MODELAGEM E DESCRIÇÃO DE SOCS EM DIFERENTES NÍVEIS DE ABSTRAÇÃO

Edson I. Moreno, Taciano R.Ares e Ney L. V. Calazans

Pontifícia Universidade Católica do Rio Grande do Sul – PUCRS
Faculdade de Informática – FACIN
Av. Ipiranga, 6681 – Prédio 30/ Bloco 4 Telefone: +55 51 3320-3611 – Fax: +55 51 33203621
90619-900 – Porto Alegre – RS – BRASIL

{emoreno,taciano,calazans}@inf.pucrs.br

SUMMARY

Transaction level (TL) modeling is regarded today as the next step in the direction of complex integrated circuits and systems design entry. This means that as this modeling level definition evolves, automated synthesis tools will increasingly support it, allowing design capture to start at a higher abstraction level than today. This work presents a comparison of traditional register transfer level (RTL) modeling and transaction level modeling through the implementation of a simple processor case study. SystemC is a language that naturally supports hardware transaction level descriptions. The R8 processor was described in SystemC TL and RTL versions and these were compared to an equivalent hand-coded VHDL RTL description in some key points, such as simulation efficiency and implementation results. The experiments indicate that TL descriptions present a faster path to system validation and that it is possible to envisage the automation of the design flow from this level of abstraction without significant impact on the quality of the final implementation.

RESUMO

Modelagem em nível de transação (TL) é considerada atualmente como o próximo passo para projetos de sistemas e circuitos integrados. Isto significa que com a evolução da definição dos níveis de modelagem, ferramentas de síntese automatizadas deverão gradativamente acompanhar esta evolução, permitindo que a captura de projetos inicie em níveis de abstração superiores aos atuais. Este trabalho apresenta uma comparação entre a modelagem tradicional em nível de transferência entre registradores (RTL) e a modelagem em nível de transação através da implementação de um estudo de caso de um processador simples. SystemC é uma linguagem que dá suporte a descrições de hardware em nível de transação de forma natural. O processador R8 foi descrito em SystemC nas versões TL e RTL e comparado a uma descrição manual em VHDL RTL em pontos chave, tais como eficiência de simulação e resultados de implementação. Os experimentos indicam que a descrição TL representa um caminho mais rápido para validação de sistemas, sendo possível idealizar a automação do fluxo de projeto deste nível de abstração sem um impacto significante na qualidade da implementação final.

MODELAGEM E DESCRIÇÃO DE SOCS EM DIFERENTES NÍVEIS DE ABSTRAÇÃO

Edson I. Moreno, Taciano R.Ares e Ney L. V. Calazans

Pontifícia Universidade Católica do Rio Grande do Sul – PUCRS
Faculdade de Informática – FACIN
Av. Ipiranga, 6681 – Prédio 30/ Bloco 4 Telefone: +55 51 3320-3611 – Fax: +55 51 33203621
90619-900 – Porto Alegre – RS – BRASIL

{emoreno,taciano,calazans}@inf.pucrs.br

RESUMO

Este trabalho apresenta uma abordagem de modelagem e descrição de SoCs em diferentes níveis de abstração. Como estudo de caso é apresentada a descrição do processador R8 em diferentes níveis de abstração. O processador R8 está descrito em VHDL no nível RT e em SystemC nos níveis de transação e RT. Estas descrições são utilizadas para comparar tempo de simulação, tamanho de hardware e curva de aprendizagem. O trabalho também contém uma revisão do estado da arte de níveis de abstração para captura de projetos.

1. INTRODUÇÃO

O avanço tecnológico tem permitido cada vez mais a utilização da automatização de atividades cotidianas do homem. Sistemas computacionais são uma forma moderna de prover esta automatização. Eles são formados por módulos de *hardware* e *software*, cujas operações combinadas provêem um serviço [1]. Sistemas computacionais utilizados em controle de veículos, em comunicação à distância e na automação de residências [2] são cada vez mais importantes para disponibilizar principalmente conforto e segurança.

Sistemas embarcados [3] constituem uma classe de sistemas computacionais. Apesar de possuírem a mesma estrutura interna de computadores, sendo [2] formados por processadores, memórias, barramentos e controladores de interrupção, e dotados de software, sistemas embarcados se caracterizam por não serem percebidos como computadores pelos seus usuários finais [2].

Gordon Moore observou, na década de 60 que a cada 18 meses a capacidade dos CIs duplicava [4]. Esta observação vem se mantendo válida desde então, passando a ser conhecida como a "lei de Moore". A lei de Moore funciona como um acionador da velocidade de evolução da indústria de semicondutores. Nos últimos vinte anos, tal velocidade de avanço na tecnologia de fabricação de circuitos integrados (CIs) permitiu o aumento sempre exponencial do número de transistores por área de *chip* [5]. Favorecido por este aumento de área de silício e apoiado pela necessidade de avanço tecnológico, a complexidade dos projetos eletrônicos cresceu bruscamente desde o surgimento dos circuitos integrado. Com isto, os sistemas embarcados, montados até meados da década de 70 sobre placas de circuito impresso, puderam ser implementados em um único CI, surgindo o conceito de SoC (em inglês, *System-on-a-chip*) [6]. SoCs podem ser construídos sob a forma de CIs de aplicação específica (em inglês, *Application Specific Integrated Circuit*, ASIC) ou, mais recentemente, dispositivos programáveis como FPGAs (*Field Programmable Gate Array*), e são vistos como uma nova tendência de desenvolvimento de projetos de hardware [7].

O uso de SoCs traz vantagens como a diminuição da área de um projeto de hardware, a minimização do gargalo de desempenho dado pela comunicação entre diferentes módulos e a diminuição no tempo de projeto [8]. Atualmente, ASICs com algumas dezenas de transistores já são uma realidade e contêm complexidade comparável a produtos finais em um único circuito integrado [7]. Conforme *SIA roadmap* [9], em algum momento entre os anos de 2010 e 2013, o estado da arte em ASICs vai conter mais de um bilhão de transistores. Com isso, a inclusão de módulos complexos em um único circuito integrado comparado ao estado atual é ordens de grandeza superior. Porém, a complexidade das interconexões entre os módulos e a validação global do SoC vem tornando a tarefa de concepção e gerenciamento dos projetos extremamente complexa [6], tendendo a aumentar o tempo de projeto. Por outro lado, o tempo entre o início do desenvolvimento de um produto e a momento de lançá-lo no mercado tem de ser cada vez menor para garantir a maior lucratividade (em inglês, *time-to-market*), forçando a diminuição do tempo de projeto [10]. Tais fatores exigem que novas metodologias de especificação e concepção de projetos de hardware sejam continuamente propostas.

Soluções parciais para garantir as vantagens do uso de SoCs e minimizar suas desvantagens podem ser obtidas via *técnicas de reuso de projetos*, pela *evolução dos métodos de interconexão de módulos on-chip*, e pela *elevação dos níveis de abstração* das fases de captura de projeto.

O reuso de *núcleos de propriedade intelectual* (em inglês, *Intellectual property cores*, IP cores ou núcleos IP) permite a redução de tempo de projeto através da reutilização de módulos de hardware previamente descritos. Com o aumento da capacidade dos CIs, a possibilidade de inserir cada vez mais núcleos IP em um único SoC também cresce. Todavia, núcleos IP desenvolvidos para uma aplicação específica normalmente não seguem uma padronização da sua interface externa, aquela que permite interconectá-los a outros módulos do SoC. O grande número de núcleos IP em um SoC e a ineficiência de interfaces padronizadas de interconexão atuais faz com que a reutilização de módulos torne-se uma tarefa complexa.

O uso de meios de comunicação compartilhados, tais como barramentos, para interconectar módulos de um SoC torna-se ineficiente à medida que cresce o número de módulos de hardware em um único CI. Este problema deve-se principalmente a fatores elétricos, incluindo o carregamento capacitivo dos fios do barramento, o conseqüente aumento de consumo de energia, além da limitação topológica de apenas permitir uma mensagem trafegando no meio de comunicação a cada instante. Propostas para resolver este problema podem ser obtidas através da utilização de barramentos hierárquicos ou através da adoção de conceitos de redes de computadores e sistemas distribuídos em SoCs, criando redes *intra-chip* (ou em inglês, *Networks on a chip*, NoCs) [11]. NoCs tendem a ser mais eficientes que barramentos hierárquicos por evitarem situações de bloqueamento da rede quando da comunicação entre dois núcleos IP. NoCs pressupõe núcleos IP como elementos computacionais em uma rede, e sua interconexão com outros núcleos respeita a organização de transferência de informações em camadas de protocolos.

A elevação do nível de abstração de projeto também é vista como uma contribuição para aumentar a eficiência de concepção dos projetos de hardware. Novamente, devido ao aumento do número de núcleos IP em um SoC e ao aumento da complexidade da tecnologia de interconexão destes IPs, descrições de um sistema completo em nível de transferência entre registradores (em inglês, *Register Transfer Level*, RTL) são difíceis de se desenvolver e alterar. A elevação dos níveis de abstração de projeto para os chamados *níveis sistêmicos* permite que detalhes sejam ocultados, tornando mais fácil descrever a funcionalidade de cada módulo e suas interconexões. Apesar disto, não há formalismo para definição de níveis mais altos de abstração entre os níveis sistêmicos e o RTL. A lacuna existente entre o nível de especificação e os de concepção do projeto usados hoje pode ser preenchida com níveis intermediários como o *nível não temporizado* e o *nível de modelagem de transações* (em inglês, *Transaction Level Modeling*, TLM) conforme proposto, por exemplo, por Arnout em [12].

SystemC é uma linguagem de descrição de hardware que contempla níveis sistêmicos de abstração de projeto. Esta linguagem atende à necessidade de elevação do nível de abstração de captura e validação de projeto. Construída como um conjunto de classes sobre a linguagem C++, SystemC foi desenvolvida com a finalidade de preencher a lacuna entre a especificação do projeto de sistemas computacionais e sua concepção. SystemC permite a descrição de *hardware* e *software* em um ambiente homogêneo, facilitando o processo de compreensão, descrição e validação de projeto. Com isto, a detecção de erros funcionais pode ser realizada nos estágios iniciais de desenvolvimento, diminuindo os esforços e custos de projeto.

Observada a tendência da utilização de SoCs em projetos de sistemas e as vantagens daí advindas, nota-se a complexidade intrínseca de projetos com múltiplos núcleos IP. Assim, a aplicação de técnicas de reuso de núcleos IP, o emprego de novas abordagens para definir a interconexão destes núcleos, e a elevação dos níveis de abstração para a descrição de projetos são vistos como temas relevantes a serem investigados. Este trabalho aborda o emprego da modelagem e descrição de sistema em altos níveis de abstração para projetos de sistemas.

O resto deste documento está organizado da seguinte forma. A seção 2 é realizada a revisão do estado da arte para níveis de abstração de captura de projetos. Na seção 3 são apresentados conceitos adotados para níveis de abstração e fluxo de projeto. Ainda nesta seção é apresentada a descrição do processador R8 em VHDL, no nível RT, e em SystemC nos níveis de transação e RT. As conclusões e trabalhos futuros são apresentados na seção 4.

2. NÍVEIS DE ABSTRAÇÃO EM PROJETOS DE SOCS

A complexidade do processo de projeto de sistemas digitais implica à decomposição hierárquica deste em um conjunto de passos de projeto. Um mecanismo fundamental para conduzir a decomposição do processo de projeto é o uso de abstração de informações. *Abstração* é o nome que se dá ao processo de representar um modelo via um conjunto de informações limitado a aspectos relevantes deste para um dado tipo de manipulação. Também se pode usar o termo para referir-se ao resultado do processo em si. Segundo [13], pode-se definir *nível de abstração* no contexto de projeto de sistemas computacionais como um conjunto de descrições de projeto com o mesmo grau de detalhamento.

Durante o projeto de um SoC, um grande número de descrições de projeto com diferentes níveis de detalhamento são manipulados. Por exemplo, diagramas de por-

tas lógicas contêm muito menos detalhes que uma descrição elétrica do tipo SPICE, mas ambos são descrições do projeto relevantes em algum contexto. Os primeiros podem ser usados para rapidamente validar blocos grandes, com dezenas de milhares de portas lógicas, enquanto a partir de descrições SPICE podem-se obter validações com alto grau de detalhamento, para pequenos blocos, tipicamente com não mais que alguns milhares de transistores. O número de níveis de abstração manipulados durante o projeto de sistemas complexos tem aumentado à medida que aumenta a complexidade dos projetos. Descrições de projetos de hardware no nível de transferência entre registradores (em inglês, *register transfer level*, RTL) são atualmente o estado da arte para ferramentas de síntese, conforme Jerraya et al. [14].

Com a crescente complexidade dos projetos de sistemas computacionais e a necessidade de aumento de produtividade em menor tempo, o uso de níveis de abstração superiores ao RTL têm sido propostos. Adicionalmente, com o objetivo de facilitar o processo de descrição dos projetos, abordagens diferenciadas para níveis de abstração voltados para a computação e a comunicação são adotadas, conforme detalhamento a seguir.

2.1. Níveis de abstração de computação em projetos de SoCs

Arnout [12] propôs dois níveis de abstração superiores ao RTL: o *nível não temporizado*, e o *nível de transação*. No nível não temporizado, o que se busca é descrição da funcionalidade do projeto sem precisão temporal. Nele não há separação dos diferentes módulos de hardware e software, apenas a geração de uma especificação executável. No nível de transação, a funcionalidade de cada módulo é descrita de forma abstrata e a comunicação é baseada em ciclo de transações entre os módulos. Com isto, pode-se ter a garantia do correto comportamento do sistema com simulações mais velozes por diminuir a quantidade de detalhamento.

Outra proposta de níveis de abstração em projetos de sistemas computacionais é apresentada pela Synopsys em [15]. Nesta referência são definidos como níveis de abstração superior ao RTL os níveis de abstração *não temporizado*, *temporizado* e *de transação*. No nível *não temporizado* [15], algoritmos são mais facilmente capturados, verificados e otimizados. Neste nível, os modelos do projeto que está sendo desenvolvido comunicam-se de modo ponto-a-ponto através de canais abstratos do tipo fila, acessados via operações bloqueantes de leitura e escrita. Assim, projetistas são beneficiados de duas formas. Primeiro, facilidade de desenvolvimento, pois a comunicação é simples e a sincronização é implícita. Segundo, maior velocidade de simulação, pois muitos detalhes de implementação são abstraídos neste nível.

O projetista trabalha no nível não temporizado para capturar, verificar e otimizar algoritmos. Assim, a comunicação dos módulos do sistema pode ser simulada para otimizar a transmissão e recepção através de simulações eficientes. Neste nível o desempenho obtido pode ser avaliado com métricas tais como a taxa de transferência de bits realizada com sucesso entre os módulos de transmissão e recepção..

No *nível temporizado*, empregam-se modelos funcionais dotados de modelos de atraso. A modelagem de atrasos pode considerar atrasos de processamento, de comunicação, ou ambos. Este nível de abstração é usado para analisar os efeitos de latência no comportamento do sistema e a arquitetura do sistema nos estágios iniciais do projeto. Através de especificações de atraso de processamento, é possível definir modelos funcionais temporizados para determinar se o sistema vai gerar resultados em tempo hábil ou não, conforme a especificação. Normalmente, empregam-se aqui esquemas de comunicação ponto-a-ponto, a exemplo do que ocorre no nível não temporizado. Modelos funcionais temporizados são geralmente usados para analisar o comprometimento de hardware e software nos estágios iniciais de projeto. Isto é feito pela avaliação do impacto do mapeamento de processos para ambos, hardware e software [15]. A idéia de tempo neste nível é normalmente expressa através de uma quantidade de ciclos de relógio..

No *nível de transação*, os componentes da arquitetura, tais como memória, unidade aritmética, e geradores de endereço comunicam-se através de meios compartilhados, como barramentos. Barramentos costumam envolver mecanismos de arbitragem para resolver conflitos de acesso ao meio de comunicação, que ocorrem quando vários componentes requisitam acesso a este simultaneamente. Um grande esforço é necessário para projetar e verificar os modelos comportamentais em HDL pois sua simulação é lenta. Além disto, não existem técnicas eficientes para transformar descrições puramente comportamentais HDL em hardware. A depuração precisa de software normalmente requer a simulação de hardware em níveis próximos ao RTL, o que pode tornar a verificação de um sistema complexo inviáveis.

Com a modelagem em nível de transação (em inglês, *transaction level modeling*, TLM), a implementação detalhada dos canais de comunicação não é necessária, apenas seu comportamento é modelado, expresso em termos de transações. O conceito de transação varia de autor para autor. Por exemplo, segundo Pasricha [16] transação é uma troca de dados qualquer entre módulos, independentemente do protocolo de troca empregado ou do tempo necessário para efetuar esta troca.

Descrições TLM são mais fáceis de desenvolver e usar, se comparadas a descrições RTL. A modelagem no nível de transação cria uma especificação executável do modelo que simula ordens de magnitude mais rápido que modelos

RTL. Ela provê um modelo abstrato para desenvolvedores de software testarem seus códigos no contexto de um SoC. É preferencialmente no nível de transação que todas as diferentes equipes de um projeto de SoC comunicam-se. É bem menos custoso encontrar e resolver problemas ao nível de transação do que em RTL.

2.2. Níveis de abstração de comunicação em projetos de SoCs

No passado, níveis de abstração eram definidos para expressar os módulos componentes do sistema com maior ou menor grau de detalhamento, enquanto que a descrição da interconexão entre os módulos era vista como acessória e realizada da mesma forma que a descrição dos módulos. Contudo, o crescimento da escala dos sistemas leva a uma crescente importância da interconexão em si. Surge então a necessidade de processos de modelagem específicos para a comunicação entre núcleos IP usando um conjunto de níveis de abstração próprios. Exemplos de propostas de níveis de abstração para tratar questões de comunicação são as de Haverinen et al. [17] e de Jerraya et al. [14].

Na Figura 1 é apresentada a abordagem proposta por Haverinen et al. [17]. Esta descreve diferentes níveis de abstração em uma modelagem de comunicação onde o nível mais baixo de abstração, RTL, é descrito pelo protocolo OCP. O protocolo OCP é uma proposta de padronização de interfaces para a interconexão de núcleos IP.

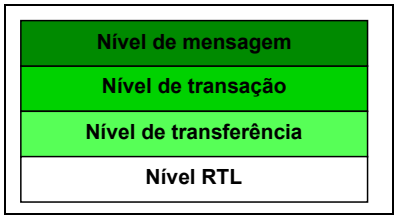

Figura 1 - Camadas de abstração na modelagem de comunicação proposta por Haverinen et al. [17].

A *camada de mensagem* não é temporizada. A transferência de dados entre os módulos envolve a passagem de vários dados, que podem ser de tipos abstrato, tais como registros estruturados em software (e.g. construções *struct* na linguagem C).

A *camada de transação* é temporizada mas abstrai o conceito de ciclos de relógio. Uma única transação nesta camada envolve a passagem de vários dados. O tempo decorrido da execução de uma transação é estimado internamente nos núcleos que trocam dados. Sistemas descritos neste nível são independentes dos protocolos de barramento específico, caso estes sejam usados.

A *camada de transferência* se caracteriza pelo comportamento baseado em ciclos de relógio (*clock*), como na camada RTL. Esta camada segue um protocolo de comunicação específico, como por exemplo *Amba*, para descrever um barramento mestre-escravo.

Jerraya [14] propõem 3 níveis de abstração superiores ao nível RTL para abstrair detalhes de comunicação: nível de serviço, nível de mensagem e nível de *driver*.

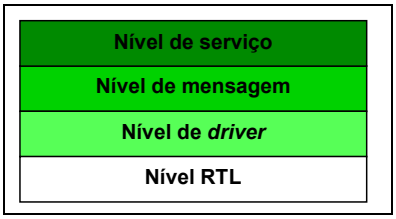

Figura 2 - Camadas de abstração na modelagem de comunicação proposta por Jerraya [14].

O nível de serviço é definido como a implementação de métodos que descrevem ações de um módulo, denominados serviços pela abstração de detalhamento. Neste nível, um módulo pode requisitar um serviço que é disponibilizado por outro. Questões como camadas de protocolo a serem adotadas, estratégias de conexão e questões de temporização não são detalhadas neste nível de abstração. Assim, a funcionalidade a ser atingida é mais facilmente descrita. Exemplos de serviços que podem ser requisitados são pedidos de impressão ou transferência de arquivos. Com esta abstração, detalhes de paralelismo e temporização são suportados.

No nível de mensagem, a comunicação é modelada através de canais com capacidade de interconectar módulos independentes, ocultando questões de protocolo e tamanho dos dados. Assim, canais implementando métodos de envio e recebimento de dados são modelados neste nível. Estes canais executam a transferência de dados ponto-a-ponto entre módulos.

No nível de *driver* a comunicação entre os módulos é definida através de uma conexão lógica que realiza trocas de tipos de dados fixos ou enumerados. O tempo de comunicação é diferente de zero, mas é previsível, visto que o tamanho e a estrutura de dados são bem definidos além do protocolo de transmissão de dado. Assim, modelos de representação de barramentos mestre-escravo são mais facilmente definíveis.

Com as características dos níveis de abstração apresentados nesta Seção, é possível observar vantagens em descrever projetos de hardware ou mesmo modelos de comunicação em níveis de abstração superiores ao RTL. O uso de modelagem ao nível de transação no desenvolvimento de projetos de hardware está sendo ativamente empregado e pesquisado. Alguns trabalhos, tais como [18] e [16], foram desenvolvidos para analisar e compreender melhor as vantagens da descrição neste nível de modelagem.

Para o presente trabalho foram adotados níveis de abstração tanto para computação quanto para comunicação

baseados nos estudos realizados. O detalhamento dos níveis de abstração utilizados neste trabalho é vistos na seção 3.1.

3. TRABALHO EM ANDAMENTO

O presente trabalho tem por finalidade apresentar uma abordagem para descrição de SoCs em níveis de abstração superiores ao RTL. O resultado a ser obtido, a partir de refinamentos sucessivos da descrição realizada em nível de transação, deve ser competitivo com a descrição realizada em nível RTL, quanto a funcionalidade, custos de área e desempenho, o que justifica o presente trabalho. Para a modelagem do SoC é necessária a definição de um fluxo de projeto, conforme apresentado na Figura 3.

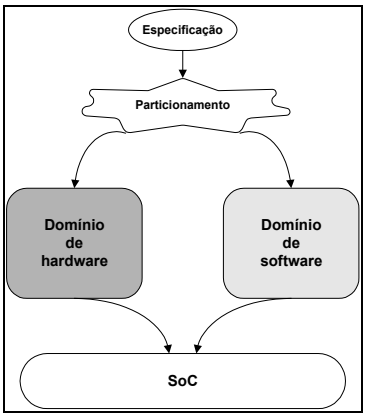

Figura 3 - Fluxo de projeto de um SoC adotado.

O nível topo do fluxo de projeto apresenta o processo de especificação do que se pretende implementar, e tendo como prioridade a criação de modelos de especificação executável. Tal modelo é particionado em domínios de hardware e software conforme análise de requisitos de desempenho e custo a serem atingidos no projeto. O particionamento está fora do contexto deste trabalho. No domínio de software, recursos como o sistema operacional embarcado que será utilizado, algoritmos parametrizáveis para atender um dado domínio de aplicação e *drivers* de interfaceamento de módulos de hardware podem ser descrito. O domínio de software também está fora do contexto deste trabalho. No domínio de hardware, módulos que descrevem a computação e a comunicação são implementados. Assim, microprocessadores, controladores de memória, barramentos e NoCs podem ser implementados. O domínio de hardware é o foco deste trabalho. O resultado final da junção tanto do domínio de hardware e software é o SoC.

3.1. Níveis de abstração adotados

Com base no fluxo de projeto apresentado, foram definidos níveis de abstração que representassem o detalhamento e o compromisso de cada modelo de descrição do sistema em cada fase. Assim, formam definidos os seguintes níveis de abstração, conforme Figura 4.

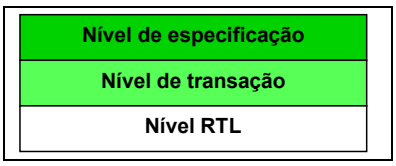

Figura 4 – Níveis de abstração adotados para o projeto.

No nível de especificação é implementada uma representação do que se pretende atingir com o sistema, sem a preocupação da separação em módulos de hardware ou temporização. Esta implementação denomina-se modelo de especificação executável. Basicamente o que se busca é a representação algorítmica do sistema. Também não são levadas em conta questões do que realmente deve ser descrito em hardware e o que necessita ser descrito em software. Neste nível a preocupação é implementar o algoritmo que descreva a solução do problema proposto da forma mais otimizada.

O particionamento da especificação executável em hardware e software caracteriza o fim do nível de especificação e o início da implementação de um domínio de hardware e um domínio de software. O nível de transação é o nível mais abstrato dentro do domínio de hardware. Este nível se caracteriza pela separação da computação e da comunicação de cada módulo de hardware. Neste nível a busca é pela eficiência na descrição de modelos de computação e de comunicação. Por não haver formalismo que defina o grau de detalhamento da funcionalidade do sistema e por possuírem características distintas, um conjunto de níveis de abstração é definido para modelos de computação e modelos de comunicação. Tais níveis foram definidos de acordo com a Figura 5 e serão melhor definidos nas seções 3.1.1 e 3.1.2.

O nível de transferência entre registradores é visto como o último nível de abstração do domínio de hardware, pois é neste nível que as ferramentas de síntese trabalham, conforme Jerraya [14]. Neste nível, a computação e a comunicação de cada módulos não é mais vista separadamente. Neste nível a computação é representada a partir de lógica combinacional e seqüencial e a comunicação é realizada a partir de sinais bem definidos assim como o protocolo de comunicação.

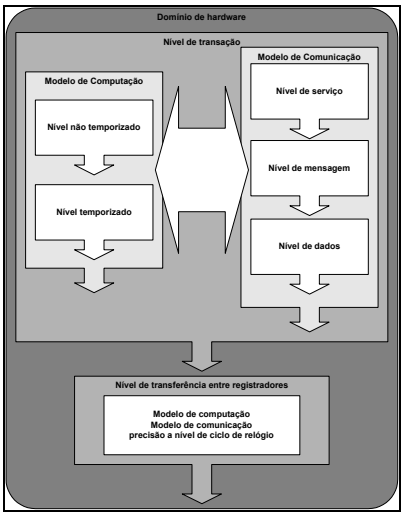

Figura 5 – Níveis de abstração no domínio de hardware. Nível topo representado pelo nível de transação e o nível logo abaixo representado pelo nível de transferência entre registradores.

A Tabela 1 apresenta a comparação com outras abordagens de níveis de abstração propostos por Cai [19], Arnout [12], Haverinen [17] e a proposta deste trabalho.

Tabela 1 - Comparação com outros níveis de abstração propostos.

	Separação entre computação e comunicação	Número de níveis superiores ao RTL	Níveis de abstração diferenciados para comunicação e computação
Lukai Cai	Sim	4	Não
Guido Arnout	Não	2	Não
Haverinen	Sim	3	Sim
Edson Moreno	Sim	5	Sim

Cai define 3 níveis de abstração para projetos de SoC, quais sejam: não temporizado, temporizado aproximado e temporizado em ciclos de relógio. Destes níveis de abstração, os dois primeiros são superiores ao RTL. Apesar de separar comunicação e computação, não há diferença entre os níveis de abstração adotados para cada um. Com esta abordagem a exploração da comunicação fica prejudicada em relação a computação.

Arnout define 2 níveis superiores ao RTL. Apesar disto, não há separação entre comunicação e computação, ou formalismo entre os níveis apresentados por ele. Com isto, a definição de metodologias de projeto se torna imprecisa, permitindo diversas abstrações para um mesmo nível de detalhamento.

Haverinen propõem 3 níveis de abstração para comunicação. Assim, há visão de separação entre comunicação e computação. Apesar desta proposta, não há preocupação com os níveis de abstração para descrição da computação, o que torna o fluxo de projeto incompleto.

No presente trabalho, há separação entre aspectos de computação e comunicação. São definidos 3 níveis de abstração para comunicação e 2 níveis de abstração para computação, superiores ao nível RTL. Com esta abordagem, características distintas de cada aspecto podem ser melhor descritas facilitando a criação de uma metodologia de projeto.

3.1.1. Níveis de abstração adotados para comunicação

Com base nos estudos realizados, os seguintes níveis de abstração para descrever comunicação nos projetos de SoCs são adotados, conforme Figura 6.

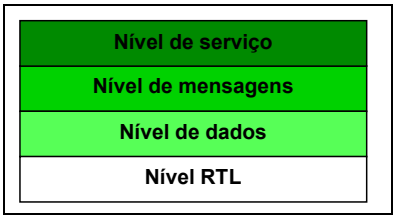

Figura 6 – Níveis de abstração da comunicação adotados

No nível de serviço, o meio de comunicação é abstraído, sendo o principal objetivo descrever as funções exercidas por cada módulo. Exemplos de serviço podem ser o de escrita em uma memória por parte de um processador, onde ao invés de definir um conjunto de sinais a serem ativados, o processador apenas faz uma chamada de um método escreveNaMemoria passando por parâmetro a posição da memória e o dado a ser armazenado. Tal método é definido em uma interface de forma abstrata e implementado pelo canal que interconecta a memória e o processador. Esta operação de escrita ocorre através da chamada de métodos da interface. Com esta abordagem, não há a necessidade de interpretação de uma cadeia de bits, apenas a requisição do serviço provido pelos núcleos IP. Este nível é equivalente ao nível de serviço proposto por Jerraya em [14] e prove maior abstração do que o nível de mensagem proposto por Haverinen et al. [17]. Questões como temporização e protocolo de comunicação são abstraídos em prol da funcionalidade da comunicação.

No nível de mensagem, não há precisão de tempo em ciclos de relógio. O objetivo é priorizar a funcionalidade comportamental da comunicação do que a precisão temporal. Apesar disto, os eventos de troca de dados ocorre de forma sincronizada. São descritas interfaces que definem os métodos para envio e recebimento de informações entre módulos, assim como no nível de serviço. As informações repassadas entre os módulos podem representar as operações a serem realizadas e o conjunto de dados adi-

cionais. Por exemplo, no caso de uma comunicação entre o processador e a memória, cria-se o que seria uma estrutura ou um pacote contendo o sinal da operação a ser realizada (escrita ou leitura), o endereço referente a posição de memória e dado a ser armazenado, caso seja uma operação de escrita. Esta definição do nível de mensagem apóia-se na proposta de Jerraya [14] e de Haverinen et al. [17], porém o principal foco é permitir um abordagem que facilite o refinamento para o próximo nível, tendo como foco a estrutura que será adotada para a troca de mensagens. O importante neste nível é definir quais as informações que terão de ser transferidas. Esta definição pode ser realizada tanto através do uso de pacotes de dados quanto através do uso de estruturas de dados bem definas. Pacotes de dados permitem a abstração do tamanho do dado apesar de criar complexidade, enquanto estruturas de dados já não permitem tanta flexibilidade porém facilitam a interpretação do dado. Todavia ambas abordagens focam nas informações que terão de ser trocadas, como o endereço, o dado, as respostas à requisição e a operação a ser realizadas.

No nível de dado, há precisão de ciclos de relógio, como no nível RTL. Todavia, o que o diferencia do nível RTL é a troca de dados a partir de tipos bem definidos além de mecanismos que descrevam protocolos específicos, como barramentos mestre-escravo, de modo funcional. Assim, canais de comunicação com a pinagem de endereçamento e operação, por exemplo, são definidos e o que resta é a implementação de como ocorre a comunicação internamente ao canal. Esta abordagem é muito próxima a proposta por Haverinen et al [17] no nível de transferência e por Jerraya [14] no nível de dados.

3.1.2. Níveis de abstração de computação adotados

Além dos níveis de comunicação, também foram adotados níveis de abstração superiores ao RTL para descrever os modelos de computação particionados após o modelo de especificação executável conforme Figura 7.

O nível topo de descrição de um módulo de computação é dado pelo modelo não temporizado. Neste nível, o que se descreve é a funcionalidade do módulo sem o uso de um sinal de sincronismo nem a idéia de inserção de atrasos temporais. Apesar da despreocupação com o sincronismo interno do módulo, a execução dos eventos do módulo ocorre de forma ordenada. Com esta abordagem podem ser validados parâmetros como a validade do modelo quanto a sua funcionalidade bem como o melhor algoritmo para descrever tal módulo.

No nível temporizado de descrição de um modelo computacional, há a preocupação com o sincronismo das operações realizadas assim como o conhecimento do atraso temporal de cada execução. Apesar disto, o nível temporizado se diferencia do nível RTL devido a abstração da comunicação e computação. Com tal nível de detalhamento, o comportamento do processo pode ser mapeado e parâmetros de atraso de processamento podem ser analisados de forma não precisa. Com esta abordagem é possível validar o desempenho do módulo e definir sua validade ou não de acordo com as especificações do sistema.

Figura 7 – Níveis de abstração para modelos de computação

3.1.3. Conclusões sobre níveis de abstração

O uso de níveis de abstração superiores ao de transferência entre registradores ainda é uma questão em aberto. Durante o desenvolvimento deste trabalho foram definidas características que facilitassem o desenvolvimento do trabalho. Dentre as principais vantagens que se busca com a elevação do nível de abstração estão a facilidade de modelagem do sistema, seu gerenciamento e principalmente a possibilidade de validação do sistema descrito logo nos primeiros estágios do projeto. Com esta abordagem, os domínios de hardware e software podem ser explorados paralelamente e validados de forma homogênea, diferentemente do processo ocorrido até então.

A definição do nível de transação é uma das principais características do projeto e tem por finalidade possibilitar mecanismos de refinamento, manuais ou automatizados, que diminuam o tempo de projeto de um sistema completo. Resumidamente, a modelagem de projetos a partir do nível de transação permite o encapsulamento de detalhes de implementação focando na chamada de funções que implementam as operações desejadas entre os modelos computacionais. Esta abordagem permite a descrição funcional da operação a ser realizada, o que permite vantagens importantes como o ganho de velocidade de simulação para a validação de um sistema. Como o sinal de sincronismo é abstraído no nível de transação, detalhes como comunicação bloqueante ou não bloqueante podem ser mais facilmente descritos o que facilita a descrição funcional de barramentos.

3.2. Linguagem de descrição de hardware adotada

Sendo previsível a necessidade de elevação do nível de abstração de captura de projetos, tal como a utilização do nível de transação, o uso de linguagens de descrição de hardware convencionais, tal como VHDL e Verilog, dificultam tal processo por não apresentarem mecanismos eficientes tais como os providos por linguagens que fazem uso de orientação a objetos. Assim, o uso de linguagens

de descrição de projetos em nível de sistema tem de ser adotadas.

Devido à exigência da indústria de semicondutores para maior produtividade de projetos de SoCs, é interessante que algumas características sejam alcançadas com novas linguagens de descrição de projetos em nível de sistema. A primeira é a facilidade com a qual os projetistas vão aprender a programar. Os subsídios para a descrição do projeto e a velocidade com que o resultado obtido vai ser apresentado são vistos como a segunda característica a ser alcançada.

Projetistas têm ou tiveram maior contato com linguagens de programação de mais alto nível de abstração tais como C/C++ e em menor número Java. Assim, a curva de aprendizagem para utilizar uma destas linguagens torna-se menor. Todavia, quanto à velocidade de simulação e os recursos disponibilizados pela linguagem C/C++ são superiores se comparados a Java. Java é uma linguagem interpretada enquanto C/C++ cria um arquivo executável, o que possibilita menor tempo de simulação. Quanto a recursos, C/C++ possui mais mecanismos se comparado a Java, tal como o uso de estruturas genéricas. Assim, C/C++ ganha por ser uma linguagem mais tradicional se comparada a Java, uma linguagem considerada nova conforme exposto Grötker por [20].

SystemC é uma biblioteca de extensão de C/C++ que facilita o processo de modelagem de hardware, por possuir características como tipos de dados próprios para definição de hardware(*sc_lv*), estruturas (*sc_module*) e processos(*sc_method*) que flexibilizam a descrição de paralelismo natural em hardware. Adicionalmente, SystemC permite a descrição em diferentes níveis de abstração propostos. Além de SystemC, a *xilinx* disponibiliza o forge, uma ferramenta de conversão de Java para verilog a partir de níveis de abstração superiores ao de transferência entre registradores. Apesar disto, SystemC foi escolhido como a linguagem a ser adotada dois motivos. O primeiro motivo é o de atender as necessidades previamente expostas, tal como menor curva de aprendizagem, maior velocidade de simulação e mairo confiabilidade do projeto descrito. O segundo motivo da adoção de SystemC foi o suporte a linguagem. Sendo vista com a mais nova tendência de padronização para descrição de hardware, SystemC possui uma lista mundial de suporte além de contar com grandes empresas de microeletrônica para desenvolvimento de ferramentas que facilitem o trabalho, tais como as ferramentas da *Synopsys Inc.* e da *CoWare Inc.*.

3.2.1. Comparando SystemC e VHDL

O critério para adoção de SystemC passou por sua comparação com VHDL, a linguagem de descrição de hardware adotada pelo grupo. Basicamente foram definidos 2 critérios de validação. O primeiro foi o de tempo de simulação e o segundo o de tamanho do hardware gerado. Para tanto foram refeitas descrições de módulos de hardware previamente projetados pelo grupo. Tais descrições forma então feitas em nível transação e posteriormente descritas em nível de transferência entre registradores [21].

O módulo de hardware descrito em SystemC foi o processador multiciclo R8. Este processador apresenta instruções e dados de 16 bits cada [22]. Há apenas alguns modos de endereçamento. Este processador é praticamente uma máquina RISC, faltando para tanto algumas características como o uso de *pipelines*. As principais características organizacionais deste processador multiciclo são:

- Endereçamento e dados em 16 bits;
- Banco de registradores com 16 registradores de propósitos gerais de 16 bits;
- Quatro flags de status de operações: negativo, zero, carry, e overflow;
- Execução de instruções variável de 2 a 4 ciclos de clock.

Com base na especificação do processador R8, este foi descrito em nível de transação, onde a computação foi descrita de forma não temporizada e a comunicação entre os módulos em nível de serviço. Assim, foram implementadas interfaces definindo os métodos de comunicação e canais, implementado-os conforme a estrutura mostrada na

Figura 8.

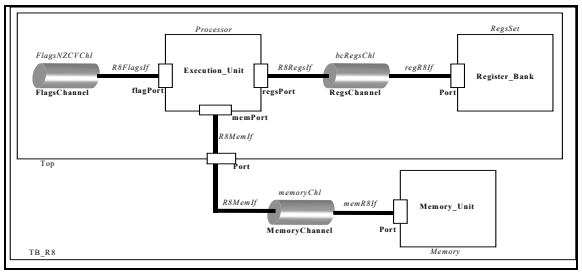

Figura 8 – Processador R8 descrito em nível de transação

Foram definidos 3 módulos e 3 canais, conforme visto na

Figura 8 para a implementação em SystemC. Nos três módulos foram descritos o processador, a banco de registradores e a memória. Os canais implementam o mecanismo de comunicação do processador com o banco de registradores, o mecanismo de comunicação do processador com a memória e a comunicação do processador com os flags de negativo, zero, carry e overflow. Para cada canal foram definidas 2 interfaces. Para o canal de comunicação do processador e memória, por exemplo, o registrador PC ficou foi implementado dentro do canal. O acesso à memória ocorre a partir de chamadas de métodos de interface. Assim, sempre que o processador quer fazer uma leitura ou escrita na memória, o método readMem ou

writeMem são chamados e resolvidos pelo canal, respectivamente.

Para comparação do tempo de simulação, foi gerado um algoritmo de *bubble sort* e variou-se o tamanho da memória de valores a serem ordenados. Os resultados obtidos com o tempo de simulação podem ser vistos na Figura 8..

A velocidade de simulação utilizando níveis de abstração superiores ao RTL e SystemC foi superior ao mesmo exemplo executado em RTL utilizando VHDL em aproximadamente uma ordem de magnitude. Este ganho deve-se basicamente a duas diferenças. A primeira é o tipo de execução. SystemC prove um modelo executável enquanto VHDL prove uma descrição que tem de ser interpretada por uma ferramenta como o *ModelSim* da Mentor Graphics. A segunda diferença é a quantidade de detalhamento proposto em cada uma das linguagens.

A segunda comparação a ser realizada entre as linguagens foi o de área ocupada no FPGA pelo projeto descrito em nível de transferência entre registradores. O resultado obtido pode ser visto na Figura 10.

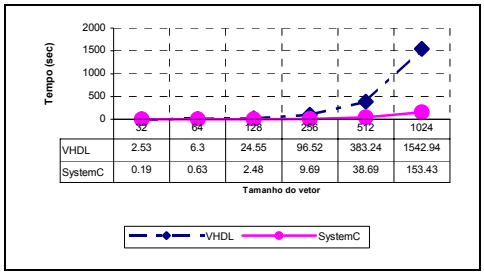

Figura 9- Comparação do tempo de simulação entre SystemC e VHDL

Esta comparação permitiu avaliar que SystemC é uma linguagem que permite ganhos de ocupação de área de projeto equivalentes as descrições realizadas em VHDL. Apesar desta equivalência, a dificuldade de descrever um módulo sintetizável é grande pois nem todas as operações descritas são passíveis de tradução para VHDL por parte do compilador que realiza esta operação. Por isso um subconjunto de tipos de dados e operações é disponibilizado. Outro problema desta ferramenta de tradução do módulo SystemC para HDLs é que o resultado de área obtido nem sempre é o mais otimizado, podendo gerar hardware com tamanhos muito maiores dependendo da forma como é descrito em SystemC.

Com base nas comparações obtidas, e na tendência de padronização de SystemC como linguagem de descrição de projetos em nível de sistema, o restante do trabalho foi todo baseado na utilização desta linguagem. Assim, ferramentas de modelagem, descrição, validação por simulação, conversão de SystemC para HDL e síntese lógica foram utilizadas. Tais ferramentas foram disponibilizadas a partir de convênio universitário com a *Synopsys Inc*.

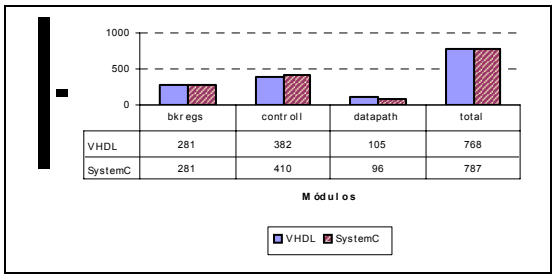

Figura 10 - Comparação do tamanho obtido com a descrição VHDL e SystemC em LUTs.

3.2.2. Descrevendo SystemC RTL sintetizável

SystemC possui um conjunto bem definido de regras para gerar uma descrição sintetizáveis [23]. Apesar disto, nem todas as regras primam por questões de melhor descrição de hardware com relação a tamanho ou validade do que se descreve. Durante a descrição do trabalho descrito na seção 3.2.1. algumas abordagens foram adotadas para tanto.

O uso de um vetor de tamanho fixo de um determinado módulo não é uma prática aconselhável. Quando esta descrição passa pelo compilador SystemC para gerar o VHDL, o resultado é a tradução literal mais o número de módulos definido pelo tamanho do vetor. Exemplo deste problema pode ser dado na descrição do banco de registradores. São usados 16 registradores de propósito geral, os quais eram definidos segundo um vetor de registradores de tamanho 16. O resultado foi a definição de um vetor de registradores de tamanho 16 mais 16 registradores soltos. A solução para este caso foi o de descrever 16 registradores eliminando a definição do vetor.

Outro ponto importante é na descrição de um processo. SystemC flexibiliza descrições utilizando tipos tais como *logic vector* e sua conversão para valores tais como *integer* e *unsigned* facilmente. O uso destes métodos sem a definição de variáveis locais para armazenar o resultado acarreta em atribuição de variáveis novas para cada conversão realizada dentro de um processo por parte do compilador SystemC ao VHDL gerado. Este problema pode ser facilmente contornado quando variáveis locais são definidas e compartilhadas.

O uso de tipos booleanos é interessante na descrição de hardware. O primeiro ponto interessante é o de facilitar a descrição de processos sensíveis a borda de subida ou descida de um sinal. Apesar disto, quando utilizada dentro de um processo, a comparação a ser realizada tem de ser explícita ou o resultado em VHDL pode não ser satisfatório. Exemplo disto é dado para uma situação de *reset*. No processo foi implementado *if(reset.read()){...}* o que re-

sultou em um VHDL com um sinal reset do tipo *std_logic* e a seguinte linha *if(reset)then* .... Durante a fase de simulação do VHDL gerado, embora o sinal de reset estivesse sendo realmente alterado, a condição de reset nunca ocorria. Ao substituir a descrição em SystemC por *if(reset.read()==true){...}*, o resultado obtido em VHDL foi *if(reset='1')then*.... Com tal mudança a simulação do VHDL foi realizada de foram correta. Com base nesta observação, notou-se a necessidade de explicitar todas comparações.

4. CONCLUSÕES

O estudo de SystemC e suas vantagens formam analisadas e comparadas a VHDL. Concluiu-se que SystemC tem vantagens quanto à velocidade de simulação e é competitivo quanto a tamanho de hardware gerado. Apesar destas vantagens, um conjunto de regras para a descrição de hardware em SystemC para a geração de um hardware sintetizável e de tamanho otimizado, está sendo proposto.

A revisão e definição dos níveis de abstração superiores ao RTL são vistas como uma contribuição deste trabalho. Não havendo uma definição formal para tais níveis, a definição destes, juntamente com um fluxo de projeto podem ser utilizados como metodologia de projetos de SoCs onde a separação de comunicação e computação são utilizados.

5. REFERÊNCIAS BIBLIOGRÁFICAS

[1] Micheli, G. D.; Ernst, R. and Wolf, W. **Readings in Hardware/Software Co-Design**. Morgan Kaufmann, San Francisco, CA, 2001, ed., 697.

[2] Edwards, S.; Lavagno, L.; Lee, E. A. and Vincentelli, A. S. **Design of Embedded Systems: Formal Models, Validation and Synthesis**. In: Proceedings of the IEEE, 1997, pp 366-390.

[3] Gajski, D.; Vahid, F.; Narayan, S. and Gong, J. **Specification and Design of Embedded Systems**. Prentice Hall, Englewood Cliffs, NJ, 1994, ed., 450.

[4] Schaller, R. R. **Moore's law: past, present and future**. IEEE Spectrum, vol. 34-6, 1997, pp. 52-59

[5] Weste, N. H. E. **Principles of CMOS VLSI design: a systems perspective**. Addison-Wesley, Reading, MA, 1994, 2nd ed. ed., 713.

[6] Rajsuman, R. **System-on-a-Chip: Design and Test**. Artech House, Santa Clara, CA, 2000, 1st ed. ed., 277.

[7] Martin, G. and Chang, H. **Tutorial - System on Chip Design**. In: 9th International Symposium on Integrated Circuits, Devices & Systems - ISCAS, 2001, 6 p.

[8] Zhu, Q.; Matsuda, A.; Kuwamura, S.; Nakata, T. and Shoji, M. **An object-oriented design process for system-on-chip using UML**. In: 15th International Symposium on System Synthesis - ISSS, 2002, pp. 249-254.

[9] **International Techology Roadmap for Semiconductors**. Captured at http://public.itrs.net, 2001.

[10] Rauscher, T. G. **Time to market problems - The organization is the real cause**. In: Proceedings of the IEEE International Engineering Management Conference, 1994, pp. 338-345.

[11] Micheli, G. D. and Benini, L. **Network on Chips: A New SoC Paradigm**. IEEE Computer, vol. 35-1, 2002, pp. 70-78

[12] Arnout, G. **EDA moving up, Again!** In: 6th European SystemC Users Group, 2002, 6 p.

[13] Calazans, N. L. V. **Projeto Lógico Automatizado de Sistemas Digitais Seqüenciais**. Imprinta Gráfica e Editora Ltda, 11ª Escola de Computação, Rio de Janeiro, 1998, 1ST ed., 346 P.

[14] Jerraya, A. A.; Svarstad, K.; Ben-Fredj, N. and Nicolescu, G. **A higher level system communication model for object-oriented specification and design of embedded systems**. In: Conference on Asia South Pacific Design Automation Conference - ASPDAC, 2001, pp. 69-77.

[15] **CoCentric System Studio Enables Verification at Multiple Levels of Abstraction with SystemC**. Captured at http://www.synopsys.com/products/cocentric_studio/cocentric_studi_wp.pdf, Jan. 2002.

[16] Pasricha, S. **Transaction level modeling of SoC with SystemC 2.0**. In: Synopsys Users Group Conference - SNUG, 2002, 5 p.

[17] Haverinen, A.; Leclercq, M.; Weyrich, N. and Wingard, D. **SystemC™ based SoC Communication Modeling for OCP™ Protocol**. White paper, Jul. 2002, 39 p., available at: http://www.ocpip.org/data/ocpip_wp_SystemC_Communication_Modeling_2002.pdf.

[18] Moreno, E. **Modelagem, descrição e validação de hardware em diferentes níveis de abstração usando SystemC**. Trabalho de individual II, PPGCC - FACIN - PUCRS, Nov. 2002, 33 p., available at: http://www.inf.pucrs.br/~gaph/documents/emoreno_ti_2.pdf.

[19] Cai, L. and Gajski, D. **Transaction Level Modeling in System Level Design**. Technical report, University of California, 2003, 23 p, available at: http://www.ics.uci.edu/~cecs/technical_report/TR03-10.pdf.

[20] Grötker, T.; Liao, S.; Martin, G. and Swan, S. **System Design with SystemC**. Kluwer Academic, Boston, 2002, 1st ed., 219 p.

[21] Calazans, N. L. V.; Moreno, E.; Hessel, F.; Rosa, V. and Moraes, F. G. **From VHDL Register Transfer Level to SystemC Transaction Level Modeling: a Comparative Case Study**. In: SBCCI'03, 2003, 6 p.

[22] Moraes, F. G. and Calazans, N. L. V. **R8 Processor Specification and Design Guidelines**. PPGCC - FACIN - PUCR, 2003, available at: http://www.inf.pucrs.br/~gaph.

[23] **Describing synthezsizable RTL in SystemC™**. Captured at www.synopsys.com, 2003.