SANDRO RENATO DIAS

SRD: UMA FERRAMENTA DE APOIO AO PROJETISTA DE SISTEMAS DE HARDWARE UTILIZANDO A LINGUAGEM SYSTEMC

Trabalho apresentado ao Programa de Mestrado em Engenharia Elétrica como requisito parcial à obtenção do grau de Mestre em Engenharia Elétrica, área de conhecimento Engenharia da Computação.

BELO HORIZONTE, 2007

SANDRO RENATO DIAS

SRD: UMA FERRAMENTA DE APOIO AO PROJETISTA DE SISTEMAS DE HARDWARE UTILIZANDO A LINGUAGEM SYSTEMC

Este trabalho foi julgado adequado à obtenção do grau de Mestre em Engenharia Elétrica e aprovado em sua forma final pelo Curso de Mestrado em Engenharia Elétrica da Universidade Federal de Minas Gerais.

Belo Horizonte, 01 de março de 2007.

Prof. PhD. Diógenes Cecílio da Silva Júnior

Departamento de Engenharia Elétrica - Universidade Federal de Minas Gerais

---

Prof. Dr. Guilherme Augusto Silva Pereira

Departamento de Engenharia Elétrica - Universidade Federal de Minas Gerais

---

Prof. Dr. Antônio Otávio Fernandes

Departamento de Ciência da Computação - Universidade Federal de Minas Gerais

Dedico este trabalho ao meu filho Gabriel.

AGRADECIMENTOS

Aos meus pais, sempre. Ao meu filho Gabriel pelo carinho e por entender que o papai tinha que trabalhar no computador o tempo todo. À minha querida Letícia pela força e compreensão. Aos que colaboraram direta e indiretamente, em particular meus alunos e amigos Túlio Teixeira Cota, André Luiz B. P. e Maximiliano Mitchel M. Ramos, meus colegas do mestrado e do grupo de pesquisa Adriano, Alair, Rafael, Ramon e Fernando, aos doutores John Aynsley, Frank Oppenheimer e ao meu orientador, Prof. PhD Diógenes Cecílio da Silva Júnior. Não posso me esquecer do meu notebook, fiel companheiro das madrugadas, dos finais de semana, salas de espera, salas de aula, poltronas de ônibus e qualquer outro local que tenha sido usado.

"É melhor tentar e falhar, que preocupar-se e ver a vida passar. É melhor tentar, ainda que em vão, que sentar-se fazendo nada até o final. Eu prefiro na chuva caminhar, que em dias tristes em casa me esconder. Prefiro ser feliz, embora louco, que em conformidade viver."
Martin Luther King

RESUMO

O projeto de sistemas de hardware tem migrado de linguagens puras de descrição de hardware, como Verilog e VHDL, para linguagens de alto nível baseadas em C ou C++, visando obter um maior nível de abstração mantendo-se a semântica de uma linguagem de descrição de hardware. Dessa forma, o projetista pode se aprofundar em pontos específicos do projeto, no momento necessário, enquanto mantém outros pontos em outros níveis de abstração.

A Metodologia de Refinamentos Sucessivos permite esta melhoria progressiva no código do projeto e o uso da linguagem SystemC possibilita a aplicação desta metodologia. Este trabalho apresenta um estudo desta e de outras metodologias, além de ferramentas e linguagens de projeto de hardware. Diversos níveis de abstração foram propostos na literatura e neste trabalho é apresentada uma proposta de modelo de níveis para a aplicação da Metodologia de Refinamentos Sucessivos com a definição das regras necessárias para a passagem de um nível para outro.

O presente trabalho, além da proposição dos níveis de refinamento, ainda apresenta uma ferramenta desenvolvida com o objetivo de facilitar o projeto de sistemas de hardware. A ferramenta, denominada SRD (*Successive Refinement Design*) além de permitir o projeto do sistema através de um diagrama gráfico, ainda facilita o refinamento orientando e executando, em alguns casos, os passos do refinamento para cada nível. A ferramenta gera todos os arquivos de código do projeto na linguagem SystemC, de forma que possa ser simulado e/ou sintetizado.

Palavras-chave: Refinamento Sucessivo, Projeto de Hardware, SystemC, SoC, IP.

ABSTRACT

The hardware systems design has moved from pure hardware description languages (HDL), like Verilog and VHDL, to high level languages based on C/C++, aiming at a higher abstraction level mantaining the HDL semantics. Thus, the designer can change specific points in the design leaving other points at other abstraction levels.

The successive refinement methodology allows this progressive improvement in the design code and the use of SystemC language allows the application of this methodology. This work presents a methodology and compares with similar ones. Also, presents tools and hardware design languages. Several abstraction levels were proposed in the literature and in this work proposes a level model with four levels of abstraction to apply the Successive Refinement Methodology with rules definitions necessary to move between levels.

The present work also presents a tool designed with the objective to ease the hardware systems design. This tool, called SRD (Successive Refinement Design), allows the system designer to use graphical diagrams, guiding through the refinement steps, and in some cases executing the refinement steps. The tool generates all code files of the project in SystemC language, such that, the designer can simulates and/or sinthesizes the design.

Keywords: Successive Refinement, Hardware Design, SystemC, SoC, IP.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 Introdução

O projeto de sistemas de hardware tem migrado de linguagens puras de descrição de hardware, como Verilog e VHDL, para linguagens de uso geral como C++. Esta migração visa obter um maior nível de abstração, permitindo ao projetista aprofundar em pontos específicos do projeto, no momento necessário, enquanto mantém outros pontos no mesmo nível de abstração, adiando seu refinamento.

A abstração é uma técnica poderosa para o projeto e a implementação em hardware de sistemas complexos. Ela nos permite enfrentar a complexidade escondendo detalhes desnecessários para poder lidar com os mesmos mais tarde. Uma ferramenta que possibilite o uso da abstração para projeto de hardware facilita o trabalho do projetista. Já uma ferramenta que auxilie as etapas do refinamento, pode agilizar ainda mais o fluxo do projeto orientando e direcionando o projetista nos pontos onde se deve atuar para efetuar o refinamento correto.

Este trabalho visa propor um modelo de níveis de abstração para o uso da Metodologia de Refinamentos Sucessivos bem como apresentar uma ferramenta que auxilie o projetista no desenvolvimento de sistemas de hardware utilizando tal metodologia.

1.1 Metodologia de Refinamentos Sucessivos

É bem conhecido o fato de que projetistas de SoCs estão deparando com um gap de produtividade crescente entre tecnologia semicondutora, metodologias e ferramentas de suporte a projeto de hardware [10]. Muitos esforços têm sido focados numa maior abstração

dos níveis de modelagem dos projetos. Com níveis de abstração maiores, o número de objetos no projeto cai exponencialmente. Isto permite que o projetista e que as ferramentas possam concentrar-se em aspectos críticos e explorar uma grande parte do projeto sem se preocupar com detalhes desnecessários naquele momento. Ferramentas irão ajudar o projetista a gradualmente refinar o projeto em níveis cada vez mais baixos de abstração, em direção à síntese.

A metodologia *top-down* de projeto que trabalha com níveis de abstração diferentes e mudanças contínuas no projeto em direção à síntese é chamada de Metodologia de Refinamentos Sucessivos. Um requisito para qualquer fluxo de projeto é um conjunto de níveis de abstração bem definidos. O número de níveis e suas propriedades devem ser claramente definidos para que os projetistas possam otimizar decisões e mover-se entre níveis eficientemente. O objetivo é diminuir o número de objetos a serem manipulados com o aumento do nível de abstração enquanto se provê detalhes suficientes em cada etapa do refinamento, levando a um momento de decisão entre precisão e eficiência contra velocidade de simulação. Além disso, uma definição clara e livre de ambigüidade desses níveis é necessária para possibilitar a automação do projeto para a síntese e verificação.

O fluxo clássico do projeto de circuitos integrados, Figura 1, não vislumbra o refinamento sucessivo e muito menos a possibilidade de co-desenvolvimento. O desenvolvimento do hardware e do software se dá de forma independente e sem comunicação, além disso, o software só será finalizado após a conclusão do hardware [90]. Geralmente, as linguagens utilizadas pelos projetistas de hardware são VHDL ou Verilog HL enquanto que os projetistas de software usam linguagens baseadas em C/C++, Java, dentre outras. No uso de uma metodologia de refinamentos sucessivos, o desenvolvimento do software pode se iniciar a partir do modelo de referência, que é simulável e, portanto, passível de teste do software. Ao longo do desenvolvimento/refinamento do projeto, o software pode ser validado sobre a arquitetura ainda não finalizada como um protótipo, mas completamente funcional. Isto permite que o projeto tenha uma redução em seu tempo total.

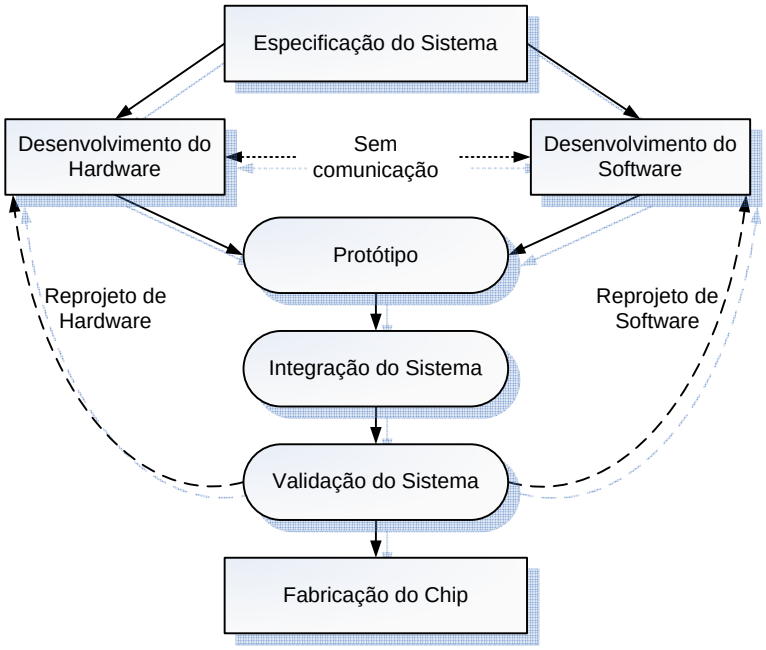

Figura 1 – Fluxo clássico de projeto de circuitos integrados, traduzida de [90].

1.2 SystemC

SystemC é largamente suportado por um grande e crescente número de *system houses*, empresas produtoras de semicondutores, provedores de IPs, de sistemas embutidos e de ferramentas de EDA (*Electronic Design Automation*), através do *Open SystemC Initiative* (OSCI), que visa à difusão da linguagem e de sua metodologia de projeto.

As características da linguagem a tornam capaz de suportar o reúso, o refinamento sucessivo, o co-desenvolvimento hardware-software, além de poder ser utilizada em qualquer fase do projeto, permitindo, inclusive que o projeto em SystemC seja sintetizado (fase final), o que só era possível através de traduções da linguagem utilizada no desenvolvimento para uma linguagem como VHDL ou Verilog HDL.

Migrar de VHDL, Verilog HDL ou qualquer outra linguagem de descrição de baixo nível para SystemC não é tarefa fácil. Projetistas com domínio destas linguagens já estão familiarizados com suas características e fluxos de projeto. O fluxo normal de projeto com estas linguagens parte de uma modelagem em outra linguagem de alto nível, que é traduzida para VHDL ou

Verilog HDL. Esta etapa de tradução adiciona muitos erros ao projeto devido à interpretação incorreta ou a falhas do projeto. Usar SystemC implica em iniciar o projeto com esta linguagem e refiná-lo até que chegue ao mais baixo nível, permitindo-se assim, sua síntese, sem que se tenha que utilizar outra linguagem durante qualquer fase do projeto.

Existem no mercado diversas ferramentas que permitem o uso de SystemC para o projeto de hardware, porém das ferramentas aqui estudadas, nenhuma suporta o refinamento sucessivo. Assim a ferramenta aqui desenvolvida possibilita o uso de SystemC juntamente com a metodologia de refinamentos sucessivos, orientando o projetista durante o processo de refinamento.

1.3 Motivação

Os avanços na indústria de fabricação de circuitos integrados na área de nanotecnologia permitem a evolução dos simples circuitos integrados em ASICs (*Application Specific Integrated Circuit*), IPs (*Intellectual Property cores*), SoCs (*System-on-Chip*), NoCs (*Network-on-chip*), sistemas embutidos. Um SoC é usado tipicamente num pequeno, porém complexo produto eletrônico de consumo, como um telefone sem fio ou uma câmera digital. Os SoCs são construídos com os blocos de Propriedade Intelectual (IP), que são blocos de hardware reusáveis, projetados para realizar uma determinada tarefa. Esses blocos de IP podem ser processadores, memória, dispositivos de entrada/saída, dispositivo de rádio freqüência, timer, dentre outros. Os diferentes blocos são interconectados através de barramentos compartilhados ou de redes (NoC). Os ASICs são circuitos integrados desenvolvidos para uma tarefa específica, com um grau de complexidade menor que um SoC, pois não são sistemas completos. Além disso, ASICs não são programáveis, enquanto os SoCs o são.

Uma das motivações para o desenvolvimento deste trabalho é a possibilidade de redução do *time-to-market* dos sistemas embutidos. *Time-to-market* é o tempo que um projeto gasta para estar disponível no mercado. Este tempo tem se reduzido de forma muito rápida, dada a evolução dos sistemas. Se tomarmos como exemplo que a TV em preto e branco demorou 17 anos para vender um milhão de unidades, a TV em cores demorou 10 anos, o computador

pessoal gastou 6 anos, o celular 4 anos e o aparelho de DVD em um ano alcançou esta meta, podemos ter uma idéia da necessidade de redução do *time-to-market* no projeto de sistemas embutidos.

O aumento da produtividade com a redução do tempo de projeto é outra motivação. O desenvolvimento de blocos de IPs e o seu posterior reúso em outros projetos favorece este aumento de produtividade eliminando o tempo do projeto que seria utilizado para o desenvolvimento do bloco reusado. Além disso, o tempo total do projeto também é reduzido com a possibilidade de desenvolvimento simultâneo de hardware e software, permitindo que o software que será executado sobre o hardware final esteja pronto antes mesmo que o protótipo físico esteja finalizado. Isto é possível com o uso de linguagens que permitam o desenvolvimento e simulação do hardware e do software concomitantemente, geralmente, linguagens baseadas em C ou C++, como SystemC por exemplo, facilitam este procedimento.

Mais uma motivação é a redução do tempo do *First-time Silicon Success* (FTSS). Este tempo refere-se ao primeiro protótipo físico com funcionamento totalmente correto. A facilidade de refinamento do código e sua análise e verificação em tempo de projeto reduzem as possibilidades de erro, permitindo que se chegue ao estágio final de desenvolvimento com uma quantidade muito menor de erros de funcionamento. Um exemplo disso é o projeto BrazilIP[1] que, a partir de IPs desenvolvidos no Brasil utilizando uma metodologia que visa o refinamento e a verificação constantes, recebeu o *Best IP/SoC 2006 Design Award*[2], na Conferência IP/SoC 2006 realizada em Grenoble, França. Dentre os IPs desenvolvidos estão um microcontrolador 8051 e um decodificador MPEG4.

A grande maioria das ferramentas de apoio ao projeto de hardware são ferramentas proprietárias de alto custo. Além disso, ferramentas que trabalham com a linguagem SystemC são um número pequeno, porém crescente. Esta é outra motivação para o desenvolvimento de uma ferramenta que permita o uso da linguagem e da metodologia de refinamento sucessivo, facilitando o trabalho do projetista. Não foram encontradas ferramentas que dessem suporte ao refinamento sucessivo.

[1] Rede de pesquisa que envolve 8 universidades brasileiras, dentre elas a UFMG, com o principal objetivo de desenvolver IPs e metodologia de desenvolvimento de projetos.

[2] Matéria sobre a premiação disponível em <http://www.eetimes.eu/design/196602527>, acessado em 08/12/2006.

Por fim, o crescente uso da linguagem SystemC é outra motivação. Esta linguagem permite o refinamento sucessivo e simulação, co-desenvolvimento hardware-software, além de diversas características úteis ao projetista de hardware, porém, estas características não são usadas e/ou são pouco conhecidas, conforme pesquisas independentes realizadas entre desenvolvedores nos anos de 2003 [97] e 2005 [98] [99] [100]. Apesar disso, muitos desenvolvedores escolheram SystemC como a linguagem a ser usada em seus próximos projetos, independente de estar utilizando-a nos projetos em desenvolvimento. Dentre os usuários da linguagem o nível de insatisfação é bem pequeno (média em torno de 14%) em quatro itens que são facilidade de modelagem, análise de performance, implementação e *testbenches*. Soma-se a isso a utilização comum de uma linguagem diferente em cada fase do projeto, ou seja, uma linguagem para a especificação, outra para a modelagem, o desenvolvimento do projeto, o desenvolvimento do software, outra linguagem para a síntese e mais uma para a verificação. Com o SystemC é possível se usar a mesma linguagem em todas as fases de desenvolvimento.

1.4 Objetivo

Tendo em vista as motivações descritas anteriormente, este trabalho visa contribuir com o projeto de sistemas de hardware, através do uso da linguagem SystemC e da metodologia de refinamentos sucessivos. Dessa forma, objetiva o desenvolvimento de uma ferramenta para apoio ao projetista no uso da metodologia de projeto. Para tanto, faz-se necessário:

- Definição clara dos níveis – é preciso definir os níveis a serem utilizados no refinamento sucessivo e seus escopos, visando uma maior clareza no desenvolvimento e na definição de cada estágio do projeto.

- Estimulo do uso da metodologia – apesar de adotar a linguagem SystemC, o projetista/desenvolvedor/engenheiro pode não conhecer ou utilizar toda a sua potencialidade. O uso da metodologia de refinamentos sucessivos permite que a linguagem SystemC seja utilizada em todo o projeto desde o nível mais alto (definição da especificação) até a síntese passando pelo nível RTL (*Register Transfer Level*, nível mais baixo de descrição de hardware).

- Auxílio do projetista na aplicação da metodologia – com a definição dos níveis, guiar o projetista no refinamento entre os níveis, esclarecendo cada passo de refinamento e a forma como deve ser feito.

1.5 Organização deste documento

A definição da necessidade de uma ferramenta, para aplicar a metodologia de refinamentos sucessivos dando o devido suporte ao projetista de sistemas embutidos, deve ser baseada numa pesquisa de metodologias, linguagens e ferramentas similares disponíveis. A pesquisa, não exaustiva, sobre os trabalhos relacionados, que envolve algumas metodologias para projetos de hardware, linguagens de descrição de hardware mais utilizadas além de ferramentas de projeto, simulação e síntese, encontra-se no capítulo 2.

O capítulo 3 aborda a linguagem escolhida para servir de base para a ferramenta desenvolvida. Sua justificativa de uso e motivação, bem como suas características mais importantes estão neste capítulo.

A Metodologia de Refinamentos Sucessivos e uma proposta de níveis de refinamento, para sua aplicação, são detalhadas no capítulo 4, onde um exemplo de projeto é utilizado para ilustrar o uso da metodologia e dos níveis propostos. Para isto utilizou-se o projeto de um sistema para geração do GCD (*Greatest Common Divisor*), ou máximo divisor comum, de dois números. Seu código é refinado ao longo do capítulo.

A ferramenta implementada neste trabalho, seu desenvolvimento, as decisões de projeto, necessidades específicas, características, funcionamento, dentre outros, se encontram no capítulo 5.

O capítulo 6 traz as considerações finais deste trabalho levando em conta a metodologia, a linguagem e a ferramenta desenvolvida. Logo em seguida encontram-se as referências consultadas e, posteriormente os anexos, onde temos: Anexo A que apresenta o conjunto de estruturas não sintetizáveis da linguagem SystemC; Anexo B que apresenta o conjunto não sintetizável das linguagens C e C++ para síntese usando SystemC; Anexo C que apresenta o

conjunto sintetizável da linguagem SystemC; Anexo D que apresenta os códigos utilizados no desenvolvimento do sistema GCD no primeiro nível de abstração; Anexo E com os códigos refinados para o segundo nível; Anexo F para o terceiro nível e por fim o Anexo G apresentando o código no último nível de refinamento, o RTL.

2 Trabalhos Relacionados

Com o crescimento da linguagem SystemC e sua confirmação como um padrão de fato para descrição de sistemas embutidos complexos, várias ferramentas foram propostas para auxiliar o projetista em sua tarefa de descrição desses sistemas. Esta descrição vai do nível do hardware (mais baixo nível) ao nível do sistema (mais alto nível), percorrendo os detalhes intrínsecos da arquitetura e de cada nível específico. Dessa forma, muitas ferramentas apresentaram soluções para problemas específicos de um determinado nível ou modelagem, por outro lado, outras linguagens também foram desenvolvidas com o mesmo objetivo de SystemC, e até mesmo a partir dessa linguagem.

Este capítulo visa abordar algumas metodologias de projeto de sistemas embutidos, algumas linguagens e ferramentas utilizadas para isso. Visa ainda, fazer o comparativo entre essas opções, analisando uma maior facilidade para o projetista.

2.1 Metodologias

Os projetos de sistemas embutidos demandam novas metodologias, linguagens e ferramentas diferenciadas à medida que a complexidade desses sistemas aumenta. O uso de uma metodologia específica pode ajudar no desenvolvimento do SoC, agilizando o seu processo de projeto. Diversas metodologias têm sido desenvolvidas com o objetivo de facilitar o projeto de circuitos, dentre elas destaca-se a Metodologia de Refinamentos Sucessivos, utilizada neste trabalho.

Esta seção visa abordar de uma forma geral algumas das metodologias existentes, sendo que a Metodologia de Refinamentos Sucessivos será mais bem detalhada no Capítulo 4.

2.1.1 Orientada ao projetista

A metodologia orientada ao projetista (*Designer Oriented Methodology – DO*) proposta por Silveira e Van Noije [2] consiste num ambiente capaz de oferecer ao usuário/projetista condições apropriadas para que o mesmo possa trabalhar livre de interrupções e distrações, mantendo-o sempre atento e em prontidão para resolver qualquer problema que venha a ocorrer assim que o mesmo seja percebido. Isto é possível a partir da apresentação de todas as informações ao usuário reforçando sua atenção ao projeto e seus aspectos específicos, assim como um jogo que captura as atenções dos jogadores durante o seu curso.

Os conceitos e abstrações empregados nesta metodologia são chamados de *Game Like Development (GLD)* e a ferramenta gerada pelos autores foi implementada utilizando-se a linguagem Self, orientada a objeto. Silveira e Van Noije propõem a criação de um *framework* a partir da integração das ferramentas de simulação, síntese e verificação além de muitas outras de forma que o *framework* possa evoluir ao longo do tempo de acordo com as necessidades do projeto.

2.1.2 Projeto Hierárquico

Gullapalli e Shi [1] descrevem um modelo de metodologia de projeto hierárquico, consistindo na divisão de um projeto em múltiplos blocos (também chamados de sub-blocos, sub-chips, blocos de baixo nível, módulos, blocos hierárquicos, dentre outros). Projetistas podem trabalhar em blocos separadamente em direção à implementação física. Trabalhando com pequenos blocos obtém-se um tempo de execução menor e maior proximidade em termos da precisão de tempo, o que seria mais difícil de se alcançar ao se compilar todo o projeto (todo o chip) numa metodologia de projeto num único módulo (*flat design*). A partir do momento em que todos os blocos estão finalizados, eles são integrados para criar a implementação final do chip, onde os blocos são modelados como caixas-pretas, o que pode incluir precisão de tempo,

teste, interfaces e informações de entrada e saída. A Figura 2 mostra um fluxo de projeto hierárquico físico simplificado, onde o fluxo de projeto contínuo, porém, requer a aplicação de técnicas de particionamento em blocos físicos e a extração dos modelos de tempo destes blocos, o que pode dificultar o trabalho do projetista de um único bloco.

A metodologia hierárquica é útil quando o projeto é muito grande e complexo a ponto de, no momento de sua execução, exceder o espaço de memória física. Outra razão poderia ser o fato da necessidade do projeto ser dividido entre equipes de projetistas e cada parte ser desenvolvida por uma equipe de profissionais em diferentes cidades ou países.

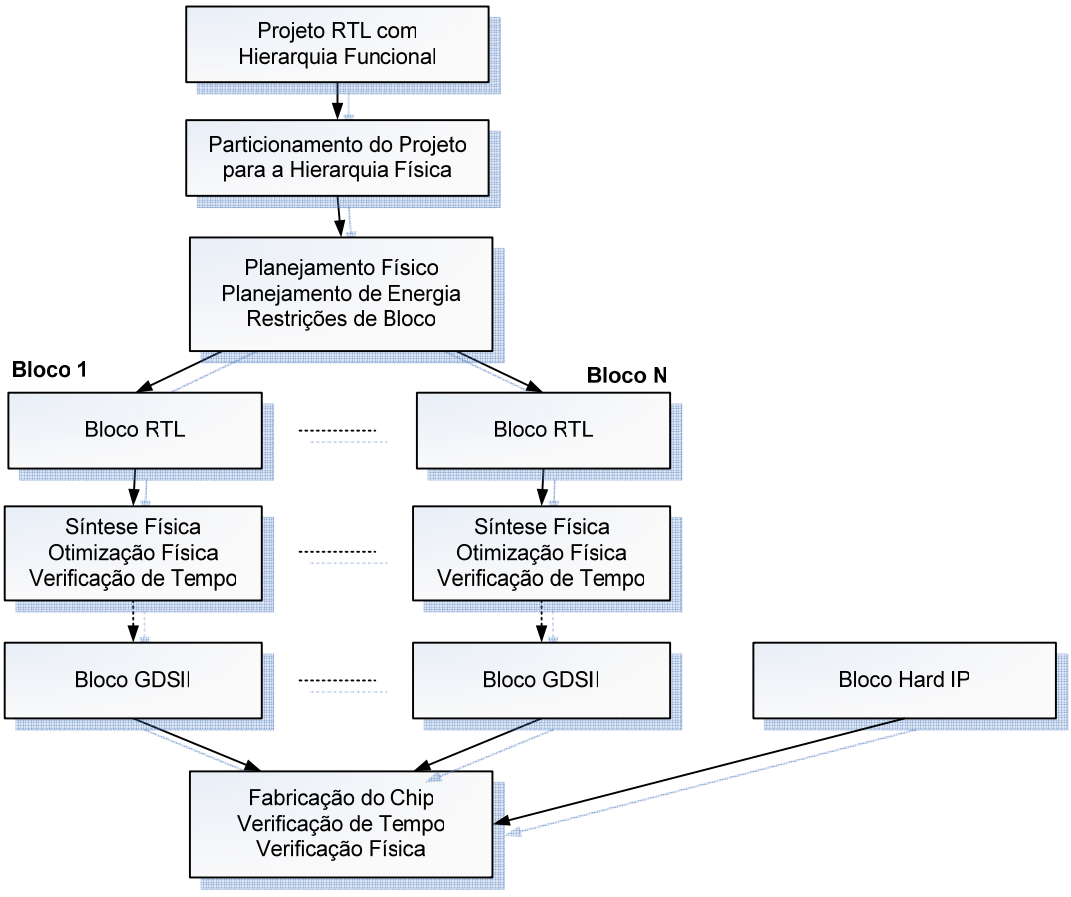

Figura 2 – Metodologia de desenvolvimento hierárquica, traduzida de [1].

2.1.3 Refinamentos Sucessivos

A partir da necessidade de se projetar um sistema embutido depara-se com a escolha da linguagem a ser utilizada para o projeto. As linguagens de programação de software podem

ser utilizadas para a geração de uma especificação executável de um sistema embutido, que pode ser simulada com maior velocidade que um projeto físico. Porém, estas linguagens não descrevem os detalhes do hardware necessários para uma simulação que reflita o hardware final. Além disso, as linguagens de descrição de hardware mais comuns, como VHDL e Verilog, por exemplo, exigem um conhecimento muito preciso e profundo do projetista com relação aos detalhes do hardware, tornando o projeto mais complexo de ser desenvolvido neste nível.

A partir de linguagens de descrição baseadas em C e C++, como SpecC e SystemC, pode-se utilizar a metodologia de Refinamentos Sucessivos para o desenvolvimento de um projeto de sistema embutido. Esta metodologia consiste em se partir de uma especificação executável (nível mais alto de abstração do sistema) e refinar-se o projeto agregando detalhes de cada nível até chegar-se a algum nível que possibilite a síntese. Estes detalhes referem-se, por exemplo, à temporização de uma computação. Uma parte do sistema final demandaria algumas unidades de tempo para ser executada, porém, num nível de abstração maior, este detalhe não é relevante, pois se visa apenas à funcionalidade do sistema e não sua precisão. No momento oportuno este detalhe pode ser agregado ao sistema com a inclusão de um atraso para simular o comportamento mais próximo do sistema final, e assim por diante. Uma das vantagens dessa metodologia é a possibilidade de simulação do projeto a todo o momento, permitindo que possíveis problemas possam ser corrigidos antes da implementação física do protótipo, em cada uma das etapas de desenvolvimento, retornando-se ao nível anterior para a correção e submetendo o projeto a nova simulação e refinamento, conforme Figura 3.

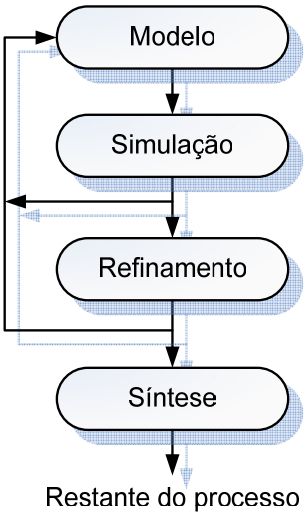

Figura 3 – Fluxo da Metodologia de Refinamentos Sucessivos.

Esta metodologia é citada por Gajski et al [6], Muller, Rosenstiel e Ruf [8], Berner [9], Bhasker [16], Grötker et al [20], dentre outros, e será melhor abordada no capítulo 4. Sua utilização requer a adoção de uma linguagem de descrição de hardware baseada em C/C++, possibilitando que a especificação executável (nível de abstração mais alto) seja refinada até o nível RTL (nível de especificação mais baixo), conforme Figura 4, [20]. A divisão dos níveis pode variar de autor para autor, bem como o seu escopo. Quanto maior a abstração (nível mais alto) mais rápida será a simulação e menor será o número de detalhes de implementação, por outro lado, quanto menor a abstração (nível mais baixo), mais preciso e mais próximo o projeto estará do objetivo final, ou seja, mais próximo do circuito que será o resultado da síntese.

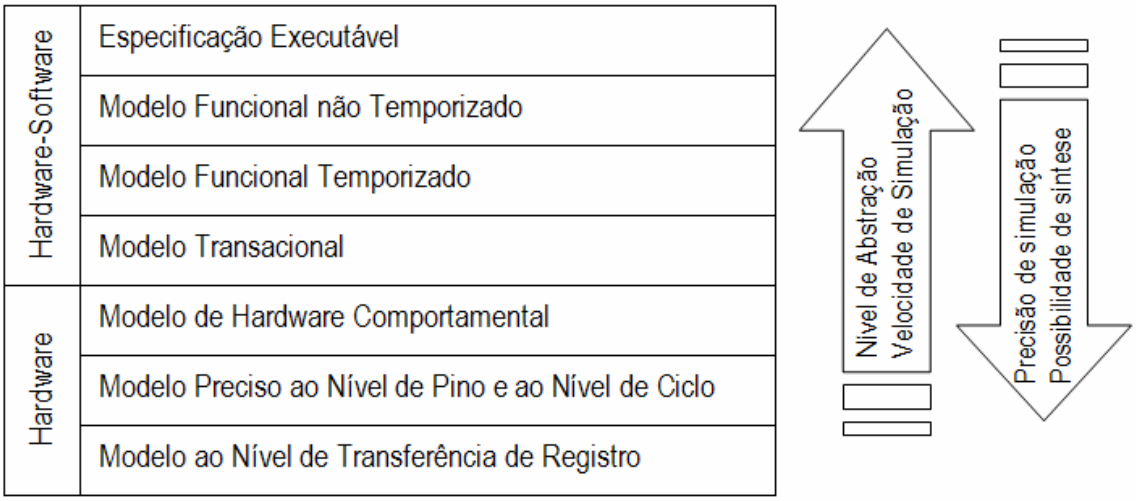

Figura 4 – Níveis de modelagem para refinamentos sucessivos.

2.1.3.1 Níveis de Modelagem

Uma hierarquia de níveis aplicável a projetos de sistemas digitais foi primeiramente descrita por Bell e Newell [95], em 1971, que a aplicaram especificamente a sistemas computacionais. Uma visão dessa hierarquia (Tabela 1) foi proposta por MacFarland, Parker e Camposano [14], em 1990, como uma atualização da proposta de Bell e Newell. Nessa hierarquia temos níveis e seus mapeamentos em cada um dos domínios em que os sistemas podem ser descritos: comportamental (a forma como o sistema interage com o ambiente), estrutural (conjunto de componentes interconectados que implementam o sistema) e físico (especificação de como o sistema está/será construído).

Nível	Domínios		
	Comportamental	**Estrutural**	**Físico**
PMS (Sistema)	Comunicação Processos	Processadores Memórias Chaves	Gabinetes Cabos
Conjunto de Instruções(Algorítmo)	Entrada-Saída	Memory Ports Processors	Board Floorplan
Transferência de Registro	Transferência de Registro	ALU Registradores Multiplexadores Barramentos	Circuitos Integrados Macro Células
Lógico	Equações Lógicas	Portas Lógicas Flip-flops	Standard Cell Layout
Circuito	Equações de Rede	Transistores Conexões	Transistores Layout

Tabela 1 – Hierarquia de níveis de projeto, [14].

Gong, Gajski e Bakshi [15] descrevem um modelo de refinamento para desenvolvimento simultâneo hardware-software. O modelo proposto utiliza a linguagem SpecCharts para a descrição da modelagem e seus refinamentos.

Gajski e Vahid [68], em 1995, propuseram um modelo de níveis de modelagem de sistemas hardware-software embutidos. Em seu artigo, os autores descrevem uma metodologia que intitularam "*specify-explore-refine*", além de discutirem os problemas do projeto e especificação de sistemas, incluindo captura da especificação, exploração do projeto, modelagem hierárquica, síntese de software e hardware e co-simulação. Os níveis de modelagem propostos, são bastante detalhados no artigo, consistindo de uma especificação de alto nível, capturada a partir da descrição geral do sistema e da criação do modelo inicial, que por sua vez são refinados para modelar a funcionalidade do sistema. Após isso, um nível de descrição no nível de sistema é proposto, onde há a descrição de um barramento interligando todos os módulos do sistema que agora já estão com suas funcionalidades bem definidas. Logo em seguida, um nível de projeto de hardware e software se volta para a síntese de software, síntese de alto nível e síntese lógica, definindo e particionando o sistema para cada uma delas. Assim, a descrição no nível RTL, próxima etapa, é facilitada e melhor refinada, para posteriormente o projetista se voltar para o projeto físico com as fases de Posicionamento[3], Roteamento[4] e Temporização[5].

[3] Placement: Fase de otimização na escolha da localização física dos blocos de hardware no chip

Bhasker [16] apresenta um fluxo de desenvolvimento baseado em SystemC. Os níveis são bem definidos, consistindo de 4 modelos (*System Level Model, Timed Model, Behavior Level Model e RTL Model*) que agregam, a cada refinamento, uma característica ao projeto. A Figura 5 aborda a metodologia descrita por Bhasker com algumas características de cada nível, como possibilidade de síntese, precisão de tempo e de ciclo.

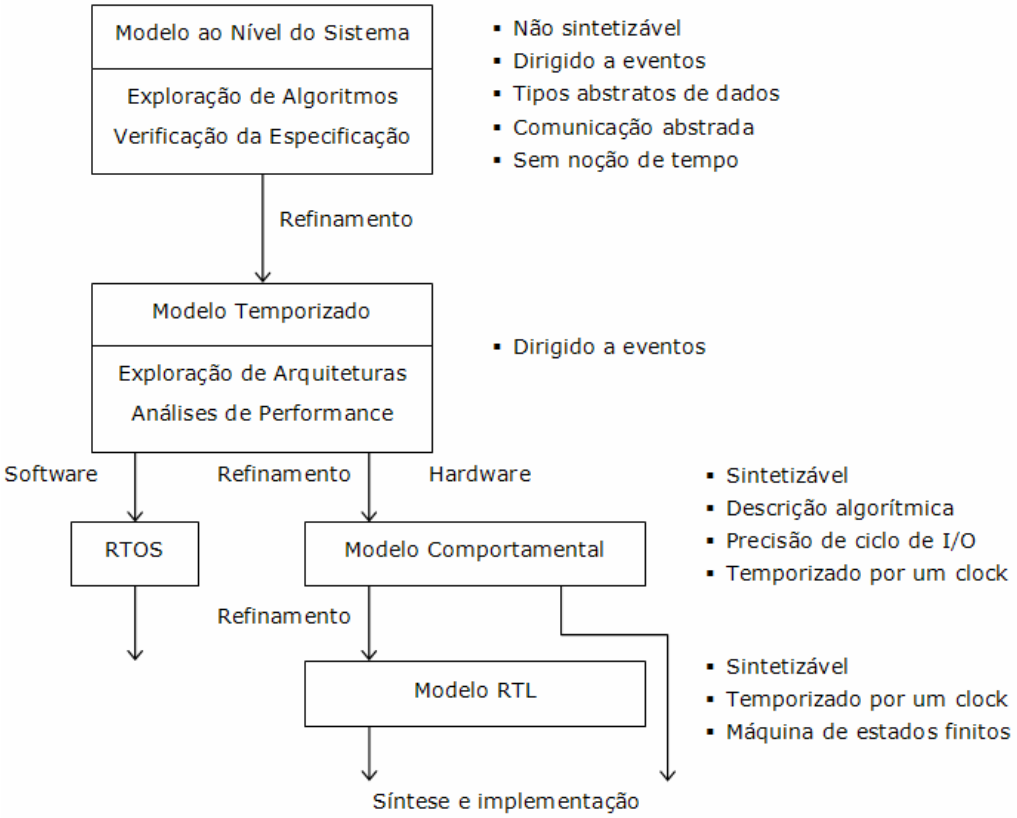

Figura 5 – Fluxo do projeto de sistemas, segundo Bhasker [16].

2.1.3.2 Transaction-Level Modeling (TLM)

A modelagem no nível transacional é atualmente vista não como mais um nível na modelagem de sistemas, como proposto por Grötker [20] e descrito na Figura 4, mas sim como uma metodologia de projeto da comunicação, independente de linguagem. SystemC possibilita uma melhor modelagem usando-se TLM pois permite o refinamento independente,

[4] Routing: Fase de otimização na geração das interconexões de sinais necessárias entre os blocos de hardware no chip

[5] Timing: Fase de otimização que leva em conta os tempos de atraso dos componentes

tanto da comunicação quanto da computação, assim, a metodologia é melhor aplicada usando-se esta linguagem.

Segundo Black e Donovan [83] pode-se usar TLM para a modelagem do sistema desde o primeiro nível (SAM - *System Architectural Model*), que define uma especificação inicial do sistema, até o nível RTL, última instância da descrição, voltada para a síntese. A Figura 6, extraída da obra dos autores, permite observar o fluxo de projeto usando-se TLM e, através de diversos refinamentos sucessivos a partir de SAM (fluxo à direita) chega-se ao nível RTL. Estes refinamentos podem ser percebidos no gráfico à esquerda, onde há um refinamento inicial da comunicação, do nível sem temporização (*Un-Timed*) para um nível com uma temporização aproximada (*Approximate-Timed*). Posteriormente o refinamento se dá na funcionalidade, que agora passa do nível *Un-Timed* para o *Approximate-Timed*. A partir daí, o próximo refinamento pode ser primeiramente tanto na comunicação quanto na funcionalidade, levando-os a uma precisão de ciclo (*Cycle-Timed*) nessa característica, para posteriormente ocorrer o refinamento na outra característica, levando o sistema ao nível RTL.

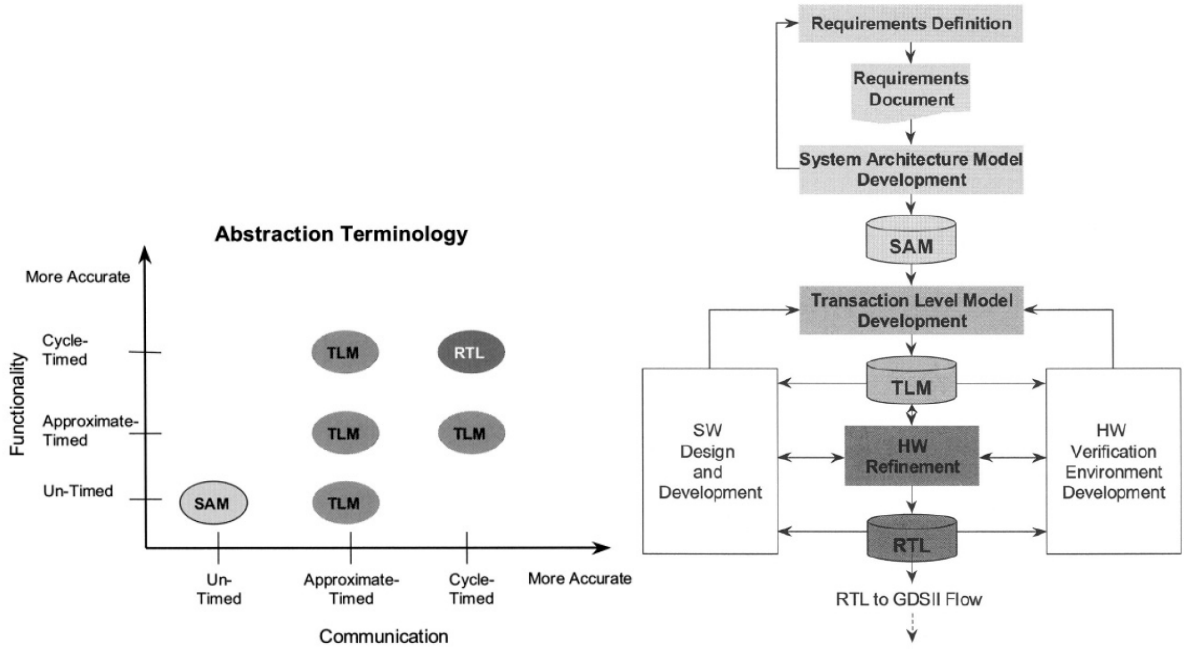

Figura 6 – Relação ente TLM e os níveis (esq.) e Fluxo do projeto usando TLM (dir.), [83].

Já Rose, Swan, Pierce e Fernandez [91], descrevem cada uma das construções que podem e/ou devem ser usadas na modelagem TLM em SystemC, como portas, canais, interfaces e transferências. Em seu trabalho criam exemplos com estas novas interfaces e projetam seus protocolos de comunicação. Pasricha [92] também aborda a modelagem TLM detalhando o

seu uso em um exemplo de SoC, onde descreve a utilização de um canal criado na plataforma para comunicação, usando interfaces para isso. Um canal pode implementar mais de uma interface e esta interface é acessada através da porta do módulo, que é definida em função da interface usada.

Para Ghenassia [90], componentes em TLM são modelados como módulos com um conjunto de processos concorrentes que realizam a computação além de representar o comportamento do componente. Estes módulos comunicam-se em forma de transações através de canais abstratos que usam interfaces. Estas são implementadas ao longo dos canais para encapsular protocolos de comunicação. Para estabelecer uma comunicação, um processo necessita, simplesmente, acessar estas interfaces através das portas do módulo. Essencialmente, a interface é a parte que separa a comunicação da computação num sistema TLM, como por exemplo, `tlm_transport_if`, `tlm_put_if`, `tlm_get_if`, `tlm_master_if`, `tlm_slave_if`, `tlm_fifo_if`, dentre outras.

Ainda segundo Ghenassia [90], TLM define uma transação como uma transferência de dados ou sincronização entre módulos. Essa comunicação pode se dar com qualquer estrutura de palavra ou bit, um exemplo disso seriam transferências de palavras entre registradores de periféricos ou transferências de imagens completas entre dois buffers de memória. A transação pode ser refinada como uma estrutura baseada num protocolo de barramento, incluindo informações como largura do barramento e sua capacidade de transferência.

Um exemplo de uso de interfaces de comunicação está na Figura 7, onde cada interface tem seus métodos para implementar assim o protocolo de comunicação. O método *ready*, na Figura 7, informa que a interface de um módulo está apta a receber dados de forma que a interface oposta possa escrever.

Figura 7 – Interfaces de comunicação (unidirecionais) e seus métodos, [93].

Swan [93], além desta e outras interfaces, descreve também uma pilha de níveis de abstração da modelagem TLM, Figura 8. Ghenassia e Maillet-Contoz [90] abordam os mesmos níveis em função da granularidade dos dados trafegados e em função da precisão de tempo.

A pilha inicia a partir de um nível algorítmico (AL), puramente funcional, sem implementação de aspectos de comunicação. O próximo nível implementa a visão do programador (PV) com uma comunicação baseada em módulos categorizados em *master* (que envia um *request* para iniciar uma comunicação) e slave (que aguarda os *requests* e envia um *response*), sendo uma comunicação baseada em trocas de pacotes de dados sem sincronização temporal. O nível seguinte já implementa a temporização no protocolo de comunicação (PVT), refinando a comunicação para uma arquitetura de barramento, com troca de pacotes menores, baseados no barramento. Logo sendo seguido por um nível que trata de transferências de palavras (largura do barramento, bits), sendo uma comunicação que é sincronizada pelo ciclo do clock. O último nível, RTL, é um nível preciso no nível de pino e de tempo, onde as transferências são dadas por sinais.

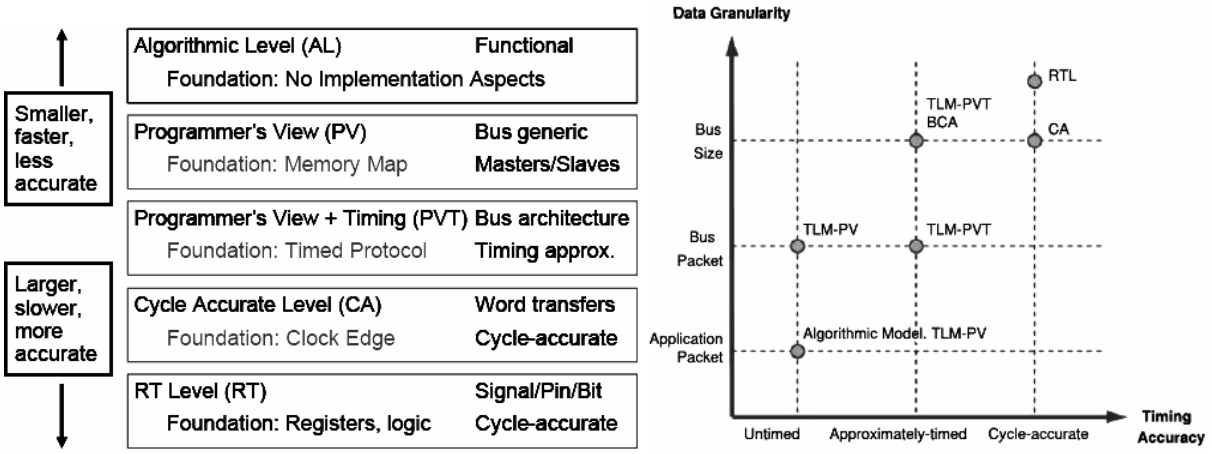

Figura 8 – Níveis de abstração da modelagem TLM (esq) [93], (dir) [90].

2.2 Linguagens

Para adotar a Metodologia de Refinamentos Sucessivos deve-se utilizar uma linguagem que dê suporte ao refinamento. Linguagens baseadas em C/C++ são mais propensas a permitirem o refinamento sucessivo, porém este fato não é suficiente, é preciso que a linguagem tenha

estruturas que permitam o uso do refinamento. SystemC, uma linguagem que foi reconhecida como um padrão reconhecido pelo IEEE, é uma linguagem que vem sendo amplamente utilizada para projetos de SoCs por ter um suporte a refinamentos sucessivos que vai desde a metodologia de desenvolvimento até ferramentas que utilizam a linguagem para tal.

Edwards [26] faz uma análise de diversas linguagens de descrição de hardware e propõe um conjunto de critérios que uma linguagem deste tipo deve seguir. As linguagens analisadas por Edwards em seu artigo foram: Cones, HardwareC, Transmogrifier C, SystemC, Ocapi, C2Verilog, Cyber, Handel-C, SpecC, Bach C e Cash. Algumas destas linguagens serão abordadas aqui. Dentre os critérios pesquisados estão: modelos de concorrência, especificação de temporização, tipos de dados, padrões de comunicação e restrições.

Tanto Edwards [26] quanto De Micheli [27] concordam que o simples fato do uso da linguagem C não é suficiente para descrição de hardware. Linguagens específicas baseadas em C/C++ facilitam o trabalho do projetista permitindo a síntese sem que tenha que haver uma tradução para uma linguagem como Verilog ou VHDL, além da possibilidade da migração de modelos de software já prontos. Para tanto, estas linguagens necessitam atender aos critérios citados anteriormente.

Habibi e Tahar [31] vislumbram como futuro para o projeto de hardware o uso de linguagens baseadas em C/C++ e em Java. Acrescentam, ainda, que projetos poderiam ser sintetizados com um “cc –silicon”. Afirmam também que as ferramentas atuais não são boas o suficiente e o refinamento de um nível para outro deve ser feito manualmente. “Isto significa que a versão da linguagem que será usada deve permitir a descrição de hardware em baixos níveis, não somente no nível algorítmico”.

Nesta seção serão abordadas algumas linguagens de descrição de hardware (HDL – Hardware Description Language) e sua relação com a Metodologia de Refinamentos Sucessivos. Não é objetivo aqui apresentar uma lista exaustiva de linguagens. Serão abordadas as linguagens mais citadas na literatura. A linguagem SystemC, usada neste trabalho será melhor detalhada no Capítulo 3.

2.2.1 VHDL

Juntamente com Verilog, VHDL [30] (***V**ery High Speed Integrated Circuit **H**ardware **D**escription **L**anguage*) foi uma das duas linguagens de descrição de hardware mais utilizadas pela indústria e academia. Tornou-se padrão pelo IEEE em 1987 e tem sintaxe semelhante a Pascal e Ada.

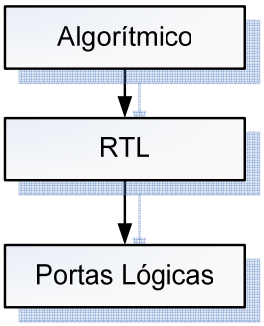

Figura 9 – Níveis de modelagem usando VHDL.

Os possíveis níveis que podem ser usados para se descrever usando VHDL são mostrados na Figura 9. No nível algorítmico não há detalhes de atraso ou mesmo clock no projeto. Algumas ferramentas de síntese podem usar este nível como entrada, mas neste caso, é preciso inferir um clock do algoritmo para que as operações possam ser sincronizadas por ele. Já no nível RTL, há um clock explícito, mas não há detalhes de atraso, que estão presentes no nível Gates, onde os atrasos refletem precisamente a tecnologia utilizada, pois se trata de uma rede de portas e registradores baseados em uma biblioteca desta tecnologia.

O refinamento possível em VHDL consiste na migração para o nível RTL a partir do nível Algorítmico, porém este se trata de um nível comportamental (*behavioral*) onde a descrição está mais voltada para a tecnologia do que para uma descrição em alto nível, priorizando somente o comportamento do sistema final.

2.2.2 Verilog

Verilog [29] tornou-se padrão pelo IEEE em 1995 e é uma linguagem parecida com C, diferentemente de VHDL. Verilog foi introduzida em 1985 pela *Gateway Design System*

Corporation, uma parte da *Cadence Design Systems* [49], sendo uma linguagem proprietária desta empresa até maio de 1990, quando foi formado o *Open Verilog International*.

> As linguagens de descrição de hardware VHDL e Verilog ambas se tornarão as linguagens assembly do projeto de hardware. Os projetistas não terão escolha a não ser escrever código RTL para algo que necessite de performance crítica. Para qualquer outra coisa, eles resolverão em um nível mais alto. (Tradução nossa – HABIBI e TAHAR, 2003, [31]).

Além da citação anterior, Habibi e Tahar [31] ainda classificam as linguagens de descrição de hardware em três grupos e um deles é exatamente o das linguagens clássicas como VHDL e Verilog. Os outros dois grupos são: das linguagens que readaptam linguagens e metodologias de software; das novas linguagens específicas para projeto no nível de sistema.

Verilog permite ao projetista descrever seus projetos em níveis de abstração mais altos que o nível RTL, como nível da arquitetura (*Stocastic*) e nível comportamental (*Algorithmic*). Por outro lado, abaixo deste, é possível descrever utilizando-se de níveis mais baixos, como os níveis *Gates* e *Switch*, conforme Figura 10.

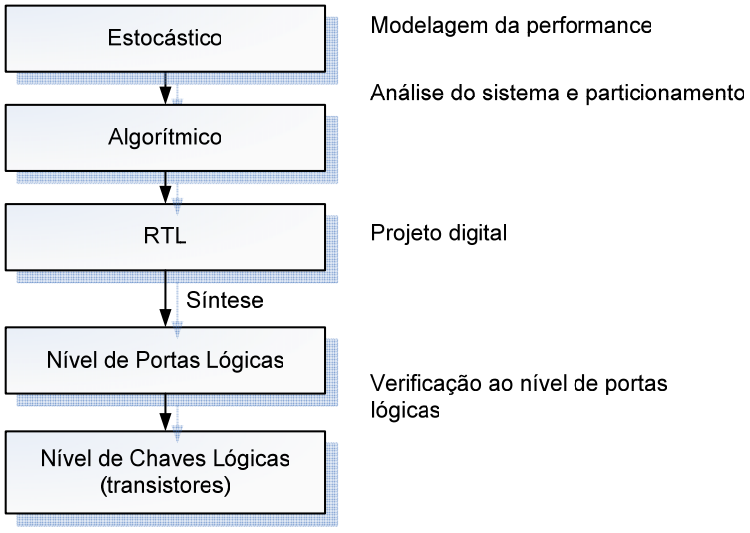

Figura 10 – Níveis de modelagem usando Verilog, traduzida e adaptada de [35].

No nível mais alto (*Stocastic*), Verilog contém funções estocásticas para modelar a performance do sistema a ser desenvolvido. O nível seguinte (*Algorithmic*) é usado para modelar a funcionalidade do sistema, assim como em VHDL, os níveis RTL e *Gate* também estão presentes. O nível *Switch*, abaixo do nível *Gate*, possui um detalhamento maior da arquitetura a ser utilizada na síntese.

O refinamento em Verilog assim como em VHDL é bastante modesto, não permitindo níveis mais altos de abstração que permitam ao projetista trabalhar a funcionalidade do sistema sem se preocupar com detalhes de implementação final.

2.2.3 SystemVerilog

SystemVerilog [32] foi baseado no padrão Verilog e foi introduzido em 2002 pela *Accellera Organization Inc* [34] como uma extensão à linguagem Verilog. Esta extensão visa permitir a descrição e a verificação em níveis mais altos de abstração como, por exemplo, o nível transacional, onde se abstrai das comunicações de forma a se expressar algoritmos complexos eficientemente.

De acordo com Habibi e Tahar [31] as melhorias trazidas por SystemVerilog incluem interfaces que permitem conexões entre módulos em níveis mais altos de abstração além de estruturas e tipos de dados da linguagem C, que possibilitam uma maior liberdade no desenvolvimento. Estas melhorias, ainda segundo o autor, dão a Verilog maior capacidade de verificação de grandes projetos.

Segundo Shutten [11], SystemVerilog é preferido pelos projetistas fluentes em projetos RTL que desejam projetar em níveis mais altos de abstração. Esta possibilidade de aumento da abstração não é permitida por Verilog puramente.

Apesar da possibilidade de maiores níveis de abstração, SystemVerilog não suporta a descrição do projeto em níveis muito elevados como o nível de especificação executável devido à sua herança de Verilog, uma linguagem mais voltada para RTL, porém, assim como Verilog, possui o suporte a níveis mais baixos como *Gates* e *Switches*.

2.2.4 Handel-C

Programas seqüenciais podem ser escritos em Handel-C [37] como são escritos em C convencional, mas o ganho em performance no hardware final será o uso do paralelismo

permitido pela linguagem. Assim como linguagens de alto nível convencionais, Handel-C foi projetada para permitir que um algoritmo possa ser descrito sem a preocupação do conhecimento do funcionamento do hardware final. Segundo [37], a empresa que a desenvolveu – Celoxica –, Handel-C é para o hardware o que uma linguagem de alto nível convencional é para a linguagem assembly do microprocessador. Segundo [26], a linguagem é uma variante de C estendida com construções para processamento paralelo de instruções, além disso, segundo o autor, o modelo de tempo da linguagem é simples onde cada sentença assertiva toma um ciclo de clock para ser executada.

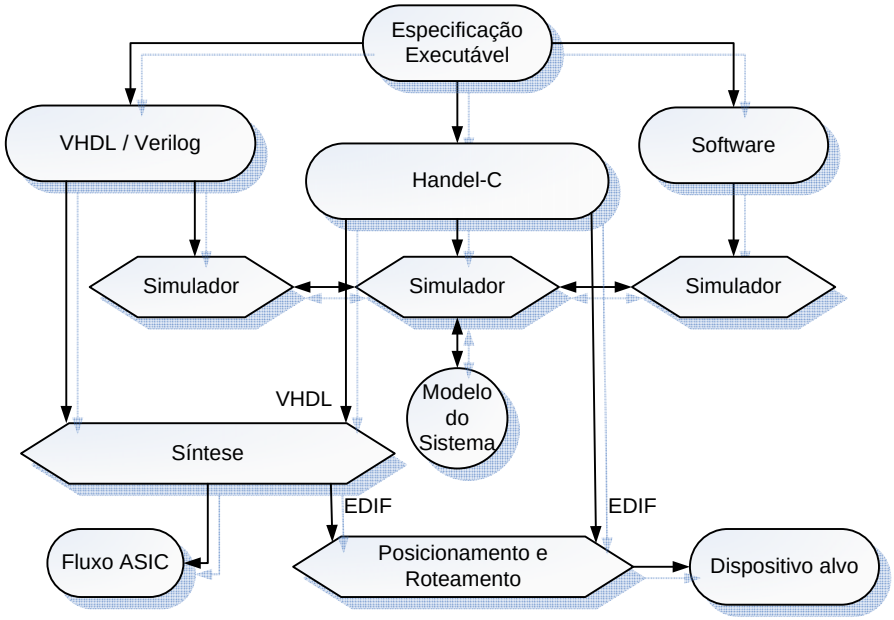

Figura 11 – Fluxo de desenvolvimento em Handel-C, traduzida de [38].

Handel-C é uma linguagem usada para implementação de algoritmos diretamente numa FPGA, reduzindo o tempo do projeto em 3-4 meses. A linguagem não apresenta suporte ao refinamento, mas uma facilidade de desenvolvimento de algoritmos complexos em C e sua síntese direta, como mostrado na Figura 11. Isto livra o projetista/programador de se preocupar com problemas/decisões de baixo nível como tipo apropriado de porta ou registrador a ser usado. Um exemplo do seu uso é a síntese de protocolos e algoritmos para roteadores ou encriptação de forma rápida.

2.2.5 HardwareC

HardwareC foi desenvolvida por Ku e De Micheli [39], com o objetivo de se ter uma linguagem de descrição de hardware que possibilite o uso de restrições para direcionar a síntese. Estas restrições são inseridas no código através de *tags* e *labels*, conforme Figura 12. É uma linguagem baseada na sintaxe da linguagem C, estendendo esta última com a noção de concorrência, passagem de mensagens, restrições de tempo via tags, instanciação explícita de modelos, dentre outras características. Assim, HardwareC possui sua própria semântica que difere em muitos aspectos da linguagem C, pois suporta tanto a semântica declarativa como a procedural. Programas nesta linguagem são sintetizados e otimizados pelos sistemas Hercules e Hebe [40].

```
constraint maxtime from label1 to label3 = 10 cycles;
constraint delay of label2 = 5 cycles;
label1:
A = read(X);
Y = A + 1;
label2:
Z = Y * Y;
label3:
```

Figura 12 – Exemplo de código em HardwareC.

Segundo Habibi e Tahar [31], a linguagem possui muitas melhorias que torna mais expressivo o seu poder bem como sua facilidade de descrição. Os autores ainda levantam que a implementação final satisfaz as restrições de tempo e recursos impostas pelo projetista, sendo em termos de interconexões de lógica e registradores, descrita no formato chamado Structural Logic Intermediate Format [40]. Apesar disso, segundo De Micheli [27], a tradução de C para HardwareC não é trivial, o que torna a linguagem apenas mais uma linguagem de descrição de hardware.

A linguagem HardwareC, apesar de sua facilidade na descrição e das diversas possibilidades de imposição de restrições ao projeto, não dá suporte ao refinamento sucessivo e tampouco a descrição em níveis diferentes de abstração.

2.2.6 Cynlib

Cynlib [41] é um conjunto de classes de C++ que implementam muitas características encontradas em Verilog e VHDL, provendo um vocabulário para modelagem de hardware em C++, sendo, portanto, uma biblioteca que implementa muitas características semânticas de Verilog. Por exemplo, a linguagem suporta o equivalente à sentença "casex" de Verilog, tendo uma sintaxe diferente da sentença "switch" da linguagem C. O objetivo desta biblioteca é criar um ambiente C++ em que tanto o hardware quanto o ambiente de teste possam ser modelados e simulados. Algumas características que Cynlib estende a C/C++ são: modelo de execução concorrente, módulos, portas, threads, tipos de dados orientados a hardware, dentre outras. Para simulação nesse ambiente, os códigos devem ser compilados e executados como numa utilização normal de um ambiente de programação C/C++. O uso do formato VCD (*Value Change Dump*) permite a visualização em formas de onda, dos resultados coletados nas execuções do sistema.

Este ambiente permite a descrição de hardware em C++ em alto nível, porém sua síntese depende de uma tradução para linguagens de baixo nível, como Verilog/VHDL, já que não existe síntese a partir de Cynlib diretamente. Apesar disso a linguagem permite uma descrição em alto nível e seu refinamento até o nível Cynlib RTL (mais baixo nível), porém não se encontram definições para os níveis intermediários.

2.2.7 SpecC

Desenvolvido por Gajski et al, SpecC é mantido pelo SpecC Technology Open Consortium [36] cujo objetivo é criar formatos e linguagens padrões para projeto de especificação de sistemas usando a tecnologia SpecC. Além disso, este consórcio estabelece e dissemina métodos de desenvolvimento de SoCs (System on Chip).

SpecC é uma linguagem de descrição de hardware no nível de sistema, bem como uma metodologia de desenvolvimento. A linguagem é um superconjunto de C, aumentado com estruturas para modelagem de software e hardware, incluindo máquinas de estado finito, concorrência, pipeline, dentre outras.

Segundo Edwards [26], SpecC impõe uma metodologia de refinamento de tal forma que um sistema possa ser sintetizado a partir de uma série de refinamentos manuais e automatizados. A Figura 13 apresenta o fluxo de desenvolvimento em SpecC assim como os possíveis níveis de implementação e refinamento na linguagem.

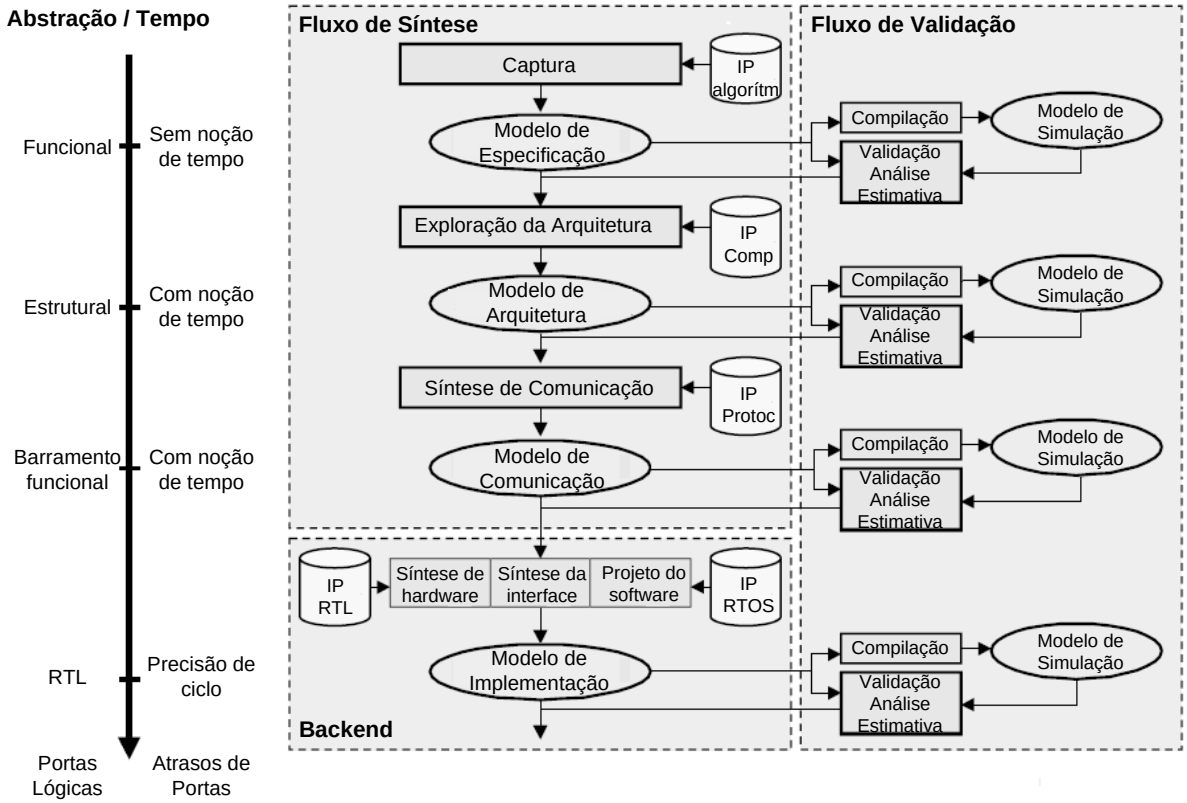

Figura 13 – Fluxo de desenvolvimento em SpecC, traduzida de [38].

De acordo com Habibi e Tahar [31], SpecC foi desenvolvida com objetivo de ser uma linguagem que possibilitasse uma maior facilidade de síntese e verificação, o que pode ser confirmado pela Figura 13.

Gerstlauer [44] apresenta muito detalhadamente a metodologia, detalhes da linguagem bem como o fluxo de projeto utilizando SpecC. Cada um dos níveis de abstração são bem detalhados e explorados pelo autor em seu trabalho que aborda quatro níveis/modelos, sendo eles: *Specification Model*, *Architecture Model*, *Communication Model* e *Implementation Model*.

Já Cai et al [5] apresentam um comparativo entre SpecC e SystemC, levantando detalhes intrínsecos de cada linguagem e comparando seu desempenho e facilidade em detalhes

específicos em cada um dos cinco níveis definidos para este teste. Cai et al concluem que SpecC é superior a SystemC devido aos aspectos analisados.

SpecC mostra-se uma linguagem bem definida e com níveis de abstração que facilitam o projeto de sistemas embutidos. Em comparação com as outras linguagens apresentou um dos melhores suportes ao refinamento e definição de níveis.

2.2.8 SystemC

SystemC [21], uma biblioteca de classes C++, é mantida pelo Open SystemC Initiative (OSCI) [43], uma organização formada por empresas envolvidas em projetos, semicondutores, provimento de IP, desenvolvimento de software embutido e ferramentas de automação de design, bem como universidades e interessados em suportar o uso da linguagem como um padrão aberto para projeto no nível de sistema.

SystemC é uma plataforma de modelagem baseada em C++ permitindo a descrição de hardware no nível RTL, comportamental e níveis de sistema. Consiste de uma biblioteca de classes C++ e um *kernel* de simulação, o que possibilita o uso de tipos abstratos de dados, modularidade e orientação a objeto. Segundo Panda [45], SystemC possui a habilidade de reusar IPs e testbenches entre diferentes níveis de abstração e mesmo entre projetos diferentes, de forma eficiente e fácil. Em seu trabalho, Panda descreve detalhes da linguagem bem como do seu fluxo de projeto.

Diferentemente do descrito por Habibi e Tahar [31], SystemC não objetiva somente o desenvolvimento direto no nível RTL. A linguagem é bastante robusta e poderosa o suficiente para possibilitar o desenvolvimento em diversos níveis de abstração, utilizando-se o refinamento sucessivo, para permitir a incorporação de detalhes a cada nível.

Esta linguagem será descrita com maiores detalhes no Capítulo 3.

2.2.9 SystemC++ ou SystemC-Plus

SystemC-Plus é uma linguagem de descrição de hardware baseada em SystemC. Segundo Grimpe et al [47], SystemC oferece altos níveis de modelagem, porém não para síntese, pois a mesma ocorre nos níveis mais baixos de abstração. Ainda segundo os autores, o refinamento manual em direção ao modelo sintetizável, usando esta linguagem, é lento e passível de erros. Assim, propuseram a linguagem SystemC-Plus [3] que consiste de um conjunto sintetizável de SystemC, com a adição de conjuntos de construções orientadas a objeto e que são aceitas por ferramentas de síntese. Além disso, à SystemC-Plus foi adicionada outra biblioteca de classes (OOHWLib) que adiciona a possibilidade de uso do polimorfismo na descrição do projeto. O fluxo de simulação e síntese a partir de SystemC-Plus está descrito na Figura 14.

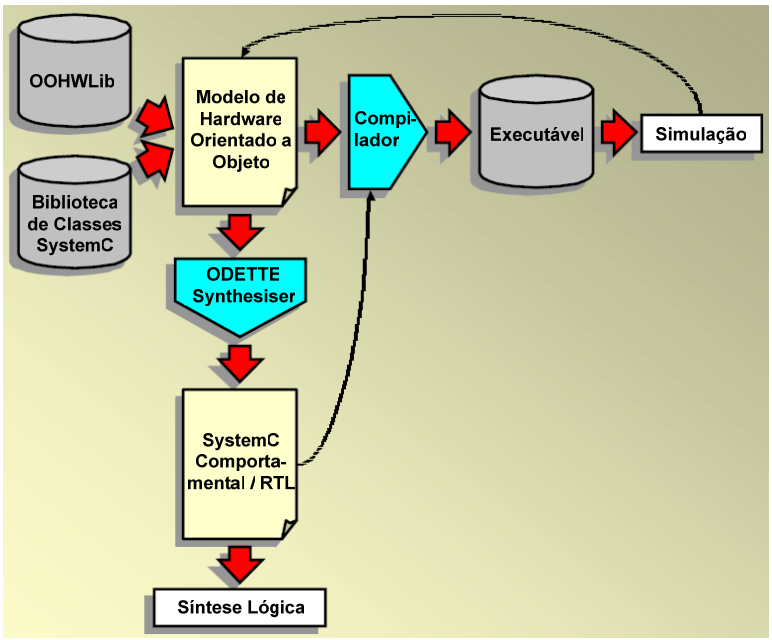

Figura 14 – Fluxo de simulação e síntese usando SystemC Plus, traduzida de [47].

Por ser baseada em SystemC, a linguagem possui o suporte ao refinamento sucessivo, porém é uma linguagem desenvolvida para ser usada com a ferramenta ODETTE [4], visando uma síntese de alto nível usando uma descrição de hardware com características de orientação a objeto como polimorfismo, herança, objetos globais, para modelagem de recursos compartilhados e comunicação, dentre outras. Até o presente momento, esta linguagem não se tornou padrão pelo IEEE.

2.2.10 Comparação entre as linguagens

O uso de uma linguagem baseada em C/C++ facilita o projeto de sistemas de hardware devido à abstração permitida pela linguagem. Dentre as linguagens aqui analisadas, SystemC e SpecC são as únicas que permitem o uso da metodologia de refinamentos sucessivos, por terem as estruturas necessárias para isso. A Figura 15, extraída de Dömer [48], foi adaptada com a inclusão de SystemC na tabela, permitindo-nos visualizar sua comparação com as demais linguagens.

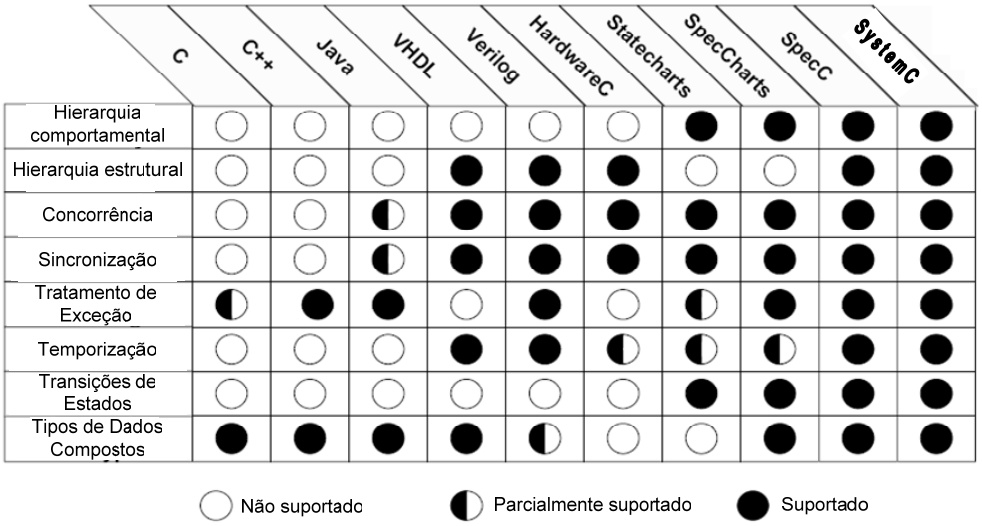

Figura 15 – Comparação entre algumas linguagens de descrição de hardware, a partir de requisitos específicos, adaptada de [48].

SystemC não foi bem explorado por Cai, Verma e Gajski [5] porque foram utilizados níveis pré-definidos e o potencial de SystemC não foi aproveitado ou, melhor vislumbrado, por estes níveis, como por exemplo, a possibilidade de se desenvolver projetos em SystemC no nível TLM, dentre outros. Assim, a escolha por SpecC justificada pelo aumento de complexidade de projeto usando SystemC, como relatado por Cai, não é bem fundamentada por não serem analisados todos os pontos fortes da linguagem SystemC. O autor utiliza os critérios analisabilidade, explorabilidade, refinabilidade e validabilidade, por ele definidos como essenciais a uma linguagem de descrição de hardware no nível de sistema.

Na análise de Al-Junaid e Kazmierski [28], em seu artigo de 2005, SystemC foi a linguagem escolhida para ser estendida com características que a permitam descrever sistemas analógicos. Segundo os autores, SystemC tem tido uma imensa popularidade desde sua

introdução em 1999 e alcançou a aceitação da indústria, além disso, é a única linguagem que permite a modelagem de IPs (*Intellectual Property*) em diversos níveis, para sistemas contendo componentes de hardware e software. A linguagem proposta no trabalho foi SystemC-A, cujo escopo é mostrado na Figura 16.

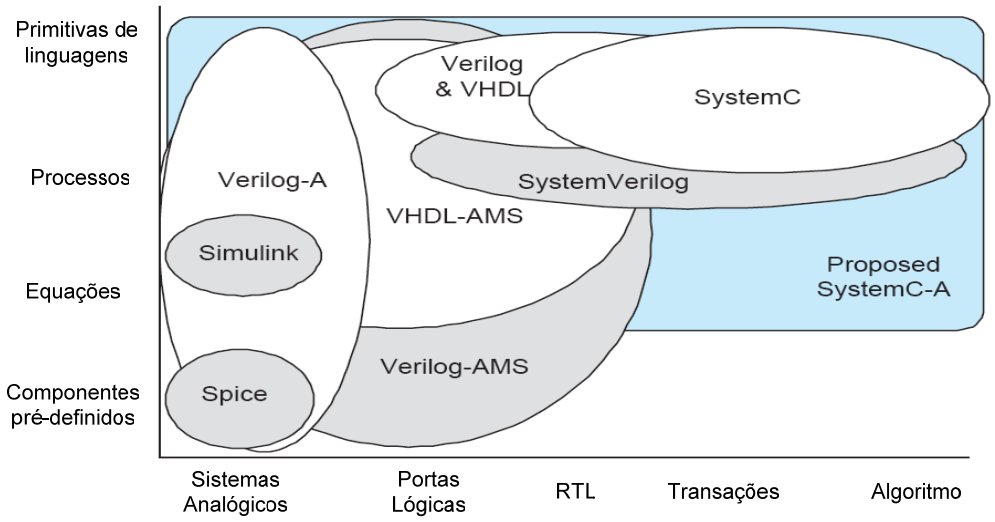

Figura 16 – Comparativo entre algumas linguagens a partir do escopo e níveis de abstração, traduzida de [28].

Habibi e Tahar [31], analisando diversas linguagens comparam SpecC a SystemC e concluem que SystemC atende a todos os requisitos necessários para suportar o projeto no nível de sistema.

Em seu trabalho, intitulado *Top-down system level design methodology using SpecC, VCC and SystemC*, Cai et al [46] propõem o uso de três linguagens de descrição de hardware no projeto de sistemas embutidos, sendo elas SpecC, VCC[6] e SystemC. A idéia principal é poder aproveitar o potencial de cada uma dessas linguagens, baseadas em C/C++, em fases específicas do projeto, como descrito na Figura 17.

[6] VCC é uma ferramenta desenvolvida pela *Cadence Design Systems* [49] e descontinuada. Seu objetivo é o projeto comportamental/arquitetural de um sistema, descrito na linguagem *whitebox* C, baseada em C.

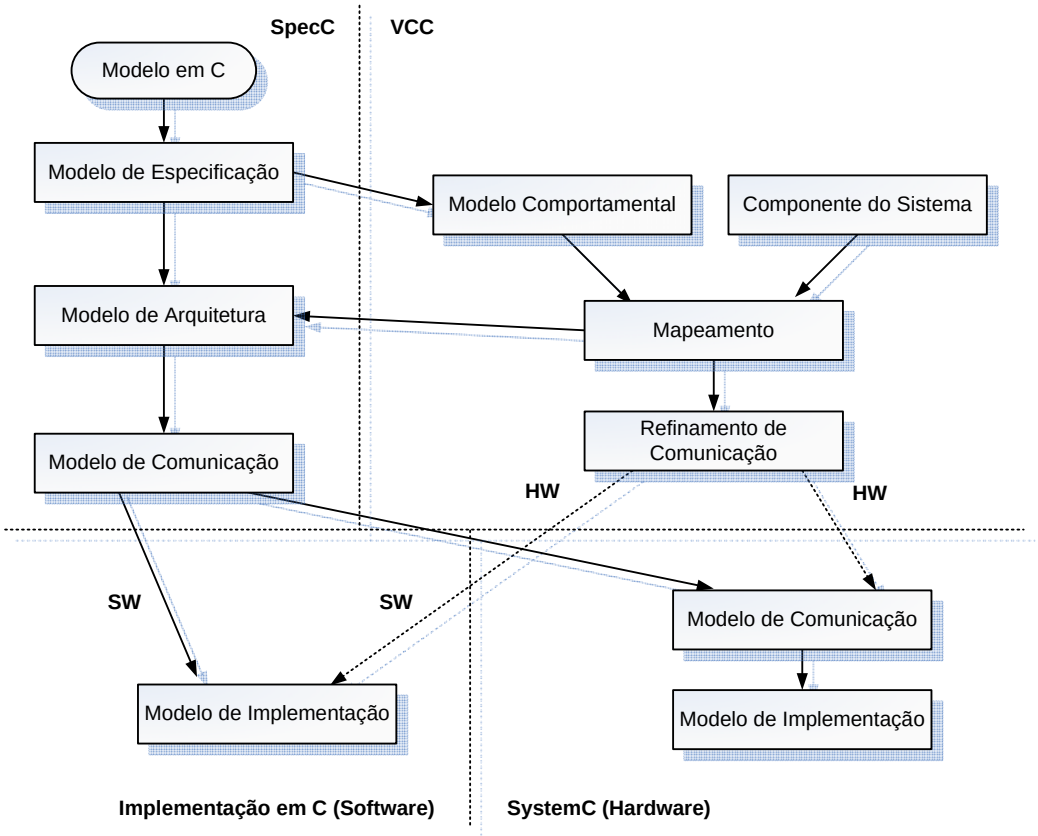

Figura 17 – Fluxo de desenvolvimento proposto por Cai et al, traduzida de [46].

Apesar desta abordagem de Cai et al, SystemC poderia ter sido utilizada em todo o fluxo do projeto, por ter suporte ao refinamento sucessivo e à descrição em diversos níveis de abstração, do mais alto ao mais baixo, sem a necessidade de tradução/adaptação entre as linguagens envolvidas. Isto dá, ao projetista, maior liberdade e fluência no projeto, reduzindo tempo com adaptações, mesmo que, neste caso, sejam linguagens com a mesma base (C/C++), pois possuem detalhes semânticos e sintáticos que diferem em alguns pontos.

Portanto, a linguagem escolhida para o desenvolvimento deste trabalho foi SystemC.

2.3 Ferramentas

O objetivo desta seção é fazer uma análise a fim de verificar quais ferramentas oferecem ao projetista o suporte necessário para trabalhar com refinamento sucessivo, como a definição de níveis e a orientação dos passos para a mudança/refinamento, de um nível de abstração para

outro. Não é objetivo aqui comparar as ferramentas a fundo e muito menos ser uma lista exaustiva das ferramentas disponíveis. Além disso, esta análise busca melhores soluções que possam ajudar a implementar uma ferramenta que dê o devido suporte ao projetista.

As ferramentas foram divididas em três categorias: Ferramentas de Síntese, voltadas para a finalização do projeto, com a geração do código VHDL ou Verilog; Ferramentas de Desenvolvimento, que visam auxiliar o projetista no desenvolvimento do projeto; Ferramentas de Apoio, que são úteis antes ou durante o desenvolvimento do projeto, pois facilitam a adição de módulos, incorporação ou mesmo tradução/adaptação.

2.3.1 Ferramentas de Síntese

2.3.1.1 ODETTE/ICODES

Utilizando a linguagem SystemC Plus, a ferramenta de síntese ODETTE (*Object-Oriented co-Design and functional Test TEchniques*) [3][4], cujo projeto foi concluído em 2003, visa utilizar as bibliotecas padrão de C/C++ e SystemC, além do suporte à orientação a objeto para descrição de hardware, para gerar código no nível comportamental ou RTL de SystemC ou até mesmo código em VHDL. A ferramenta ainda facilita a co-simulação HDL/SystemC, a criação de bibliotecas e técnicas de verificação de projeto. A saída da ferramenta é facilmente sintetizada por outras ferramentas como *Synopsys CoCentric SystemC Compiler* ou *Synopsys Design Compiler*. Este projeto é desenvolvido por um consórcio europeu envolvendo Synopsys, Siemens, OFFIS, Bosch, IBM e ECSI, visando a disponibilização gratuita da ferramenta através do site do projeto [4].

Não há referência alguma, na página do projeto, sobre o suporte da ferramenta ao refinamento sucessivo. Seu objetivo é apenas a síntese de alto nível usando bibliotecas e estruturas que permitam o uso de orientação a objeto na descrição de sistemas de hardware.

Apesar o projeto ter sido finalizado em 2003, seu sucessor, ICODES [67] (*Interface and Communication based Design of Embedded Systems*) encontra-se em desenvolvimento pelos mesmos autores, sendo um complemento do trabalho anterior (ODETTE). O projeto visa o

desenvolvimento de novas tecnologias de modelagem, análise e síntese de sistemas complexos de hardware/software embutidos com muitos componentes de comunicação. Porém, esta ferramenta, assim como sua precessora, não possui suporte ao refinamento sucessivo de forma que auxilie o projetista no momento do refinamento, com orientações e/ou passos automatizados.

2.3.1.2 Forte Cynthesizer

Ferramenta proprietária desenvolvida pela Forte Design Systems, o Cynthesizer [58] é um ambiente de projeto que possibilita a síntese comportamental, a verificação, prototipagem em FPGA, otimização de consumo de energia, redução de área de silício e gerenciamento do projeto. O uso da ferramenta depende da aquisição da licença. A partir de um projeto SystemC no nível TLM (*Transaction Level Model*), o ambiente gera o RTL correspondente, reduzindo a necessidade de refinamentos para se alcançar este nível. Segundo o fabricante, o uso do nível TLM aumenta em 200 vezes a velocidade de simulação em comparação com o nível RTL o que justifica a síntese a partir deste nível. O fabricante ainda destaca a possibilidade do uso do mesmo testbench durante todo o projeto. Porém a ferramenta não dá suporte ao refinamento sucessivo, apenas permite a síntese a partir de um nível de abstração alto, como o nível TLM ou de um nível intermediário, como o nível comportamental.

2.3.1.3 Synopsys Design Compiler

O Design Compiler [86], de propriedade da Synopsys Inc., é uma ferramenta para síntese a partir do código RTL, tendo como entradas: o código RTL, uma biblioteca lógica e outra física além das restrições do projeto. Seu uso depende da aquisição da licença. Sendo desenvolvida para projetistas RTL, não requer nenhuma mudança no projeto para seu uso, consistindo de uma linha de comando com alguns parâmetros, que não requer conhecimento físico. Sua saída é uma netlist otimizada para tempo, área e consumo de energia, com um layout com precisão de tempo, pronto para a implementação física. Não se trata de ferramenta de desenvolvimento, é apenas uma ferramenta de síntese. Possui uma interface de linha de comando bastante simples, chamada DC_SHELL.

2.3.1.4 Celoxica Agility Compiler

Desenvolvido pela Celoxica Inc., o Agility Compiler [84] é uma ferramenta comercial para desenvolvimento comportamental em SystemC e sua síntese RTL ou em netlists EDIF, conforme Figura 18. O objetivo é não gastar tempo com refinamentos e sim com a prototipação do projeto e sua síntese, dessa forma, a ferramenta permite o desenvolvimento no nível TLM, assim como o Forte Cynthesizer. A ferramenta ainda trabalha com verificação do projeto, prototipagem e otimização em FPGA (Stratix II e Virtex 4, da Altera). A ferramenta ainda suporta desenvolvimento em C/C++ e Handel C, com simulação e depuração com precisão de ciclo.

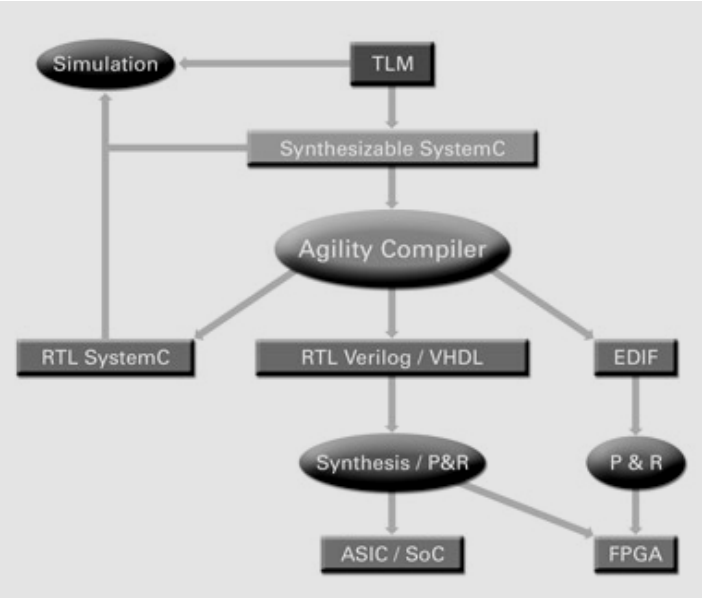

Figura 18 – Fluxo de design do Agility Compiler [84].

Basicamente é uma ferramenta de síntese com possibilidade de desenvolvimento do projeto, porém com limitações com relação ao refinamento, não havendo suporte para isso. Numa visualização gráfica, **Erro! Fonte de referência não encontrada.**, pode-se perceber que a ferramenta permite a visualização gráfica do projeto através da ligação dos módulos além da sua associação com o código.

2.3.2 Ferramentas de simulação e desenvolvimento

2.3.2.1 Cadence Incisive Unified Simulator

O *Incisive Unified Simulator* [50] é um ambiente de simulação desenvolvido pela *Cadence Design Systems* [49], dependendo também da aquisição de licença. Seu objetivo é ser um ambiente que permita a simulação do projeto com a possibilidade de suporte de diversas linguagens (*mixed language simulation*), como SystemC, Verilog, SystemVerilog, VHDL, SystemC Verification Library (SCV), PSL (*Property Specification Language*) e OVL. Este suporte se dá devido a um *kernel* único e heterogêneo.

Segundo o site da empresa desenvolvedora [51], a ferramenta provê um ambiente que permite o refinamento sucessivo do projeto escrito em SystemC até o nível RTL. Porém não se encontra mais informações a respeito disso.

2.3.2.2 Synopsys CoCentric System Studio

CoCentric System Studio [52], desenvolvido e comercializado pela Synopsys Inc., é um ambiente para projeto no nível de sistema, consistindo de ferramentas, metodologias e bibliotecas para facilitar o projeto e simulação de *systems-on-a-chip*. Este ambiente trabalha com dois modelos de projeto:

- Modelo Algorítmico – Descreve a funcionalidade do sistema num nível sem precisão de tempo, consistindo de uma mistura de *data flow* com máquina de estados hierárquica. Neste caso, detalhes como clock e reset não são modelados. Os modelos algorítmicos possíveis são: DFG (Data Flow Graph), OR, AND, GATED, PRIM, e SDS.

- Modelo Arquitetural – Modelagem da arquitetura do sistema em diversos níveis de abstração, usando SystemC.

Apesar da possibilidade do uso do Modelo Arquitetural, não há suporte ao refinamento sucessivo neste ambiente, ou seja, a ferramenta não ajuda o projetista a refinar o seu projeto. Existe, sim, uma facilidade de uso e de desenvolvimento do projeto, mas não um apoio ou orientação ao projetista quanto ao refinamento sucessivo.

2.3.2.3 Coware ConvergenSC Advanced System Designer

O ambiente *ConvergenSC Advanced System Designer* [7], desenvolvido pela CoWare Inc. é uma ferramenta comercial que oferece ao projetista um ambiente gráfico que permite a criação dos blocos do sistema a ser desenvolvido e suas conexões. É possível se importar blocos RTL, VHDL ou Verilog, com a geração automática do nível transacional e as interfaces de conexão com os blocos RTL. A ferramenta ainda possibilita o desenvolvimento através de duas metodologias, sendo Top-Down (*Hardware-Software Co-Design*), desenvolvimento a partir de uma especificação executável, e *Platform-Based*, que parte de IPs reusáveis que direcionam o fluxo do projeto. Porém esta ferramenta não implementa um suporte ao refinamento sucessivo dos níveis de abstração possíveis da metodologia de desenvolvimento em SystemC.

2.3.2.4 Summit Vista Design Environment for SystemC

Assim como o CoCentric e o ConvergenSC, o *Vista Design Environment for SystemC* [53], ambiente de projeto da Summit Design Inc. possui diversas características para facilitar o projeto de sistemas, porém sem o suporte ao refinamento sucessivo. Existe uma versão para teste válida por trinta dias, mas a licença deve ser adquirida para a continuidade do uso. Segundo seu Datasheet, o ambiente permite o desenvolvimento em qualquer nível de abstração suportado por SystemC, além de ser uma plataforma de verificação e simulação. A síntese é feita a partir da integração com outras ferramentas de outros fabricantes, como o Forte Cynthesizer (Figura 19). A ferramenta permite visualizar o projeto como um diagrama de blocos, que podem ser visualizados na forma gráfica, e permite ainda a visualização do código de cada um. Trata-se de uma ferramenta recente, lançada em 2006, e bem completa enquanto ambiente de desenvolvimento. Uma característica interessante é a Transaction Sequence Viewer, que permite a captura automática do método de interface utilizado nas

comunicações apresentando automaticamente um protocolo de comunicação, facilitando o seu desenvolvimento pelo projetista.

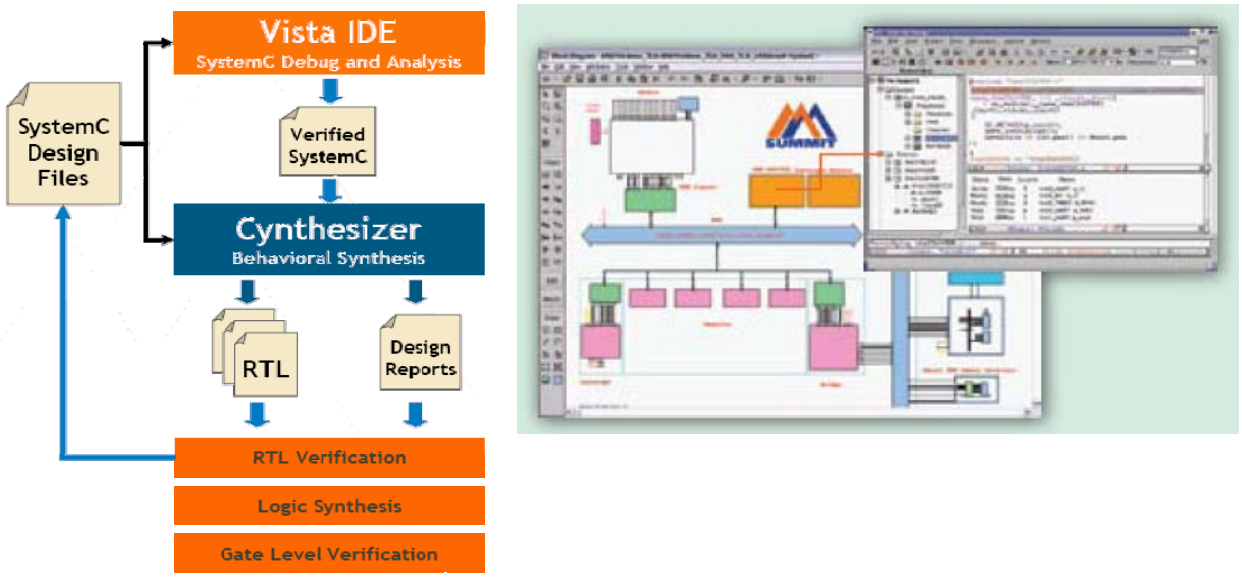

Figura 19 – Fluxo de projeto do Vista (esquerda) e visualização gráfica dos blocos do projeto (direita), [53].

2.3.2.5 Prosilog Magillem Graphical Platform Builder

Em sua versão 3.0 (lançada em maio de 2005), Magillem [88], é uma IDE comercial para projeto de hardware, desenvolvida pela Prosilog SA, que permite a interconexão de plataformas (em VHDL, Verilog ou SystemC) e barramentos, pontes, adaptadores de protocolos, providos em modelos sintetizáveis ou simuláveis. A versão 3 da ferramenta apresenta uma interface baseada no ambiente aberto Eclipse[7], possibilitando o uso maciço de plug-ins e de estruturas de dados internas baseadas em API (*Application Program Interface*). Além disso, o uso do SPIRIT[8] Packager Module permite a geração de código XML a partir dos arquivos do projeto, além de possibilitar uma melhor integração com ferramentas desenvolvidas por terceiros. A ferramenta é um *front end* gráfico que permite não só a integração de blocos de IP como também o seu desenvolvimento tanto no nível transacional quanto no RTL. Porém nenhuma informação foi encontrada sobre seu suporte ao refinamento entre os níveis.

[7] Eclipse Platform. http://www.eclipse.org

[8] Structure for Packaging, Integrating and Re-using IP within Tool flows. http://www.spiritconsortium.org

Na data em que esta parte do texto foi escrita (outubro/2006) o site da empresa Prosilog (http://www.prosilog.com) não se encontrava disponível na Internet. As informações foram colhidas de sites da área que fazem referência à ferramenta. Por exemplo, [89] apresenta muitos detalhes da ferramenta como especificação (uso de bibliotecas TLM, barramentos AMBA, adaptadores, dentre outros), verificação (análise de performance, interface de co-verificação, dentre outros) e integração (controladores de memória, DMA, interrupções, UART, CPIO, dentre outros).

2.3.2.6 SyCE

SyCE [59] é um ambiente integrado para projetos em SystemC, desenvolvido na Universidade de Bremen em 2005. Segundo o autor, é o primeiro ambiente integrado para SystemC que permite o projeto, verificação, depuração de sistemas e visualização do resultado da sua depuração. O ambiente possui suporte a descrição em diversos níveis de abstração, mas não apresentou o suporte ao refinamento sucessivo necessário para orientação do projetista. O desenvolvimento do ambiente consistiu na integração de quatro componentes que foram desenvolvidos separadamente na mesma Universidade (Figura 20, esquerda):

- ParSyC [60] - Um parser para projetos em SystemC, visando à geração de uma representação intermediária do código do projeto, possibilitando o seu uso para visualização, verificação formal ou outras propostas.

- CheckSyC [62] – Uma ferramenta de verificação de equivalência formal, propriedades e geração de verificadores para simulação ou síntese. A partir da representação intermediária gerada por ParSyC, esta ferramenta implementa esta representação em forma de uma máquina de estados finita que por sua vez é traduzida para um problema de decisão booleana, que é resolvido usando-se um *Solver* (SAT).

- DeSyC [61] – Uma ferramenta para depuração automática e localização de erros em netlists, usa como entrada a netlist gerada por ParSyC e o resultado de CheckSyC. Provê uma localização automática dos pontos candidatos a erro, diferentemente da maioria das ferramentas de depuração, que necessitam de uma simulação e depuração quase manual, segundo o autor [61].

- ViSyC [63] – Uma ferramenta para visualização esquemática e do código fonte, podendo ser usada na forma "*stand-alone*" a partir da informação gerada pelo ParSyC ou em combinação com a saída do DeSyC. ViSyC permite a fácil navegação através de projetos complexos além de permitir a visualização dos resultados da depuração, ajudando o projetista a encontrar mais facilmente as razões dos problemas, como mostrado na Figura 20, direita.

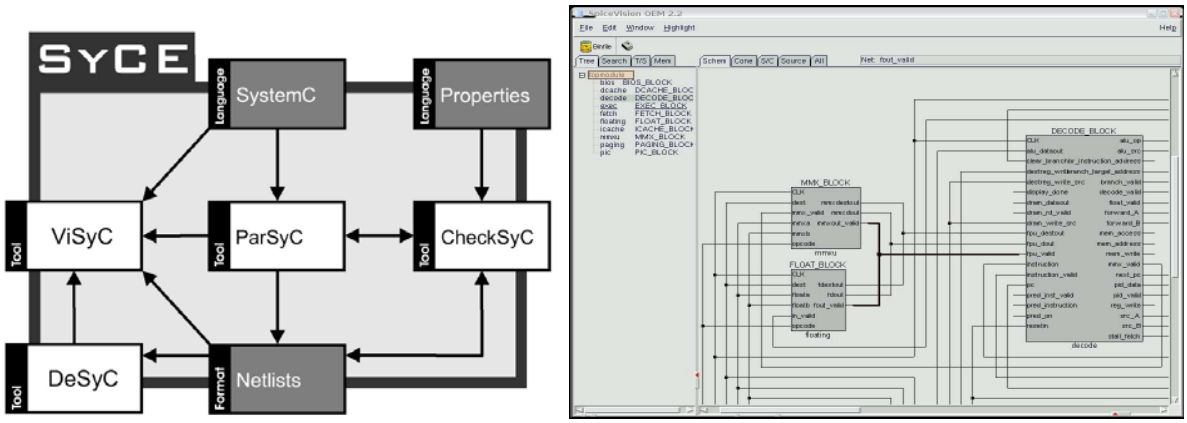

Figura 20 – Conjunto de ferramentas do SyCE (esquerda) e visualização gráfica do projeto (direita), [59].

Trata-se de um ambiente bastante interessante, porém, através de um contato direto (em setembro de 2006) com um dos seus desenvolvedores (Rolf Drechsler <drechsle@informatik.uni-bremen.de>), obteve-se a informação de que o projeto ainda não está finalizado e que não há uma *deadline* para sua finalização, porém, assim que os módulos estiverem prontos serão liberados para uso no site da universidade[9].

2.3.2.7 gSysC

gSysC [87] é uma GUI (*Graphical User Interface*) para SystemC baseada em Qt[10]. A aplicação desenvolvida por Eibl, Albrecht e Hagenau não altera a biblioteca SystemC muito menos o *kernel* de simulação. As características são introduzidas na interface gráfica através do uso de macros e de chamadas das funções providas por SystemC. A ferramenta se encontra disponível para *download* gratuitamente no site do Instituto de Engenharia da Computação da

[9] http://www.informatik.uni-bremen.de – University of Bremen – Faculty 03 – Computer Science

[10] Desenvolvido pela norueguesa Trolltech, Qt é um sistema multiplataforma para desenvolvimento de programas de interface gráfica, baseado em C++.

Universidade de Lübeck[11]. Na documentação da ferramenta não há referência ao suporte ao refinamento sucessivo.

A Figura 21 apresenta algumas telas do gSysC a partir de um projeto. A tela em primeiro plano é uma janela do controlador de simulação da ferramenta, enquanto que a tela em segundo plano é uma janela de observação de portas, onde define-se cada porta que será observada durante a simulação. Já a tela no plano mais baixo é uma janela de visualização da estrutura do projeto, de forma gráfica. A Figura 22 mostra a interação entre a ferramenta gSysC e o *kernel* do SystemC, onde o usuário não pode interagir com a simulação em execução, pois esta interação se dá através de arquivos de entrada e saída.

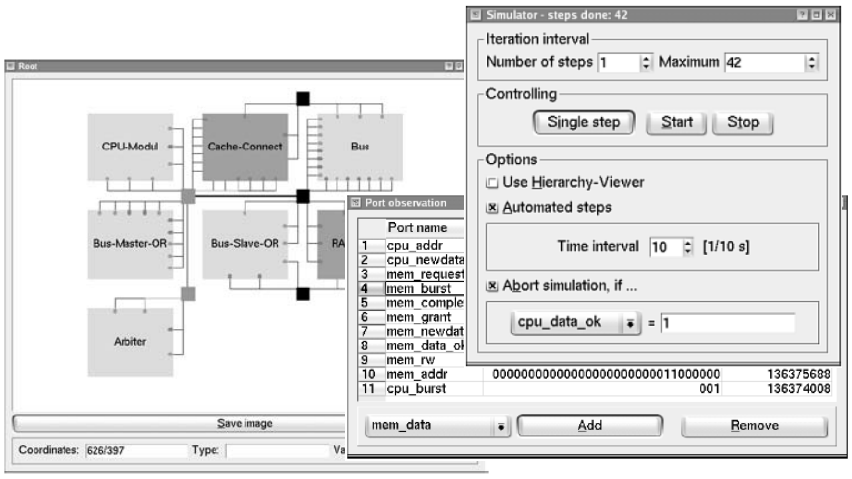

Figura 21 – Variadas telas do front end gSysC [87].

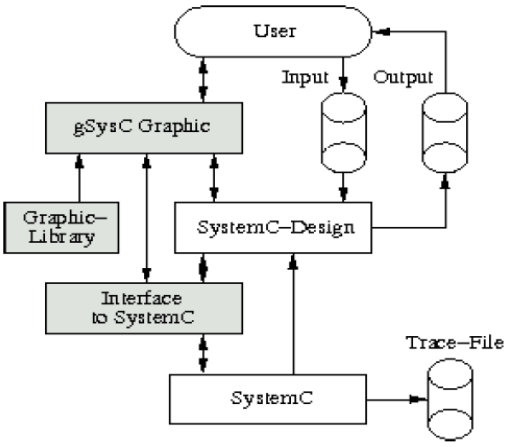

Figura 22 – Interação entre gSysC e SystemC [87].

[11] http://www.iti.uni-luebeck.de – Institute of Computer Engineering, University of Lübeck

2.3.3 Ferramentas de apoio ao projeto

2.3.3.1 YAML

Yet Another UML front end (YAML) [64] é uma ferramenta gratuita cujo objetivo é ajudar o projetista a conceitualizar e modelar um projeto em uma interface gráfica usando a notação UML estendida. Dessa forma, YAML foi desenvolvido para incluir tanto a semântica estrutural quanto a funcional no projeto, além dos objetos e seus relacionamentos, permitindo ao projetista conceitualizar e implementar seu projeto com mais facilidade utilizando a notação UML para isto. Na Figura 23 pode-se verificar a versatilidade da ferramenta através de diversas telas que permitem a modelagem a partir de um diagrama UML ou de um diagrama de blocos, com a inclusão dos relacionamentos e objetos no projeto, além da geração do código SystemC do projeto implementado no nível de abstração funcional. Uma abordagem similar é o uso de SysML (*Systems Modeling Language*) [65], para a modelagem de SoCs e NoCs (*Network-on-chip*), tratando-se de uma linguagem baseada em UML e permitindo a descrição do projeto, também, usando a notação UML.

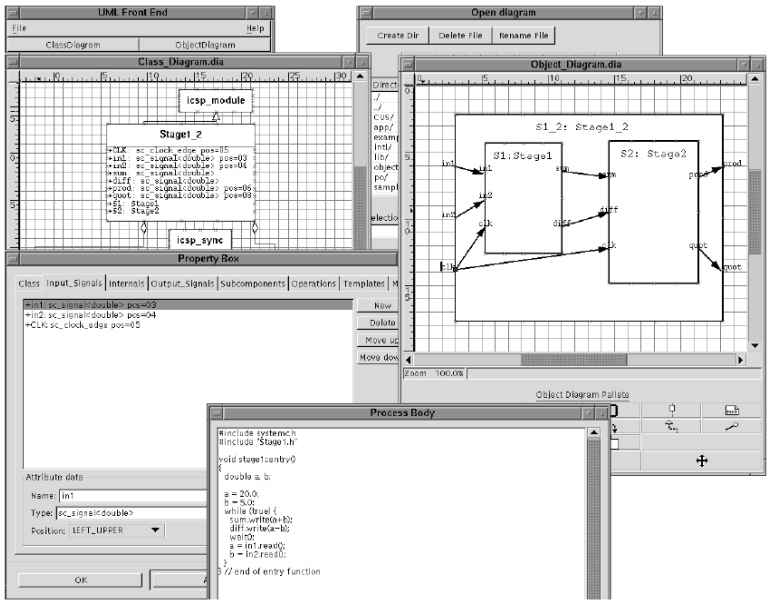

Figura 23 – Variadas telas da modelagem utilizando-se o YAML [64].

2.3.3.2 SystemCXML, SystemPerl, Pinapa

SystemCXML [54] é um projeto *open-source* mantido na SourceForge [55] desenvolvido por Berner et al com o objetivo de recuperar informações estruturais de modelos SystemC, usando tecnologias *open-source* como Doxygen e XML parser. Seu funcionamento está descrito na Figura 24.

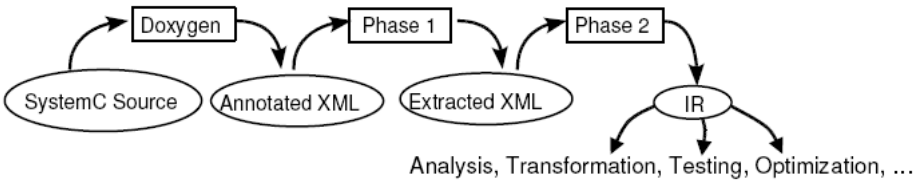

Figura 24 – Etapas do desenvolvimento usando o Parser SystemCXML [54].

A partir da união de todos os arquivos (.cpp e .h) do projeto em um único arquivo, este é processado pela ferramenta Doxygen, gerando um arquivo XML com a estrutura do código, formatada usando cores e tags de marcação. Na Phase 1, esta saída é então convertida em uma representação intermediária chamada *Abstract Syntax Language Definition* (ASLD), outro formato XML que captura toda a informação estrutural do código SystemC como hierarquia, portas, sinais, tipos, dentre outros. Já na Phase 2, a ASLD é convertida numa estrutura de dados que represente o que foi extraído do código.

Outros projetos similares a este são:

- SystemPerl [56], fornecido gratuitamente pela Veripool, segundo Berner, este projeto requer a intervenção no código fonte com o objetivo de direcionar a extração da estrutura. Consiste de bibliotecas de SystemC para Perl, possibilitando o uso das facilidades de tratamento de texto da linguagem Perl para a extração da estrutura do código SystemC.

- Pinapa [57], outro projeto disponível na SourceForge.net, baseado em GNU GCC, que, segundo Berner, possui como vantagem a extração da informação estrutural bem como da informação comportamental do código. Sua desvantagem, também segundo Berner comparando à SystemCXML, é a necessidade de alterações em bibliotecas

SystemC para o seu uso assim como modificações no compilador GCC, o que dificulta o uso de outros compiladores.

2.3.3.3 LusSy

Usando o Pinapa como parser, o GCC como compilador e as bibliotecas SystemC, dentre outros módulos, LusSy [66] é um ambiente para análise de SoCs descritos em SystemC, no nível transacional. A partir da extração de informação do código SystemC e da construção de um conjunto de autômatos paralelos que capturam a semântica do projeto incluindo as construções específicas do nível transacional, o resultado é uma semântica formal executável de SystemC, sendo submetida a ferramentas de verificação formal embutidas. Ferramentas como Lesar, Nbac e a linguagem Lustre, permitem a geração e visualização dos autômatos e a conexão entre eles, visando à verificação do código. A ferramenta é aberta e maiores informações podem ser obtidas no site do autor[12].

2.4 Considerações sobre o capítulo

Neste capítulo foram apresentadas diversas metodologias, linguagens e ferramentas utilizadas no projeto de sistemas de hardware. Dentre as metodologias, a de refinamentos sucessivos mostrou-se mais interessante para ser utilizada no desenvolvimento de sistemas de hardware. Para seu uso, a melhor linguagem é SystemC, que possui suporte nativo a esta metodologia, diferentemente das outras com as quais foi comparada. Dentre as ferramentas pesquisadas, consistindo de ferramentas necessárias para a síntese, desenvolvimento e apoio, não se encontrou alguma que desse suporte ao refinamento sucessivo, apenas algumas ferramentas que suportam o desenvolvimento em diversos níveis de abstração.

No próximo capítulo será abordada com mais detalhes a linguagem escolhida para este trabalho, SystemC e no capítulo seguinte, a Metodologia de Refinamentos Sucessivos.

[12] http://www-verimag.imag.fr/~moy – Matthieu Moy

3 SystemC

Após tornar-se o padrão IEEE 1666 [12], SystemC [21] confirma-se como uma das melhores opções para a descrição de hardware em diversos níveis de abstração. Por anos, projetistas de sistemas têm procurado uma forma de validar a possível implementação do hardware ao longo do desenvolvimento da arquitetura e do software, isto implica na necessidade de uma linguagem que possa suportar o projeto e a verificação do sistema a partir do conceito de implementação por hardware e software. SystemC atende a estes requisitos e por isso foi padronizada pelo IEEE como padrão de linguagem de projeto de chip no nível de sistema.

Além de linguagem, SystemC suporta metodologias de desenvolvimento consistindo na modelagem em elevados níveis de abstração com o uso do refinamento sucessivo para se alcançar o nível desejado para síntese. A simulação é permitida a qualquer momento do desenvolvimento do projeto e em qualquer nível, o que facilita o acompanhamento da funcionalidade do sistema em desenvolvimento além do desenvolvimento conjunto de hardware e software.

Além disso, a linguagem tem sido muito explorada tanto no desenvolvimento de novas ferramentas quanto na proposição de novas metodologias de projeto. Como exemplo, no ano de 2006, Chevalier [69] apresenta um ambiente de desenvolvimento para a linguagem enquanto Déharbe e Medeiros [70] apresentam uma metodologia de desenvolvimento usando a orientação a aspectos, inserindo uma nova visão da modelagem em SystemC.

A manutenção e incentivo do uso da linguagem são feitos pelo consórcio Open SystemC Initiative OSCI [43], que tem o objetivo de estabelecer um estilo de modelagem C++ comum para a indústria eletrônica, permitindo a troca e reúso de IPs no nível de sistema, além de

suportar o co-desenvolvimento e particionamento de hardware e software numa mesma linguagem. Diversas empresas adotam SystemC como sua linguagem de descrição de hardware, como Broadcom Corporation, Canon Inc., Freescale Semiconductor Inc., JEDA Technologies Inc., Intel Corporation, NEC Corporation, dentre muitas outras.

Nesta seção abordaremos alguns detalhes da linguagem e de seu fluxo de projeto.

3.1 Por que usar SystemC?

Apesar das linguagens bem estabelecidas para descrição de hardware como Verilog e VHDL já possuírem ferramentas e fluxos de projeto baseados nelas, o uso de uma linguagem única, desde a concepção do projeto até a sua síntese, facilita e reduz o trabalho do projetista. O documento "*Describing Synthesizable RTL in SystemC*" [85] apresenta uma análise do ponto de vista do projetista de hardware que recebe uma especificação em alto nível do projeto do sistema, chamada modelo arquitetural, descrito em SystemC. A especificação contém uma variedade de modelos de processador, modelos abstratos de barramentos e de periféricos, que capturam toda a funcionalidade do sistema, porém em um nível alto de abstração. Isto permite que o projetista de sistema possa manipular os modelos como uma especificação executável, facilitando a simulação e entregando ao projetista de hardware estes modelos juntamente com os requisitos de área, velocidade e consumo de energia para os periféricos.

O uso da linguagem SystemC possibilita um ganho enorme de produtividade, com uma maior velocidade de simulação além da possibilidade de síntese em diversos níveis. Associado a isso, a eliminação dos erros de interpretação e o reuso da verificação são outras características que fortemente endossam o uso da linguagem no projeto de hardware desde o nível mais alto ao mais baixo.

3.1.1 Erros de interpretação / Gap semântico

A partir de um projeto entregue ao projetista de hardware, uma das soluções possíveis que ele dispõe é reescrever os modelos, descritos inicialmente em uma linguagem de alto nível, como C, para Verilog ou VHDL, através de um processo passível de erros e que consome muito tempo, pois se trata da tradução de uma linguagem para outra. Outra solução é sintetizar o código entregue a partir da própria linguagem SystemC. Para tanto será necessário que o projeto inicial, num alto nível de abstração e, portanto, não sintetizável, seja refinado num modelo sintetizável. Isto evita desperdício de retrabalho, reescrevendo códigos. A comparação entre o modelo convencional de fluxo de projeto e o modelo adotado por SystemC encontra-se na Figura 25. Nela podemos perceber o retrabalho na reescrita do código escrito inicialmente em C/C++ para VHDL/Verilog. Os erros gerados nesta conversão manual caracterizam o gap semântico entre as linguagens, uma vez que muitas estruturas de dados e controle não são convertidas diretamente de uma para outra, necessitando alterações/adaptações no código reescrito. Por outro lado, o uso de SystemC elimina este gap semântico uma vez que a linguagem será a mesma do início ao fim do projeto.

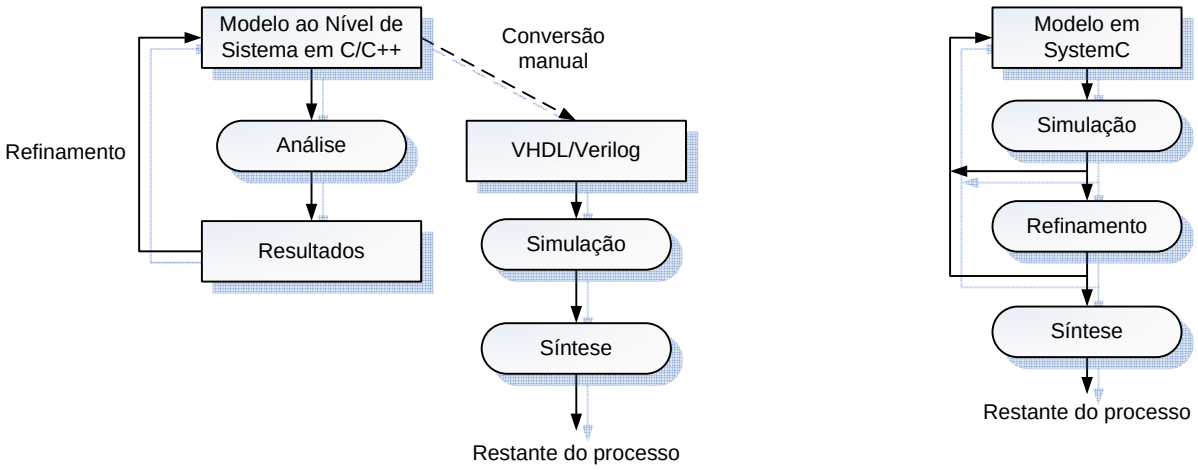

Figura 25 – Fluxo de desenvolvimento convencional (esquerda) e fluxo de desenvolvimento usando-se SystemC (direita), traduzida de [21].

3.1.2 Reúso de verificação

O refinamento do projeto ainda na mesma linguagem permite que todo o ambiente criado para verificação e teste possa ser utilizado nos vários níveis permitindo a coerência entre os diversos níveis de refinamento desde a especificação inicial até a síntese. Toda a estrutura de testbench criada para a especificação executável, mesmo que esteja num alto nível de abstração, poderá ser utilizada durante todo o projeto, inclusive no nível RTL (mais baixo). A possibilidade de se ter diversos níveis de abstração dentro do mesmo projeto reduz a necessidade de refinamento dos trechos de código do testbench, reduzindo o tempo total gasto no projeto.

3.1.3 Aumento de produtividade

Refinar o código SystemC é mais produtivo e mais rápido do que reescrevê-lo em outra linguagem como VHDL ou Verilog, isto por que a funcionalidade descrita é feita através de algoritmos ou máquinas de estado finitas. Algoritmos podem ser sintetizados a partir de ferramentas de síntese comportamental enquanto que máquinas de estados finitos podem ser sintetizadas a partir de técnicas de síntese RTL (nível mais baixo alcançado por SystemC). Além disso, a verificação de um projeto refinado em SystemC é mais rápida do que a verificação de uma recodificação de um projeto em outra linguagem. O refinamento, como já dito anteriormente, permite o uso das mesmas estruturas de verificação e testbenches durante todo o projeto, do mais alto nível até o mais baixo.

O aumento da produtividade também está associado à maior velocidade de simulação alcançada com níveis de abstração mais altos. A Figura 4 demonstra estes níveis e a crescente velocidade de simulação associada à proximidade do maior nível de abstração que é contrária à precisão de simulação do hardware final, maior quanto mais próximo do nível RTL.

3.2 Padrão IEEE 1666 e versões da linguagem

Em dezembro de 2005 o Instituto de Engenheiros Elétricos e Eletrônicos (IEEE) ratificou o padrão IEEE1666 [12], consistindo do Manual de Referência da Linguagem SystemC 2.1 (*Standard SystemC Language Reference Manual 2.1*). Segundo o IEEE, a descrição definitiva da linguagem SystemC expande a capacidade de modelagem de hardware e software para altos níveis de abstração. O Instituto ainda afirma que o padrão permite que engenheiros arquitetem todo o sistema desde o início, agilizando o processo de projeto, permitindo, inclusive o compartilhamento e reúso dos IPs (*Intellectual Property*, Propriedade Intelectual, módulos de projetos de hardware).

A partir da sua primeira versão lançada em 2000, a linguagem tem se aprimorado e o consórcio OSCI [43] disponibiliza as novas versões, atualizações e documentações em seu site. A Tabela 2 apresenta algumas versões da linguagem já disponibilizadas e outras que ainda estão em desenvolvimento, como a versão 3.0, que visa possibilitar não só uma modelagem de software como também a modelagem do escalonador do sistema operacional de tempo real, dentre outras características. A versão 4.0 é um projeto de melhoria e extensão da versão 3, objetivando incluir na modelagem os sinais analógicos e mistos.

Versão	Data de lançamento	Características
SystemC 1.0	2000	Refinamento do comportamento nos baixos níveis Preocupação com a descrição do hardware próximo do RTL Sensibilidade (ativação do processo) estática a sinais Conjunto fixo de canais de comunicação
SystemC 2.0	2001	Análise de performance Exploração do projeto Particionamento Hardware/Software Sensibilidade estática e dinâmica Conjunto expansível de canais de comunicação com o uso de canais definidos pelo usuário
SystemC 2.1	2004	Nova API de manipulação de erros Criação dinâmica de threads Alta compatibilidade com 2.0.1
SystemC 3.0	em desenvol.	Interrupção e aborto de threads Modelagem de software Modelagem do escalonador do sistema operacional de tempo real Reconfiguração dinâmica de hardware Especificação e verificação de restrições de tempo
SystemC 4.0	em desenvol.	Modelagem de sinais analógicos/mistos Adição de bibliotecas para facilitar o desenvolvimento de testbenches

Tabela 2 – Algumas versões de SystemC e suas características principais.

3.3 A Linguagem

SystemC é uma biblioteca de classes C++ que permite a criação de modelos com precisão de ciclo a partir de algoritmos de software, arquiteturas de hardware e interfaces entre SoCs e projetos no nível de sistema. Por ser baseada em C++, ferramentas de desenvolvimento desta última podem ser utilizadas para o desenvolvimento de projetos em nível de sistema, simulação rápida para validação e otimização do projeto, exploração de diversos algoritmos e elaboração de uma especificação executável do projeto. Uma especificação executável é um programa C++ que exibe o mesmo comportamento do sistema final quando executado.

SystemC tem todas as construções necessárias para suportar o projeto a partir do nível do sistema até o nível de blocos lógicos. Outra vantagem é a possibilidade de descrição dos detalhes tanto do nível de sistema quanto do nível de blocos lógicos com a mesma linguagem. Além disso, existe também a possibilidade de desenvolvimento tanto do software quanto do hardware embutido na mesma linguagem. Isto permite que as decisões do particionamento hardware/software sejam tomadas no momento oportuno do projeto.

A base C++ permite a introdução de modularidade, como o uso de classes, herança, dentre outras características da linguagem, o que possibilita o desenvolvimento de IPs comportamentais, ou seja, que descrevem o comportamento do sistema com uma boa precisão, próxima do mais baixo nível. A separação de comunicação e funcionalidade, permitida por SystemC, modelos de comunicação abstrata (modelos do nível transacional – transaction-level models) podem ser sintetizados em outros modelos usando protocolos de barramento detalhados. Com a síntese comportamental, algoritmos abstratos podem ser sintetizados em implementações RTL otimizadas. Tudo isto se torna possível devido à capacidade de SystemC de possibilitar ao projetista implementar os seus projetos nos diversos níveis abordados na Figura 4 (Capítulo 2, Seção 2.1.3.1 - Níveis de Modelagem). Desta forma, o usuário pode modelar um sistema em vários níveis de abstração [73]. No mais alto nível, somente a funcionalidade do sistema deve ser modelada, já no nível mais baixo, existe a possibilidade de síntese direta do sistema em um SoC a partir do nível RTL [71], do nível comportamental [72], ou até mesmo de outros níveis [3], através de ferramentas de síntese específicas. Para implementação de hardware, modelos podem ser escritos num estilo funcional ou RTL. O software do sistema poderá também ser escrito em C++.

Interfaces entre hardware e software e entre blocos de software podem ser descritas em níveis de precisão de transação ou precisão de ciclo. De qualquer forma, diferentes partes do sistema podem ser modeladas em diferentes níveis de abstração e estes modelos podem coexistir durante a simulação do sistema [74]. Classes C++ e SystemC podem ser usadas não somente para o desenvolvimento do sistema, mas também para a geração dos testbenches (plataformas para teste do sistema durante a simulação).

Black e Donovan [83] descrevem SystemC não como uma linguagem, mas como uma biblioteca de classes definida sobre uma linguagem bem estabelecida, que é C++. Afirmam ainda que SystemC não é uma panacéia que irá resolver todos os problemas de produtividade do projeto, porém possui um escopo que permite o uso dessa linguagem em diversos âmbitos do projeto, como descrito na Figura 26.

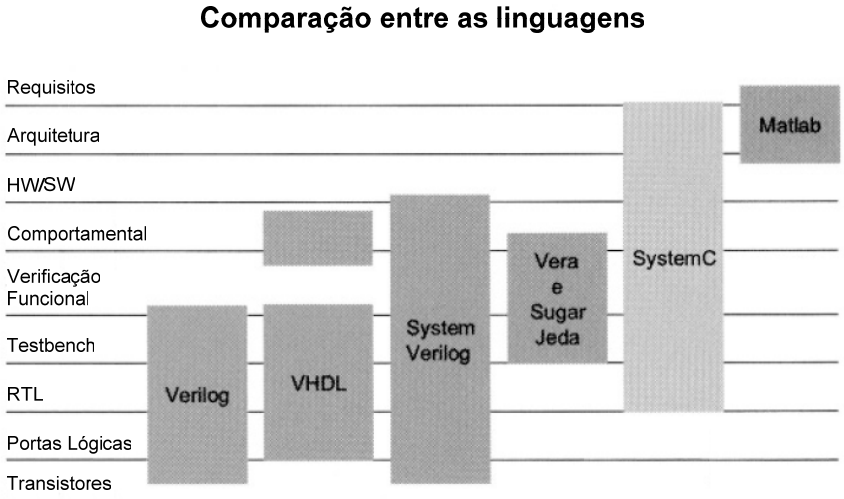

Figura 26 – Comparação de SystemC com outras linguagens, segundo Black e Donovan, traduzida de [83].

3.3.1 Características

SystemC suporta o co-desenvolvimento hardware-software da arquitetura de sistemas complexos, consistindo de componentes de hardware, software e interfaces, descritos na mesma linguagem. As seguintes características (Figura 27) permitem que este desenvolvimento concorrente aconteça:

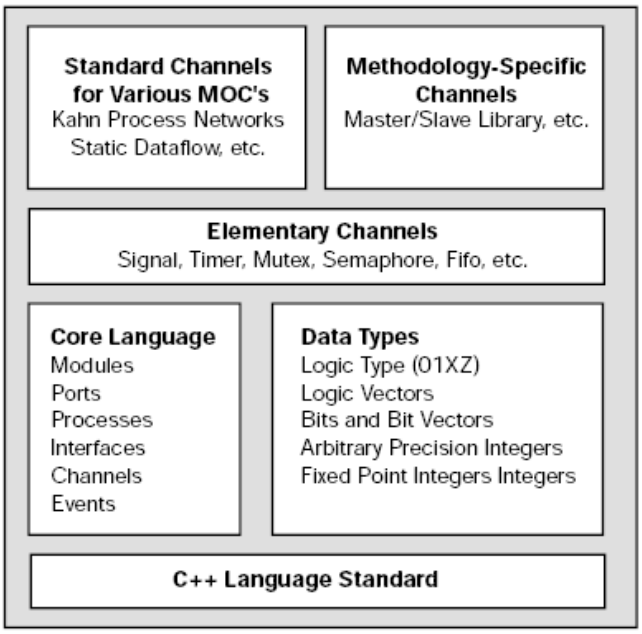

Figura 27 – Características suportadas por SystemC, [21].

- Módulos: Entidade hierárquica que pode conter outros módulos ou processos dentro dela. Contém canais e portas para comunicação entre módulos, que pode ocorrer através do acesso direto a estas estruturas ou através do uso de interfaces de comunicação.

- Processos: Usados para descrever funcionalidade. Localizam-se dentro de módulos. A linguagem provê três diferentes tipos de abstração de processos que podem ser usados para modelar hardware ou software.

- Sinais: Suporte a sinais que podem ter um ou mais de um driver ou barramento. Além disso, SystemC possui um vasto conjunto de tipos de sinais e portas que podem ser usados nos diferentes níveis de abstração desde a especificação executável até o RTL. Esta característica a difere de linguagens como Verilog que suportam, somente, tipos bit e bit-vector para sinais e portas. SystemC ainda suporta tipos de sinais *two-valued* e *four-valued*.

- Tipos de dados: O conjunto de tipos de dados da linguagem é muito vasto, possibilitando os múltiplos domínios de projeto e níveis de abstração. Além de herdar os tipos de dados de C++, a linguagem define um sub-conjunto de tipos de dados sintetizáveis. Tipos de dados de precisão fixa possibilitam a simulação rápida, tipos de precisão arbitrária podem ser usados para computações de números muito grandes e tipos de ponto fixo podem ser

usados para aplicações DSP. Há suporte, ainda, para tipos two-valued e four-valued para tipos de dados e não há limitação de tamanho para tipos de precisão arbitrária.

- Clock: Há suporte à noção de clock, como sinais especiais, bem como múltiplos clocks, que agem como controladores/sincronizadores durante a simulação.

- Simulação baseada em ciclo: A linguagem inclui um *kernel* de simulação muito leve que permite uma simulação muito rápida, baseada em eventos distintos.

- Múltiplos níveis de abstração: SystemC suporta desde modelos sem precisão de tempo em diferentes níveis de abstração, variando dos mais altos modelos funcionais aos modelos RTL com precisão de ciclo de clock, possibilitando o refinamento interativo de um nível mais alto para outro mais baixo, através da agregação, transformação entre os níveis.

- Protocolos de comunicação: Uma semântica de comunicação multi-nível permite a descrição de SoCs e protocolos de sistemas de I/O em diferentes níveis de abstração.

- Suporte à depuração: As classes SystemC possuem um verificador de erro em tempo de execução que pode ser ativado através de um flag na compilação.

- Rastreamento de forma de onda: O rastreamento de formas de onda usando-se formatos VCD, WIF e ISDB são possíveis nesta linguagem, isto facilita a observação de sinais, na forma de diagramas de tempo, em diversos pontos do sistema.

- Concorrência: Possibilidade de descrever processos concorrentes, síncronos ou assíncronos.

3.3.2 Orientação a aspecto usando SystemC

Diversos paradigmas de programação de software estão migrando e sendo incorporados nas linguagens de descrição de hardware. A Programação Orientada a Aspecto (AOP) [70] é um

novo paradigma que provê novas construções modulares no topo da orientação a objeto em linguagens como Java e C++.

Segundo Déharbe e Medeiros [70], a abordagem da AOP é baseada na idéia de que a abordagem estruturada e a abordagem orientada a objeto não facilitam o uso de alguns aspectos do sistema, como geração de log, restrições de tempo real, sincronização, dentre outros, que não podem ser encapsulados em uma unidade simples, como uma classe, um método, ou outro, de uma forma clara. Esta limitação do paradigma tradicional leva a uma dificuldade extra para entender, manter e reusar o código onde estes aspectos se tornam relevantes. Usando a abordagem AOP, um sistema pode ser projetado de tal forma que o código com diferentes objetivos seja encapsulado em diferentes unidades que podem, posteriormente, ser compostas seguindo regras específicas. Para o uso de AOP com SystemC, os autores adotaram a ferramenta AspectC++ [82], que possibilita o uso de AOP com C++.

Este novo paradigma de programação demonstra o quanto as linguagens de descrição de sistemas de hardware e metodologias se aproximam das linguagens e metodologias de desenvolvimento de sistemas de software, justificando o uso de uma linguagem comum, como SystemC, para o desenvolvimento tanto do hardware como do software do sistema final.

3.4 Fluxo de projeto usando SystemC

Um projeto em SystemC consiste de um conjunto de arquivos de descrição de classes e uma biblioteca de vínculo que contém o *kernel* (núcleo) de simulação. Qualquer compilador de C++ padrão ANSI pode compilar SystemC. Durante a compilação, é usada a biblioteca do SystemC, que contém o *kernel* de simulação, gerando ao término um código executável contendo o simulador do sistema descrito. O projeto ainda pode ser depurado utilizando-se um depurador padrão de C++ e sinais gerados durante a simulação podem ser rastreados para um arquivo (*trace file*) para ser visualizado em formas de onda, conforme Figura 28, adaptada de [45]. O resultado obtido na forma de onda serve para verificar se a saída gerada está de acordo com a especificação, ou seja, se o hardware ali descrito gerará os sinais corretos quando for sintetizado.

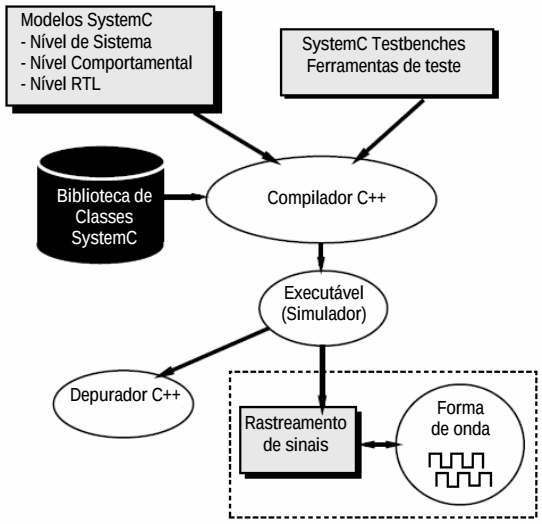

Figura 28 – Simulação em SystemC, adaptada de [45].

A característica mais importante do fluxo de implementação em SystemC é o fato de que a especificação é escrita em uma linguagem comum tanto para a parte do software quanto para o hardware. Isto elimina a necessidade da ferramenta de projeto do sistema ter de entender e analisar a sintaxe e semântica de ambientes de modelagem distintos [73].

A Figura 29 mostra um Fluxo de Síntese/Implementação em SystemC. Um projeto pode partir de uma especificação em alto nível (nível funcional) sendo refinada para um nível transacional, ponto onde ocorre a divisão entre o que será implementado em software e o que será implementado em hardware. O próximo nível de refinamento é o nível comportamental, onde também pode ocorrer a síntese do projeto ou o refinamento para o próximo e último nível, RTL, onde ocorre a síntese final e a geração da *netlist*, ou seja, o mapeamento físico dos componentes em portas lógicas.

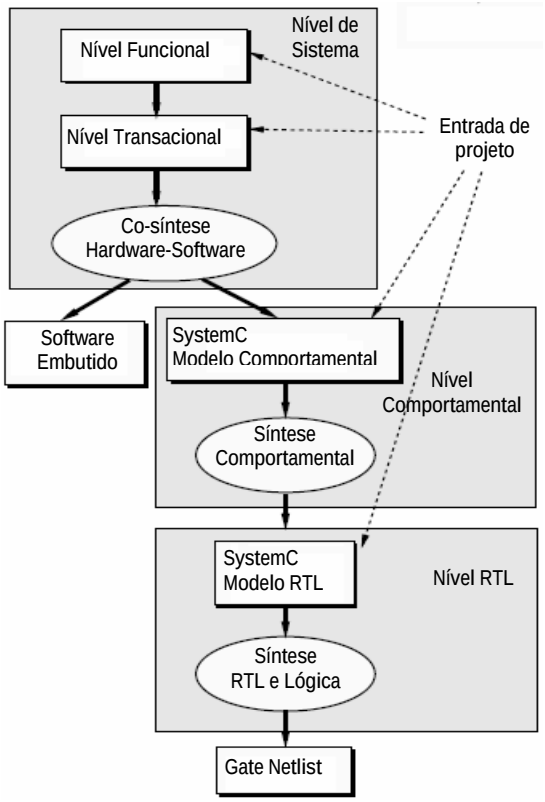

Figura 29 – Fluxo de síntese e implementação em SystemC, adaptada de [45].

Um testbench pode ser usado para validar modelos em diferentes níveis de abstração, permitindo que o mesmo vetor de testes seja submetido ao projeto em qualquer nível, seja ele System, Behavioral ou RTL. O nível transacional (*Transaction Level*), por tratar de uma modelagem da funcionalidade com comunicação entre os módulos, é muito estudado e usado para descrição de hardware e software [74] [75] [76] [77] [78] [79] [80] [81], sendo o ponto para o refinamento para o nível comportamental (*Behavioral Level*).

4 Metodologia de Refinamentos Sucessivos

O uso da metodologia de refinamentos sucessivos é crescente e, como explorado no Capítulo 2, muitas linguagens, metodologias e ferramentas surgiram e/ou foram adaptadas para facilitar o uso desta metodologia. Seu foco principal é trabalhar com níveis de abstração diferentes do sistema de forma a facilitar e agilizar o desenvolvimento do projeto. Isto é possível com a utilização de linguagens de descrição de hardware baseadas em C/C++, pois facilitam o co-desenvolvimento hardware-software, ou seja, o desenvolvimento do software antes mesmo que o hardware esteja finalizado em um protótipo físico testável. Este ganho em produtividade é bem demonstrado na Figura 30, um gráfico da *Toshiba SpecC Technology Open Consortium* [36], que apresenta o tempo gasto com depuração de projetos usando as metodologias comuns em comparação com a metodologia de Refinamentos Sucessivos. Normalmente a maior parte do tempo de depuração é gasto com correções de especificações traduzidas de forma incorreta ou de forma ambígua, estes problemas são facilmente diminuídos utilizando-se refinamentos, pois se utiliza a mesma linguagem desde a especificação até o nível de síntese.

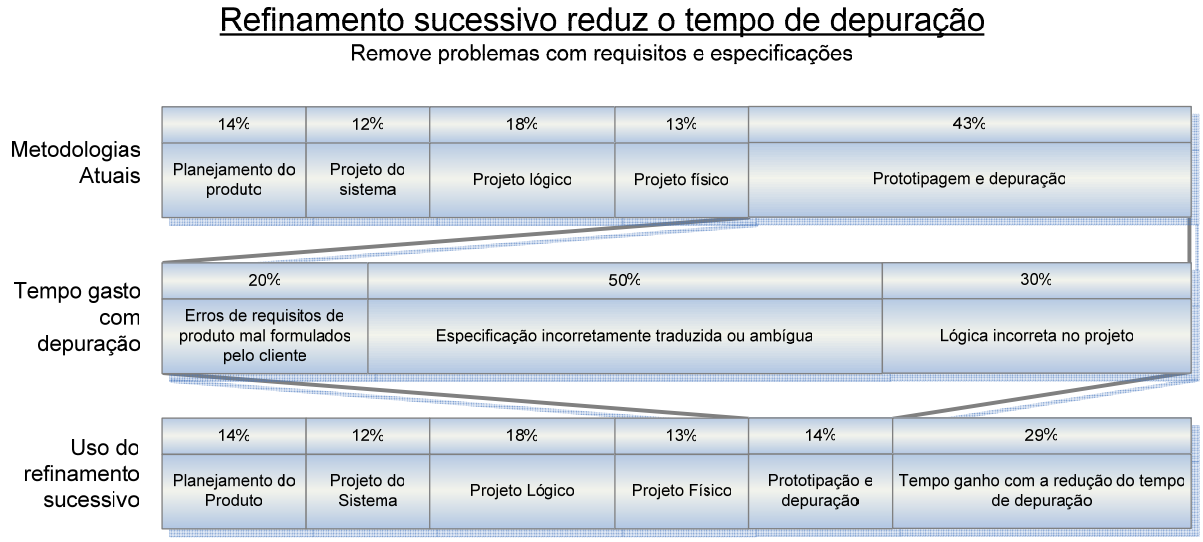

Figura 30 – Redução de tempo de depuração com o uso de refinamentos sucessivos, traduzida de [36].

Abstração é uma técnica poderosa para o projeto e a implementação em hardware de sistemas complexos. Ela nos permite enfrentar a complexidade escondendo detalhes desnecessários para poder lidar com os mesmos mais tarde. Diferentes quantidades de detalhes correspondem a diferentes níveis de abstração, de acordo com Müller, Rosenstiel e Ruf [8]. A semântica da modelagem baseada na abstração é bem descrita por Gerstlauer e Gajski [10], onde ressaltam a importância do seu uso e as vantagens advindas dele. Abdi e Gajski [13] desenvolveram um formalismo matemático, chamado *Model Algebra*, para ser usado para representar os modelos de sistemas e suas transformações entre os níveis de modelagem a partir dos refinamentos. Esse formalismo cria as transformações corretas dos modelos e garante sua correção através dos seus axiomas, teoremas e provas.

A Figura 31, extraída de [13], mostra como um modelo de especificação de sistema (*Specification Model*) é refinado em um modelo de arquitetura de sistema (*Architecture Model*). O comportamento de cada componente pode ser compilado em um código assembly (para componentes de software) ou sintetizado em RTL (para componentes de hardware). Os canais abstratos de dados podem ser sintetizados em barramentos de sistema e as interfaces de barramentos sintetizadas em componentes, através de refinamentos de comunicações. Esses fluxos análogos são mostrados na Figura 32, extraída de [10] e complementar à figura anterior, onde os fluxos análogos mapeiam a computação e a comunicação em direção à arquitetura para síntese (*System Architecture*). As etapas intermediárias desse mapeamento são os níveis de abstração por onde serão refinados os projetos.

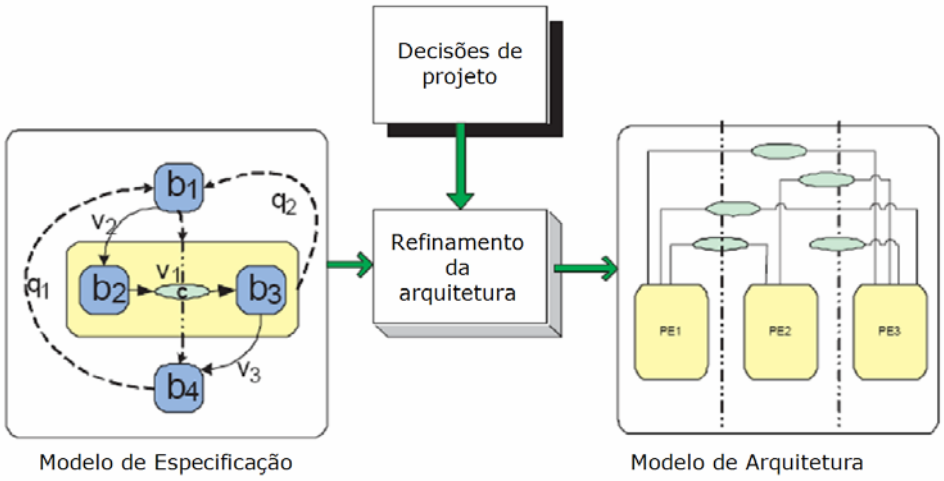

Figura 31 – Arquitetura de refinamento, [13].

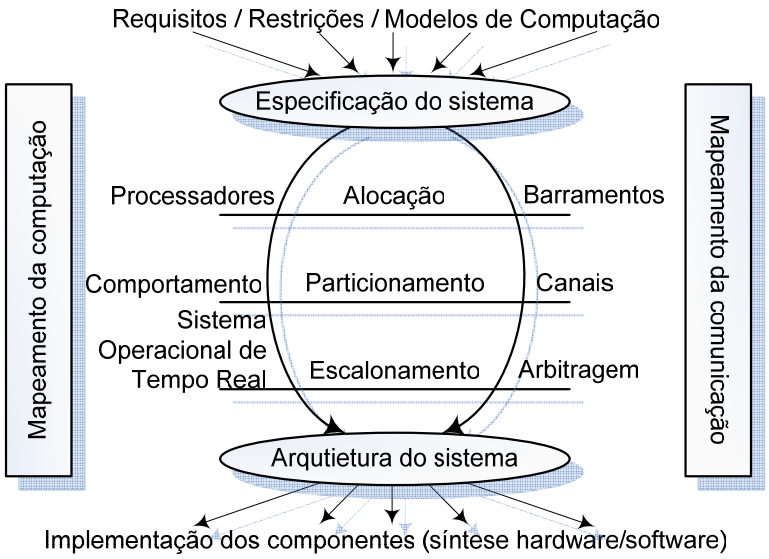

Figura 32 – Tarefas de projeto de sistemas, traduzida de [10].

Schutten [11] aborda a metodologia de aumento dos níveis de abstração com conseqüente redução da necessidade de verificação do SoC, redução do tempo gasto com o projeto além de ganhos em melhoria dos resultados obtidos. Soma-se a isso, segundo Schuten, a possibilidade de se antecipar o desenvolvimento concorrente de hardware e software.

Usando a linguagem SystemC, o projeto não necessita ser convertido, através de um grande esforço, de uma descrição em C para uma linguagem de descrição de hardware como Verilog ou VHDL. O projeto é lentamente refinado em pequenas seções para adicionar as construções de hardware e tempo em direção ao objetivo final do sistema. Usando a metodologia de refinamentos sucessivos, o projetista pode facilmente implementar mudanças no projeto e,

assim, detectar e solucionar problemas durante o refinamento. O uso de *testbenches* num nível diferente do nível do sistema também é permitido, o que facilita o seu reúso durante o refinamento.

4.1 GCD de Euclides

Para demonstrar os níveis da metodologia, será implementado e refinado o código do GCD (*Greatest Common Divisor*, Máximo Divisor Comum) baseado no algoritmo Euclidiano.

O algoritmo de Euclides, como pode ser visto no fluxograma da Figura 33, consiste no seguinte: admitindo que A é maior que B, a determinação do máximo divisor comum baseia-se no cálculo do resto da divisão de A por B; sempre que o resto é diferente de zero, procede-se à substituição de A por B e de B pelo resto obtido, voltando a executar-se o cálculo do resto da divisão; quando o resto for nulo, então o máximo divisor comum é igual a B. Nesse algoritmo, o cálculo do resto pode ser um agravante e pode ser facilmente resolvido através de subtrações sucessivas, como na Figura 33. Este algoritmo implementado encontra-se na Figura 34.

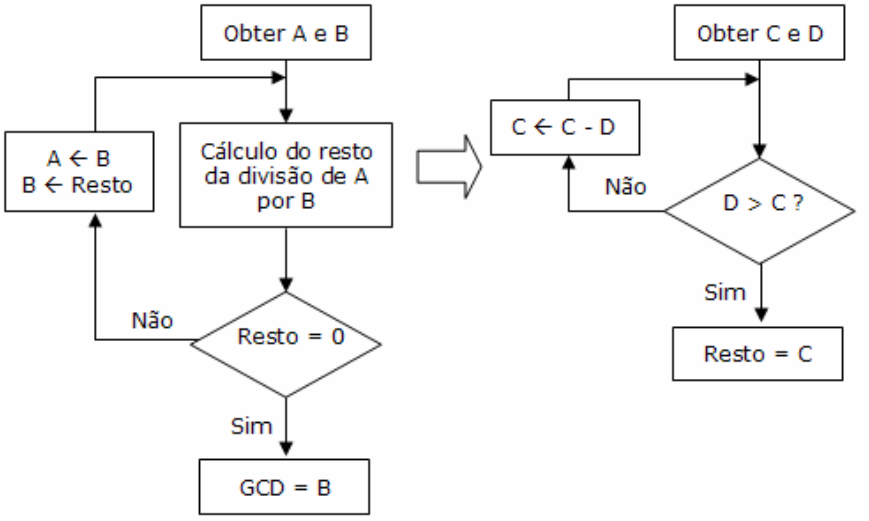

Figura 33 – Algoritmo de Euclides (esquerda) e cálculo do resto (direita).

```
#include <iostream>

int GCD(int a, int b) {
    while( 1 ) {
       a = a % b;  if( a == 0 ) return b;
       b = b % a;  if( b == 0 ) return a;
    }
}

int main() {
       int x, y;
       cout << "This program allows calculating the GCD\n";
       cout << "Value 1: ";  cin >> x;
       cout << "Value 2: ";  cin >> y;
       cout   << "\nThe Greatest Common Divisor of "
              << x << " and " << y << " is " << GCD(x, y) << endl;
       return 0;
}
```

Figura 34 – GCD implementado em C++.

4.2 Testbenches

A implementação aqui utilizada será baseada na utilização de testbenches (estruturas de teste) desde a especificação executável (primeiro nível). O uso de testbenches, conforme Figura 35, permite a validação do projeto em cada etapa de refinamento a partir da comparação dos valores (resultados) obtidos em cada etapa com os obtidos anteriormente. A estrutura de um testbench consiste no uso de um gerador de estímulos (stimulus) e um receptor (monitor), além do dispositivo a ser testado (DUT – *device under test*). O gerador de estímulos gera as entradas do DUT de acordo com as especificações do projeto, a cada processamento os valores obtidos pelo receptor podem ser comparados com os valores obtidos em outra etapa de refinamento a fim de se observar se o sistema mantém o mesmo comportamento e funcionalidades. O DUT é o módulo a ser refinado em cada iteração e pode necessitar de outro refinamento no mesmo nível ou de correções de acordo com o resultado obtido no receptor do testbench. O uso de testbenches reduz a quantidade de erros em projetos além de facilitar o fluxo contínuo dos refinamentos rumo à implementação alvo.

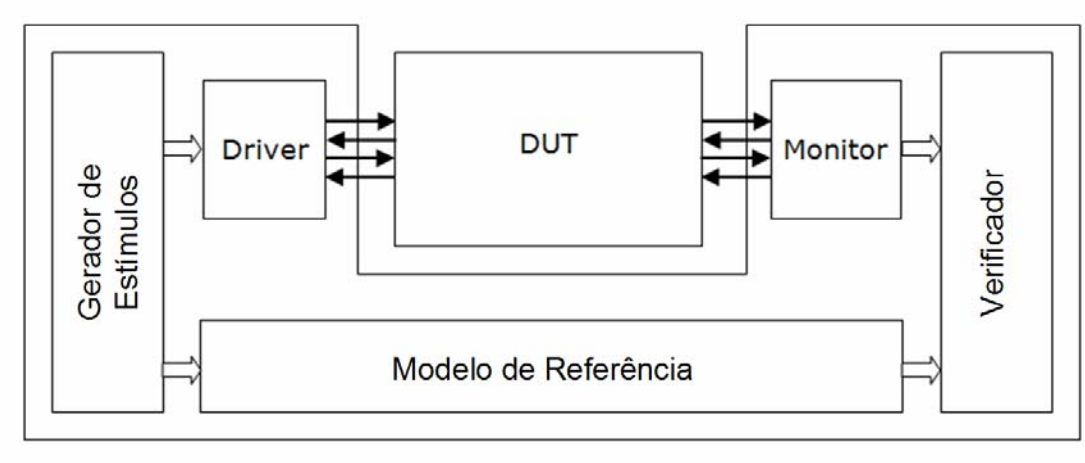

Figura 35 – Modelo genérico de um testbench.

A comunicação entre os módulos DUT e o testbench se dá através do uso de drivers, de forma que o dado a ser utilizado pelo testbench seja convertido no tipo exato a ser utilizado na implementação do módulo. Sendo assim é possível utilizar um valor inteiro desde o início do projeto e este valor ser convertido em diferentes estruturas de dados utilizadas ao longo dos refinamentos dependendo da necessidade do projeto e/ou do nível em que se está refinando. Assim, o tipo inteiro da linguagem C pode ser convertido para um tipo inteiro do SystemC, ou um tipo bit-vector ou qualquer outro necessário, sem que se tenha que alterar o dado de entrada, bastando fazer a alteração no driver específico que se comunica com o módulo que se quer alimentar.

A Figura 36 detalha a estrutura do testbench baseada nos módulos de implementação do projeto, sendo um arquivo ".cpp" e um ".h" para cada módulo a ser implementado. Isto facilita o refinamento bem como a implementação do projeto, pois divide cada módulo em arquivo de configuração e declaração (dut.h, por exemplo) e arquivo de processamento, onde são implementadas as funções e comportamentos do módulo (dut.cpp, por exemplo).

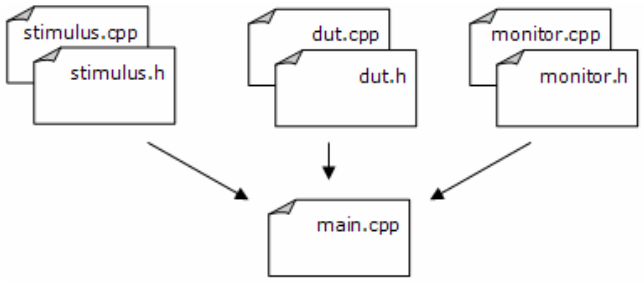

Figura 36 – Estrutura de arquivos de um testbench, [17].

4.3 Níveis de Modelagem propostos

Para o desenvolvimento deste trabalho, será proposto um modelo tomando-se como base todos os outros aqui já descritos e, cujos níveis serão melhores detalhados a seguir. Esta definição de níveis se faz necessária para delimitação do escopo do trabalho e está detalhada na Figura 37. Consiste numa especificação executável, sendo refinada para um nível funcional, com inserção de uma temporização aproximada, que por sua vez, é refinado para um nível comportamental que é o nível anterior ao RTL. Nos três primeiros níveis temos uma descrição e refinamento transacional (TLM) ocorrendo simultaneamente ao refinamento da funcionalidade. As regras de refinamento e decisões de implementação aqui apresentadas foram compiladas a partir dos trabalhos de [15], [16], [17], [18], [19], [20], [21], [25] e [94].

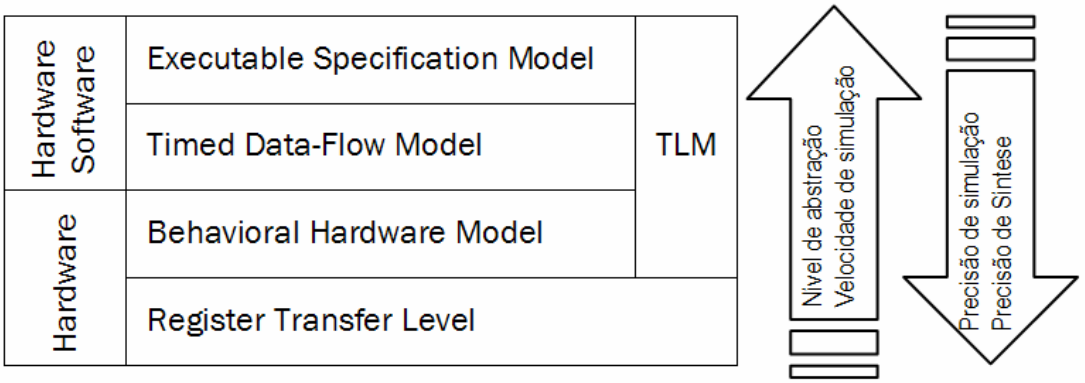

Figura 37 – Níveis de refinamento e sua relação com velocidade e precisão.

Juntamente com o refinamento nível a nível descrito a seguir, ocorrerá o refinamento TLM, sendo descritas a cada nível as regras para estes refinamentos.

4.3.1 Relação entre os níveis

Níveis de abstração mais altos implicam em maior velocidade de simulação. Isto porque um nível de abstração mais elevado implica em menos detalhes da implementação do hardware que devem ser descritos, gerando um código mais simples e de execução mais rápida. Por outro lado, quanto mais próximo do nível RTL, maior será a precisão da simulação e a capacidade de síntese, além de haver um atraso maior na simulação por refletir o

funcionamento do hardware final em todos os seus detalhes. Esses detalhes aumentam, desde a especificação executável, a cada refinamento em direção ao RTL.

Métrica \ Níveis	TLM			RTL
	ESM	TDM	BHM	
Funcionalidade	Sim	Sim	Sim	Sim
Atraso da computação	Não	Aprox	Aprox	Ciclo
Atraso da comunicação	Não	Aprox	Ciclo	Ciclo
Protocolo de comunicação	Não	Aprox	Exato	Exato
Estrutura	Não	Aprox	Aprox	Exato
Pino	Não	Não	Sim	Sim

Tabela 3 – Precisão dos níveis de modelagem em SystemC.

A Tabela 3 mostra os níveis descritos anteriormente e as métricas de precisão para cada um, comparando-os. A Tabela 4 mostra os pontos de modelagem de cada nível e suas características. O nível mais alto, ESM (*Executable Specification Model*) não é preciso, mas descreve toda a funcionalidade do sistema (Tabela 3), usando canais primários para comunicação de dados abstratos, usando classes tipo T, (Tabela 4). O nível TDM (*Timed Data-Flow Model*) introduz, no código da especificação executável, a temporização aproximada para a computação, para a comunicação, bem como um algoritmo de comunicação que implementa um protocolo para comunicação entre os módulos (Tabela 3), mas ainda não há precisão de pino, pois os dados são de tipos abstratos e usam-se canais hierárquicos (Tabela 4). Já o nível BHM (*Behavioral Hardware Model*) insere maior precisão, no nível de ciclo, na temporização da comunicação além de usar um protocolo exato para a comunicação (Tabela 3) e possuir precisão de pino/bit (Tabela 4). Por outro lado, o nível mais baixo, RTL (*Register Transfer Level*), é preciso em todas as métricas da Tabela 3, inclusive a funcionalidade, possuindo precisão de pino/bit (Tabela 4). O nível transacional (*Transactional Level Model* – TLM) compreende desde o nível mais alto (ESM) até o nível comportamental (BHM). O TLM é responsável pelo desenvolvimento/refinamento da comunicação entre os módulos do sistema.

Os passos do refinamento nesta metodologia são mostrados na Figura 38. O primeiro modelo (A), ESM, não é preciso no nível de tempo em termos da comunicação e da computação, sendo refinado para um nível com precisão aproximada de tempo tanto da comunicação quanto da computação, (B) TDM. Este por sua vez é refinado para um modelo comportamental com uma precisão na comunicação no nível de ciclo, BHM (C). Logo em

seguida o refinamento leva ao último, o RTL (D), que é preciso em termos de comunicação e computação.

		Níveis	Computação	Comunicação	Tempo	Dado	Processo
C++ / SystemC	TLM	ESM	funcionalidade	canal primário `sc_fifo`	evento	classe T	`SC_THREAD`
		TDM	elementos de processamento	canais hierárquicos interfaces	aproximado `wait()`	tipos abstratos	`SC_THREAD`
		BHM	estruturas sintetizáveis	protocolo adaptadores	ciclo	tipos definidos/bit	`SC_CTHREAD`
		RTL	lógica controle FSM / datapath	pino/bit `sc_signal`	ciclo	sinal/bit sintetizável	`SC_METHOD`

Tabela 4 – Modelagem da computação e comunicação para cada nível.

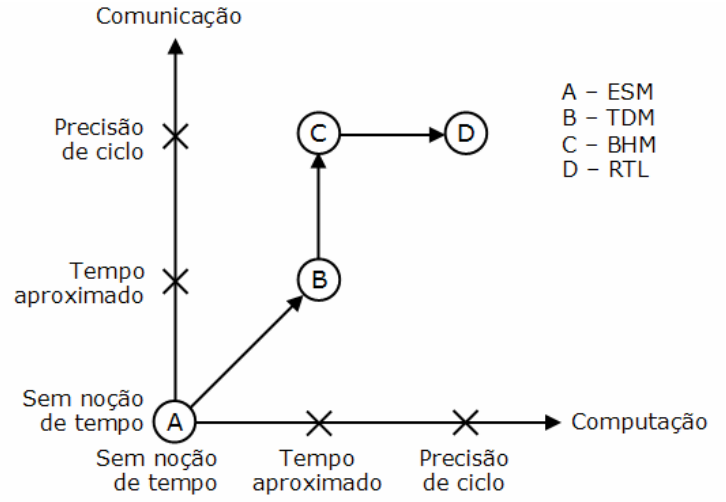

Figura 38 – Etapas de refinamento entre os níveis.

4.3.2 Executable Specification Model

A especificação executável representa a tradução direta da especificação do projeto em SystemC, de modo que ela é completamente independente da implementação alvo. Pode-se dizer que ela representa uma forma de mostrar ao cliente se o que os projetistas entenderam é o que ele realmente quer, além de funcionar como referência para os outros níveis de abstração.

Este nível não implementa atraso de tempo e a comunicação entre módulos é ponto-a-ponto (utilizando-se FIFOs, por exemplo), de uma forma abstrata com o uso de classes T (estrutura do SystemC), que podem transportar qualquer valor de qualquer tipo, por exemplo. O uso de fifos dá ao projetista liberdade para se preocupar com a especificidade/funcionalidade do código sem que tenha que se preocupar com o algoritmo de comunicação. Esta comunicação entre os módulos deverá ser tratada no momento em que o protocolo de comunicação tenha que ser definido, no nível BHM (Behavioral Hardware Model). Os tipos de dados são também abstratos. Pelo alto grau de abstração deste nível e pela falta de especificidade no código em relação ao objetivo final do projeto (inexistência de precisão de ciclo, de pino, de tempo) este nível não pode, portanto, ser sintetizado.

Trata-se de um modelo conceitual do sistema, dirigido a eventos e sem comportamento temporal, ou seja, as respostas do sistema não refletem a implementação final em termos de tempos de processamento, comunicação, dentre outros. Somente o comportamento do sistema é capturado e diversos algoritmos podem ser explorados para isso. Não há preocupação, neste nível se o componente será implementado em hardware ou software, preocupa-se somente com o comportamento do sistema como um todo.

Dessa forma, o diagrama dos módulos do GCD para o nível de especificação executável se encontra na Figura 39, consistindo do módulo GCD (dut) e os módulos stimulus e monitor, compondo o testbench do módulo GCD.

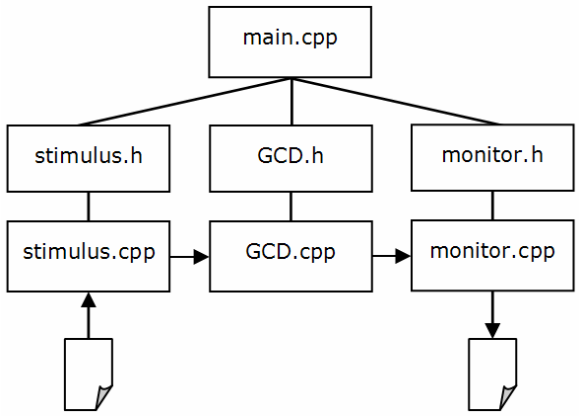

Figura 39 – Diagrama de módulos do GCD para o nível de especificação executável.

4.3.2.1 Decisões de Implementação

Segundo Walstrom [19] o primeiro passo para o desenvolvimento de uma especificação executável é determinar quais funcionalidades do sistema deverão ser implementadas através de processos distintos. A granularidade dos processos pode variar, mas cada processo deve ser independente, podendo-se ter mais de um processo dentro de um módulo. Os processos podem ser de três tipos: `SC_THREAD`, `SC_CTHREAD` e `SC_METHOD`, sendo que no presente nível (mais alto), todos os processos devem ser instanciados como processos `SC_THREAD`. Isto porque este tipo de processo possui a capacidade de interromper a execução usando diretivas `wait()` ou outras formas de bloqueio de execução, diferentemente dos processos `SC_METHOD` que executam do início ao fim sem interromper a execução, passando em seguida o controle ao *kernel* de simulação do SystemC. Já os processos `SC_CTHREAD` são um caso especial dos processos `SC_THREAD`, sendo que são ativados em apenas uma das bordas do clock, mapeando o comportamento do hardware implementado por ferramentas de síntese.

Toda a comunicação e sincronização entre módulos devem ser implementadas utilizando-se o canal primitivo `sc_fifo`, como descrito por [19], [96], dentre outros. Este canal pode ser acessado utilizando-se os métodos `read()` e `write()`, que são bloqueadores, ou seja, se um método `read()` é chamado e não há dado disponível para leitura, ele irá bloquear a execução do processo até que o dado esteja disponível na FIFO. Da mesma forma, se um método `write()` é chamado e a FIFO está cheia, ele interromperá a execução do processo até que haja espaço disponível na FIFO para escrita. Este canal primitivo não implementa protocolo de comunicação, apenas permite a transferência do dado de um ponto a outro.

A simulação deve ser inicializada com o uso da diretiva `sc_start(-1)`. Neste caso, o argumento -1 indica que o simulador deve executar o modelo no tempo de simulação zero. Há duas possibilidades de parar a simulação: usando a chamada `sc_stop()`, que gera um sinal de parada de execução, ou a parada normal da simulação ocorrida pelo travamento ou término da execução.

A Figura 40 apresenta o código no nível de especificação executável para o GCD, ilustrando as decisões de implementação necessárias neste nível. Todo o código deste nível encontra-se no Anexo D , o que inclui todos os módulos e arquivos necessários para o funcionamento do

sistema (main.cpp, Makefile.linux, Makefile.defs, stimulus.cpp, stimulus.h, stimulus_file, monitor.cpp, monitor.h).

1esm_gcd.h

```
/*
 1esm_gcd.h - gcd in the Executable
Specification Model
 author: Sandro Renato Dias
 last date: 09/04/2006
 version: 1.0
*/

#include "systemc.h"

template <class T> SC_MODULE ( gcd ){

  // Portas
  sc_fifo_in<T> input_a, input_b;
  sc_fifo_out<T> result;

  // Processos
  void prc_gcd();

  // Construtor do modulo
  SC_CTOR( gcd ){
    SC_THREAD( prc_gcd );
    sensitive << input_a << input_b;
  }
};
```

1esm_gcd.cpp

```
/*
 1esm_gcd.cpp - gcd in the Executable
Specification Model
 author: Sandro Renato Dias
 last date: 09/04/2006
 version: 1.2
*/

#include "1esm_gcd.h"

void gcd::prc_gcd() {

   // Variaveis
   T tmp_a, tmp_b, tmp_result;

   // Leitura das portas
   tmp_a = input_a.read();
   tmp_b = input_b.read();

   if ((tmp_a == 0) | (tmp_b == 0))
      tmp_result = 0;
   else {
     while( 1 ) {
       tmp_a = tmp_a % tmp_b;
       if (tmp_a == 0) {
         tmp_result = tmp_b;
         break;
       }
       tmp_b = tmp_b % tmp_a;
       if (tmp_b == 0) {
         tmp_result = tmp_a;
         break;
       } // if
     } // while
   } // else
   result.write(tmp_result);
} // gcd
```

Figura 40 – Código do GCD no nível ESM.

4.3.3 Timed Data-Flow Model

Neste nível, os atrasos de tempo são adicionados aos processos dentro do projeto para refletir as restrições de tempo da especificação, atrasos de processamento da implementação alvo e latências de implementações específicas. Atrasos são adicionados às comunicações para modelar latência de comunicação. Este modelo é o ponto inicial para análise do compromisso (*trade-off*) de hardware/software, ou seja, o momento em que se decide qual parte do sistema será implementada em hardware e qual parte será implementada em software. A proposta é inserir a noção de tempo enquanto se mantém o nível puramente funcional.

No modelo TLM, neste nível, a comunicação entre os módulos é modelada usando chamadas de função, de forma precisa em termos de funcionalidade e, freqüentemente, em termos de tempo. Porém, ela não é modelada de forma estruturalmente precisa com relação à implementação alvo, pois detalhes de comunicação (protocolo usado, por exemplo) são deixados para serem tratados posteriormente. Uma vez obtido o modelo de tempo para o sistema, várias arquiteturas de hardware e software podem ser exploradas, analisadas em termos de performance, e então, podem ser escolhidas para determinar o melhor cenário do particionamento entre hardware e software.

De uma maneira geral, o próximo passo para a implementação da parte em software é a seleção do sistema operacional de tempo real e a partir daí, o desenvolvimento do código. Este sistema operacional será executado no hardware final sintetizado. Já o software implementará as funcionalidades que não deverão ser implementadas em hardware diretamente. SRD foca apenas o projeto de hardware e a sua implementação consistirá no refinamento da descrição do hardware do projeto a fim de se obter um modelo comportamental (próximo nível) e em seguida um novo refinamento em direção ao nível RTL, quando o projeto será sintetizado. O projeto de software neste nível ainda não é suportado pela versão 2.0 da linguagem SystemC, mas deve ser implementado na sua versão 3.0.

4.3.3.1 Regras de refinamento para o modelo TDM

A inserção de atrasos de tempo na computação e na comunicação entre os processos se dá através do uso das diretivas `wait()` em pontos específicos dos processos, após computações e comunicações. A freqüência com que essas intervenções são feitas é determinada pelo projetista. Assim, se uma estimativa de tempo de execução de uma porção de código é conhecida, ela deverá figurar ao final do bloco de código. Isto pode ser feito a cada linha de código, a cada bloco de código ou como o projetista preferir, pois a forma como serão inseridas as diretivas `wait()` é indiferente para o simulador. O tempo definido nestes pontos é estimado e deve direcionar o funcionamento da implementação alvo.

A diretiva `wait()` tem como argumento um tipo de dado SystemC, `sc_time`. Deve-se declarar a variável `sc_time` além de se definir a quantidade de unidades e a unidade da

magnitude, que é um tipo enumerado `time_unit`, podendo ser: *femtosecond* (`SC_FS`), *picosecond* (`SC_PS`), *nanosecond* (`SC_NS`), *microsecond* (`SC_US`), *millisecond* (`SC_MS`) ou *second* (`SC_SEC`).

O tempo de simulação especificado em `sc_start()` deve ser modificado indicando que a simulação deverá avançar no tempo para refletir o comportamento do sistema final. Esta alteração é feita no arquivo principal (main.cpp), onde são feitas as ligações entre os módulos. O código deste arquivo, além do código dos outros arquivos alterados neste nível, encontra-se no Anexo E .

Outro passo de refinamento neste nível é quebrar a funcionalidade em módulos descritos como função, com entradas e saídas definidas, modularizando uma funcionalidade e mantendo a abstração da arquitetura. Ao se fazer essa modularização, deve-se manter processos concorrentes em elementos separados porque os módulos dentro do mesmo elemento serão executados de forma seqüencial. Para se manter o fluxo do processamento de acordo com a funcionalidade do sistema, deve-se preservar a seqüência de execução como no nível anterior, resultando na inserção de sincronização adicional entre os elementos de processamento.

Deve-se, ainda, mapear as variáveis globais em memórias locais dos elementos de processamento que as usam ou numa memória compartilhada.

Canais globais devem ser alocados para substituir as instâncias de `sc_fifo` usadas no nível anterior. Além disso, devem-se agrupar canais entre elementos de processamento em um ou mais canais hierárquicos, que são usados para formar barramentos transacionais a partir de canais globais ou sinais entre elementos. A vantagem do uso do canal hierárquico é que os detalhes do barramento são abstraídos e os elementos são conectados através de uma porta usando uma interface simples. O acesso ao barramento é permitido através da interface e é feito em forma de chamadas de função. O canal hierárquico forma uma representação aproximada do barramento que será definido na implementação alvo. Os atrasos das comunicações devem também ser inseridos nos canais hierárquicos para modelar os atrasos de comunicação do sistema.

Uma vez que os barramentos foram definidos usando-se canais hierárquicos e interfaces, as portas dos módulos devem ser atualizadas de forma que elas permitam acesso à sua respectiva interface. Os processos também precisam ser atualizados, substituindo a porta antiga por chamadas de função que implementam o acesso ao barramento através da interface. Dessa forma, modela-se a comunicação usando a abordagem baseada em transações. O código do GCD neste nível encontra-se na Figura 41.

4.3.4 Behavioral Hardware Model

O modelo BHM (*Behavioral Hardware Model*) é preciso no nível de pinos e também ao nível de ciclos e sua estrutura interna está próxima, mas não necessariamente reflete a implementação alvo. Este modelo é sintetizável por, apesar de estar num nível algorítmico, demonstrar as entradas e saídas do projeto com precisão de ciclo de clock do sistema final. Além disso, sua simulação é mais rápida do que a do nível posterior, RTL, devido ao seu nível de abstração mais elevado.

4.3.4.1 Regras de refinamento para o modelo BHM

O comportamento do sistema é descrito como um algoritmo, e o uso de funções deve servir para gerenciar a complexidade do sistema, se necessário.

Embora o modelo inicial (especificação executável) possa conter tipos `int` e `float`, é necessário que estes tipos sejam refinados neste nível para tipos de dados que tenham precisão de bits, como os tipos de dados sintetizáveis descritos no Anexo C. Tipos sintetizáveis são tipos que podem ser implementados fisicamente, o que não é possível com alguns tipos de dados utilizados em C/C++. Os Anexos A e B trazem, ainda, as estruturas não sintetizáveis de SystemC e C/C++, com a respectiva ação de correção para tornar o código sintetizável. As tabelas dos anexos foram extraídas de [18]. O código do GCD neste nível encontgra-se na Figura 42.

2tdm_gcd.h

```
/*
 2tdm_gcd.h - gcd in the Timed Data-Flow
Model
 author: Sandro Renato Dias
 last date: 21/10/2006
 version: 1.3
*/

#include "systemc.h"

#define READ_LATENCY “10, SC_NS”
#define WRITE_LATENCY “10, SC_NS”
#define COMP_LATENCY “5, SC_NS”

// Interfaces para comunicacao
template <class T> class gcd_write_if :
virtual public sc_interface {
public:
  // Metodos da interface
  virtual void gcd_write(const T&) = 0;
};

template <class T> class gcd_read_if :
virtual public sc_interface {
public:
  // Metodos da interface
  virtual const T& gcd_read() const = 0;
};

template <class T> SC_MODULE ( gcd ){

  // Portas
  sc_port<gcd_read_if> input_a, input_b;
  sc_port<gcd_write_if> result;
  sc_port<gcd_read_if> reset;

  // Processos
  void prc_gcd();

  // Construtor do modulo
  SC_CTOR( gcd ){
    SC_THREAD( prc_gcd );
    sensitive << input_a << input_b <<
reset;
  }
};
```

2tdm_gcd.cpp

```
/*
 2tdm_gcd.cpp - gcd in the Timed Data-Flow
Model
 author: Sandro Renato Dias
 last date: 21/10/2006
 version: 1.1
*/

#include "2tdm_gcd.h"

// Implementacao da interface de escrita
void gcd_write_if::gcd_write(const T& tmp) {
   (this)->write(tmp);
   return *this;
}

// Implementacao da interface de leitura
const T& gcd_read_if::gcd_read() {
   return (*this)->read();
}

void gcd::prc_gcd() {

   // Variaveis
   T tmp_a, tmp_b, tmp_reset, tmp_result;

   // Leitura das portas
   // Uso da interface
   tmp_a = input_a->read();
   tmp_b = input_b->read();
   tmp_reset = reset->read();
   // Atraso da leitura
   wait (READ_LATENCY);

   if ((tmp_a == 0) | (tmp_b == 0) |
tmp_reset) {
      tmp_result = 0;
   }
   else {
      while( 1 ) {
        tmp_a = tmp_a % tmp_b;
        // Atraso da computacao
        wait (COMP_LATENCY);

        if (tmp_a == 0) {
            tmp_result = tmp_b;
            break;
        }
        tmp_b = tmp_b % tmp_a;
        // Atraso da computacao
        wait (COMP_LATENCY);

        if (tmp_b == 0) {
            tmp_result = tmp_a;
            break;
        } // if
      } // while
   } // else

   result->write(tmp_result);
   // Atraso da escrita
   wait (WRITE_LATENCY);

} // gcd
```

Figura 41 – Código do GCD no nível TDM.

O protocolo de entrada e saída (I/O) do projeto deve ser especificado através da definição dos ciclos de clock no momento em que as operações de entrada e saída ocorrem. Isto se dá

através da colocação de `wait()` a cada operação. É importante observar que somente as operações de entrada e saída devem ser precisas em termos de clock neste nível, o que não deve ocorrer com as outras operações descritas no algoritmo. Assim, o refinamento deve definir um protocolo de comunicação com as operações ocorrendo com precisão de ciclo de clock, não ocorrendo com a computação, neste nível.

Além disso, segundo Walstrom [19] e Grötker [20], deve-se definir *wrappers* ou adaptadores que serão usados para conectar os elementos de processamento, com precisão aproximada de tempo, à interface de canais hierárquicos a fim de se implementar um barramento no nível RTL. Esses adaptadores são capazes de chamar funções definidas pela interface para comunicar-se através do barramento. Assim a implementação dos métodos é modificada para que os sinais passem pelo adaptador antes de alcançarem o barramento, cujo protocolo, preciso no nível de ciclo, está contido na implementação dos adaptadores. Dessa forma, o sistema deve ser simulado para verificar a correção do protocolo.

O passo seguinte do refinamento é mesclar a funcionalidade dos adaptadores nos seus respectivos elementos de processamento, ou seja, o protocolo de comunicação será inserido no elemento, definindo-se uma interface no nível de pino entre o elemento e o barramento. A porta utilizada para conectar ao canal hierárquico deve ser substituída por portas de sinais que conectam diretamente à implementação RTL do barramento.

Os processos `SC_THREAD` devem ser substituídos por `SC_CTHREAD`, que são sensíveis à borda de subida ou descida do clock (de acordo com o especificado pelo projetista). Estes processos permitem a inserção de sentenças `wait()` ou `wait_until()` que aguardam pela borda do clock antes de terminar a execução. Além disso permitem o uso de sentenças `watching()` que permitem que o processo reinicie desde o seu início. É importante observar que processos `SC_CTHREAD` não são precisos ao nível de ciclo, mas são aproximados ao nível de tempo. Também permitem o uso de sentenças `wait()`. Por outro lado os processos `SC_METHOD` também são sensíveis à borda do clock, mas não permitem o uso das sentenças `wait()`. Já os processos `SC_THREAD` não permitem que um processo seja associado a um clock diretamente, como permitido pelo processo `SC_CTHREAD` [21], o que faz com que aquele clock definido ative o processo. Assim, uma porta clock deve ser definida para cada módulo, permitindo o sincronismo com o clock do sistema, definido globalmente.

3bhm_gcd.h

```
/*
 3bhm_gcd.h - gcd in the Behavioral Hardware
Model
 author: Sandro Renato Dias
 last date: 16/11/2006
 version: 1.0
*/

#include "systemc.h"

#define READ_LATENCY “10, SC_NS”
#define WRITE_LATENCY “10, SC_NS”
#define COMP_LATENCY “5, SC_NS”


SC_MODULE ( gcd ){

  // Portas
  sc_in_clk clk;   // Clock do sistema
  sc_in<unsigned int> input_a, input_b;
  sc_out<unsigned int> result;
  sc_out<bool> reset;
  // Protocolo de handshake
  sc_out<bool> send_data; // Requisita
leitura
  sc_out<bool> gcd_ready; // Saida pronta

  // Processos
  void prc_gcd();

  // Construtor do modulo
  SC_CTOR( gcd ){
    SC_CTHREAD( prc_gcd, clk.pos() );
    watching(reset.delayed() == true);
  }
};
```

3bhm_gcd.cpp

```
/*
 3bhm_gcd.cpp - gcd in the Behavioral
Hardware Model
 author: Sandro Renato Dias
 last date: 16/11/2006
 version: 1.0
*/

#include "3bhm_gcd.h"

void gcd::prc_gcd() {

   // Variaveis
   int tmp_a, tmp_b, tmp_result;

   // Operacoes de reset
   result.write(0);
   send_data.write(false);
   gcd_ready.write(false);
   wait();

   // Leitura das portas
   // Uso do protocolo de handshake
   send_data.write(true);
   wait();
   // Espera pelos ciclos de leitura
   wait(READ_LATENCY);
   send_data.write(false);
   tmp_a = input_a.read();
   tmp_b = input_b.read();
   wait();

   if ((tmp_a == 0) | (tmp_b == 0)) {
      tmp_result = 0;
   }
   else {
      while( 1 ) {
        tmp_a = tmp_a % tmp_b;
        // Atraso da computacao
        wait (COMP_LATENCY);

        if (tmp_a == 0) {
            temp_result = tmp_b;
            break;
        }
        tmp_b = tmp_b % tmp_a;
        // Atraso da computacao
        wait (COMP_LATENCY);
        if (tmp_b == 0) {
            tmp_result = tmp_a;
            break;
        } // if
      } // while
   } // else

   gcd_ready.write(true);
   result.write(tmp_result);
   // Atraso da escrita
   wait (WRITE_LATENCY);
   gcd_ready.write(false);
   wait();
} // gcd
```

Figura 42 – Código do GCD no nível BHM.

4.3.5 Register Transfer Level

O modelo RTL (*Register Transfer Level*) é o nível mais baixo de abstração suportado por SystemC e corresponde ao hardware digital sincronizado por sinais de clock. Ele descreve a arquitetura em termos de uma máquina de estados finitos do projeto final e o datapath. Tem precisão de nível de pino e ciclo de clock, onde cada operação realizada deve ter sua precisão definida. Neste modelo, toda a comunicação entre processos ocorre através de sinais. A sua estrutura reflete precisamente os registradores da implementação alvo e a lógica combinatória entre eles. A descrição RTL é independente da tecnologia. Usando um compilador SystemC é possível mudar a biblioteca da tecnologia alvo sem modificações na descrição RTL.

4.3.5.1 Regras de refinamento para o modelo RTL

Separar lógica de controle e datapath, identificando estados distintos que contenham código de um ou mais processos de um módulo. Processos `SC_CTHREAD` devem ser substituídos por `SC_METHOD`, de forma que não tenham capacidade de parar a execução. Para tanto, pode ser necessário que um processo `SC_CTHREAD` seja dividido em diversos processos `SC_METHOD` de forma a se dividir a computação. Isto fará com que se defina uma máquina de estados finita explícita para a lógica de controle. Uma vez definidos os processos `SC_METHOD` necessários, a lista de sensibilidade deve ser atualizada de forma a incluir todos os sinais necessários para que os processos possam manter a percepção das mudanças das entradas.

Determinar a arquitetura do datapath, substituindo todas as instâncias de variáveis e outros tipos abstratos em sinais. A única exceção a este caso é se uma variável é usada em somente um processo e não pode ser lida ou escrita por outros processos simultaneamente, isto porque sinais podem manipular leituras e escritas simultâneas enquanto variáveis não o fazem. Todos os valores dos sinais e variáveis devem ser definidos no construtor do módulo.

4rtl_gcd.h

```
/*
 4rtl_gcd.h - gcd in the Register Transfer
Level
 author: Sandro Renato Dias
 last date: 30/11/2006
 version: 1.0
*/

#include "systemc.h"

#define READ_LATENCY “10, SC_NS”
#define WRITE_LATENCY “10, SC_NS”
#define COMP_LATENCY “5, SC_NS”

enum state_t {
   s0_reset, s1_read, s3_gcd, s4_write
};

SC_MODULE ( gcd ){

  // Portas
  sc_in<bool> clk;
  sc_in<sc_uint<8> > input_a, input_b;
  sc_out<sc_uint<8> > result;
  sc_out<bool> reset;
  // Protocolo de handshake
  sc_out<bool> send_data; // Requisita
leitura
  sc_out<bool> gcd_ready; // Saida pronta

  // Sinais
  sc_signal<sc_uint<8> > tmp_a, tmp_b,
tmp_result;
  sc_signal<state_t> state, next_state;

  // Processos
  void prc_nextstate();
  void prc_gcd();

  // Construtor do modulo
  SC_CTOR( gcd ){
    SC_METHOD( prc_nextstate );
    sensitive_pos << clk;
    SC_METHOD( prc_gcd );
    sensitive << state;
  }
};
```

4rtl_gcd.cpp

```
/*
 4rtl_gcd.cpp - gcd in the Register Transfer
Level
 author: Sandro Renato Dias
 last date: 30/11/2006
 version: 1.0
*/

#include "4rtl_gcd.h"
```

4rtl_gcd.cpp - continuação

```
void gcd::prc_nextstate() {
   state = next_state;
}

void gcd::prc_gcd() {
   switch (state) {
      case s0_reset:
         result.write(0);
         send_data.write(false);
         gcd_ready.write(false);
         wait();
         next_state = s1_read;
         break;
      case s1_read:
         // Leitura das portas
         // Uso do protocolo de handshake
         send_data.write(true);
         wait();
         // Espera pelos ciclos de leitura
         wait(READ_LATENCY);
         send_data.write(false);
         tmp_a = input_a.read();
         tmp_b = input_b.read();
         wait();
         next_state = s2_gcd;
         break;
      case s2_gcd:
         if ((tmp_a == 0) | (tmp_b == 0)) {
            tmp_result = 0;
         }
         else {
            while( 1 ) {
               tmp_a = tmp_a % tmp_b;
               // Atraso da computacao
               wait (COMP_LATENCY);
               if (tmp_a == 0) {
                  temp_result = tmp_b;
                  next_state = s3_write;
                  break;
               }
               tmp_b = tmp_b % tmp_a;
               // Atraso da computacao
               wait (COMP_LATENCY);
               if (tmp_b == 0) {
                  tmp_result = tmp_a;
                  next_state = s3_write;
                  break;
               } // if
            } // while
         } // else
         next_state = s3_write;
         break;
      case s4_write:
         gcd_ready.write(true);
         result.write(tmp_result);
         // Atraso da escrita
         wait (WRITE_LATENCY);
         gcd_ready.write(false);
         wait();
         next_state = ;
         break;
   } // switch
} // gcd
```

Figura 43 – Código do GCD no nível RTL.

4.3.6 Considerações sobre os níveis

O aumento da complexidade e as escalas de integração crescentes nos dias atuais forçam os projetistas a redefinir suas metodologias de projeto. O uso de uma linguagem que permita ao projetista trabalhar com diversos níveis de abstração através de refinamentos sucessivos provê ao projetista mais liberdade no desenvolvimento. Além disto, a possibilidade de desenvolvimento simultâneo de hardware e software reduz o tempo total do projeto, permitindo que o software seja desenvolvido sobre um ambiente implementado em um nível de abstração que simula o hardware final em todas as suas funcionalidades. Conclui-se, claramente, que SystemC é uma linguagem poderosa para o projetista e o uso da Metodologia de Refinamentos Sucessivos aumentará a produtividade deste projetista, facilitando o projeto de sistemas cada vez mais complexos.

Analisando o código usado para implementar o GCD do nível ESM (Figura 40) ao nível RTL (Figura 43) e o seu crescimento (Tabela 5) pode-se observar a variação em termos de tipos de estruturas de dados (portas, sinais e variáveis), lógica de controle e quantidade de linhas de código. No nível ESM, a especificação executável consiste de um código mais legível, mais voltado para a funcionalidade do sistema enquanto que o código no nível RTL é implementado na forma de uma máquina de estados finitos e seu *datapath*, sendo específico deste nível, por ser sintetizável. O código RTL é preciso ao nível de pino e ciclo e, portanto, necessita de estruturas, lógicas de controle e protocolos que não existem no nível ESM, razão do aumento médio de 82% na quantidade de linhas de código do projeto total.

Observa-se ainda, que o código do arquivo gcd.h foi o único que teve redução do nível TDM (34) para o nível BHM (27), pois neste nível as interfaces de comunicação implementadas foram substituídas por protocolos de comunicação, removendo-se os trechos de código que as implementaram, resultando na diminuição da quantidade de linhas.

O arquivo Makefile não possui alterações no seu tamanho mas possui alterações no seu conteúdo, apontando para os arquivos do projeto que devem ser compilados no momento da execução/simulação.

Módulo	Níveis / Módulos	TLM			RTL	Crescimento (ESM-RTL)
		ESM	TDM	BHM		
Módulo Principal	gcd.h	19	34	27	36	89%
	gcd.cpp	31	51	53	69	123%
	Subtotal GCD	50	85	80	105	110%
Testbench	stimulus.h	32	33	36	46	44%
	stimulus.cpp	18	20	31	45	150%
	stimulus_file (vetor de testes)	5	10	10	10	100%
	monitor.h	19	19	24	34	79%
	monitor.cpp	16	17	30	43	169%
	Subtotal Testbench	90	99	131	178	98%
Ligações	main.cpp	33	38	40	40	21%
Compila/Simula	Makefile	11	11	11	11	0%
	Totais	184	233	262	334	82%

Tabela 5 – Quantidade de linhas de código por arquivo para cada nível de modelagem.

O arquivo main.cpp não possui grandes alterações na quantidade de linhas de código pois possui praticamente a mesma estrutura durante todo o processo, consistindo nas ligações entre os módulos. O acréscimo que ocorreu nos primeiros níveis se deu devido à inclusão de outros sinais e portas nos módulos para a implementação dos protocolos de comunicação.

5 Ferramenta de apoio ao projetista

Esta seção visa propor e definir uma ferramenta que auxilie o projetista no uso do refinamento sucessivo. Isto porque diversas ferramentas já foram propostas a fim de auxiliar o projetista na confecção do projeto de hardware com uma maior usabilidade na descrição, mas não se encontrou ferramenta que atuasse especificamente apoiando o projetista no uso do refinamento sucessivo. Aqui também será descrita a ferramenta desenvolvida com este objetivo, ou seja, orientar e apoiar o projetista no refinamento do seu projeto seguindo-se a metodologia descrita neste trabalho.

5.1 Características necessárias à ferramenta

Cada um dos níveis descritos anteriormente possui características únicas de implementação, referentes a módulos, ligações, canais, entre outros, que delimitam o nível em que a implementação está abstraída. A maior dificuldade enfrentada no momento de se implementar utilizando esta metodologia de abstração em níveis está em definir e manter o código dentro dos requisitos e exigências de cada nível.

Encontra-se na literatura referência a algumas ferramentas que visam facilitar o processo de projeto, porém não se encontra algo que permita ao projetista abstrair-se dos detalhes da linguagem utilizada para a modelagem em níveis diferenciados de abstração (Capítulo 2, Trabalhos Relacionados). Esta ferramenta é necessária para que o projetista possa simular, em SystemC por exemplo, o seu projeto a partir da idéia inicial, sem que tenha que desenvolver grande quantidade de código em linguagem de descrição de hardware de baixo nível. Como a

linguagem SystemC possibilita ainda a utilização de testbenches incorporados no código, é interessante que a ferramenta facilite também este uso.

A ferramenta aqui desenvolvida visa facilitar o processo de refinamento sucessivo possibilitando ao projetista avançar e regredir nos estágios mediante pequenas alterações manuais e/ou automatizadas, podendo ater-se com maior dedicação aos detalhes inerentes ao funcionamento do sistema em desenvolvimento ao invés dos detalhes referentes aos diversos níveis de refinamento.

Dias e Silva Jr. [17] levantam algumas características para a implementação de uma ferramenta de apoio a projeto de sistemas embutidos, além delas, podemos observar as seguintes características necessárias:

5.1.1 Interface gráfica

Uma interface gráfica permite uma melhor visualização do projeto em desenvolvimento bem como um melhor entendimento das ligações entre os módulos. É difícil, para projetos grandes, manter todo o diagrama do projeto sem se ter um esboço gráfico de suas ligações. Isto porque cada porta de cada módulo deve estar ligada à outra porta de outro módulo através de um sinal. Tanto portas quanto sinais possuem nomes e tipos de porta e de dados. Gerenciar estas informações sem um diagrama é tarefa tediosa e desgastante. Portanto, deve ter uma interface gráfica de forma que se tenha um desenho dos módulos e esse desenho gere automaticamente os códigos SystemC necessários para a implementação do projeto. Assim, qualquer modificação no módulo gráfico deve gerar automaticamente uma modificação nos arquivos ".cpp" e ".h" e até mesmo no arquivo "main.cpp" (principal, contendo as bibliotecas, instanciações dos módulos e suas ligações). Isto gera um código compilável e coerente com o diagrama apresentado.

5.1.2 Curva de aprendizado

A implementação da ferramenta deve priorizar a interface com o usuário para reduzir ao máximo a curva de aprendizado e facilitar o processo de desenvolvimento de um System-on-Chip. A usabilidade e a orientação quanto ao que deve ser feito auxiliam o usuário e diminuem a dificuldade no uso da ferramenta, permitindo a fixação das atividades rotineiras. A não necessidade de conhecimento profundo do processo de refinamento, e dos detalhes específicos da linguagem para este processo, pode ser possível pelo uso de assistentes para estas tarefas, além do uso de uma interface bastante acessível. Ou seja, interface, assistentes e orientação ao usuário diminuem a curva de aprendizado permitindo que a ferramenta seja utilizada até mesmo por projetistas pouco experientes.

5.1.3 Importação de projetos/módulos

A ferramenta deve ser capaz de ler um módulo criado anteriormente e importá-lo no projeto em desenvolvimento. O módulo inserido deve manter todas as características originais (portas, processos, métodos) a fim de ser incorporado ao projeto. Esta característica permite o reúso de IPs e sua incorporação fácil em novos projetos, uma vez que um projeto já desenvolvido pode ser inserido em outro projeto, evitando que todo o seu diagrama seja refeito. Para isso, é necessário implementar uma biblioteca de módulos que permita que os módulos desenvolvidos estejam à disposição do projetista.

5.1.4 Suporte a refinamentos

Outra característica que se faz necessária é o suporte a refinamentos, onde o projetista, geralmente, encontra dificuldades. Uma definição clara dos níveis de abstração bem como das alterações necessárias para o refinamento a cada nível envolve características da linguagem e de estruturas utilizadas. Esta definição deve estar à disposição do projetista ou ser utilizada pra orientá-lo no refinamento. Esse suporte pode ser implementado como um assistente para

refinamento do projeto, indicando pontos que deverão ser modificados a fim de se alcançar o nível desejado.

5.1.5 Geração de Testbenches

Um projeto necessita de testbenches para acompanhamento do seu funcionamento e desenvolvimento, dessa forma, qualquer alteração feita que implique na geração de resultados indevidos será percebida com o uso de um testbench. Os testbenches consistem de dois módulos, um gerador de estímulos e um monitor, conforme Seção 4.2 (Testbenches). O testbench pode ser gerado automaticamente, a partir da definição, do projetista, de quais sinais receberão entrada e quais sinais serão monitorados na saída. A estrutura completa do testbench pode ser gerada a partir dessa definição, facilitando e otimizando o seu uso posterior.

5.1.6 Lista de sensibilidade

Com o uso de uma interface gráfica para a programação dos módulos, uma nova facilidade poderá ser agradável ao projetista, consistindo da disponibilização da lista de sensibilidade de cada módulo e a sua possibilidade de alteração nesta interface. A lista de sensibilidade é uma lista de sinais cuja alteração deverá ser observada para a ativação do módulo, ou seja, qualquer alteração em um sinal da lista poderá fazer com que o módulo seja executado. A manutenção incorreta desta lista leva o projeto a resultados indesejados. Assim, sua definição a partir da interface gráfica dá ao projetista maior controle do projeto.

5.2 Desenvolvimento

A partir das características levantadas uma ferramenta foi desenvolvida com o objetivo de facilitar o projeto de sistemas de hardware usando SystemC. Por meio de uma interface

gráfica, o projetista pode desenvolver seu projeto a partir da inserção de módulos, portas, ligações entre as portas e inserção automática de testbench. Neste ponto o projetista solicita a geração dos arquivos do projeto, obtendo, automaticamente todos os arquivos de cada módulo e, principalmente, o main.cpp, onde se encontram as instanciações dos módulos, as definições de ligações entre eles, bem como a geração do arquivo vcd, para obtenção das formas de onda.

5.2.1 Decisões de implementação

5.2.1.1 Delimitação de escopo de desenvolvimento/funcionamento

A fim de se obter um protótipo inicial funcional da ferramenta, as seguintes características e limites foram impostos, para que se tenha sucesso na implementação e, a partir daí, realizarem-se as atualizações e/ou alterações necessárias para se incluir outras características e funcionalidades à ferramenta pronta.

- A ferramenta irá efetuar o refinamento do projeto como um todo, não sendo possível, no desenvolvimento inicial, o refinamento individualizado de um único módulo mantendo-se os outros módulos em níveis diferentes. Isto dá uma maior liberdade e direciona o foco do desenvolvimento, permitindo que se alcance o sucesso no refinamento. Uma vez alcançado, a migração para um refinamento individualizado de cada módulo implicará na inserção de adaptadores e pequenas alterações para a comunicação de módulos em níveis diferentes.

- Não será possível importar um código pronto para uso na ferramenta, os projetos devem ser inseridos através da inserção de módulos novos e/ou do reúso de módulos já criados na ferramenta. Posteriormente esta facilidade pode ser incorporada permitindo que um código em SystemC seja lido e importado, gerando-se automaticamente o seu projeto na ferramenta, permitindo o refinamento. Outra razão para esta decisão é que o projetista irá iniciar o projeto dentro da metodologia aqui proposta, tanto de desenvolvimento quanto de refinamento sucessivo.

- O arquivo Makefile foi criado especificamente para a compilação no ambiente linux. Posteriormente as outras possibilidades deste arquivo podem ser geradas de forma atender a todos os ambientes possíveis.

- Apesar da possibilidade de co-desenvolvimento hardware/software e do particionamento ocorrer após o nível TDM, não faz parte do objetivo da ferramenta dar suporte ao desenvolvimento do software do projeto. A ferramenta SRD deverá ser utilizada para o desenvolvimento/refinamento da descrição da parte do projeto que será implementada em hardware.

5.2.1.2 Estrutura dos arquivos gerados

A fim de facilitar o refinamento e organizar os arquivos de cada módulo, optou-se por fragmentar o código de cada módulo em um arquivo de computação (chamado modulo.cpp) e um arquivo de declaração e definição (chamado modulo.h). Além disso, as ligações entre as instâncias dos módulos bem como as demais definições do módulo principal do programa se encontram no arquivo main.cpp. Esta organização dos arquivos padroniza o projeto, pois diversos projetistas implementam todo o projeto num único arquivo, ou o fragmentam sem critério ou objetivo.

Assim, cada projeto é composto de um arquivo principal (main.cpp) e outros arquivos .h e .cpp para cada módulo presente. Por exemplo, um projeto que implemente somente um módulo chamado GCD terá os seguintes arquivos, contendo:

- main.cpp – as instanciações dos módulos, sinais, as ligações entre esses, além do código para o rastreamento de sinais, geração do arquivo .vcd, definição de tempo de simulação, entre outros

- GCD.h – as definições das classes, variáveis, portas, sinais, processos, funções, entre outros para o módulo somador

- GCD.cpp – a parte funcional do código, a implementação das estruturas definidas para o módulo

- stimulus.h – as definições das classes, variáveis, portas, sinais, processos, funções, arquivo com o vetor de testes, entre outros (este módulo faz parte do testbench)

- stimulus.cpp – a parte funcional do código, a implementação das estruturas definidas para o módulo, leitura do vetor de testes e envio dos sinais ao módulo principal (este módulo faz parte do testbench)

- monitor.h – as definições das classes, variáveis, portas, sinais, processos, funções, arquivo com o resultado dos testes, entre outros (este módulo faz parte do testbench)

- monitor.cpp – a parte funcional do código, a implementação das estruturas definidas para o módulo, escrita dos resultados dos testes em arquivo (este módulo faz parte do testbench)

- makefile.linux – contém as definições básicas para a compilação do projeto de acordo com o ambiente linux

Para cada nível de abstração, os nomes dos arquivos serão precedidos da identificação do nível, ou seja, 1ESM, 2TDM, 3BHM e 4RTL.

5.2.1.3 Geração automática do main.cpp

A descrição das ligações entre os módulos de um projeto em SystemC é uma tarefa muito precisa que envolve um grande trecho de código, no caso da metodologia aqui apresentada, significando a alteração do arquivo main.cpp. A geração automática da descrição desta ligação reduz a probabilidade de erro além de reduzir tempo gasto pelo projetista que necessitaria apontar, simplesmente, quais portas de um módulo estariam ligadas às quais portas de outro módulo, recebendo o código da instanciação de cada módulo e das interconexões entre as portas. Este é um ponto onde ocorrem falhas no código além de problemas de comunicação entre módulos que podem ser facilmente resolvidos com essa geração automática.

5.2.1.4 Geração automática do testbench

A partir do módulo escolhido, o testbench será gerado com os módulos Stimulus e Monitor sendo criados com as mesmas portas do módulo escolhido, porém invertidas de forma que possam alimentá-lo e receber seu resultado.

5.2.1.5 Linguagem para a implementação

Para o desenvolvimento da ferramenta optou-se pelo uso da linguagem Java, por ser um ambiente multiplataforma, além disso, o uso das classes swing para criação do ambiente gráfico possibilita a sua execução em outros ambientes sem a necessidade de grandes adaptações.

5.2.1.6 SystemC 2.0.1

A versão da linguagem SystemC, utilizada na geração do código a partir da ferramenta, é 2.0.1, e a sua documentação, disponível no site da *Open SystemC Initiative*, é a seguinte:

- Guia do usuário da versão 2.0.1, 2002 [21]
- Conjunto sintetizável, 2004 [22]
- Especificação Funcional para a versão 2.0.1, 2002 [23]
- Manual de Referência da Linguagem para a versão 2.0.1, revisão 1.0, 2003 [24]

Os desenhos utilizados na ferramenta para representar módulos, threads, portas, foram baseados nos apresentados por Black e Donovan [83].

5.2.1.7 Uso de xml

Todo o diagrama implementado é descrito em arquivos xml[13], Figura 44, sendo um para cada arquivo de código a ser gerado (.h ou .cpp). A adoção do xml possibilita o uso de tags criadas para delimitar cada trecho que é passível de refinamento bem como cada detalhe do diagrama. Isto facilita a busca destes trechos e sua alteração a cada etapa do processo. O Anexo H apresenta um exemplo de um dos arquivos xml gerados.

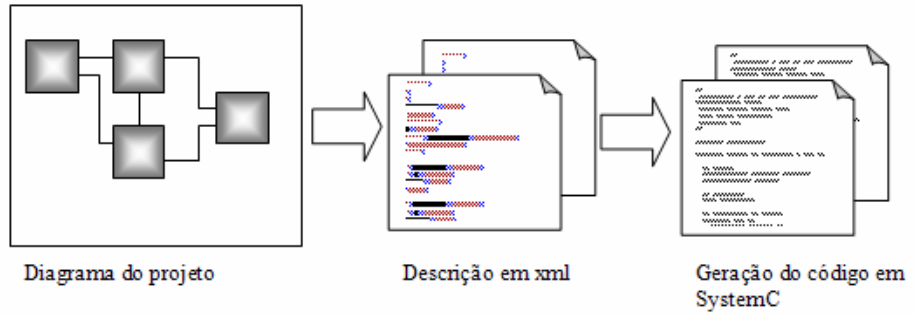

Figura 44 – Fases do projeto.

5.2.2 Testes

Os seguintes testes foram realizados com a ferramenta desenvolvida:

- **Geração do testbench** – A geração automática do testbench evita que o projetista tenha que escrever o código de cada módulo do testbench, permitindo que ele se preocupe somente em preencher o vetor de testes ou fazer pequenas alterações. Os arquivos do testbench gerados seguem os arquivos 1ESM_Stimulus.h, 1ESM_Stimulus.cpp, 1ESM_Monitor.h e 1ESM_Monitor.cpp, que se encontram no Anexo D juntamente com os outros arquivos do projeto. A geração dos arquivos se deu como esperado nos testes não ocorrendo problemas, uma vez que as portas são replicadas tanto em relação ao tipo quanto em relação ao sinal.

[13] XML (eXtensible Markup Language) – Linguagem de marcação como HTML com tags não pré-definidas, usada para descrever dados focando no que o dado é (http://www.w3.org/XML).

- **Geração dos arquivos** – A ferramenta foi testada quanto à eficiência na geração dos arquivos a partir do projeto diagramado. De acordo com o teste, todos os arquivos foram gerados corretamente. Um projeto foi escrito manualmente e logo em seguida desenvolvido na ferramenta, o tempo do projeto usando a ferramenta foi muito menor que o projeto manual. De 2 horas gastas no desenvolvimento manual de um projeto simples, usando-se a ferramenta este tempo caiu para 20 minutos, uma redução de 84% no tempo total do projeto.

- **Refinamento** – Os testes com o refinamento apontaram para os bugs em algumas tarefas, que estavam relacionados à busca incorreta de padrões no código ou à geração incorreta de arquivos. Ao final das correções, os testes do refinamento permitiram comprovar que a ferramenta permite um refinamento do código bem orientado e auxiliado, uma vez que algumas tarefas podem ser executadas automaticamente pela ferramenta. Não se pôde testar o refinamento do código como um todo, pois algumas tarefas de refinamento são responsabilidades do projetista, uma vez que envolvem análise do código e alterações em seu fluxo de desenvolvimento e até mesmo em estruturas de dados.

5.3 Funcionamento

Uma vez executada, a ferramenta abre uma tela inicial (Figura 45) onde o projetista identifica-se e pode criar ou escolher um diretório para um novo projeto (botão Escolher) ou abrir um projeto já criado (botão Abrir). A identificação do usuário será utilizada no comentário de cada arquivo do projeto como o autor do código.

Figura 45 – Tela inicial do sistema.

Ao clicar-se em OK, a ferramenta abre a sua tela principal onde o projeto será desenvolvido (Figura 46). Para o início do desenvolvimento, deve-se clicar sobre o ícone do Módulo no menu à esquerda e arrastá-lo para dentro da área do diagrama. Após as alterações no módulo, gera-se o testbench (no menu popup) e os arquivos do projeto (no menu Sistema - Gerar Arquivos). Por fim, o refinamento é iniciado (no menu Sistema - Refinamento) e o projeto é refinado em direção ao último nível, RTL. O menu Arquivo permite criar Novo projeto, Abrir um projeto já existente (arquivo de diagrama ".dia"), Salvar o projeto corrente ou Fechar o programa. Para o desenvolvimento do projeto no nível inicial (ESM) uma janela lateral exibirá as informações que o projetista deverá levar em consideração para a descrição neste nível, que é identificado no canto superior esquerdo, logo abaixo do menu Sistema.

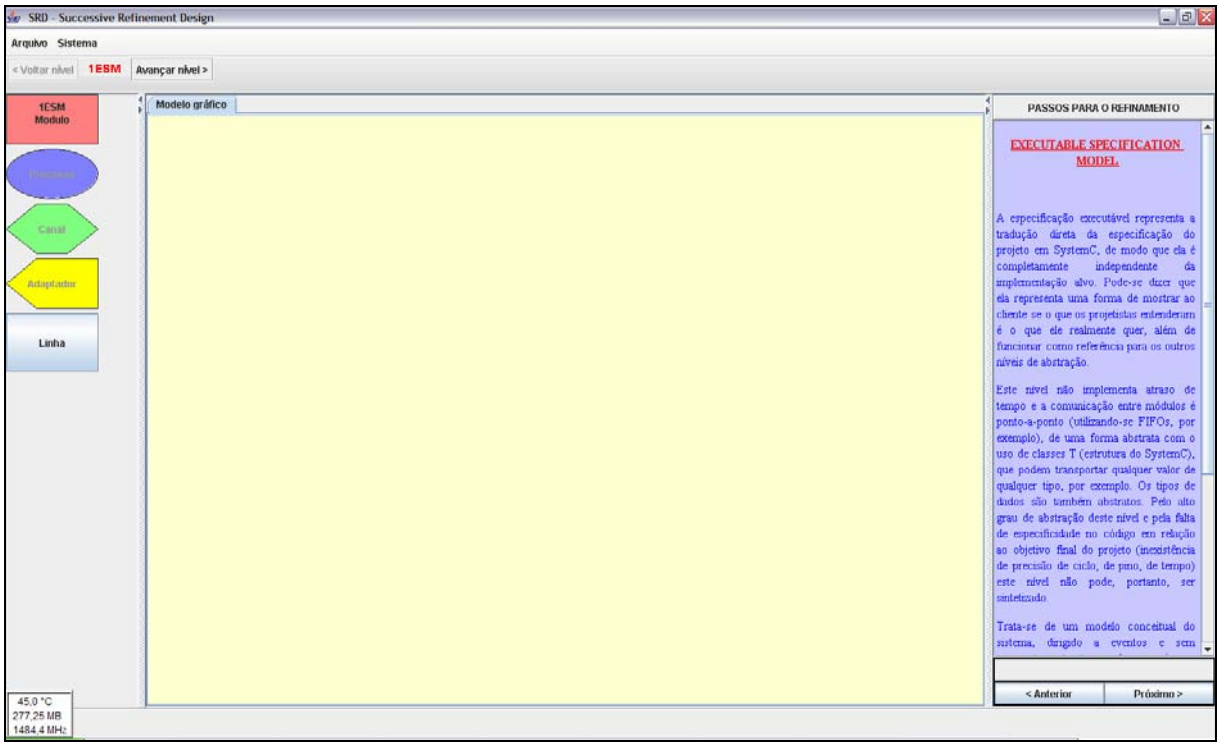

Figura 46 – Tela principal do sistema.

5.3.1 Criação de um módulo

Ao arrastar-se o ícone do Módulo do menu à esquerda para a área do diagrama, o módulo é inserido no projeto contendo um processo SC_THREAD e nenhuma porta. Em seguida, clicando-se sobre o módulo com o botão direito abre-se o menu popup que permite alterar nome, deletar, copiar o módulo, inserir portas de entrada e saída, definir o tipo, definir o módulo como principal, bem como gerar testbench (Figura 47). Estas características serão exploradas a seguir.

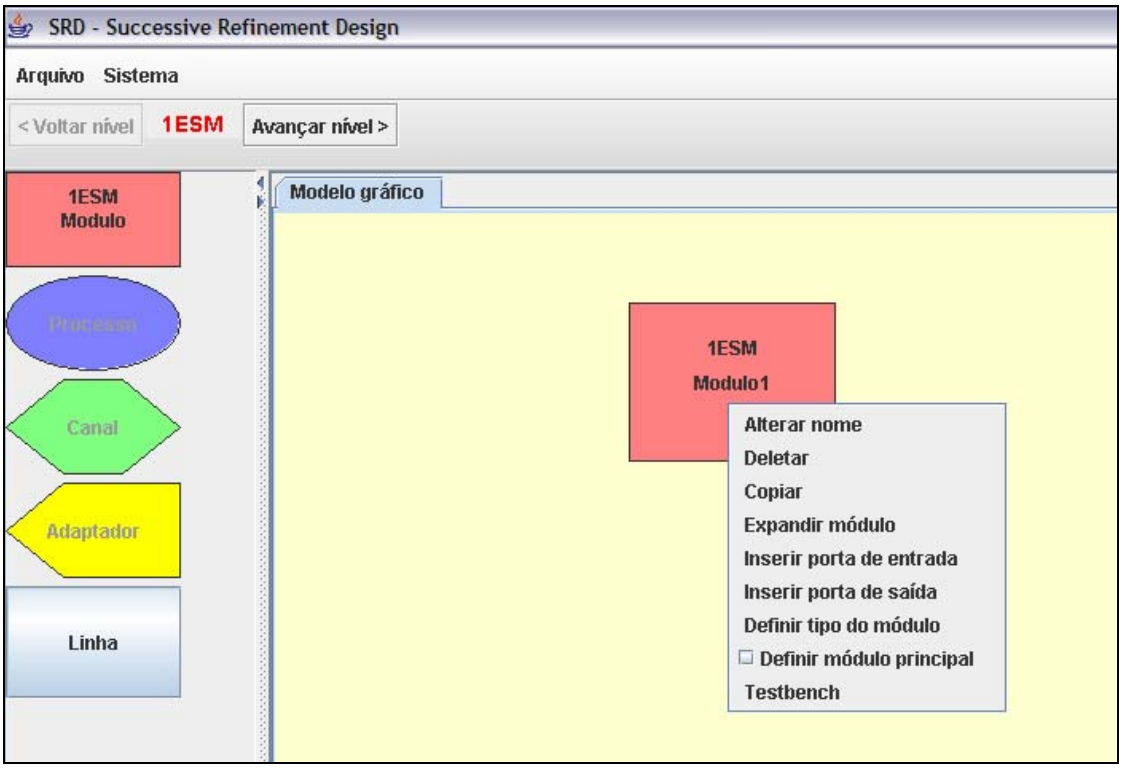

Figura 47 – Módulo inserido e menu popup com opções.

5.3.1.1 Alteração do nome

Para alterar o nome do arquivo, depois de selecionado o comando no menu popup, a janela (Figura 48) surgirá e o nome digitado será substituído no módulo. Isto fará com que todos os outros itens que utilizem este nome também sejam alterados, ou seja, o processo interno do módulo, os arquivos .h e .cpp bem como sua instanciação no arquivo principal e qualquer outra incidência do nome.

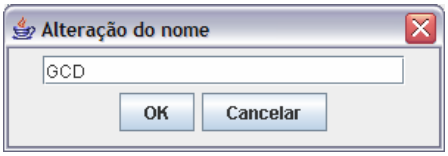

Figura 48 – Tela de alteração de nome do módulo.

5.3.1.2 Deletar módulo

Deletar um módulo implica em excluir o seu desenho do diagrama bem como todas as suas incidências nos arquivos do projeto (portas, sinais, includes). Se este módulo estiver ligado a outros, as conexões (sinais) serão também excluídas.

5.3.1.3 Copiar módulo

A cópia de um módulo implica na inserção de outro desenho do módulo no diagrama e a inclusão de uma nova instância do módulo no main.cpp com as devidas ligações. Para se fazer isso, após escolher a opção “Copiar” no menu popup, basta clicar na área do diagrama com o botão direito e escolher “Colar”. O novo módulo terá a mesma estrutura do copiado, pois provém do mesmo arquivo de origem, tratando-se de apenas outra instância. Isto faz com que alterações no arquivo original sejam refletidas no novo módulo.

5.3.1.4 Inserção de portas de entrada e saída

A inserção de portas, tanto de entrada (Figura 49, esquerda), quanto de saída (Figura 49, direita), se dá através do item “Inserir porta de entrada” ou “Inserir porta de saída” do menu popup. As janelas seguintes se abrem para a respectiva ocasião permitindo que o projetista escolha o tipo da porta e o tipo do dado que ela carregará, bem como defina um nome para a porta. No caso de portas de entrada existe a possibilidade de a porta ser do tipo “Clock”, o que é uma situação peculiar, pois este tipo de porta não é ligada a outra porta, recebendo o sinal de clock diretamente do kernel de simulação da linguagem SystemC. Assim, uma porta definida como “Clock” incidirá no arquivo main.cpp com o sinal de clock do sistema sendo enviado diretamente a ela, sem a necessidade do projetista definir este sinal.

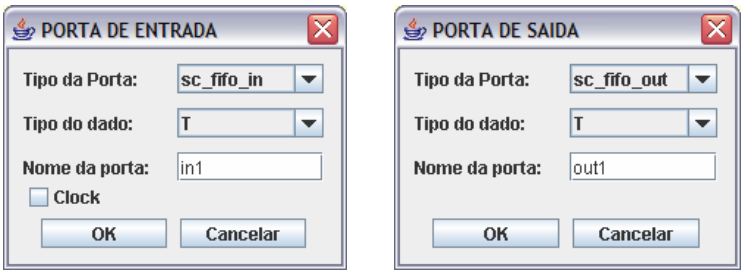

Figura 49 – Telas de inserção de porta de entrada (esquerda) e saída (direita).

Como o projeto é criado no nível ESM, os tipos usados para as portas são os tipos `sc_fifo_in` e `sc_fifo_out`, que são canais de comunicação de alto nível, o tipo usado para os dados é o tipo abstrato T.

Para se remover as portas inseridas (Figura 50), basta clicar em cada uma e pressionar a tecla "delete".

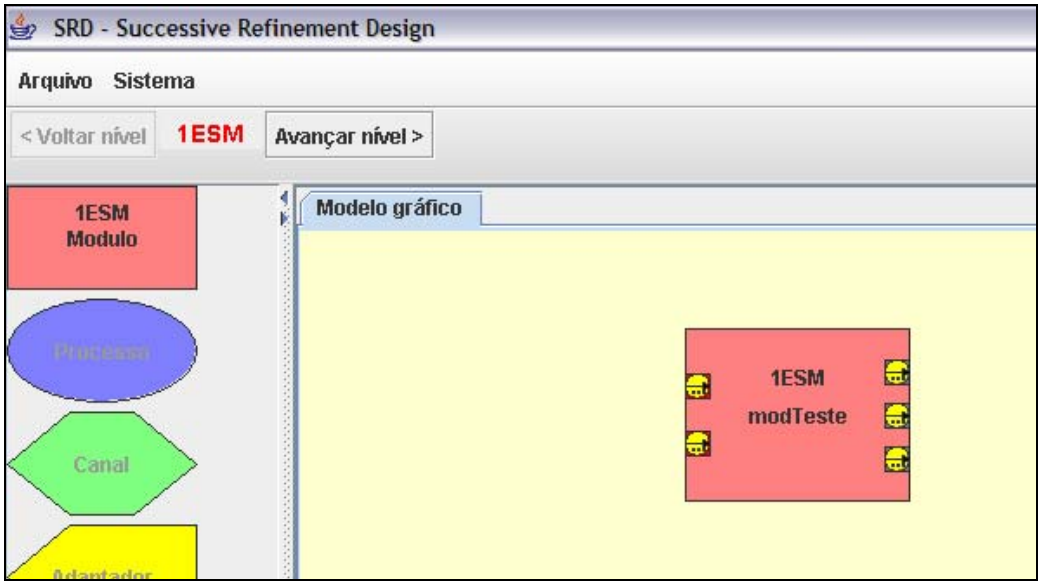

Figura 50 – Módulo com 2 portas de entrada e 3 portas de saída.

5.3.1.5 Expandir módulo

Ao selecionar a opção "Expandir módulo" no menu popup, o módulo escolhido é visualizado em zoom (Figura 51), permitindo que outros processos sejam inseridos no módulo. Como o nível de criação é o ESM, os processos a serem inseridos são do tipo SC_THREAD.

Com o módulo expandido é possível alterar o nome dos processos internos, deletar, copiar, editar o código ou configurar as portas associadas ao processo e à lista de sensibilidade de cada processo.

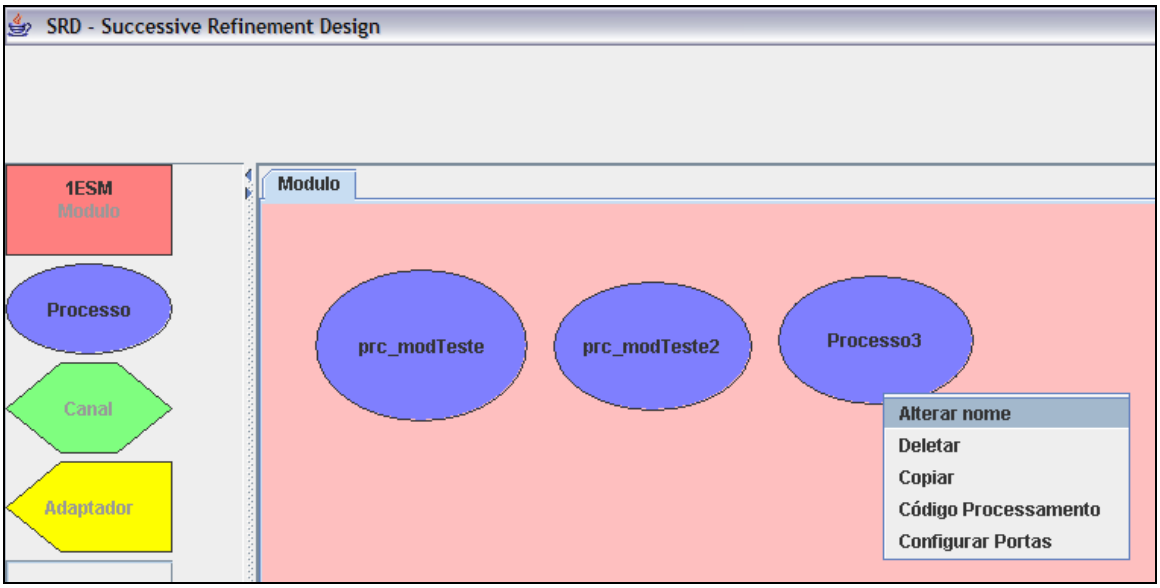

Figura 51 – Módulo expandido com mais dois outros processos inseridos.

A opção “Configurar portas” do menu popup do processo, nesta tela, abre uma nova janela com as portas do módulo (Figura 52). No campo “Portas disponíveis” estão listadas as portas do módulo que não estão associadas ao processo escolhido. No campo “Portas ativadas” estão listadas as portas que serão utilizadas dentro do processo. Se for porta de entrada, a porta será incluída automaticamente na lista de sensibilidade do processo, terá uma variável criada e associada à leitura dessa porta no código do processo. Se for porta de saída, terá uma variável criada e associada à sua escrita no código do processo. No caso de mais de um processo dentro de um mesmo módulo, cada processo pode ter portas diferentes ou compartilhar de algumas portas, tanto de entrada quanto de saída.

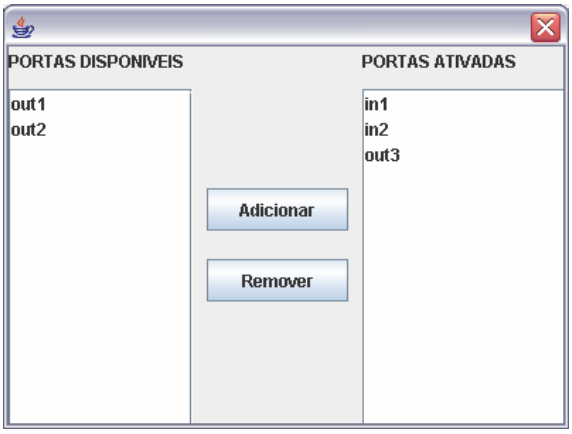

Figura 52 – Definição de portas do processo.

5.3.2 Código processamento

Toda a definição de portas e estrutura dos módulos é feita através do diagrama, e deve ser alterada neste. O código que fará parte do processo a ser executado pelo módulo deverá ser descrito através da opção “Código Processamento” do menu popup de cada processo, o que abrirá uma janela, permitindo que este código seja digitado ou importado a partir de um arquivo (Figura 53). Isto fará com que este código seja inserido após a declaração de variáveis e antes da escrita nas portas, dentro do código que implementa o processo.

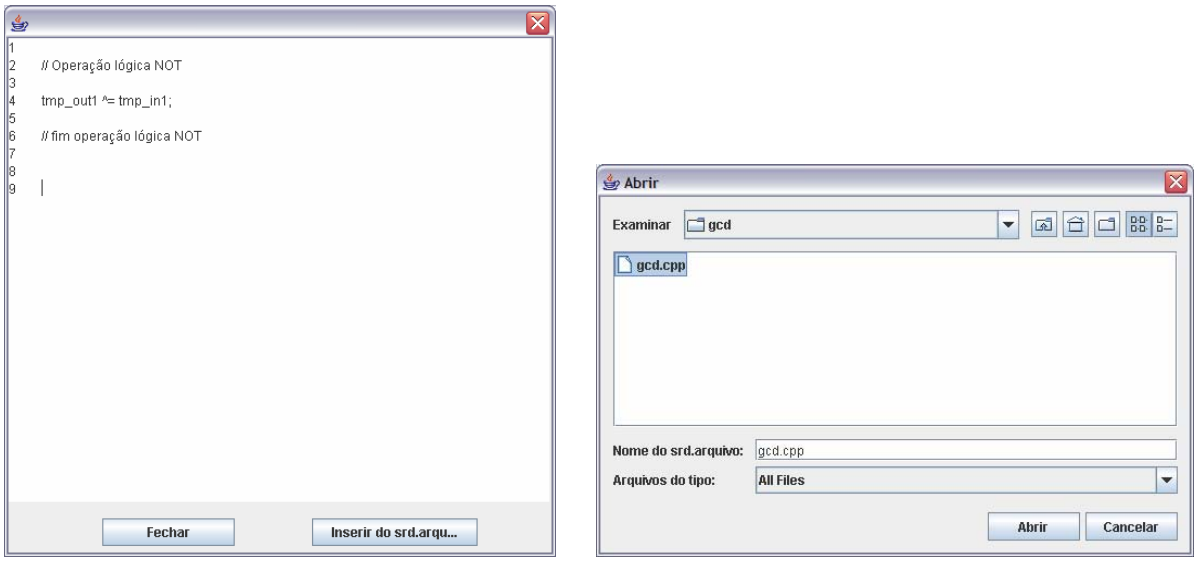

Figura 53 – Inserção de código no processo por digitação (esquerda) ou importação de arquivo (direita).

5.3.2.1 Definição do tipo do módulo

Os módulos criados, nos dois primeiros níveis, são `template <class T>` e, portanto, é necessário definir um tipo a ser associado a cada módulo na sua instanciação no main.cpp. Este tipo é definido através da opção do menu popup do módulo "Definir tipo do módulo" (Figura 54).

Figura 54 – Definição do tipo do módulo.

5.3.2.2 Definição do módulo principal

No menu popup do módulo, a opção "Definir módulo principal" define o módulo principal do projeto, informação necessária para a geração dos arquivos de diagrama, os arquivos com o código em SystemC bem como para o refinamento.

5.3.2.3 Visualização de portas e arquivos

O duplo clique no módulo abre a janela de visualização de portas e arquivos do módulo. Na primeira aba (Portas) pode-se observar a tabela com o tipo da porta, tipo de dado e nome associado a cada porta (Figura 55).

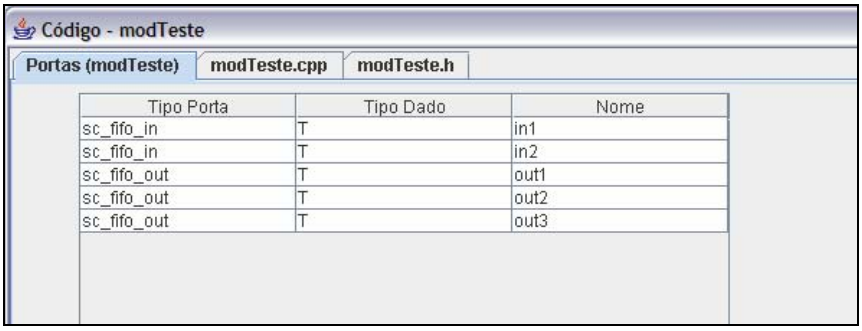

Figura 55 – Visualização da tabela das portas do módulo.

A segunda aba (Figura 56, esquerda) apresenta o código do arquivo “.cpp” do módulo, onde pode-se observar que o código é gerado com as portas de entrada e saída sendo lidas e escritas através de variáveis que são criadas com o prefixo “tmp_” mais o nome da porta. Se a porta não fizer parte do processo (definida através da opção Configurar portas, Seção 5.3.1.5 Expandir módulo) não figurará no código deste processo. A terceira aba (Figura 56, direita) apresenta o código do arquivo “.h” do módulo, que traz a declaração das portas e processos desse módulo. Neste código, as variáveis de entrada definidas pra cada processo são inseridas na lista de sensibilidade daquele processo.

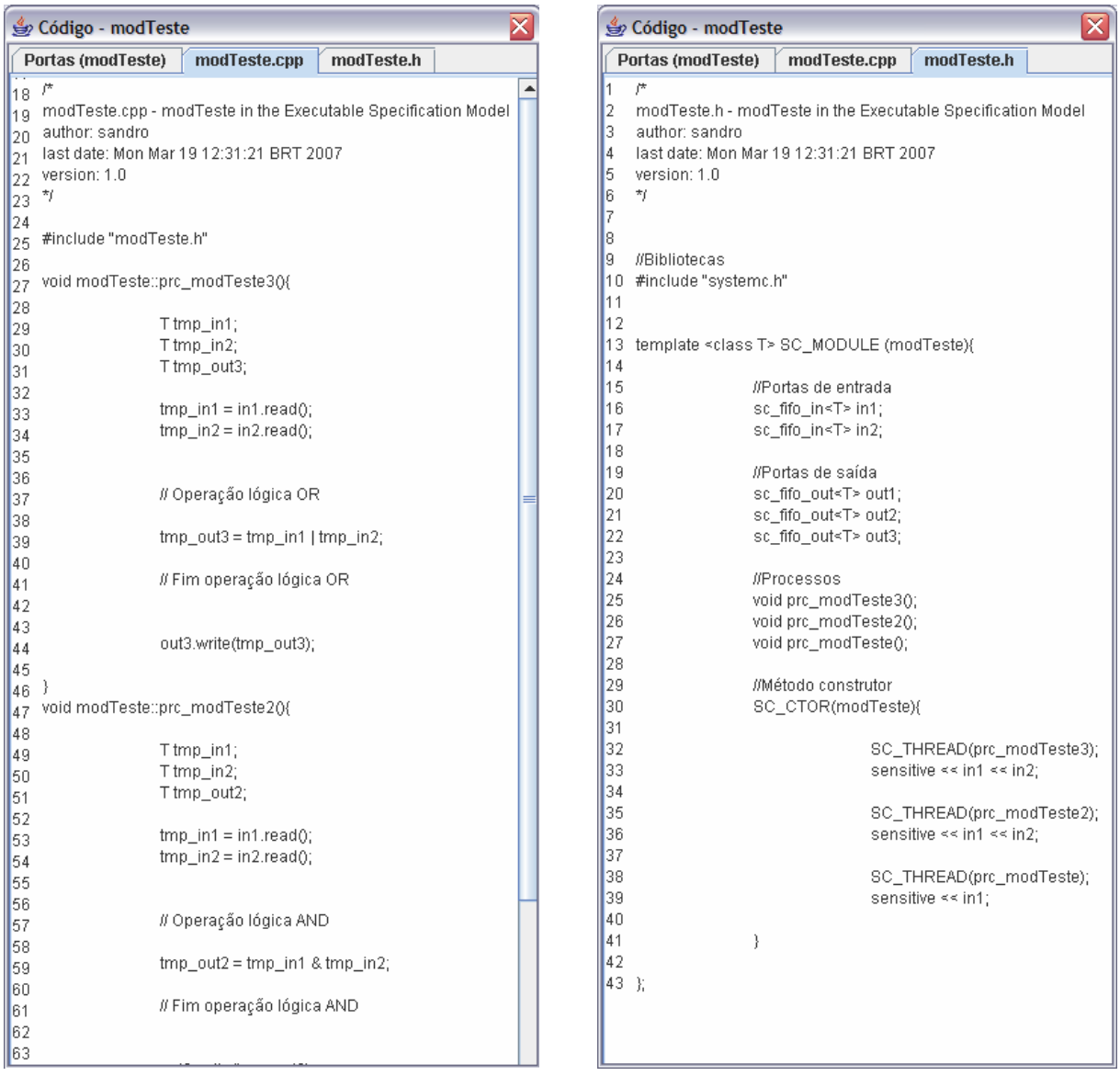

Figura 56 – Visualização do código dos arquivos .cpp (esquerda) e .h (direita) do módulo.

5.3.3 Geração do testbench

A partir da opção Testbench do menu popup do módulo, clicando-se sobre o módulo gera-se o testbench para ele, consistindo de dois módulos, Figura 57. O Stimulus é criado com portas de saída com o mesmo nome das portas de entrada do módulo escolhido e os sinais para ligação entre estas portas também já são criados. O Monitor é criado com portas de entrada com o mesmo nome das portas de saída do módulo escolhido e das portas de saída do módulo Stimulus, assim, é ligado a estes dois módulos, recebendo, tanto sinais que são gerados pelo Stimulus quanto os sinais que são o resultado do processamento, provenientes do módulo escolhido.

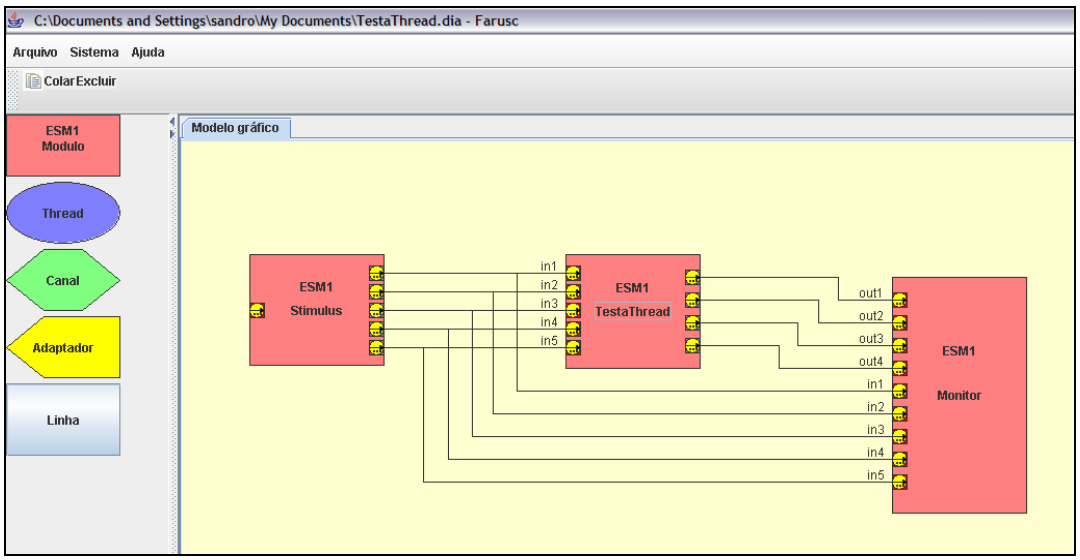

Figura 57 – Geração automática de testbench.

5.3.4 Geração dos arquivos

Uma vez que todo o projeto encontra-se diagramado, basta acessar o menu Sistema, depois clicar no item “Gerar Arquivos” para que os arquivos “.h” e “.cpp” de cada módulo, bem como o principal, main.cpp, sejam gerados (Figura 58). Estes arquivos são criados no diretório do projeto.

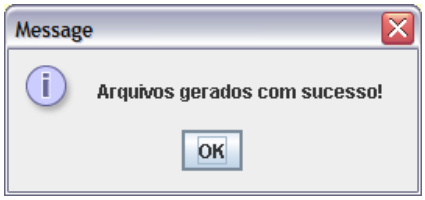

Figura 58 – Confirmação de geração dos arquivos do projeto.

5.3.5 Salvamento do projeto

O menu Arquivo – Salvar, da janela principal, permite que o projeto seja salvo (Figura 59). Assim um arquivo do tipo arquivo de diagrama (extensão .dia) é criado. Este arquivo pode ser aberto através do menu Arquivo – Abrir, possibilitando que o projeto possa ser reaberto e alterado.

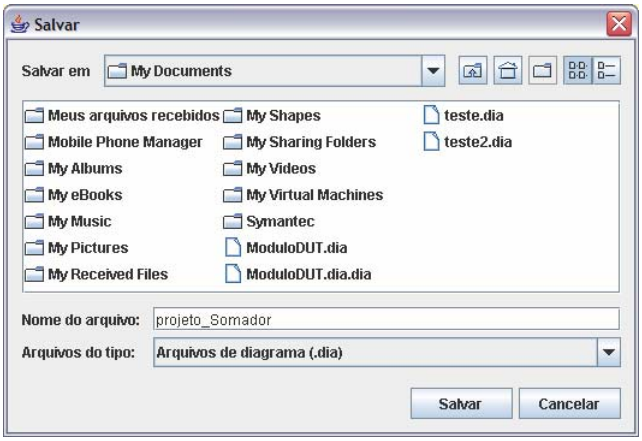

Figura 59 – Salvamento do projeto como arquivo de diagrama.

5.3.6 Refinamento

Uma vez terminado o projeto do sistema a partir do diagrama e da inserção do código em seus processos, pode-se iniciar o refinamento do código clicando-se no botão “Avançar nível” ao lado da definição do nível atual, no canto superior esquerdo da janela principal. Este procedimento iniciará o refinamento e, como primeiro passo, mostrará informações sobre o

próximo nível de refinamento, ou seja, o *Timed Data-Flow Model* (Figura 60), uma vez que o projeto é todo construído no nível *Executable Specification Model*.

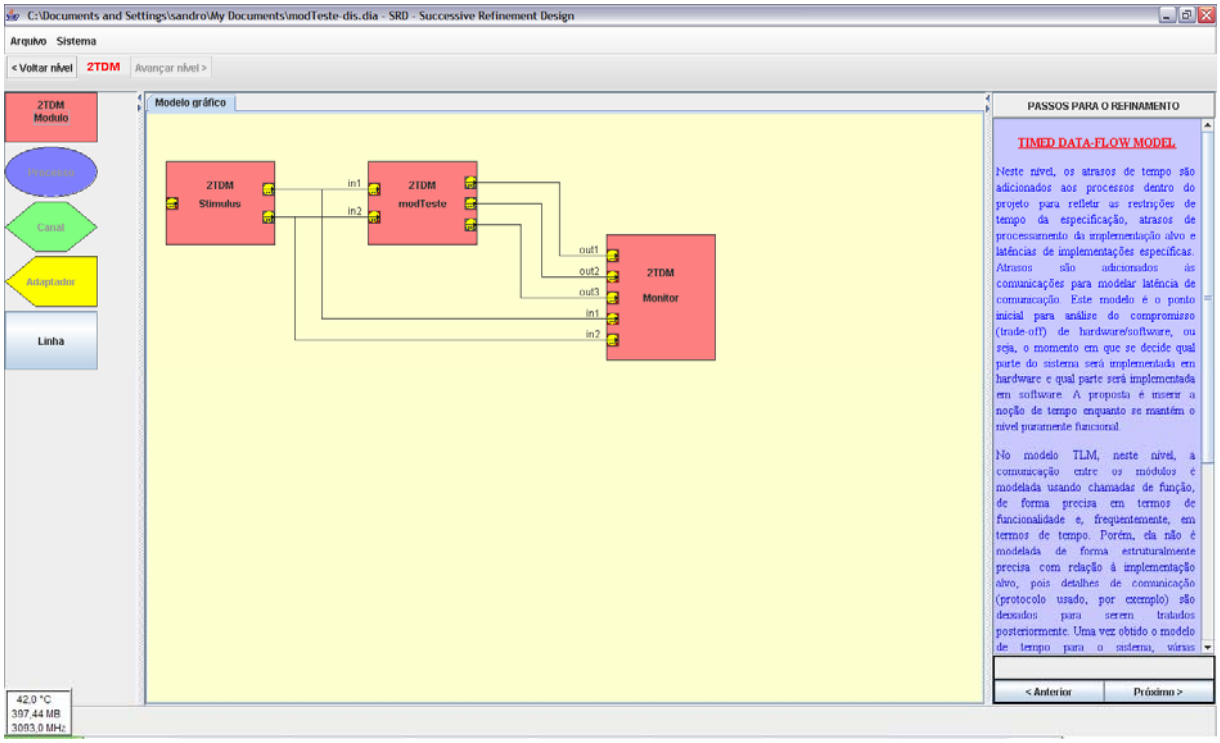

Figura 60 – Início do refinamento para o nível TDM.

Os botões Próximo e Anterior permitem navegar entre as opções do refinamento que, quando são tarefas específicas a serem desenvolvidas pelo projetista, apresentam apenas um texto orientando o que deve ser feito para que aquela tarefa seja concluída. Caso a alteração possa ser feita pela ferramenta de uma forma automatizada, a mesma apresenta o botão “Executar Tarefa”, logo acima dos botões Anterior e Próximo na janela Passos para o Refinamento, que, ao ser clicado, permite a conclusão da tarefa sem a participação do projetista. Se for necessária a participação do mesmo, a interação se dará através de pequenas janelas com as solicitações necessárias, como na Figura 61, que trata da definição das temporizações que serão inseridas no código e sua posição (antes ou depois do código do processamento), necessárias no nível TDM.

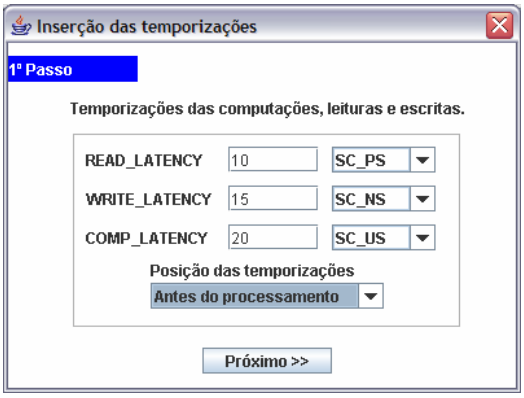

Figura 61 – Definição das latências de leitura, escrita e computação

A Figura 62 mostra o próximo passo do refinamento, consistindo na varredura do código em busca de trechos onde possa ocorrer uma computação, para que o projetista possa definir se será inserida a latência da computação naquele ponto. As latências de leitura e escrita são inseridas automaticamente no código, não necessitando da confirmação do projetista a cada ponto de inserção.

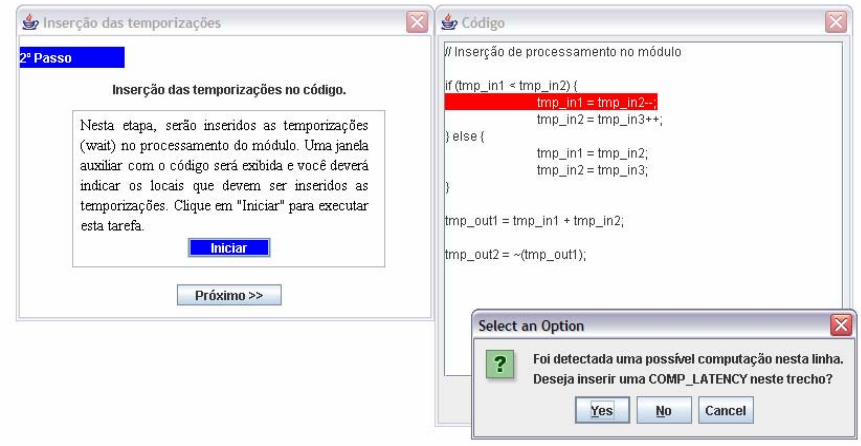

Figura 62 – Inserção de temporizações no código.

Todo o refinamento do projeto, para cada nível, se dá dessa forma. Ou seja, as tarefas que possam ser executadas automaticamente pela ferramenta, sem intervenção do projetista são feitas e se necessária sua participação, surgem janelas solicitando que o mesmo informe os dados relevantes como, por exemplo, os pontos onde devem ser inseridas as latências de computação (Figura 62).

Ao se passar de um nível para o outro, todo o projeto é salvo, permitindo que o projetista possa voltar em um nível anterior para efetuar alterações, que possam ser necessárias devido

às falhas ou necessidades detectadas ao longo do refinamento. Assim, essas alterações num nível anterior podem ser feitas abrindo-se o projeto daquele nível e efetuando-se o refinamento novamente após essas alterações, propagando-as para o próximo nível.

Todo o refinamento efetuado pela ferramenta segue as regras descritas neste documento (Capítulo 4 - Metodologia de Refinamentos Sucessivos), facilitando o trabalho do projetista e orientando-o a cada passo, utilizando a janela “Passos para o Refinamento” e/ou janelas auxiliares na execução de tarefas.

5.3.7 Tempo total gasto no projeto

O menu Sistema – Tempo do Projeto, da janela principal, permite a visualização do tempo total gasto no projeto (Figura 63). Este tempo é calculado de forma simples, sendo o tempo total corrido enquanto o projeto estiver aberto, sendo incrementado a cada reabertura do projeto. Desta forma, tem-se o tempo total desde a especificação inicial até o último nível refinado.

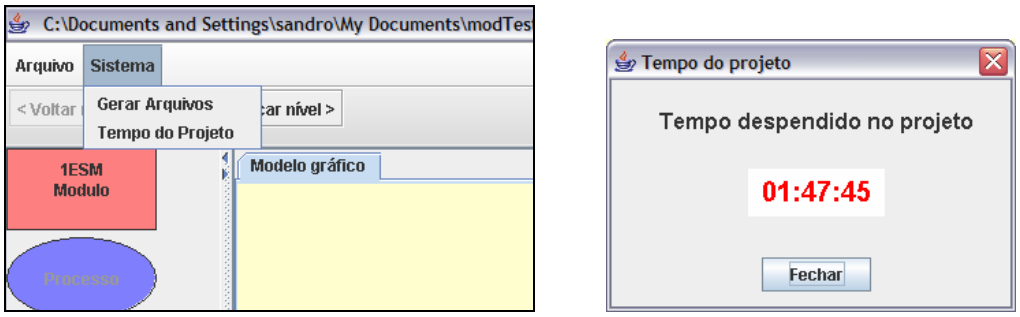

Figura 63 – Visualização do tempo total gasto no projeto.

5.3.8 Dificuldades encontradas no desenvolvimento

Na implementação da ferramenta, alguns problemas surgiram e foram contornados e aqui relatados:

- **Sobreposição das linhas e módulos** – No desenho do diagrama do projeto, após a interligação dos módulos usando linhas (sinais) para ligar as portas se tornava impossível movimentar os módulos ou fazer alterações em suas portas ou sinais. A solução encontrada foi armazenar as posições das linhas e das quebras de linha de forma a evitar que a sobreposição atrapalhasse a manipulação dos módulos.

- **Serialização** – Alguns bugs de serialização do diagrama ocorreram no momento em que o projeto era salvo, pois ao reabrir o diagrama os ícones perdiam a mobilidade. Neste momento, eram salvos todos os objetos, porém os eventos desses objetos não eram salvos. A solução foi o registro de todos os eventos de todos os objetos do diagrama, de forma a recuperar tudo o que havia sido feito até o último momento em que foi salvo.

- **Interface de edição do código** – Uma vez que o código é armazenado em arquivos xml através de tags, a edição do código do arquivo pelo projetista implica na alteração destas tags. A solução para se incluir o código do processamento do módulo foi adicionar uma janela ao projeto que permite ao projetista digitar o código que entrará no processo entre a declaração de variáveis e a escrita nas portas. Qualquer outra alteração no restante do código deve ser feita através do diagrama do projeto, evitando alterações diretas no código que possam implicar em problemas no projeto. Para se ativar esta janela basta clicar no módulo com o botão direito do mouse e escolher “Código Processamento”, isto permitirá tanto a inclusão quanto a edição do código.

- **Interface para o refinamento** – Outro problema encontrado foi com relação à interface do refinamento junto ao usuário. É importante que o usuário tenha conhecimento do que está ocorrendo durante o refinamento e o que deve ser feito para a mudança de um nível para o outro. Esta interface deve fornecer ao usuário informações sobre os passos do refinamento bem como permitir que o mesmo faça algumas alterações manualmente. A solução encontrada para a interface do refinamento foi dividir a janela principal do projeto inserindo na lateral direita uma pequena janela intitulada “Passos para o refinamento”. Nesta janela serão mostradas as informações sobre cada nível bem como cada passo necessário para o refinamento, utilizando-se botões para navegação entre os passos do refinamento e para executar algumas tarefas de refinamento automaticamente.

- **Impossibilidade de compilação de um template usando-se arquivos separados .h e .cpp** – Por um problema da própria linguagem C++, é impossível compilar-se um projeto que usa templates e foi desenvolvido em arquivos separados para a descrição e definição das estruturas de dados e funções (.h) e outro para a implementação destas funções e da lógica de controle do projeto (.cpp). Este problema não é detectado na compilação de projetos que não usam a estrutura de templates. Dessa forma, o nível inicial implementado na ferramenta e o nível seguinte, fruto do refinamento, geram arquivos que não são compiláveis, obtendo-se erros de compilação. A solução adotada para este problema é a geração de um arquivo .h para cada módulo contendo não só a descriçao do módulo, suas estruturas de dados e das funções como também a sua implementação, ou seja, uma união dos arquivos .h e .cpp. Isto permite que o projeto seja compilado sem os erros gerados pelos `template <class T>`.

5.4 Considerações sobre o capítulo

Neste capítulo foi apresentada a ferramenta SRD (Successive Refinement Design) desenvolvida a partir dos detalhes da Metodologia de Refinamentos Sucessivos (Capítulo 4) e das características da linguagem SystemC (Capítulo 3). Esta ferramenta permite o projeto de um sistema de hardware utilizando-se uma interface gráfica com geração automática de código e refinamento orientado. Desta forma o projeto é desenvolvido num nível mais alto (ESM) e passa por três níveis de refinamento até que chegue em um nível sintetizável (RTL).

6 Considerações Finais

Neste trabalho foram apresentados alguns instrumentos de grande utilidade ao projetista de sistemas embutidos, ou seja, uma metodologia de desenvolvimento, consistindo de refinamentos sucessivos; uma linguagem para a descrição do hardware e do software, linguagem esta que dá suporte à metodologia; e uma ferramenta, desenvolvida para a utilização da metodologia e da linguagem.

O aumento da complexidade e as escalas de integração crescentes nos dias atuais forçam os projetistas a redefinir suas metodologias de projeto. A evolução rápida da tecnologia de fabricação de circuitos integrados aliada à necessidade de redução do time-to-market implicam na redução do tempo de projeto dos sistemas embutidos. A união da Metodologia de Refinamentos Sucessivos com a linguagem SystemC possibilita partir de um código de descrição de hardware em alto nível (Especificação Executável) e refiná-lo até um código sintetizável, reduzindo em muito o tempo de projeto. A ferramenta desenvolvida e aqui apresentada, além de facilitar o projeto dos sistemas de hardware ainda dá o suporte necessário ao projetista orientando-o durante este refinamento.

A divisão do fluxo do projeto em quatro níveis de abstração permite um projeto mais rápido, focando decisões críticas em estágios específicos, permitindo ao projetista testar cada nível e validá-lo, antes de refinar outros pontos específicos do projeto. A cada nível, é possível visualizar, observar, focar e refinar detalhes específicos enquanto que outros são abstraídos em níveis mais altos, dando-se importância a eles em momentos específicos. O refinamento permite que grandes partes do sistema permaneçam sem modificação enquanto que outras partes/detalhes sejam explorados em profundidade.

Uma metodologia bem definida e uma ferramenta de suporte permitem que modelos de baixo nível possam ser automaticamente gerados a partir de modelos em alto nível mais próximos adicionando-se os detalhes correspondentes a partir de regras de refinamento e transformações.

A definição dos níveis aqui abordada é suficiente para facilitar o uso do refinamento sucessivo pelo projetista, pois usa os níveis mínimos necessários para a aplicação da metodologia. O refinamento da especificação executável para um modelo temporizado e posteriormente para um comportamental, seguido de um RTL, concentra, em cada nível, suas características e requisitos, permitindo que estas características sejam adicionadas/refinadas no momento oportuno.

O uso de uma linguagem que permita ao projetista trabalhar com diversos níveis de abstração através de refinamentos sucessivos provê ao projetista mais liberdade no desenvolvimento. Além disto, a possibilidade de desenvolvimento simultâneo de hardware e software reduz o tempo total do projeto, permitindo que o software seja desenvolvido sobre um ambiente implementado em um nível de abstração que simula o hardware final em todas as suas funcionalidades. SystemC mostrou ser uma linguagem muito poderosa para o projetista e o uso da Metodologia de Refinamentos Sucessivos com esta linguagem aumentará a produtividade do projetista, facilitando o projeto de sistemas cada vez mais complexos.

A Seção 2.3 apresentou diversas ferramentas úteis ao projeto de sistemas embutidos. A busca principal para tal seção era de ferramentas que atuassem diretamente no refinamento do código, porém não se encontrou alguma que suportasse isso. Dessa forma, o resultado deste trabalho preenche a lacuna encontrada dentre as opções disponíveis no mercado.

O desenvolvimento de projetos em SystemC, por tratar-se de um ambiente sem interface gráfica, pode ser facilitado com o uso de uma ferramenta que possua tal interface, se aliado a isso houver uma metodologia de desenvolvimento e um suporte a esta metodologia pela mesma, o trabalho do projetista será agilizado. O ganho com a adoção da ferramenta aqui apresentada é a redução do tempo total de projeto, uma vez que ela agiliza a geração dos arquivos e ainda orienta o projetista durante o refinamento.

Dentre as características da ferramenta desenvolvida estão: rapidez no projeto, suporte ao refinamento, geração automática dos arquivos de cada módulo, geração automática das ligações entre os módulos e instanciações dos mesmos, geração automática do testbench para o projeto, reúso de módulos, curva de aprendizado pequena, dentre outras.

A partir do resultado do desenvolvimento melhorias deverão ser feitas na ferramenta, a título de trabalho futuro, visando melhor atender as necessidades do projetista assim como diminuir a quantidade de instrumentos necessários para a finalização do projeto (ferramentas para especificação executável, desenvolvimento em linguagem de descrição de hardware, simulação, teste, verificação, síntese, dentre outras).

Os itens que contribuem fortemente para a redução do tempo de projeto, dentre outros, são:

- Geração automática do testbench, com a inclusão de um módulo de estímulo e um monitor de sinais – este testbench é criado a partir das portas de entrada e saída, permitindo as portas do testbench sejam criadas de acordo com as portas do projeto, agilizando a sua implementação.

- Criação automática das variáveis e suas leituras e escritas associadas às portas – para cada porta é criada uma variável de mesmo tipo, que é associada à sua leitura, se for porta de entrada, ou sua escrita, se for de saída.

- Definição automática das ligações dos módulos (arquivo main.cpp) – este é um ponto muito suscetível a erros no projeto, uma vez que as ligações devem ser feitas entre portas de mesmo tipo e com o sinal com o mesmo tipo da porta. A ligação visual permite uma melhor visualização deste ponto além de permitir que a mesma seja feita automaticamente no arquivo principal, eliminando o tempo de escrita deste arquivo, uma vez que o projetista não precisa preenchê-lo.

- Tarefas do refinamento automáticas – permitir que a ferramenta execute algumas tarefas do refinamento, reduzem também o tempo de projeto pois o projetista não precisa varrer o código em busca de estruturas ou de trechos que necessitam de refinamento. Estas tarefas automatizadas permitem que mudanças mais simples e rotineiras possam ser feitas mais rapidamente.

6.1 Contribuições deste trabalho

Diversos autores tratam da Metodologia de Refinamentos Sucessivos, mas não há um consenso com relação aos níveis e ao escopo de cada nível. A contribuição deste trabalho consistiu na definição clara tanto da pilha de níveis quanto da definição e escopo de cada um deles, permitindo que possa ser usada independente de ferramenta e linguagem.

Geralmente os projetistas de hardware baseiam-se em sua bagagem de desenvolvimento para definir como vão lidar com os arquivos de descrição do projeto. Alguns utilizam um único arquivo com todo o código do projeto dentro, outros particionam em um arquivo por módulo. A metodologia de codificação usada neste trabalho, mais precisamente de organização dos arquivos em “.h” e “.cpp”, com um arquivo principal de interconexão (main.cpp), agiliza o processo de refinamento e entendimento do código. Sua contribuição está na padronização da codificação do projeto e facilidade de entendimento por outro projetista.

Em seu artigo [101], Ghenassia et al descreve a utilização da metodologia de refinamentos sucessivos utilizada na implementação de um SoC usando SystemC. Cita ainda a grande vantagem do desenvolvimento simultâneo de hardware e software, uma vez que a plataforma desenvolvida em um determinado nível de abstração permite a simulação com uma precisão suficiente para teste do software em desenvolvimento. Segundo ele, utilizando-se o método tradicional de desenvolvimento de hardware, o software somente seria desenvolvido após o hardware sintetizado, pois somente neste momento haveria uma plataforma confiável para teste do software a ser desenvolvido. O desenvolvimento simultâneo permitiu um ganho de 6 meses para o término do projeto total (hardware-software).

A contribuição da ferramenta desenvolvida concentra-se no fato de tornar mais rápido o processo de desenvolvimento de Sistemas de Hardware, pois além da metodologia e da linguagem, há o suporte da ferramenta tanto na elaboração do projeto quanto na aplicação da metodologia de refinamento. Essa contribuição é ainda mais valorizada pelo fato de não haver ferramenta similar.

Além do ganho no projeto de hardware com a redução do tempo total do projeto e por tratar-se de ferramenta de projeto utilizando uma linguagem de descrição de hardware com

possibilidade de diferentes níveis de abstração, esta ferramenta pode ser utilizada também como instrumento didático em disciplinas de arquitetura de computadores, projeto de sistemas de hardware e afins tanto de graduação quanto de pós-graduação, onde pode-se explorar o projeto de hardware usando a linguagem SystemC. Seu uso educacional é uma boa contribuição na área de treinamento de projeto de hardware usando a Metodologia de Refinamentos Sucessivos, pois dá ao projetista uma visão funcional do projeto e permite que ele aprofunde este projeto inserindo a cada etapa detalhes necessários em cada nível.

6.2 Trabalhos Futuros

Este documento não tem a pretensão de finalizar aqui todo o trabalho desenvolvido. Pesquisas e outros desenvolvimentos serão necessários para complementar e aprimorar o resultado aqui apresentado. Segue uma relação não exaustiva de itens que podem fazer parte dos trabalhos futuros a partir deste.

6.2.1 Integração com simulação

A ferramenta desenvolvida não efetua a simulação do projeto em SystemC. Para que isto ocorra o projetista necessita executar os arquivos no ambiente escolhido para simulação e efetuar as correções necessárias utilizando a ferramenta. A possibilidade de simulação do projeto a partir da própria ferramenta cria mais uma facilidade ao projetista que, assim, no mesmo ambiente consegue desenvolver o projeto, simulá-lo e corrigi-lo a cada refinamento.

6.2.2 Integração com síntese

A ferramenta desenvolvida não efetua a síntese do projeto em SystemC. Existem diversas ferramentas comerciais que produzem a síntese a partir de código em SystemC diretamente, sem a necessidade de tradução para Verilog/VHDL (ver Seção 2.3.1 - Ferramentas de

Síntese). Uma integração da ferramenta de projeto com uma ferramenta de síntese completa o ciclo de desenvolvimento do sistema embutido, permitindo maior agilidade na obtenção do protótipo físico funcional.

6.2.3 Inclusão de parser para importação de código

Como a ferramenta gera os arquivos e efetua o refinamento a partir do projeto elaborado através do diagrama, torna-se impossível importar um código SystemC de um projeto que não foi desenvolvido na ferramenta. A inclusão de um parser SystemC, como os descritos na Seção 2.3.3 - Ferramentas de apoio ao projeto, permite esta importação, pois o parser seria responsável por identificar as estruturas do código e elaborar os arquivos xml no formato utilizado pela ferramenta. Assim seria possível gerar os diagramas e proceder ao refinamento do projeto.

6.2.4 Controle de versão

O fluxo de projeto usando-se SystemC e a Metodologia de Refinamentos Sucessivos (Figura 3) é baseado em modelagem, simulação e refinamento, tratando-se de um fluxo cíclico. Após uma simulação ou refinamento, pode haver a necessidade de alterar-se o projeto com o objetivo de corrigir algum erro detectado ou mesmo inserir nova funcionalidade. Esta alteração pode implicar em mudanças no nível anterior, necessitando de novos refinamentos e simulação. O uso de um controlador de versão (CVS[14] por exemplo) permite o armazenamento de todos os códigos usados nesses estágios de desenvolvimento, facilitando a retomada de qualquer ponto do desenvolvimento do projeto.

[14] *Concurrent Versions System* – Sistema que visa armazenar o histórico de arquivos fonte e documentos de um projeto.

6.2.5 Geração de vetores de teste automaticamente

A partir da geração automática do testbench, baseado no módulo a ser testado, a geração dos vetores de teste pode se dar também de forma automática utilizando-se alguma técnica para geração destes vetores, que pode ser incorporada à ferramenta de forma a criar-se toda a estrutura de teste do projeto em desenvolvimento.

Referências

[1] GULLAPALLI, V., SHI, K. *Hierarchical Design Techniques*. Synopsys, Inc. January, 2004.

[2] SILVEIRA, R., VAN NOIJE, W.A.M. *Metodologia orientada ao projetista para system-level design*. LSI/Politécnica/USP, IX Workshop de IBERCHIP, La Havana, Cuba, 25-28/03/2003. Artigo - 06 páginas. CD-ROM. ISBN: 959-261-105-X.

[3] GRIMPE, E., OPPENHEIMER, F. *Object-Oriented High Level Syntesis Based on SystemC*, OFFIS Research Institute, University of Oldenburg, Germany, IEEE 2001.

[4] ODETTE Project. Acessado em: 10/06/05. Disponível em: <http://odette.offis.de> e em <http://www.ecsi-association.org/ecsi/projects/odette/>.

[5] CAI, L., VERMA, S., GAJSKI, D. *Comparison of SpecC and SystemC Languages for System Design*, CECS, UC Irvine, Technical Report CECS-TR-03-11, May 2003.

[6] GAJSKI, D., PENG, J., GERSTLAUER, A. YU, H., SHIN, D. "System Design Methodology and Tools," CECS, UC Irvine, Technical Report CECS-TR-03-02, January 2003.

[7] CoWare Inc. *ConvergenSC Advanced System Designer*. Acessado em: 10/06/05. Disponível em: <http://www.coware.com/products/convergensc_asd.php>.

[8] MÜLLER, W., ROSENSTIEL, W., RUF, J. *SystemC – Methodologies and Applications*. Kluwer Academic Publishers, USA, 2003.

[9] BERNER, D. *Development of a visual refinement and exploration tool for SpecC*. Masterthesis of David Berner, Center for Embedded Computer Systems, University of California, Irvine, CA, USA. February 22, 2001.

[10] GERSTLAUER, A., GAJSKI, D. *System-Level Abstraction Semantics*, Technical Report CECS-02-17. July 12, 2002. Center for Embedded Computer Systems, University of California, Irvine, USA.

[11] SCHUTTEN, R. *Raising the Level of Abstraction Reduces System-on-Chip Verification*, March 2004, Synopsys, Inc.

[12] IEEE. *IEEE ratifies SystemC 2.1 standard for System-Level chip design.* Disponível em: <http://www.ieee.org>. Acessado em: 10/01/2006.

[13] ABDI, S., GAJSKI, D. *Provably Correct Architecture Refinement*. Technical Report CECS-03-29. September 30, 2003. Center for Embedded Computer Systems, University of California, Irvine, USA.

[14] MCFARLAND, M., PARKER, A., CAMPOSANO, R. The high-level synthesis of digital systems. Proceedings of the IEEE, special issue on *The future of computer-aided design*, Vol. 78, No. 2, pp. 301-318, Feb. 1990.

[15] GONG, J., GAJSKI, D., BAKSHI, S. *Model Refinement of Hardware-Software Codesign.* University of California, Irvine, ACM Transactions on Design automation of Electronic Systems, Vol. 2, No. 1, pp. 22-41, Jan. 1997.

[16] BHASKER, J. *A SystemC Primer.* Star Galaxy Publishing, 2002.

[17] DIAS, S., SILVA Jr., D. *Proposta de uma ferramenta de apoio ao projetista de sistemas embutidos usando SystemC.* 1ª Semana de Pós-Graduação do Centro de Pesquisa e Desenvolvimento em Engenharia Elétrica/UFMG, setembro, 2005.

[18] Sypnopsys Inc. *CoCentric SystemC Compiler Behavioral Modeling Guide.* Version 2000.11-SCC1, March 2001.

[19] WALSTROM, R. *System Level Design Refinement using SystemC*. Master thesis. Iowa State University, Ames, Iowa, 2004.

[20] GRÖTKER, T., LIAO, S., MARTIN, G., SWAN, S. *System Design with SystemC*. Kluwer Academic Publishers – Boston/Dordrecht/London, 2002.

[21] Open SystemC Initiative. *SystemC Version 2.0 User's Guide – Update for SystemC 2.0.1*. 2002. Disponível em: <http://www.systemc.org>.

[22] Open SystemC Initiative. *SystemC Synthesizable Subset – Draft 1.1.18*. December 23, 2004. Disponível em: <http://www.systemc.org>.

[23] Open SystemC Initiative. *Functional specification for SystemC 2.0 – Update for SystemC 2.0.1*. Version 2.0-Q. April 5, 2002. Disponível em: <http://www.systemc.org>.

[24] Open SystemC Initiative. *SystemC 2.0.1 Language Reference Manual – Revision 1.0*. 2003. Disponível em: <http://www.systemc.org>.

[25] Sypnopsys Inc. *CoCentric SystemC Compiler RTL User and Modeling Guide.* Version U-2003.06, June 2003.

[26] EDWARDS, S. A. *The Challenges of Hardware Synthesis from C-like Languages.* Design, Automation and Test in Europe, 2005. Proceedings. 7-11 March 2005. On page(s): 66- 67 Vol. 1.

[27] DE MICHELI, G. *Hardware Synthesis from C/C++ Models*. Proc. Design, Automation and Test in Europe Conference and Exhibition, March 1999.

[28] AL-JUNAID, H. and KAZMIERSKI, T. An Analogue and Mixed-Signal Extension to SystemC. *The Institution of Electrical Engineers Proceedings Circuits, Devices & Systems,* 2005.

[29] HYDE, D. C. *CSCI 320 Computer Architecture Handbook on Verilog HDL*. Computer Science Department, Bucknell University, Lewisburg, August, 1997.

[30] ASHENDEN, P. J. *The VHDL cookbook*. University of Adelaide, South Australia, Technical Report 1990.

[31] HABIBI, A., TAHAR, S. A survey on System-On-a-Chip Design Languages. in *IEEE 3rd International Workshop on System-on-Chip IWSOC*, (Alberta, Canada), June-July 2003, pp.212-215.

[32] SystemVerilog. Acessado em: 15/07/2006. Disponível em: <http://www.systemverilog.org>.

[33] Doulos Ltd. *The Designer's Guide to VHDL*. Acessado em: 10/07/2006. Disponível em: <http://www.doulos.com/knowhow/vhdl_designers_guide>.

[34] Accellera Organization Inc. Acessado em: 10/07/2006. Disponível em: <http://www.accellera.org/home>.

[35] Doulos Ltd. *The Designer's Guide to Verilog*. Acessado em: 10/07/2006. Disponível em: <http://www.doulos.com/knowhow/verilog_designers_guide>.

[36] SpecC Technology Open Consortium. Acessado em: 25/05/2006. Disponível em: <http://www.specc.gr.jp/eng/index.htm>.

[37] Celoxica Ltd. *Handel-C Language Reference Manual*. Version 2.1, 1998. Disponível em: <http://www.celoxica.com>. Acessado em: 10/05/2005.

[38] Celoxica Ltd. *Introducing Software Paradigms to Hardware Design*. August, 2002. Disponível em: <http://www.celoxica.com/techlib/files/CEL-W0307171L56-64.pdf>. Acessado em: 15/07/2006.

[39] KU, D., DE MICHELI, G. *HardwareC -- A Language for Hardware Design (version 2.0)*, Computer Systems Lab Technical Report CSL-TR-90-419, Stanford University, Califórnia, August 1990.

[40] KU, D., DE MICHELI, G. "Hercules - a system for high-level synthesis," in Proc. of *the 25th ACM/IEEE Conference on Design Automation*, Atlantic City, New Jersey, United States, 1988, pp. 483–488.

[41] CynApps, Inc. *Cynlib: A C++ Class Library for Hardware Description Reference Manual*. Santa Clara, CA, USA, 1999.

[42] CynApps, Inc. *Cynlib User's Guide – Release 1.0 Beta*. Santa Clara, CA, USA, 1999.

[43] Open SystemC Initiative. Acessado em: 10/04/2004. Disponível em: <http://www.osci.org>.

[44] GERSTLAUER, A. *SpecC Modeling Guidelines*. Technical Report CECS-02-16. Center for Embedded Computger Systems, University of California, Irvine, USA, April, 2002.

[45] PANDA, P. *SystemC – A modeling platform supporting multiple design abstractions.* Synopsys Inc. ISSS'01, October 1-3, 2001, Montreal, Québec, Canadá.

[46] CAI, L., et al. *Top-Down System Level Design Methodology Using SpecC, VCC and SystemC.* Proceedings of the 2002 Design, Automation and Test in Europe Conference and Exhibition (DATE'02) 1530-1591/02, 2002 IEEE.

[47] GRIMPE, E. et al. *SystemC Object-Oriented Extensions and Synthesis Features*. In Proc. of the Forum on Specification & Design Languages (FDL'02), Marseille, France, 2002.

[48] DÖMER, R. *System-level Modeling and Design with the SpecC Language*. Dissertation, University of Dortmund, Germany, 2000.

[49] Cadence Design Systems. Acessado em: 20/05/05. Disponível em: <http://www.cadence.com>.

[50] Cadence Design Systems. *Incisive Unified Simulator Datasheet 2005*. Acessado em: 20/05/05. Disponível em: <http://www.cadence.com>.

[51] Cadence Design Systems. *Soc Verification Challenges*. Acessado em: 25/05/05. Disponível em: < http://www.cadence.com/support/education/verification.aspx>.

[52] Synopsys Inc. *Getting Started with CoCentric System Studio*. Version U-2003.03, March 2003. Acessado em: 15/08/05. Disponível em: <http://www.synopsys.com>.

[53] Summit Design Inc. *Vista Design Environment for SystemC*. 2006. Acessado em: 15/07/06. Disponível em: <http://www.summit-design.com/products/pdfs/Web_Vista_8_29.pdf>.

[54] BERNER, D. et al. *SystemCXML: An Extensible SystemC Front End Using XML*, Proceedings of the Forum on specification and design languages (FDL). Lausanne, Switzerland, September 2005.

[55] SourceForge.net. Acessado em: 12/07/05. Disponível em: <http://sourceforge.net>.

[56] Veripool – Free Verilog Software. SystemPerl. Acessado em: 02/05/06. Disponível em: <http://www.veripool.com/systemperl.html>.

[57] MOY, M., MARANINCHI, F., MAILLET-CONTOZ. M. *Pinapa: An Extraction Tool for SystemC descriptions of Systems-on-a-Chip*. EMSOFT, 2005, September. Acessado em: 15/12/2005. Disponível em: <http://www-verimag.imag.fr/~moy/publications/sc-compil.pdf>, Código fonte em: <http://greensocs.sourceforge.net/pinapa>.

[58] Forte Design Systems. *Cynthesizer Closes the ESL-to-Silicon Gap*. Acessado em 02/07/05. Disponível em: <http://www.forteds.com/products/cynthesizer.asp>.

[59] FEY, G. et al. SyCE: An Integrated Environment for System Design in SystemC. *IEEE International Workshop on Rapid System Prototyping 2005*: 258-260.

[60] FEY, G. et al. ParSyC: An efficient SystemC parser. *In 12th Workshop on Synthesis And System Integration of Mixed Information technologies (SASIMI 2004)*, pages 148–154, Kanazawa, 2004.

[61] GROßE, D. and DRECHSLER, R. Checkers for SystemC designs. *In Second ACM & IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, pages 171–178, 2004.

[62] GROßE, D. and DRECHSLER, R. CheckSyC: An efficient property checker for RTL SystemC designs. *In IEEE International Symposium on Circuits and Systems*, 2005.

[63] GROßE, D. et al. Efficient automatic visualization of SystemC designs. *In Forum on Specification and Design Languages*, pages 646–657, 2003.

[64] SINHA, V. et al. YAML: A Tool for Hardware Design Visualization and Capture. *International Symposium on Systems Synthesis. Proceedings of the 13th international symposium on System synthesis*, 2000, Madrid, Spain.

[65] VANDERPERREN, Y., DEHAENE, W. UML 2 and SysML: An Approach to Deal with Complexity in SoC/NoC Design, *date*, pp. 716-717, *Design, Automation and Test in Europe (DATE'05)* Volume 2, 2005.

[66] MOY, M., MARANINCHI, F., MAILLET-CONTOZ. M. LusSy: A Toolbox for the Analysis of Systems-on-a-Chip at the Transactional Level. *International Conference on Application of Concurrency to System Design*, 2005, June. Acessado em: 15/12/2005. Disponível em: <http://www-verimag.imag.fr/~moy/publications/acsd05.pdf>.

[67] ICODES Project. Acessado em: 10/09/06. Disponível em: < http://icodes.offis.de/>.

[68] GAJSKI, D., VAHID, F. *Specification and Design of Embedded Hardware-Software Systems*. IEEE Design & Test of Computers, Spring 1995.

[69] CHEVALIER, J. "A SystemC Refinement Methodology for Embedded Software", *IEEE Design and Test of Computers* ,vol. 23, no. 2, pp. 148-158, March/April, 2006.

[70] DÉHARBE, D., MEDEIROS, S. *Aspect-Oriented Design in SystemC: Implementation and Applications*. In: Proceedings of the 19th Brazilian Symposium on Integrated Circuits and System Design, 2006, Ouro Preto, MG. (SBCCI2006). p. 1-6.

[71] SCHLEBUSCH, H-J. *SystemC based Hardware Synthesis Becomes Reality*, Synopsys System Level Design SystemC at Euromicro DSD, Maastricht, The Netherlands.

[72] ECONOMAKOS, G. et al, *Behavioral Synthesis with SystemC*, National Technical University of Athens, Department of Electrical and Computer Engineering, Greece, IEEE 2001.

[73] GERLACH, J. and ROSENSTIEL, W. *System level design using the SystemC modeling platform*. In SDL'2000.

[74] GHOSH, A., TJIANG, S., CHANDRA, R. *System Modeling with SystemC*, Synopsys Inc., Tensilica Inc. e STMicroeletronics, International Conference on ASIC, pp.18-20, 2001.

[75] LIAO, S. Y. *Towards a New Standard for System-Level Design*. In Proceedings of the Eighth International Workshop on Hardware/Software Codesign, San Diego, CA, USA, May 2000.

[76] KLINGAUF, W. *Systematic Transaction Level Modeling of Embedded Systems with SystemC*. Technical University of Braunschweig, Abt. E.I.S., Germany, IEEE 2005.

[77] CAI, L., GAJSKI, D. *Transaction Level Modeling: An Overview*. Center of Embedded Computer Systems, University of California, USA, 2003.

[78] CALDARI, M., et. al. *Transaction Level Model for AMBA Bus Architecture Using SystemC 2.0*. Proceedings of the Design Automation and Test in Europe.

[79] PASRICHA, S. *Transaction Level Modeling of SoC with SystemC. Design Flow and Reuse*, STMicroelectronics Ltd, SNUG, Bangalore, India, 2002.

[80] CALAZANS, N., et. al. *From VHDL Register Transfer Level to SystemC Transaction Level Modeling: a Comparative Case Study*. Pontifícia Universidade Católica do Rio Grande do Sul, Brasil, In: 16 Symposium on Integrated Circuits and Systems Design, (SBCCI03). IEEE Computer Society Press, 2003.

[81] MOUSSA, I., GRELLIER, N., *Exploring SW Performance using SoC Transaction-Level Modeling*. Design, Automation and Test in Europe Conference and Exhibition, 2003, TNI-Valiosys, France, 2003.

[82] SPINCZYK, O., GAL, A., and SCHRÖDER-PREIKSCHAT, W. *AspectC++: an aspect-oriented extension to the C++ programming language*. In Proceedings of the 40th International Conference on Tools Pacific, pages 53–60, 2002. Australian Computer Society.

[83] BLACK, D., DONOVAN, J. *SystemC: From The Ground Up*. Kluwer Academic Publishers. Boston, 2004.

[84] Celoxica Inc. *Agility Compiler Product Brief Version 1.2*. 2006. Disponível em: <http://www.celoxica.com/agility>. Acessado em: 17/10/2006.

[85] Synopsys Inc. *Describing Synthesizable RTL in SystemC*. Version 1.0, May 2001, USA. Disponível em: <http://www.synopsys.com>. Acessado em: 09/05/2005.

[86] Synopsys Inc. *Synopsys Design Compiler 2005 Datasheet*. 2005, USA. Disponível em: <http://www.synopsys.com>. Acessado em: 17/10/2006.

[87] EIBL, C., ALBRECHT, C. and HAGENAU, R. gSysC: A Graphical Front End for SystemC. *Proceedings 19th European Conference on Modelling and Simulation*, 257-262, ECMS, Riga, Latvia 2005.

[88] Prosilog Supports Eclipse with Magillem 3.0 Platform Based Design Tool. Disponível em: <http://www.embeddedstar.com/press/content/2005/5/embedded18360.html.> Acessado em: 20/10/2006.

[89] Magillem Graphical Platform Builder for SoCs (Prosilog AS 6136). Disponível em: <http://www.soccentral.com/results.asp?CategoryID=345&entryID=6136>. Acessado em: 24/10/2006.

[90] GHENASSIA, F. (Ed). *Transaction Level Modeling with SystemC – TLM Concepts and Applications for Embedded Systems*. Springer, 2005, Dordrecht, The Netherlands.

[91] ROSE, A., SWAN, S., PIERCE, J., FERNANDEZ, J-M. *Transaction level modeling in SystemC*. Technical report, Mentor Graphics; Cadence Design Systems, 2005.

[92] PASRICHA, S. *Transaction level modeling of SoC with SystemC 2.0*. In Synopsys User Group Conference (SNUG), 2002, Bangalore, India.

[93] SWAN, S., *Introduction to Transaction Level Modeling in SystemC*. Cadence Design Systems, Inc. September, 2005.

[94] DIAS, S. , SILVA Jr., D. *The Successive Refinement Methodology in the Embedded Systems Design*. VI Microelectronics Student Forum 2006. Chip on the Mountains. Ouro Preto, Brazil, august, 2006.

[95] BELL, C., NEWELL, A. *Computer Structures: Readings and Examples*. New York, NY: McGraw-Hill, 1971.

[96] KRAUSE, M., BRINGMANNN, O., ROSENSTIEL, W. *A SystemC-based Software and Communication Refinement Framework for Distributed Embedded Systems*. 13th Workshop on Synthesis And System Integration of Mixed Information Technologies, Nagoya, 2006.

[97] AYNSLEY, J. *SystemC in Europe - Current Usage and Future Requirements*, Doulos Ltd, 40th Design Automation Conference, June 2003.

[98] MCCLOUD, S. *Mentor Graphics European ESL Survey 2005*, Mentor Graphics Ltd, August, 2005.

[99] Celoxica Ltd. *Survey of System Design Trends*. Survey Analisys, Celoxica Ltd. August, 2005.

[100] BALL, R. *Survey cuts through the ESL hype*. August, 2005. Disponível em: <http://www.electronicsweekly.com/ARTICLES/2005/08/30/36158/Survey+cuts+through+the+ESL+hype.HTM>. Acessado em: 04/12/2006.

[101] GHENNASSIA, F., DUCOUSSO, L., BULONE, J., KHAN, N., VIZINHO-COUTRY, A. *SystemC based Virtual SoC – an Integrated System Level and Block Level Verification approach from Simulation to Co-Emulation*. Technical Paper, November 16, 2004. Acessado em 10/05/05. Disponível em: http://www.cadence-europe.com/eEuronews/may_05/images/SystemC.pdf.

Anexo A Conjunto não sintetizável de SystemC

Categoria	Estrutura	Comentário	Ação Corretiva
Processo Thread	`SC_THREAD`	Usado para modelar um testbench, mas não suportado na síntese.	Substituir por `SC_CTHREAD` com as devidas alterações.
Processo Method	`SC_METHOD`	Usado para modelagem e simulação no nível RTL, mas não suportado para síntese no BHM.	Substituir por `SC_CTHREAD` com as devidas alterações.
Canais	`sc_channel`	Usado somente nos estágios iniciais de modelagem funcional do sistema.	Substituir por `sc_signal`.
Geradores de clock	`sc_start()`	Usado para simulação.	
Portas bidirecionais	`sc_inout`	Não são permitidas.	Substituir por portas separadas para entrada (`sc_in`) e saída (`sc_out`).
Watching local	`W_BEGIN, W_END, W_DO, W_ESCAPE`	Não suportados.	Excluir ou deixar dentro de um comentário no código.
Múltiplos resets globais	Múltiplos `watching()`	Um reset global é suportado para síntese, porém muitos não são possíveis.	Combinar múltiplos resets em um único reset usando um operador AND.
Varredura	`sc_trace, sc_create* trace_file`	Criam formas de onda de sinais, canais e variáveis para simulação.	Excluir ou deixar dentro de um comentário no código.

Tabela 6 – Conjunto não sintetizável de SystemC para síntese BHM, [18]

Categoria	Estrutura	Comentário	Ação Corretiva
Processo Thread	`SC_THREAD`	Usado para modelar um testbench, simulação e modelagem a nível comportamental, mas não suportado na síntese.	Substituir por `SC_METHOD` com as devidas alterações.
Processo CThread	`SC_CTHREAD`	Usado para modelagem e simulação no nível BHM.	Substituir por `SC_METHOD` com as devidas alterações.
Função principal	`sc_main()`	Usado para simulação.	
Canais	`sc_channel`	Usado somente nos estágios iniciais de	Substituir por `sc_signal`.

Categoria	Estrutura	Comentário	Ação Corretiva
		modelagem funcional do sistema.	
Geradores de clock	`sc_start()`	Usado para simulação.	Use somente no `sc_main`.
Comunicação	`sc_interface`, `sc_port`, `sc_mutex`, `sc_fifo`	Usado para modelagem de comunicação.	Deixar dentro de um comentário no código para síntese.
Watching global	`watching()`	Não suportado para síntese RTL.	Excluir ou deixar dentro de um comentário no código.
Watching local	`W_BEGIN`, `W_END`, `W_DO`, `W_ESCAPE`	Não suportados.	Excluir ou deixar dentro de um comentário no código.
Sincronização	Biblioteca master-slave de SystemC	Usado para sincronização de eventos	Deixar dentro de um comentário no código para síntese.
Varredura	`sc_trace`, `sc_create* trace_file`	Criam formas de onda de sinais, canais e variáveis para simulação.	Excluir ou deixar dentro de um comentário no código.

Tabela 7 – Conjunto não sintetizável de SystemC para síntese RTL, [25]

Anexo B Conjunto não sintetizável de C/C++

Categoria	Estrutura	Comentário	Ação Corretiva
Declaração de classe local	`class`	Não permitido.	Substituir.
Declaração de classe aninhada	`class`	Não permitido.	Substituir.
Classe derivada	`class`	Somente módulos SystemC e processos são permitidos.	Substituir.
Alocação dinâmica de memória	`malloc(), free(), new(), new[], delete, delete[]`	Não suportado.	Usar alocação estática de memória.
Tratamento de exceção	`try, catch, throw`	Não permitido.	Excluir ou deixar dentro de um comentário no código.
Chamada recursiva de função		Não permitido.	Substituir por iterações.
Overloading de funções		Não permitido (exceto para classes de SystemC).	Substituir por chamadas únicas de função.
Funções do C++		As bibliotecas de matemática, entrada e saída, manipulação de arquivos, além da biblioteca de funções do C++ não são permitidas.	Substituir ou deixar dentro de um comentário no código.
Função virtual	`virtual`	Não permitido.	Substituir por função não virtual.
Herança	`class X : public Y {`	Não permitido.	Substituir.
Herança múltipla		Não permitido.	Substituir.
Especificadores de controle de acesso de membros	`public, protected, private, friend`	Permitido no código, mas são ignorados na síntese. Todos os acessos são públicos.	Mudar para público, os acessos a funções e classes, ou ignorar as advertências (warnings) do compilador.
Acesso a membros de uma `struct` usando o operador "->"	Operador "->"	Não permitido.	Substituir pelo acesso usando o operador ".".

Categoria	Estrutura	Comentário	Ação Corretiva
Membro estático	`static`	Não permitido.	Substituir por membros não estáticos ou variáveis.
Operador de referência	Operadores "*" e "&"	Não permitido.	Substituir pelo acesso direto à variável ou vetor.
Operador `sizeof`	`sizeof`	Não permitido.	Determinar o tamanho estaticamente para uso em síntese,
Ponteiro	"*"	Ponteiros são permitidos somente em módulos hierárquicos que não são suportados pelos compiladores correntes de SystemC. Um *char é tratado como uma string e não como um ponteiro para a memória.	Substituir todos os ponteiros por acesso aos elementos do vetor ou por elementos individuais.
Conversão de tipo de ponteiro		Não permitido.	Não usar ponteiros. Usar referência explícita à variável.
Ponteiro `this`	`this`	Não permitido.	Substituir.
Referência, C++	"&"	Permitido somente para passagem de parâmetros para funções.	Substituir.
Conversão de referência		É suportada para conversão implícita de sinais somente.	Substituir.
Classes "`template`" definidas pelo usuário	`template <class T>`	Somente classes "templates" SystemC como sc_int<> são suportados.	Substituir.
Conversão de tipo em tempo de execução		Não permitido.	Substituir.
Identificação de tipo em tempo de execução		Não permitido.	Substituir.
Conversão explícita de tipo definido pelo usuário		Os tipos nativos de C++ e systemC são suportados para conversão explícita.	Substituir em todos os outros casos.
Salto incondicional	`goto`	Não permitido.	Escrever código estruturado com `break` e `continue`.
Uso de `unions`	`union`	Não permitido.	Substituir por `struct`.
Variáveis globais		Não suportado para síntese.	Substituir por variáveis locais.
Variáveis compartilhadas		Variáveis acessadas por dois ou mais processos SC_THREAD não são suportadas. De qualquer forma o acesso à variável é permitido a somente um processo.	Usar sinais ao invés de variáveis para comunicação entre processos.
Variáveis voláteis	`volatile`	Não permitido.	Uso de variáveis não voláteis.

Tabela 8 – Conjunto não sintetizável de C/C++, [18]

Anexo C Conjunto sintetizável de SystemC

Tipo SystemC e C++	Descrição
sc_bit	Um bit simples de valor verdadeiro (true) ou falso (false).
sc_bv<n>	Vetor de bits de tamanho arbitrário.
sc_logic	Um bit 0, 1, X, ou Z para síntese RTL somente.
sc_lv<n>	Um vetor de sc_logic de tamanho arbitrário para síntese RTL somente.
sc_int<n>	Inteiro de precisão fixa, restrito a até 64 bits.
sc_uint<n>	Inteiro de precisão fixa, restrito a até 64 bits, sem sinal.
sc_bigint<n>	Inteiro de precisão arbitrária, recomendado para tamanhos acima de 64 bits.
sc_biguint<n>	Inteiro de precisão arbitrária, recomendado para tamanhos acima de 64 bits, sem sinal.
bool	Um bit simples de valor verdadeiro (true) ou falso (false).
int	Inteiro, tipicamente 32 ou 64 bits, dependendo da plataforma.
unsigned int	Inteiro, tipicamente 32 ou 64 bits, dependendo da plataforma, sem sinal.
long	Inteiro, tipicamente 32 ou 64 bits, ou maior, dependendo da plataforma.
unsigned long	Inteiro, tipicamente 32 ou 64 bits, ou maior, dependendo da plataforma, sem sinal.
char	Inteiro, com sinal, para representar caracteres individuais ou pequenos inteiros, tipicamente de -128 a 127.
unsigned char	Inteiro, sem sinal, para representar caracteres individuais ou pequenos inteiros, tipicamente de 0 a 255.
short	Inteiro, tipicamente 32 bits, dependendo da plataforma.
unsigned short	Inteiro, tipicamente 32 bits, dependendo da plataforma, sem sinal.
struct	Agregação de tipos de dados sintetizáveis, definido pelo usuário.
enum	Tipo de dado enumerado, definido pelo usuário, associado com uma constante inteira.

Tabela 9 – Conjunto sintetizável de SystemC e C/C++, [18]

Operadores	sc_bit	sc_bv
Bit a bit & (and), | (or), ^ (xor), e ~ (not)	sim	sim
Bit a bit << (shift esquerda) e >> (shift direita)	não	sim
Atribuição =, &=, |=, e ^=	sim	sim
Igualdade ==, !=	sim	sim
Seleção de bits [x]	não	sim
Seleção de faixa de bits (x-y)	não	sim
Concatenação (x,y)	não	sim
Redução: and_reduce(), or_reduce(), e xor_reduce()	não	sim

Tabela 10 – Operadores para os tipos de dados (SystemC) Bit e Bit Vector, [18]

Operadores	sc_int, sc_uint, sc_bigint, sc_biguint
Bit a bit & (and), | (or), ^ (xor), e ~ (not)	sim
Bit a bit << (shift esquerda) e >> (shift direita)	sim
Atribuição =, &=, |=, ^=, +=, -=, *=, /=, %=	sim
Igualdade ==, !=	sim
Relacional <, <=, >, >=	sim
Autoincremento ++ e autodecremento --	sim
Seleção de bits [x]	sim
Seleção de faixa de bits (x-y)	sim
Concatenação (x,y)	sim
Redução: and_reduce(), or_reduce(), e xor_reduce()	sim

Tabela 11 – Operadores para os tipos de dados (SystemC) Integer, [18]

Anexo D Código do Executable Specification Model

6.3 1esm_gcd.h

```
/*
 gcd1_esm.h - gcd in the Executable
Specification Model
 author: Sandro Renato Dias
 last date: 09/04/2006
 version: 1.0
*/

#include "systemc.h"

template <class T> SC_MODULE ( gcd ){

  // Portas
  sc_fifo_in<T> input_a, input_b;
  sc_fifo_out<T> result;

  // Processos
  void prc_gcd();

  // Construtor do modulo
  SC_CTOR( gcd ){
    SC_THREAD( prc_gcd );
    sensitive << input_a << input_b;
  }
};
```

6.4 1esm_gcd.cpp

```
/*
 1esm_gcd.cpp - gcd in the Executable
Specification Model
 author: Sandro Renato Dias
 last date: 09/04/2006
 version: 1.2
*/

#include "1esm_gcd.h"

void gcd::prc_gcd() {

   // Variaveis
   T tmp_a, tmp_b, tmp_result;

   // Leitura das portas
   tmp_a = input_a.read();
   tmp_b = input_b.read();

   if ((tmp_a == 0) | (tmp_b == 0))
      tmp_result = 0;
   else {
     while( 1 ) {
       tmp_a = tmp_a % tmp_b;
       if (tmp_a == 0) {
         tmp_result = tmp_b;
         break;
       }
       tmp_b = tmp_b % tmp_a;
       if (tmp_b == 0) {
         tmp_result = tmp_a;
         break;
       } // if
     } // while
   } // else
   result.write(tmp_result);
} // gcd
```

6.5 1esm_stimulus.h

```
/*
 stimulus.h - stimulates the gcd input
 author: Sandro Renato Dias
 last date: 19/04/2006
 version: 2.0
*/

#include "systemc.h"
#include "fstream.h"
#include "iostream.h"

template <class T> SC_MODULE( stimulus ){

  // Arquivos
  ifstream stimulus_file;

  // Portas
  sc_in<bool> clk;                    // -
-> Clock automatico
  sc_fifo_out<T> output_a, output_b;   // -
-> portas do modulo principal

  // Processos
  void prc_stimulus();
```

```
  // Construtor do modulo
  SC_CTOR( stimulus ){
    SC_THREAD( prc_stimulus );
      sensitive_neg << clk;

       stimulus_file.open
("1esm_stimulus_file");
       if (!stimulus_file) {
           cerr << "ERROR: Unable to open
vector file stimulus_file!\n";

           sc_stop(); // stop simulation
       }
  }

  // Fecha o arquivo no destrutor
  ~stimulus() {
       stimulus_file.close();
  }

};
```

6.6 1esm_stimulus.cpp

```
/*
 stimulus.cpp - stimulates the gcd input
 author: Sandro Renato Dias
 last date: 19/04/2006
 version: 2.0
*/

#include "stimulus.h"

void stimulus::prc_stimulus() {

   // Variaveis
   int a, b;

   // Leitura linha a linha
   while (stimulus_file >> a >> b) {
        output_a.write(a);
        output_b.write(b);
        wait(10, SC_NS);
   }

   sc_stop();
}
```

6.7 1esm_stimulus_file

```
250 50
25 5
20 5
120 10
200 120
```

6.8 1esm_monitor.h

```
/*
 monitor.h - monitoring the gcd output
 author: Sandro Renato Dias
 last date: 05/04/2006
 version: 1.1
*/

#include "systemc.h"

template <class T> SC_MODULE ( monitor ){

  // Portas
  sc_fifo_in<T> input_a, input_b;
  sc_fifo_in<T> result;

  // Processos
  void prc_monitor();

  // Construtor do modulo
  SC_CTOR( monitor ){
    SC_THREAD( prc_monitor );
    sensitive << input_a << input_b <<
result;
  }
};
```

6.9 1esm_monitor.cpp

```
/*
 monitor.cpp - monitoring the gcd output
 author: Sandro Renato Dias
 last date: 05/04/2006
 version: 1.1
*/

#include "monitor.h"

void monitor::prc_monitor() {

   // Variaveis
   T tmp_a, tmp_b, tmp_result;

   tmp_a = input_a.read();
   tmp_b = input_b.read();
   tmp_result = result.read();

   cout << "\ngcd (" << tmp_a << ", " <<
tmp_b << ") = "
        << tmp_result << endl;
}
```

6.10 1esm_main.cpp

```
/*
 main.cpp - main file for the gcd
 author: Sandro Renato Dias
 last date: 09/04/2006
 version: 1.0
*/

#include "1esm_stimulus.h"
#include "1esm_gcd.h"
#include "1esm_monitor.h"

int sc_main(int argc, char* argv[]){

  sc_trace_file *tf =
sc_create_vcd_trace_file("1esm_gcd");

  sc_clock clk;
  sc_fifo<int> a, b, r;

  stimulus<int> stimulus_i ("stimulus_i");
  stimulus_i.clk(clk);
  stimulus_i.output_a(a);
  stimulus_i.output_b(b);

  gcd<int> gcd_i ("gcd_i");
  gcd_i.input_a(a);
  gcd_i.input_b(b);
  gcd_i.result(r);

  monitor<int> monitor_i ("monitor_i");
  monitor_i.input_a(a);
  monitor_i.input_b(b);
  monitor_i.result(r);

  sc_trace(tf, a, "a");
  sc_trace(tf, b, "b");
  sc_trace(tf, r, "result");
  sc_trace(tf, clk, "clock");

  sc_start(-1);

  sc_close_vcd_trace_file(tf);

  return 0;
}
```

6.11 1esm_Makefile.linux

```
TARGET_ARCH = linux

CC     = g++
OPT    = -O3
DEBUG  = -g
OTHER  = -Wall
CFLAGS = $(OPT) $(OTHER)
# CFLAGS = $(DEBUG) $(OTHER)

MODULE = gcd
SRCS = 1esm_gcd.cpp 1esm_stimulus.cpp
1esm_monitor.cpp 1esm_main.cpp
OBJS = $(SRCS:.cpp=.o)

include Makefile.defs
```

Anexo E Código do Timed Data-Flow Model

6.12 2tdm_gcd.h

```
/*
 gcd2_tdm.h - gcd in the Timed Data-Flow
Model
 author: Sandro Renato Dias
 last date: 21/10/2006
 version: 1.3
*/

#include "systemc.h"

#define READ_LATENCY “10, SC_NS”
#define WRITE_LATENCY “10, SC_NS”
#define COMP_LATENCY “5, SC_NS”

// Interfaces para comunicacao
template <class T> class gcd_write_if :
virtual public sc_interface {
public:
  // Metodos da interface
  virtual void gcd_write(const T&) = 0;
};

template <class T> class gcd_read_if :
virtual public sc_interface {
public:
  // Metodos da interface
  virtual const T& gcd_read() const = 0;
};

template <class T> SC_MODULE ( gcd ){

  // Portas
  sc_port<gcd_read_if> input_a, input_b;
  sc_port<gcd_write_if> result;
  sc_port<gcd_read_if> reset;

  // Processos
  void prc_gcd();

  // Construtor do modulo
  SC_CTOR( gcd ){
    SC_THREAD( prc_gcd );
    sensitive << input_a << input_b <<
reset;
  }
};
```

6.13 2tdm_gcd.cpp

```
/*
 2tdm_gcd.cpp - gcd in the Timed Data-Flow
Model
 author: Sandro Renato Dias
 last date: 21/10/2006
 version: 1.3
*/

#include "2tdm_gcd.h"

// Implementacao da interface de escrita
void gcd_write_if::gcd_write(const T& tmp)
{
   (this)->write(tmp);
   return *this;
}

// Implementacao da interface de leitura
const T& gcd_read_if::gcd_read() {
   return (*this)->read();
}

void gcd::prc_gcd() {

   // Variaveis
   T tmp_a, tmp_b, tmp_reset, tmp_result;

   // Leitura das portas
   // Uso da interface
   tmp_a = input_a->read();
   tmp_b = input_b->read();
   tmp_reset = reset->read();
   // Atraso da leitura
   wait (READ_LATENCY);

   if ((tmp_a == 0) | (tmp_b == 0) |
tmp_reset) {
      tmp_result = 0;
   }
   else {
     while( 1 ) {
       tmp_a = tmp_a % tmp_b;
       // Atraso da computacao
       wait (COMP_LATENCY);

       if (tmp_a == 0) {
           tmp_result = tmp_b;
           break;
       }
       tmp_b = tmp_b % tmp_a;
       // Atraso da computacao
```

```
       wait (COMP_LATENCY);

       if (tmp_b == 0) {
           tmp_result = tmp_a;
           break;
       } // if
     } // while
   } // else

   result->write(tmp_result);
   // Atraso da escrita
   wait (WRITE_LATENCY);
} // gcd
```

6.14 2tdm_stimulus.h

```
/*
 stimulus.h - stimulates the gcd input
 author: Sandro Renato Dias
 last date: 19/04/2006
 version: 2.0
*/

#include "systemc.h"
#include "fstream.h"
#include "iostream.h"

template <class T> SC_MODULE( stimulus ){

  // Arquivos
  ifstream stimulus_file;

  // Portas
  sc_in<bool> clk;
  sc_port<gcd_write_if> output_a, output_b;
  sc_port<gcd_write_if> reset;

  // Processos
  void prc_stimulus();

  // Construtor do modulo
  SC_CTOR( stimulus ){
    SC_THREAD( prc_stimulus );
      sensitive_neg << clk;

       stimulus_file.open
("2tdm_stimulus_file");
       if (!stimulus_file) {
           cerr << "ERROR: Unable to open
vector file stimulus_file!\n";

           sc_stop(); // stop simulation
       }
  }

  // Fecha o arquivo no destrutor
  ~stimulus() {
       stimulus_file.close();
  }
};
```

6.15 2tdm_stimulus.cpp

```
/*
 stimulus.cpp - stimulates the gcd input
 author: Sandro Renato Dias
 last date: 19/04/2006
 version: 2.0
*/

#include "2tdm_stimulus.h"

void stimulus::prc_stimulus() {

   // Variaveis
   int a, b;
   bool rs;

   // Leitura linha a linha
   while (stimulus_file >> a >> b >> rs) {
       output_a->write(a);
       output_b->write(b);
       reset->write(rs);
       wait(10, SC_NS);
   }
   sc_stop();
}
```

6.16 2tdm_monitor.h

```
/*
 monitor.h - monitoring the gcd output
 author: Sandro Renato Dias
 last date: 05/04/2006
 version: 1.1
*/

#include "systemc.h"

template <class T> SC_MODULE ( monitor ){

  // Portas
  sc_port<gcd_read_if> input_a, input_b;
  sc_port<gcd_read_if> reset, result;

  // Processos
  void prc_monitor();

  // Construtor do modulo
  SC_CTOR( monitor ){
    SC_THREAD( prc_monitor );
    sensitive << input_a << input_b <<
result;
  }
};
```

6.17 2tdm_monitor.cpp

```
/*
 monitor.cpp - monitoring the gcd output
 author: Sandro Renato Dias
 last date: 05/04/2006
 version: 1.1
*/

#include "2tdm_monitor.h"

void monitor::prc_monitor() {

   // Variaveis
   T tmp_a, tmp_b, tmp_reset, tmp_result;

   tmp_a = input_a->read();
   tmp_b = input_b->read();
   tmp_result = result->read();
   tmp_reset = reset->read();

   cout << "\ngcd (" << tmp_a << ", " <<
tmp_b << ") = "
        << tmp_result << ", reset = " <<
tmp_reset << endl;
}
```

6.18 2tdm_main.cpp

```
/*
 main.cpp - main file for the gcd
 author: Sandro Renato Dias
 last date: 09/04/2006
 version: 1.0
*/

#include "2tdm_stimulus.h"
#include "2tdm_gcd.h"
#include "2tdm_monitor.h"

int sc_main(int argc, char* argv[]){

  sc_trace_file *tf =
sc_create_vcd_trace_file("2tdm_gcd");

  sc_clock clk;
  sc_signal<int> a, b, r,
  sc_signal<bool> rs;

  stimulus<int> stimulus_i ("stimulus_i");
  stimulus_i.clk(clk);
  stimulus_i.output_a(a);
  stimulus_i.output_b(b);
  stimulus_i.reset(rs);

  gcd<int> gcd_i ("gcd_i");
  gcd_i.input_a(a);
  gcd_i.input_b(b);
  gcd_i.result(r);
  gcd_i.reset(rs);

  monitor<int> monitor_i ("monitor_i");
  monitor_i.input_a(a);
  monitor_i.input_b(b);
  monitor_i.result(r);
  monitor_i.reset(rs);

  sc_trace(tf, a, "a");
  sc_trace(tf, b, "b");
  sc_trace(tf, r, "r");
  sc_trace(tf, rs, "rs");
  sc_trace(tf, clk, "clk");

  sc_start(100, SC_NS);
  sc_close_vcd_trace_file(tf);
  return 0;
}
```

6.19 2tdm_stimulus_file

```
250 50 1
250 50 0
25 5 1
25 5 0
20 5 0
20 5 1
120 10 0
120 10 1
200 120 1
200 120 0
```

6.20 2tdm_Makefile.linux

```
TARGET_ARCH = linux

CC     = g++
OPT    = -O3
DEBUG  = -g
OTHER  = -Wall
CFLAGS = $(OPT) $(OTHER)
# CFLAGS = $(DEBUG) $(OTHER)

MODULE = gcd
SRCS = 2tdm_gcd.cpp 2tdm_stimulus.cpp
2tdm_monitor.cpp 2tdm_main.cpp
OBJS = $(SRCS:.cpp=.o)

include Makefile.defs
```

Anexo F Código do Behavioral Hardware Model

6.21 3bhm_gcd.h

```
/*
 3bhm_gcd.h - gcd in the Behavioral Hard-
ware Model
 author: Sandro Renato Dias
 last date: 22/02/2007
 version: 1.1
*/

#include "systemc.h"

#define READ_LATENCY “10, SC_NS”
#define WRITE_LATENCY “10, SC_NS”
#define COMP_LATENCY “5, SC_NS”


SC_MODULE ( gcd ){

  // Portas
  sc_in_clk clk;   // Clock do sistema
  sc_in<int> input_a, input_b;
  sc_out<int> result;
  sc_in<bool> reset;
  // Protocolo de handshake
  sc_out<bool> send_data; // Requisita lei-
tura
  sc_out<bool> gcd_ready; // Saida pronta


  // Processos
  void prc_gcd();

  // Construtor do modulo
  SC_CTOR( gcd ){
    SC_CTHREAD( prc_gcd, clk.pos() );
    watching(reset.delayed() == true);
  }
};
```

6.22 3bhm_gcd.cpp

```
/*
 3bhm_gcd.cpp - gcd in the Behavioral Hard-
ware Model
 author: Sandro Renato Dias
 last date: 16/11/2006
 version: 1.1
*/

#include "3bhm_gcd.h"

void gcd::prc_gcd() {

   // Variaveis
   int tmp_a, tmp_b, tmp_result;

   // Operacoes de reset
   result.write(0);
   send_data.write(false);
   gcd_ready.write(false);
   wait();

   // Leitura das portas
   // Uso do protocolo de handshake
   send_data.write(true);
   wait();
   // Espera pelos ciclos de leitura
   wait(READ_LATENCY);
   send_data.write(false);
   tmp_a = input_a.read();
   tmp_b = input_b.read();
   wait();

   if ((tmp_a == 0) | (tmp_b == 0)) {
      tmp_result = 0;
   }
   else {
     while( 1 ) {
       tmp_a = tmp_a % tmp_b;
       // Atraso da computacao
       wait (COMP_LATENCY);

       if (tmp_a == 0) {
           tmp_result = tmp_b;
           break;
       }
       tmp_b = tmp_b % tmp_a;
       // Atraso da computacao
       wait (COMP_LATENCY);

       if (tmp_b == 0) {
           tmp_result = tmp_a;
           break;
       } // if
     } // while
   } // else

   gcd_ready.write(true);
   result.write(tmp_result);
   // Atraso da escrita
   wait (WRITE_LATENCY);
   gcd_ready.write(false);
   wait();
```

```
} // gcd
```

6.23 3bhm_stimulus.h

```
/*
 stimulus.h - stimulates the gcd input
 author: Sandro Renato Dias
 last date: 22/02/2007
 version: 2.0
*/

#include "systemc.h"
#include "fstream.h"
#include "iostream.h"

#ifndef READ_LATENCY
#define READ_LATENCY “10, SC_NS”
#endif

SC_MODULE( stimulus ){

  // Arquivos
  ifstream stimulus_file;

  // Portas
  sc_in_clk clk;   // Clock do sistema
  sc_out<int> output_a, output_b;
  sc_out<bool> reset;
  // Protocolo de handshake
  sc_in<bool> send_data; // GCD requisita
leitura

  // Processos
  void prc_stimulus();

  // Cosntrutor do modulo
  SC_CTOR( stimulus ){
    SC_CTHREAD( prc_stimulus, clk.pos() );

       stimulus_file.open
("2tdm_stimulus_file");
       if (!stimulus_file) {
           cerr << "ERROR: Unable to open
vector file stimulus_file!\n";

           sc_stop(); // stop simulation
       }
  }

  // Fecha o arquivo no destrutor
  ~stimulus() {
       stimulus_file.close();
  }

};
```

6.24 3bhm_stimulus.cpp

```
/*
 stimulus.cpp - stimulates the gcd input
 author: Sandro Renato Dias
 last date: 22/02/2007
 version: 3.0
*/

#include "3bhm_stimulus.h"

void stimulus::prc_stimulus() {

   // Variaveis
   int a, b;
   bool rs;

   // Operacoes de reset
   reset.write(false);
   wait();
   reset.write(true);
   wait();
   reset.write(false);
   wait();

   // Read each line
   while (stimulus_file >> a >> b >> rs) {
      // Escrita das portas
      // Uso do protocolo de handshake
      // Receptor inicializa o handshake
      wait_until(send_data.delayed() ==
true);
      wait(READ_LATENCY);
       output_a.write(a);
       output_b.write(b);
       reset.write(rs);
       wait();
       wait(10, SC_NS);
   }

   sc_stop();
}
```

6.25 3bhm_monitor.h

```
/*
 monitor.h - monitoring the gcd output
 author: Sandro Renato Dias
 last date: 22/02/2007
 version: 2.0
*/

#include "systemc.h"

#ifndef WRITE_LATENCY
#define WRITE_LATENCY “10, SC_NS”
#endif

SC_MODULE ( monitor ){
```

```
  // Portas
  sc_in_clk clk;   // Clock do sistema
  sc_in<int> input_a, input_b;
  sc_in<int> result;
  sc_in<bool> reset;
  // Protocolo de handshake
  sc_in<bool> gcd_ready; // GCD saida
pronta

  // Processos
  void prc_monitor();

  // Construtor do modulo
  SC_CTOR( monitor ){
    SC_CTHREAD( prc_monitor, clk.pos() );
  }
};
```

6.26 3bhm_monitor.cpp

```
/*
 monitor.cpp - monitoring the gcd output
 author: Sandro Renato Dias
 last date: 22/02/2007
 version: 2.0
*/

#include "3bhm_monitor.h"

void monitor::prc_monitor() {

   // Variaveis
   int tmp_a, tmp_b, tmp_result;
   bool tmp_reset;

   while (1) {
      // Leitura das portas
      // Uso do protocolo de handshake
      // Transmissor inicializa o handshake
      wait_until(gcd_ready.delayed() ==
true);
      wait(WRITE_LATENCY);
      tmp_a = input_a.read();
      tmp_b = input_b.read();
      tmp_result = result.read();
      tmp_reset = reset.read();

      cout << "\ngcd (" << tmp_a << ", " <<
tmp_b << ") = "
           << tmp_result << ", reset = " <<
tmp_reset << endl;
   }
}
```

6.27 3bhm_main.cpp

```
/*
 main.cpp - main file for the gcd
 author: Sandro Renato Dias
 last date: 22/02/2007
 version: 1.0
*/

#include "3bhm_stimulus.h"
#include "3bhm_gcd.h"
#include "3bhm_monitor.h"

int sc_main(int argc, char* argv[]){

  sc_trace_file *tf =
sc_create_vcd_trace_file("3bhm_gcd");

  sc_clock clk;
  sc_signal<int> a, b, r;
  sc_signal<bool> rs;
  sc_signal<bool> ready, data;

  stimulus stimulus_i ("stimulus_i");
  stimulus_i.clk(clk);
  stimulus_i.output_a(a);
  stimulus_i.output_b(b);
  stimulus_i.reset(rs);
  stimulus_i.send_data(data);

  gcd gcd_i ("gcd_i");
  gcd_i.clk(clk);
  gcd_i.input_a(a);
  gcd_i.input_b(b);
  gcd_i.result(r);
  gcd_i.reset(rs);
  gcd_i.send_data(data);
  gcd_i.gcd_ready(ready);

  monitor monitor_i ("monitor_i");
  monitor_i.clk(clk);
  monitor_i.input_a(a);
  monitor_i.input_b(b);
  monitor_i.result(r);
  monitor_i.reset(rs);
  monitor_i.gcd_ready(ready);

  sc_trace(tf, a, "a");
  sc_trace(tf, b, "b");
  sc_trace(tf, r, "r");
  sc_trace(tf, rs, "rs");
  sc_trace(tf, clk, "clk");
  sc_trace(tf, data, "data");
  sc_trace(tf, ready, "ready");

  sc_start(100, SC_NS);

  sc_close_vcd_trace_file(tf);

  return 0;
}
```

6.28 3bhm_Makefile.linux

```
TARGET_ARCH = linux

CC     = g++
OPT    = -O3
DEBUG  = -g
OTHER  = -Wall
CFLAGS = $(OPT) $(OTHER)
# CFLAGS = $(DEBUG) $(OTHER)

MODULE = gcd
SRCS = 3bhm_gcd.cpp 3bhm_stimulus.cpp
3bhm_monitor.cpp 3bhm_main.cpp
OBJS = $(SRCS:.cpp=.o)

include Makefile.defs
```

Anexo G Código do Register Transfer Level

6.29 4rtl_gcd.h

```
/*
 4rtl_gcd.h - gcd in the Register Transfer
Level
 author: Sandro Renato Dias
 last date: 23/02/2007
 version: 1.1
*/

#include "systemc.h"

#define READ_LATENCY “10, SC_NS”
#define WRITE_LATENCY “10, SC_NS”
#define COMP_LATENCY “5, SC_NS”

enum gcd_state_t {
   s0_reset, s1_read, s2_gcd, s3_write
};

SC_MODULE ( gcd ){

  // Portas
  sc_in<bool> clk;
  sc_in<sc_uint<8> > input_a, input_b;
  sc_out<sc_uint<8> > result;
  sc_in<bool> reset;
  // Protocolo de handshake
  sc_out<bool> send_data; // Requisita lei-
tura
  sc_out<bool> gcd_ready; // Saida pronta

  // Sinais
  sc_signal<sc_uint<8> > tmp_a, tmp_b,
tmp_result;
  sc-signal<bool> tmp_reset;
  sc_signal<gcd_state_t> state, next_state;

  // Processos
  void prc_nextstate();
  void prc_gcd();

  // Construtor do modulo
  SC_CTOR( gcd ){
    SC_METHOD( prc_nextstate );
    sensitive_pos << clk;
    SC_METHOD( prc_gcd );
    sensitive << state << input_a << in-
put_b << reset;
  }
};
```

6.30 4rtl_gcd.cpp

```
/*
 4rtl_gcd.cpp - gcd in the Register Trans-
fer Level
 author: Sandro Renato Dias
 last date: 23/02/2007
 version: 2.0
*/

#include "4rtl_gcd.h"

void gcd::prc_nextstate() {
   if (reset.read() == 1) {
      state = s0_reset;
   } else {
      state = next_state;
   }
}

void gcd::prc_gcd() {
   switch (state) {
      case s0_reset:
         result.write(0);
         send_data.write(false);
         gcd_ready.write(false);
         next_state = s1_read;
         break;
      case s1_read:
         // Leitura das portas
         // Uso do protocolo de handshake
         send_data.write(true);
         wait();
         // Espera pelos ciclos de leitura
         wait(READ_LATENCY);
         send_data.write(false);
         tmp_a = input_a.read();
         tmp_b = input_b.read();
         wait();
         next_state = s2_gcd;
         break;
      case s2_gcd:
         if ((tmp_a == 0) | (tmp_b == 0)) {
            tmp_result = 0;
         }
         else {
            while( 1 ) {
               tmp_a = tmp_a % tmp_b;
               // Atraso da computacao
               wait (COMP_LATENCY);
               if (tmp_a == 0) {
                  temp_result = tmp_b;
                  next_state = s3_write;
```

```
               break;
            }
            tmp_b = tmp_b % tmp_a;
            // Atraso da computacao
            wait (COMP_LATENCY);
            if (tmp_b == 0) {
               tmp_result = tmp_a;
               next_state = s3_write;
               break;
            } // if
         } // while
      } // else
      next_state = s3_write;
      break;
   case s3_write:
      gcd_ready.write(true);
      result.write(tmp_result);
      // Atraso da escrita
      wait (WRITE_LATENCY);
      gcd_ready.write(false);
      wait();
      next_state = s0_reset;
      break;
} // switch
} // gcd
```

6.31 4rtl_stimulus.h

```
/*
 stimulus.h - stimulates the gcd input
 author: Sandro Renato Dias
 last date: 29/11/2006
 version: 1.0
*/

#include "systemc.h"
#include "fstream.h"
#include "iostream.h"

enum stimulus_state_t {
   s0_reset, s1_read, s3_stimulus, s4_write
};

SC_MODULE( stimulus ){

  // Arquivos
  ifstream stimulus_file;

  // Portas
  sc_in<bool> clk;   // Clock do sistema
  sc_out<sc_uint<8> > output_a, output_b;
  sc_out<bool> reset;
  // Protocolo de handshake
  sc_out<bool> send_data; // Requisita
leitura
  sc_out<bool> monitor_ready; // Saida
pronta

  // Sinais
  sc_signal<sc_uint<8> > tmp_a, tmp_b;
  sc_signal<bool> tmp_rs;
  sc_signal<stimulus_state_t> state,
next_state;

  // Processos
  void prc_nextstate();
  void prc_stimulus();

  // Cosntrutor do modulo
  SC_CTOR( stimulus ){
    SC_METHOD( prc_nextstate );
    sensitive_pos << clk;
    SC_METHOD( prc_stimulus );
    sensitive << state;

       stimulus_file.open
("2tdm_stimulus_file");
       if (!stimulus_file) {
           cerr << "ERROR: Unable to open
vector file stimulus_file!\n";

           sc_stop(); // stop simulation
       }
  }

  // Fecha o arquivo no destrutor
  ~stimulus() {
       stimulus_file.close();
  }

};
```

6.32 4rtl_stimulus.cpp

```
/*
 stimulus.cpp - stimulates the gcd input
 author: Sandro Renato Dias
 last date: 29/11/2006
 version: 1.1
*/

#include "4rtl_stimulus.h"

void stimulus::prc_nextstate() {
   state = next_state;
}

void stimulus::prc_stimulus() {
   switch (state) {
      case s0_reset:
         // Operacoes de reset
          output_a.write(0);
          output_b.write(0);
          reset.write(0);
          send_data.write(false);
          monitor_ready.write(false);
          wait();
         next_state = s1_read;
         break;
      case s1_read:
         next_state = s2_stimulus;
         break;
      case s2_stimulus:
         next_state = s3_write;
         break;
      case s3_write:
          while (stimulus_file >> tmp_a >>
tmp_b >> tmp_rs) {
             // Uso do protocolo de
handshake
       monitor_ready.write(true);
              output_a.write(tmp_a);
              output_b.write(tmp_b);
              reset.write(tmp_rs);
             // Atraso da escrita
             wait (WRITE_LATENCY);
```

```
            monitor_ready.write(false);
             wait(10, SC_NS);
          }
         next_state = s0_reset;
         break;
       } // switch

   sc_stop();
}
```

6.33 4rtl_monitor.h

```
/*
 monitor.h - monitoring the gcd output
 author: Sandro Renato Dias
 last date: 29/11/2006
 version: 1.0
*/

#include "systemc.h"

enum monitor_state_t {
   s0_reset, s1_read, s2_monitor, s3_write
};

SC_MODULE ( monitor ){

  // Portas
  sc_in<bool> clk;   // Clock do sistema
  sc_in<sc_uint<8> > input_a, input_b;
  sc_in<sc_uint<8> > result;
  sc_out<bool> reset;
  // Protocolo de handshake
  sc_out<bool> send_data; // Requisita
leitura
  sc_out<bool> monitor_ready; // Saida
pronta

  // Sinais
  sc_signal<sc_uint<8> > tmp_a, tmp_b,
tmp_result;
  sc_signal<bool> tmp_reset;
  sc_signal<monitor_state_t> state,
next_state;

  // Processos
  void prc_nextstate();
  void prc_monitor();

  // Construtor do modulo
  SC_CTOR( monitor ){
    SC_METHOD( prc_nextstate );
    sensitive_pos << clk;
    SC_METHOD( prc_monitor );
    sensitive << state << input_a <<
input_b << result;
  }
};
```

6.34 4rtl_monitor.cpp

```
/*
 monitor.cpp - monitoring the gcd output
 author: Sandro Renato Dias
 last date: 29/11/2006
 version: 1.1
*/

#include "4rtl_monitor.h"

void monitor::prc_nextstate() {
   state = next_state;
}

void monitor::prc_monitor() {
   switch (state) {
      case s0_reset:
         send_data.write(false);
         monitor_ready.write(false);
         wait();
         next_state = s1_read;
         break;
      case s1_read:
         // Leitura das portas
         // Uso do protocolo de handshake
         send_data.write(true);
         wait();
         // Espera pelos ciclos de leitura
         wait(READ_LATENCY);
         send_data.write(false);
           tmp_a = input_a.read();
           tmp_b = input_b.read();
           tmp_result = result.read();
           tmp_reset = reset.read();
         wait();
         next_state = s2_monitor;
         break;
      case s2_monitor:
           cout << "\ngcd (" << tmp_a << ",
" << tmp_b << ") = "
          << tmp_result << ", reset = " <<
tmp_reset << endl;
         next_state = s3_write;
         break;
      case s3_write:
         next_state = s0_reset;
         break;
   } // switch
} // monitor
```

6.35 4rtl_main.cpp

```
/*
 main.cpp - main file for the gcd
 author: Sandro Renato Dias
 last date: 23/02/2007
 version: 1.0
*/

#include "4rtl_stimulus.h"
#include "4rtl_gcd.h"
#include "4rtl_monitor.h"
```

```
int sc_main(int argc, char* argv[]){

  sc_trace_file *tf =
sc_create_vcd_trace_file("4rtl_gcd");

  sc_clock clk;
  sc_signal<sc_uint<8> > a, b, r;
  sc_signal<bool> rs;
  sc_signal<bool> ready, data;

  stimulus stimulus_i ("stimulus_i");
  stimulus_i.clk(clk);
  stimulus_i.output_a(a);
  stimulus_i.output_b(b);
  stimulus_i.reset(rs);
  stimulus_i.send_data(data);

  gcd gcd_i ("gcd_i");
  gcd_i.clk(clk);
  gcd_i.input_a(a);
  gcd_i.input_b(b);
  gcd_i.result(r);
  gcd_i.reset(rs);
  gcd_i.send_data(data);
  gcd_i.gcd_ready(ready);

  monitor monitor_i ("monitor_i");
  monitor_i.clk(clk);
  monitor_i.input_a(a);
  monitor_i.input_b(b);
  monitor_i.result(r);
  monitor_i.reset(rs);
  monitor_i.gcd_ready(ready);

  sc_trace(tf, a, "a");
  sc_trace(tf, b, "b");
  sc_trace(tf, r, "r");
  sc_trace(tf, rs, "rs");
  sc_trace(tf, clk, "clk");
  sc_trace(tf, data, "data");
  sc_trace(tf, ready, "ready");

  sc_start(100, SC_NS);

  sc_close_vcd_trace_file(tf);

  return 0;
}
```

6.36 4rtl_Makefile.linux

```
TARGET_ARCH = linux

CC     = g++
OPT    = -O3
DEBUG  = -g
OTHER  = -Wall
CFLAGS = $(OPT) $(OTHER)
# CFLAGS = $(DEBUG) $(OTHER)

MODULE = gcd
SRCS = 4rtl_gcd.cpp 4rtl_stimulus.cpp
4rtl_monitor.cpp 4rtl_main.cpp
OBJS = $(SRCS:.cpp=.o)

include Makefile.defs
```

Anexo H Código do arquivo xml do 1ESM_GCD.cpp

```
<1ESM__GCD>
 <codigoCpp>
  <includes>
   <nome>
      1ESM_GCD.h
   </nome>
  </includes>
  <processos>
   <processo>
    <tipo__retorno>
       void
    </tipo__retorno>
    <nome__prc>
       prc_GCD
    </nome__prc>
    <parametros />
    <sensitive />
    <portaSaida />
    <variavelEntrada>
     <variavel>
      <tipo>
         void
      </tipo>
      <nome>
         tmp_input_a
      </nome>
     </variavel>
     <variavel>
      <tipo>
         void
      </tipo>
      <nome>
         tmp_input_b
      </nome>
     </variavel>
    </variavelEntrada>
    <variavelSaida>
     <variavel>
      <tipo>
         void
      </tipo>
      <nome>
         tmp_result
      </nome>
     </variavel>
    </variavelSaida>
    <leituraEntrada>
     <item>
      <nome__variavel>
         tmp_input_a
      </nome__variavel>
      <nome__porta>
         input_a
      </nome__porta>
      <metodo>
         read
      </metodo>
     </item>
     <item>
      <nome__variavel>
         tmp_input_b
      </nome__variavel>
      <nome__porta>
         input_b
      </nome__porta>
      <metodo>
         read
      </metodo>
     </item>
    </leituraEntrada>
    <escritaSaida>
     <item>
      <nome__porta>
         result
      </nome__porta>
      <metodo>
         write
      </metodo>
      <nome__variavel>
         tmp_result
      </nome__variavel>
     </item>
    </escritaSaida>
   </processo>
  </processos>
  <nomeArquivo>
     1ESM_GCD
  </nomeArquivo>
  <nivel>
     Executable Specification Model
  </nivel>
  <abrevNivel>
     1ESM
  </abrevNivel>
  <comentarios>
   <author>
      Autor
   </author>
   <lastDate>
      Wed Jan 10 21:01:01 BRST 2007
   </lastDate>
   <version>
      1.0
   </version>
  </comentarios>
 <processos defined-
in="arquivo.ItensModulo" />
 </codigoCpp>
</1ESM__GCD>
```