JOHAN SEBASTIAN ESLAVA GARZON

ESTIMATIVAS DE DESEMPENHO DA ESTRUTURA DE COMUNICAÇÃO DE SoC A PARTIR DE MODELOS DE TRANSAÇÕES

São Paulo
2009

JOHAN SEBASTIAN ESLAVA GARZON

ESTIMATIVAS DE DESEMPENHO DA ESTRUTURA DE COMUNICAÇÃO DE SoC A PARTIR DE MODELOS DE TRANSAÇÕES

Tese apresentada á Escola Politécnica da Universidade de São Paulo para obtenção do título de Doutor em Engenharia Elétrica.

Área de concentração: Microeletrônica

Orientador: Prof. Dr. Marius Strum

São Paulo

2009

Este exemplar foi revisado e alterado em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, de maio de 2009.

Assinatura do autor _____

Assinatura do orientador _____

FICHA CATALOGRÁFICA

Eslava Garzon, Johan Sebastian
Estimativas de desempenho da estrutura de comunicação SoC a partir de modelos de transações / J.S. Eslava Garzon. -- ed.rev. -- São Paulo, 2009.
111 p.

Tese (Doutorado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Sistemas Eletrônicos.

1. Simulação 2. Sistemas integrados em larga escala (Proje - to) 3. CAD 4. Circuitos integrados I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Sistemas Eletrônicos II. t.

Á DEUS fonte de todo

À minha família por quem eu fiz este percurso

AGRADECIMENTOS

Ao meu orientador Prof. Marius Strum, uma pessoa que sempre foi um apoio, um exemplo e um grande amigo.

Aos meus grandes irmãos que sempre me acompanharam neste percurso aqui no Brasil: Leonardo e Eduard. Duas pessoas maravilhosas que foram sempre leais, prestativas e uma excelente companhia. Eu sempre vou contar com vocês e vocês comigo.

Aos meus colegas de apartamento atuais e passados, pelo seu calor humano, sua paciência, longas conversas e risos inesquecíveis: Paola, Eduard, Leonardo, Nuri e Beatriz.

A todos os colegas que passaram no grupo e no laboratório, pelas grandes conversas, trocas de idéias, churrascos e o apoio, Prof. Wang Jiang Chau, Prof. Denise, Mauricio, Jair, os dois João, Ivan, Leonardo, Gino, John, Raul, Gustavo, Artur, Mario, Jonas e muitos mais...

Todas as pessoas que conheci e contribuíram para ter uns maravilhosos anos neste prezado Brasil. Grandes amigos Colombianos, Mexicanos e Brasileiros sempre vão estar nas minhas ótimas lembranças.

As pessoas que confiaram em mim na Colômbia, que sempre tiveram palavras de apoio e alegria para que eu pudesse continuar no meu caminho: Elizabeth, Margarita, Rene, Albin, Lina, Diana e a minha família

À CNPQ pela bolsa de doutorado e pelo apoio econômico da taxa de bancada para a realização desta pesquisa

RESUMO

A complexidade crescente (tanto da funcionalidade como da arquitetura) dos sistemas eletrônicos digitais sobre silício (conhecidos na literatura como *System-on-Chip*, *SoC*) exige novas metodologias que permitam diminuir seu tempo de desenvolvimento. O projeto no nível de sistemas (SLD) é proposto para aumentar a eficiência do projeto de SoC. SLD exige novas linguagens (como SystemC) e níveis de abstração (como TLM).

A estrutura de comunicação (EC) de um SoC tem apresentado uma crescente importância devido à presença de uma maior quantidade (e funcionalidade) dos módulos a serem comunicados. Portanto, a EC apresenta um grande impacto no desempenho global do SoC.

Nesta tese é proposta uma metodologia de projeto da EC chamada de MaLOC ("*Multi-Abstraction Level On-Chip communications structure design*") que é baseada num enfoque *top-down* que percorre três níveis de transações (TLM). A tomada de decisões é feita utilizando-se duas importantes características que dão a originalidade a nossa proposta: 1) baseadas num conjunto de diversas métricas de desempenho que permite obter resultados mais confiáveis. 2) decisões ASAP (o mais rápido possível), antecipando a tomada de decisões utilizando níveis mais abstratos do que o RTL, permitindo diminuir o tempo de projeto da EC.

Para validar a proposta uma série de análises de fidelidade foram realizadas, os resultados indicaram fidelidades maiores do que 96% e em cenários extremos maiores do que 72%. Adicionalmente os tempos de simulação no nível TLM atemporal foi até 2,6 vezes mais rápido do que o nível TLM de precisão de transferências, que foi até 1,6 vezes mais rápido do que o nível TLM de ciclos de relógio (menos abstrato). Estes resultados indicam a validade da metodologia para realizar a tomada de decisões, permitindo uma melhor exploração do espaço de projeto

Os estudos de caso permitiram observar que além de configurar a EC procurando o melhor desempenho, MaLOC identificou soluções com menor consumo de energia, através do uso de um conjunto diverso de métricas, e configuração de parâmetros do sistema (tamanho da

memória). Estas duas situações indicam o potencial que a metodologia apresenta para o projeto de diferentes tipos de EC, assim como de diferentes componentes de um SoC.

ABSTRACT

Modern and future System on Chip design requires several methodologies in order to handle their growing complexity (of both functional and architectural issues). System Level Design has emerged as a solution to handle the complex of nowadays and future SoC designs, increasing their efficiency and reducing the time to market. SLD requires new modeling languages (such as SystemC) and abstraction levels (such as Transaction Level Modeling - TLM).

The integration of very different and composite IP cores into a SoC makes their physical and logical integration a very difficult task. Hence, the communication structure (CS) presents a significant impact on the SOC global performance.

This thesis proposes a novel methodology named MaLOC (Multi-Abstraction Level On-Chip communications structure design) that uses a top-down approach. The parameters configuration is driven by two important considerations: 1) performance metrics based, this enables to obtain a most reliable solution; 2) an ASAP configuration schedule, this enables to reduce the CS design time through the use of higher abstraction levels.

A fidelity test was performed. The results showed that in extreme conditions (such as burst size higher than time between transactions) the fidelity obtained was higher than 72%. In normal cases (burst size similar to the time between transactions) the fidelity was higher than 96%. The simulations execution times were compared among the three TLM levels and the results showed that TLM untimed simulations were 2.6 times faster than the TLM transfer accurate, also these were 1.6 times faster than the TLM cycle accurate. This means that TLM untimed simulations are 4 times faster than TLM Cycle accurate, enabling a enhanced space design exploration.

The case studies performed showed that MaLOC can be useful to identify solutions that satisfy the performance required reducing the power consumption (reducing activities across the bus). Also, a system parameter was defined using the methodology (memory banks). These two situations indicate the MaLOC potential to design several CS types and SoC configuration parameters.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de abreviaturas e siglas

AHB	*Advanced high performance bus*
AMBA	*Advanced microcontroller bus architecture*
APB	*Advanced peripheral bus*
ASB	*Advanced system bus*
ASIC	*Application specific integrated circuit*
CAD	*Computer aided design*
CCATB	*Cycle Count at Transaction Boundaries*
CI	Circuito Integrado
DMA	*Direct memory access*
DCR	*Device control register*
EC	Estrutura de comunicação
EDA	*Electronic design automation*
FIFO	*First-in First-out*
HDL	*Hardware description languaje*
IP	*Intellectual property*
MP-SoC	*Multi-Processor System on Chip*
NoC	*Network-on-chip*
OCP	*Open core protocol*
OPB	*On-chip peripheral bus*
OSCI	*Open SystemC initiative*
PBD	*Platform-based design*
PLB	*Processor local bus*
RTL	*Registrer transfer logic*
SLD	*System Level Design*
SoC	*System-on-Chip*
TDMA	*Time Division Multiple Access*
TLM	*Transaction level modeling*
ULA	Unidade lógico aritmética
VHDL	*VHSIC hardware description language*
VHSIC	*Very high speed integrated circuit*

1. Introdução

Os processos de fabricação de circuitos integrados permitem a integração de uma quantidade cada vez maior de componentes, tais como processadores, memórias, periféricos, sistemas de comunicações (*bluetooth*, *ethernet*), lógica especializada como criptografia, codificadores-decodificadores de áudio e vídeo; tudo sobre uma única pastilha de silício. Este paradigma de integração é conhecido pela designação de "sistema sobre silício" (do inglês *System-on-Silicon*, ou ainda *System-on-Chip, SoC*). Estes sistemas apresentam uma complexidade crescente, não apenas em termos de integração de componentes (quantidade e heterogeneidade), mas também de funcionalidade (comportamento), área, desempenho e consumo de energia (dispositivos portáteis).

Um *SoC* pode conter um ou mais processadores. Neste último caso um *SoC* é conhecido pela sigla *MP-SoC* (do inglês *Multi-Processor SoC*). O conjunto de processadores pode ser homogêneo ou heterogêneo[1]. A operação de *SoCs* e de *MP-SoCs* implica numa intensa troca de informações entre os componentes do sistema. Esta situação faz com que a estrutura de comunicação (EC)[2] apresente uma grande importância no desempenho, consumo de energia e no funcionamento do *SoC*.

No desenvolvimento de um *SoC* é importante utilizar uma metodologia de projeto que permita atingir as especificações e que contribua para diminuir os custos. Uma metodologia de projeto é composta por uma série de tarefas (veja capítulo 3) que permitem realizar um refinamento gradual do projeto desde sua idéia inicial (representada pela especificação do projeto) até sua implementação final no silício (seja *full-custom* ou *semi-custom*). Um dos aspectos essenciais deste refinamento consiste na criação de sucessivos modelos do projeto, que vão sendo cada vez mais detalhados à medida que o projeto avança em direção à sua implementação

[1] Diferentes conjunto de instruções e capacidade de processamento

[2] Nesta tese a sigla EC é utilizada para se referir ao termo "estrutura de comunicação"

final, tipicamente através de uma solução hardware-software.

As metodologias de projeto baseadas em síntese RTL[3], que utilizam linguagens de descrição de hardware como VHDL ou Verilog, têm se mostrado ineficientes como ponto de partida e de análise para projetos de grande tamanho, como são os SoCs. Isto é devido ao esforço no desenvolvimento dos modelos e o esforço computacional, representado pelo tempo utilizado nas simulações. Por exemplo, em [FAR07] a simulação de 1 segundo de execução de um processador de rede modelado no nível RTL demorou 58 horas e 24 minutos utilizando um equipamento com dois processadores *Opteron*. Isto indica que numa semana só duas condições são analisadas, num mês só oito condições, o que é ineficiente devido à complexidade da aplicação e à quantidade de parâmetros de configuração existentes. Esta ineficiência fica exposta na chamada "brecha na produtividade" (do inglês *productivity gap*). Isto é, o que é possível de projetar é inferior ao que pode ser integrado no silício, indicando que as metodologias baseadas na síntese RTL não são eficientes como ponto de entrada para o projeto de SoC (veja figura 1.1).

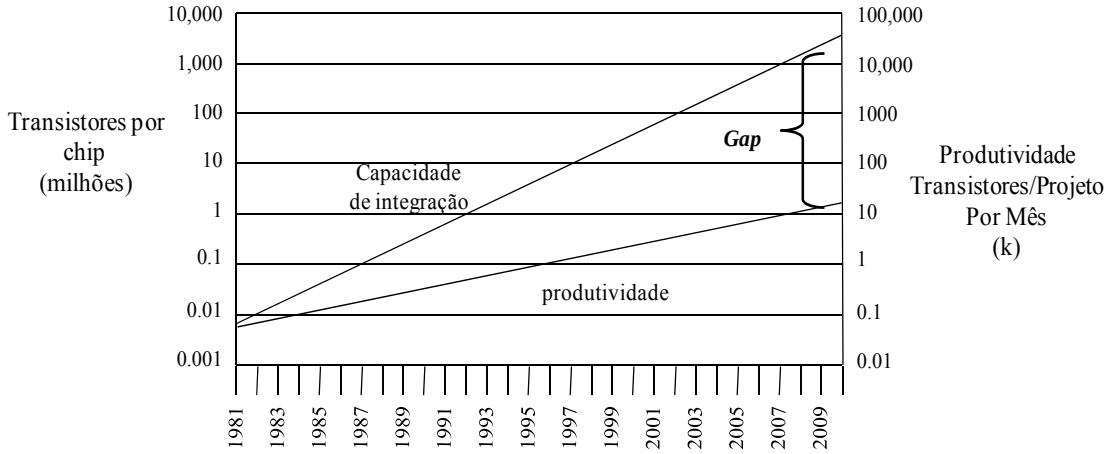

Figura 1.1. Brecha na produtividade.

Para solucionar esta brecha novos paradigmas de projeto são necessários, um dos

[3] A partir do qual se considera que é possível fazer a síntese do hardware até sua implementação física sobre silício.

quais é o projeto ao nível de sistema (do inglês *system level design* - SLD)[4]. Este nível é caracterizado por:

1. Uso de modelos em níveis de abstração além do nível RTL, criados através do uso de linguagens como C++/SystemC [SYSC], Java/Ptolemy [PTOL], C++/SpecC [SPEC], System Verilog [SVER].
2. Particionamento do projeto em:
    a. Funcionalidade – Arquitetura (metodologia)
    b. Hardware – Software (aplicação)
    c. Computação (processos) - comunicação (entre os processos)
3. Re-aproveitamento de núcleos de hardware[5] ou elementos de propriedade intelectual (do inglês *IP-Cores*)

A principal motivação para o uso de modelos em níveis de abstração além do RTL é permitir que o projeto seja rapidamente capturado e validado, isto é, que o modelo permita confirmar que a funcionalidade prevista nas especificações esteja presente ao longo do projeto. Outro motivo para o desenvolvimento de modelos em altos níveis de abstração é que eles possibilitam diminuir o esforço de projeto, através da antecipação da tomada de decisões de projeto que, de outra forma, seriam tomadas no nível RTL. Este é o aspecto central que será explorado no nosso trabalho e iremos nos referir a esta característica como "decisões ASAP" (sigla do inglês *as soon as possible*). Esta estratégia permite diminuir o tempo de desenvolvimento do projeto (*time-to-market*). A figura 1.2 apresenta os níveis de abstração presentes no projeto de *SoC*. As setas indicam os níveis que apresentam maior complexidade de modelagem e simulação, assim como os níveis de maior abstração.

[4] Neste texto a sigla SLD é utilizada para se referir ao termo "projeto ao nível de sistema".

[5] O re-aproveitamento sempre está presente no projeto de sistemas digitais, como as células padrão. No projeto de SoC, este é realizado num maior nível de complexidade dos componentes.

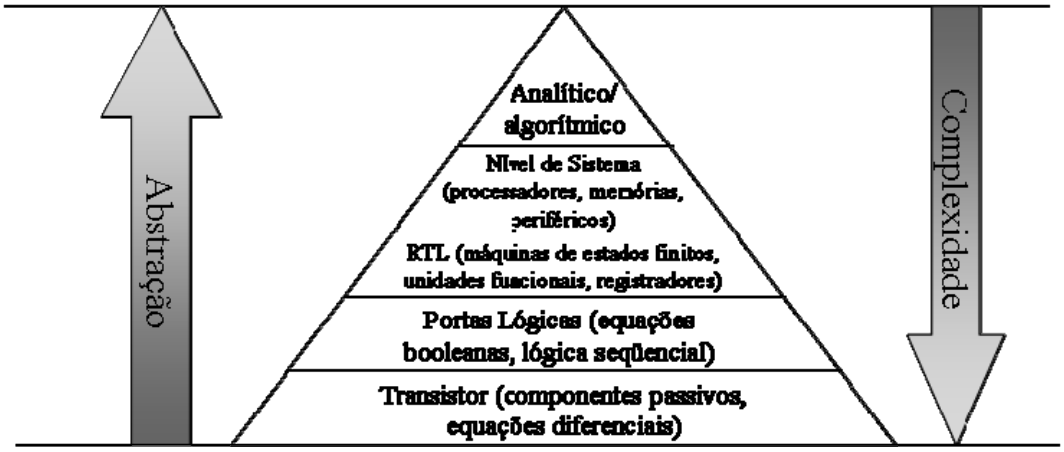

Figura 1.2. Abstração e complexidade no projeto de *SoC*.

O particionamento do projeto, também conhecido como ortogonalização de preocupações (do inglês *orthogonalization of concerns*) [KEU00], permite diminuir o esforço de projeto aplicando o princípio de *divide and conquer"*. A divisão em funcionalidade – arquitetura permite mapear os diferentes processos da aplicação nos elementos de hardware ou software do sistema que constituem a arquitetura proposta. A divisão em hardware e software aumenta a eficiência do projeto na medida em que permite dimensionar adequadamente os recursos do hardware às necessidades do projeto, por exemplo, a quantidade e tipos de processadores (de uso geral, de uso específico, de aplicação específica) a quantidade, tamanhos e tipos de memórias (RAM, FIFO, *buffers*). Outra importante vantagem é facilitar a integração destas duas partes, o que decorre da possibilidade de se especificar nas etapas iniciais do projeto as interfaces *hardware*/*software* que serão necessárias para esta finalidade, cuja funcionalidade pode ser validada através de co-simulações. Já a divisão do hardware em computação e comunicação permite projetar, refinar o otimizar cada uma destas partes separadamente, otimizando o reaproveitamento dos elementos de hardware (do inglês *IP hardware re-use*). O fato de se projetar separadamente permite que sejam feitas de forma concorrente, reduzindo o tempo do projeto.

Muitas decisões precisam ser tomadas ao longo do projeto. Dentre outras (que não cabem ser citadas aqui), queremos focalizar o problema da seleção dos recursos de hardware (tanto para a estrutura de computação como para a estrutura de

comunicação) que serão adotados para compor a arquitetura do sistema. É preciso definir tipos e quantidades de recursos. Para a estrutura de computação, por exemplo: 1- Quantos e quais microprocessadores de uso geral; 2- Quantos e para que funções serão adotadas microprocessadores de aplicação específica; 3- Tipos, quantidades e tamanhos das memórias; 4- Tipos e quantidades de periféricos; e assim por diante. Para a estrutura de comunicação (EC) é preciso definir: 1- Tipo da EC (conexões ponto-a-ponto, barramentos, redes intrachip[6]); 2- Interfaces/Protocolo da EC; 3- Componentes para cada tipo de EC, por exemplo, pontes e árbitros para barramentos, roteadores, buffers e interfaces para redes intrachip, e assim por diante. Além da seleção de tipos e quantidades de recursos de hardware, é necessário também configurar (instanciar) cada um, já que freqüentemente os recursos são disponibilizados de forma paramétrica. Por exemplo, a quantidade e o tamanho das memórias do sistema.

Todas as decisões que precisam ser tomadas ao longo do projeto (seja para o exemplo acima citado da seleção de tipos, quantidades e parâmetros dos recursos de hardware ou para qualquer outra decisão) devem ser orientadas para atingir a funcionalidade (através da verificação funcional) e o desempenho desejado (através de métricas de desempenho). Quanto mais cedo tais decisões forem tomadas, mais rapidamente o projeto será encerrado com sucesso diminuindo o *time-to-market*.

Para aumentar a eficiência da metodologia de projeto de *SoC* é necessário aumentar o nível de abstração além do RTL. Na figura 1.2 existe uma abstração a ser utilizada para realizar o projeto no nível SLD. A modelagem no nível de transações TLM (sigla do inglês *transaction level modeling*) foi proposta para satisfazer esta abstração. TLM é utilizada para realizar a modelagem tanto da estrutura de computação [CAL03] como da estrutura de comunicação [GRO02]. A modelagem TLM pode ser feita utilizando linguagens como SpecC [SPECC] e SystemC [SYSC]. O consórcio OSCI (*Open SystemC Initiative)* desenvolve e

[6] Também conhecido na literatura como NoC (do inglês *Network on Chip*). Nesta tese os termos NoC e redes intrachip são utilizados como sinônimos.

promove o uso do SystemC. Dentro do consórcio existe um grupo de trabalho responsável por definir e atualizar o padrão TLM [ROS05]. Diversos trabalhos acadêmicos propuseram diferentes níveis de abstração utilizando o conceito de transações [CAI03, CON03, HAV02, DON04, ATI07]. Cada nível permite representar diferentes características funcionais e temporais dos diferentes elementos do SoC. A literatura designa cada nível em função de sua característica temporal:

1) Atemporal.
2) Tempo estimado.
3) Precisão de ciclos de relógio.

Embora o nível de sistema seja muito referenciado na literatura, existe uma carência de ferramentas e metodologias que possam aproveitar as suas vantagens [BAI07].

Nesta tese é proposta uma metodologia para o projeto da EC. A estrutura de comunicação cumpre um papel importante no projeto de um *SoC*, porque realiza a integração física e lógica dos diferentes elementos (o sistema operacional realiza a integração no nível de aplicações). A EC também facilita o reaproveitamento dos elementos de hardware através do uso de interfaces padrão. O projeto da EC apresenta um grande espaço de soluções (veja capítulo 3 e 5), o qual decorre da quantidade de elementos que a compõem e de suas diferentes opções de configuração (parâmetros), assim como dos requisitos de desempenho a serem atingidos. Nossa metodologia explora ao máximo o nível *SLD.* Foram usados os 3 modelos TLM acima citados, todos descritos em *SystemC.* A nossa metodologia baseia-se na possibilidade de se realizar um grande número de simulações num curto espaço de tempo explorando diferentes configurações da EC para diferentes condições de tráfego (real e pseudo-aleatório) de forma a encontrar uma ou mais configurações que satisfaçam as especificações de desempenho exigidas para cada aplicação.

Neste trabalho foram propostas soluções para os seguintes problemas:

1) Análise de desempenho: Que métricas de desempenho podem ser adotadas, ou seja, quantificadas e qualificadas em cada um dos níveis de abstração TLM a fim de estimar o desempenho da EC?
2) Decisões de projeto: Que parâmetros de configuração da EC podem ser definidos em cada nível TLM (e, qual é a confiabilidade destas decisões)?

As contribuições obtidas nesta tese foram:

1) Demonstrar a possibilidade de se realizar o projeto da EC em níveis mais abstratos do que RTL, através de um refinamento progressivo através de 3 modelos todos no nível de transações.
2) Criar uma metodologia eficiente para projetar estruturas de comunicação de um *SoC* no nível de sistema.
3) Identificação das métricas de desempenho que podem ser calculadas a partir dos modelos da EC em cada nível TLM, assim como identificação dos parâmetros de configuração da EC capazes de serem definidos em cada nível TLM

No Capítulo 2 é apresentado o estado da arte sobre: a) modelagem TLM; b) análise de desempenho; e c) projeto da EC. No capítulo 3 descrevemos nossa visão sobre a metodologia para projetar uma EC assim como para projetar um *SoC*. No capítulo 4 são discutidos diferentes conceitos de transações e suas correlações com diferentes níveis de abstração. No capítulo 5 é apresentada a metodologia MaLOC, principal contribuição desta tese. No capítulo 6 são apresentados os resultados obtidos para diferentes estudos de caso. Finalmente as conclusões e propostas de trabalhos futuros estão no capítulo 7.

2. Trabalhos correlatos

Neste capítulo são apresentados os trabalhos publicados que possuem correlação com esta tese. A Seção 2.1 apresenta os trabalhos relacionados sobre modelagem no nível de transações (TLM). A Seção 2.2 apresenta os trabalhos relacionados sobre avaliação de desempenho a partir de diferentes modelos: analíticos (redes de Petri, cadeias de Markov e teoria de filas), nível de transações (TLM) e modelos RTL. A seção 2.3 apresenta os trabalhos relacionados sobre o projeto da EC (barramentos e/ou NoC). A seção 2.4 apresenta o sumário do capítulo e as contribuições propostas para esta tese.

2.1 Trabalhos sobre o nível de transações (TLM).

Em [CHA99] foram apresentadas as características de uma linguagem que descreve transações entre componentes virtuais (ou núcleos de hardware) através de um barramento (estrutura de comunicação). Esta linguagem poderia descrever diferentes níveis de abstração. Várias linhas de pseudocódigo são apresentadas sem identificar uma linguagem especifica para ser utilizada. Com a disponibilidade da versão 2.0 da linguagem SystemC, TLM se tornou um tópico de maior importância na área de projetos de *SoC* [GRO02]. Nesta versão existe o exemplo "*simple_bus*" desenvolvido pela Synopsys que apresenta as características básicas da modelagem TLM: portos, interfaces e canais, descrevendo a comunicação entre elementos mestres e escravos através de um barramento. O foco do *simple_bus* é apresentar como realizar a modelagem utilizando TLM e não detalha níveis de abstração.

Trabalhos posteriores começaram a apresentar TLM para realizar a modelagem de diferentes aspectos do projeto do *SoC*: a comunicação interna (*on-chip*) [HAV02], comunicação-computação [CAI03], hardware-software [CON03]. Nestes trabalhos, diferentes níveis de abstração são propostos: mensagem, transação,

transferência [HAV02]; atemporal, tempo aproximado e tempo de ciclos [CAI03]; visão do programador, visão do programador com tempo, função de ciclos [CON03]. Em [CAI03], os três níveis estão presentes tanto para a computação como para a comunicação, gerando diferentes pontos no espaço de projeto que foram designados por: conjunto de componentes, barramento com arbitragem, funcional do barramento.

Na referência [ESL04], as três propostas anteriores foram analisadas e comparadas. As conclusões foram que as três propostas apresentam características similares. O nível atemporal modela a comunicação e a computação através de um conjunto de elementos "perfeitos" que descrevem a funcionalidade do sistema, sem apresentar nenhum detalhe da arquitetura; 2) O nível de tempo estimado modela os diferentes elementos com um conjunto reduzido das suas características arquiteturais. Nenhum trabalho formaliza quais considerações são utilizadas para definir o que é modelado neste nível nem sua granularidade temporal. O nível de ciclos de relógio modela todas as características funcionais e arquiteturais do sistema (semelhante ao nível RTL). Cada sinal do sistema precisa ser representado no modelo. A diferença entre os níveis RTL e de ciclos de relógio decorre da forma como os sinais são modelados. No RTL eles estão explícitos, no TLM eles estão integrados dentro das funções utilizadas.

Em [KIM05] os autores mostram que TLM pode ter uma precisão de 97% em relação aos modelos RTL e que o poder computacional requerido é menor podendo a velocidade de simulação ser até 353 vezes superior em relação ao nível RTL.

Em [DON04] são apresentados 5 diferentes fluxos de projeto que evoluem ao longo de diferentes níveis TLM baseados nas propostas de [CAI03 e CAN03]. Os cinco diferentes fluxos de projeto dependem do grau de customização do hardware:

- Dominado por software: O hardware é fixo, a maior parte da funcionalidade do sistema é implementada em software.
- ASIC estruturado orientado por bibliotecas: A arquitetura é customizada através de diferentes núcleos de hardware, partindo de uma arquitetura inicial.
- ASIC estruturado *semi-custom*: O grau de customização é maior, representado numa maior quantidade de elementos que podem ser acrescentados e mais parâmetros de configuração. O grau de implementação da funcionalidade no hardware é de aproximadamente 10%.
- Projeto do processador e da EC: A customização do projeto é ainda maior, o grau de implementação da funcionalidade do sistema no hardware é de aproximadamente 50%.
- Projeto da arquitetura: Neste cenário todos os aspectos do hardware do sistema podem ser customizados para atingir as especificações.

Em [CLO06] é apresentado o nível de precisão de fases[7] que é uma particularização do modelo de precisão de ciclos. Os autores identificam que este nível pode ter ou não informações temporais da funcionalidade. Eles não utilizam modelos mais abstratos como o TLM atemporal de [CAI03], pois neste nível as variáveis não são acessadas pelo seu endereço. Isto não permite a co-simulação com o software do projeto (aspecto fundamental da proposta dos autores).

Em [PAS04] é apresentado um modelo chamado de CCATB (*Cycle Count at Transaction Boundaries).* Este modelo possui as características do nível TLM de tempo estimado. O autor melhora a precisão do modelo, inserindo informações do tempo de execução de cada transferência (leitura ou escrita) dentro do modelo. Os autores modelam uma EC baseada no barramento AMBA nos níveis CCATB e TLM com precisão de ciclos de relógio para apresentar a utilidade de sua

[7] Cada operação de escrita e leitura através de um barramento é composta por fases: acesso (*request*), endereço (*address*), envio/recepção de dados (*data*).

proposta, mas nenhuma análise é realizada.

Devido à grande diversidade de trabalhos e propostas sobre o nível TLM, foi criado um grupo de trabalho dentro do consórcio OSCI para definir um padrão para este nível. A versão 1.0 (abril 2005) definiu um conjunto de API (do inglês *application program interface*) para facilitar a modelagem. Nesta versão nenhum nível de abstração foi definido. A versão 2.0 (Junho de 2008) além de melhorar as API da versão 1.0 (para facilitar a interoperabilidade entre modelos desenvolvidos por diferentes projetistas e/ou vendedores), descreve dois estilos de codificação que podem servir de base para criar modelos em diferentes níveis de abstração. A diferença entre os estilos é definida por dois fatores: os pontos de sincronismo temporal das transações e o desacoplamento temporal de funções. No estilo mais abstrato, tempo impreciso (*loosely-timed*), o sincronismo é feito só no começo e no fim da transação. Utilizar poucos pontos de sincronismo permite aumentar o desempenho da simulação. Adicionalmente este estilo permite realizar um desacoplamento temporal das funções, isto é, que sejam executadas funções de maneira concorrente até o seguinte ponto de sincronismo, independente do tempo simulado. O segundo estilo é o tempo aproximado (*approximately-timed*). O sincronismo é feito utilizando uma maior quantidade de pontos. As funções não podem ser desacopladas, elas devem ser executadas em sintonia com a evolução da simulação. Este estilo oferece uma maior precisão com um menor desempenho da simulação. Estes dois estilos podem ser usados por diferentes níveis de abstração.

2.1.1 Sumário

O uso de TLM foi proposto para realizar a modelagem de diversos componentes de um *SoC*: computação, comunicação (hardware) e software. Diversos trabalhos indicam a existência de diferentes níveis de abstração. Desde o nível mais abstrato (o atemporal) até o mais preciso (ciclos de relógio). Entre estes dois níveis diferentes níveis foram propostos com diversas considerações de acordo com o

objetivo da modelagem.

O uso de TLM facilita desenvolver e utilizar modelos mais abstratos que permitem e podem ser utilizados na tarefa de verificação presentes ao longo do projeto. Com o uso do TLM também é possível antecipar o projeto do software para que seja realizado ao mesmo tempo que o projeto do hardware. TLM é composto tanto pelo estilo de modelagem (padrão 1.0 e 2.0) e diferentes níveis de abstração além do RTL com características funcionais diferentes. Todas estas considerações permitem diminuir o tempo do projeto de um *SoC*.

2.2 Trabalhos sobre avaliação de desempenho.

O uso de modelos analíticos para estimar o desempenho é amplamente utilizado em sistemas computacionais [KAN92]. Este tipo de modelo também é utilizado para realizar a modelagem e a análise de desempenho da EC de um *SoC* [BLU04, KAL03, PAN05a].

Em [BLU04] é apresentado o uso de redes de Petri para estimar o desempenho da estrutura de comunicação de um *SoC*. Dois exemplos são apresentados. O primeiro modela o comportamento de um DSP comercial (TMS320C6416). Os autores modelam uma transformada rápida de Fourier (FFT) e uma operação de turbo compressão (Viterbi). Os tempos de execução da FFT foram estimados, assim como das operações de turbo compressão, sendo os resultados comparados com medições feitas na placa. No segundo exemplo apresentam a modelagem de um barramento em diferentes condições da política de arbitragem. Os autores apresentam o nível de utilização do barramento em função das faixas de tempo da política de arbitragem (dois níveis - TDMA). Os resultados obtidos não são utilizados para realizar o dimensionamento de algum parâmetro de configuração da EC.

Os autores de [PAN05a] utilizam cadeias de Markov para a modelagem do comportamento da comunicação entre processos. A métrica de desempenho adotada é a latência, representada pelo atraso na comunicação de ida e retorno entre o mestre do processamento e a memória (*round trip communication delay)*. Os resultados obtidos são utilizados na definição da largura do barramento.

Os autores de [KAL03] modelam um DSP da Hitachi utilizando teoria de filas. Nesse trabalho implementam diferentes políticas de arbitragem e avaliam o desempenho do sistema com cada uma. A latência e o *throughput* de dados são estimados em função do nível de preenchimento de cada fila utilizada no sistema. Os resultados obtidos não são utilizados para realizar o dimensionamento de nenhum parâmetro de configuração da EC.

Em [FAR07], um sistema processador de rede é modelado usando o modelo analítico de *network calculus*. Nesta análise, diferentes parâmetros do sistema são definidos incluindo a configuração da EC (barramento): mapeamento de elementos, largura do barramento, configuração da política de arbitragem. Os resultados desta análise são depois validados através de uma comparação com os resultados obtidos na simulação do VHDL-RTL do sistema. O fato de esta metodologia requerer uma iteração entre os dois níveis de análise indica uma baixa flexibilidade para analisar diferentes topologias do processador de rede, aumentando o tempo de projeto.

Em [ESL03], um modelo de EC (barramentos) no nível TLM foi utilizado para obter estimativas de seu desempenho a partir de simulações. As métricas utilizadas foram: latência das transferências, *throughput* de dados, *throughput* de transações, nível de utilização do barramento, e participação dos elementos mestres no uso do barramento. Nenhum critério de análise de resultados foi apresentado.

Em [DON04], o autor indica que a análise de desempenho é realizada nos níveis

TLM temporais (tempo estimado e ciclos de relógio), em qualquer um dos fluxos. O autor não apresenta nenhum detalhe de como realizar esta tarefa (métricas e critérios de análise).

Em [PAS04], o modelo da EC é chamado de CCATB (*cycle count accurate boundaries*) (veja seção 2.1). Os autores realizam uma comparação entre diferentes políticas de arbitragem (baseada em prioridades, aleatório, *round-robin*, TDMA), protocolos (AMBA 2 – AHB e AMBA 3 – AXB) e topologias (diferentes conjuntos de barramentos hierárquicos), utilizando 3 diferentes *benchmarks* (*COMPLY*, *USBDRV*, *SWITRN*). As métricas estimadas são *throughput* de transações e influência da arbitragem sobre a latência (representada pela execução das tarefas). As análises apresentadas comparam os tempos de simulação dos modelos utilizados.

Na análise de desempenho os geradores de tráfego são essenciais para garantir resultados adequados [KAN92]. Quatro tipos diferentes de geradores de tráfego podem ser utilizados: determinísticos, aleatórios, baseados em traces e aplicação. Estes são empregados em diferentes trabalhos. Em [ESL04] são utilizados geradores determinísticos, em [LOZ05] geradores aleatórios. Os geradores de tráfego utilizados nos dois trabalhos anteriores possuem vários parâmetros que podem ser modificados: o tamanho da rajada e o tempo entre transações. A escolha e quantificação de cada parâmetro estão integradas dentro da metodologia utilizada para realizar a avaliação de desempenho apresentada em [ESL03]. Em [SEP06] estes tipos de geradores de tráfego foram utilizados para realizar a análise de desempenho de uma rede intra-chip (NoC). No trabalho, vários critérios de análise foram apresentados como: valores ótimos, padrões de qualidade e análise a partir da especificação. Estes critérios são aplicados aos resultados obtidos durante as estimativas de desempenho.

Em [TED05] são utilizados geradores de tráfego aleatórios para a avaliação de desempenho de uma *NoC*. Os parâmetros utilizados para os geradores são:

distribuição espacial de pacotes, taxa de injeção de pacotes e tamanho do pacote. As métricas utilizadas são *throughput*, taxa de uso dos canais, latência e tempo ocioso (*idle*) dos canais. O modelo da NoC é no nível RTL (VHDL).

Em [ATI07] é apresentado um *framework* para realizar a análise de desempenho utilizando dois níveis TLM baseados no nível de visão de programador com tempo (PVT) de [CON03]. Os níveis são: PVT-TA (*PVT transaction accurate*), vista de programador com precisão de transações e o PVT-EA (*PVT event accurate*) vista do programador com precisão de eventos. Eles realizam uma análise de desempenho de um sistema MP*SoC* executando diferentes *benchmarks* como: multiplicação de matrizes, *downscaler* de alta definição, DCT e decodificador H263. As métricas analisadas referiram-se ao desempenho da simulação e não ao do *SoC*: reducao do tempo da simulação (*speedup*), precisão das estimativas e esforço para gerar os diferentes modelos. Nenhuma métrica sobre o desempenho do sistema e sua utilidade é apresentada.

Em [LOG04] os autores estudaram as características do tráfego através da EC, utilizando a comparação de duas diferentes estruturas de comunicação baseadas em barramento AMBA e em *crossbar* STBUS. Os autores desenvolveram uma plataforma de simulação que permite uma completa simulação do sistema. Portanto, o tráfego presente na EC é baseado na aplicação. Os modelos utilizados são TLM com precisão de ciclos (*cycle-accurate*) (da biblioteca *designware* da Synopsys) e SystemC RTL.

2.2.1 Sumário

Métricas de desempenho são bastante mencionadas na literatura sobre análise de desempenho de sistemas computacionais [KAI92]. Estas podem ser obtidas com diferentes graus de precisão, utilizando modelos analíticos como redes de Petri ou cadeias de Markov que apresentam tempos curtos de desenvolvimento. Estes modelos são específicos para a tarefa alvo, isto é, eles não são reaproveitados no

projeto. Modelos simuláveis em diferentes níveis de abstração como o RTL ou TLM, também podem ser utilizados para estimar o desempenho. Estes apresentam um melhor reaproveitamento em diferentes tarefas do projeto, com diferentes graus de precisão. A utilidade das métricas de desempenho está presente no momento em que decisões de projeto úteis sejam tomadas a partir dos valores obtidos.

2.3 Propostas sobre o projeto da EC

Em [DRI00] é estudado o projeto da EC e o mapeamento de elementos no barramento. Os autores realizam o projeto da EC (barramentos) a partir da análise do tráfego (*communication profiler*). Depois, diferentes algoritmos são aplicados para projetar a EC (diminuindo a latência) e realizar o *floorplanning* do projeto (diminuir a área do projeto). Os núcleos de hardware utilizado são de tipo *hard-core*, isto implica um grande esforço computacional para sua simulação. Os parâmetros da EC definidos são: quantidade de barramentos e a distribuição de elementos por barramento. A política de arbitragem e sua configuração não são definidos.

Em [LAH01] os autores propõem uma técnica de análise de desempenho de uma EC utilizando geradores de tráfego baseados em *traces* que são obtidos de uma simulação inicial (da especificação executável). A técnica utiliza duas ferramentas: PTOLEMY [PTOL] para gerar um modelo do sistema que permite a exploração arquitetural e POLIS [POLI] para realizar a co-simulação dos modelos hardware/software do sistema. A geração deste modelo (com a ferramenta POLIS) requer uma tarefa adicional no projeto, mas deste resultado são extraídas informações de tempo que vão compor os *traces*. As comparações feitas entre os dois modelos (arquitetural e Hw/Sw) correspondem aos tempos de simulação. Os autores mencionaram que a técnica proposta por eles apresentou uma diferença de até 228 vezes no tempo da simulação com os *traces* do que simulando o sistema

completo. As métricas utilizadas foram: a latência e o nível de utilização de barramentos. O resultado é um relatório indicando os conflitos de acesso ao barramento e o caminho crítico na execução das tarefas. As tarefas apresentadas para o projeto da EC são: 1) Escolha da topologia, baseada em barramentos (hierarquia); 2) Mapeamento da comunicação (em ECs pré-definidas); 3) Customização dos protocolos utilizados.

Em [PAN05] é apresentada uma metodologia de projeto da EC. A proposta do autor gera uma EC customizada baseada em barramento. Três tarefas são propostas: 1) Identificação da topologia e tamanho do barramento; 2) Definição do protocolo de comunicação; 3) Síntese das interfaces. O ponto de partida neste fluxo é o particionamento hardware/software e o mapeamento do software nos diferentes elementos de hardware. O comportamento da comunicação é modelado através de intervalos de vida da comunicação (do inglês *communication lifetime interval -CLTI*) que é composto por um tempo de início e um tempo final para transmitir certa quantidade de dados num determinado tempo. É assumido que todas as transferências possuem o mesmo tamanho (medido em bytes). Vários algoritmos de programação linear são utilizados para definir a topologia baseada em barramentos e a largura. As métricas utilizadas não são informadas. Esta proposta gera estruturas customizadas tanto dos elementos físicos como dos lógicos, oferecendo um menor nível de reaproveitamento do que o uso de elementos padrão.

Os autores de [ZER04] apresentam uma metodologia de projeto de *SoC* que se caracteriza por possuir uma arquitetura alvo (plataforma) que é composta por processadores, memórias e um conjunto de ECs (barramentos e/ou redes complexas, NoCs). A arquitetura alvo para cada aplicação é customizada através do dimensionamento de parâmetros como: o número de processadores, o tamanho das memórias, os portos de E/S, a interconexão entre os processadores, os protocolos de interconexão. Três níveis de abstração são apresentados: nível de sistema, nível de macro-arquitetura e nível de micro-arquitetura. A geração da EC

consiste na implementação de *wrappers* (nos elementos de hardware) e a definição (quantidade) e implementação (definição de atributos) dos canais de comunicação. Isto acontece no momento de descer do nível macro-arquitetura ao nível de micro-arquitetura. A definição dos parâmetros da EC é uma tarefa desenvolvida pelo projetista. Nenhuma técnica para o dimensionamento dos parâmetros foi apresentada. Nenhuma métrica de desempenho foi apresentada.

Os autores de [GOO05] apresentam um fluxo para gerar e configurar NoCs customizadas. Eles dividem o fluxo em três tarefas: 1) Geração; 2) Configuração; 3) Validação. Os autores customizam a NoC definindo automaticamente a quantidade de canais e de roteadores. Finalmente um código SystemC é produzido para a simulação e um código RTL VHDL para a síntese. Na tarefa de validação é realizada automaticamente uma avaliação de desempenho para verificar se o *throughput*, a máxima latência e o tamanho dos buffers especificados foram atingidos.

A referência [SCH06] apresenta uma comparação entre diferentes níveis TLM modelados com a linguagem SpecC. As características dos níveis utilizados não são explicadas. O compromisso entre velocidade – precisão é avaliado, através da comparação de só uma característica da comunicação, a contenção. Esta é definida como a porcentagem de superposição entre duas transações. Comparações de outras características da comunicação ou de outras métricas não foram realizadas.

2.4 Sumário

Este capítulo apresentou trabalhos que discutem a existência e as características dos diferentes níveis TLM, apresentando sua evolução desde o conceito inicial [CHA99] até a definição do padrão TLM pelo consórcio OSCI [SYSC].

Também apresentou trabalhos que obtiveram estimativas de desempenho de EC,

utilizando modelos analíticos (teoria de filas, redes de Petri) e modelos baseados nos diferentes níveis TLM e no RTL. Nestes trabalhos, foi apresentado o procedimento para obter os valores das métricas de desempenho. Só num deles os resultados foram utilizados para dimensionar um parâmetro de configuração da EC: largura do barramento utilizando modelos analíticos (cadeias de Markov) [PAN05a]. Em outro trabalho foi utilizado *Network calculus* para configurar: mapeamento de elementos, largura do barramento, configuração da política de arbitragem [FAR07].

Estimativas de desempenho também podem ser obtidas a partir de uma plataforma de desenvolvimento [LOG04, ZER04, DON04, GOO05]. Os modelos utilizados nestas plataformas são RTL com um conjunto de simuladores da arquitetura de instruções (do inglês *instruction set simulator*). Estes tipos de plataformas requerem um grande poder computacional para realizar a simulação, portanto, simulações sucessivas destas plataformas, para ajustar algum parâmetro, vão aumentar o tempo de desenvolvimento do projeto. Nestas plataformas o projeto da EC está limitado ao ajuste de alguns parâmetros, como o tamanho do barramento e ajustes nos protocolos, não oferecendo flexibilidade para dimensionar adequadamente a EC.

Três autores [LAH01, PAN05, GOO05] apresentaram diferentes propostas para realizar o projeto da EC, baseados em barramento e em NoC. Outro trabalho [ZER04] apresenta o projeto da EC integrado no projeto de um *SoC*, através de uma plataforma de desenvolvimento.

Em [SCH06] é realizada uma comparação entre diferentes níveis TLM modelados com a linguagem SpecC. Este trabalho só modelou uma condição específica de tráfego, superposição. Esta condição não é função de nenhum parâmetro do sistema ou da EC, portanto, ela pode ou não acontecer a qualquer momento da execução do sistema. O trabalho não avalia outras métricas que podem ser utilizadas, portanto, os resultados obtidos são restritos e refletem um conjunto

limitado de condições de operação.

Nosso trabalho é fundamentado em três dissertações que foram desenvolvidas no nosso grupo de trabalho (GSEIS-LME).

1. Em [ESL03] é estudado o problema da avaliação de desempenho de uma EC baseada em barramentos hierárquicos a partir de modelos de alto nível de abstração. Foram definidos um conjunto de métricas e suas equações. Também foram definidas as características dos geradores de tráfego (paramétricos) e um modelo da comunicação.
2. Em [LOZ05] o estudo foi continuado utilizando geradores de tráfego pseudo-aleatórios. Foram adicionadas novas métricas de desempenho assim como uma classificação das características do tráfego através do barramento (intenso, moderado).
3. Finalmente em [SEP06] foi realizado um estudo sobre EC baseadas em redes intrachip (*NoC*), identificando um conjunto de métricas (com as suas equações), características dos geradores de tráfego (paramétricos e pseudo-aleatórios), assim como critérios de análise baseados em valores ótimos e em padrões de qualidade.

O objetivo de nosso trabalho foi desenvolver uma metodologia de projeto da EC orientada pelos resultados da análise de desempenho. Esta análise de desempenho é realizada de forma gradual a partir de modelos mais abstratos do que o RTL. Os aspectos que conferem originalidade ao nosso trabalho em relação aos trabalhos já publicados são:

1. Níveis de abstração: Consideramos a existência de 3 (três) níveis de abstração TLM, o que permitiu criar modelos mais simples para a *EC*.
2. Métricas de desempenho: Consideramos um amplo espectro de métricas globais e locais o que ampliou a confiabilidade das decisões tomadas.

3. Parâmetros de configuração: Estabelecemos a associação entre as métricas de desempenho (por nível de abstração) e o máximo número de parâmetros de configuração.
4. Decisões ASAP: Identificamos um conjunto de métricas por nível de abstração TLM o que permitiu antecipar decisões de projeto que, de outra forma seriam tomadas em níveis mais baixos.

Este conjunto de contribuições associado à existência de um ambiente que permite obter rapidamente os parâmetros de desempenho para uma grande variedade de tráfego [ESL07, ESL08], leva a uma maior eficiência (medida em tempo de projeto) do projeto de estruturas de comunicação de um *SoC*.

3. Projeto de sistemas sobre silício.

Neste capítulo, os principais conceitos relacionados ao projeto da estrutura de comunicação (EC) são apresentados na seção 3.1. A seção 3.2 apresenta diversos tipos de EC disponíveis na literatura, enfatizando os barramentos hierárquicos e as redes intrachip (NoC). A seção 3.3 apresenta o fluxo de projeto de um *SoC*, destacando o projeto da EC. Finalmente, na seção 3.4 é apresentado o sumário do capítulo.

3.1. Projeto da Estrutura de Comunicação

A estrutura de comunicação realiza a conexão física e lógica entre os diversos componentes de hardware do sistema. Cada tipo de EC é composto por um conjunto diverso de parâmetros. Estes parâmetros podem ser classificados como: 1) parâmetros físicos que são os componentes que permitem estabelecer o caminho para a comunicação entre os diferentes elementos, como trilhas condutoras, pontes, árbitros e roteadores; e 2) atributos lógicos que são os componentes que permitem que os elementos possam estabelecer a comunicação, como: os protocolos, os sinais de controle, as interfaces, política do arbitro, protocolo de acesso, algoritmo de roteamento, fluxo de controle e os modos especiais de comunicação.

A EC realiza um conjunto de tarefas para permitir a comunicação entre o mestre[8] e o escravo[9]; como: 1) Escolha do mestre que pode realizar a comunicação[10]; 2)

[8] Exemplos de elementos mestres são processadores (de propósito geral), processadores de sinais (DSP), processadores gráficos (GPU), micro-controladores, controladores de acessos diretos a memória (DMA)

[9] Exemplos de elementos escravos são memórias, periféricos, controladores de rede (*Ethernet*, *Bluetooth*).

identificação do escravo que vai ser acessado; 3) estabelecimento do caminho. EC baseadas em barramentos apresentam condições diferentes para realizar estas operações (leituras e escritas) do que em NoCs. No caso da NoC, existe o conceito de pacote como unidade para realizar a transferência. Um pacote é uma unidade auto-contida[11] que leva os dados da fonte até seu destino. No caso de barramentos, o conceito de pacote não se aplica, porque dentro dos barramentos existe um conjunto de trilhas dedicadas que apresentam explicitamente as informações ou condições necessárias para estabelecer a comunicação (tipo da operação, sinais de *handshake*, endereço, controle). A configuração interna de cada EC também pode acrescentar novos eventos para realizar a comunicação, isto é, a presença de roteadores (no caso da NoC) e diferentes pontes (no caso de barramentos) pode ocasionar acessos a diferentes elementos, ocasionando novos eventos. Estes eventos são locais (entre dois roteadores, entre uma ponte e um barramento) e sua soma constitui a comunicação entre o elemento mestre (fonte) e o escravo (alvo).

O projeto da EC apresenta uma grande complexidade devido à quantidade de parâmetros a serem definidos dentro de seu espaço de projeto. O projeto da estrutura de comunicação começou a ser discutido desde o início da "revolução" do *SoC* [CHA99] através da geração de *wrappers* para permitir a interconexão entre elementos heterogêneos. Depois foi introduzido o uso de interfaces padrão como *OCP (Open Core Protocol)* [OCP01] e *VSIA (Virtual SoCket Alliance Interface)* [VSIA], que definem as características lógicas da EC, deixando os elementos físicos a critério do projetista. Vários conjuntos de barramentos com suas interfaces formalizadas também foram propostos, como *AMBA* [ARM99] e *CoreConnect* [COR99]. Uma importante vantagem que o uso de interfaces padrão apresentou foi facilitar um maior reaproveitamento dos componentes (*hardware reuse*). Posteriormente, as redes intrachip ou NoC (do inglês *Network-on-Chip*) [BEN02] foram propostas como uma nova alternativa de EC, um exemplo de NoC comercial é a *arteris* [ARTE]. A figura 3.6 apresenta quando o fluxo da EC é

[10] Em caso de haver contenção, isto é, vários elementos pedindo acesso ao mesmo barramento.

[11] Que inclui toda informação necessária para que os dados cheguem ao seu destino

realizado dentro do projeto de um *SoC* (demarcado pelas linhas pretas).

O projeto da EC parte de um conjunto de informações previamente definidas no projeto [STR07].

- Requisitos da especificação: Quais são as restrições que a EC deve satisfazer.
- Descrição da comunicação entre processos: Listagem da comunicação entre os diferentes elementos de processamento. Obtida a partir de uma análise do sistema feita no projeto funcional ou da análise dos geradores de tráfego num cenário de tráfego independente da aplicação.
- Elementos de hardware: Quais são os elementos que estão ligados à EC, esta informação é definida após o particionamento hardware/software na etapa de definição da arquitetura do *SoC*.
- Biblioteca de elementos: Quais são os componentes disponíveis para definir e configurar a EC.

A partir destas informações é realizada uma análise da comunicação que permite identificar e classificar o comportamento da mesma, permitindo tomar diferentes decisões de projeto. Das diferentes propostas de fluxos de projeto da EC [LAH01, PAN05, ZER04, GOO05] analisadas, três grupos de tarefas são identificados.

1. Definição do tipo da EC (barramento, NoC, outros)
2. Definição do tamanho da EC.
3. Configuração dos protocolos.

A escolha do tipo de EC é baseada em diferentes critérios pragmáticos como: quantidade de componentes, aplicação e/ou custo. Nesta escolha também influi a disponibilidade de elementos e o conhecimento prévio do projetista (muito subjetivo).

A tarefa 2 define o tamanho da EC, No caso de NoC inclui definir sua topologia (*mesh*, *torus*, anel, outros) e o número de roteadores, no caso de H-BUS, inclui definir a quantidade de barramentos e pontes. Esta tarefa pode ser feita utilizando planilhas (*templates*) [ZER04] ou pode ser customizada [PAN05]. Para realizar esta escolha diferentes técnicas são utilizadas, como algoritmos a partir da análise do tráfego [DRI00], geração de *traces* a partir da especificação executável [LAH01].

As tarefas do grupo 3 definem: 1) quais elementos (mestres e escravos) vão ser mapeados em cada barramento ou em cada roteador da NoC, o que pode ser feito através da análise da comunicação usando *traces* [LAH01], utilizando *templates* [PAN05] ou algoritmos de otimização [DRI00]; 2) O protocolo a ser utilizado na EC que pode ser customizado [PAN05] ou comercial (por exemplo, AMBA ou CoreConnect) ou disponível como código aberto (OCP). 3) A configuração e implementação das interfaces o que depende do tipo de protocolo a ser utilizado.

A figura 3.1 apresenta um fluxo de projeto "genérico" de uma EC (extraído das diferentes propostas encontradas na literatura).

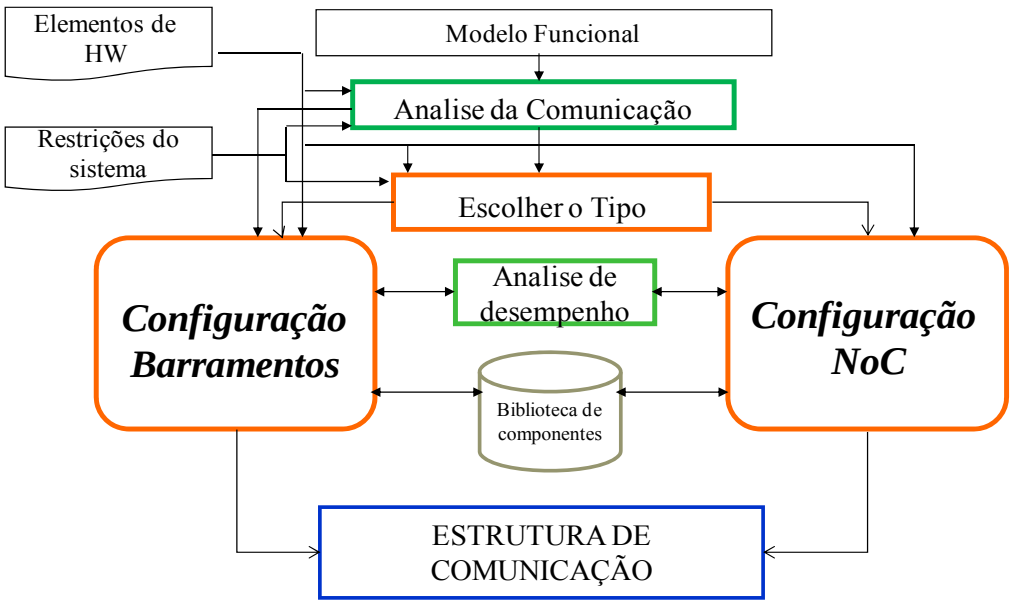

Figura 3.1. Projeto da estrutura de comunicação.

3.2. Tipos de EC

A crescente complexidade dos *SoC* representada pelo aumento de componentes integrados e o seu poder computacional, implica num maior tráfego *on-chip*. Num *SoC* a comunicação é composta por:

- Troca de dados: Informação processada ou a ser processada que é transmitida da fonte (mestre) para seu destino (escravo). Esta transmissão é feita através de operações de leitura (*load*) e de escrita (*store*) com diferentes comprimentos. Inclui os endereços
- Sincronismo: Comunicação ocasionada pela política de coerência da memória e por sincronismo entre as tarefas executadas em paralelo que apresentam alguma dependência.
- Controle: Sinais que ajudam a estabelecer o caminho dos dados que vão ser transferidos.

Em [BAI07], o domínio da comunicação é analisado[12], definindo vários níveis da comunicação (figura 3.2):

- Canais ponto a ponto (*point-to-point*): A forma de comunicação mais simples entre dois elementos sem nenhum controle.
- Buffers (*buffered*): A forma de comunicação mais simples entre dois elementos com controle. Oferece um isolamento elétrico e temporal entre os elementos que se comunicam.
- Co-processador (*coprocessor*): Acontece quando existe uma relação de produtor-consumidor de dados entre dois elementos (dados compartilhados). Neste caso existe um fluxo bidirecional de informação incluindo controle.

[12] O projeto de um SoC pode ser dividido em diferentes domínios, veja seção 3.3

- Memória (*memory*): Neste caso os elementos produtor e consumidor estão ligados direto na memória (multi-porto), permitindo acessar e armazenar dados em diferentes velocidades.
- Baseado em barramentos internos (*bus based – high speed*): A comunicação entre os elementos é feita através de memórias que são acessadas usando os barramentos internos (as NoC pertencem a este grupo).
- Baseado em barramentos externos (*bus based – low speed*): A comunicação pode ser feita entre elementos *on-chip* e *off-chip*, portanto, barramentos como USB ou PCI são utilizados.
- Nenhum: Quando os elementos não precisam se comunicar, isto é, não existe nenhuma dependência (ou sincronismo) entre eles.

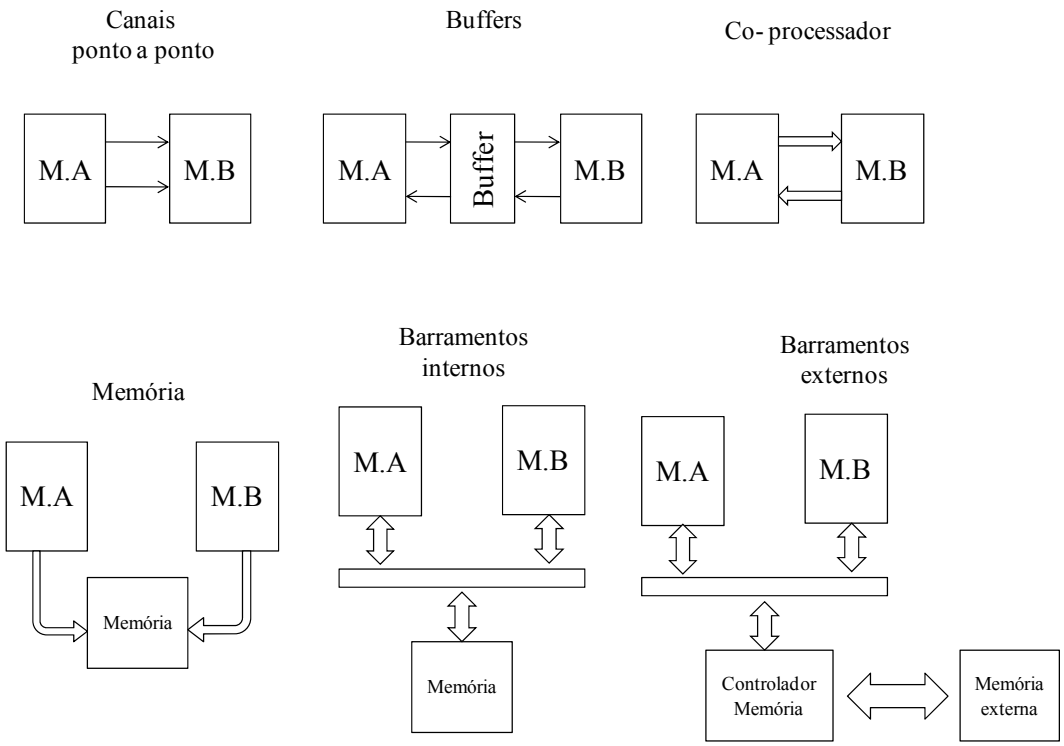

Figura 3.2. Abstração das formas de comunicação segundo [BAI07]

Dentro de um *SoC* é possível ter qualquer uma das EC anteriores, mas os tipos mais utilizados são: barramentos internos (simples e hierárquicos) porque permitem um maior reaproveitamento de elementos [GRO02]. Nesta tese focalizamos os barramentos hierárquicos.

3.2.1 Espaço de projeto usando barramentos.

Os barramentos são o tipo de EC mais utilizado no projeto de *SoC*. Um barramento é composto por um conjunto de trilhas condutoras que são compartilhadas por todos os elementos a ele conectados. Nestas trilhas podem trafegar: 1) os dados a serem transferidos; 2) o endereço a ser acessado e 3) os sinais de controle necessários para estabelecer a comunicação. Um barramento simples é aquele que liga todos os elementos de hardware do sistema, apresentando uma menor complexidade no seu projeto do que barramentos hierárquicos ou NoC. Os simples são muito utilizados em *SoC* de tipo mono-mestres.

Quando a quantidade de elementos a serem interligados apresenta vários elementos mestres, a comunicação apresenta uma maior complexidade. Portanto é recomendável utilizar barramentos hierárquicos que permitem aumentar o desempenho. Este tipo de EC apresenta um conjunto de barramentos que são interligados através de pontes. A figura 3.3 apresenta uma EC baseada em barramentos simples e hierárquicos.

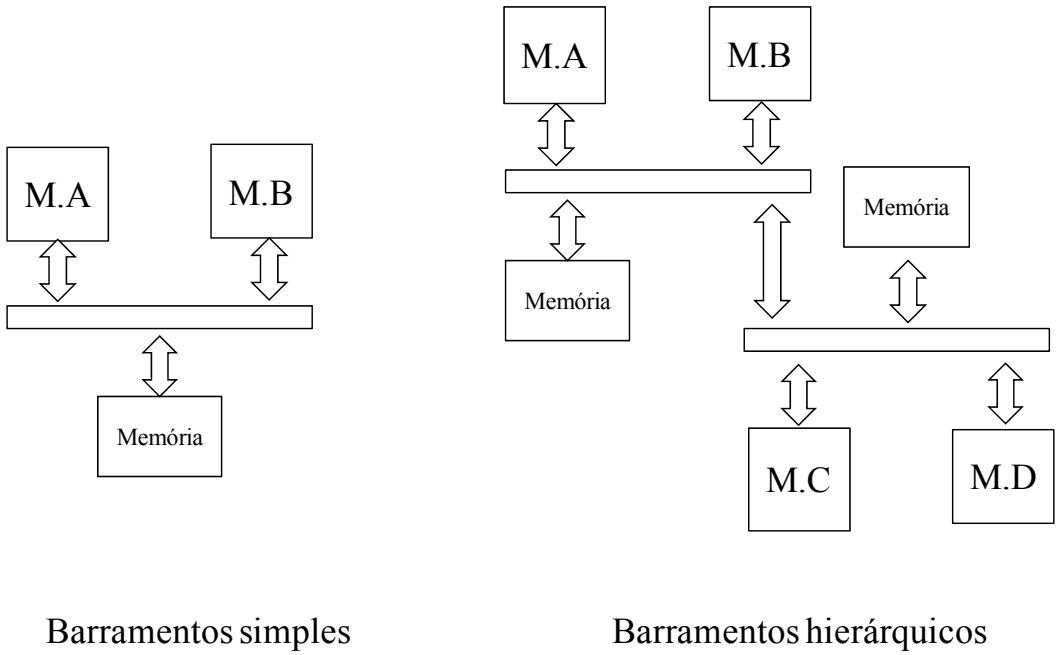

Figura 3.3. EC baseada em barramentos simples e hierárquicos

Existe uma grande diversidade e variedade de barramentos comerciais que apresentam diferentes características, o que se reflete no espaço de projeto [AMB99, COR99]. Na tabela 3.1 são apresentados os parâmetros comuns aos diferentes barramentos.

Tabela 3.1. Espaço de projeto utilizando barramentos.

Componentes	Parâmetro	Opções
Físicos	Número de barramentos	1.....N
	Elementos por barramento	1.....N
	Tipo de barramento	Alta velocidade Baixa velocidade
	Interconexão entre barramentos	Pontes (número e configuração) *Crossbars*.
	Política de Arbitragem	Prioridades fixas (prioridades) *Round-Robin* (ordem) Dois níveis (identificação, faixa de tempo) *TDMA* (ordem, faixa de tempo) Customizados
	Identificação Mestres	1.....N
	Largura do barramento	32, 64, 128, 256 bits
Lógicos	Acesso ao barramento	Tipo de Interfaces
	Pipeline.	N Estágios
	Transferências avançadas	*Split Transfer*
	Rajadas	Básicas Avançadas

3.2.2 Espaço de projeto usando NoC

As redes intrachip (NoC) apresentam uma maior complexidade de operação do que os barramentos, pois incorporam funcionalidades adicionais como política de roteamento, controle de fluxo e técnica de comutação, para poder estabelecer a comunicação. Existem diferentes topologias de redes intrachip, como: *mesh*, *torus, hipercube, octagon,* anéis, árvore. Cada uma apresenta diferentes conexões entre os roteadores presentes na NoC. A figura 3.4 apresenta diferentes topologias de NoC.

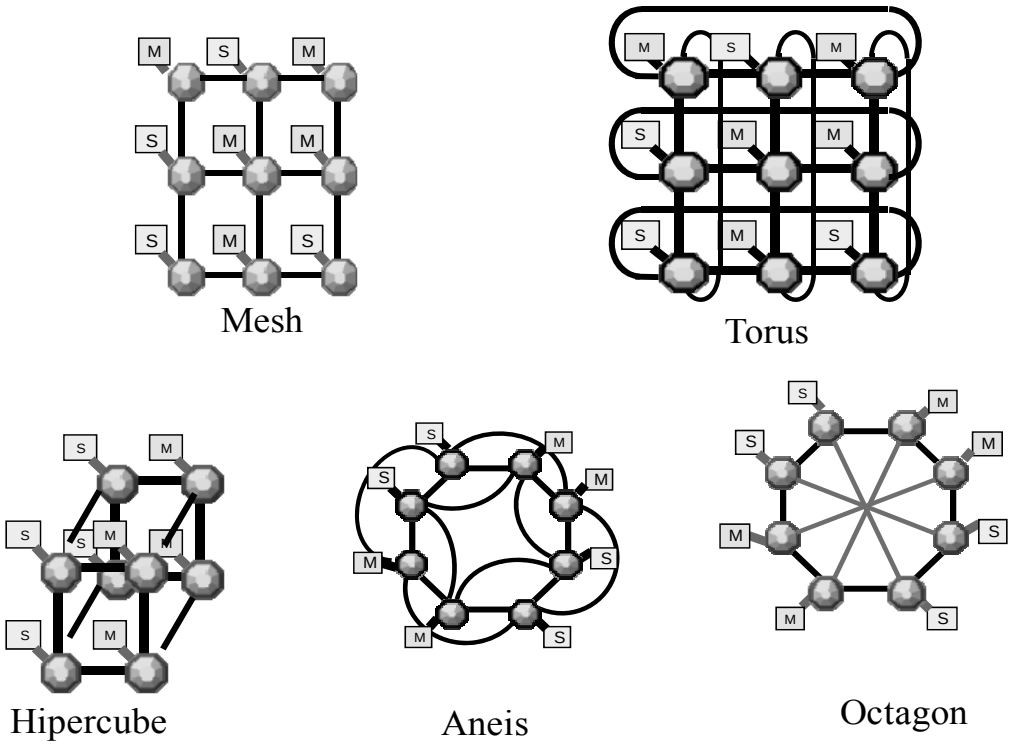

Figura 3.4. Topologias de NoC

O espaço de projeto de uma NoC também é constituído por seus componentes físicos e lógicos. Este espaço de projeto pode variar em função de alguma característica adicional própria para alguma topologia, como quantidade de canais por roteador. Na Tabela 3.2 são apresentados os parâmetros que determinam o espaço de projeto, mostrando a diversidade de valores ou alternativas existentes.

Tabela 3.2. Espaço de projeto utilizando NoC

Componentes	Parâmetro	Valor
Físicos	Topologia	Mesh, Árvore, outras.
	Tipo	Homogênea/Heterogênea
	Roteadores	1.....N
	Portos/roteador	1.....N
	Canais	1.....N
	Interfaces	OCP, customizadas
	Buffers/roteador	Entrada (Tamanho)
		Saída (Tamanho)
	Largura dos canais	N bits
Lógicos	Roteamento/Arbitragem	Prioridade estática/dinâmica, deadline, FCFS, LRS, RR
	Fluxo nos Canais	Unidirecionais
		Bidirecionais

	Qualidade do serviço *Switch technique* *Flow control*	Melhor esforço Serviço garantido *Circuit/packet* *Slack buffer/virtual channel*

3.3. Fluxo de projeto de um *SoC*

Um fluxo para o projeto de um *SoC* deve apresentar as seguintes características [WAG04] :

- Modularidade: Permitir a decomposição do projeto do sistema em tarefas menores. Dois importantes casos são: A separação entre : 1) software e hardware; 2) comunicação e computação.
- Flexibilidade: Poder explorar a maior quantidade possível de alternativas de projeto (*design space exploration*).
- Escalabilidade: Permitir a expansão do projeto em termos de funcionalidade e desempenho.

A modularidade permite reduzir o esforço do projeto através de uma separação do foco de cada tarefa do projeto. Um exemplo é apresentado em [JAN04a], onde são apresentados quatro domínios, ilustrados na figura 3.5, num marco chamado de meta-modelo *rugby* (*rugby meta-model*): temporal; dados; computação e comunicação. Cada domínio apresenta diferentes níveis de abstração.

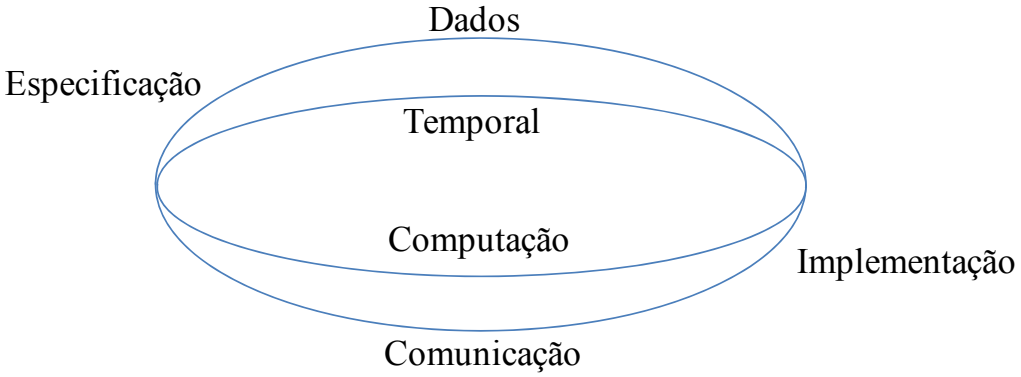

Figura 3.5. Meta-modelo Rugby [JAN04a]

Não existe uma metodologia de projeto de um *SoC* universalmente aceita. De uma forma muito genérica, o projeto de um *SoC* pode ser dividido em três fases (veja figura 3.6) [JEY05]:

1. Especificação do sistema → da lista de requisitos até um modelo executável.
2. Projeto no nível de sistema (do inglês "*System Level Design*") → da especificação executável até o modelo RTL sintetizável (para o hardware) e até o código C (ou outra linguagem de programação) compilável (para o software).
3. Implementação física do *SoC* → dos modelos RTL e C até a solução hardware-software.

3.3.1 Especificação do sistema

A primeira fase do projeto de um *SoC* consiste na definição do conjunto de especificações funcionais e de desempenho. Estas especificações indicam como o sistema vai se comportar e em quais condições vai operar. Por exemplo, ao desenvolver uma placa de rede *Giga-Ethernet* o projetista está especificando o funcionamento (*Ethernet*) e o desempenho a ser atingido que é de 1 *Gigabit/s*. Neste estágio as restrições do projeto também são definidas. Estas restrições podem ser físicas (tamanho do sistema, padrões de conectividade), elétricas (alimentação, níveis dos sinais), potência (consumo de energia) e de custo.

A partir das especificações é gerado um modelo executável[13] que é uma descrição puramente funcional do sistema, sem detalhes da arquitetura nem da implementação final do sistema. Este modelo poderá também ser usado ao longo do projeto para realizar a verificação funcional dos diferentes modelos que serão utilizados. Segundo [GAJ94], o modelo executavel é composto de:

[13] Processo conhecido como captura da especificação

- **Processos**: tarefas que o sistema vai executar e que serão mapeados em elementos de software ou de hardware.
- **Variáveis**: servem para fazer o dimensionamento dos *buffers* dos blocos de hardware e da(s) memória(s) do sistema.
- **Canais**: representam a comunicação entre os diferentes processos do sistema. Esta informação é utilizada no projeto da EC, por exemplo, identificar condições de tráfego tais como: elementos que não se comunicam ou que apresentam uma grande troca de dados.

A segunda fase (projeto no nível de sistema) parte da identificação e da análise destes elementos (processos, variáveis, canais).

3.3.2 Projeto no nível de sistema

O objetivo final do projeto de um *SoC* é um circuito integrado cujos recursos de hardware e de software implementem de forma eficiente a funcionalidade especificada para o sistema e satisfazendo as demais condições/restrições impostas pela especificação. Os resultados obtidos no projeto no nível de sistema são:

1) Um conjunto com os módulos de hardware, (que podem ser ASICs e/ou núcleos de hardware), representados pelos seus modelos no nível RTL (usando núcleos *soft*), ou menos abstratos como no nível de portas ou leiaute (no caso de serem usados núcleos *firm* ou *hard*) com os seus parâmetros de configuração definidos.
2) Um conjunto de módulos de software para serem compilados (controladores, sistema operacional, aplicações) e mapeados no(s) processador(es) do sistema.

O projeto no nível de sistema não apresenta um conjunto universalmente aceito de tarefas a serem utilizadas e seus respectivos métodos. Mas estas tarefas podem ser

agrupadas em quatro categorias [STR07]:

1) Exploração de espaço de projeto
2) Modelagem
3) Verificação funcional
4) Projeto para teste/testabilidade.

3.3.2.1 Exploração do espaço de projeto

Existem muitas soluções Hw-Sw capazes de implementar uma funcionalidade desejada. Entretanto nem todas (mas possivelmente muitas) satisfazem os requisitos (de desempenho e outros) definidos na especificação.

A exploração do espaço de projeto consiste em encontrar uma solução no menor espaço de tempo possível. Esta tarefa apresenta uma grande complexidade devido ao grande número de possíveis soluções. Por exemplo, há muitas alternativas de se particionar a funcionalidade do sistema em hardware e software. Há também muitas opções de microprocessadores de uso geral (assim como microcontroladores e DSPs) capazes de executar a parte software do sistema. Há também muitas alternativas para o projeto da EC e há muitas outras opções.

Dentro da exploração do espaço do projeto quatro grandes tarefas são realizadas:

1) Projeto funcional, composto pela análise da funcionalidade seguida pelo particionamento hardware/software.
2) Seleção dos componentes da arquitetura do sistema[14] (microprocessadores, memórias, periféricos, estrutura de comunicação, etc.)

[14] Tarefa que também denominamos de alocação dos recursos de hardware.

3) Mapeamento das funções do sistema nos elementos de hardware. Deve ser lembrado que esta tarefa é realizada através do máximo reaproveitamento possível de componentes pré-existentes (comerciais ou próprios).
4) Configuração e instanciação dos diferentes elementos do sistema sejam eles módulos IP (desenvolvidos por terceiros) ou próprios (desenvolvidos pelo grupo do projeto)

O projeto funcional parte da análise que é realizada sobre o modelo executável (gerado na fase anterior do projeto). Isto conduz ao particionamento hardware/software do sistema, que consiste em definir e alocar quais processos funcionais serão implementados como software e quais como hardware. A partir do particionamento, o fluxo é separado no projeto do software e do hardware. Neste ponto o projeto da EC pode ser iniciado[15]. Na figura 3.6 é possível observar que o projeto da EC parte após o particionamento, através da definição da arquitetura do sistema.

O projeto no nível de sistema permite realizar o teste do software utilizando modelos abstratos do hardware, obtendo uma redução no tempo de desenvolvimento do projeto. Nas metodologias tradicionais baseadas na síntese RTL, o projeto do software só podia ser testado assim que um primeiro protótipo do hardware estivesse implementado [GHE05].

Após o particionamento hardware/software, a seleção dos componentes do sistema é realizada. Neste ponto acontece o segundo particionamento do sistema: computação/comunicação. Exemplos de seleção de componentes são [WAG04, JEY05]:

- Tipo e número de processadores: de uso geral e/ou especializados.
- Tamanho e hierarquia da memória: interna, externa, coerência.

[15] Como foi apresentado na seção anterior, a listagem dos elementos mestres e escravos é condição necessária para começar o projeto da EC

- Tipo da estrutura de comunicação: barramentos, redes ponto a ponto, NoC.
- Periféricos, portos de entrada/saída.

Assim que os diferentes componentes são definidos, eles devem ser configurados. Isto é feito separadamente para as partes em que projeto do *SoC* foi decomposto: hardware (comunicação, computação) e software. No caso da EC, os diferentes parâmetros de configuração apresentados nas tabelas 3.1. e 3.2 são definidos. Diferentes critérios podem ser utilizados para realizar esta tarefa como baseado em considerações físicas (área, consumo de energia), desempenho, custo.

3.3.3 Implementação física do *SoC*

A implementação física do SoC é feita utilizando as metodologias tradicionais baseadas na síntese RTL, cabe ressaltar que o nível RTL não desaparece, ele continua como ponto de início para a implementação em silício.

Neste estágio do projeto, os elementos de computação e da EC são integrados para realizar sua síntese através das ferramentas de síntese RTL e lógica. Partindo de modelos descritos em VHDL, Verilog ou SystemC-RTL. As ferramentas de síntese realizam a integração de todos os módulos de hardware entre si (*netlist*) assim como a distribuição no chip através das tarefas *floorplanning, e place and route,* assim como, a integração do hardware com o software. Após cada estágio, o sistema é verificado e em caso de identificar erros (*bugs*) eles devem ser corrigidos.

A implementação em silício pode ser em células padrão ou em dispositivos programáveis (caso em que se usa a sigla SoPC, do inglês *system on programmable chip*). Finalmente, o protótipo do sistema é testado e validado.

Na tese não pretendemos realizar todas as tarefas presentes no fluxo da figura 3.6. A idéia é inserir o “nosso_projeto_da_EC” dentro do fluxo apresentado.

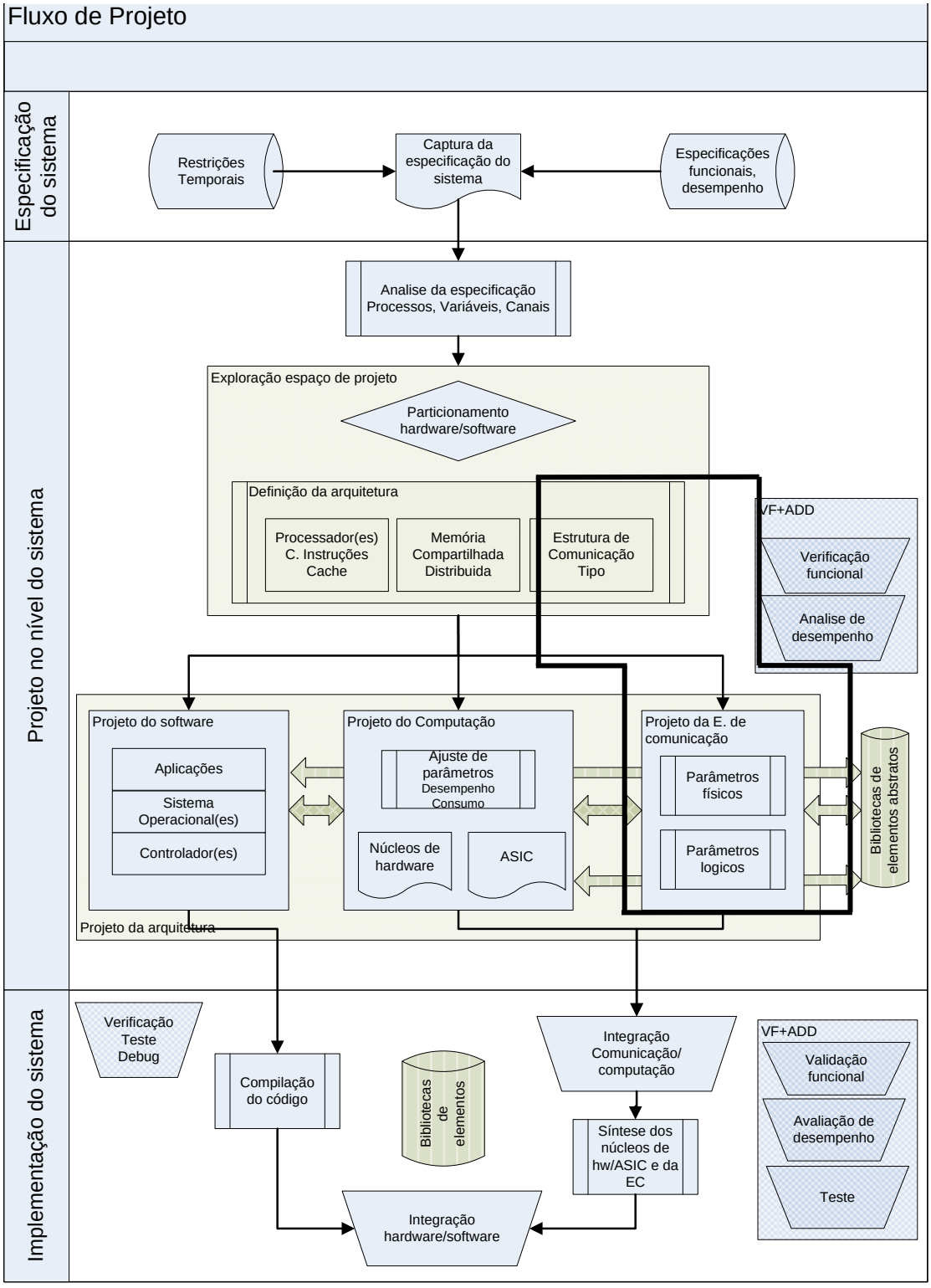

Figura 3.6. “Nosso_fluxo_de_projeto” de *SoC*.

3.4 Sumário

Neste capítulo apresentamos as características dos diferentes fluxos de projeto da EC propostas na literatura, focalizando os requisitos, tarefas e os resultados, desde a escolha do tipo de EC até a implementação dos protocolos a serem utilizados e a configuração das interfaces. Os tipos de EC mais utilizados foram apresentados, barramentos hierárquicos e redes intrachip, mostrando o espaço de projeto existente e os possíveis valores que cada parâmetro de configuração pode apresentar. As tabelas apresentadas permitem verificar o grande espaço de projeto existente. Finalmente, foi apresentado o fluxo de projeto de *SoC*, indicando o lugar que a EC é projetada e como cada tarefa realizada contribui para projetar a EC.

4. Modelagem da comunicação no nível de transações

A modelagem no nível de transações é utilizada em diversas tarefas de projeto, como: análise de desempenho [ESL07], verificação funcional [ROM05], desenvolvimento do software [GHE05]. O estudo sobre a modelagem da comunicação num *SoC* foi dividida em duas partes[16]. A seção 4.1 apresenta as características de comunicação num *SoC* e algumas definições de transação. A seção 4.2 discute diferentes níveis de abstração baseados no conceito de transação (designados por níveis TLM) nos quais é possível modelar a comunicação num *SoC*. Finalmente, a seção 4.3 apresenta o sumário do capítulo.

4.1. Comunicação num sistema sobre silício.

O dicionário Aurélio da língua portuguesa [AUR] define a comunicação como: "*Transmissão de mensagem entre uma fonte e um destinatário, distintos no tempo e/ou no espaço, utilizando um código comum*". No caso da comunicação entre os elementos de um sistema-sobre–silício (*SoC*) é necessário adotar um meio físico[17]. A fonte da comunicação (que geralmente não é única) é o elemento mestre, o destinatário (que também não é único) é o elemento escravo e o "código em comum" é o protocolo da EC.

A comunicação pode ser modelada, entre outras, de forma abstrata através de *tokens* utilizando redes de Petri ou através de transações descritas usando recursos de linguagens de descrição de circuitos. *Tokens* modelam o estado da EC (escrita, leitura, espera, erro) sem entrar nos detalhes da sua implementação e do conteúdo

[16] Nesta tese adotamos SystemC para ilustrar os conceitos apresentados assim como para realizar a parte experimental (ver capítulo 6).

[17] Nesta tese são considerados dois casos, os barramentos hierárquicos e as redes intrachip ambos genericamente designados como estrutura de comunicação (EC)

da comunicação [BLU04]. Transações modelam o estado da EC e o conteúdo da comunicação, sem detalhar todos os sinais envolvidos e suas características (tempos de propagação, tempo de subida, descida) [GRO02].

4.1.1 O que é transação?

O dicionário Aurélio da língua portuguesa [AUR] define uma transação como: "*4. Inform. Em um sistema de informações, operação lógica que não fere a coerência dos dados armazenados; 3. Operação em que há troca ou transferência de valores*".

Quando aplicado ao estudo a respeito do projeto de um *SoC* podemos considerar que "transação é uma forma abstrata[18] de se representar qualquer troca de informação entre 2 elementos presentes no circuito". Num *SoC* uma transação pode representar, por exemplo, o envio de uma mensagem ou uma solicitação para armazenar dados num *buffer* (ou ler de um *buffer*) ou numa memória (*load* ou *store*). Uma transação pode ser descrita mais ou menos detalhadamente de acordo com os atributos e/ou detalhes que se deseja. Esta diversidade de possibilidades de se representar uma transação permite distinguir vários níveis de abstração. Por exemplo, na descrição do barramento [AMB03] uma transação representa uma ou várias transferências (figura 4.1) (variam de acordo com o modelo de comunicação adotado). Cada transferência é composta por várias fases que realizam uma determinada tarefa. No caso de barramentos, estas podem ser: solicitação de acesso (*request)*, envio ao destino (*address)*, execução (*data)*. Cada uma das fases agrupa um conjunto de sinais que são utilizados para realizar a transação. Desta decomposição da transação é possível identificar diferentes níveis de abstrações que podem ser utilizadas na representação da mesma, por exemplo, na figura 4.1 o nível de abstração mais detalhado inclui os sinais, e o

[18] O termo "abstrato" é usado para contrastar com o que seria a representação "detalhada" das mesmas trocas a partir de todos os sinais envolvidos.

menos só as transferências.

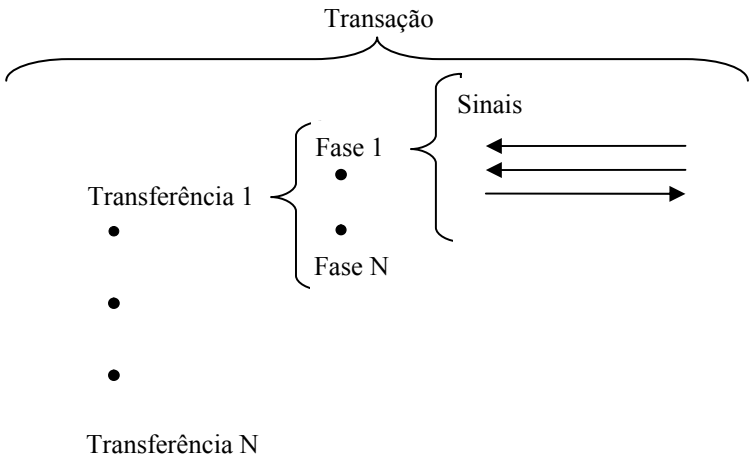

Figura 4.1. Abstração da comunicação.

Na literatura científica sobre o projeto de *SoC* aparecem diferentes definições de transação. Pasricha [PAS02] define uma transação como "*o intercâmbio de dados ou eventos entre dois elementos, exemplo de eventos são as interrupções*". Esta definição abrange o conceito de comunicação (apresentado na seção anterior).

Em [GHE05] uma transação é "*definida como um elemento unificado que representa um conjunto de dados trocados, estes incluem uma lista de parâmetros caracterizados pelo seu nome e valor, chamados de atributos da transação*". Esta definição é mais detalhada do que a de Pasricha e indica que cada transação apresenta um conjunto de atributos. Estes atributos podem ser informações relativas ao tráfego/controle da comunicação. Toda transação tem um instante inicial ("*start time"*) e um instante final ("*end time"*), isto é, a duração é finita.

Para o autor desta tese, uma transação é: "uma estrutura que modela os diferentes eventos que acontecem durante a troca de informações entre dois componentes (através da EC), podendo apresentar diferentes níveis de detalhe (em função do nível de abstração utilizado). Toda transação terá um evento que ocasiona sua ativação, um conteúdo e uma notificação de sua finalização. O conteúdo apresenta

um conjunto de parâmetros/atributos necessários para a realização da transação". Esta definição completa as definições anteriores, porque apresenta o conceito de propriedades da transação e acrescenta a existência de diferentes níveis TLM.

4.1.2 Exemplos de transações e mensagens.

No modelo da comunicação utilizado em [ESL03], cada transação representa uma transferência de leitura seguida de uma de escrita. Este modelo se aplica para as operações realizadas por um processador que lê os dados da memória (*"load"*), faz alguma tarefa e armazena o resultado de volta na memória (*"store"*). No entanto, existem situações em que cada transação representa uma só transferência ou as transferências podem ser invertidas, por exemplo, um controlador de *Ethernet* escreve (através de um DMA) na memória os dados que chegam e depois lê os dados que vão ser encaminhados. Outros autores apresentam que uma transferência é representada por uma transação [AMB03].

As diferentes atividades e eventos que acontecem através de barramentos e de redes intrachip podem ser modelados usando transações [PAS02, JAN04]. Cada tipo de atividade e evento engloba todos os sinais de controle, endereços e dados.

Na comunicação entre o processador e o controlador *Bluetooth* da Plataforma FENIX [FENI] identificamos 16 transações entre estes dois elementos. A seguir são apresentadas três transações que ocorrem durante o estabelecimento de um enlace com outro dispositivo *Bluetooth* (figura 4.2).

A primeira transação é o pedido de estabelecimento da conexão (chamada de *"create_connection()"*). Esta transação acontece quando o processador realiza uma escrita num campo (*flag*) do buffer de entrada do controlador. A segunda transação corresponde à confirmação de recebimento do pedido (*"acknowledge"*). Após estabelecer um enlace, a terceira transação corresponde à informação (enviada pelo controlador ao processador) de que a conexão foi estabelecida.

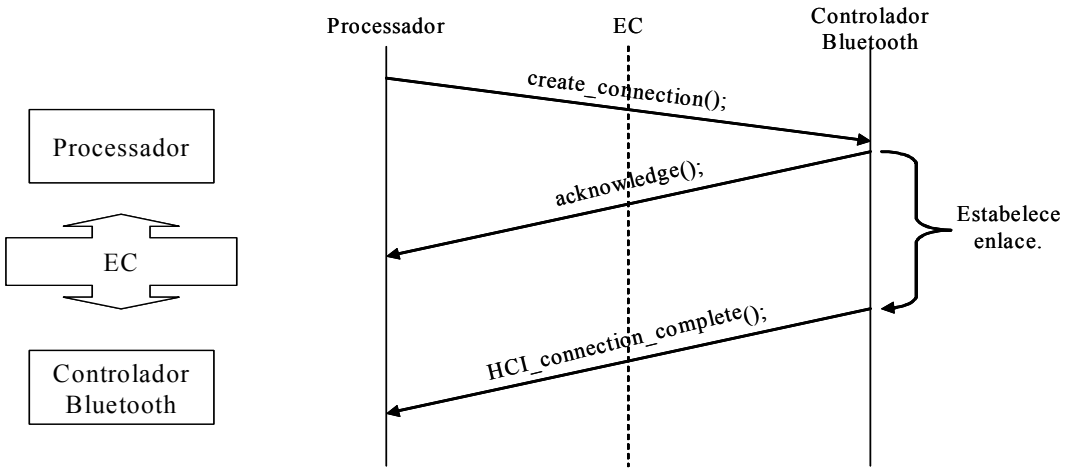

Figura 4.2. Exemplos de transação entre um processador e um controlador Bluetooth.

Cada uma destas três transações pode ser representada de diversas formas (tanto funcional como arquitetural).

1. Todos os sinais e características funcionais e arquiteturais de cada módulo (processador e controlador *bluetooth*) são modelados, tanto as internas (computação) como as externas (comunicação). O modelo é conhecido como caixa branca (*white box*), estando presentes características do modelo RTL.
2. As características internas e externas são modeladas através de funções, incluindo alguns sinais de controle, dados e endereço, presentes dentro de cada função. O modelo é conhecido por caixa preta (*black box*).
3. Só funcionalmente. Uma troca de dados ou eventos entre funções, sem detalhes da arquitetura (mensagens) nem questões temporais de cada atividade.

4.2. Níveis de abstração TLM

Diferentes trabalhos têm apontado a existência de diferentes níveis TLM [HAV02, CAI03, CAN03, ATI07], estes modelam os três aspectos do projeto de

SoC: o hardware (tanto a comunicação e a computação) e o software.

Haverinen et al [HAV02] propuseram três níveis TLM (figura 4.3).

- Mensagens (*messages*)
- Transações (*transaction*)
- Transferências funcionais (*transfers*)

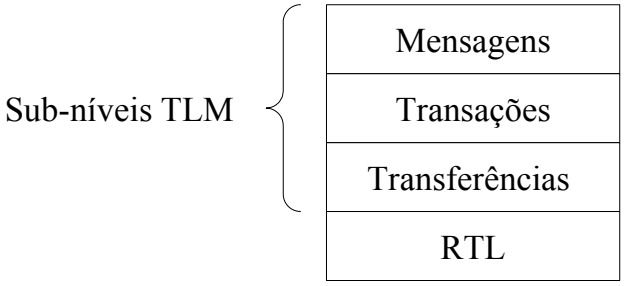

Figura 4.3. Níveis TLM para a comunicação propostos por [HAV02].

Estes três níveis foram utilizados para modelar só os aspectos da comunicação *on-chip*. O nível de mensagens é atemporal. As transações são representadas como tipos de dados abstratos (tipos de dados de C++).

No nível de transações os detalhes do protocolo da EC não são modelados. A comunicação é feita através de transferências de leitura-escrita que podem ser de modo simples (*non-blocking*) ou em rajada (*blocking*).

No nível de transferências a precisão é de ciclos de relógio, a comunicação inclui os detalhes do protocolo da EC (as fases de cada transferência)

Cai e Gajski [CAI03] propuseram três níveis TLM que podem ser aplicados tanto para a computação como para a comunicação. Estes são baseados em três diferentes níveis de abstração: atemporal, tempo estimado e baseado em ciclos de relógio. Isto permite obter diferentes combinações. Quatro delas são ressaltadas (figura 4.4).

- Modelo do conjunto de componentes (*component-assembly model*) (B)
- Modelo do barramento com arbitragem (*bus-arbitration model*) (C)
- Modelo funcional do barramento (*bus-functional model*) (D)
- Modelo da computação com precisão de ciclos (*cycle-accurate computation model*) (E)

O nível representado pelo modelo do conjunto de componentes (ponto B da figura 4.4) modela a comunicação como um conjunto de canais atemporais ponto a ponto. A computação do sistema é modelada no nível TLM de tempo estimado.

O nível representado pelo modelo do barramento com arbitragem (ponto C da figura 4.4) modela a comunicação com canais abstratos. Os eventos através do canal são representados como transferências em modo simples (*non-blocking*) ou em rajada (*blocking*). A computação do sistema é modelada no nível TLM de tempo estimado.

O nível representado pelo modelo funcional do barramento (ponto D da figura 4.4) modela a comunicação como canais hierárquicos que descrevem o protocolo da EC. Os sinais do protocolo são representados dentro do canal como variáveis e a funcionalidade do protocolo como processos. Este modelo tem precisão em ciclos de relógio. A computação do sistema é modelada no nível TLM de tempo estimado.

O nível representado pelo modelo de computação com precisão de ciclos (ponto E da figura 4.4) modela a comunicação no nível de barramento com arbitragem. A computação do sistema é modelada no nível de precisão de ciclos de relógio.

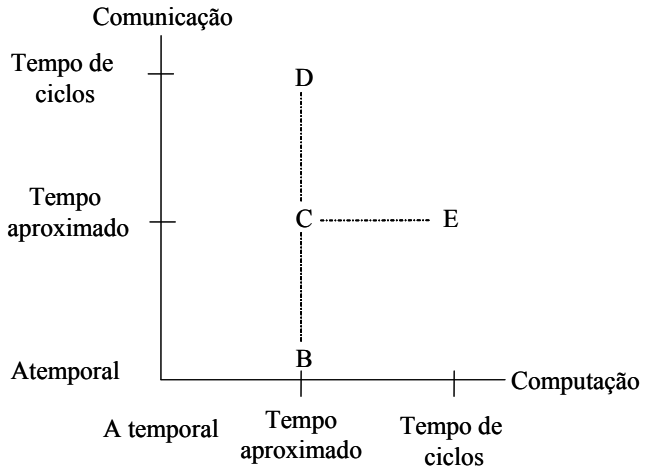

Figura 4.4. Níveis TLM dos elementos da comunicação e da computação de [CAI03].

Connell [CON03] apresentou também três níveis TLM diferentes. Estes níveis podem ser aplicados para o projeto do hardware e do software do *SoC*. (Figura 4.5).

1. Vista do programador (*programmer´s view*)
2. Vista do programador com noções de tempo (*programmer´s view with timing*)
3. Função de ciclos (*cycle callable*)

O nível representado pelo modelo de vista de programador modela a comunicação entre os diferentes processos (seja hardware ou software) através de canais atemporais ponto a ponto.

O nível representado pelo modelo vista de programador com noções de tempo é uma extensão do modelo anterior. Este modelo acrescenta informações temporais ao modelo.

O nível representado pelo modelo de função de ciclos permite trasladar qualquer transferência para o nível RTL. Neste nível a comunicação inclui a descrição do protocolo da *EC*.

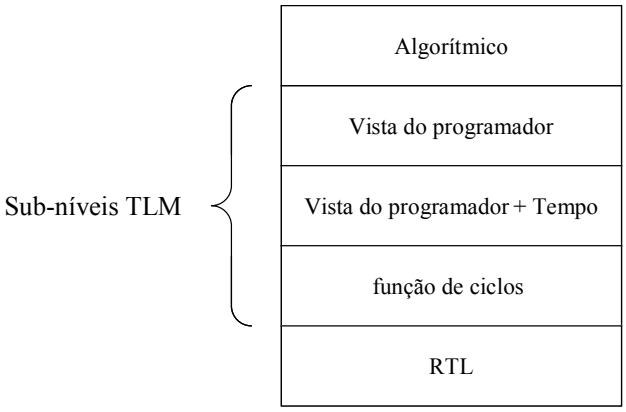

Figura 4.5. Níveis TLM do projeto do hardware e do software propostos por [CON03].

Em [ATI07] dois níveis TLM são apresentados: PVT-TA (*PVT transaction accurate*) vista do programador com precisão de transações e o PVT-EA (*PVT event accurate*) vista do programador com precisão de eventos. Estes dois níveis são uma extensão do nível vista do programador com tempo (PVT) apresentado em [CON03]. Podem ser aplicados tanto para a computação como para a comunicação. A figura 4.6 apresenta os níveis propostos.

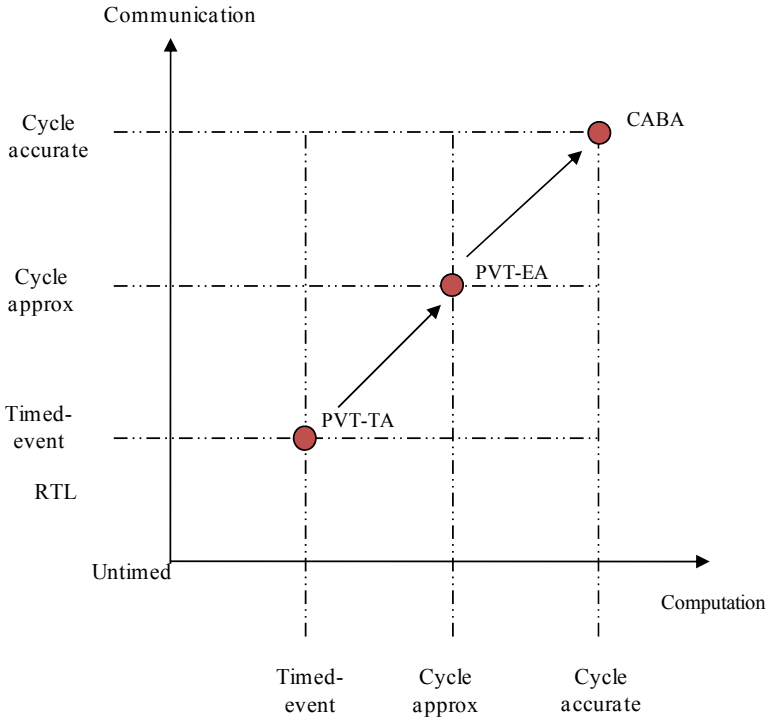

Figura 4.6. Níveis TLM propostos por [ATI07]

O nível PVT-TA foi proposto para realizar uma verificação rápida do sistema e monitorar a contenção na estrutura de comunicação. Neste nível, detalhes sobre os recursos da computação são omitidos (cachê e unidade de controle), assim como detalhes da comunicação (protocolo).

O nível PVT-EA foi proposto para explorar um espectro maior de soluções e aumentar a precisão do nível TLM. Para isto um conjunto de detalhes é acrescentado ao modelo (detalhes que estão ocultos no nível anterior), como o protocolo da EC. Estes detalhes acrescentam informações de atraso de modo a ter uma maior precisão dos modelos, estas informações podem ser agregadas usando declarações de espera (*wait();*) ou através de anotações temporais.

A tabela 4.1 apresenta um resumo das quatro propostas anteriores.

Tabela 4.1. Resumo propostas de níveis TLM.

Ref.	Níveis TLM		
	Atemporal	Tempo estimado	Tempo de ciclos
Comunicação [HAV02]	Mensagem	Transação	Transferência
Comunicação, computação [CAI03]	Ensamble?? de componentes	Barramento com arbitragem	Funcional do barramento
Hardware e software [CoN03]	Vista do programador	Vista do programador com tempo	Função de ciclos
Comunicação, computação [ATI07]	-	Vista de programador com precisão de transferências. PVT-TA	Vista de programador com precisão de eventos. PVT-ET

Nas próximas seções são apresentados os três níveis TLM da forma como são usados nesta tese, descrevendo: sua utilidade, suas características funcionais e temporal, suas representações da computação e da comunicação, a forma de acessar as variáveis e o sincronismo entre eventos.

4.2.1 TLM atemporal funcional ou sincronização por eventos

O nível mais abstrato é o atemporal funcional ou de sincronização por eventos. Neste nível os modelos não apresentam anotações de tempo, só descrevem as características funcionais da aplicação modelada, através de um conjunto de processos ou funções[19] que interagem mutuamente. O sincronismo entre eles é feito através de eventos ou de chamados entre as funções. Isto é, a sincronização é determinística. A comunicação entre os processos é feita através de um conjunto de interfaces bem definidas. Portanto, existe a separação computação-comunicação neste nível.

Os processos representam a computação, as interfaces utilizadas podem ser customizadas ou podem reutilizar interfaces do padrão TLM. As memórias não são explicitamente modeladas, pois as variáveis utilizadas (acessadas) pelos processos estão presentes como elementos públicos ou privados[20]. As características da computação neste nível correspondem ao sistema antes do particionamento hardware-software, mas depois da especificação do sistema[21] (veja figura 3.6).

A comunicação neste nível é representada através de um conjunto de canais "perfeitos" que ligam os diferentes processos do sistema. Duas condições podem existir na comunicação neste nível (devido à ortogonalização computação-comunicação). Os processos podem ser definidos: 1) antes ou; 2) depois do particionamento hardware-software. Para cada situação, diferentes interfaces são necessárias devido às características dos modelos.

---

[19] Estes processos são definidos ou identificados no projeto no nível de sistema, antes do particionamento hardware-software.

[20] Elementos públicos ou privados dentro de uma classe, semelhante ao utilizado em programação orientada a objetos.

[21] Na especificação do sistema, o modelo executável descreve o sistema como uma grande função.

No primeiro caso (antes da partição) os modelos representam processos que compõem o sistema, sem nenhuma representação da arquitetura do mesmo. Neste caso a comunicação é a mera troca de dados entre as funções, acessando explicitamente as variáveis.

No segundo caso (após a partição), a arquitetura do sistema começa a ser definida. A comunicação é realizada entre os elementos que compõem a arquitetura da aplicação, portanto as interfaces são diferentes. Estas utilizam um conjunto de operações de escrita e leitura (com diversas complexidades). O acesso às variáveis é feito através da sua posição na memória.

A abstração dos dados a serem transferidos depende da granularidade da computação, independente da abstração da comunicação, portanto, para cada um dos casos anteriores a granularidade da comunicação é diferente. Aqui podemos observar que é possível simular a computação e a comunicação simultaneamente em níveis TLM diferentes. Isto é possível, porque as funções presentes numa interface podem ser implementadas em diferentes formas, facilitando a modelagem multi-TLM.

A estrutura de comunicação ainda não está determinada neste nível, mas a partir de diferentes análises é possível definir o tipo da mesma e alguns dos seus parâmetros. Esta é representada através de um conjunto de canais perfeitos que interligam todos os elementos presentes, sem apresentar nenhuma latência de comunicação nem contenção.

A figura 4.7 apresenta o modelo da comunicação para este nível, composto por duas operações, uma leitura e uma escrita. Pode-se observar que os pontos de início e fim da operação são os mesmos, mas as latências presentes correspondem aos elementos mestres (processamento dos dados) e escravos (armazenamento dos dados).

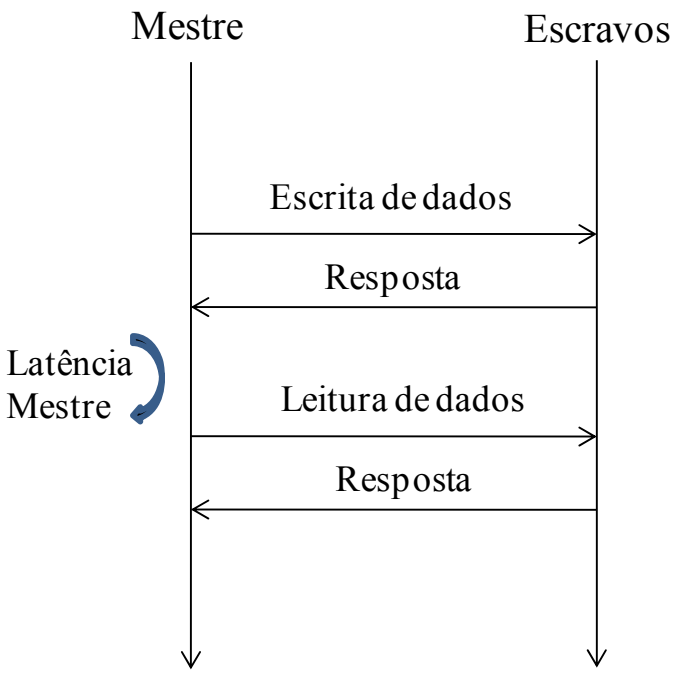

Figura 4.7. Modelo da comunicação utilizado no nível TM atemporal

4.2.2 TLM de tempo estimado ou precisão de transferências.

O segundo nível identificado é o TLM com precisão de transferências[22]. Para evoluir do nível TLM atemporal para este, uma série de tarefas devem ser realizadas no fluxo do projeto:

- Definição da arquitetura do *SoC* (elementos de hardware mestres e escravos)
- Definição do tipo de EC (H-bus ou NoC)

Adicionar características temporais a um modelo permite representar o atraso temporal entre a ativação de um processo e a terminação/suspensão do mesmo. O atraso temporal num modelo da computação é o tempo que transcorre para executar alguma tarefa/instrução, enquanto que o atraso num modelo da comunicação representa o tempo que transcorre para acessar (ler) ou transmitir

[22] O nome utilizado é similar ao da referência [ATI07].

(escrever) dado(s). Todos os elementos modelados neste nível apresentam as seguintes características:

- Precisão funcional: A funcionalidade e a seqüência de execução é semelhante à sua implementação em silício.
- Eventos com durações semelhantes: Todas as transferências têm a mesma duração, todas as instruções têm o mesmo tempo de execução. Os tempos de execução em qualquer unidade funcional são semelhantes.
- Execuções simples: Unidades de *pipeline* não são modeladas, protocolos são simplificados.

Sob o aspecto arquitetural, a computação é representada pelo conjunto de elementos de hardware. As suas características internas são abstraídas de acordo com a listagem anterior: por exemplo, o *pipeline* não é modelado, mas o conjunto de instruções é modelado com uma duração de execução semelhante. As variáveis são acessadas pelo seu endereço na(s) memória(s).

A comunicação neste nível apresenta o tipo de estrutura de comunicação definido, por tanto a arquitetura da aplicação está mais estruturada. Neste nível, a EC é modelada simplificando os protocolos da comunicação. No caso de H-bus, os diferentes elementos de hardware são mapeados ao(s) barramento(s), pelo fato de serem compartilhados existe a contenção, portanto a política de arbitragem deve ser configurada. O tempo de execução de um acesso à memória (escrita ou leitura) sem contenção é semelhante (1 ciclo). O uso do sinal de relógio permite estabelecer o sincronismo necessário no caso de modelos que possuem estados de espera internos ("*wait();"*).

Neste nível é utilizado o modelo da comunicação apresentado na figura 4.8 que é uma generalização do modelo da figura 4.7. Uma transferência começa quando o elemento mestre pede o acesso ao barramento (M1, M3). A transferência termina no instante M2, M4 quando o escravo atende ao pedido do mestre. Entre cada

transferência existe a latência de processamento que representa o tempo das operações feitas com os dados transferidos. No caso de um escravo apresentar uma latência interna, o barramento fica bloqueado esperando a resposta do escravo.

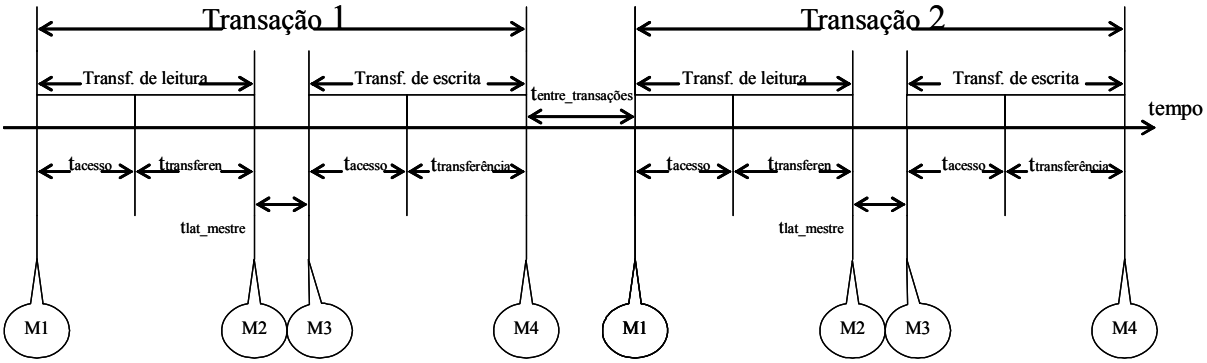

Figura 4.8. Modelo da comunicação do nível de tempo estimado.

Para realizar a sincronização dos eventos, o conceito de ponto de avaliação do barramento, PAB (do inglês, *bus evaluation point*) está presente. Este ponto corresponde ao momento em que as funções realizadas pela EC são executadas. O PAB acontece na borda de descida do relógio. Os eventos dos elementos mestres[23] são ativados na borda de subida do sinal de relógio. Assim é possível garantir que a EC consiga detectar todos os pedidos de acesso. Esta característica é utilizada só para considerações da modelagem e não para a implementação física (figura 4.9).

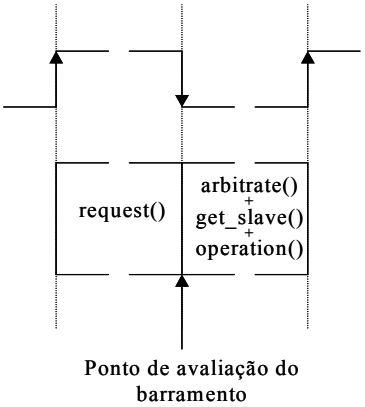

Figura 4.9. Sincronismo na modelagem da comunicação no nível de tempo estimado.

[23] Representados como *Threads* em SystemC, ou *Process* em VHDL.

4.2.3 TLM com precisão de ciclos.

Evoluir do nível de tempo estimado ao de precisão de ciclos permite refinar os elementos do sistema, através da modelagem e definição de todos os seus parâmetros de configuração das características arquiteturais. Neste nível a precisão do comportamento é de ciclos de relógio, semelhante ao nível RTL[24]. Como no nível anterior, os elementos da computação e da comunicação já estão definidos. Todos os elementos modelados apresentam as seguintes caracteristicas:

- Precisão funcional.
- Eventos com durações explícitas: Todas as atividades instruções/transferências são modeladas incluindo suas durações.
- Execuções detalhadas: Unidades de *pipeline* são modeladas, protocolos estão presentes.

Do ponto de vista arquitetural, a computação é modelada incorporando todas as características e detalhes da arquitetura, por exemplo, os caminhos de execução são totalmente modelados incluindo se é segmentada (*pipeline*). A duração de cada instrução é modelada de acordo com a organização e arquitetura do processador. As memórias são detalhadas incluindo todos os protocolos de acesso.

A EC é modelada incluindo explicitamente todos os detalhes dos protocolos necessários para realizar a comunicação. Como no nível anterior, o tipo de EC já está definido assim como os seus parâmetros físicos; agora os atributos lógicos (relativos aos protocolos) são acrescentados.

O modelo da comunicação neste nível é apresentado na figura 4.10. Este pode ser usado para representar as transferências através de barramentos como o *AMBA*.

[24] Precisão de ciclos de relógio significa que os eventos ciclo a ciclo no modelo são semelhantes aos obtidos nos modelos RTL.

Neste nível cada transferência é composta por fases: fase de acesso (*request*), fase de endereço (*address*), fase de dados (*data*). A transferência inicia no momento em que o elemento mestre pede o uso de barramento (*request*). Este pedido é recebido pelo barramento que vai enviar a solicitação ao árbitro (M1). A fase de pedido culmina quando o elemento mestre recebe a autorização para usar o barramento (*grant)*. A duração desta fase pode ser de vários ciclos. Após a autorização a fase de endereço começa, os elementos escravos são preparados para poder completar a transferência. A duração desta fase é de um ciclo de relógio, embora situações como tempo de espera ou latências dos escravos faça com que a duração seja maior. Finalmente durante a fase de dados, a leitura (ou escrita) no elemento escravo acontece (M2). Este caso acontece para transferências simples, no caso de transferências em modo rajada, as três fases podem ser concorrentes (realizando diferentes transações), isto é, fazer transferências com *pipeline* (figura 4.12).

O sincronismo nesat modelagem é semelhante a do nível de tempo estimado, utilizando o PAB. Nas bordas de subida as tarefas dos elementos mestres são executadas, nas bordas de descida as tarefas da EC são executadas (figura 4.11).

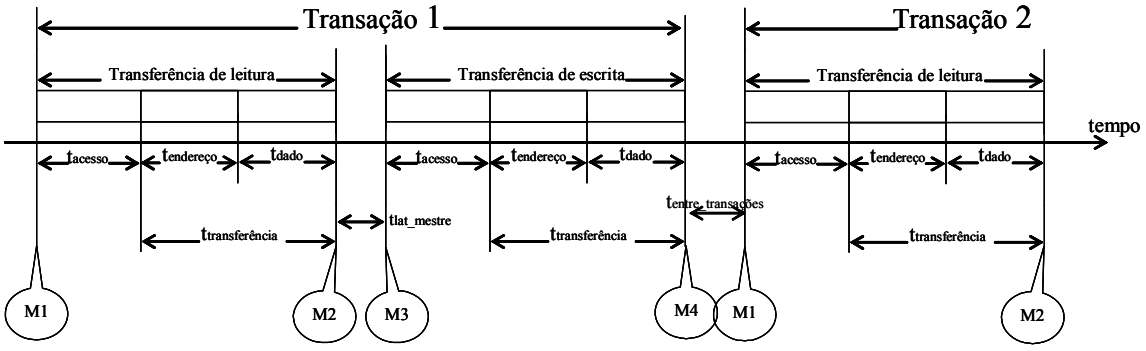

Figura 4.10. Modelo da comunicação do nível de precisão de ciclos

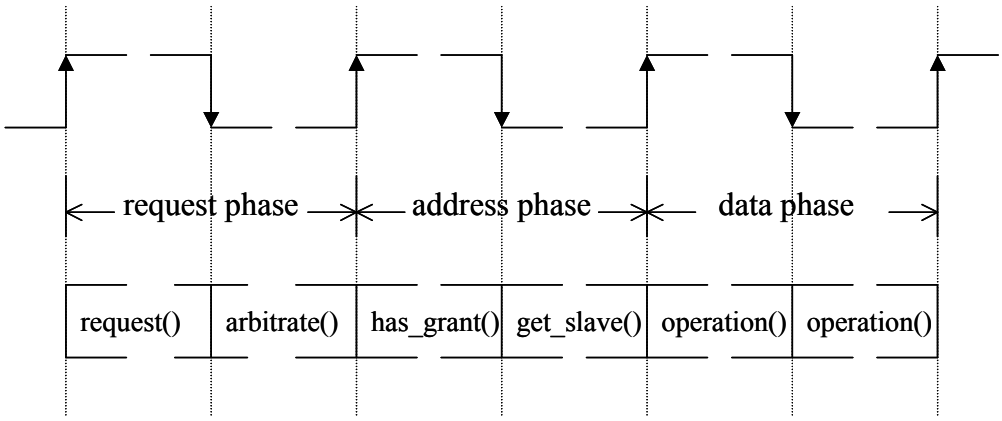

Figura 4.11. Sincronismo da modelagem da comunicação do nível de precisão de ciclos

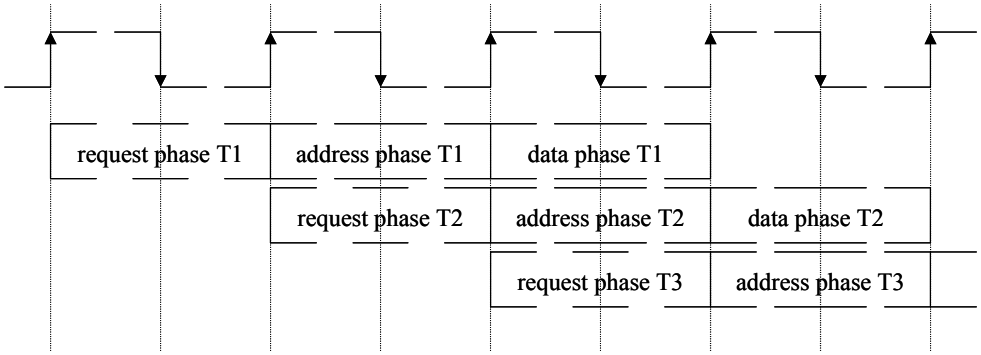

Figura 4.12. *Pipeline* da comunicação no nível de precisão de ciclos.

4.3 Sumário

Neste capítulo apresentamos e comparamos diferentes propostas sobre níveis de abstração TLM. Os trabalhos foram analisados e três níveis foram identificados como úteis para o projeto da estrutura de comunicação de um *SoC*. O nível mais abstrato é o TLM atemporal-funcional com sincronização por eventos. O segundo nível é o TLM de tempo estimado com precisão de transferências. O nível menos abstrato é o TLM com precisão de ciclos de relógio.

No capítulo 5 mostraremos de que forma associamos métricas de desempenho a cada um destes níveis e quais parâmetros de configuração de uma EC podem ser definidos em cada nível. Desta forma o projeto da EC evolui de forma mais gradual (em comparação ao uso de somente TLM com precisão de ciclos) levando a uma otimização do projeto do *SoC*.

5. Metodologia de projeto da EC - MaLOC

Neste capítulo é apresentada a metodologia MaLOC para o projeto da EC. A seção 5.1 apresenta as métricas de desempenho e sua associação a cada nível TLM. A seção 5.2 apresenta as dependências (implicando em precedências ao longo do projeto de uma EC) entre os parâmetros de configuração de barramentos hierárquicos e de redes intrachip. A seção 5.3 apresenta a relação comunicação-computação através dos níveis TLM utilizados. A seção 5.4 apresenta a metodologia proposta baseada nas características apresentadas nas seções anteriores. Finalmente, a seção 5.5 apresenta o sumário do capítulo.

5.1 Análise de desempenho

Uma análise de desempenho é genericamente constituída por duas fases. A primeira consiste em determinar os valores das métricas de desempenho que se deseja estimar (quantificação). A segunda consiste em realizar uma avaliação dos resultados obtidos, em outras palavras qualificar o desempenho previsto de acordo com critérios aceitáveis para a fase do projeto em que se encontra[25].

Existem muitas métricas de desempenho de uma EC encontradas na literatura. Entretanto, elas têm sido calculadas com base em valores obtidos a partir de simulações usando modelos de baixo nível (RTL) ou a partir de modelos analíticos. A seguir vamos explicar como associamos tais métricas aos 3 níveis TLM adotados na metodologia MaLOC.

[25] Os termos "estimar" e "avaliar" são usados em contraste com "determinar", pois o real desempenho somente pode ser conhecido quando o circuito estiver pronto para uso.

5.1.1 Métricas de desempenho

Em [KAN92] são apresentadas métricas de desempenho utilizadas para sistemas computacionais. Estas foram agrupadas em diferentes categorias, classificadas de acordo com sua finalidade: reação, utilização, durabilidade, confiabilidade e produtividade. Em [ESL04] estas foram adotadas para a análise de barramentos. Em [SEP06] foi apresentado um conjunto de métricas de desempenho para redes intrachip. Nesta tese, os dois conjuntos de métricas foram analisados para associá-los aos diferentes níveis TLM.

5.1.1.1 Métricas no nível TLM atemporal.

As métricas no nível atemporal TLM identificam as características e a distribuição do tráfego gerado. Lembrando que, neste nível, a EC é representada através de canais ponto a ponto. Três métricas foram identificadas.

1) Geração de tráfego: Quantidade de tráfego gerado pelos elementos mestres do sistema. Os diferentes valores de *DGi* representam a quantidade de dados gerado por cada mestre, estes podem ser obtidos a partir de diferentes simulações ou da especificação do sistema.

$$TG = \sum_{i=o}^{n-1} DG_i \cdots [Bytes] \qquad \text{Equação 5.1.}$$
$$DG_i = Dados\_gerados\_nó\_i$$

2) Localidade da comunicação atemporal: Relação entre o tráfego que é transmitido através dos diferentes canais ligados a um mestre em relação ao tráfego por ele gerado (Equação 5.2). Os valores de *D_Cij* são obtidos analisando o destino dos dados gerados pelos elementos mestres do sistema.

$$LCA_{ij} = \frac{D\_C_{ij}}{DG_i} \cdots [\%]$$
$$D\_C_{ij} = Dados\_pelo\_Canal\_de\_i\_até\_j \qquad \text{Equação 5.2.}$$

3) Nível de uso por canal: Relação entre o tráfego que circula por cada canal em relação ao tráfego total gerado por todos os elementos mestres (Equação 5.3).

$$NUc = \frac{D\_C_{ij}}{TG} \cdots [\%] \qquad \text{Equação 5.3.}$$

5.1.1.2 Métricas nos níveis temporais TLM

As métricas M1 até M5 são utilizadas para barramentos [ESL07]. As métricas M6 até M12 são utilizadas para NoC [SEP06]. Estas métricas são utilizadas nos níveis TLM de precisão de ciclos e de tempo estimado.

Métricas para barramentos.

M1) Duração média: Duração de cada uma das operações de leitura e de escrita. Os resultados apresentados incluem a latência média de todas as operações (Equação 5.4). A latência de cada operação é o tempo que transcorre desde que é solicitado o acesso ao barramento até que a operação seja concluída.

$$LM = \frac{\sum_{i=0}^{n} L_{Ti}}{\#\_Op} \cdots \left[\frac{ciclos}{word}\right]$$
$$L_{Ti} = Latencia\_de\_cada\_operação \qquad \text{Equação 5.4.}$$
$$\#\_Op = Numero\_de\_operações$$

M2) Vazão *(throughput)* de dados: Quantidade total de dados que trafegam pelo(s) barramento(s) durante o tempo de operação (Equação 5.5). O valor de *DGi* é obtido através da simulação do sistema.

$$TD = \frac{\sum_{i=0}^{n} DG_i}{TSim} \cdots \left[\frac{bytes}{ciclo} \right] \qquad \text{Equação 5.5.}$$
$$TSim = Tempo\_simulação$$

M3) Nível de utilização global: Relação entre o tempo de utilização do(s) barramento(s) em relação ao tempo de operação do sistema (Equação 5.6).

$$NUG = \frac{T\_Op}{TSim} \cdots [\%] \qquad \text{Equação 5.6}$$
$$T\_Op = Tempo\_todas\_operações$$

M4) Participação por elemento: Participação de cada elemento mestre no tráfego gerado (Equação 5.7).

$$NU_i = \frac{DG_i}{TG} \cdots [\%] \qquad \text{Equação 5.7.}$$

M5) Localidade da comunicação: Relação entre o tráfego que é transmitido através dos diferentes barramentos em relação ao tráfego gerado pelo sistema (Equação 5.8). O valor de *TBij* é obtido da simulação do sistema.

$$LC_{ij} = \frac{TB_{ij}}{DG_i} \cdots [\%] \qquad \text{Equação 5.8}$$
$$TB_{ij} = Trafego\_Barramentos\_i\_j$$

Métricas para NoC.

M6) Duração média das transações: Número médio de ciclos de relógio necessários para transferir um *flit*[26] através da *NoC* (Equação 5.9).

[26] Um *flit (flow control digit)* é a menor unidade de informação que sobre a qual é realizado o controle de fluxo no roteamento dentro da NoC. Cada pacote que trafega através da NoC é composto por vários *flits* [BEN05].

$$Duração\_Média = \frac{\sum(tempo\ transações)}{\sum(Flits\ transmitidos)} ..... \left[\frac{ciclos}{flits}\right] \quad \text{Equação 5.9}$$

M7) Vazão (*throughput*) das transações: Número total de transações realizadas em toda a rede durante a simulação (Equação 5.10).

$$T\_transações = \frac{\sum(tempo\ transações)}{Tsim} ..... \left[\frac{transações}{ciclos}\right] \quad \text{Equação 5.10}$$

M8) Largura de banda: Quantidade total de *flits* transmitidos em toda a rede ao longo da simulação (Equação 5.11).

$$Largura\_banda = \frac{\sum(Flits\_Transferidos)}{Tsim} ..... \left[\frac{flits}{ciclos}\right] \quad \text{Equação 5.11}$$

M9) Tempo de espera médio por roteador: Tempo médio decorrido entre um pacote chegar a um roteador e sua transferência ser realizada (Equação 5.12).

$$Tempo\_espera\_medio = \frac{\sum(Tempo\_espera\_comutação)}{Numero\_transferências} ..... \left[\frac{ciclos}{transferencias}\right]$$

Equação 5.12

M10) Nível de utilização por roteador: Participação, em termos de ocupação, de cada roteador na transmissão da informação durante o tempo da simulação (Equação 5.13).

$$Utilização\_chave = \frac{\sum(Flits\_Trasnferidos\_por\_roteador)}{ciclos\_simulados} ..... \left[\frac{flits}{ciclos}\right]$$

Equação 5.13

M11) Dispersão: Relação entre a porcentagem de pacotes transmitidos que empregaram o caminho mínimo ótimo (menor número de roteadores) e a porcentagem de pacotes transmitidos que empregaram um caminho diferente (Equação 5.14).

$$Dispersão = \frac{(pacotes\_caminho\_minimo)(\%)}{(pacotes\_não\_caminho\_minimo)(\%)} \quad \text{Equação 5.14}$$

5.2 Dependências entre parâmetros de configuração

Três tipos de relações foram analisadas, entre os parâmetros de configuração da EC:

1. Precedência: indica a obrigatoriedade de se conhecer um determinado parâmetro antes de se poder determinar o valor de outro. Por exemplo, antes definir o número de pontes é preciso saber a quantidade e os tipos de barramento de um H-BUS.
2. Concatenação: indica parâmetros a serem definidos um imediatamente em seguida do outro. Por exemplo, a política de arbitragem deve ser definida e imediatamente configurada.
3. Concorrência: indica que dois parâmetros não apresentam nenhuma dependência entre si podendo ser definidos independentemente um do outro.

Vamos a seguir apresentar as características dos fluxos de projeto para barramentos hierárquicos e redes intrachip, levando em conta os três tipos de relações existentes.

5.2.1 Fluxo para EC baseadas em Barramentos

O fluxo do projeto da EC baseada em barramentos parte da definição do seu tamanho. Isto é, definir o número de barramentos. Esta decisão é realizada em função do número de elementos de computação (mestres e escravos), a localidade da comunicação e o nível de utilização dos canais. A quantidade de barramentos não pode ser maior do que o número de elementos.

De forma interdependente com a definição do tamanho da EC, o mapeamento dos elementos mestres e escravos (núcleos IP do *SoC*) é realizado nos diferentes barramentos. Está decisão é tomada em função do número de barramentos, a localidade da comunicação e o nível de utilização dos barramentos. Esta decisão procura mapear os elementos com maior comunicação entre si, no mesmo barramento ou o mais próximo possível para aumentar o *throughput*.

Após ter definido o tamanho da EC e o mapeamento dos elementos, a quantidade de pontes é definida em função do tamanho da EC e da localidade da comunicação. Esta decisão é realizada para definir a estrutura global da EC. No caso de um único barramento, a solução para o mapeamento é trivial, todos ligados ao único barramento, portanto, nenhuma ponte será instanciada.

Assim que a comunicação entre barramentos é resolvida, através das pontes, o próximo parâmetro a ser definido é a política de arbitragem. Esta escolha permite ter a EC escolhida pronta para ser simulada. Esta decisão é tomada a partir da duração média das transferências (leitura e escrita), *throughput* de dados, nível de utilização (global, por barramento), participação na utilização. Cada política de arbitragem (prioridades fixas, *round-robin, TDMA*, dois níveis) apresenta um conjunto próprio de parâmetros de configuração (prioridade mestres, seqüência dos mestres, duração das faixas de acesso).

O tipo de barramento é escolhido em função do *throughput* de dados (alta ou

baixa velocidade), a localidade de comunicação e o mapeamento de elementos. Esta escolha pode ser feita desde o momento que o mapeamento foi realizado.

A largura do barramento é escolhida em função do tipo de barramento, a duração média das operações e o *throughput* de dados. Aumentar a largura dos barramentos permite aumentar o desempenho (*throughput* e diminuir a duração média das transações), mas pode apresentar uma maior área e consumo de energia em função do chaveamento presente no barramento (nível de utilização).

O protocolo é implementado em função do tipo de barramento, definindo tipos especiais de rajadas (*burstiness*) dentro das interfaces de acesso à EC, incorporando *pipeline* dentro dos barramentos (alta velocidade), transferências partidas (*split-transfers*) e suporte para transferências entre elementos de diferentes larguras (baixa velocidade).

A tabela 5.1 apresenta a lista dos parâmetros, as dependências/métricas utilizadas, os critérios para realizar a escolha e seus possíveis valores. A análise é baseada nos parâmetros em comum dos diferentes barramentos disponíveis.

Tabela 5.1. Parâmetros, métricas e critérios para EC baseadas em barramentos

Parâmetro	Dependências/Métricas	Critérios	Valores
Tamanho (# de barramentos)	# Elementos Localidade da comunicação Utilização do canal	Maximizar *throughput* (paralelismo) Área Energia	B1, B2, ... Bn
Mapeamento	# Elementos Localidade da comunicação Utilização do canal	Maximizar *throughput* (paralelismo) Área Energia	Elementos/Barramento
Pontes (#)	# de barramentos Localidade da comunicação	Comunicação entre elementos	P1, P2, ...Pm
Arbitragem (escolha e configuração)	Latência Duração média Nível de utilização *Throughput* dados Participação no uso	Sistema balanceado Diminuir latência Maximizar *throughput*	Prioridades fixas (identificação) round-robin (seqüência) Dois-níveis (seqüência, tempos)

			TDMA (seqüência, tempos)
Largura dos barramentos	Latência Nível de utilização *Throughput* dados Localidade	Diminuir latência Maximizar *throughput* Área Energia	2^n bits
Tipo de barramento	Mapeamento *Throughput* dados Localidade	Maximizar *throughput* (paralelismo) Área Energia	Alta velocidade Baixa velocidade
Protocolo	Latência Duração média Nível de utilização *Throughput* Localidade	Diminuir latência Maximizar *throughput* (paralelismo) Área Energia	Burstiness Pipeline Split-Transfers

Com base na discussão acima, a figura 5.1 apresenta uma árvore de dependências.

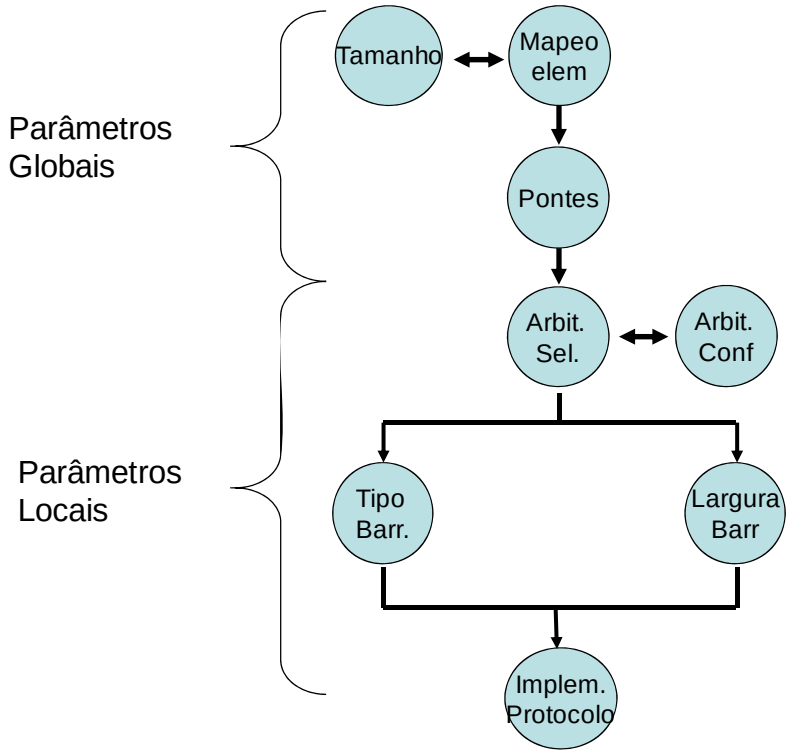

Figura 5.1. Árvore de dependências para EC baseadas em barramentos

5.2.2 Fluxo para EC baseadas em NoC

A primeira decisão a ser tomada é escolher a topologia da NoC. Isto é realizado a

partir do número de elementos (mestres e escravos) e a geração de dados. Existem diferentes topologias propostas que apresentam diferentes características funcionais e de desempenho, assim como de área e consumo de energia. Estas podem ser agrupadas como: 1) Barras cruzadas (*crossbars*), 2) grades (*mesh*); 3) árvores; 4) cubos (*torus*); 5) irregulares (heterogêneas). Baseado na topologia escolhida, pode-se definir o tamanho da NoC medido pela quantidade de roteadores e os canais que os interligam. O maior tamanho possível da NoC é o número de elementos presentes no sistema.

Em seguida é realizado o mapeamento de componentes, partindo da localidade da comunicação e do número de roteadores (e a quantidade de portos). Procura-se minimizar a latência das operações posicionando próximos os elementos que apresentam a maior comunicação.

Assim que a topologia e seu tamanho estão definidos, a técnica de comutação (*switch technique*) deve ser determinada. Neste ponto do projeto é definido o tamanho dos pacotes e dos *flits*. Estas decisões são baseadas na topologia, localidade da comunicação e na latência dos pacotes.

Distintos caminhos podem ser utilizados para que os dados (pacotes) cheguem até seu destino. A decisão de qual caminho percorrer é determinada pelo fluxo de controle. Este é definido em função da latência dos pacotes, os tempos de espera médios por roteador e o caminho mínimo/dispersão por operação.

A técnica de roteamento é definida baseada na duração média das transferências, localidade da comunicação, *througput* de pacotes/dados, tempo de espera por roteador e o caminho mínimo/dispersão dos pacotes. Adicionalmente as tabelas de roteamento são definidas para identificar os destinos dos pacotes.

No meio do caminho, diferentes pacotes podem tentar utilizar o mesmo recurso (buffers, portos de saída), esta contenção pode provocar um congestionamento na

NoC. Para solucionar este problema utiliza-se uma política de arbitragem e suas configurações. A decisão é tomada a partir da latência dos pacotes, *throughput* de dados, tempos de espera no roteador, caminho mínimo/dispersão dos pacotes. Como no caso dos barramentos, cada política de arbitragem (prioridades fixas, round-robin) apresenta um conjunto de parâmetros de configuração (prioridades dos mestres, seqüência dos mestres).

A configuração dos buffers (número e tamanho) é definida em função da técnica de comutação (pelo tamanho do pacote), o fluxo de controle e a técnica de roteamento, assim como pela latência dos pacotes, o *throughput* de dados/pacotes, tempos de espera por roteador. A largura do canal é definida em função da configuração dos pacotes e do tamanho dos buffers.

Finalmente, as políticas de qualidade de serviço como melhor esforço (*best effort*), representado pelo caminho mínimo dos pacotes/dispersão, e *throughput* garantido (*guaranted throughput*) são configuradas.

A tabela 5.2 apresenta um resumo dos parâmetros, as dependências/métricas utilizadas, os critérios para realizar a escolha e seus possíveis valores.

Tabela 5.2. Parâmetros, métricas e critérios para EC baseadas em NoC

Parâmetro	Dependências/Métricas	Critérios	Valores
Topologia	Número de elementos Dados gerados Localidade da comunicação Área Energia	Desempenho Área/Energia	Crossbars Grid (Mesh) n-cubos (Torus) Arvores Irregulares
Tamanho da NoC (Número de roteadores)	Topologia Número de elementos Localidade da comunicação Nível de utilização de canal Área Energia	Maximizar *throughput* (paralelismo) Área/Energia	N= (#Elementos/#pts_rot-1) N= #Elementos IP
Mapeamento de elementos	Topologia Número de portos-roteador Localidade da comunicação Nível de utilização de canal	Maximizar *throughput* (paralelismo) Área/Energia	Elemento(s)/roteador

	Área Energia		
Switch Technique	Latência dos pacotes Localidade da comunicação *Throughput* de dados	Granularidade dos dados (tamanho do pacote) Área/Energia	Circuit-switch Packet-Switch
Configuração pacote	*Switch technique* Topologia	*Switch Technique*	Tamanho do pacote Tamanho dos *flits*
Flow control	Latência dos pacotes Localidade da comunicação *Throughput* pacotes/dados Tempo de espera roteador Caminho mínimo/Dispersão	Maximizar o *throughput* Diminuir tempo de espera no roteador	Single queue Slack Buffer Virtual channel
Routing	Latência dos pacotes Localidade da comunicação *Throughput* pacotes/dados Tempo de espera roteador Nível de utilização Caminho mínimo/Dispersão	Maximizar *throughput* (paralelismo) Diminuir latência Evitar Área/Energia	Tabelas de endereços Estático Dinamico X-Y west-first
Arbitragem (escolha e configuração)	Latência dos pacotes Localidade da comunicação *Throughput* pacotes/dados Tempo de espera roteador Caminho mínimo/Dispersão	Maximizar *throughput* (paralelismo) Área/Energia	Prioridades fixas Round-robin Deadline Dinamico
Instanciação dos buffers (número e tamanho)	Latência dos pacotes Número de portos-roteador *Throughput* pacotes/dados Tempo de espera roteador Nível de utilização Caminho mínimo/Dispersão	Maximizar *throughput* (paralelismo) Diminuir latência Tamanho do pacote Evitar deadlock, *starving* Área/Energia	1, 2, ...n x portos-roteador n-bits/flits/packets
Largura do canal	Latência dos pacotes Localidade da comunicação *Throughput* pacotes/dados Tempo de espera roteador Caminho mínimo/Dispersão	Tamanho do pacote	n bits
QoS	*Throughput* pacotes/dados Tempo de espera roteador Caminho mínimo/Dispersão		*Best Effort* *Guaranted throughput*

A árvore de dependências é mostrada na figura 5.2.

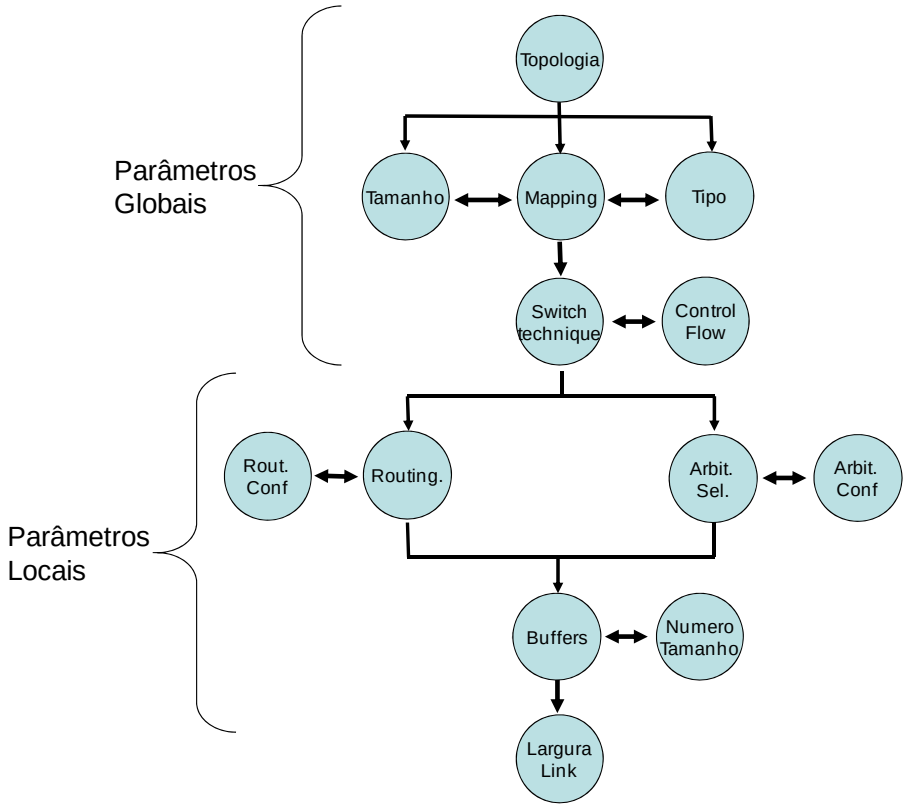

Figura 5.2. Árvore de dependências para EC baseadas em NoC

5.2.1 Diferenças entre os fluxos do projeto de barramentos e de NoC

A primeira diferença entre os dois fluxos é na topologia. No caso das NoC esta decisão acontece no primeiro estágio do fluxo, no caso de barramentos a topologia é formada pela definição dos três primeiros parâmetros: o tamanho; o mapeamento de elementos e o número de pontes.

Outra diferença entre os dois fluxos é a configuração do pacote de dados. No caso de barramentos isto não acontece de forma explícita porque os endereços para acessar os elementos escravos, a configuração da política de arbitragem e a implementação do protocolo estabelecem as condições para realizar a comunicação. No caso da NoC estas informações estão presentes nos diferentes campos que compõem o pacote a ser transmitido.

No caso da NoC, vários parâmetros configuram a comunicação da fonte até o destino (a técnica de comutação, o fluxo de controle e a técnica de roteamento). No caso de barramentos, isto é mais simples, os parâmetros que configuram a comunicação são o número de pontes, a escolha e configuração da política de arbitragem e o protocolo.

As pontes têm uma função semelhante à técnica de comutação, porque estabelecem a comunicação entre diferentes barramentos, mantendo a informação da fonte e do destino.

As políticas de qualidade de serviço não estão presentes no projeto de barramentos hierárquicos, mas, elas poderiam fazer parte da implementação do protocolo. O CoreConnect e o AMBA não possuem nenhuma política de qualidade.

5.3 Refinamento da computação-comunicação através dos níveis TLM

Baseado na ortogonalização comunicação-computação e na existência de três níveis TLM (seção 4.2), existe um conjunto de pontos que identificam diferentes possibilidades de análises na exploração do espaço de projeto. A figura 5.3 apresenta as diferentes possibilidades. A área cinza corresponde ao projeto realizado com as ferramentas de síntese RTL.

Esta análise apresenta a evolução (refinamento) do projeto da EC ao percorrer os diferentes caminhos no espaço de abstrações TLM. Cada ponto da figura representa o *SoC* com diferentes características, modelado em diferentes níveis TLM. Cada transição[27] entre pontos indica um conjunto de tarefas que são

[27] As evoluções apresentadas foram consideradas refinando só um dos eixos.

realizadas para poder atingir as condições representativas do ponto destino. A análise a seguir foi realizada para barramentos hierárquicos.

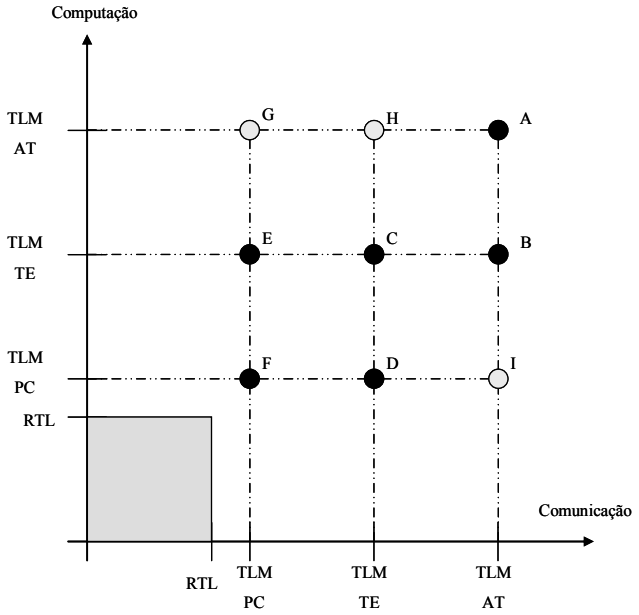

Figura 5.3. Espaço de abstrações TLM

O ponto A: reflete o modelo executável do sistema, este ponto é o de entrada ao projeto ao nível de sistema apresentado na figura 3.6.

Evolução do ponto A para o ponto B

Nesta evolução o sistema foi particionado (hardware/software), as memórias foram dimensionadas. O tipo de EC ainda não foi definido no momento desta evolução.

O ponto B: representa o sistema com os elementos da computação, mestres e escravos definidos e modelados no nível TLM de tempo estimado. O tipo de EC ainda não está definido. Este ponto é representado na figura 3.6 após do particionamento hardware/software.

Ponto B para o ponto C

Nesta transição, são definidos: o tipo da EC, seu tamanho (quantidade de

barramentos), o mapeamento dos elementos (nos barramentos) e o número de pontes.

O ponto C representa o sistema com o tipo da EC definido, assim como os seus parâmetros globais. Aplicando as métricas de desempenho do nível de tempo estimado podemos escolher a política de arbitragem (incluindo os parâmetros de configuração) assim como a velocidade dos barramentos (alta/baixa velocidade)

Ponto C para o ponto D

Refinamento dos elementos de computação, através da definição e implementação de todas as características funcionais e arquiteturais. Para realizar a integração entre a computação e a comunicação (diferentes níveis) é preciso utilizar *wrappers* que permitam realizar a simulação utilizando modelos em níveis diferentes.

O ponto D é uma representação do sistema com os elementos de computação totalmente definidos e configurados. O tipo da EC é definido assim como os seus parâmetros globais e parte dos locais. Neste ponto a situação do projeto da EC é semelhante ao ponto C. Portanto, aplicando as métricas de desempenho do nível de tempo estimado podemos escolher e configurar a política de arbitragem (incluindo os parâmetros de configuração), assim como a velocidade dos barramentos

Ponto C para o ponto E

Refinamento da EC adicionando e configurando as características do protocolo a ser utilizado.

O ponto E é uma representação do sistema com os elementos de computação modelados no nível de tempo estimado. A EC é configurada e modelada com todas suas características funcionais e arquiteturais.

Ponto D para o ponto F

Refinamento dos elementos de computação, através da definição detalhada das suas características, assim como das interfaces dos elementos. As interfaces podem utilizar qualquer funcionalidade da EC, incluindo as transferências e as rajadas especiais. Não há necessidade de instanciação de wrappers (elementos no mesmo nível).

O ponto F representa o sistema com todos os seus parâmetros e características tanto da computação e comunicação definidos, prontos para ser instanciados no nível RTL (através de blocos IP, ou desenvolvimento de modelos).

Ponto E para o ponto F

Refinamento completo dos protocolos da EC. Não há necessidade de instanciação de wrappers (elementos no mesmo nível), as interfaces podem utilizar qualquer funcionalidade da EC, incluindo as transferências e as rajadas especiais.

Ponto A para o ponto H

Esta transição não é possível porque para definir o tipo de EC é necessário ter definido os elementos de hardware (seção 3.2). No ponto H, os elementos de computação ainda estão modelados no nível TLM atemporal, portanto o particionamento hardware/software não foi realizado e a listagem de elementos mestres e escravos não está disponível. Esta listagem é necessária para iniciar o projeto da EC. Todos os caminhos que passam através do ponto H não são factíveis.

Ponto B para o ponto I

Esta transição é possível, mas o objetivo desta tese é realizar o projeto da EC ASAP. Ir através do ponto I atrasaria o início do projeto da EC (comunicação atemporal), indo na contra mão do nosso objetivo. Portanto, o caminho que passa através do ponto I não é satisfatório.

Aplicando como ponto de partida (no nosso projeto) o ponto A que representa a especificação funcional do sistema, o objetivo é chegar até o nível RTL. Para isso primeiro deve ser atingido o ponto F que representa a descrição detalhada do sistema e é o ponto mais próximo da descrição RTL. Para fazer o percurso podem existir múltiplos caminhos através dos diferentes pontos. Assumindo que cada percurso só pode mudar de nível num dos dois eixos (por transição) num ambiente *top-down,* seis possíveis caminhos foram identificados:

- Caminho 1: A – B – C – D – F
- Caminho 2: A – B – C – E – F
- Caminho 3: A – B – I – E – F
- Caminho 4: A – H – C – E – F
- Caminho 5: A – H – C – D – F
- Caminho 6: A – H – G – D – F

A partir das análises apresentadas, dois caminhos podem ser utilizados ao longo do projeto: O caminho 1 e o caminho 2. O caminho 3 pode ser utilizado para a verificação funcional da arquitetura, mas não é útil para o projeto da EC utilizando o critério ASAP (de definição de seus parâmetros). Os outros três caminhos (4, 5 e 6) não são viáveis pelo fato de que o ponto H não é factível. O caminho a ser percorrido para o projeto da EC depende da disponibilidade de elementos, assim como o critério do projetista.

5.4 MaLOC conceitos básicos

MaLOC (*Muti Abstraction Levels On-Chip Communication Design*) é uma metodologia de projeto da EC de um *SoC* que é realizada no nível de sistema (SLD). Decisões de projeto são tomadas com base em duas importantes considerações: 1) Adoção da estratégia *ASAP* (*as soon as possible*), isto é, os parâmetros de configuração são definidos o mais cedo possível; 2) valores de cada

parâmetro são decididos avaliando a maior quantidade de métricas de desempenho possíveis para cada nível de abstração. Esta metodologia é utilizada para o projeto de barramentos hierárquicos e NoC.

A metodologia fundamenta-se em 4 tipos de relações ou associações:

1. Relações entre os parâmetros de configuração apresentadas na seção 5.2. Estas relações são: precedência, concatenação, concorrência.
2. Precedência hierárquica entre os níveis TLM.
3. Associação entre métricas de desempenho e os níveis TLM.
4. Associação entre métricas de desempenho e parâmetros de configuração.

A figura 5.4 apresenta o uso da metodologia aplicada para barramentos hierárquicos. Ela inicia no nível TLM atemporal (seção 4.2.1). A análise é feita a partir da distribuição dos dados gerados por cada mestre do sistema e a taxa de comunicação entre elementos. Desta análise obtêm-se diferentes opções de: tamanho da EC e mapeamento dos elementos de hardware na EC.

No estágio seguinte, o nível TLM, o número e o tipo de pontes são definidos. Adicionalmente, é preciso solucionar a contenção presente na EC o que se faz definindo e configurando a política de arbitragem. Ainda neste nível são definidos a largura e o tipo do barramento.

Finalmente, no nível de ciclos de relógio as características lógicas da EC e as interfaces são implementadas.

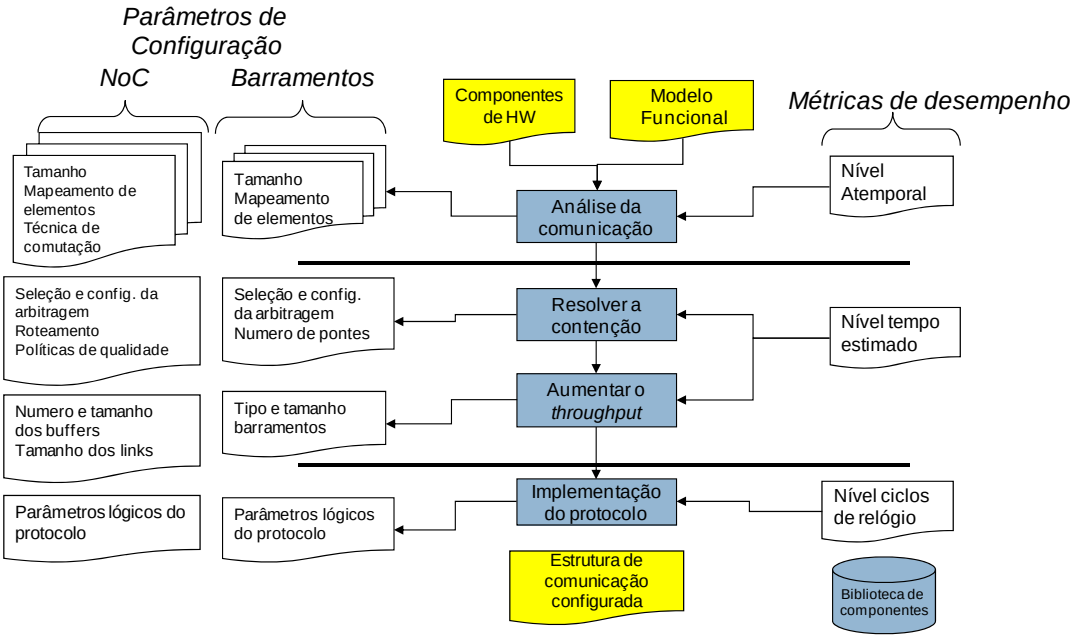

Figura 5.4. Metodologia MaLOC para EC baseadas em barramentos

MaLOC permite projetar tanto EC baseadas em barramentos e em NoC, neste tesa, MaLOC foi estabelecido para os dois tipos de estrutura, porem a parte experimental só foi realizada com barramentos.

5.5 Sumário

Neste capítulo foi apresentada a metodologia MaLOC que percorre os três níveis TLM apresentados no capítulo 4. A tomada de decisões da metodologia está baseada em: 1) decisões ASAP, para diminuir o tempo do projeto; 2) decisões orientadas por métricas de desempenho, que permitem aumentar a confiabilidade do resultado final. MaLOC apresenta quatro tipos de relações: 1) entre os parâmetros de configuração (precedência, concatenação, concorrência); 2) Dos níveis TLM utilizados; 3) Um conjunto de métricas de desempenho por nível de abstração; 4) O conjunto de parâmetros de configuração por nível utilizado.

6. Utilização de MaLOC - Estudo de caso

Neste capítulo é apresentado o uso da metodologia MaLOC para dimensionar a EC baseada em barramentos. A Seção 6.1 apresenta as condições utilizadas para realizar as diferentes simulações apresentadas neste capítulo. A Seção 6.2 apresenta a análise de robustez dos modelos utilizados. A Seção 6.3 apresenta a análise de fidelidade dos modelos utilizados e os três estudos de caso: 1) sistema composto por geradores de tráfego; 2) um sistema duplo FFT e 3) um roteador IP. Finalmente, a Seção 6.4 apresenta o sumário deste capítulo.

6.1 Condições gerais

As diferentes simulações e análises realizadas neste capítulo utilizaram um *framework* de avaliação de desempenho que é apresentado na figura 6.1 e constituído por:

- Um conjunto de geradores de tráfego que podem ser determinísticos ou pseudo-aleatórios.
- A aplicação que é composta pelos seus elementos mestres, escravos e o seu software.
- O motor da simulação que é composto pela linguagem SystemC V2.2 e a biblioteca TLM V1.0 utilizando o compilador *gcc* V4.3.
- Ferramentas e algoritmos de captura das informações necessárias para obter o valor das métricas de desempenho a serem aferidas.
- Critérios de análise que ajudam na tomada de decisões a partir dos valores obtidos das métricas de desempenho.
- Os modelos da EC para ser avaliados nos diferentes níveis de abstração apresentados no capítulo 4. Nesta tese, os modelos utilizados são da EC AMBA [AMB03].

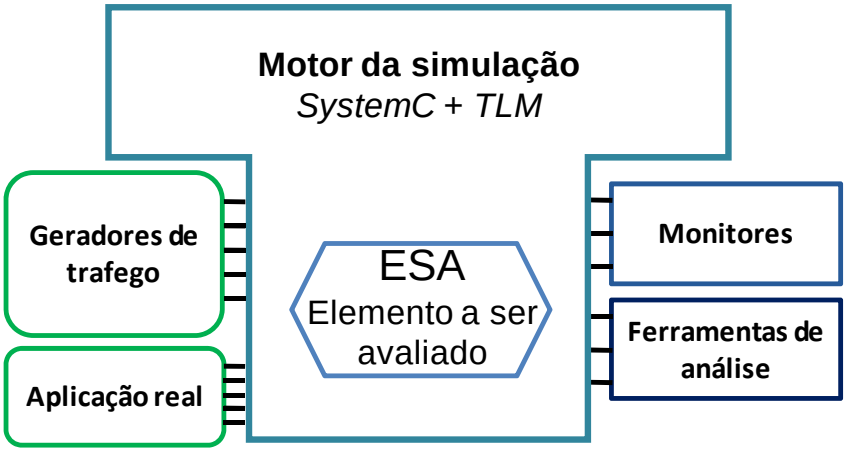

Figura 6.1. Framework utilizado para a análise de desempenho

Cada simulação integra na sua execução o processo de obter o valor das métricas escolhidas (Ferramenta de captura do *framework*). Os resultados podem ser apresentados de duas formas: 1) no ambiente da simulação; 2) num arquivo com as informações importantes da simulação como: tempo da mesma, valor das métricas escolhidas e qualquer outra informação que seja relevante para a análise a ser realizada. Estes resultados são analisados utilizando ferramentas e os critérios apresentados no capítulo 5 e o valor dos parâmetros de configuração são definidos. Para facilitar a análise de resultados, um formato de geração e entrada dos resultados foi desenvolvido. O fluxo da utilização do formato desenvolvida e das ferramentas a serem utilizadas é apresentado na figura 6.2

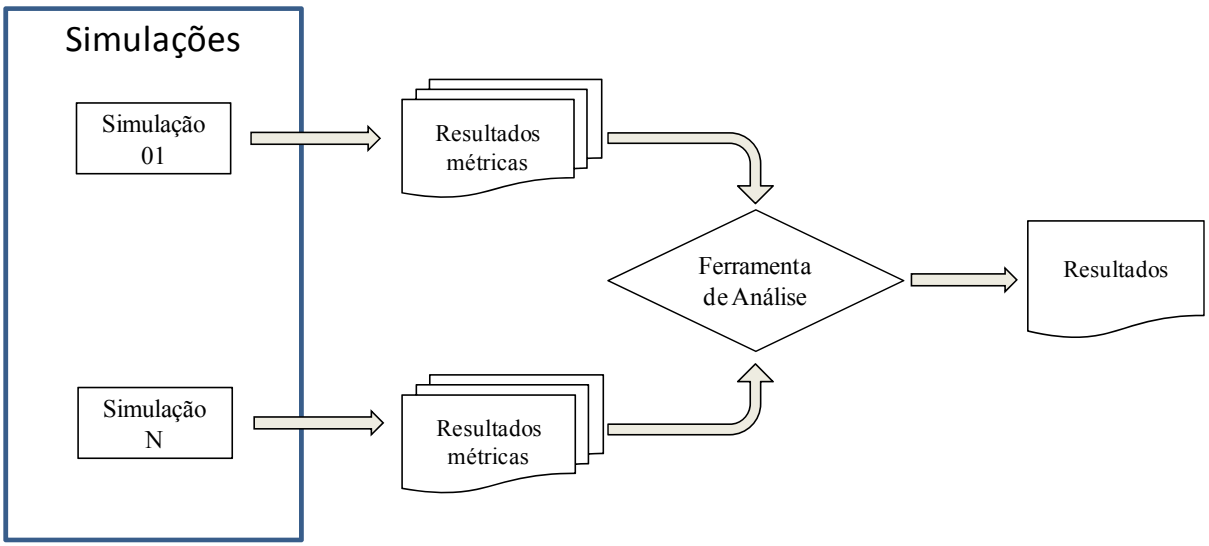

Figura 6.2. Fluxo de análise dos resultados

6.2 Análise de robustez

A primeira análise realizada consiste em verificar se os modelos utilizados transferem corretamente os dados que devem ser lidos ou escritos. Cada modelo da EC a ser utilizado neste capítulo foi avaliado. Um gerador de tráfego paramétrico foi utilizado. Este realiza o seguinte procedimento: 1) geração dos dados a serem armazenados; 2) uma operação de escrita em modo rajada dos mesmos em endereços contínuos na memória; 3) um tempo de espera (latência mestre); 4) uma operação de leitura em modo rajada de todos os dados gerados; 5) a comparação dos dados armazenados. (figura 6.3)

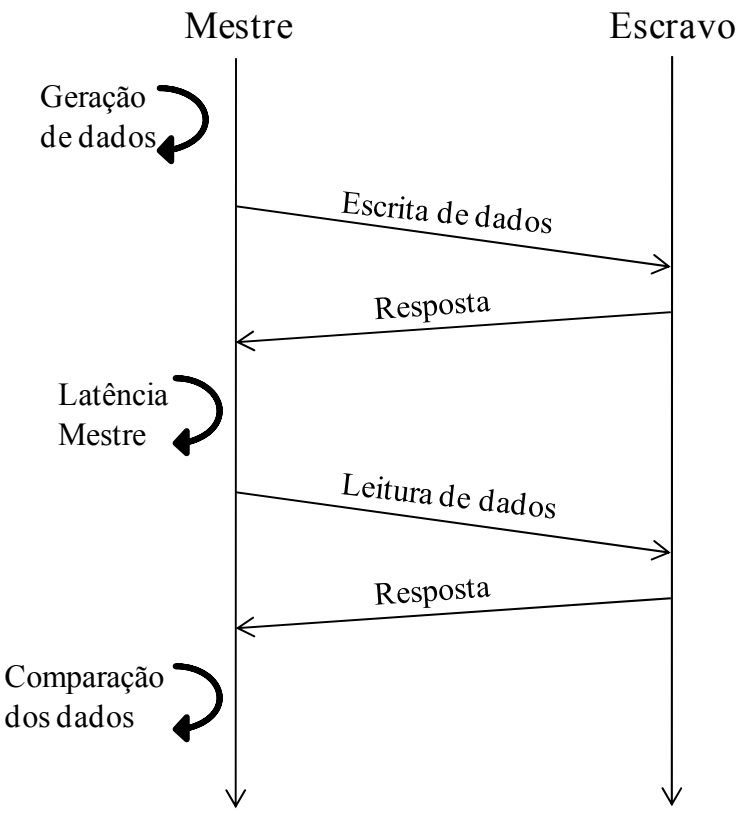

Figura 6.3. Gerador de tráfego para análise de robustez

A análise de robustez foi aplicada aos modelos temporais utilizados. A tabela 6.1 apresenta os resultados obtidos.

Tabela 6.1. Resultados robustez dos dados

Endereço inicial:	0x00001000
Total dados gerados:	10000
Total dados escritos na memória:	10000
Total dados lidos da memória:	10000
Dados comparados corretos:	100%
Erros:	0%

6.3 Estudos de caso

Nesta seção é apresentada a utilização da metodologia MaLOC. A primeira parte desta seção apresenta um estudo de fidelidade das métricas obtidas nos diferentes níveis temporais. A segunda parte apresenta MaLOC sendo utilizado para realizar o dimensionamento da EC de três aplicações: 1) Utilizando geradores de tráfego determinísticos; 2) um sistema duplo FFT; 3) um roteador de pacotes IP (*internet protocol*).

6.3.1 Fidelidade

A fidelidade é definida em [AUR] como: "1. Qualidade de fiel; lealdade. 3. Observância rigorosa da verdade; exatidão". A fidelidade na nossa tese consiste na "lealdade" que os resultados obtidos nos diferentes níveis TLM podem oferecer em relação aos valores que podem ser obtidos no modelo de implementação. É normal que os níveis mais abstratos ofereçam menor precisão, mas a fidelidade que eles podem oferecer deve ser analisada, para poder definir a utilidade dos diferentes níveis. Os resultados são considerados com boa fidelidade se ao mudar algum parâmetro do sistema, os resultados obtidos (nos níveis analisados) apresentam a mesma tendência.

A análise de fidelidade faz uma comparação entre os resultados obtidos nos dois níveis TLM temporais. Para realizar esta análise foi utilizado um conjunto de geradores de tráfego que utilizam o modelo de comunicação composto por: uma

transferência de leitura em rajada com tamanho T1. Uma latência do mestre que é de T1 ciclos. A continuação uma escrita em rajada com tamanho T2 é feita. A seguinte transação é definido pelo parâmetro (*t_entretr* ciclos). Este modelo da comunicação apresenta dois parâmetros que podem ser modificados, o tamanho de cada rajada (T1 e T2) e o tempo entre transações. (figura 6.4).

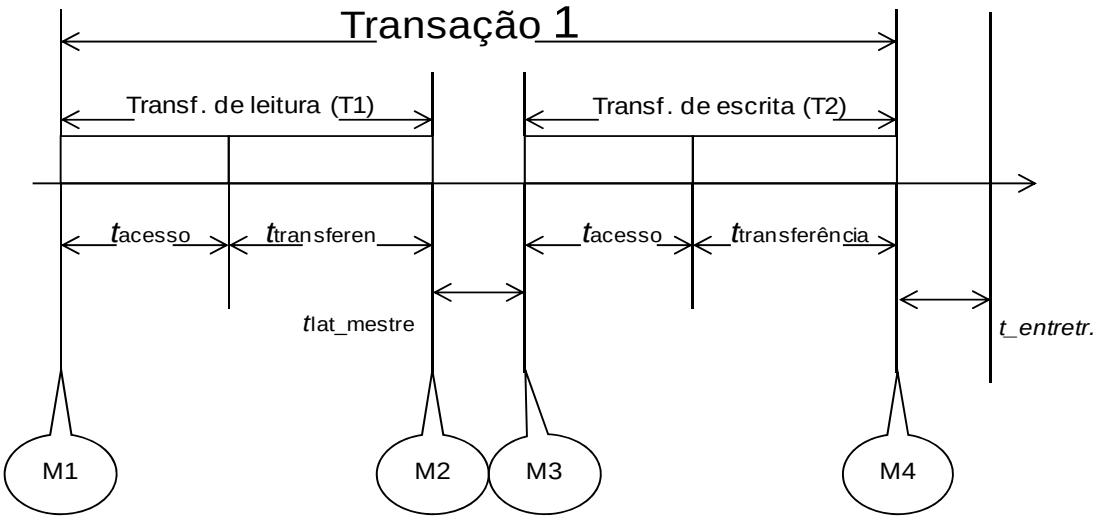

Figura 6.4. Modelo da comunicação dos geradores de tráfego

Para realizar esta análise foi utilizado um sistema composto por:

- Quatro geradores de tráfego,
- O barramento AMBA modelado nos dois níveis temporais TLM
- Política de arbitragem é de prioridades fixas.
- Uma memória

Dois cenários foram analisados: com e sem bloqueio do barramento no uso. O primeiro cenário permite que ao se ter acesso ao barramento (*grant*), o mestre tenha uso exclusivo até terminar a operação (leitura ou escrita), mesmo que outro elemento mestre com maior prioridade peça para utilizar o barramento (*request*). No segundo cenário (sem bloqueio), a operação (leitura ou escrita) é interrompida assim que um mestre com maior prioridade pede para usar o barramento. Para cada cenário analisado os dois parâmetros do modelo da comunicação (tamanho da rajada e tempo entre transações) foram modificados de forma a criar quatro condições diferentes de trafego (tabela 6.2):

Tabela 6.2. Condições de tráfego simuladas

Condição de tráfego	Tamanho da rajada	Tempo entre transações (ciclos)
1	2, 2, 2, 2 4, 4, 4, 4 8, 8, 8, 8 10, 10, 10, 10 12, 12, 12, 12 16, 16, 16, 16 24, 24, 24, 24 32, 32, 32, 32 50, 50, 50, 50 100, 100, 100, 100 200, 200, 200, 200 500, 500, 500, 500	10, 10, 10, 10 20, 20, 20, 20 50, 50, 50, 50 100, 100, 100, 100 200, 200, 200, 200 500, 500, 500, 500 1000, 1000, 1000, 1000 2000, 2000, 2000, 2000 5000, 5000, 5000, 5000 10000, 10000, 10000, 10000
2	2, 1, 2, 1 4, 2, 4, 2 8, 4, 8, 4 10, 5, 10, 5 12, 6, 12, 6 16, 8, 16, 8 24, 12, 24, 12 32, 16, 32, 16 50, 25, 50, 25 100, 50, 100, 50 200, 100, 200, 100 500, 250, 500, 250	10, 10, 10, 10 20, 20, 20, 20 50, 50, 50, 50 100, 100, 100, 100 200, 200, 200, 200 500, 500, 500, 500 1000, 1000, 1000, 1000 2000, 2000, 2000, 2000 5000, 5000, 5000, 5000 10000, 10000, 10000, 10000
3	2, 2, 2, 2 4, 4, 4, 4 8, 8, 8, 8 10, 10, 10, 10 12, 12, 12, 12 16, 16, 16, 16 24, 24, 24, 24 32, 32, 32, 32 50, 50, 50, 50 100, 100, 100, 100 200, 200, 200, 200 500, 500, 500, 500	10, 5, 10, 5 20, 10, 20, 10 50, 25, 50, 25 100, 50, 100, 50 200, 100, 200, 100 500, 250, 500, 250 1000, 500, 1000, 500 2000, 1000, 2000, 1000 5000, 2500, 5000, 2500 10000, 5000, 10000, 5000
4	2, 1, 2, 1 4, 2, 4, 2 8, 4, 8, 4 10, 5, 10, 5 12, 6, 12, 6 16, 8, 16, 8 24, 12, 24, 12 32, 16, 32, 16 50, 25, 50, 25 100, 50, 100, 50 200, 100, 200, 100 500, 250, 500, 250	10, 5, 10, 5 20, 10, 20, 10 50, 25, 50, 25 100, 50, 100, 50 200, 100, 200, 100 500, 250, 500, 250 1000, 500, 1000, 500 2000, 1000, 2000, 1000 5000, 2500, 5000, 2500 10000, 5000, 10000, 5000

As 24 opções de configuração para a identificação dos mestres (usado na política

de arbitragem) foram simuladas para cada condição de tráfego. O tempo de simulação foi de 4 milhões de ciclos para cada um. O total de simulações para cada cenário foi de (120 possibilidades de tráfego * 24 configurações de mestres * 2 níveis de abstração * 4 situaçoes) 23040 simulações.

As tabelas 6.3 e 6.4 apresentam os resultados para as quatro condições de tráfego nos dois cenários analisados. Para cada análise foi comparada a duração média de cada gerador de tráfego. Os resultados apresentam as fidelidades: total (do maior ao menor resultado); da maior e da menor duração. A tabela 6.5 apresenta a comparação dos tempos da simulação obtidos em cada nível numa maquina Linux com um processador Pentium 4HT executando a 3,6GHz.

Tabela 6.3. Resultados da fidelidade operações com bloqueio do barramento

	Cenários			
	1	**2**	**3**	**4**
Total (%)	99.58	82.29	83.61	72.71
Erros (%)	0.42	17.71	16.39	27.29
Max comum (%)	99.58	88.75	93.43	81.07
Min. comum (%)	100	87.083	88.47	93.19

Tabela 6.4. Resultados da fidelidade operações sem bloqueio do barramento

	Cenários			
	1	**2**	**3**	**4**
Total (%)	96.875	72.29	83.26	98.33
Erros (%)	3.125	27.71	16.74	1.67
Max comum (%)	96.875	90.07	88.29	99.03
Min. comum (%)	100.00	75.97	99.02	100

Os resultados das tabelas 6.3 e 6.4 indicam uma alta fidelidade nos resultados totais (>72%). Existem 3 cenários com resultados maiores que 96%. É possível obter resultados ainda melhores filtrando situações que possivelmente não acontecem como tamanhos da rajada muito maiores do que o tempo entre transações. Os resultados obtidos analisando os resultados maiores e menores apresentam uma maior fidelidade (>87% com bloqueio e >75% sem bloqueio). Estes resultados indicam que os resultados obtidos com MaLOC no nível de tempo de precisão de transferências são confiáveis.

Tabela 6.5. Resultados tempos de simulação com bloqueio de barramento

	Cenários			
	1	**2**	**3**	**4**
Média execução TLM-AT	2,47	2,42	2,59	2,62
Média execução TLM-PT	6,12	5,94	6,51	6,26
Média execução TLM-CR	9,82	9,54	10,02	9,75
Diferencia AT-TE	**2,47X**	**2,45X**	**2,51X**	**2,39X**
Diferencia TE-CR	**1,60X**	**1,6X**	**1,54X**	**1,55X**
Total execução TLM-PT	17637	17109	18736	18036
Total execução TLM-CR	28306	27475	28876	28082
Diferencia (%)	60,49%	60.58%	54,12%	55,69%

Tabela 6.6. Resultados tempos de simulação sem bloqueio de barramento

	Cenários			
	1	**2**	**3**	**4**
Média execução TLM-AT	2,47	2,42	2,59	2,62
Média execução TLM-PT	10,28	6,14	6,71	6,42
Média execução TLM-CR	12,53	9,78	10,35	10,01
Diferencia AT-TE	4,17X	2,53X	2,59X	2,45X
Diferencia TE-CR	1,21X	1,59X	1,54X	1,56X
Total execução TLM-PT	3950	17693	19328	18514
Total execução TLM-CR	4813	28171	29809	28810
Diferencia (%)	21,84%	59,22%	54,22%	55,61%

Os resultados dos tempos da simulação indicam que além de uma boa fidelidade no valor das métricas, os tempos de simulação são até 60% menores. Indicando um bom compromisso fidelidade-tempo de execução.

6.3.2 Geradores de tráfego

Este exemplo é um sistema composto por quatro geradores de tráfego (com diferentes condições de operação), duas memórias, a EC baseada em barramentos com política de arbitragem baseada em prioridades fixas. A figura 6.5 apresenta as condições de tráfego entre os diferentes elementos do sistema. Os geradores

utilizam o modelo da comunicação da figura 6.4. A comunicação entre elementos mestres é feita através de acessos a memórias e modificação de variáveis. Os parâmetros utilizados são apresentados na tabela 6.7:

Tabela 6.7. Parâmetros do exemplo usando geradores de tráfego

Modelos	AMBA TLM atemporal e tempo aproximado
Geradores de tráfego	Paramétricos
Métricas usadas	Localidade da comunicação Nível de utilização por canal Duração média *Throughput* de dados Nível de utilização do barramento
Restrições	Cenários de comunicação
Parâmetros a serem configurados	Tamanho da EC Mapeamento dos elementos Configuração da política de arbitragem

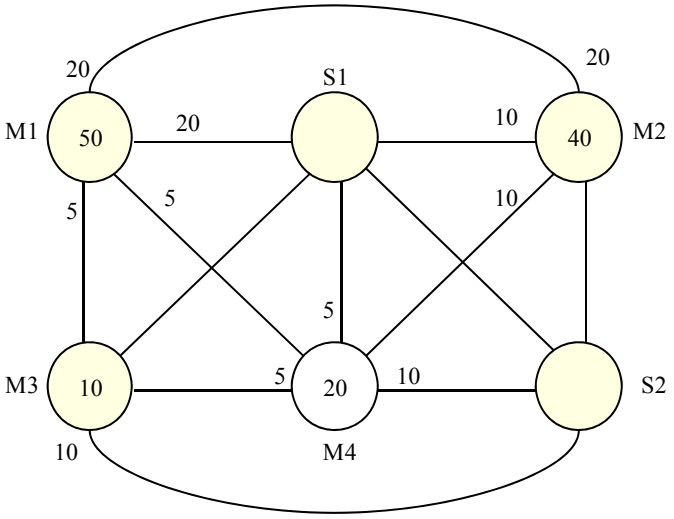

Figura 6.5. Condições de tráfego entre os geradores de tráfego

Utilizando MaLOC é configurada a EC. A primeira análise é feita utilizando as métricas atemporais (seção 5.3.1): localidade da comunicação e nível de utilização do canal. Os resultados destas métricas são apresentados na tabela 6.8.

Tabela 6.8. Resultados das métricas atemporais exemplo geradores de tráfego

$LCA_{M3S2} = 100\%$
$LCA_{M2M1} = 50\%$
$LCA_{M4S2} = 50\%$
$LCA_{M1S1} = 40\%$
$LCA_{M1M2} = 40\%$
$LCA_{M2S1} = 25\%$
$LCA_{M2M4} = 25\%$
$LCA_{M4S1} = 25\%$
$LCA_{M4M3} = 25\%$
$LCA_{M1M3} = 10\%$
$LCA_{M1M4} = 5\%$

$NUC\big|_{M1M2}\big| = 33,33\%$
$NUC_{M1S1} = 16,6\%$
$NUC_{S1M2} = 8,33\%$
$NUC_{M2M4} = 8,33\%$
$NUC_{M1M3} = 4,16\%$
$NUC_{M1M4} = 4,16\%$
$NUC_{M3M4} = 4,16\%$
$NUC_{S1M4} = 4,16\%$
$NUC_{M3S2} = 8,33\%$
$NUC_{M4S2} = 8,33\%$

Os resultados anteriores indicam entre quais geradores existe o maior tráfego do sistema (nível de utilização) e entre quais geradores existe a maior comunicação (localidade da comunicação) e quais nunca se comunicam: M1 com S2; M2 com M3 e S2; M3 com S1. Analisando os elementos com NUC maior de 5% pode-se observar que eles apresentam dois grupos de elementos que geram a maior quantidade de tráfego M1, M2 e M4 se comunicando com S1. Os outros elementos M3 e S2 apresentam baixos resultados. Analisando a LCA maior do que 30% observam-se que o elemento M4 apresenta a maior comunicação com os elementos S2 e M3, portanto, a partir destes resultados, estes dois elementos podem estar mapeados no mesmo barramento. Adicionalmente, M1 apresenta sua maior comunicação com M2 e S1. Destes resultados duas possíveis arquiteturas podem ser identificadas chamadas de arquitetura 01 e 02, cada arquitetura apresenta dois barramentos e diferentes opções de mapeamento dos elementos (geradores e memórias). As arquiteturas são apresentadas nas figuras 6.6 e 6.7.

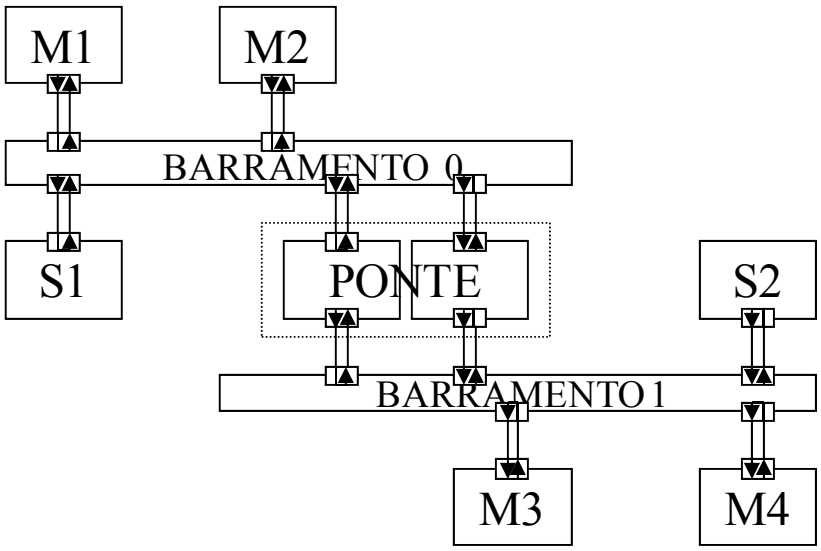

Figura 6.6. Arquitetura 01 do exemplo com geradores de tráfego

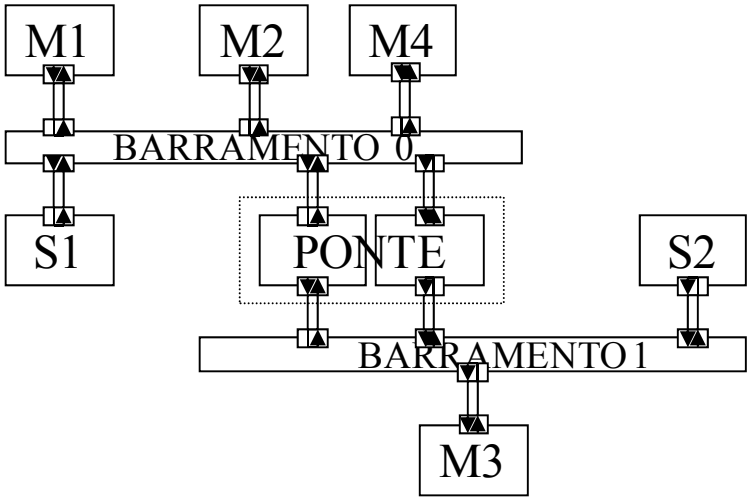

Figura 6.7. Arquitetura 02 do exemplo com geradores de tráfego

A seguinte parte da análise é configurar a política de arbitragem (prioridades fixas), para isto, três estratégias foram adotadas para definir as prioridades de acesso:

1. Maior prioridade para as pontes nos dois barramentos; depois: maiores geradores de tráfego → maior prioridade.
2. Maior prioridade para as pontes nos dois barramentos; depois: maiores geradores de tráfego → menor prioridade.
3. Maior prioridade no barramento 1 da ponte. Maiores geradores de tráfego → maior prioridade.

Quatro simulações foram realizadas utilizando as três estratégias anteriores. A tabela 6.9 apresenta os parâmetros utilizados:

Tabela 6.9. Parâmetros das simulações do exemplo com geradores de tráfego.

	Bus 0	**Bus 1**
Simulação A Arquitetura 01	M1 = P2 M2 = P3 Ponte = P1	M3 = P2 M4 = P3 Ponte= P1
Simulação B Arquitetura 01	M1 = P3 M2 = P2 Ponte = P1	M3 = P3 M4 = P2 Ponte = P1
Simulação C Arquitetura 01	M1 = P1 M2 = P2 Ponte = P3	M3 = P2 M4 = P3 Ponte = P1
Simulação D Arquitetura 02	M1 = P2 M2 = P3 Ponte = P1 M4 = P4	M3 = P2 Ponte = P1

As métricas obtidas foram: duração média e nível de utilização do barramento. Os resultados obtidos são apresentados nas tabelas 6.10 e 6.11:

Tabela 6.10. Duração média do exemplo com geradores de tráfego.

Duração média (ciclos/palavra)	**M1**	**M2**	**M3**	**M4**
Simulação A Arquitetura 01	1,39	2,76	1,15	3,36
Simulação B Arquitetura 01	2,29	1,49	2,40	2,12
Simulação C Arquitetura 01	1,36	2,28	7,43	9,10
Simulação D Arquitetura 02	1,26	1,75	2,68	11,47

Tabela 6.11. Nível de utilização dos barramentos do exemplo com geradores de tráfego

NUB (%)	Total
Simulação A Arquitetura 01	119,05
Simulação B Arquitetura 01	108,79
Simulação C Arquitetura 01	85,73
Simulação D Arquitetura 02	102,15

Analisando os resultados do NUB se observa que a simulação C apresenta o menor resultado, indicando que as prioridades definidas para as pontes ocasionam um "gargalo" no sistema. Por tanto, neste exemplo, as pontes devem apresentar a maior prioridade para permitir um maior aproveitamento dos barramentos. Deste resultado a alternativa C é descartada. Os resultados da duração média indicam que a simulação D apresenta um sistema desbalanceado[28] (figura 6.10), representado na diferencia entre a maior e a menor duração média para cada gerador, (M1: 1,26; M4: 11,47 ciclos/palavra). Por tanto a alternativa D de configuração é descartada. As outras duas alternativas apresentam os menores resultados de duração média, nas duas as pontes têm as maiores prioridades. Comparando as duas, a alternativa escolhida é a B devido a que apresenta um sistema mais balanceado (tabela 6.12).

[28] Um sistema balanceado permite equilibrar o uso do barramento entre os diferentes elementos, adicionalmente evita efeitos de *starving* aonde um dos elementos apresenta um maior uso aumentando a duração média das transferências dos outros.

Tabela 6.12. EC configurada do exemplo com geradores de tráfego

Número de barramentos	Dois
Mapeamento de elementos	Arquitetura 1
Política de arbitragem	Prioridades fixas
Configuração da politica	Simulação B
Largura do barramento	32 bits
Tipo de barramento	Os dois barramentos de alta velocidade

6.3.3 Duplo conjunto codec/decoder FFT

Este exemplo está composto por um par de funções que realizam a transformada rápida de Fourier (*FFT*) sobre um conjunto de dados gerados e outro par que realizam a função inversa da FFT, baseadas no benchmark SPLASH [SPL]. Cada função é implementada em software. O sistema está composto por quatro processadores MIPS modelados com a ferramenta ARCHC [ARCHC]. O software foi compilado utilizando um cross-compilador MIPS num sistema Linux X86. Os códigos de maquina ASM obtidos foram carregados nas três memórias do sistema (M1, M2M3 e M4). M1 é a memória de dados do processador P1; M2M3 é a memória dos processadores P2 e P3 e M4 do P4. As memórias M2 e M3 estão integradas gerando condições de tráfego adicionais.

O primeiro processador de cada conjunto (P1 e P3) gera um vetor de dados, realiza a FFT e finaliza sua execução. O segundo processador (P2 e P4) recebe a resposta e realiza a FFT inversa. Finalmente, o vetor original e os resultados do segundo processador são comparados para verificar que o processo foi correto. A figura 6.8 apresenta os componentes do exemplo. A tabela 6.13 apresenta os parâmetros de configuração deste exemplo.

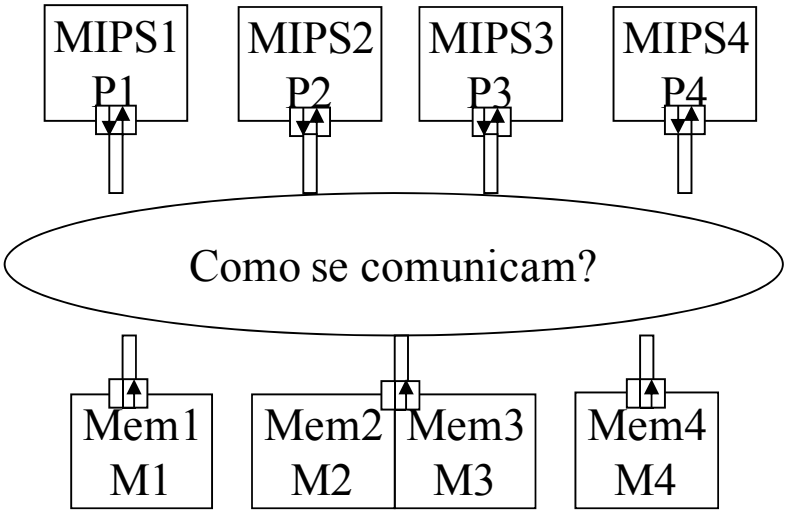

Figura 6.8. Arquitetura exemplo dupla FFT

Tabela 6.13. Condições iniciais do exemplo duplo FFT

Modelos	AMBA TLM atemporal e tempo aproximado
Geradores de tráfego	Aplicação – Dupla FFT codec/decoder
Métricas usadas	Localidade da comunicação Nível de utilização por canal Duração média *Throughput* de dados Nível de utilização do barramento
Restrições	M2M3 um elemento.
Parâmetros a serem configurados	Tamanho da EC Mapeamento dos elementos Configuração da política de arbitragem Tipo do barramento

Utilizando MaLOC é configurada a EC. A primeira análise é feita utilizando as métricas atemporais: localidade da comunicação e nível de utilização do canal. Os resultados destas métricas são apresentados nas tabelas 6.14 e 6.15

Tabela 6.14. NUC do exemplo duplo FFT

Nível de utilização de canal (%) (NUC)	
P1M1	28,03
P2M2M3	22,65
P1P2	0,39

P3M2M3	28,03
P4M4	16,61
P3P4	4,26

Tabela 6.15. LCA do exemplo duplo FFT

Localidade de comunicação atemporal (LCA) (%)	
P1M1	99,15
P2M2M3	99,32
P2P1	0,68
P1P2	0,85
P3M2M3	99,15
P4M4	80,51
P3P4	0,85
P4P3	19,49

Os resultados do NUC indicam dois grupos pelos quais a maior parte do tráfego da aplicação circula: 1) P1M1 e P2M2M3; 2) P3M2M3 e P4M4. A LAC indica três grupos aonde o tráfego gerado entre eles fica: 1) P1M1 e P2M2M3; 2) P3M2M3 e P4M4; 3) P4P3. A partir destas análises as seguintes arquiteturas são propostas (tabela 6.16).

Tabela 6.16. Arquiteturas propostas para o exemplo duplo FFT

Arquitetura	**Quantidade de barramentos**	**Mapeamento de elementos**	
Arquitetura 1	3	P1→B1 M1→B1 P2→B2 M2M3→B2	P3→B2 P4→B3 M4→B3
Arquitetura 2	2	P1→B1 M1→B1 P2→B2 M2M3→B2	P3→B2 P4→B2 M4→B2
Arquitetura 3	2	P1→B1 M1→B1 P2→B1 M2M3→B1	P3→B1 P4→B2 M4→B2

Arquitetura 4	3	P1→B1 M1→B1 P2→B2 M2M3→B1	P3→B3 P4→B2 M4→B2

Cada proposta procura aumentar a comunicação dentro de cada barramento (LAC) segmentando os elementos que apresentam uma menor comunicação entre se. A tarefa seguinte é configurar a política de arbitragem utilizando as mesmas três estratégias utilizadas no exemplo da seção 6.3.2. Sete alternativas (simulações) foram escolhidas (tabela 6.17).

Tabela 6.17. Simulações a serem realizadas no exemplo duplo FFT

Arquitetura	**Simulação**	**Prioridades das pontes**	**Prioridades maiores geradores**
Arquitetura 1	A	Maior	Maior
Arquitetura 1	B	Maior	Menor
Arquitetura 2	C	Maior	Maior
Arquitetura 2	D	Maior	Menor
Arquitetura 2	E	Menor	Menor
Arquitetura 3	F	Maior	Maior
Arquitetura 4	G	Maior	Maior

As métricas obtidas são: duração média, nível de utilização dos barramentos, participação dos elementos, *throughput* de dados, localidade da comunicação. Os resultados obtidos são apresentados na figura 6.9 e nas tabelas 6.18 até 6.21.

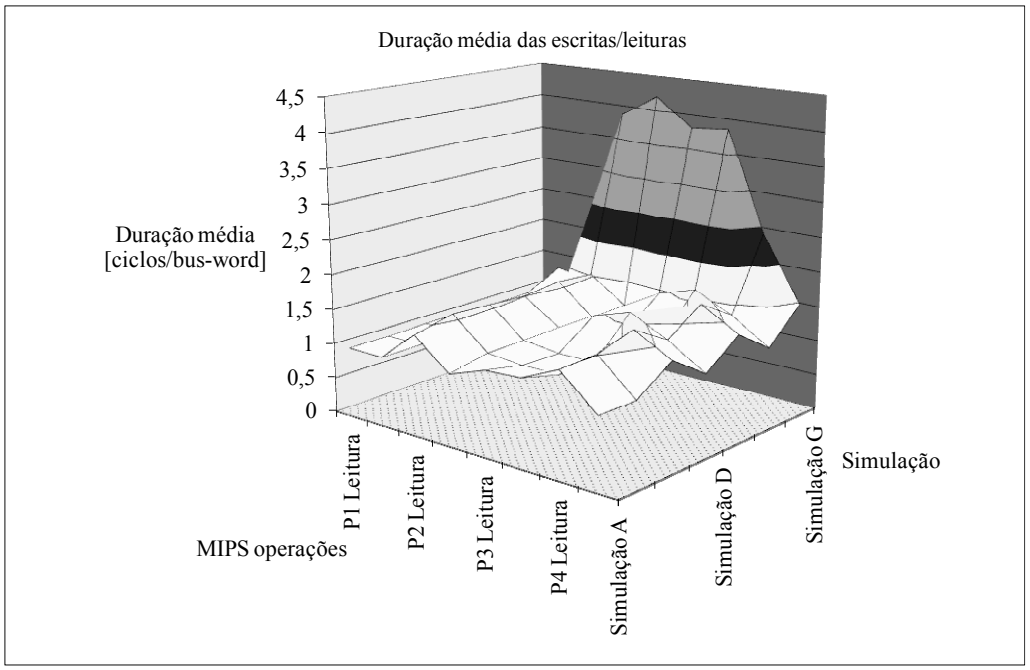

Figura 6.9. Duração média do exemplo duplo FFT

Tabela 6.18. Nível de utilização do barramento (NUB) do exemplo duplo FFT

NUB (%)	**Simulação A**	**Simulação B**	**Simulação C**	**Simulação D**
B1	17,83%	18,29%	18,17%	17,15%
B2	29,51%	29,97%	41,94%	40,98%
B3	12,40%	12,27%	0,00%	0,00%
NUB (%)	**Simulação E**	**Simulação F**	**Simulação G**	
B1	17,14%	49,74%	28,43%	
B2	40,86%	11,99%	29,64%	

Tabela 6.19. Participação dos elementos do exemplo duplo FFT

Participação dos elementos (%)						
Barramento 01	**Simulação A**	**Simulação B**	**Simulação C**	**Simulação D**	**Simulação E**	**Simulação F**
P1	99,25%	99,06%	99,07%	99,01%	99,00%	35,88%
P2	0,75%	0,94%	0,93%	0,99%	1,00%	24,34%
P3	0,00%	0,00%	0,00%	0,00%	0,00%	35,69%
P4	0,00%	0,00%	0,00%	0,00%	0,00%	4,09%
Barramento 02						
P1	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
P2	37,24%	37,02%	28,59%	27,72%	27,59%	0,00%
P3	55,36%	56,01%	43,29%	41,21%	41,31%	0,00%
P4	7,40%	6,96%	28,12%	31,07%	31,09%	100,00%
Barramento 03						
P1	0%	0%				
P2	0%	0%				
P3	0%	0%				
P4	100%	100%				

Tabela 6.20. *Throughput* de dados do exemplo duplo FFT

	Simulação A	**Simulação B**	**Simulação C**	**Simulação D**
***Throughput* de dados (bytes/cycle)**	2,384038317	2,414609295	2,397409341	2,318413581
	Simulação E	**Simulação F**	**Simulação G**	
***Throughput* de dados (bytes/cycle)**	2,313290378	2,388047333	1,705444369	

Tabela 6.21. Localidade da comunicação do exemplo duplo FFT

Localidade da comunicação (%)						
	Simulação A	**Simulação B**	**Simulação C**	**Simulação D**	**Simulação E**	**Simulação F**
B1	29,69%	30,02%	30,03%	29,29%	29,34%	79,91%
B2	49,69%	49,87%	69,68%	70,42%	70,36%	16,68%
B3	17,02%	16,84%	0,00%	0,00%	0,00%	0,00%
B1B2	0,23%	0,28%	0,28%	0,29%	0,30%	3,41%
B2B3	3,78%	3,50%	0,00%	0,00%	0,00%	0,00%

Os resultados da duração média indicam que a simulação G (arquitetura 4) apresentam os maiores valores devido ao alto fluxo de comunicação entre os barramentos 2 e 3 (baixa localidade da comunicação). Por tanto, esta arquitetura é descartada. Analisando a localidade da comunicação, observa-se que as simulações com a arquitetura 1 apresentam a maior localidade por barramento, reduzindo a troca de operações entre barramentos, isto aumenta o *throughput* e diminui a duração média das transferências. A simulação F (arquitetura três) apresenta a menor localidade da comunicação, portanto é descartada. As duas arquiteturas que podem ser escolhidas (1 e 2) apresentam resultados semelhantes por tanto qualquer uma poderia ser utilizada, mas a arquitetura dois apresenta um menor conjunto de componentes (dois barramentos ao invés de três) com um desempenho ligeiramente inferior da arquitetura um. Por tanto, a escolha é a arquitetura dois que oferece além do um adequado desempenho um menor consumo de energia e área (menor quantidade de componentes da EC). A tabela 6.22 apresenta os parâmetros de configuração da EC

Tabela 6.22. Parâmetros configurados da EC do exemplo dupla FFT

Número de barramentos	Dois
Mapeamento de elementos	Arquitetura 2
Política de arbitragem	Prioridades fixas
Configuração da política	Simulação C

Largura do barramento	32 bits
Tipo de barramento	Todos os barramentos de alta velocidade

6.3.4 Roteador de pacotes IP (*Internet Protocol*)

Este exemplo é um roteador de pacotes IP baseado no exemplo disponível na ferramenta *SystemStudio* da *Synopsys*. Composto por quatro elementos mestres: 1) DMA que armazena os pacotes de entrada na memória (*controller*); 2) o algoritmo de roteamento que escolhe o porto de saída de cada pacote armazenado na memória (*sw*). 3-4) Dois portos de saída da aplicação (*local* e *forward*).

A aplicação é estimulada através de um gerador de pacotes IP. O tempo que transcorre entre pacote gerado é um parâmetro que é variado para obter diferentes condições de tráfego. Todo pacote gerado é avaliado pelo *parser* que analisa se a sua estrutura, em caso do pacote estiver errado ele é descartado, se estiver correto ele é encaminhado para o DMA para ser armazenado na memória. Em caso que a memória esteja cheia o pacote é descartado. O algoritmo de roteamento continuamente lê cada pacote armazenado na memória, identifica o endereço de destino e com base numa tabela informa (escrevendo num campo do pacote) seu porto de saída e sua liberação. Finalmente, os portos de saída lêem o status dos pacotes na memória, se ele está liberado, eles identificam se o pacote deve ser enviado por ele, liberam a memória e realizam o envio. A memória é parametrizável em função da quantidade máxima de pacotes que podem ser armazenados simultaneamente. A figura 6.10 apresenta os componentes do roteador e o gerador dos pacotes

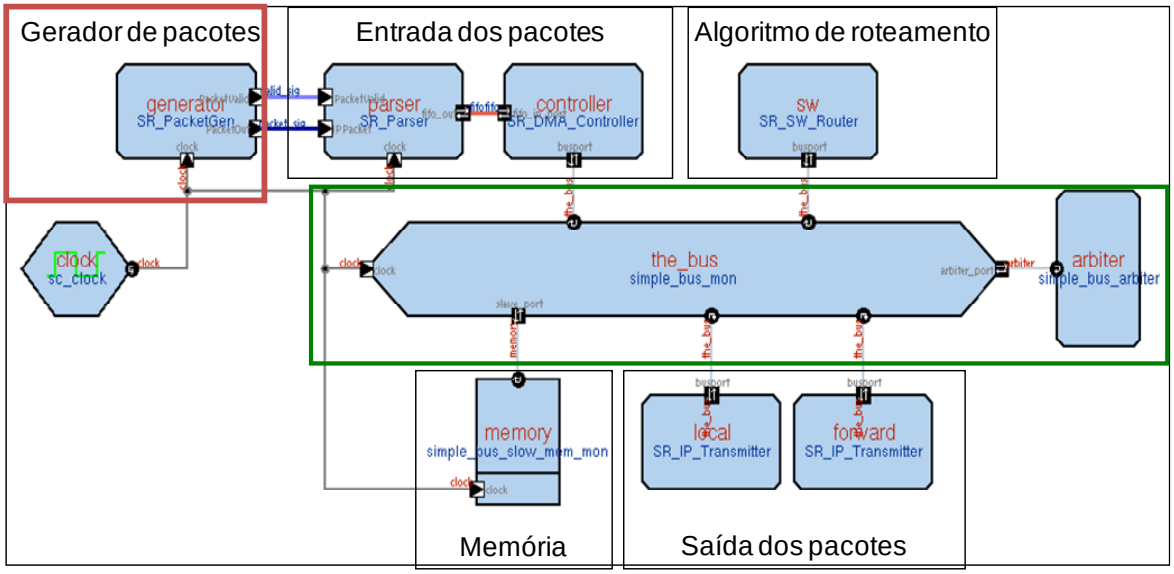

Figura 6.10. Exemplo roteador IP

Nesta análise três condições de tráfego foram utilizadas (baixo, típico e extremo). Cada uma é obtida modificando a geração de pacotes IP. Estes são gerados seguindo uma distribuição normal. Os dois parâmetros modificados são: o intervalo entre pacotes gerados e a semente utilizada na distribuição. A quantidade de pacotes foi a mesma utilizando os mesmos tempos de simulação: 10.000.000 de ciclos

Tabela 6.23. Condições de tráfego utilizadas no exemplo do roteador IP

	Semente	**Intervalo entre pacotes (ciclos)**	**Pacotes gerados**
Extremo	50	20	19429
Meio	176	50	14989
Baixo	226	100	10891

Procura-se obter a EC solução mais adequada para as três condições. A tabela 6.24 apresenta as condições iniciais deste exemplo:

Tabela 6.24. Condições iniciais do exemplo do roteador IP

Modelos	AMBA TLM atemporal e tempo aproximado
Geradores de tráfego	Aplicação – roteador IP
Métricas usadas	Localidade da comunicação Nível de utilização por canal Duração média *Throughput* de dados Nível de utilização do barramento
Restrições	Três níveis de tráfego. Tamanho da EC = 1 barramento
Parâmetros a serem configurados	Configuração da política de arbitragem Tipo do barramento Tamanho da memória

Este exemplo apresenta como restrição o tamanho da EC: um barramento; por tanto todos os elementos de hardware devem ser mapeados nele. Para configurar a política de arbitragem três estratégias são utilizadas:

- Maiores geradores de tráfego → maior prioridade (Arquitetura 1).
- Maiores geradores de tráfego → menor prioridade (Arquitetura 2).
- Entrada pacotes (DMA) → maior prioridade; saída pacotes (*forward*, *local*) → menor prioridade (Arquitetura 3)

O primeiro parâmetro a ser identificado é o tamanho da memória, representado na quantidade máxima de pacotes que podem ser armazenados simultaneamente na memória. Para isto é analisada a taxa de pacotes perdidos no sistema. A figura 6.11 apresenta os resultados para as três condições de tráfego estabelecidas e as três estratégias apresentadas. Os resultados mostram que para bancos maiores a 32, a taxa apresenta um comportamento quase similar com um custo maior de área. Para bancos menores que 8 a taxa aumenta significativamente. Em conclusão, os valores mais adequados são 8, 16 e 32.

Os resultados do *throughput* de dados e do nível de utilização do barramento (figuras 6.12 e 6.13) indicam um uso intensivo da EC (> 78%) nas diferentes

condições de tráfego. Mesmo com condições de tráfego com quase o dobro de geração de pacotes, a diferencia não é maior que 6% nos resultados obtidos para ambas as métricas.

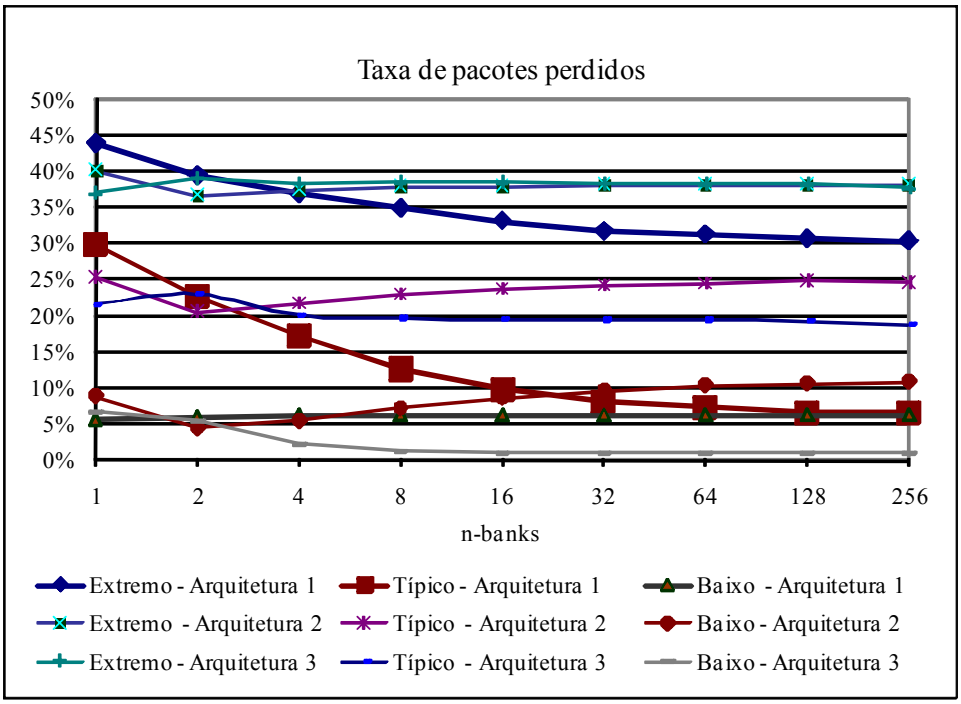

Figura 6.11. Taxa de pacotes perdidos do exemplo roteador IP

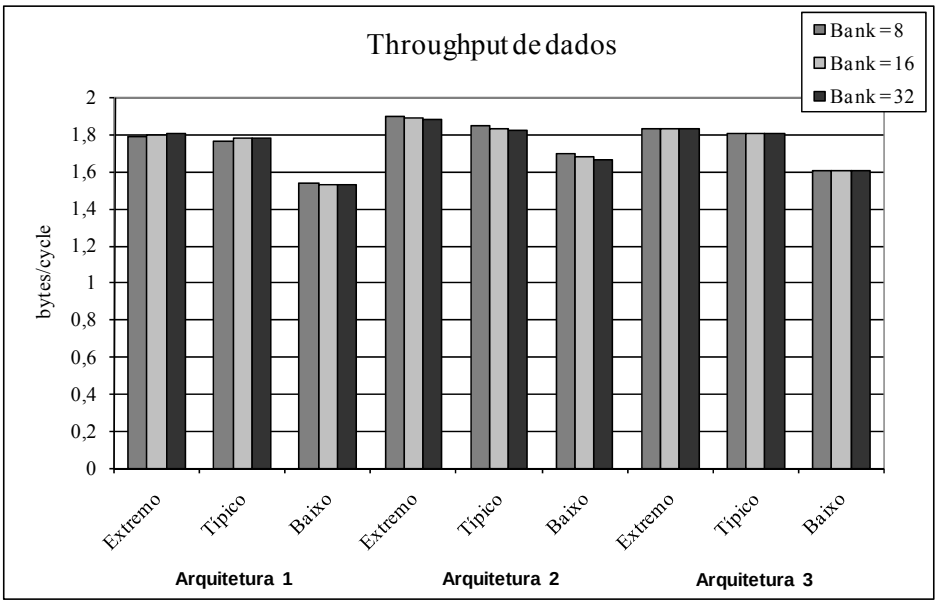

Figura 6.12. *Throughput* de dados do exemplo roteador IP

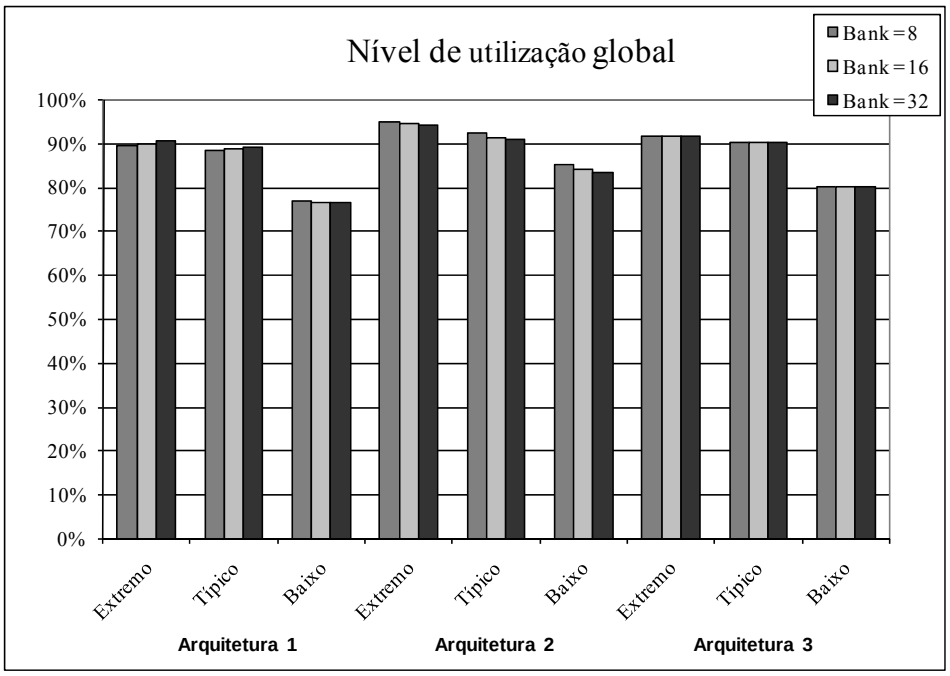

Figura 6.13. Nível de utilização do barramento do exemplo roteador IP

Os resultados da duração média das diferentes arquiteturas são apresentados nas figuras 6.14, 6.15 e 6.16. Os resultados indicam que a arquitetura 2 apresenta o maior valor (*controller* = 36 ciclos/palavra). Esta situação explica porque tem as maiores taxas de pacotes perdidos nas três condições de tráfego (figura 6.11), indicando que o "gargalo" está no armazenamento dos pacotes. Esta arquitetura é descartada. Cabe ressaltar que o uso de duas métricas (taxa de pacotes perdidos e duração média) permitiu identificar a deficiência da estrutura, oferecendo uma maior confiabilidade nos resultados obtidos.

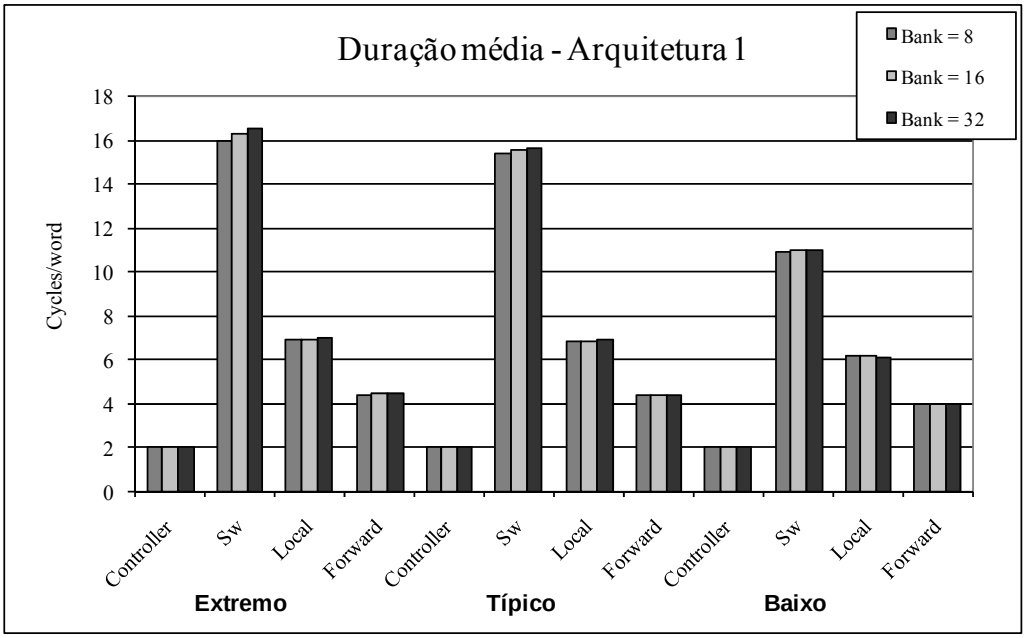

Figura 6.14. Duração média da arquitetura 1 do exemplo roteador IP

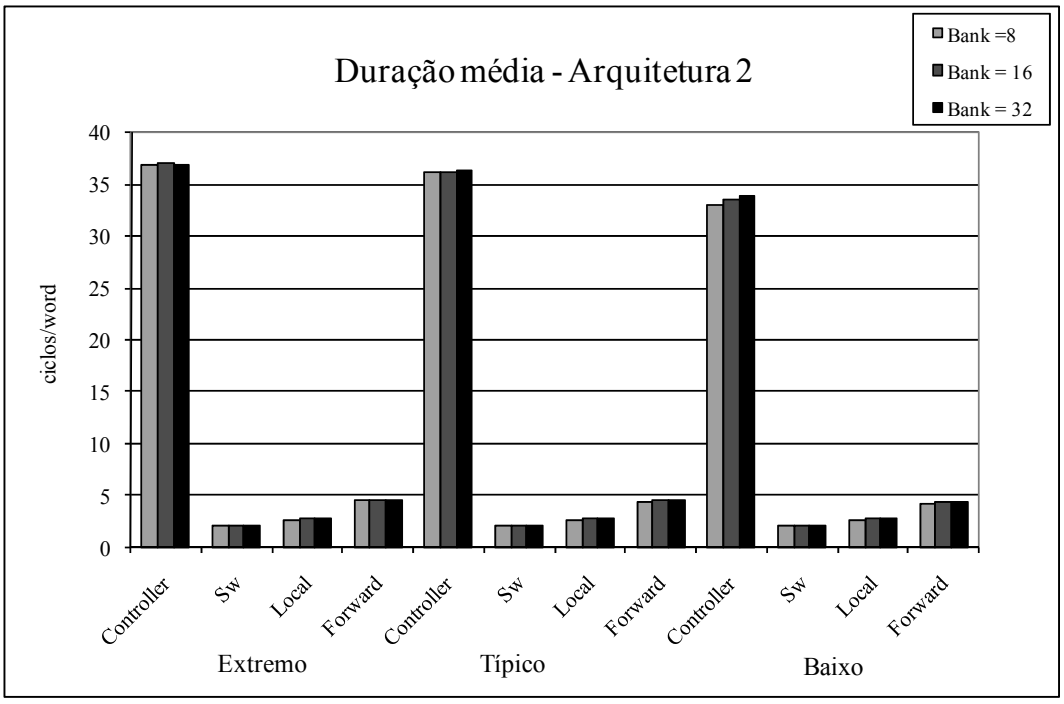

Figura 6.15. Duração média da arquitetura 2 do exemplo roteador IP

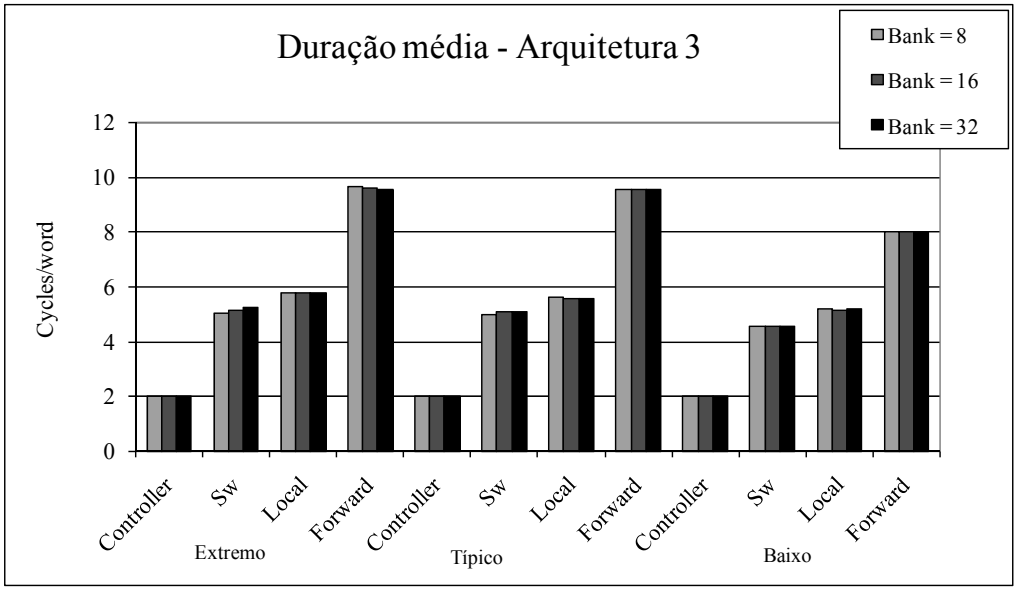

Figura 6.16. Duração média da arquitetura 3 do exemplo roteador IP

Das outras duas arquiteturas, os resultados indicam duas diferentes situações de tráfego. Na arquitetura 1, o software de roteamento apresenta a maior duração, mas os elementos de saída e entrada de pacotes apresentam os menores resultados, isto permite ter um maior *throughput* de pacotes através do roteador. Na arquitetura 2, o sistema é mais balanceado, mas as durações médias dos elementos de entrada e saída dos pacotes são maiores que na arquitetura 1. As duas arquiteturas apresentam resultados satisfatórios, mas o fato da arquitetura 1 apresentar um menor nível de utilização do barramento indica um possível menor consumo de energia. Finalmente, a arquitetura 1 apresenta um menor consumo de energia (representado pelo nível de utilização do barramento) com adequado desempenho, a arquitetura 2 apresenta um sistema mais balanceado com adequado desempenho. Os dois resultados são válidos (tabelas 6.25 e 6.26).

Tabela 6.25. Opção 1 da EC configurada do exemplo roteador IP

Parâmetro	Valor
Número de barramentos	1
Mapeamento	Todos no único barramento
Política de arbitragem	Baseada em prioridades
Configuração da política	Controller = 1 Sw = 4 Local = 3 Forward = 2
Largura do barramento	32 bits
Tipo de barramento	Alta velocidade
Número de bancos da memória	16

Tabela 6.26. Opção 2 da EC configurada do exemplo roteador IP

Parâmetro	Valor
Número de barramentos	1
Mapeamento	Todos no único barramento
Política de arbitragem	Baseada em prioridades
Configuração da política	*Controller* = 1 *Sw* = 2 *Local* =3 *Forward* = 4
Largura do barramento	32 bits
Tipo de barramento	Alta velocidade
Número de bancos da memória	32

6.4 Sumário

Neste capítulo foi apresentada a utilização da metodologia MaLOC (proposta nesta tese). Os modelos utilizados nesta tese foram verificados através de uma análise de robustez que comprovaram a sua correta funcionalidade em operações

de escrita e leitura (usando vários comprimentos). Uma análise de fidelidade foi feita, os resultados indicam uma fidelidade entre 73% e 96% para os resultados de duração média (completo), este resultado permite garantir que uma maior eficiência de MaLOC. Os resultados obtidos nos estudos de caso mostraram que o uso de um conjunto variado de métricas de desempenho permite aumentar a confiabilidade na tomada das decisões. A metodologia também foi utilizada para configurar parâmetros que não são parte da EC, como no exemplo do roteador IP, o tamanho da memória.

7. Conclusões e trabalhos futuros

A complexidade crescente (tanto em funcionalidade como na arquitetura) dos sistemas eletrônicos digitais sobre silício (conhecidos na literatura como *System-on-Chip*, *SoC*) exige novas metodologias de projeto. O projeto no nível de sistemas (SLD) é proposto para aumentar a eficiência do projeto de *SoC*. SLD exige novas linguagens (como SystemC) e níveis de abstração (como TLM). Embora SLD seja muito mencionado na literatura, não existe um conjunto de tarefas universalmente aceito.

O uso de TLM está presente em várias tarefas do projeto de *SoC* como: análise da arquitetura, análise de desempenho, verificação funcional, desenvolvimento do software. A modelagem TLM apresenta dois tópicos: 1) a sintaxe e semântica das linguagens utilizadas (nesta tese SystemC) e 2) os níveis de abstração existentes (três níveis utilizados nesta tese).

A comunicação dentro do *SoC* (também chamada de comunicação on-chip) pode ser abstraída de seu nível de sinais (RTL) até níveis mais abstratos que modelam a comunicação entre processos através de funções que apresentam diferentes graus de detalhes. Nesta tese a modelagem da comunicação foi realizada através dos diferentes níveis de transações (TLM) apresentados.

A comunicação entre os elementos que compõem o *SoC* (comunicação on-chip) apresenta grandes desafios devido à complexidade crescente dos mesmos. Nesta tese foi proposta a metodologia MaLOC para o projeto da estrutura de comunicação de um *SoC*. A tomada de decisões é dirigida por duas importantes considerações: 1) tomar as decisões ASAP (sigla do inglês *as soon as possible*) que possibilita diminuir o esforço de projeto, através da antecipação da tomada de algumas decisões de projeto que de outra forma seriam tomadas no nível RTL. O que exigiria um maior esforço representado pelo desenvolvimento dos modelos e pelo poder computacional requerido para realizar as simulações. 2) guiadas por um conjunto diverso de estimativas de desempenho que aumenta a confiabilidade

nos valores obtidos.

MaLOC exige um conjunto de relações: 1) entre parâmetros de configuração (precedência, seqüência, dependência, concorrência); 2) entre níveis TLM válidos e úteis; 3) entre um conjunto de métricas de desempenho por nível de abstração; 4) entre métricas de desempenho e os parâmetros de configuração. O fato de utilizar na metodologia níveis TLM permite que os modelos sejam re-aproveitados em outras tarefas do projeto.

Esta tese partiu de três dissertações de mestrado [ESL03, LOZ05, SEP06] que estudaram o problema da avaliação de desempenho de EC, baseadas em barramento e em NoC. Utilizando o nível TLM de precisão de transferências.

Os resultados obtidos apresentam uma fidelidade total da duração media maior do que 72%. Analisando resultados parciais, a fidelidade aumenta, no caso da maior duração, o resultado é superior a 81%; no caso da menor duração, o resultado é superior 75%. Os tempos de simulação obtidos apresentam que o tempo das simulações no nível TLM de ciclos de relógio são 60% maiores. Estes resultados indicam que MaLOC apresenta uma alta confiabilidade nas decisões tomadas no nível mais abstrato.

Os resultados dos estudos de caso indicaram que o uso de um conjunto variado de métricas de desempenho permitiu tomar decisões mais confiáveis permitindo reduzir as atividades através do barramento (nível de utilização do barramento) refletindo numa redução do consumo de energia, com um desempenho similar; reduzir a área (menor quantidade de elementos). MaLOC também foi utilizada para a definição de parâmetros do sistema (que não pertencem a EC) como a quantidade de pacotes que podem ser armazenados simultaneamente. Este fato mostra que MaLOC apresenta um grande potencial para o projeto de outros componentes de um *SoC*.

7.1 Trabalhos futuros

Existem três grandes áreas de pesquisa a partir dos resultados desta tese:

1) A geração de uma ferramenta que automatize a tomada das decisões. A partir do *framework* apresentado no capítulo 6. para aumentar a eficiência e confiabilidade do projeto da EC.

2) Verificar a fidelidade e utilidade de MaLOC no projeto de outras EC como crossbars e NoC.

3) A utilização de MaLOC para o projeto de diferentes elementos de um *SoC*

Estes três propostas de pesquisa permitirão fortalecer os princípios básicos de MaLOC e estender sua utilidade.

8. Referências

[ALE01] P. Alexander, D. Barton "*A Tutorial Introduction to Rosetta*", Hardware Description Languages Conference, San Jose, CA, 2001

[AMB03] "*AMBA AHB Cycle Level Interface Specification, Design Methodology and Tools*" ARM, 2003.

[ARTE] Arteris, disponivel em http://www.arteris.com

[ATI07] R. Atitallah et al, "*An MPSoC Performance Estimation Framework Using Transaction Level Modeling*" 13th International Conference on embedded and Real-Time Computing Systems and Applications, RTCSA 2007.

[ARCHC] Ferramenta ARCHC disponível em http://www.archc.org

[ARM99] "*AMBA Specification"* ARM – 1999.

[AUR] Dicionário Aurelio, Versão eletrônica.

[BAI07] B. Bailey, G. Martin, A. Piziali; "*ESL Design and Verification*"; Morgan Kauffman, 2007

[BEN02] L. Benini, G. De Micheli; *"Networks on Chips: A new SoC Paradigm"*. Transactions on Computer , Volume: 35 Issue: 1 , Jan. 2002

[BEN05] L. Benini, D. Bertozzi, "*Network-on-Chip Architectures and Design Methods*", IEE Proceedings on Computer and Digital Techniques. Vol 152 No. 2. March 2005

[BLU04] H. Blume, T. von Sydow, T.G. Noll, *"Performance Analysis of SoC Communication by Application of Deterministic and Stochastic Petri Nets"*, 4th International Workshop on Systems, Architectures, Modeling, and Simulation, SAMOS IV, 2004.

[CAI03] L. Cai, D. Gajski, *"Transaction Level Modeling: An Overview in System Level Design"*, International Conference on hardware/software codesign and system synthesis CODES-ISSS, 2003.

[CAL03] N. Calasanz et al. "From VHDL Register Transfer Level to SystemC Transaction Level Modeling a comparative case study", 16th SBCCI, 2003

[CHA99] H. Chang et al.; "*Surviving the SoC revolution: A guide to platform-Based Design"*, Kluwer Academic Publishers, Boston 1,999.

[CON03] J. Connell, *"ARM System-Level Modleing"*, Technical Document at http://www.SystemC.org

[COR99] "*The CoreConnect Bus Architecture",* IBM Microelectronics, 1999.

[DON04] A. Donlin, "Transaction level modeling: flows and use models", ISSS-CODES, 2004.

[ESL03] S.Eslava "Estimativas de desempenho de estruturas de comunicação para projetos de sistemas sobre silício" Dissertação de Mestrado, Universidade de São Paulo, 2003.

[ESL04] S. Eslava, M. Strum, J.C. Wang "*Performance estimation of on-chip communication structures using SystemC-TLM*", X WorkShop Iberchip, Cartagena - Colombia. 2004

[ESL05] S. Eslava, G.Lozada, M.Strum, J.C. Wang "*Performance estimation for on-chip communication structures at different TLM sub-levels*", XI WorkShop Iberchip, Salvador-Bahia. Março 2005

[ESL07] S. Eslava, J.C.Wang, M.Strum "*A system level design methodology for on-chip communication structures using TLM levels: MaLOC* " 2nd IEEE Colombian Workshop on circuits and systems, Bogotá 2007.

[FAR07] F. de Faria, M. Strum, J.C.Wang. "*A system-level performance evaluation methodology for Network Processors Based on Network calculus Analytical Modeling*", IEEE Computer Society annual symposium on VLSI, ISVLSI 2007.

[FENI] Projeto Fenix, disponível em http://www.brazilip.org.br

[GAJ94] D. Gajski, et al, "*Specification and design of embedded systems*" Ed. Prentice Hall, 1994

[GHE05] F. Ghenessia Ed., *"Transaction Level Modeling with SystemC; TLM concepts and Applications for Embedded Systems"*. Ed. Springer, 2005.

[GOO05] K. Goosens et al., "*A Design Flow for application-specific networks on chip with guaranteed performance to accelerate SoC design and verification"*, Proceedings of the Design, Automation and Test in Europe Conference and Exhibition, DATE 2005.

[GRO02] T.Grotket et al., "*System design with SystemC*", Kluwer academic publishers, 2002.

[HAB03] A. Habibi, S. Tahar, "*A Survey on System-on-a-Chip Design Languages"*, Proceedings of the third international Workshop on Systems on Chip for Real Time applications, 2003.

[HAV02] A. Haverinen et al., *"SystemC based SoC Communication Modeling for the OCP Protocol"*, White paper at http://www.SystemC.org

[JAN04] A. Jantsch, H. Tenhunen (Editores), "*Networks on Chips*", Kluwer Academic Publishers, 2004

[JAN04a] A. Jantsch; "*Modeling Embedded systems and SoC*"; Morgan Kauffman, 2004.

[JEY97] A.A. Jerraya et al., "*Behavioral synhtesis and component reuse with VHDL*", Kluwer Academic Publishers, 1997

[JEY05] A.A. Jerraya, W. Wolf, *"Multiprocessors System-on-Chip"*, Morgan Kaufman publishers, 2005

[KAL03] S. Kallakuri, A. Doboli, S. Doboli, "*Applying Stochastic Modeling to Bus Arbitration for Network-on-Chip-Systems"*, Proceedings of the International Conference on VLSI, 2003

[KAN92] K. Kant, "*Introduction to computer system performance evaluation"* Mc. Graw Hill, 1992.

[KEU00] K. Keutzer et al, *"System Level Design: Orthogonalization of Concerns and Platform-Based Design"*, IEEE Transactions on Computer-Aided Design of Circuits and Systems, Vol.19, No. 12, Dec 2000.

[KIM05] Y. Kim et al., "*Fast and Accurate Transaction Level Modeling of an Extenden AMBA 2.0 Bus Architecture*", Proceedings of the Design, Automation and Test in Europe Conference and Exhibition, DATE 2005.

[LAH01] K. Lahiri, A. Raghunathan, S. Dey, "*System-level Performance Analysis for Designing On-Chip Communication Architectures"*, IEEE Transactions on computer-aided design of integrated circuits ans systems, Vol. 20, No. 6, June 2001.

[LOG04] M.Loghi, F. Angiolini, D. Bertozzi, L. Benini, "*Analyzing on-chip Communication in a MPSoC Environment*", Proceedings of the Design, Automation and Test in Europe Conference and exhibition – DATE, 2004.

[LOZ05] G.Lozada, S.Eslava, M.Strum, J.C.Wang, "*Random Traffic Generator for Performance Estimation of on-chip Communication Structures*", XI WorkShop Iberchip, Salvador-Bahia. Março 2005

[OCP01] Open Core Protocol Specification, http://www.ocpip.org

[PAN05] S.Pandey, M. Glesner, M. Muhlhauser, "*Performance Aware On-Chip Communication Synthesis for Shared Multi-Bus Based Architecture*", Proceedings of the 18 SBCCI, 2005

[PAN05a] S. Pandey et al., "*High Level Hardware/Software Communication Estimation in Shared Mamory Architecture*", Symposium on Circuit and Systems (ISCAS), Kobe, Japan, 2005.

[PAS02] S. Pasricha, "Transaction Level modeling of *SoC* with SystemC 2.0", Synopsys User Group Conference SNUG, 2002.

[PAS04] S. Pasricha, N. Dutt, M. Ben-Rodhane, "*Extending the transaction level modeling approach for fast communication architecture exploration*", Proceedings of the Design automation Conference – DAC, June 2005.

[POLI] Projeto *POLIS* disponível em: http://embedded.eecs.berkeley.edu/Respep/Research/hsc/abstract.html

[PTOL] Projeto *Ptolemy* disponível em: http://ptolemy.eecs.berkeley.edu

[ROM05] E.L. Romero, J.C.Wang. M.Strum, "*Comparing two testbench methods for hierarchical functional verification of Bluetooth baseband adapter*";

International Conference on hardware/software codesign and system synthesis, CODES-ISSS, 2005.

[ROS05] Adam Rose et al., *"Transaction Level Modeling in SystemC"*, TLM Workgroup, 2005, Disponível em http://www.SystemC.org

[SCH06] G. Schirner, R. Dömer, "*Quantitavi analysis of Transaction Level models for the AMBA Bus*", Proceedings of the Design, Automation and Test in Europe Conference and exhibition – DATE, 2006.

[SEP06] M.J. Sepulveda, "Dissertação de Mestrado, Escola Politécnica, Universidade de São Paulo", 2006.

[SPECC] Linguagem SpecC, disponível em: http://www.cecs.uci.edu/~specc/

[SPL] SPLASH, Stanford Parallel Applications for Shared Memory, http://www-flash.stanford.edu/apps/SPLASH

[STR07] M.Strum et al., "*Digital System on Chip Design: An Overview and selected topics*"; 2nd IEEE Colombian Workshop on circuits and systems, Bogotá 2007.

[SUT99] S. Sutherland, "*The Verilog, PLI handbook*", Ed. Kluwer Academic Pub, 1999.

[SVER] Linguagem SystemVerilog disponível em http://www.systemverilog.org

[SYSC] Linguagem *SystemC* disponível em: http://www.SystemC.org

[TED05] L. Tedesco et al."*Traffic Generation and Performance Evaluation for Mesh-Based NoCs",* 18th Symposium on Integrated Circuits and Systems Design, SBCCI 2005.

[VSIA] VSI Alliance, disponível em: http://www.visi.org

[WAG04] F. Wagner et al., "*Strategies for the integration of hardware and software IP components in embedded systems-on-chip"*, Integration the VLSI Journal, 2004.

[WIL05] B. Wile, J. Goss, W. Roesner, "Comprehensive functional verification: The complete industry cycle"; Morgan Kauffman, 2005

[ZER04] N. E. Zergainoh, A. Baghdadi, A. Jerraya, "*Hardware/Software Codesign of On-chip Communication Architecture for Application-Specific Multiprocessor System-on-Chip"*, International Journal of embedded Systems Vol. 1 No. 12, September 2004.