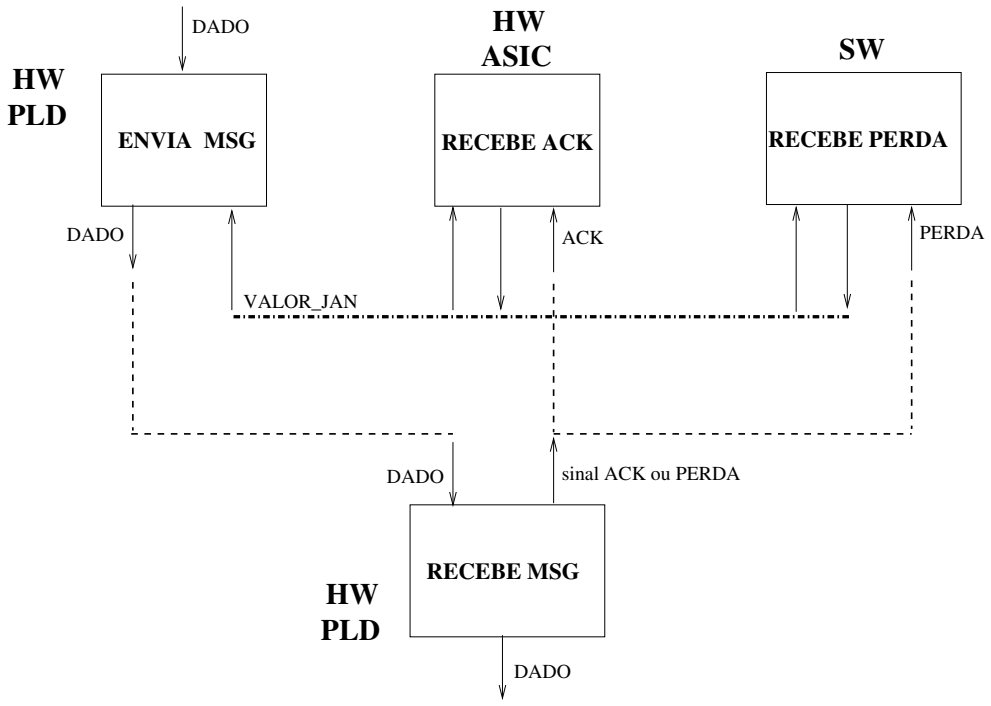

Figura 5.11: Nova disposição dos módulos após a etapa de refinamento.

Esta etapa deverá se preocupar, basicamente, em implementar os esquemas de comunicação dos módulos de *software*. Esquemas com *sockets* ou RPC (*Remote Procedure Call*) podem ser implementados. Quanto ao *hardware*, alguns detalhes de conflitos de acesso à barramentos devem ser resolvidos. A figura 5.12 ilustra como o protocolo se comporta ao final do ciclo de projeto.

5.3 Protocolo de Controle de *Handoff* para uma rede ATM sem fio

A expansão das telecomunicações, principalmente no uso de celulares, vem reforçando o interesse por redes sem fio. Novas e complexas aplicações para este tipo de redes aparecem a cada instante juntamente com novos tipos de aparelhos. Além de celulares, a utilização de redes sem fio é atrativa para uso em computação. Computadores móveis ocupam cada vez mais espaço no mercado internacional e a utilização da tecnologia ATM vem de encontro à necessidade de se obter taxas de transmissão

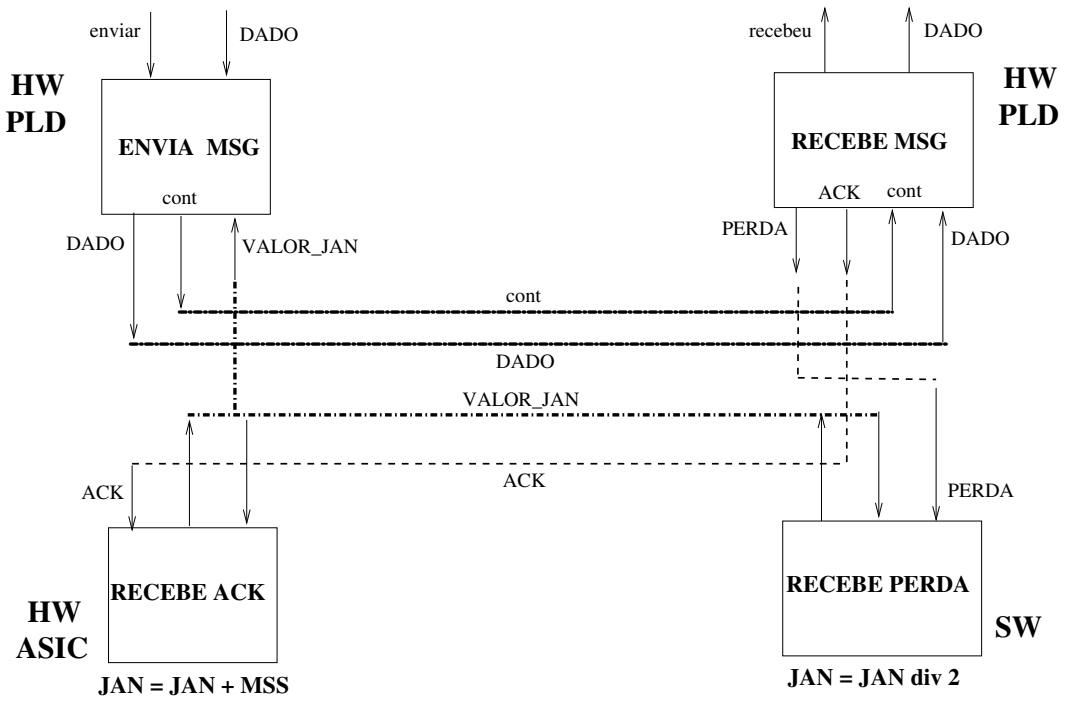

Figura 5.12: Disposição final dos módulos do protocolo.

elevadas.

Em redes móveis, um problema comumente encontrado se deve a passagem de um terminal móvel de uma célula para outra durante a conversação ou troca de mensagens. Esta troca de células denomina-se *handoff* e pode resultar em uma simples perda de alguns pacotes ou até mesmo a perda da conexão.

Nesta seção, a metodologia apresentada é aplicada a um protocolo de controle de *handoff* para redes ATM sem fio [56, 57] e o desempenho do protocolo é analisado utilizando-se a seqüência de primitivas fornecidas em Acharya [58]. Nesse tipo de protocolo, a localização de um terminal em relação à rede não pode ser determinada apenas através de seu endereço. Esquemas adicionais de endereçamento são necessários para localizar e rastrear os terminais móveis, juntamente com modificações apropriadas no processo de iniciação da conexão. Um protocolo de controle de *handoff* eficiente estabelece novas rotas dinamicamente, ao invés de estabelecer toda a conexão novamente.

5.3.1 Descrição das Funcionalidades do Protocolo

A figura 5.13 apresenta a especificação do protocolo dado por uma rede de Petri condição-ação. Quando um terminal móvel identifica uma perda considerável em seu sinal, ele automaticamente conclui que necessita de uma conexão com uma nova estação base.

Primeiramente, ele envia uma primitiva de *handoff_request (select cos)* para a estação base a qual está conectado. Esta estação base envia, então, uma primitiva *handoff_request (setup)* para uma outra estação base que poderá acolher este terminal móvel. Neste ponto, as duas estações trocam algumas primitivas e a estação base inicial avisa ao terminal móvel (*handoff_response*) da possível troca de célula.

O terminal envia um *handoff_confirm* para a estação inicial que responde com um *handoff_confirm_complet* confirmando a nova configuração. Por fim, o terminal envia um *handoff_join* à nova estação base que responde com a primitiva *handoff_join_complet* estabelecendo a nova configuração.

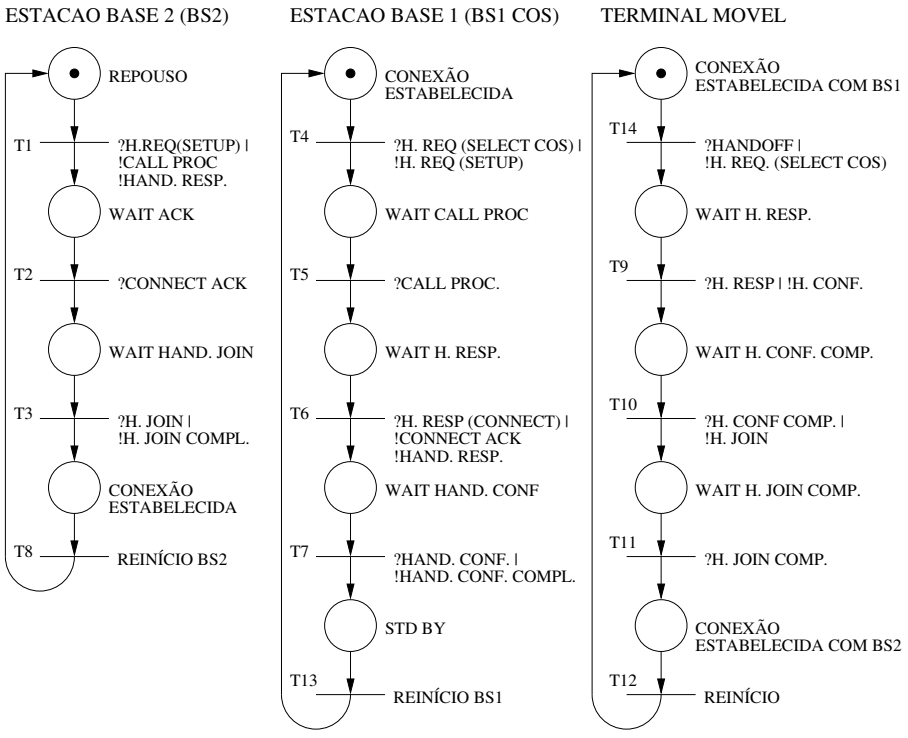

Figura 5.13: Rede de Petri condição-ação do Protocolo de controle de *Handoff*.

5.3.2 Síntese de Hardware e Síntese de Software

Cada transição do modelo é descrita em VHDL e C visando sua implementação. Após estas implementações, obtem-se uma tabela contendo os diferentes valores de atraso/desempenho e custo para cada transição do protocolo. Estes valores serão utilizados pelo processo de particionamento para a avaliação da qualidade de diferentes alternativas de particionamento.

A tabela 5.9 apresenta todos os valores obtidos para as implementações em *hardware* das diversas transições do protocolo utilizando a ferramenta Synopsys.

Transição	**Atraso total**	**Frequência de operação**	**Área mm^2**
1	20 ns	50 MHz	0,150
2	5 ns	200 MHz	0,015
3	5 ns	200 MHz	0,023
4	16 ns	62,5 MHz	0,223
5	5 ns	200 MHz	0,015
6	20 ns	50 MHz	0,130
7	5 ns	200 MHz	0,023
8	5 ns	200 MHz	0,015
9	5 ns	200 MHz	0,024
10	5 ns	200 MHz	0,015
11	5 ns	200 MHz	0,015
12	5 ns	200 MHz	0,015
13	5 ns	200 MHz	0,015
14	16 ns	62,5 MHz	0,223

Tabela 5.9: Medidas de área e atraso para cada transição do protocolo em ASIC.

A tabela 5.10 apresenta todos os valores obtidos para as implementações em *hardware* das diversas transições do protocolo utilizando a ferramenta ALTERA.

A tabela 5.11 apresenta valores arbitrados para o desempenho das transições

Transição	Atraso total	Frequência de operação
1	60 ns	16,7 MHz
2	30 ns	33,3 MHz
3	30 ns	33,3 MHz
4	60 ns	16,7 MHz
5	30 ns	33,3 MHz
6	60 ns	16,7 MHz
7	30 ns	33,3 MHz
8	30 ns	33,3 MHz
9	30 ns	33,3 MHz
10	30 ns	33,3 MHz
11	30 ns	33,3 MHz
12	30 ns	33,3 MHz
13	30 ns	33,3 MHz
14	60 ns	16,7 MHz

Tabela 5.10: Medidas de área e atraso para cada transição do protocolo em PLD.

do protocolo de controle de *handoff* em *software*. Os valores das implementações em ASIC e PLD foram tomados como base de cálculo. Vale a pena ressaltar que os valores de desempenho de implementações em *software* variam de acordo com o ambiente utilizado.

Os valores referentes aos custos de implementação de cada transição do protocolo não foram levados em consideração. O custo de *software* foi ignorado a partir da suposição que existe memória disponível o suficiente em nosso sistema para acomodar todas as descrições. Os custos de implementação em PLD e ASIC também deixaram de ser levados em consideração, pois as transições deste protocolo são relativamente de simples implementação, e como dito anteriormente, as transições são dominadas por fluxo de controle. Como visto na implementação do protocolo TCP na seção anterior, as transições que foram classificadas como transições de controle, no final,

Transição	Atraso total	Frequência de operação
1	250 ns	4 MHz
2	120 ns	8,33 MHz
3	120 ns	8,33 MHz
4	250 ns	4 MHz
5	120 ns	8,33 MHz
6	250 ns	4 MHz
7	120 ns	8,33 MHz
8	120 ns	8,33 MHz
9	120 ns	8,33 MHz
10	120 ns	8,33 MHz
11	120 ns	8,33 MHz
12	120 ns	8,33 MHz
13	120 ns	8,33 MHz
14	250 ns	4 MHz

Tabela 5.11: Medidas de atraso para cada transição do protocolo em *software*.

acabaram sendo englobadas por uma transição de operação de dados, de modo a melhorar o custo da comunicação e da própria implementação total.

Esses dados serão encaminhados ao processo de particionamento e ajudarão nos cálculos das aptidões das diversas possibilidades de partições. As partes que serão implementadas em *software* e as partes que serão implementadas em *hardware* serão arbitradas a partir da análise deste conjunto de medidas mostrado.

5.3.3 Particionamento HW/SW

O processo de particionamento determina efetivamente quais partes do protocolo serão implementadas em *hardware* e quais partes serão implementadas em *software*. A primeira etapa deste processo consiste em se modelar as funcionalidades do pro-

tocolo no modelo aceito pelo Tangram-II. A figura 5.13 ilustra o modelo utilizado como base para a construção do modelo inserido no Tangram-II.

A medida de desempenho utilizada neste caso é a probabilidade de perda de pacotes em função da taxa de transmissão. O atraso ocasionado pelo *handoff* (T_d) fornece o tempo mínimo para se alterar a tabela de roteamento para múltiplos VCs. Esse tempo é diretamente proporcional à taxa de perda causada pelo procedimento de *handoff*. No modelo utilizado em Yuan [59], uma perda ocorre quando uma mudança na tabela de roteamento é realizada durante a transmissão de um fluxo de células dentro do mesmo pacote. Considere que o evento de *handoff* ocorre aleatoriamente dentro do intervalo $T = 1/R$, onde T é o tempo de transmissão de um pacote e R é a taxa de transmissão de pacotes. Portanto, a probabilidade de perda de pacotes P é dada por $P = T_d * R$. A figura 5.14 mostra o gráfico P versus R.

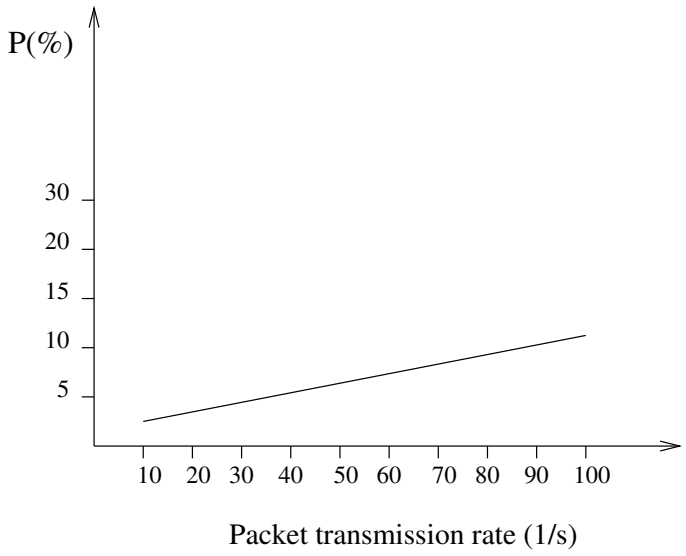

Figura 5.14: Probabilidade de Perda em função da taxa de transmissão.

Foram escolhidos 6 pontos da curva para se determinar os parâmetros do modelo, o tamanho da população é de 100 indivíduos e o número de gerações utilizado como critério de terminação é igual a 10. Esses valores foram determinados após vários testes com diferentes tamanhos de populações e gerações. Tais valores fornecem resultados com um bom grau de precisão. A função objetivo é a diferença entre

a probabilidade de perda obtida e a probabilidade de perda desejada, ou seja, os parâmetros do protocolo são extraídos aproximando-se o gráfico da figura 5.14. Os resultados são apresentados na tabela 5.12. O erro encontrado para a soma das diferenças entre as probabilidades desejadas e obtidas foi de 0,006.

Transição	**Taxa (1/seg.)**
t_1 (λ_1)	42,34
t_2 (λ_2)	60,12
t_3 (λ_3)	421,63
t_4 (λ_4)	344,91
t_5 (λ_5)	477,38
t_6 (λ_6)	397,72
t_7 (λ_7)	14,31
t_8 (λ_8)	156,63
t_9 (λ_9)	20,76
t_{10} (λ_{10})	8,43
t_{11} (λ_{11})	114,90
t_{12} (λ_{12})	338,55
t_{13} (λ_{13})	474,19
t_{14} (λ_{14})	68,03

Tabela 5.12: Taxas para os eventos do protocolo de *Handoff* obtidas pelo AG.

Utilizando as taxas da tabela 5.12 e as probabilidades em estado estacionário calculadas pelo Tangram-II, o mesmo critério de partição da seção 4.3.4 pode ser aplicado e o produto $\lambda_i * \pi_j$ calculado para cada um dos eventos (transições) do protocolo. Os resultados são mostrados na tabela 5.13.

As tarefas do protocolo relacionadas às transições t_4, t_5, t_6, t_7, t_{12} e t_{13} devem ser implementadas em HW porque as respectivas transições possuem um produto $\lambda_i * \pi_j$ maior, enquanto aquelas relacionadas às demais transições, com um produto menor, podem ser implementadas em SW. Essa é a primeira partição a ser avaliada em relação ao custo de implementação em HW.

Evento	Valor
$\lambda_1 * \pi_1$	1,52
$\lambda_2 * \pi_2$	3,05
$\lambda_3 * \pi_3$	5,22
$\lambda_4 * \pi_4$	97,61
$\lambda_5 * \pi_5$	14,31
$\lambda_6 * \pi_6$	27,75
$\lambda_7 * \pi_7$	70,32
$\lambda_8 * \pi_8$	0,20
$\lambda_9 * \pi_9$	1,17
$\lambda_{10} * \pi_{10}$	0,17
$\lambda_{11} * \pi_{11}$	1,88
$\lambda_{12} * \pi_{12}$	80,21
$\lambda_{13} * \pi_{13}$	13,60
$\lambda_{14} * \pi_{14}$	3,67

Tabela 5.13: Valores obtidos para o produto $\lambda_i * \pi_j$.

De acordo com os resultados obtidos e analisando-se o modelo da figura 5.13, as transições a serem implementadas em HW aceleram o processo de *handoff* porque são aquelas associadas às tarefas de mudança da tabela de roteamento na *cross-over switch* e ao restabelecimento da conexão do terminal móvel com a nova estação base. Isso indica que o comportamento do modelo aproxima o mecanismo real de protocolos de controle de *handoff* encontrados em diversas implementações de redes ATM sem fio.

5.3.4 Avaliação da Partição Escolhida

Tendo em vista os resultados fornecidos pelas ferramentas Synopsys e ALTERA mostrados nas tabelas contidas na seção 5.3.2, a partição proposta é avaliada. Levando-se em conta, por exemplo, um custo de área especificado pelo projetista de

1 mm^2, pode-se observar que todas as transições indicadas pelo processo de particionamento (t_4, t_5, t_6, t_7, t_{12} e t_{13}) podem ser implementadas em HW utilizando-se *standard cells* ou PLDs pois todas as medidas de área são menores do que 1 mm^2 e os atrasos associados satisfazem as taxas calculadas pelo AG.

Vale a pena ressaltar que, apesar das transições indicadas pelo processo de particionamento poderem ser implementadas em HW utilizando-se *standard cells* ou PLDs, existem características específicas para cada processo de implementação. Utilizando *standard cells*, as transições possuirão um desempenho mais significativo, entretanto, seu custo será maior. Por outro lado, utilizando PLD, o desempenho será menor que o encontrado no projeto em *standard cells*, apesar de ainda estar dentro da faixa calculada pelo AG e Tangram-II. Outra vantagem em um projeto PLD é o custo. Mas, talvez a maior vantagem deste tipo de projeto seja a possibilidade de reprogramação sem necessidade de uma nova rodada de fabricação, como acontece em um projeto utilizando ASIC.

5.3.5 Análise do Desempenho da Partição Escolhida

A figura 5.15 compara o desempenho do protocolo de *handoff* totalmente implementado em *software*, em *standard cells* e em PLDs (ALTERA) com o desempenho da partição HW/SW escolhida. Para isso, duas implementações da partição escolhida foram realizadas: a primeira, usando parte das transições implementadas em PLDs (aquelas designadas para *hardware*) e parte implementada em *software*. A segunda, usando parte das transições implementadas em *standard cells* e parte em *software*. O parâmetro de desempenho utilizado para comparar essas implementações é a probabilidade de perda em função da taxa de transmissão.

Pode-se observar que a probabilidade de perda na implementação utilizando *standard cells* é a menor de todas as taxas quando comparadas às outras implementações. Porém, o custo deste tipo de tecnologia é alto, devido a necessidade de se produzir um *layout* do circuito e enviá-lo a uma fábrica de circuitos para a produção dos integrados. A implementação em PLDs fornece perdas maiores que a implementação

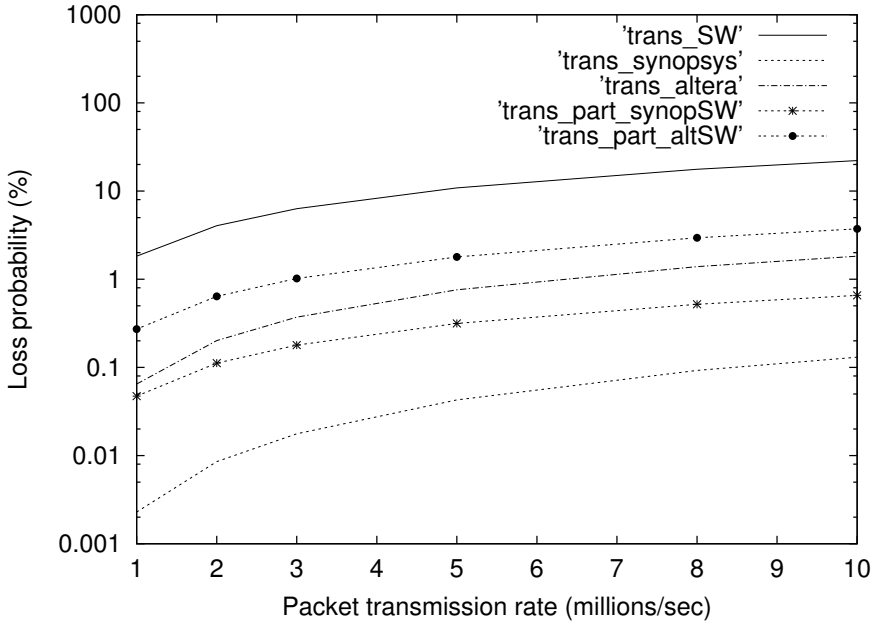

Figura 5.15: Comparação entre as diversas implementações.

em *standard cells*, porém menores do que as taxas encontradas na implementação em SW. Não obstante, implementações em PLDs são ferramentas importantes para prototipagem rápida de sistemas de comunicação a um custo baixo. Sistemas são criados em um chip reprogramável adquirindo uma placa PLD que pode ser acoplada ao computador. Possíveis modificações em seu sistema são efetuadas rapidamente através de programação. Por outro lado, modificações no projeto em *standard cells* necessitariam de uma nova rodada de fabricação.

A partição HW/SW utilizando PLDs para implementar a parte em HW fornece resultados melhores do que a implementação do protocolo todo em *software*. A partição HW/SW utilizando *standard cells* para implementar a parte em HW fornece resultados muito melhores do que a implementação do protocolo todo em *software* e melhores do que a partição HW/SW com PLDs. Porém, neste caso, a solução utilizando PLDs satisfaz os requisitos de desempenho com um custo menor do que a implementação em *standard cells*. Caso fosse necessário obter um desempenho ainda melhor em detrimento do custo do projeto, poderia se optar pela partição HW/SW utilizando *standard cells*.

5.4 Protocolo ARM - Active Reliable Multicast

O projeto de protocolos *multicast* confiáveis para a Internet é um problema difícil de se tratar. Esta dificuldade se deve à capacidade limitada da rede e do próprio emissor em responder a sinalizações de perdas. Pedidos de retransmissão simultâneos vindos de um grande número de receptores podem causar sobrecarga na rede através do problema conhecido como **implosão de NACKs**. O uso da otimização em *software* na elaboração desses protocolos nem sempre permite uma operação em alta velocidade. Mais importante do que nunca, a utilização de implementações em *hardware* para aumentar o desempenho destes projetos é necessária.

Nesta seção, a metodologia apresentada é aplicada ao projeto de um protocolo *multicast* confiável [60]. O protocolo ARM (*Active Reliable Multicast*) [61] tem como objetivo prover um esquema de recuperação de perdas para um grupo em grande escala. Com este intuito, ARM utiliza roteadores intermediários, denominados roteadores **ativos**, que protegem o emissor e a rede de tráfego desnecessário de NACKS e de pacotes de reparo. Esses roteadores realizam um processamento local que permite a recuperação de pacotes de dados perdidos sem a necessidade de retransmissão desse pacote para todo o grupo. Dessa forma, o problema de implosão de NACKs é evitado, a carga de retransmissões é distribuída e as retransmissões são restritas a um escopo local da rede, levando a uma queda significativa no consumo de banda passante.

Neste projeto, o desempenho do protocolo é analizado levando em consideração o comportamento do roteador ativo, em termos de tempo de processamento, em função do número de receptores por grupo para cada partição HW/SW determinada pelo algoritmo genético.

5.4.1 Descrição das Funcionalidades do Protocolo

O protocolo ARM tem como objetivo prover um esquema de recuperação de perdas para um grupo em grande escala. Com este intuito, ARM utiliza roteadores intermediários, denominados roteadores **ativos**, que protegem o emissor e a rede

de tráfego desnecessário de NACKS e de pacotes de reparo. Os roteadores ativos empregam três tipos de recuperação de perdas: supressão de NACKs duplicados, um esquema de recuperação local de perdas baseado nos roteadores ativos e transmissão *multicast* parcial, ou seja, o pacote de reparo é transmitido para um escopo reduzido e não para todo o grupo. A supressão de NACKs duplicados reduz o número de NACKs navegando em direção ao emissor e o tráfego que cruza os enlaces de gargalo da rede. A recuperação local de perdas reduz a latência fim-a-fim e distribui a carga de retransmissões. A transmissão *multicast* parcial reduz o tráfego na rede limitando o escopo de dados retransmitidos [61].

Roteadores ativos colocados em posições estratégicas armazenam dados de forma *best effort* para possíveis futuras retransmissões. Normalmente, esses roteadores são colocados imediatamente antes de enlaces onde ocorrem muitas perdas. Esse esquema permite aos nós-destino recuperarem-se rapidamente de perdas de pacotes de dados.

O protocolo ARM é do tipo *receiver-reliable*, ou seja, os receptores são responsáveis pela detecção da perda e por requisitar os pacotes perdidos, através de seu número de seqüência. Os receptores detectam perdas através do recebimento de um pacote com número de seqüência errado. Considera-se um cenário onde existe apenas um emissor e vários receptores no grupo *multicast*. O receptor envia um NACK ao emissor no momento em que é detectada uma perda. Múltiplos NACKs de diferentes receptores são armazenados e "fundidos" nos nós ativos ao longo da árvore *multicast*. Nós inativos simplesmente retransmitem o pacote em direção ao emissor. O emissor responde ao primeiro NACK recebido transmitindo um pacote de reparo para todos os participantes do grupo *multicast* e ignora NACKs subseqüentes para esse pacote por um determinado tempo.

Quando um roteador ativo recebe um NACK, indicando que um receptor detectou uma perda, ele retransmite o pacote solicitado, caso o mesmo esteja armazenado. Caso contrário o NACK é processado para se saber se será descartado, no caso de ser um NACK duplicado, ou enviado em direção ao emissor. Além disso, conforme mencionado anteriormente, os roteadores ativos retransmitem pacotes somente para

aqueles receptores que, previamente, os tenham requisitado (*multicast* parcial).

5.4.2 Modelagem do Protocolo

A figura 5.16 ilustra a modelagem das funcionalidades do protocolo através do modelo de MEF adaptado às características da síntese de *hardware*. Observa-se que o modelo consiste de seis estados: Trans_Envia (S1), Rot_Processa_Msg (S2), Rot_Envia (S3), Trans_Processa_Msg (S4), Recep_Processa_Msg (S5) e Recep_Envia (S6). O modelo possui também nove transições: Trans_Envia_Dado (t_1), Trans_Processa_NACK (t_2), Rot_Processa_Msg (t_3), Rot_Envia_Pct (t_4), Rot_Envia_NACK_Fonte (t_5), Rot_Descarta_NACK (t_6), Recep_Processa_Pct (t_7), Recep_Envia_NACK (t_8) e Recep_Pct_OK (t_{10}). Quando da implementação do modelo do protocolo na ferramenta Tangram-II é necessário a utilização de mais uma transição. Esta décima transição será utilizada somente com intuito de garantir o correto funcionamento da especificação do protocolo em Rede de Petri.

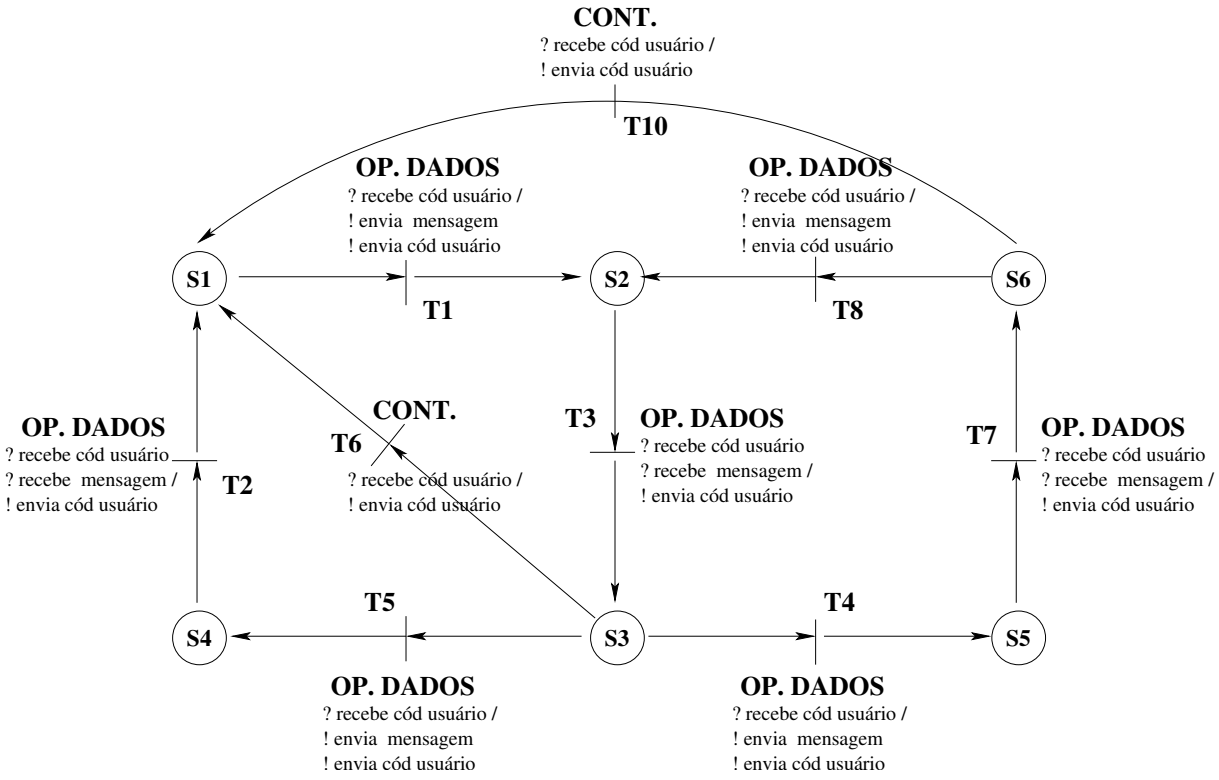

Figura 5.16: Modelo adotado para descrever as funcionalidades do protocolo.

As transições t_1 e t_2 são funções implementadas pelo transmissor. As transições t_3, t_4, t_5 e t_6 são implementadas nos roteadores ativos. O algoritmo do protocolo ARM é implementado nestes roteadores ativos. Nos diferentes receptores *multicast*, são implementadas as transições t_7, t_8 e t_{10}.

Ressalta-se na figura, a inclusão dos sinais de sincronização importantes na descrição destas transições em VHDL durante o processo de síntese de *hardware*. Esses sinais de sincronização possibilitam as trocas de mensagens entre os diversos módulos. A próxima seção explica detalhadamente o funcionamento desses sinais. Ressalta-se também, a classificação das transições em transições de controle e transições de operações de dados.

5.4.3 Detalhamento das Transições para Descrições em Hardware

A finalidade principal desta seção é apresentar um detalhamento das transições do protocolo sob o ponto de vista da síntese de *hardware*. Baseado no modelo descrito na figura 5.16, cada transição é detalhada para posterior especificação em VHDL.

Transição de Recebimento de MSG pelo Roteador

A figura 5.17 ilustra com detalhes a transição responsável pelo recebimento de mensagens pelo roteador e o respectivo tratamento dado a estas mensagens. A primeira providência a ser tomada consiste na inserção dos sinais de *start*, *clock* e *reset*. Estes sinais comandarão o funcionamento do circuito a ser sintetizado.

A operação se inicia a partir do sinal de *start* fazendo com que a transição indique que está pronta a receber dados através do sinal *podemandar_roteador*. O sinal de *start* provoca também a inicialização de diversas variáveis internas e diversos sinais de saída. A mensagem é recebida através dos sinais *msg_roteador_cod* e *msg_roteador_sdu*.

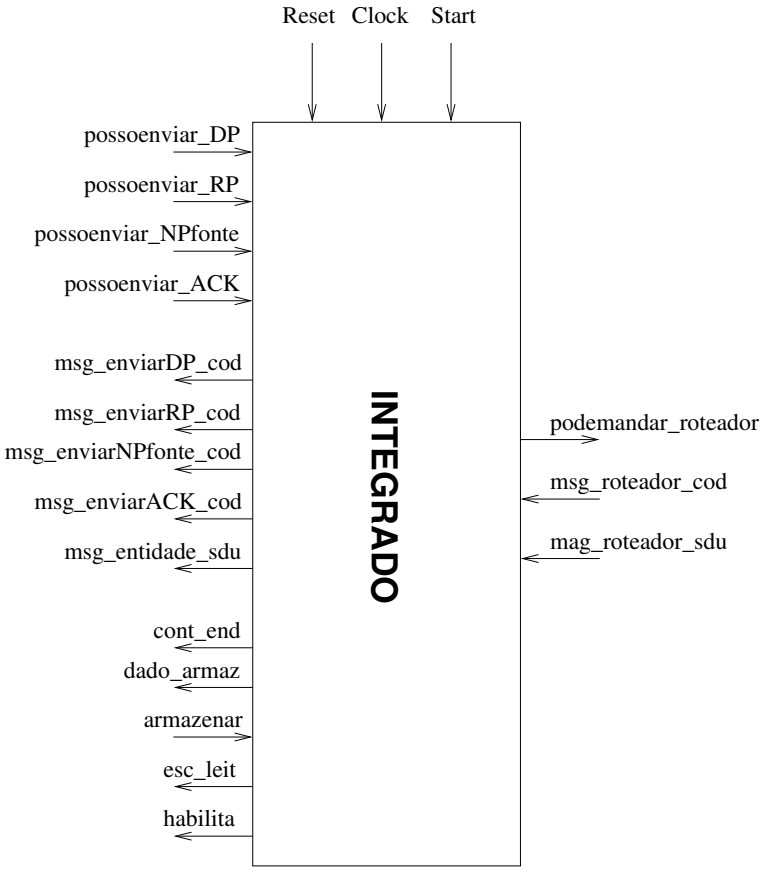

Figura 5.17: Detalhamento da transição de recebimento de mensagem pelo roteador.

Após o recebimento da mensagem, a transição inicia o procedimento de análise do pacote. Primeiramente, verifica-se o tipo da mensagem. De acordo com o tipo da mensagem, efetua-se o procedimento apropriado de acordo com [61]. Após os procedimentos serem executados, a mensagem é enviada ao respectivo módulo encarregado do envio da mensagem.

Transição de Envio de MSG pelo Roteador

A figura 5.18 ilustra com detalhes a transição responsável pelo envio de mensagens pelo roteador e o respectivo tratamento dado a estas mensagens. Ressalta-se, novamente, a inserção dos sinais de *start*, *clock* e *reset*. Estes sinais comandarão o funcionamento do circuito a ser sintetizado.

A operação se inicia a partir do sinal de *start* fazendo com que a transição indique

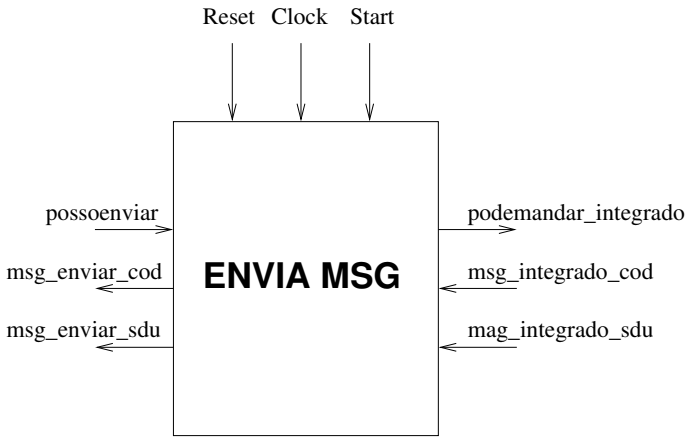

Figura 5.18: Detalhamento da transição de envio de mensagem pelo roteador.

que está pronta a receber dados através do sinal *podemandar_integrado*. O sinal de *start* provoca também a inicialização de diversas variáveis internas e diversos sinais de saída. A mensagem é recebida através dos sinais *msg_integrado_cod* e *msg_integrado_sdu*. Após o recebimento da mensagem, a transição fica aguardando o sinal *possoenviar* para envio da mensagem através dos sinais *msg_enviar_cod* e *msg_enviar_sdu*.

5.4.4 Síntese de Hardware e Síntese de Software

Cada transição do modelo é descrita em VHDL e C visando sua implementação. Após estas implementações, obtem-se uma tabela contendo os diferentes valores de atraso/desempenho e custo para cada transição do protocolo. Estes valores serão utilizados pelo processo de particionamento para a avaliação da qualidade de diferentes alternativas de particionamento.

A tabela 5.14 apresenta todos os valores obtidos para as implementações em *hardware* das diversas transições do protocolo utilizando a ferramenta Synopsys.

Vale a pena ressaltar que as transições t_1, t_2, t_7 e t_8 são análogas às transições Envia MSG e Recebe MSG do protocolo TCP apresentado na seção 5.2. Logo, os valores apresentados na tabela anterior são referentes à implementação da parte de controle bem como da parte referente ao armazenamento de dados. Por esta razão,

Transição	Atraso Total	Freqüência de operação	Área mm^2
1	51 ns	19,7 MHz	4,510
2	51 ns	19,7 MHz	4,510
3	330 ns	3 MHz	1,668
4	21 ns	47,6 MHz	0,331
5	21 ns	47,6 MHz	0,331
6	5 ns	200 MHz	0,015
7	51 ns	19,7 MHz	4,510
8	51 ns	19,7 MHz	4,510
9	5 ns	200 MHz	0,015
10	5 ns	200 MHz	0,015

Tabela 5.14: Medidas de área e atraso para cada transição do protocolo em ASIC.

as medidas de área para essas transições apresentam valores elevados. As medidas de área somente para a parte de controle valem 0,202 mm^2.

A tabela 5.15 apresenta todos os valores obtidos para as implementações em *hardware* das diversas transições do protocolo utilizando a ferramenta ALTERA.

A tabela 5.16 apresenta valores arbitrados para o desempenho das transições do protocolo ARM em *software*. Os valores das implementações em ASIC foram tomados como base de cálculo. Vale a pena ressaltar que os valores de desempenho de implementações em *software* variam de acordo com o ambiente utilizado.

Esses dados serão encaminhados ao processo de particionamento e ajudarão nos cálculos das aptidões das diversas possibilidades de partições. As partes que serão implementadas em *software* e as partes que serão implementadas em *hardware* serão arbitradas a partir da análise deste conjunto de medidas mostrado.

Transição	Atraso total	Frequência de operação
1	300 ns	3,3 MHz
2	300 ns	3,3 MHz
3	1750 ns	0,57 MHz
4	120 ns	8,3 MHz
5	120 ns	8,3 MHz
6	7 ns	142,8 MHz
7	300 ns	3,3 MHz
8	300 ns	3,3 MHz
9	7 ns	142,8 MHz
10	7 ns	142,8 MHz

Tabela 5.15: Medidas de atraso para cada transição do protocolo em PLD.

5.4.5 Particionamento HW/SW

O processo de particionamento determina efetivamente quais partes do protocolo serão implementadas em *hardware* e quais partes serão implementadas em *software*. A primeira etapa deste processo consiste em se modelar as funcionalidades do protocolo no modelo aceito pelo Tangram-II. A figura 5.19 ilustra o modelo utilizado como base para a construção do modelo inserido no Tangram-II.

A análise de desempenho será focada no tempo de processamento gasto com cada pacote em um roteador ativo como uma função do número de receptores por grupo *multicast*. Para achar a relação entre estas duas medidas, considere o número de pacotes de dados e pacotes de reparo processados por unidade de tempo em um roteador ativo, além do número de pacotes NACKs processados quando o roteador possuir este pacote armazenado. Considere, ainda, que o roteador envia o pacote processado para todos os receptores do grupo.

A transição t_3, responsável pelo recebimento e análise de cada pacote no roteador, é a principal transição a ser avaliada com o intuito de se obter uma expressão

Transição	Atraso total	Frequência de operação
1	1000 ns	1 MHz
2	1000 ns	1 MHz
3	7000 ns	0,14 MHz
4	450 ns	2,2 MHz
5	450 ns	2,2 MHz
6	40 ns	25 MHz
7	1000 ns	1 MHz
8	1000 ns	1 MHz
9	40 ns	25 MHz
10	40 ns	25 MHz

Tabela 5.16: Medidas de atraso para cada transição do protocolo em *software*.

que relacione as medidas de desempenho de tempo de processamento e número de receptores. Se a soma das probabilidades em estado estacionário dos estados que habilitam o evento t_3 é multiplicada pela sua própria taxa (λ_3), pode ser obtido um valor chamado "vazão" da transição t_3. Este valor deve ser igual à vazão da transição t_4, isto é:

$$\pi_i * \lambda_3 = \pi_j * K * \lambda_4 \tag{5.3}$$

onde:

π_i - soma das probabilidades em estado estacionário dos estados nos quais t_3 está habilitada;

π_j - soma das probabilidades em estado estacionário dos estados nos quais t_4 está habilitada;

K - número de receptores por grupo *multicast*.

Dada a equação 5.3, pode-se encontrar uma relação entre o tempo de processa-

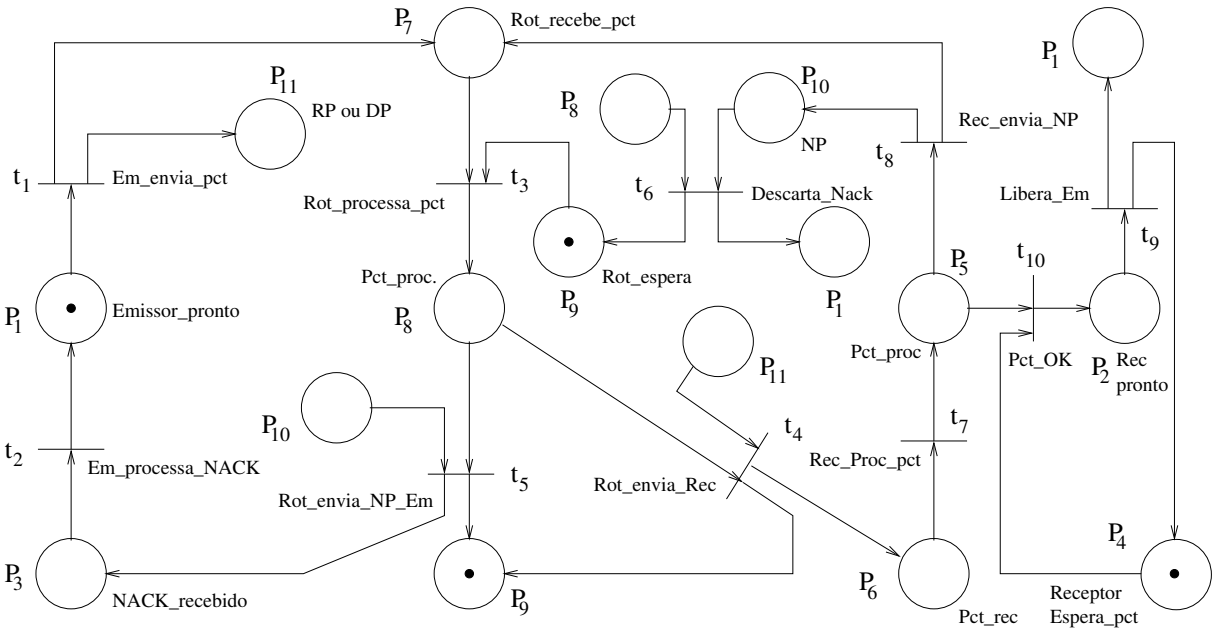

Figura 5.19: Modelo do protocolo ARM utilizado pela ferramenta de análise de desempenho.

mento no roteador e o número de receptores. O tempo de processamento por pacote no roteador, T_{proc}, é igual a $1/\lambda_3$. O aumento no número de receptores é simulado multiplicando a taxa λ_4 pelo número de receptores desejado, ou seja, o valor usado pela ferramenta de análise de dsempenho é igual a $K * \lambda_4$. O mesmo critério é utilizado para as taxas λ_7, λ_8, λ_9 e λ_{10}, referentes ao(s) receptor(es). Desta forma, o tempo de processamento esperado, por pacote, em função do número de receptores pode ser descrito por:

$$T_{proc} = \pi_i/(\pi_j * K * \lambda_4) \tag{5.4}$$

De um modo geral, em um protocolo *multicast*, quando o número de receptores em um grupo aumenta, o tempo de processamento em um roteador também aumenta e, conseqüentemente, a vazão diminui [62]. A equação 5.4 mostra que, para manter o tempo de processamento constante, o tempo de processamento por pacote para cada receptor deve diminuir à medida que aumenta o número de receptores. Isso significa que se deve aumentar a velocidade de processamento do roteador, para tal, deve-

se aumentar o número de tarefas do protocolo selecionadas para implementação em *hardware* à medida que se aumenta o número de receptores, até o limite em que todo o protocolo esteja implementado em *hardware*. A equação 5.4 é utilizada na função objetivo do algoritmo genético de modo a encontrar os parâmetros estabelecidos pelo projetista.

Para se iniciar o processo de otimização são calculados seis valores para o tempo de processamento desejado. Analisando a tabela 5.14 pode-se observar que o valor de atraso para a transição t_3, no pior caso, vale 330 ns. Esse valor é referente ao processamento de um pacote e considerando um receptor. Considera-se, para a análise de desempenho no Tangram-II, um grupo contendo 300 receptores, obtendo assim um valor máximo para o tempo de processamento no roteador ativo de 0,0001 segundo. Esse tempo deve permanecer constante à medida que se aumenta o número de receptores. Tomando este valor como base, obtém-se os outro cinco valores necessários para iniciar o processo de otimização.

Neste exemplo, devido à maior complexidade do protocolo e de seu respectivo modelo, a convergência foi mais demorada e foi necessário utilizar um tamanho de população igual a 1000 indivíduos. O número de gerações utilizado como critério de terminação do algoritmo foi igual a 10. A otimização foi realizada baseada na função objetivo formada pela diferença entre os tempos de processamento obtidos para cada número de receptores utilizado (1, 5, 10, 20, 50 e 100) e os tempos de processamento desejados. Os parâmetros encontrados são apresentados na tabela 5.17.

Utilizando as taxas da tabela 5.17 e as probabilidades em estado estacionário calculadas pelo Tangram-II, o mesmo critério de partição da seção 4.3.4 pode ser aplicado e o produto $\lambda_i * \pi_j$ calculado para cada um dos eventos (transições) do protocolo. Os resultados são mostrados na tabela 5.18.

As tarefas do protocolo relacionadas às transições t_1, t_3 e t_4 devem ser implementadas em *hardware* por serem as transições que possuem os maiores valores para o produto $\lambda_i * \pi_i$. As demais transições, com valores de $\lambda_i * \pi_i$ pequenos, podem ser implementadas em *software*.

Transição	Taxa (1/seg.)
t_1 (λ_1)	42,9
t_2 (λ_2)	2070,0
t_3 (λ_3)	9904,7
t_4 (λ_4)	3075,6
t_5 (λ_5)	2430,4
t_6 (λ_6)	7464,5
t_7 (λ_7)	5144,3
t_8 (λ_8)	9427,5
t_9 (λ_9)	8117,0
t_{10} (λ_{10})	2438,5

Tabela 5.17: Taxas para os eventos do protocolo ARM obtidas pelo AG.

Evento	valor
$\lambda_1 * \pi_1$	41,50
$\lambda_2 * \pi_2$	1,55
$\lambda_3 * \pi_3$	271,33
$\lambda_4 * \pi_4$	14,12
$\lambda_5 * \pi_5$	0,50
$\lambda_6 * \pi_6$	1,55
$\lambda_7 * \pi_7$	0,41
$\lambda_8 * \pi_8$	1,54
$\lambda_9 * \pi_9$	0,80
$\lambda_{10} * \pi_{10}$	0,24

Tabela 5.18: Valores obtidos para o produto $\lambda_i * \pi_j$.

Nota-se, analisando a partição proposta, que as transições a serem designadas para implementação em *hardware* são aquelas que tornam mais rápido o processamento de um pacote no roteador ativo. A escolha da partição se torna até intuitiva ao se analisar o modelo do protocolo ARM. A transição t_1 é responsável em fazer

com que o transmissor envie dados ao grupo *multicast*. Nada mais natural que, ao se desejar aumentar o desempenho deste protocolo, aumente-se o desempenho desta transição. A transição t_3 é responsável pelo recebimento de todos os pacotes no roteador ativo, sejam pacotes de dados, de reparo ou de NACKs. É intuitivo também que esta transição deva possuir um desempenho alto quando se deseja aumentar o desempenho do protocolo como um todo. A transição t_4 é responsável em enviar dados do roteador ativo aos receptores. Esta transição, logicamente, deve possuir alto desempenho de modo a proporcionar uma melhoria no sistema em geral.

5.4.6 Avaliação da Partição Escolhida

Tendo em vista os resultados fornecidos pela ferramenta Synopsys mostrados nas tabelas contidas na seção 5.4.4, a partição proposta é avaliada. Levando-se em conta, por exemplo, um custo de área especificado pelo projetista de 2 mm^2, pode-se observar que as transições t_3 e t_4 podem ser implementadas em HW utilizando-se *standard cells*, pois as medidas de área são menores do que 2 mm^2 e os atrasos associados satisfazem às taxas calculadas pelo AG.

Em contrapartida, a transição t_1 extrapola o requisito de área desejado. Sugere-se, então, que esta transição seja implementada em PLD. Apesar das medidas de atraso de implementção em PLD serem maiores do que em *standard cells*, elas atendem às taxas calculadas pelo AG. Esta sugestão de implementação pode ser melhor avaliada utilizando-se novamente a ferramenta Tangram-II, conforme mostrada na próxima seção.

5.4.7 Análise do Desempenho da Partição Escolhida

A figura 5.20 compara os tempos de processamento em um roteador ativo em função do número de receptores para o caso onde o protocolo é inteiramente implementado em *software* com os tempos de processamento no caso onde o protocolo é implementado a partir da partição HW/SW escolhida.

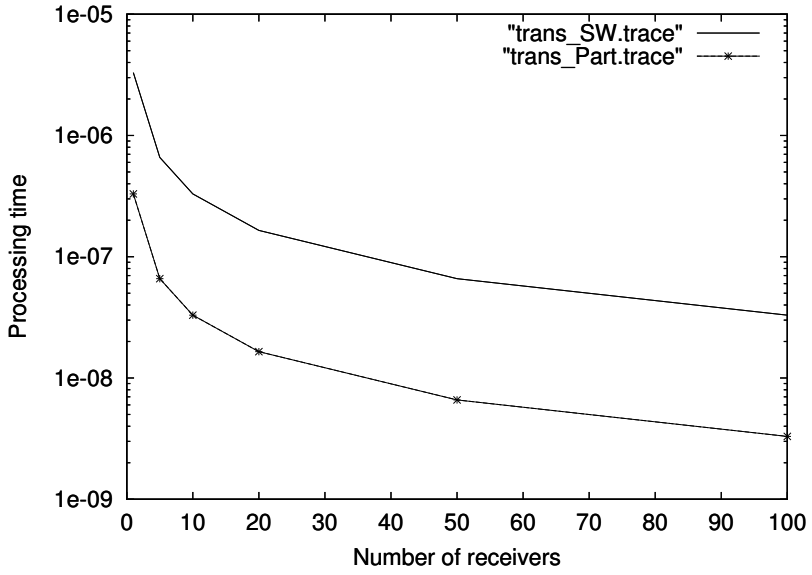

Figura 5.20: Implementação em SW versus implementação com a partição HW/SW.

Pode-se observar que os tempos de processamento da implementação utilizando a partição HW/SW são melhores que os tempos de processamento da implementação em *software*.

A figura 5.21 compara os tempos de processamento para o caso onde o protocolo é totalmente implementado em *hardware* com os tempos de processamento no caso onde o protocolo é implementado a partir da partição HW/SW designada. Neste caso, observa-se que os tempos de processamento são praticamente idênticos. Isto se deve ao fato de que as "vazões" $(\pi_i * \lambda_i)$ das transições implementadas em *hardware* são muito maiores que as "vazões" das outras transições do protocolo.

Outro caso foi também analisado. Designou-se a transição t_3 para ser implementada em *software* e as transições t_1 e t_4 para serem implementadas em *hardware*. Observou-se que os tempos de processamento no roteador ativo são quase iguais aos tempos de processamento obtidos para a implementação em *software*. Isto ocorre porque a "vazão" da transição t_3 é muito maior quando comparada às "vazões" das demais transições. Do ponto de vista da funcionalidade do protocolo, isto indica que a maior parte do processamento se concentra nesta transição.

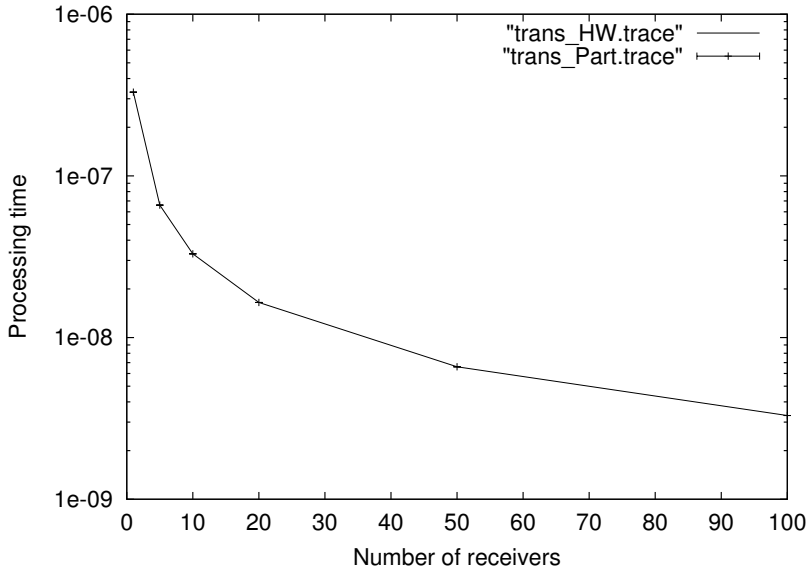

Figura 5.21: Implementação em HW versus implementação com a partição HW/SW.

5.4.8 Refinamento

A etapa de refinamento é responsável em analisar o resultado gerado pelo processo de particionamento e identificar, se possível, pontos que podem ser melhorados. Como pôde ser observado na seção anterior, a modelagem realizada para o protocolo ARM possui a característica de concentrar uma grande parte do processamento dos roteadores em uma única transição (t_3). Este fato pôde ser constatado, inclusive, analisando-se os valores calulados pela ferramenta de análise de desempenho. O valor do produto $\lambda * \pi$ da transição t_3 é muito maior quando comparado aos produtos das outras transições.

Esta modelagem provocou um fato curioso. Os tempos de processamento para a implementação somente da transição t_3 em *software* são praticamente iguais aos tempos de processamento de **TODAS** as transições em *software*. E, por outro lado, os tempos de processamento para a implementação somente da transição t_3 em *hardware* são praticamente iguais aos tempos de processamento de **TODAS** as transições em *hardware*.

Uma vez que a transição t_3 concentra a maior parte do processamento do roteador, um refinamento possível seria desmembrar esta transição em várias outras de

modo a descentralizar o processamento. A figura 5.22 ilustra a divisão da transição t_3 em quatro outras: Rot_Recebe_Msg, Rot_Proc1, Rot_Proc2 e Rot_Proc3. Agora, o pacote é recebido pela transição Rot_Recebe_Msg e dependendo do tipo de pacote uma das três transições é disparada. Após o disparo dessas transições, ocorre o envio de pacotes de reparo, pacotes de NACK para a fonte ou simplesmente envio de pacotes de dados para os receptores, dependendo da transição disparada.

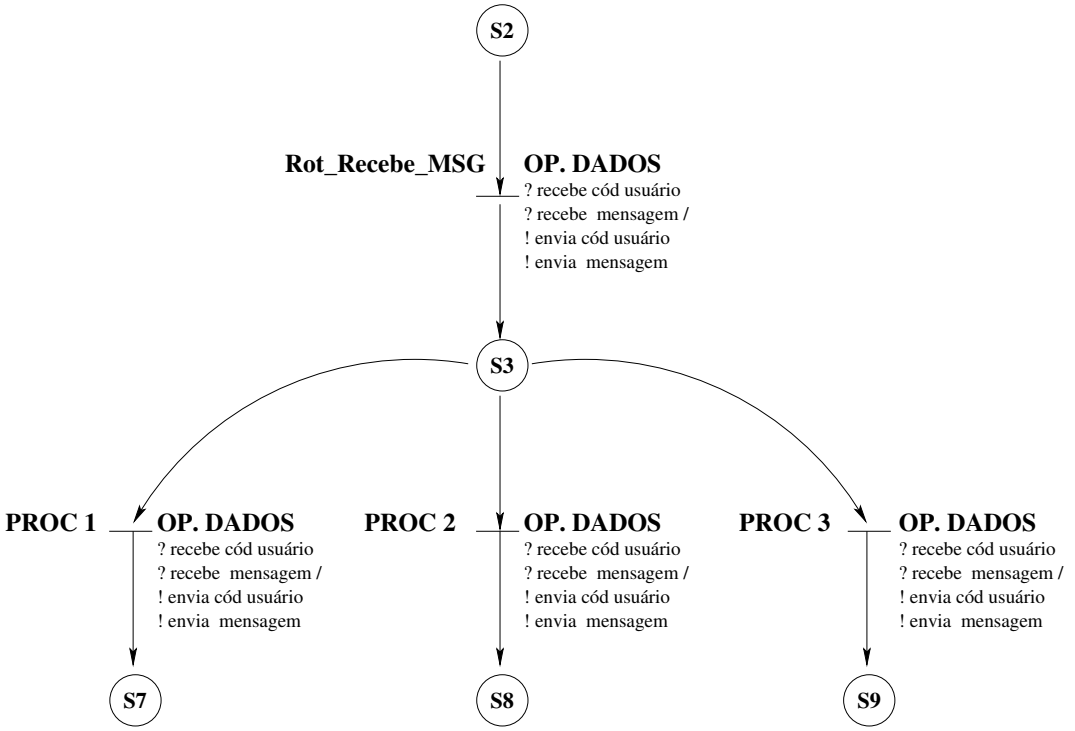

Figura 5.22: Refinamento da transição t_3 do protocolo ARM.

A tabela 5.19 apresenta os valores obtidos para as implementações em *hardware* das novas transições do protocolo, após a etapa de refinamento, utilizando a ferramenta Synopsys.

5.4.9 Integração do Sistema

A etapa de integração do sistema, que no atual estágio do projeto consiste basicamente na síntese da comunicação, é responsável em implementar o esquema de troca de informações entre os diversos módulos de *hardware* e *software*. A figura 5.23 ilustra como o protocolo se comporta ao final do ciclo de projeto.

Transição	Atraso Total	Freqüência de operação	Área mm^2
Rot Recebe MSG	272 ns	3,7 MHz	1,024
Rot Proc 1	32 ns	31,25 MHz	0,467
Rot Proc 2	32 ns	31,25 MHz	1,071
Rot Proc 3	32 ns	31,25 MHz	0,899

Tabela 5.19: Medidas de área e atraso para as transições após o refinamento.

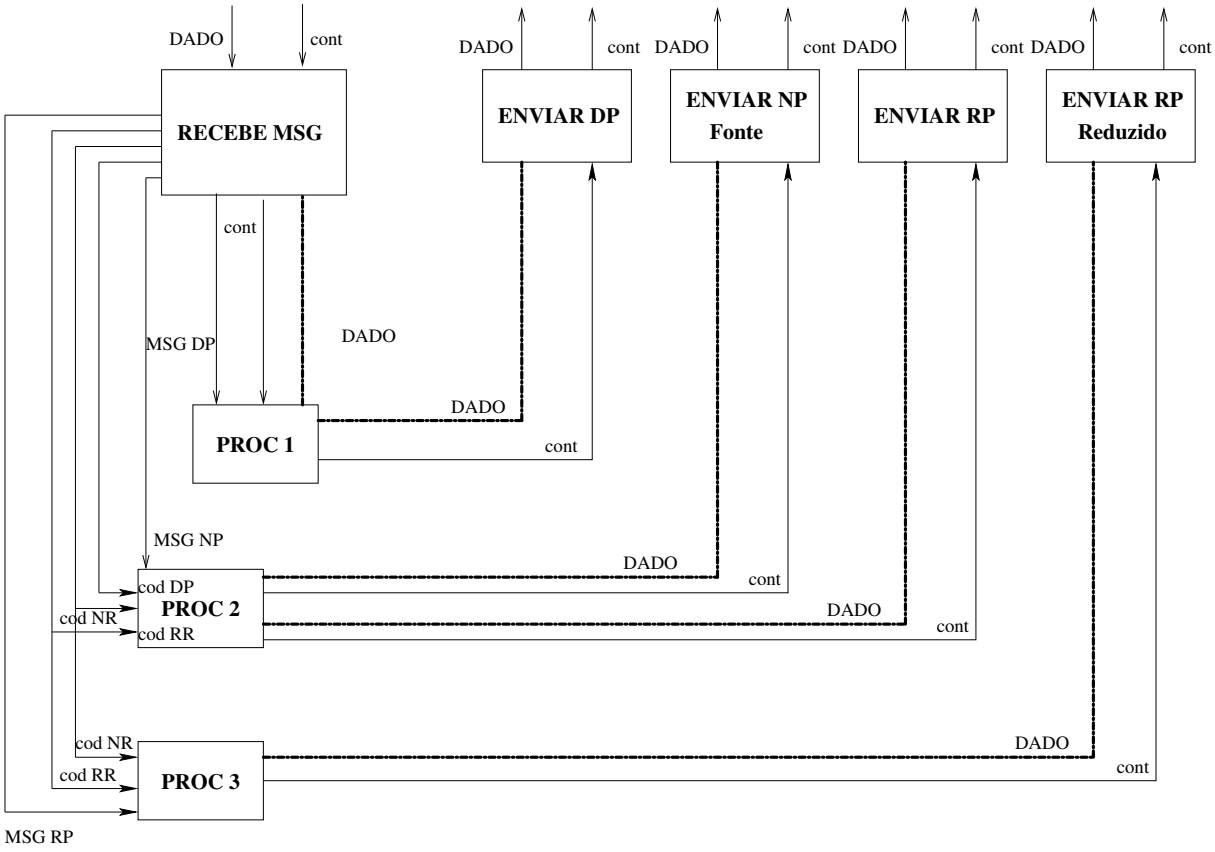

Figura 5.23: Disposição final dos módulos do protocolo nos roteadores ativos.

5.4.10 Resultados de Simulação do Protocolo ARM

Esta seção apresenta algumas figuras referentes a cenários específicos de simulação do protocolo ARM. A figura 5.24 ilustra o comportamento do módulo *Integrado* através da apresentação dos diversos sinais que o compõem. Pode-se observar clara-

mente os sinais de *start*, *reset* e *clock*, sinais estes bem particulares das implementações em *hardware*. Este módulo recebe um pacote oriundo do transmissor ou de um dos receptores do grupo *multicast* através do sinal *msg_roteador_sdu* especificado pelo sinal SG_S da figura abaixo. Primeiramente, o módulo identifica que tipo de mensagem este pacote carrega através da análise dos campos iniciais presentes em seu do cabeçalho. De acordo com o tipo de mensagem (pacote de dados, pacote de NACK, pacote de ACK ou pacote de reparo), são realizadas uma série de operações de acordo com o algoritmo do protocolo ARM. Ao final dessas operações, o pacote é repassado aos respectivos módulos Enviar.

A figura 5.25 ilustra o comportamento do módulo que implementa a transição PROC1 através da apresentação dos diversos sinais que a compõem. Pode-se observar claramente os sinais de *start*, *reset* e *clock*. Este módulo recebe um pacote oriundo do módulo Recebe MSG através do sinal *msg_recebe_sdu* especificado pelo sinal SG_P da figura abaixo. Se o roteador possuir memória *cache*, este pacote é armazenado utilizando-se os sinais *conta_end* (contador de endereços), *esc_leit* (sinal de escrita e/ou leitura) e *msg_enviar_sdu* (mensagem). Esses sinais são representados na figura pelos sinais SG_AE, SG_AF e SG_AD, respectivamente. Se o roteador não possuir memória *cache*, o pacote é simplesmente repassado para o módulo Enviar.

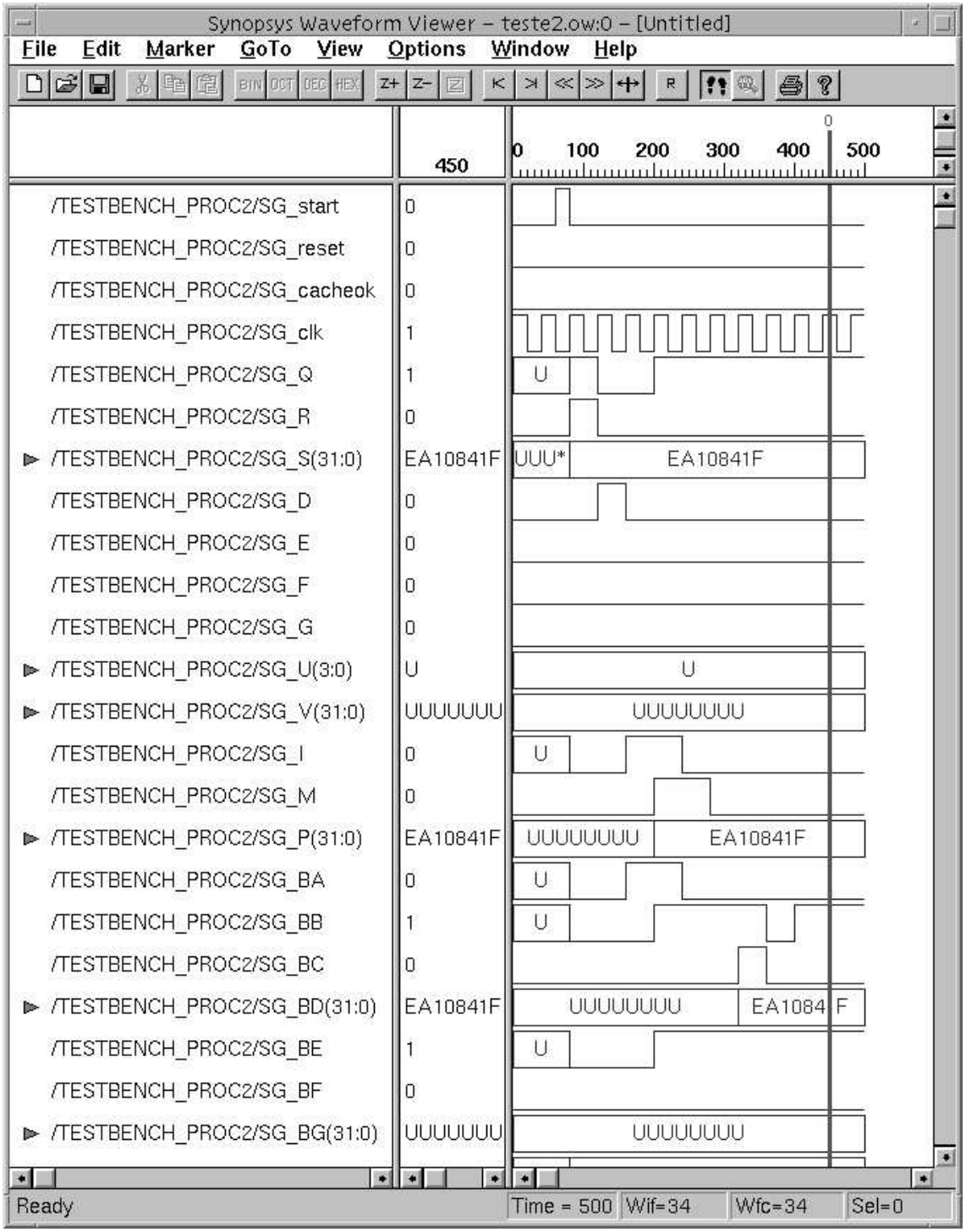

Figura 5.24: Simulação do módulo Integrado do protocolo ARM.

5.5 Avaliação dos Diferentes Estudos de Caso

A utilização de uma metodologia de Codesign no projeto de protocolos é a idéia chave deste trabalho. A implementação de sistemas em HW não é uma tarefa

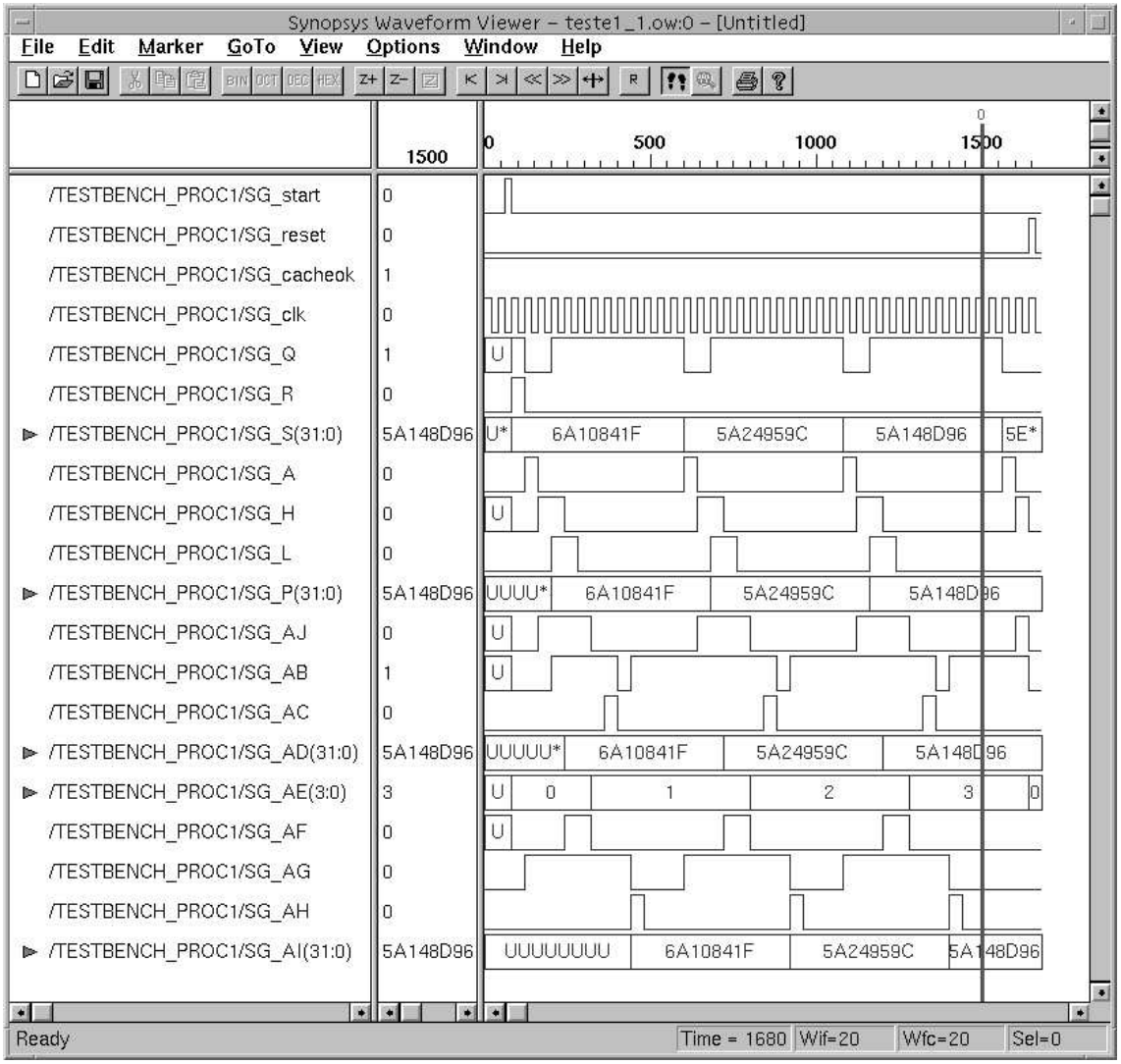

Figura 5.25: Simulação do módulo PROC1 do protocolo ARM.

trivial. Devido a este fato, este trabalho se concentra no processo de obtenção dos parâmetros de área e atraso de diferentes implementações das diversas transições de um determinado protocolo em HW. Tais medidas são apresentadas e discutidas em todos os estudos de caso realizados. Com relação às implementações em SW, são adotados valores estimados para as diferentes transições do protocolo [33].

Baseado em experiências anteriores de síntese de circuitos em HW, buscou-se identificar determinadas características que estes circuitos apresentassem e inserí-las em um nível de abstração mais elevado. A adaptação do modelo de MEF apresentada neste trabalho enfoca este objetivo. O intuito é facilitar a tarefa do projetista

de protocolos não familiarizado com implementações de circuitos em HW, pois muitas das vezes, este projetista se preocupa somente com implementações em SW, e diversas características passíveis de serem implementadas em SW dificilmente são sintetizadas em HW.

Após a etapa de particionamento, ocorre a etapa de prototipagem, que consiste em se efetuar a síntese de HW e SW e a síntese da comunicação entre os dois domínios. A síntese da comunicação é responsável pela modelagem e implementação do esquema de comunicação utilizado. Esta etapa implementa a troca de informações entre os diversos módulos do projeto, sejam eles descritos em HW ou SW. O modelo utilizado neste trabalho consiste na utilização de sinais de controle para a comunicação entre os módulos. Estes sinais de controle já são levados em consideração quando da modelagem do protocolo em MEF no início do ciclo de projeto, facilitando assim, a etapa de síntese da comunicação. Esta etapa deverá se preocupar, basicamente, em implementar os esquemas de comunicação dos módulos em SW, como por exemplo, esquemas com *sockets* ou RPC (*Remote Procedure Call*). Quanto ao HW, alguns detalhes de conflitos de acesso à barramentos devem ser resolvidos.

Uma consideração importante deve ser feita com relação ao custo que a comunicação entre os módulos de HW e SW impõe ao desempenho geral do sistema. Quando a troca de informações entre dois módulos distintos é muito intensa, talvez seja mais interessante agrupar estes módulos, diminuindo assim, o tempo gasto com esta tarefa. Este agrupamento pode melhorar sensivelmente o desempenho do sistema. O próximo passo deste estudo é incluir na função objetivo, que auxilia na escolha da melhor partição, o custo da comunicação entre os diferentes módulos do sistema.

5.6 Comentários

Este capítulo apresentou os resultados práticos da utilização da metodologia proposta neste trabalho a partir da utilização de três protocolos com características diferentes. Estes resultados ilustram todas as etapas do ciclo de projeto. O deta-

lhamento das funcionalidades dos protocolos foram apresentados. As modelagens destas funcionalidades através da utilização do modelo de MEF adaptado à síntese foram discutidas. Também foram abordados os detalhamentos das transições dos protocolos para descrições em VHDL, as sínteses de *hardware* e as medidas de área e atraso obtidas. Os processos de particionamento, refinamento e integração dos sistemas também foram abordados.

No caso de implementações de protocolos dominados por fluxo de controle, exemplo do protocolo WATM, ressalta-se que devido a determinadas transições serem de simples implementação, a etapa de refinamento ficaria encarregada de, possivelmente, agrupar transições, gerando assim um novo modelo. Este agrupamento de transições objetiva diminuir a comunicação entre domínios ou até mesmo entre módulos de HW que implementem diferentes transições.

No caso de implementações de protocolos dominados por fluxo de dados, exemplo do protocolo ARM, ressalta-se que devido à complexidade de determinadas transições, a etapa de refinamento ficaria encarregada de, possivelmente, desmembrar transições que concentrem uma grande parte do processamento do protocolo. Este desmembramento de transições visa dividir o processamento em sub-tarefas (ou sub-transições) e a partir do processo de particionamento, identificar quais partes serão implementadas em HW e quais partes serão implementadas em SW.

Capítulo 6

Considerações Finais

A Utilização conjunta de *hardware* e *software* na concepção de protocolos de comunicação assume uma maior importância devido à crescente demanda por redes de alto desempenho e por aplicações que exigem altas taxas de processamento. Nos últimos anos, as técnicas de projeto de circuitos integrados tiveram uma grande evolução. O progresso da tecnologia de fabricação de circuitos tem feito com que a concepção de circuitos migre para a concepção de sistemas mais complexos.

Atualmente, devido ao surgimento de novas e complexas aplicações, um dos sistemas candidatos à implementação em *hardware* são os protocolos de comunicação. O princípio básico da técnica de *Codesign* é a realização de um projeto cooperativo baseado em dois ambientes de projetos específicos, *hardware* e *software*. A verificação e a simulação de todo o sistema são realizadas em qualquer etapa do projeto e este tipo de concepção conduz a uma diminuição do tempo de projeto, e conseqüentemente, a um aumento de produtividade.

Portanto, a primeira contribuição deste trabalho consiste em desenvolver uma metodologia de projeto de protocolos de comunicação baseada nos conceitos da técnica *Hardware/Software Codesign*. A utilização desta técnica proporciona, ainda, alguns outros benefícios, como a possibilidade de diminuição de custos de projeto e a possibilidade de interação entre os domínios permitindo etapas de refinamento e reavaliação de partições **durante** o ciclo de projeto.

O projetista de protocolos está mais familiarizado em iniciar o projeto a partir de um nível de abstração mais elevado do que o comumente empregado em implementações de sistemas em *hardware*. Aumentando o nível de abstração, o projetista pode manipular aplicações mais complexas, com um domínio total das tecnologias. Deste modo, ele pode escolher a tecnologia que melhor implementa cada parte do sistema.

A segunda contribuição deste trabalho se baseia em "capturar" as características principais de descrições de sistemas em *hardware* e englobá-las num modelo de descrição de protocolos em alto nível. Este modelo permite, além de descrever as funcionalidades do protocolo, se preocupar de antemão com as futuras implementações em *hardware*. O modelo adotado para a descrição de protocolos na metodologia proposta foi o modelo de Máquina de Estados Finitos (MEF) adaptado à síntese. Esta adaptação se deve à inserção de sinais de controle e sinais de sincronização inerentes às descrições de circuitos integrados, o que facilitará as etapas de síntese de *hardware* e de síntese da comunicação.

Este modelo pode ser testado, excluindo assim possíveis preocupações em se carregar erros iniciais de especificação para as etapas posteriores do projeto. Além disso, este modelo permite que o sistema seja visto independentemente de cada domínio de implementação. Isto é um fator importante nas etapas de particionamento e refinamento.

Outra grande vantagem é que este tipo de modelagem pode ser tratada como um tipo de representação interna da descrição inicial do sistema. Assim, insere-se uma possibilidade de se "acoplar" diversos módulos de representação inicial baseados em diferentes linguagens de programação. Estes módulos somente terão a preocupação de gerar especificações no modelo de MEF adaptado à síntese a partir da utilização da linguagem de preferência do projetista.

Atualmente, o projetista ainda dispõe de poucos recursos para auxiliá-lo em duas etapas cruciais num projeto de *Codesign*, as etapas de particionamento e refinamento da especificação. Normalmente, a experiência do projetista aliada a técnicas informais constituem a base para se definir a melhor partição, o que limita a exploração

do espaço de soluções. A tarefa do projetista exige métodos e meios que permitam valorar e analisar os diferentes tipos de arquiteturas do sistema afim de atender melhor às restrições de desempenho e custo e garantir uma reação rápida a uma mudança da tecnologia de implementação. Neste caso, é útil fornecer ao projetista subsídios que auxiliem-no na escolha da melhor partição.

A terceira contribuição deste trabalho consiste na obtenção de um conjunto de valores de atraso e custo de implementação das possíveis transições do protocolo. Esses valores fornecerão subsídios ao projetista na tarefa de escolha da melhor partição *hardware/software* do sistema. Este trabalhou optou por utilizar a estratégia de particionamento proposta por Miranda em [43]. Desta forma, os valores extraídos de diferentes perspectivas de implementação visam automatizar o processo de particionamento.

Implementações utilizando a ferramenta Synopsys para síntese de circuitos ASIC com biblioteca de células padrão e implementações com a ferramenta ALTERA para prototipagem rápida em PLD foram consideradas. Os resultados mostram uma nítida vantagem em termos de velocidade de operação de um projeto utilizando células padrão em relação à implementação utilizando PLDs. Por outro lado deve ser notado que a implementação em PLDs é imediata e o sistema pode ser reprogramado sem a necessidade de uma nova rodada de fabricação como no caso das *standard cells*. Outro aspecto relevante é o baixo custo de implementação utilizando os dispositivos de lógica programável. A desvantagem de um possível aumento no tempo de projeto para se calcular tais parâmetros é compensada pela melhor qualidade dessas medidas em relação aos resultados obtidos por estimadores.

O ciclo de projeto se encerra com a integração do sistema após a etapa de particionamento. Como resultado da metodologia apresentada, esta integração é realizada basicamente efetuando-se a síntese da comunicação. Outra contribuição que pode ser ressaltada consiste na simplificação da etapa de síntese da comunicação. Esta etapa se torna bastante simplificada, pois o método utilizado para descrever o protocolo em alto nível já se encarregou de especificar várias características inerentes à comunicação entre entidades de *hardware*.

Um ponto interessante nesta metodologia é que da forma como o protocolo foi modelado e implementado existe a possibilidade de se criar uma "biblioteca de transições padrão".Geralmente, os protocolos de comunicação possuem muitas primitivas em comum. As diferenças entre eles se baseiam em detalhes de implementação voltados para aplicações específicas. Desta forma, existe a possibilidade de se armazenar as possíveis implementações em ASIC, PLD ou *software* de uma determinada transição. Quando outro protocolo for projetado, consulta-se esta base de dados e já se tem uma possível implementação bem como os valores de atraso e custo relativos à esta transição. Esta abordagem acarretaria numa diminuição ainda maior no tempo de projeto e nos custos de implementação.

Os experimentos descritos no capítulo 5 comprovam a potencialidade da metodologia proposta. Foram implementados três diferentes protocolos de comunicação. Primeiramente, utilizou-se como estudo de caso, o protocolo TCP. Procura-se deixar claro que este exemplo não é um bom candidato para se aplicar esta metodologia. Isto porque o desempenho do protocolo TCP (e seu controle de congestionamento) está diretamente ligado ao número de pacotes perdidos **na rede**. Portanto, mesmo que o protocolo fosse implementado **totalmente** em *hardware* não seria possível garantir um alto desempenho, devido às limitações impostas pelo tráfego **da rede**. A primeira preocupação na utilização deste exemplo consistiu em se avaliar a viabilidade da metodologia. Neste sentido, o exemplo alcançou seu objetivo.

O segundo estudo de caso apresentado foi a implementação do protocolo WATM e o terceiro foi a implementação do protocolo ARM para grupos *multicast*. Estes dois exemplos são bem interessantes do ponto de vista de se testar a viabilidade da metodologia, isso porque o primeiro caso se enquadra em um protocolo dominado por fluxo de controle e o segundo em um protocolo dominado por fluxo de dados.

No caso de se implementar protocolos dominados por fluxo de controle, caso do protocolo WATM, ressalta-se que devido a determinadas transições serem de simples implementação, a etapa de refinamento ficaria encarregada de, possivelmente, agrupar transições, gerando assim um novo modelo. Este agrupamento de transições objetiva diminuir a comunicação entre domínios ou até mesmo entre módulos

de HW que implementem diferentes transições.

No caso de se implementar protocolos dominados por fluxo de dados, caso do protocolo ARM, ressalta-se que devido a complexidade de determinadas transições, a etapa de refinamento ficaria encarregada de, possivelmente, desmembrar transições que concentrem uma grande parte do processamento do protocolo. Este desmembramento de transições visa dividir o processamento e a partir do processo de particionamento, identificar quais partes serão implementadas em HW e quais partes serão implementadas em SW.

Com o intuito de aperfeiçoar a metodologia de projeto de protocolos utilizando *Hardware/Software Codesign*, trabalhos futuros contendo novos estudos de caso devem ser implementados. Um parâmetro comumente utilizado para se avaliar diferentes partições é o custo da comunicação. No atual estágio, esse custo não é mensurado de forma automatizada e o máximo que é considerado no processo é a minimização desta comunicação na etapa de refinamento, mas de forma manual e valendo-se da experiência do projetista. Um passo importante seria incluir este parâmetro na modelagem e no processo de particionamento.

Referências Bibliográficas

[1] D. Gajski e F. Vahid. Specification and Design of Embedded Software/Hardware Systems. *IEEE Design and Tests of Computer 12*, 1 (Janeiro 1995), 53–67.

[2] D. W. Franke e M. K. Purvis. Hardware/Software Codesign: A Perspective. In *13th International Conference on Software Engineering* (1991).

[3] D. W. Franke e M. K. Purvis. An Overview of Hardware/Software Codesign. *International Symposium on Circuits and Systems* (1992).

[4] W. H. Wolf. Hardware/Software Codesign of Embedded Systems. In *IEEE Computer* (Julho 1994), vol. 82 (7), pp. 967–989.

[5] S. Kumar, J. H. Aylor, B. W. Johnson e W. A. Wulf. *The Codesign of Embedded Systems: A Unified Hardware/Software Representation.* Kluwer Academic Publishers, 1996.

[6] T. B. Ismail, M. Abid e A. Jerraya. COSMOS: A Codesign Approach for Communication Systems. *3th International Workshop on Hardware/Software Codesign* (1994), 17–24.

[7] D. Gajski, F. Vahid, S. Narayan e J. Gong. System-Level Exploration with SpecSyn. *Design Automation Conference* (1998), 812–817.

[8] F. Vahid, T. D. Lee e Y-C. Hsu. A Comparison of Functional and Structural Partitioning. *International Symposium on System Synthesis* (Novembro 1996), 121–126.

[9] A. Kalavade e E. A. Lee. Hardware/Software Codesign using Ptolomey - A Case Study. *Proceedings of IEEE International Workshop on Hardware/Software Codesign* (1992).

[10] J. Madsen, J. Grode, P. Knudsen, M. E. Petersen e A. Haxthausen. LYCOS: the Lyngby Co-Synthesis System. *Design Automation for Embedded Systems 2*, 2 (Fevereiro 1997), 1–43.

[11] P. H. Chou, R. B. Ortega e G. Borriello. The Chinook Hardware/Software Co-Synthesis System. *8th International Symposium on System Synthesis* (1995).

[12] G. de Michelli, D. Ku, F. Mailhot e T. Truong. The Olympus Synthesis System. *IEEE Design and Test* (1990).

[13] K. Tammemae, M. O'Nils, A. Jantsch e A. Hemani. AKKA: A Codesign Environment. In *Poster session of 13th IEEE NORCHIP Conference* (Novembro 1995), p. 249.

[14] A. Jantsch. BEKKA - A System Specification and Design Framework. *Internal Report TRITA-ESD-1997-01* (Royal Institute of Technology, Sweden, 1997).

[15] P. Maciel, E. Barros e W. Rosenstiel. Estimating Functional Unit Number in the PISH Codesign System by Using Petri Nets. In *XII Symposium on Integrated Circuits and Systems Design* (Setembro 1999), pp. 32–35.

[16] IEEE Std 1076-1987. *IEEE Standard VHDL Language Reference Manual*, março de 1988.

[17] P. J. Ashenden. *The VHDL Cookbook*. Computer Science Dept. of University of Adelaide, 1990.

[18] Synopsys, Inc. *Synopsys Online Documentation, v1998.02*, 1998.

[19] Altera, Corp. *Data Book and Max + PlusII Getting Started*, 1997.

[20] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, 1996.

[21] H. Horner. A C++ Class Library for Genetic Programming: The Vienna University of Economics - Genetic Programming Kernel. *Internet Draft* (maio de 1996). `http://www.wu-wien.ac.at/usr/h88/h8850092`.

[22] R. Carmo, L. R. Carvalho, E. Sousa e Silva, M. C. Diniz e R. R. Muntz. Performance/Availability Modeling with the Tangram-II Modeling Environment. *Performance Evaluation 33*, 1 (Junho 1998), 45–65.

[23] A. P. C. Silva. Tangram-ii user's manual. Relatório técnico, Universidade Federal do Rio de Janeiro, outubro de 2000. `http://www.land.ufrj.br`.

[24] F. Hessel e G. Marchioro. Concepção Conjunta de Hardware/Software (Co-Design). In *XVI Jornada de Atualização em Informática - XVII Congresso da Sociedade Brasileira de Computação* (Agosto 1997), pp. 449–494.

[25] L. F. Soares, G. Lemos e S. Colcher. *Redes de Computadores: das LANs, MANs e WANs às redes ATM*, 2 ed. Ed. Campus, Rio de Janeiro, 1995.

[26] Andrew S. Tanenbaum. *Computer Networks.* Prentice Hall, 1996.

[27] W. Doeringer et al. A Survey of Light-Weight Transport Protocols for High Speed Networks. *IEEE Transactions on Communications 388*, 11 (novembro de 1990), 2025–2038.

[28] ITU-T Recommendation I.363. *B-ISDN ATM Adaptation Layer (AAL) Specification*, 1993.

[29] S. A. Virtanen. On Communications Protocols and their Characteristics Relevant to Designing Protocol Processing Hardware. Relatório técnico, University of Turku, setembro de 1999.

[30] R. Ernst, J. Henkel e T. Benner. Hardware-Software Cosynthesis for Microcontrollers. *IEEE Design and Tests of Computer* (dezembro de 1993).

[31] R. K. Gupta e G. Michelli. Hardware-Software Cosynthesis for Digital Systems. *IEEE Design and Tests of Computer* (setembro de 1993).

[32] S. Fischer, J. Wytrebowicz e S. Budkowski. Hardware/Software Codesign of Communication Protocols. In *Proceedinds of IEEE 22nd Euromicro Conference* (1996).

[33] J. Hidalgo e J. Lanchares. Functional Partitioning for Hardware/Software Codesign using Genetic Algorithm. In *Proceedings of the 23rd Euromicro Conference* (1997).

[34] A. Jerraya e K. O'Brien. SOLAR: An Intermediate Format for System-Level Modeling and Synthesis. *in Computer Aided Software-Hardware Engineering, J. Rozenblit, K. Buchenrieder (eds), IEEE Press* (1994).

[35] P. Maciel, E. Barros e W. Rosenstiel. A Petri Net Based Approach for Performing the Initial Allocation in Hardware/Software Codesign. In *IEEE International Conference on Systems, Man, and Sybernetics* (Outubro 1998).

[36] P. Maciel e E. Barros. Capturing Time Constraints by Using Petri Nets in the Context of Hardware/Software Codesign. In *7th IEEE International Workshop on Rapid System Prototyping* (Junho 1996).

[37] E. Barros. *Hardware/Software Partitioning Using UNITY*. Tese de Doutorado, Universität Tübingen, 1993.

[38] S. Budkowski. Estelle Development Toolset. *Computer Networks and ISDN Systems, Special Issue on FDT Concepts and Tools 25(1)* (1992).

[39] E. Lallet, S. Fischer e J-F. Verdier. A New Approach for Distributing Estelle Specifications. In *G. von Bochmann, R. Dssouli, O. Rafiq (editors), 8th International Conference on Formal Description Techniques (FORTE'95)* (1995), pp. 439–448.

[40] J. Wytrebowicz e S. Budkowski. Communication Protocols Implemented in Hardware: VHDL Generation from Estelle. In *J-M.Berge, O. Levia, J. Rouillard (editors), Current Issues in Electronic Medelling: Languages for System Abstraction* (1995), Kluwer Academic Publishers, pp. 77–98.

[41] A. S. Wenban, J. W. O'Leary e G. M. Brown. Codesign of Communication Protocols. *International Workshop on Hardware/Software Codesign* (1992).

[42] D. Gajski, N. Dutt, A. Wu e S. Lin. *High-Level Synthesis - Introduction to Chip and System Design.* Kluwer Academic Publishers, 1992.

[43] M. N. de Miranda. *Uma Metodologia de Hardware/Software CoDesign de Protocolos de Comunicação Baseada na Otimização de Desempenho por Algoritmos Genéticos.* Tese de Doutorado, Universidade Federal do Rio de Janeiro, PEE/COPPE/UFRJ, Rio de Janeiro, Brasil, 2000.

[44] R. N. B. Lima, E. Carli, L. Pirmez, A. Pedroza e A. C. Mesquita. Communication Protocols in Integrated Circuits: Experiments and Results. *IEEE Global Telecommunications Conference - Globecom'99* (Dezembro 1999).

[45] R. N. B. Lima, E. Carli, L. Pirmez, A. Pedroza e A. C. Mesquita. Protocolos de Comunicação em Circuitos Integrados: Experiências e Resultados. *XVII Simpósio Brasileiro de Telecomunicações, SBT99* (Setembro 1999).

[46] R. N. B. Lima, E. Carli, L. Pirmez, A. Pedroza e A. C. Mesquita. Logic and High Level Synthesis for Communication Protocols. *XII Symposium on Integrated Circuits and Systems Design, SBCCI'99* (Setembro 1999).

[47] R. N. B. Lima. Projeto de Protocolos visando a Implementação em Hardware: Aplicação ao Caso do Protocolo AAL5 para Redes ATM. Tese de Mestrado, Universidade Federal do Rio de Janeiro, PEE/COPPE/UFRJ, Rio de Janeiro, Brasil, 1998.

[48] R. N. B. Lima, M. N. de Miranda, A. C. P. Pedroza e A. C. M. Filho. Uma Metodologia para Projeto de Protocolos Utilizando HW/SW Codesign Baseado na Otimização de Desempenho por Algoritmos Genéticos. Relatório técnico, Universidade Federal do Rio de Janeiro, outubro de 2003. `http://www.gta.ufrj.br`.

[49] M. K. Molloy. Performance Analisys Using Stochastic Petri Nets. *IEEE Transactions on Computers 31*, 9 (setembro de 1982), 913–917.

[50] S. M. Ross. *Stochastic Processes*. John Wiley & Sons, 1983.

[51] W. Chia-Whei Cheng. The Tangram Graphical Interface Facility (TGIF) Manual. *Internet Draft* (2000). `http://bourbon.cs.ucla.edu:8801/tgif/`.

[52] R. N. B. Lima, M. N. de Miranda, A. C. P. Pedroza e A. C. M. Filho. HW/SW Codesign de Protocolos Baseado na Otimização de Desempenho por Algoritmos Genéticos. In *XIX Simpósio Brasileiro de Redes de Computadores - SBRC'2001* (Maio 2001).

[53] R. N. B. Lima, M. N. de Miranda, A. C. P. Pedroza e A. C. M. Filho. HW/SW Codesign of Protocols Based on Performance Optimization Using Genetic Algorithms. In *17th International Teletraffic Congress - ITC'2001* (Setembro 2001).

[54] M. Mathis, J. Semke e J. Mahdavi. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. *Computer Communication Review 27*, 3 (Julho 1997), 1–16.

[55] Altera, Corp. *Max + PlusII Getting Started*, 1997.

[56] R. N. B. Lima, M. N. Miranda, J. V. F. Filho, A. Pedroza e A. C. Mesquita. HW/SW Codesign of Handoff Protocol for Wireless ATM Networks based on Performance Optimization using Genetic Algorithms. *15th Symposium on Integrated Circuits and System Design, SBCCI 2002* (Setembro 2002).

[57] R. N. B. Lima, M. N. Miranda, A. Pedroza e A. C. Mesquita. Projeto de Protocolos Utilizando HW/SW Codesign Baseado na Otimização de Desempenho por Algoritmos Genéticos. *XX Simpósio Brasileiro de Telecomunicações, SBT 2003* (Outubro 2003).

[58] A. Acharya, J. Li e A. Bakre. Design and Prototyping of Location Management and Handoff Protocols for Wireless ATM Networks. In *Proceedings of ICUPC'97* (1997).

[59] R. Yuan, K. Biswas, L.J. French, J. Li e D. Raychaudhuri. A Signaling and Control Architecture for Mobility Support in Wireless ATM Networks. *MONET 1*, 3 (março de 1996), 287–298.

[60] R. N. B. Lima, M. N. Miranda, A. Pedroza e A. C. Mesquita. Design of a Reliable Multicast Protocol using HW/SW Codesign based on Performance Optimization with Genetic Algorithms. *IEEE International Telecommunications Symposium, ITS2002* (Setembro 2002).

[61] L. H. Lehman, S. J. Garland e D. L. Tennenhouse. Active Reliable Multicast. In *Proceedings of INFOCOM'98* (1998).

[62] D. Towsley, J. F. Kurose e S. Pingali. A Comparison of Sender-Initiated and Receiver-Initiated Reliable Multicast Protocols. *IEEE Journal on Selected Areas in Communications 15*, 3 (1997), 398–406.