RAMON RISÉRIO DOURADO LEITE

Implementação de um Módulo Controlador de Vídeo na forma "*Intellectual Property*"

Dissertação apresentada ao curso de Mestrado do Programa de Pós Graduação em Engenharia Elétrica da Universidade Federal de Minas Gerais, como requisito parcial à obtenção do título de Mestre em Engenharia Elétrica.

Belo Horizonte

2006

AGRADECIMENTOS

Este espaço é dedicado àqueles que contribuíram para que este desafio fosse superado. A todos eles, deixo aqui meu sincero agradecimento.

Em primeiro lugar, agradeço ao Professor PhD. Diógenes Cecílio da Silva Jr pela cordialidade com que sempre me recebeu e a presteza com que me orientou.

Em segundo lugar, agradeço ao amigo e colega Rafael Nunes pelo companheirismo e pela ajuda freqüente no uso do Linux, que tornaram esta jornada mais fácil.

Agradeço também a minha irmã Fernanda e a minha prima Ana Karienina por terem me recebido em Belo Horizonte, a minha tia Ione por ter me acolhido num momento difícil, a meus pais e irmãs por terem sido pacientes e respeitado as minhas ausências.

Gostaria ainda de agradecer aos professores, colegas e funcionários do Programa de Pós-Graduação em Engenharia Elétrica e também do Departamento de Ciência da Computação da UFMG por partilhar seus conhecimentos e permitir que esta caminhada fosse possível.

Finalmente, deixo um agradecimento especial para Gilzirene Simone Oliveira pelo constante apoio, amor e compreensão.

RESUMO

Sistemas eletrônicos digitais estão se tornando cada vez mais complexos e o tempo disponível entre a concepção e a comercialização destes dispositivos tem diminuído a cada dia. Estes desafios impõem o uso de novas metodologias de projeto que sejam capazes de lidar com esta complexidade, ao mesmo tempo em que possibilitem aumentar a produtividade dos projetistas de hardware.

Este trabalho propõe uma metodologia de projeto, baseada em SystemC, que usa uma abordagem *top-down* e modela sistemas digitais em três níveis de abstração. Um módulo controlador de vídeo foi desenvolvido utilizando esta metodologia e também uma ferramenta comercial de síntese comportamental que automatiza parte do processo de desenvolvimento. Os resultados deste processo e algumas limitações da metodologia são apresentados.

ABSTRACT

Digital electronic systems are becoming more and more complex, and time to market has been decreasing every day. These challenges impose the use of new project methodologies capable to deal with this complexity and to increase the productivity of hardware designers.

This work proposes a new project methodology, based on SystemC language, which uses a top-down approach and models digital systems in three abstraction levels. A video controller module was developed using this methodology. A commercial behavioral synthesis tool was used to automate part of the development process. The results of this process and some limitations of the methodology are also presented.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

SUMÁRIO

1. INTRODUÇÃO

A quantidade de transistores que podem ser produzidos numa única pastilha de silício tem aumentado ano a ano, comprovando a previsão feita por Moore na década de 60. Isto tem possibilitado constante evolução dos sistemas digitais: transistores e fios eram os componentes fundamentais, que evoluíram para múltiplos chips conectados a uma placa de circuito impresso e, depois, para a fabricação de sistemas inteiros em um único chip [2].

Ao contrário do que se poderia imaginar, este aumento da capacidade de silício não tem elevado o preço dos chips, permitindo que os SoC's (*Systems-on-Chip*) atendam a demanda por dispositivos eletrônicos baratos, confiáveis e sofisticados [2] [5]. Em contrapartida, a concorrência entre fabricantes e a exigência dos consumidores por produtos de qualidade e com novas funcionalidades têm diminuído o tempo disponível entre a concepção e a comercialização destes dispositivos (*time-to-market*) [2] [3] [6].

O projeto de um SoC é complexo, pois além dos componentes de hardware, ele também contém componentes de software, exigindo grande integração entre todos eles. A maior presença de software nos sistemas digitais e o decrescente *time-to-market* trouxeram a necessidade de projeto conjunto e de co-verificação de hardware e software [3] [7].

A convergência de todos estes fatores resultou num grande desafio: ao mesmo tempo em que é preciso lidar com projetos de sistemas digitais cada vez mais complexos, é também necessário fazer com que o tempo de ciclo de projeto seja sempre decrescente. Este desafio exige aumento constante da produtividade dos projetistas, sem que isto prejudique a qualidade dos produtos.

No passado, para resolver este tipo de problema, houve uma transição do projeto de sistemas em nível de layout para o nível de portas lógicas, e depois para o RTL (*Register Transfer Level*). Pelo mesmo motivo os sistemas atuais precisam ser modelados em níveis de abstração ainda maiores [7].

No fluxo tradicional de projeto, o nível de abstração mais alto é representado por um modelo em RTL usando linguagens de descrição de hardware (HDL's) [15] [19]. A maioria destes projetos é escrita neste nível e mapeada para uma *netlist* em nível de portas lógicas pelas ferramentas de síntese [3].

As tecnologias de fabricação de chips atuais permitem a integração de milhões de portas lógicas. Entretanto, metodologias de projeto tradicionais não conseguem lidar com esta quantidade de detalhes de forma produtiva e pequenas adaptações destas metodologias também se mostram inadequadas [4] [8], gerando o *design gap* – a incapacidade de se projetar sistemas que utilizem toda a capacidade disponível de um dado chip que usa o estado da arte. Desta forma, novas metodologias estão sendo usadas para o projeto de sistemas digitais.

As metodologias que usam abordagens *top-down* propõem a síntese de modelos em nível de sistema e o projeto baseado em plataforma. Em contrapartida, as abordagens *bottom-up* iniciam com um conjunto de componentes fornecendo elementos básicos para construir arquiteturas [4]. Ambas usam os conceitos de reutilização de componentes pré-projetados – chamados de núcleos IP (*Intellectual Property*) [3] [6] [9] – e de uso de plataformas [1] [2] [4] [6].

O projeto em nível de sistema oferece uma maneira de obter rapidamente uma especificação executável. Expressar os projetos neste nível tornou-se importante, não só para validar a

funcionalidade dos sistemas, como também para que as otimizações e explorações de arquiteturas possam ser realizadas [6].

A especificação executável normalmente é escrita numa linguagem tal como C ou C++. Depois, geralmente, ela é convertida para RTL numa HDL de forma que seja possível sintetizá-la no nível de portas. Este método pode fazer com que a funcionalidade da descrição RTL seja diferente da especificação executável. Corrigir os erros oriundos destas diferenças é uma tarefa difícil, demorada e propensa a erros [10].

Neste contexto, encontra-se em fase de desenvolvimento a rede *Brazil* IP [23], uma colaboração entre Universidades Brasileiras cuja finalidade é a criação de uma rede distribuída de centros de projetos de CI's capazes de produzir núcleos IP de classe mundial.

Como primeiro passo para alcançar seu intento a *Brazil* IP constituiu o Projeto Fênix. Os principais objetivos deste projeto são estabelecer uma metodologia bem definida para o projeto de IP's e aumentar o conhecimento de estudantes através do uso de ferramentas EDA (Electronic Design Automation) profissionais e técnicas modernas para a especificação, simulação e síntese de núcleos IP.

Como parte do esforço do Projeto Fênix, este trabalho propõe uma metodologia de projeto através de uma abordagem *top-down* e da modelagem de sistemas digitais em três níveis de abstração. A metodologia inicia por uma especificação executável em nível de sistema e usa SystemC como ambiente de desenvolvimento. Um módulo IP com função de controlador de vídeo foi desenvolvido utilizando esta metodologia. Parte dos modelos da metodologia foi gerada através de uma ferramenta de síntese no nível comportamental, o Cynthesizer da Forte Design Systems [27], que automatizou alguns passos do processo de desenvolvimento.

1.1. Motivação

RTL é o nível de abstração mais alto usado no fluxo tradicional de projeto de sistemas digitais. Neste nível, são introduzidos conceitos de *scheduling*, alocação de recursos, *datapath* e otimização. Além disso, estruturas de barramento e ciclos de *clock* precisam ser detalhados para testar restrições lógicas. Desta forma, os subconjuntos RTL sintetizáveis em HDL's, tais como VHDL e Verilog, forçam os projetistas de hardware a especificar o comportamento a cada ciclo de *clock*.

Com a crescente complexidade dos sistemas, as simulações em HDL's têm se tornado muito lentas para a verificação funcional em nível de sistema. Técnicas de emulação de hardware têm sido usadas quando a simulação se torna muito lenta, mas elas freqüentemente limitam a visibilidade do estado para correção de erros e podem ser muito caras.

Em cada espaço de projeto, se apenas a simulação do hardware é considerada, esta simulação, em RTL, é computacionalmente cara e consome muito tempo. A simulação de espaços de projeto, por exemplo, pode facilmente se tornar impraticável.

Este nível de abstração também se mostra inadequado para a verificação dos protocolos de comunicação – pois isto consumiria muito tempo – e para a co-simulação de hardware e software.

Por estas razões, mostra-se a necessidade de uma “especificação executável”. A especificação executável é um modelo de simulação da plataforma em que seus recipientes podem executar usando bancos de teste de entrada (seus próprios e/ou aqueles dados pelo fornecedor da plataforma).

A linguagem C/C++ tem sido usada para descrever especificações executáveis. Os modelos obtidos usando-se C/C++ de forma atemporal para representar as idéias dos níveis mais baixos num nível mais alto de abstração têm se mostrado rápidos. Assim, observa-se o comportamento das várias partes da plataforma para estímulos diferentes realizando explorações da arquitetura do hardware e co-simulações de software.

Depois de corrigir os erros e verificar a funcionalidade do sistema, os projetistas precisam reescrevê-lo em RTL usando Verilog ou VHDL para que possam sintetizá-lo em nível de portas. Este processo de tradução pode fazer com que a descrição em RTL seja diferente da especificação executável e conseqüentemente torná-la propensa a erros. Corrigir erros das diferenças resultantes é um procedimento muito difícil, desafiante, demorado e também propenso a erros.

Nestas abordagens, centradas na simulação, o projetista é responsável por reescrever manualmente o modelo num determinado nível de abstração para ajustar a mudanças no projeto. Em níveis mais baixos, diferentes linguagens são integradas para co-simulação. Em contrapartida, em níveis mais altos, diferentes modelos de computação são combinados num ambiente de simulação comum para especificação. Entretanto, nenhuma destas abordagens ataca a integração vertical de modelos que é necessária para um fluxo de projeto centrado na síntese com refinamento de modelos de nível mais alto para modelos de nível mais baixo.

SystemC é uma alternativa que procura dar suporte a estas necessidades que as HDL’s tradicionais não atendem. Ela é uma biblioteca de classes que estende as capacidades do C++ ao introduzir conceitos tais como concorrência, eventos, portas e sinais, fornecendo uma linguagem única para definir e explorar arquiteturas, projetar componentes de *hardware/software* e facilitar sua co-simulação, e também modelar o projeto em diversos níveis de abstração – desde a especificação executável até o RTL.

É importante ressaltar, que não há motivo em usar SystemC quando se modela apenas em RTL, porque ela não permite explorar facilmente o espaço de projeto neste nível e sua velocidade de simulação é apenas um pouco maior que de uma HDL tradicional.

1.2. Objetivos

Este trabalho tem por finalidade propor e testar uma metodologia de projeto de módulos IP através do refinamento de sua descrição e usando uma abordagem *top-down* que integra verticalmente quatro modelos de projeto em três níveis de abstração.

Após a especificação de requisitos, o fluxo de projeto desta metodologia se inicia por uma especificação executável em nível de sistema descrita em SystemC, chamada de Modelo de Referência.

A seguir, este modelo é refinado definindo-se o comportamento e a funcionalidade externos do sistema. Neste nível o modelo é chamado Comportamental e ainda é descrito em SystemC.

É realizada uma determinação precisa dos pinos de entrada e saída e seu funcionamento externo é definido a partir de uma seqüência de entradas e saídas ao longo do tempo. Mas sua estrutura interna não deve refletir detalhes de implementação.

O Modelo Comportamental é, então, refinado para uma representação em RTL, descrita em SystemC, denominada Modelo RTL.

O último modelo da metodologia, chamado de Modelo HDL, também é representado em RTL. Trata-se de uma tradução da descrição SystemC em RTL para uma HDL a fim de possibilitar sua síntese lógica.

A mesma estrutura de teste é utilizada nos três primeiros modelos, ou seja, nos modelos de Referência, Comportamental e RTL. Este procedimento tem por objetivo minimizar os erros introduzidos pela conversão de um modelo para outro.

Com o objetivo de testar a metodologia e comparar as diferenças entre os modelos, a metodologia foi utilizada para implementar um módulo IP controlador de vídeo.

1.3. Estrutura da Dissertação

Os demais capítulos deste trabalho são estruturados da seguinte forma:

- O capítulo 2 aborda conceitos importantes e técnicas atuais do projeto de sistemas digitais tais como HDL's, linguagens em nível de sistema e ferramentas de síntese.
- O capítulo 3 discute alguns aspectos de metodologias e propõe uma metodologia para o projeto de sistemas digitais baseada, principalmente, em SystemC.
- O capítulo 4 descreve o princípio de funcionamento, os requisitos e os sinais necessários ao correto funcionamento de um controlador de vídeo.
- O capítulo 5 mostra resultados do uso da metodologia na implementação de um módulo IP controlador de vídeo.
- O capítulo 6 apresenta as conclusões e a sugestão de trabalhos futuros.
- O capítulo 7 enumera as referências bibliográficas.

2. PROJETO DE SISTEMAS DIGITAIS

No passado, as comunidades de hardware e de software usaram o aumento do nível de abstração do projeto como técnica básica para gerenciar a complexidade [8]. Para desenvolvedores de software, a transição foi do código *Assembly* para linguagens de nível mais alto como Fortran e PL/I, e então para linguagens mais abstratas como Lisp, Ada e linguagens de paradigmas múltiplos tais como C++.

Para os projetistas de hardware, a mudança se deu a partir de transistores e fios, passando pelo projeto no nível de portas lógicas, depois pelas HDL's em nível RTL, e agora pelas linguagens no nível de sistema.

A seguir são discutidos alguns tópicos importantes relacionados ao projeto de sistemas digitais.

2.1. Reutilização de IP e Plataformas

Assim como no desenvolvimento de software, a complexidade do hardware impõe a reutilização de componentes tanto para lidar com esta complexidade como também para diminuir o tempo de projeto dos sistemas digitais.

A reutilização e a modularização são técnicas complementares que dominaram o projeto RTL. Os projetos baseados em plataforma e em IP são uma evolução destas técnicas em níveis mais altos de abstração. Esta abordagem tem se consolidado como alternativa para a implementação da modularização e da reutilização de componentes [10] [11].

Um núcleo IP é um bloco de hardware para uma aplicação específica que é pré-projetado e pré-verificado. Ele também é conhecido como bloco IP ou módulo IP. Os núcleos IP normalmente estão disponíveis em três formas: *hard*, *firm* e *soft* [1] [9].

- **Núcleos *hard*** são fornecidos como caixas pretas, usualmente na forma de layout. Devido a seu alto desempenho e/ou complexidade de projeto, estes núcleos precisam ser fornecidos com um layout otimizado numa dada tecnologia e com desempenho conhecido.
- **Núcleos *firm*** são fornecidos como uma *netlist* sintetizada, isto é, depois da síntese lógica e do mapeamento da tecnologia, mas sem informação de layout. Estes núcleos não precisam ser sintetizados novamente, mas a *netlist* pode ser simulada e mudada se necessário.
- **Núcleos *soft*** são fornecidos como códigos em HDL e modelados em RTL. O usuário é responsável pela síntese e layout. Normalmente, junto com estes núcleos, os fornecedores IP também fornecem scripts de layout e síntese, e asserções de temporização.

A maioria das aplicações embutidas pode ser implementada usando blocos comuns de arquitetura e personalizadas com componentes específicos da aplicação. Estas arquiteturas comuns e as tecnologias que as suportam (bibliotecas de IP's e ferramentas) podem ser chamadas de plataformas e seus projetos podem ser chamados de projetos baseados em plataforma [1].

Na prática, uma plataforma consiste do seguinte [1]: lista de núcleos IP e suas descrições (em HDL para núcleos *soft* e *firm*, e GDSII para núcleos *hard*), o RTL HDL de alto nível especificando as interconexões com os cores baseados na arquitetura de referência escolhida, *scripts* de síntese para executar síntese lógica dos núcleos IP, asserções de temporização, *drivers* para os componentes periféricos e software para testar o chip completo.

Entretanto, um componente IP não é naturalmente reutilizável. Esforço extra deve ser despendido pelo projetista no desenvolvimento do IP a fim de facilitar a sua reutilização. Um projeto de blocos IP reutilizáveis deve incluir parametrização, versões de componentes (tal como *scripts* de síntese e vetores de teste) e interfaces padrão, dentre outros requisitos [3].

Além disso, para que blocos IP façam parte de um sistema, é preciso fazer a integração destes blocos. Esta tarefa de integração é, em grande parte, um processo manual e propenso a erros porque ele requer que o projetista entenda a funcionalidade de centenas de pinos em vários blocos IP e determine quais pinos devem ser conectados. Os módulos IP parametrizáveis também precisam ser configurados de acordo com seu uso no sistema.

Após a integração dos IP's, o projeto ainda precisa ser simulado, testado e sintetizado.

2.2. Linguagens em nível de sistema e HDL's

Devido à complexidade dos projetos, torna-se necessário descrevê-los em níveis mais altos de abstração a fim de possibilitar simulação mais rápida, co-simulação de *hardware/software* e a exploração de várias arquiteturas. Estas atividades tornam-se impraticáveis ao se modelar em RTL por causa da quantidade de detalhes a serem simulados e do suporte deficiente das HDL's para a co-simulação [6] [8] [16].

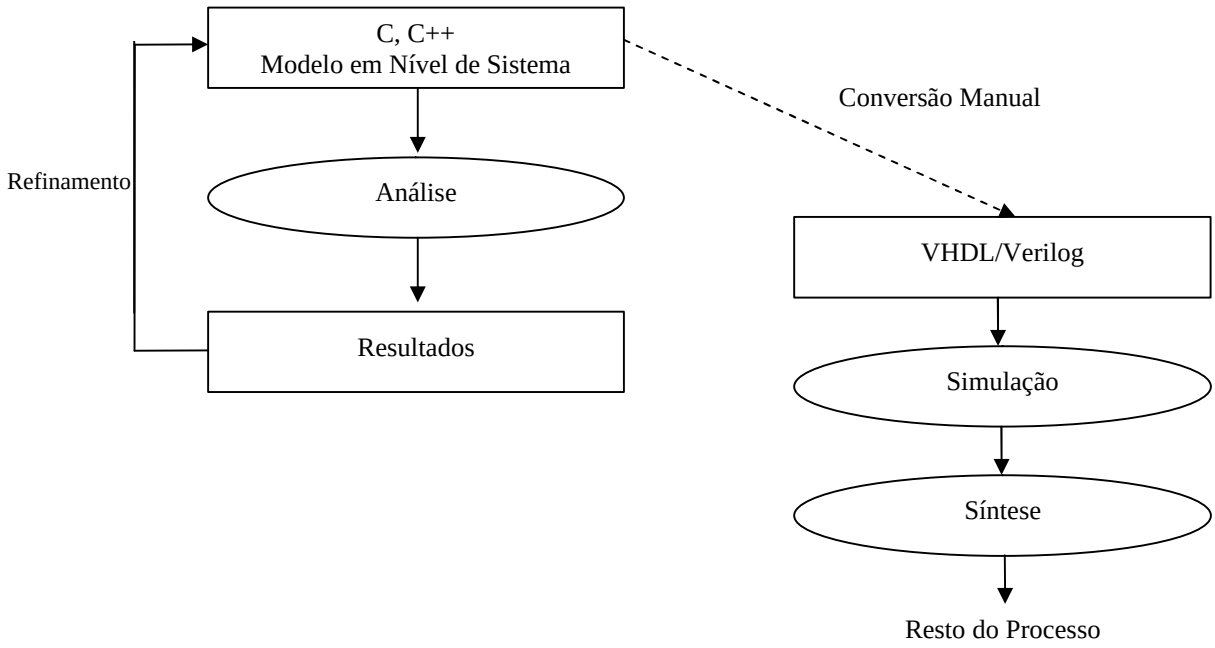


Figura 1- Metodologia de projeto tradicional [1]

[1] Traduzido [6]

Os projetistas resolveram este problema criando especificações executáveis no nível de sistema em C/C++ ou outra linguagem de programação. Depois de validado, este código é traduzido para RTL usando uma HDL. Este refinamento é um pequeno passo conceitual, mas um grande passo de implementação, que é propenso a erros. A figura 1 ilustra este processo de desenvolvimento.

Além disso, os *testbenchs* também precisam ser adaptados, o que também pode introduzir erros no projeto. O termo *testbench* normalmente se refere ao código de simulação usado para criar uma seqüência predeterminada de entradas para um projeto e, opcionalmente, observar a resposta a estes estímulos. Ele modela o comportamento externo ao sistema.

2.2.1. SystemC

SystemC é uma resposta à necessidade de uma linguagem que aumentasse a produtividade global dos projetistas de sistemas eletrônicos digitais. Ela é uma biblioteca de classes que estende as capacidades do C++ ao introduzir conceitos tais como concorrência, eventos, portas e sinais.

SystemC fornece uma linguagem única para definir e explorar arquiteturas, projetar componentes de *hardware/software* e facilitar sua co-simulação, modelar o projeto em diversos níveis de abstração, desde o nível de sistema até o RTL, e também escrever o *testbench* [6] [8]. Ela constitui um ambiente de simulação completo e padronizado que evita diversos erros resultantes da conversão entre linguagens. A figura 2 mostra um processo de desenvolvimento que utiliza SystemC.

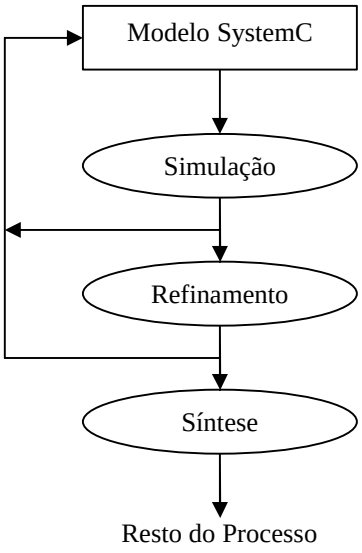


Figura 2- Metodologia de projeto com SystemC [2][6]

A seguir são apresentadas algumas definições conceituais utilizadas por SystemC que possibilitam a simulação de hardware [6]:

[2] Traduzido [6]

- **Processo:** é usado para descrever as funcionalidades do sistema, é sensível a eventos e define comportamento concorrente. Ele não é hierárquico e é definido através das palavras reservadas *SC_METHOD*, *SC_THREAD* ou *SC_CTHREAD*, dependendo da forma como o processo trabalha.
- **Módulo:** é uma entidade hierárquica que pode conter processos ou outros módulos. É definido através da palavra reservada *SC_MODULE*.
- **Portas:** módulos possuem portas que possibilitam a entrada e a saída de dados. Elas podem ser unidirecionais ou bidirecionais. São definidas pelas palavras reservadas *sc_in*, *sc_out* e *sc_inout*.
- **Sinais:** representam "fios" que são usados para conectar os módulos através de suas portas. São definidos através da palavra reservada *sc_signal*.
- ***Clocks*:** são sinais especiais usados para a sincronização do sistema durante a simulação. São definidos pela palavra reservada *sc_clock*.
- **Evento:** representa uma condição que pode ocorrer durante a simulação e é usado para controlar o acionamento de processos. Pode ser um sinal ou uma porta.

Os níveis de abstração que podem ser modelados usando SystemC vão desde o nível de sistema até o RTL, sendo que o mesmo *testbench* pode ser utilizado para a validação do projeto nos diversos níveis. Desta forma, o uso de SystemC diminui consideravelmente os erros que são introduzidos pelo processo de desenvolvimento. Entretanto, da mesma forma que Verilog e VHDL não são boas linguagens para a modelagem e simulação no nível de sistema, SystemC também não é adequado para o projeto em nível de portas lógicas.

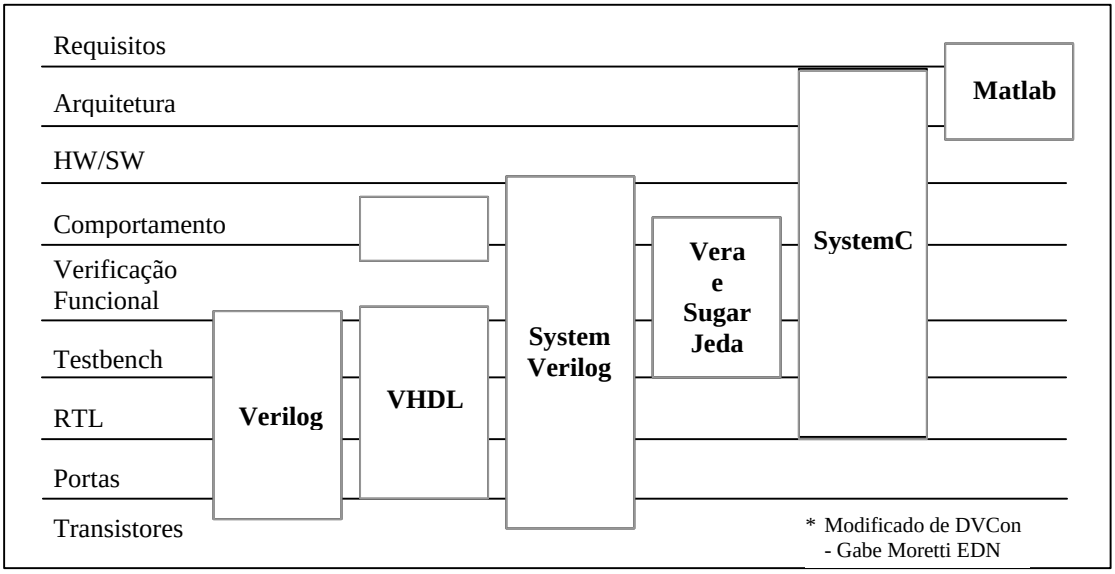


Figura 3 - SystemC comparada com outras linguagens de projeto [3]

A figura 3 realça a aplicação destas e de outras linguagens de projeto. As HDLs VHDL e Verilog são usadas para simulação e síntese de circuitos digitais em nível RTL. Vera e 'e' são

[3] Traduzido [8]

linguagens escolhidas para verificação funcional de ASICs[4] complexos. SystemVerilog é uma nova linguagem que evolui a linguagem Verilog e trabalha com muitas questões de projeto de sistema orientado a hardware. Matlab é usada para capturar os requisitos de sistema e desenvolver algoritmos de processamento de sinal. Cada linguagem ocasionalmente encontra uso fora de seu domínio primário

Para descrever modelos em RTL, SystemC possui um subconjunto denominado SystemC RTL. Um modelo descrito através deste subconjunto pode ser automaticamente convertido numa HDL e sintetizado em portas lógicas [20].

2.3. Síntese de sistemas digitais

No domínio de projeto digital, síntese é um processo automatizado de tradução e otimização onde uma especificação mais abstrata é transformada numa especificação menos abstrata. Por exemplo, a síntese lógica consiste na tradução de um código HDL em RTL em sua equivalente *netlist* de células digitais no nível de portas lógicas. Assim, o sistema é descrito como uma rede de portas e *flip-flops*, e seu comportamento é especificado por equações lógicas.

A síntese de layout traduz uma *netlist* de portas lógicas para um formato que define circuitos lógicos usando componentes eletrônicos, tais como transistores capacitores e fios, e que facilita a localização e o roteamento, fazendo o mapeamento do resultado da síntese lógica em células que constituem a arquitetura destino no domínio físico.

Os aplicativos de desenvolvimento de projetos que fazem esta tarefa de conversão de forma automática são conhecidos como ferramentas de síntese.

Um problema comum é que as implicações de algumas decisões de projeto não podem ser determinadas até que o projeto seja modelado em RTL ou *netlist*. Como realizar explorações nestes níveis é difícil, senão impraticável, surgiu a necessidade de síntese em alto nível.

Entretanto, as ferramentas de síntese tradicionalmente se concentram em questões de baixo nível [3]. O estado da arte destas ferramentas disponibilizam a síntese comportamental, no entanto, a maioria delas ainda não realiza esta síntese de forma satisfatória [14].

A síntese comportamental é o processo de interpretação de um modelo funcional, em que é introduzida a noção de tempo, e a sua decomposição em um modelo RTL. Uma ferramenta de síntese comportamental deve permitir que modelos em altos níveis de abstração sejam automaticamente traduzidos e otimizados para uma implementação em RTL.

O problema de traduzir uma descrição comportamental de alto nível de um sistema digital para uma estrutura em RTL é dividido em algumas sub-tarefas [28]. A especificação do sistema é, primeiro, compilada para uma representação interna, normalmente um gráfico *dataflow*/*controlflow*. Este gráfico é transformado de maneira a tornar o projeto resultante mais eficiente. As operações são, então, programadas em intervalos de tempo e elementos de *hardware* são alocados para os operadores e registradores que são necessários. Estes são interconectados com multiplexadores e o barramento, de acordo com a necessidade, dando a estrutura básica para os *datapahths*. A seguir, módulos específicos são selecionados para

[4] *Application Specific Integrated Circuit* (Circuitos Integrados de Aplicação Específica)

implementar os blocos de hardware abstratos, se isto ainda não foi feito. Finalmente, um controlador é projetado para gerar os sinais de controle necessários para invocar as operações de dados dos passos de controle que foram programados. Completado este projeto RTL, ele está pronto para ser enviado ao *software* de síntese lógica ou ao *software* de projeto físico.

2.4. Forte Design Systems Cynthesizer

Cynthesizer é um software aplicativo desenvolvido pela Forte Design Systems que realiza a síntese comportamental de projetos baseados em SystemC, gerando código Verilog em RTL para ser usado pelas ferramentas de síntese lógica [21].

O Cynthesizer possui várias diretivas que permitem realizar a exploração arquitetural de projetos. Estas diretivas incluem controle de latência, *pipelining*, *array flattening* e *loop unrolling*. Ele também possibilita a otimização do projeto em termos de *throughput*, latência, área e temporização. Algumas destas possibilidades são explicadas a seguir.

- **Controle de latência:** na ausência de restrições, o Cynthesizer otimizará o projeto para ocupar a menor área possível. Entretanto, é possível restringir a latência de conjuntos de operações através do uso de uma diretiva.
- ***Array flattening*:** por *default*, o Cynthesizer mapeia automaticamente vetores para memórias. Como alternativa, é possível usar uma diretiva para que determinados vetores sejam implementados usando registradores individuais para cada posição do vetor. Desta forma é possível acessar simultaneamente várias posições do vetor.
- ***Loop unrolling:*** diferente da síntese RTL tradicional onde os *loops* são sempre completamente paralelizados, o Cynthesizer permite controlar em quais *loops* isto será feito.

A síntese comportamental do Cynthesizer envolve vários passos distintos de processamento conforme descritos a seguir. A figura 4 ilustra estes passos.

- **Compilação:** simplesmente assegura que nenhum erro de sintaxe ou outros erros existam no código fonte do projeto antes que a síntese possa começar.
- **Análise de fluxo de dados/*datapath*:** determina os operadores, os multiplexadores e a lógica de controle, independentes da tecnologia, necessários para implementar o sistema.
- **Alocação:** cada operador independente de tecnologia identificado no passo de análise é alocado a um operador existente na biblioteca do processo alvo. Esta fase também extrai a informação de temporização para cada operador.
- ***Scheduling*:** cada operador alocado no projeto é designado a um ciclo de *clock* específico baseado no fluxo de dados do projeto e no atraso do operador. As restrições de latência/*throughput* ajudam neste processo. O paralelismo existente (inerente) é explorado e se possível ou necessário é introduzido durante esta fase.
- **Mapeamento e saída:** todos os operadores alocados (e os registradores para armazenar os dados) são especificamente mapeados para um componente único (*binding*) e o projeto resultante é gerado como um projeto em RTL descrito em Verilog, pronto para a síntese lógica.

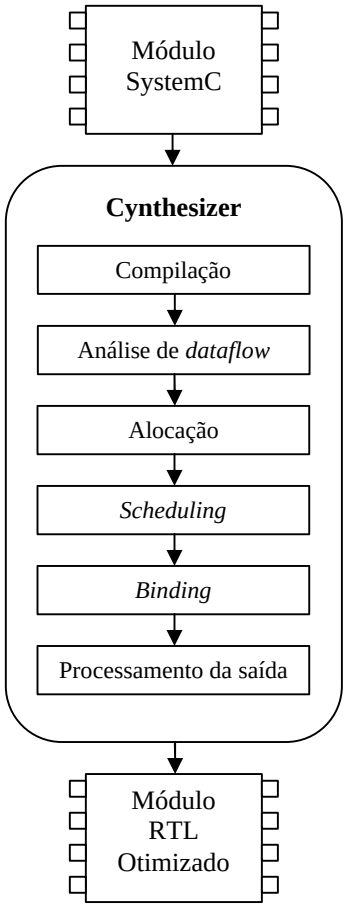


Figura 4- Etapas da síntese comportamental do Cynthesizer [5]

2.5. Altera Quartus II

O software de desenvolvimento Quartus II é a ferramenta da Altera Corporation [29] para a síntese lógica de projetos digitais, que suporta toda a sua linha de dispositivos lógicos programáveis (FPGA's), e que também permite o projeto de ASIC's.

O Quartus II integra diversas funções, tais como:

- Entrada e edição de projetos através de diagramas de bloco, AHDL, VHDL e Verilog HDL.
- Combinação de diferentes tipos de arquivos de projeto.
- Simulação funcional e temporizada.
- Localização automática de erros.
- Funções de localização e roteamento, verificação e programação de dispositivos.
- Análise e cálculo dos tempos de propagação.
- Capacidade de otimização automática do consumo de energia do projeto.
- Criação e otimização de blocos de projeto de forma independente.

[5] Traduzido [21]

Um fluxo de projeto típico usando o Quartus II é ilustrado na figura 5. Este fluxo de projeto envolve os seguintes passos básicos:

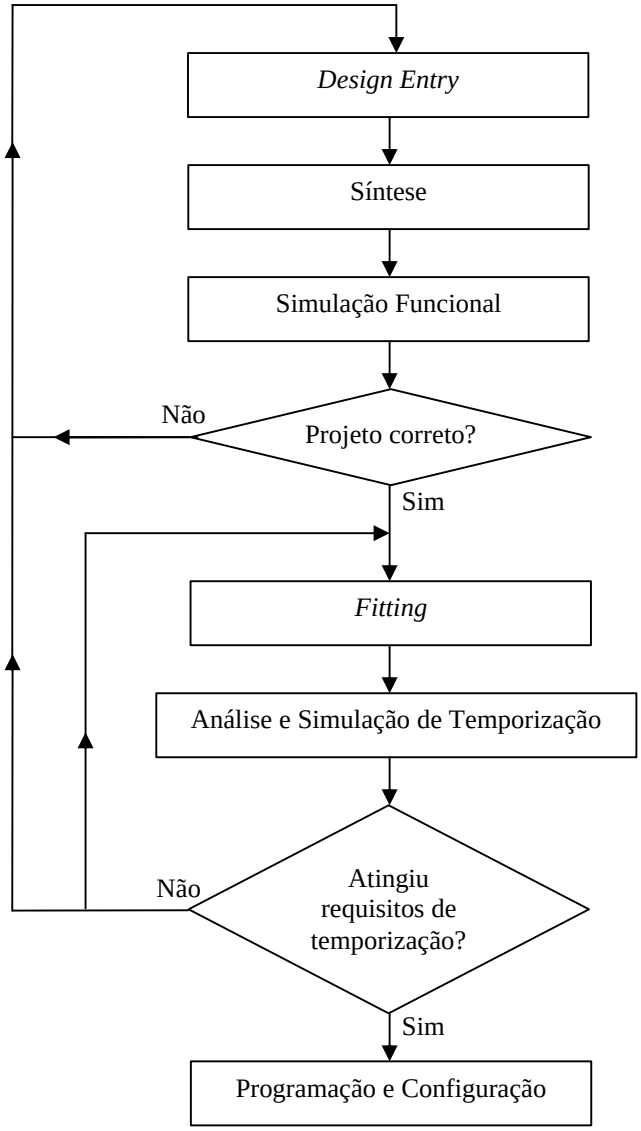


Figura 5 - Fluxo de Projeto Típico usando o Quartus II [6]

- ***Design Entry*:** o circuito desejado é especificado usando uma HDL, tal como Verilog ou VHDL, ou por meio de um diagrama esquemático.
- **Síntese:** a ferramenta de síntese sintetiza o circuito para uma *netlist* que define os elementos lógicos (LE's) necessários e as conexões entre estes elementos.
- **Simulação Funcional:** o circuito sintetizado é testado para verificar sua funcionalidade; esta simulação não considera nenhuma temporização.
- ***Fitting*:** a ferramenta determina a localização dos LE's, definidos na *netlist*, no chip FPGA real; ela também escolhe o roteamento no chip para fazer as conexões requeridas entre os LE's.

[6] Traduzido [25]

- **Análise de Temporização:** atrasos de propagação ao longo do diversos caminhos são analisados para fornecer uma indicação do desempenho esperado do circuito.
- **Simulação de temporização:** o circuito é testado para verificar tanto sua funcionalidade quanto a temporização.
- **Programação e Configuração:** o circuito projetado é implementado num chip FPGA físico através da programação das chaves de configuração que configuram os LE's e estabelecem as conexões requeridas.

3. METODOLOGIA

A distância entre os requisitos de um sistema e sua implementação geralmente não pode ser vencida num único passo. É necessário dividir o processo de desenvolvimento numa seqüência de passos menores, gerenciáveis e sem grandes descontinuidades.

As metodologias de projeto tradicionais partem de modelos muito detalhados, fazendo com que os projetistas tendam a cometer muitos erros na descrição destes modelos, causando retrabalho, aumentando o tempo de simulação e de depuração. Assim, devido a sua complexidade, o projeto de sistemas eletrônicos digitais tem imposto a utilização de modelos mais abstratos.

Existem várias abordagens que lidam com a classificação e a estrutura do processo de desenvolvimento de projeto [17] [18]. No entanto, elas não definem um fluxo real de projeto.

Um fluxo de projeto precisa determinar seus níveis de abstração com modelos para cada ponto do fluxo. A quantidade de modelos e as propriedades de cada um devem ser definidas para ajudar nas transições, fornecendo somente os detalhes suficientes e estabelecendo um compromisso entre precisão e eficiência, por exemplo, em termos de velocidade de simulação.

A metodologia de projeto proposta usa uma abordagem *top-down* e a modelagem dos sistemas digitais em três níveis de abstração. A metodologia inicia por uma especificação executável em nível de sistema e usa SystemC como ambiente de desenvolvimento.

À medida que os modelos de níveis mais altos são testados e aprovados, detalha-se cada vez mais o projeto, passando pelo nível comportamental, até o RTL, que possui maior precisão de simulação. Desta forma, o processo de desenvolvimento de hardware se torna um processo de refinamento da especificação do sistema.

O primeiro modelo de projeto em RTL da metodologia é descrito em SystemC. Um segundo modelo é escrito numa linguagem de descrição de hardware, tal como Verilog HDL e VHDL. Os modelos são melhor detalhados mais adiante.

A metodologia proposta preconiza que o *testbench* (em SystemC) seja reutilizado para assegurar que refinamentos do modelo não introduzam erros. O uso de *testbenchs* diferentes para cada nível de abstração pode introduzir erros no projeto e também tornar difícil conduzir experimentos e comparações confiáveis. Somente o modelo da metodologia escrito em HDL não utiliza este *testbench*.

É importante ressaltar que o *testbench* é o modelo do comportamento externo ao sistema, ou seja, ele também pode incluir o software que será executado no hardware que está sendo projetado. SystemC permite esta co-simulação.

A metodologia proposta prevê quatro modelos de sistema: de referência, comportamental e RTL que devem ser descritos em SystemC; e o Modelo HDL, que pode ser codificado em Verilog ou VHDL. O início da modelagem deve suceder a especificação de requisitos e o planejamento da verificação funcional.

SystemC foi escolhida como linguagem de modelagem e simulação dos três modelos iniciais da metodologia devido a diversas características e benefícios que foram evidenciados na seção 2.2.1.

VHDL e Verilog foram escolhidas por serem HDL's consolidadas e padronizadas para a modelagem e simulação de hardware em RTL. Outro motivo é que, por isso mesmo, são freqüentemente usadas como entrada por ferramentas de síntese lógica.

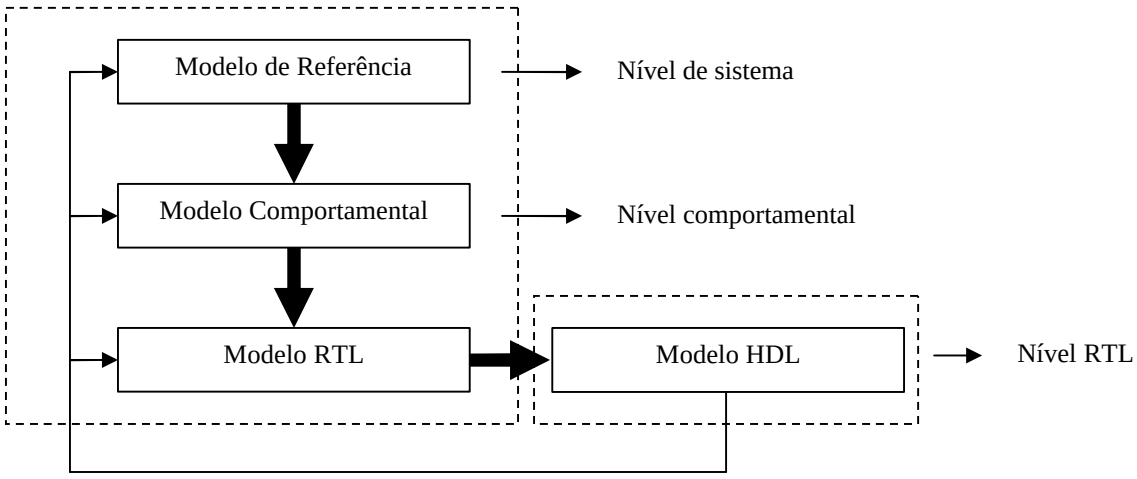


Figura 6 - Modelos da metodologia proposta

A metodologia proposta é ilustrada através da figura 6. A seguir são apresentados estes modelos.

3.1. Modelo de Referência

O Modelo de Referência, ou especificação executável, é o primeiro modelo conceitual da metodologia. É uma abstração de alto nível que visa diminuir a complexidade de forma a definir e testar os aspectos funcionais do sistema. Ele é uma tradução direta da especificação de projeto para SystemC que modela a funcionalidade do sistema de uma forma completamente independente de qualquer implementação.

Este modelo não é sintetizável, mas permite explorar algoritmos, evitar inconsistências e erros, garantir a completude e assegurar uma interpretação não ambígua dos requisitos do sistema. Além disso, possibilita a validação do desempenho do sistema através da criação de modelos de desempenho.

O *testbench*, criado para validar o Modelo de Referência, também será usado pelos modelos Comportamental e RTL. Ele representa a fonte de estímulos e também captura a resposta do sistema a estes estímulos, ou seja, simula o ambiente externo ao sistema.

A execução do Modelo de Referência produz dados em suas portas de saída. Estes dados são capturados pelo *testbench*, que cria arquivos de saída. O *testbench* também produz os arquivos de saída para os demais modelos. Assim, os arquivos de saída do Modelo de Referência são comparados aos arquivos dos modelos subseqüentes para verificar se a funcionalidade do sistema permanece a mesma.

O arquivo usado pelo *testbench* para simular os estímulos também é reutilizado. Tudo isto é feito para que a funcionalidade do sistema permaneça a mesma ao longo dos diversos modelos. A figura 7 ilustra esta utilização do *testbench* pelos três primeiros modelos da metodologia.

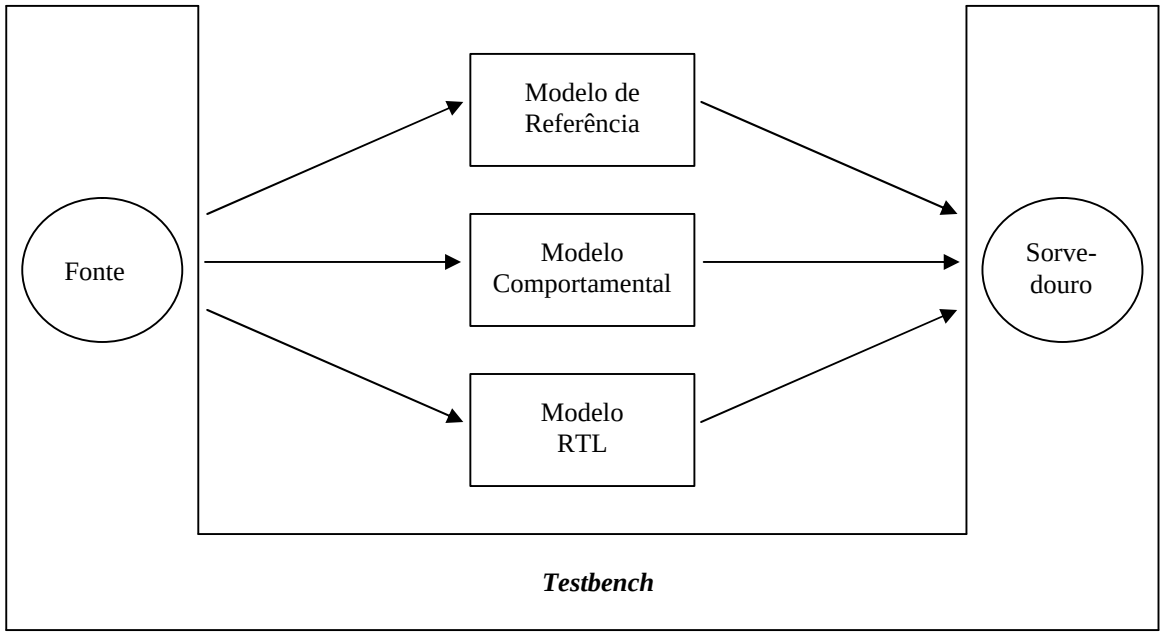


Figura 7 - Reutilização de *testbench*

3.2. Modelo Comportamental

Após sua verificação e validação, o Modelo de Referência é refinado para um modelo em nível de sistema em que a noção de tempo é introduzida. Este modelo, que também é descrito em SystemC, permite pensar o projeto como fluxo de programa. Seu comportamento externo é definido por uma seqüência de entradas e saídas ao longo do tempo.

No entanto, nesta representação, a estrutura interna não deve impor detalhes de implementação, ou seja, o mapeamento das operações internas para ciclos de *clock* e unidades funcionais é irrelevante neste nível. Seus limites e estrutura externos, por outro lado, devem ter funcionalidade acurada e precisão de pino.

A comunicação entre os módulos do sistema pode ser descrita através de transações. Se este é o caso, esta comunicação pode ser feita usando chamadas de função como em linguagens de programação.

Uma vez obtido o modelo, várias arquiteturas de hardware e software podem ser exploradas.

O Modelo Comportamental pode ser usado como entrada para as ferramentas de síntese comportamental [21] [22]. A utilização destas ferramentas elimina a necessidade de codificar manualmente os modelos RTL e HDL. Isso pode trazer uma redução significativa no tempo de projeto, além de possibilitar um melhor resultado final.

O Modelo Comportamental é verificado usando os mesmos *testbench* e estímulos do Modelo de Referência. Os arquivos gerados pelo *testbench* ao simular os modelos são comparados para validar o Modelo Comportamental. Nesta representação do sistema, as saídas refletem os atrasos devido às restrições introduzidas e à tecnologia adotada pelo modelo.
Após a validação, se uma ferramenta de síntese não está disponível, o Modelo Comportamental é decomposto manualmente no Modelo RTL.

3.3. Modelo RTL

O Modelo RTL se refere ao nível de abstração em que um circuito é descrito como transferências síncronas entre unidades funcionais. Este modelo compartilha uma característica comum com o Modelo Comportamental: os pinos e os limites estruturais já estão delineados; além de também ser descrito em SystemC.

O que distingue o Modelo RTL de modelos em níveis mais altos de abstração é sua precisão em ciclo de *clock* e sua descrição detalhada da lógica combinacional e da transferência entre registradores.

Em RTL, o projetista deve decidir o que os estados do circuito são, como são as transições do circuito de um estado a outro e que operações são realizadas em cada estado, ou seja, deve definir a máquina de estados finitos do sistema.

O *testbench* e o procedimento de verificação são os mesmos usados pelo Modelo Comportamental.

Existem ferramentas que utilizam o código SystemC em RTL para convertê-lo automaticamente numa HDL e depois proceder a síntese lógica [20]. Estas ferramentas evitam os erros e o tempo gasto em depuração que são conseqüência da conversão manual.

3.4. Modelo HDL

Este modelo é constituído pela tradução direta do Modelo RTL, descrito em SystemC, para uma HDL ainda em nível RTL. Esta tradução é necessária, pois existem várias ferramentas de síntese confiáveis que realizam a síntese lógica a partir de um Modelo RTL escrito em VHDL ou Verilog.

A partir do Modelo HDL pode-se escolher dois caminhos não excludentes para a implementação: sintetizar o sistema numa FPGA (*Field Programmable Gate Array*) para teste e depois uso, ou implementá-lo em um chip dedicado (ASIC).

3.5. Exemplo Didático – Somador

Com a finalidade de um melhor entendimento dos modelos definidos na metodologia apresenta-se a seguir trechos dos códigos para o projeto de um somador de 8 bits. Também são mostradas representações gráficas para os modelos.

Este somador deve adicionar dois números inteiros e informar, além do resultado da soma, a ocorrência de *overflow*.

Modelo de Referência

A figura 8 procura apresentar graficamente as características do modelo de referência do somador. Pode-se perceber na figura, a simplicidade do modelo.

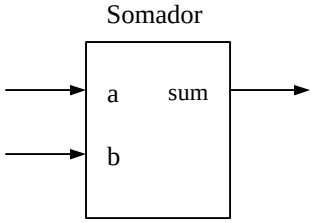


Figura 8 – Diagrama de bloco do Somador – modelo de referência

O trecho de código mostrado na figura 9 foi feito de acordo com as características do Modelo de Referência e utilizando a linguagem SystemC. Apesar de baseado em SystemC o código não usa os recursos especializados da linguagem, se limitando ao aspecto funcional do somador – a adição de dois números –, ou seja, sem se preocupar com detalhes de implementação.

```
int adder(int a, int b) {
        sum = a + b;
        return(sum);
}

void adder_driver(int& a, int& b) {
        while (infile >> a >> b);
}

void adder_ monitor (int a, int b, int sum) {
        cout << a << " + " << b << " = " << sum << endl;
}

int sc_main( ) {
        int a, b, sum;
        adder_driver(a, b);
        sum = adder(a, b);
        adder_monitor(a, b, sum);
        return (0);
}
```

Figura 9 – Trecho de código do somador – Modelo de Referência

O tempo de uso da CPU para a execução do código completo do Modelo de Referência para a realização de 255 somas foi de aproximadamente 0,05 s.

Modelo Comportamental

O trecho de código apresentado na figura 11 foi feito de acordo com as características do Modelo Comportamental e também utiliza a linguagem SystemC. Neste passo da metodologia são utilizadas características específicas da linguagem, tal como a concorrência e temporização. A figura 10 procura ilustrar graficamente estas características.

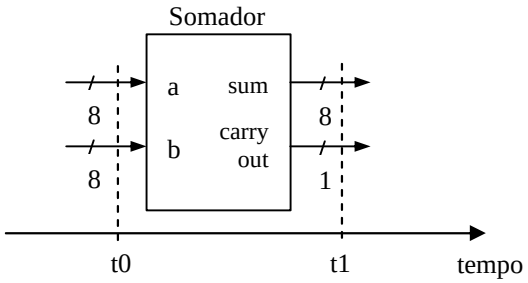


Figura 10 – Diagrama de bloco do Somador – modelo comportamental'

O trecho de código da figura 11 identifica um processo (process_adder) que está permanentemente sendo executado, ou seja, ele simula o funcionamento de um circuito real. Outros processos poderiam estar sendo executados em paralelo.

```
SC_MODULE(ADDER) {
// public:
  sc_in< sc_uint<8> > a_in, b_in; // Dados de entrada
  sc_out< sc_uint<8> > sum_out;  // Resultado
  sc_out< bool > carry_out;          // Overflow
  …
  SC_CTHREAD(process_adder);
  …
}

void ADDER:: process_adder ( ) {
  while(true) {
       a = a_in.read();        // Leitura de dado
       b = b_in.read();        // Leitura de dado
        wait(5, SC_NS);        // Simulação de atraso

       sum = a + b;            // Operação do somador
       carry = sum[8];         // Overflow

       sum_out.write(sum)              // Escrita de dado
       carry_out.write(carry);         // Escrita de dado
        wait(5, SC_NS);                // Simulação de atraso
  }
```

Figura 11 – Trecho de código do somador – Modelo Comportamental

Neste modelo os tempos são definidos através de estimativas ou impostos por requisitos de projeto.

O tempo de uso da CPU para a execução do código completo do Modelo Comportamental para a realização de 255 somas foi de aproximadamente 0,08 s, ou seja, 60% maior que o tempo de execução do modelo de referência.

Modelo RTL

A figura 12 mostra o modelo RTL e procura representar graficamente suas características.

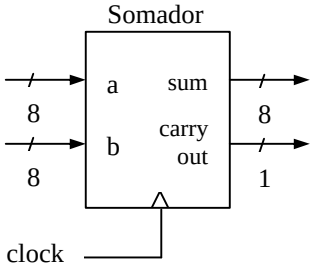


Figura 12 – Diagrama de bloco do Somador – Modelo RTL

O trecho de código mostrado na figura 13 apresenta algumas características do modelo RTL, ou seja, o código representa um processo concorrente do somador de 8 bits e, como ele precisa ser sintetizável, as características de temporização não podem ser simuladas dentro do processo. A temporização passa a ser simulada através do clock, que é definido em outra parte do código. A freqüência do clock é estabelecida através de estimativa ou imposta por requisitos de projeto.

```
SC_MODULE(ADDER) {
// public:
  sc_in< sc_uint<8> > a_in, b_in; // Dados de entrada
  sc_in_clk clock;                // Clock
  sc_in< bool > rst;              // Sinal de reset
  sc_out< sc_uint<8> > sum_out;   // Resultado
  sc_out< bool > carry_out;       // Overflow

  sc_uint <8> a, b;               // Variáveis internas
  sc_uint <9> sum;
  …
  SC_METHOD(process_adder);       // Processo somador
  sensitive << clock;             // Sensível ao clock
  …
}

void ADDER:: process_adder ( ) {
      a = a_in.read();        // Leitura de dado
      b = b_in.read();        // Leitura de dado

      sum = a + b;            // Operação do somador
      carry = sum[8];         // Overflow

      sum_out.write(sum)              // Escrita de dado
      carry_out.write(carry);         // Escrita de dado
  }
```

Figura 13 – Trecho de código do somador – Modelo RTL

Pode-se observar que este modelo aproveita os pinos e limites estruturais delineados no Modelo Comportamental.

O tempo de uso da CPU para a execução do código completo do Modelo RTL para a realização de 255 somas foi de aproximadamente 0,12 s, ou seja, 50% maior que o tempo de execução do modelo comportamental e 140% maior que o do modelo de referência.

O código deste modelo pode ser refinado, utilizando-se uma linguagem de descrição de hardware (tais como Verilog HDL e VHDL) que permita a síntese do circuito do somador numa FPGA. Percebe-se no código acima a preocupação com detalhes de implementação, tais como tamanho dos operandos (8 bits) e possibilidade de overflow (carry_out).

Modelo HDL

A figura 12 também representa o Modelo HDL, pois ele é uma tradução direta do Modelo RTL para uma HDL, permanecendo em nível RTL.

O trecho de código mostrada na figura 14 representa parte do Modelo HDL descrito em Verilog.

```
module dut(a_in, b_in, clock, rst, sum_out, carry_out);

    input [7:0] a_in;              // Dado de entrada
    input [7:0] b_in;              // Dado de entrada
    input clock;                   // Clock
    input rst;                     // Sinal de reset
    output [7:0] sum_out;          // Resultado
    output carry_out;              // Overflow

    reg [8:0] a;                   // Variáveis internas
    reg [8:0] b;
    reg [8:0] sum;
    reg carry;
    …

      always @(posedge(clock))     // Processo sensível ao clock
       begin :thread1
        case (global_state)

          2'd1: begin
            carry_out <= Add_FCLA9_1_out1[8];
          end

        endcase

      end
    …
endmodule
```

Figura 14 – Trecho de código do somador – Modelo HDL

A Tabela 1 apresenta um sumário dos tempo de execução de 255 somas do código para os níveis apresentados e o aumento do tempo de execução adicional (*overhead*) comparado aos níveis superiores.

A cada nível de refinamento, à medida que detalhes são agregados à descrição, o tempo de execução aumenta a uma taxa de aproximadamente 50% (para este exemplo em particular). Isso aponta para uma tendência que se repete em outros projetos de acordo com a literatura.

Tabela 1: Tempos de execução e overhead entre os níveis de descrição

Nível	**Execução (s)**	***Overhead* (p/ Mod. Ref)**	***Overhead* (p/ Mod. Comp.)**
Modelo de Referência	0,05		
Modelo Comportamental	0,08	60%	
Modelo RTL	0,12	140%	50%

4. CONTROLADOR DE VÍDEO

O objetivo desta seção é discutir as características de um controlador de vídeo para *raster scan displays* sem processamento gráfico e os conceitos necessários para implementá-lo.

Um controlador de vídeo é um dispositivo periférico em um sistema de computação que é usado para produzir a saída visual. Ele é o principal hardware que trabalha entre o processador e o dispositivo de exibição (monitor de vídeo), transmitindo a informação recebida dos programas e aplicações que estão sendo executadas no computador, através do processador, para o monitor, que permite visualizar a informação e as imagens na tela [30] [32].

Raster scan é um padrão de esquadrinhamento usado para criar ou capturar imagens, que percorre as linhas horizontais da imagem da esquerda para a direita e de cima para baixo [31].

4.1. Princípio de Funcionamento

Os dados enviados pelo processador para serem visualizados devem ser armazenados em um *frame buffer*, que é a parte da memória de vídeo usada para guardar a imagem a ser visualizada na tela. Estes dados correspondem aos *pixels* (*picture elements*) da imagem.

Um *pixel* representa um ponto de uma imagem e corresponde a um ponto da tela do monitor de vídeo. O conjunto de *pixels* é arrumado no formato de uma matriz para representar a imagem. Para uma tela monocromática, cada *pixel* é representado por um bit. Para uma tela colorida, cada *pixel* possui a informação que determina sua cor. O sistema de cores mais utilizado atualmente é o RGB.

RGB é a abreviatura do sistema de cores aditivas formado pelas cores primárias Vermelho (*Red*), Azul (*Blue*) e Verde (*Green*) misturadas para produzir a cor desejada. Uma cor no formato RGB é representada por três valores que indicam as intensidades dessas cores primárias. O valor zero para uma cor primária significa que ela não participa da mistura e o valor máximo significa que ela participa com intensidade máxima. Assim, se todas os valores forem zero (ausência de cor), temos a cor preta e se todas participam com intensidade máxima, temos a cor branca. O valor máximo é definido pela quantidade de bits que representa cada cor. Para uma cor representada por n bits, o valor máximo é dado pelo resultado de '$2^n - 1$'. Esta quantidade influencia a quantidade de cores que podem aparecer na tela e o tamanho do *frame buffer*.

Tabela 2 - *Color depth* x número de cores

Color Depth	**Número de cores**	**Nome comum para a Color Depth**
4-Bit	16	Standard VGA
8-Bit	256	256-Color Mode
16-Bit	65,536	High Color
24-Bit	16,777,216	*True color*

O tamanho do *frame buffer* depende da resolução da tela (definida pelo padrão gráfico utilizado) e também da quantidade de bits de cor usada por *pixel* (*color depth*). A *color depth* é constituída pelos bits usados para representar cada cor primária e determina a quantidade de cores que podem aparecer na tela. A Tabela 2 mostra as *color depth*s mais usadas pelos computadores pessoais (PC's) na atualidade.

Como mostrado na Tabela 2, no modo de 256 cores o PC tem apenas 8 bits para usar. Isto significaria, por exemplo, usar 2 bits para azul (B), 3 bits para verde (G) e 3 para vermelho (R). Escolher entre apenas 4 ou 8 diferentes valores para cada componente da cor do *pixel* produziria um resultado muito ruim. Assim, uma abordagem diferente é aplicada: o uso de uma paleta de cores, técnica também conhecida como *Color Lookup Table* (CLUT).

A fórmula para calcular o tamanho do *frame buffer* é muito simples:

Memória = hres x vres x *color_depth*

onde Memória corresponde ao tamanho do *frame buffer* em bits, *color_depth* é o número de bits de cor para representar cada *pixel*, hres e vres correspondem respectivamente às quantidades de *pixels* por linha (resolução horizontal) e a quantidade de linhas da tela (resolução vertical). A tabela 3 mostra a quantidade de *frame buffer* necessário em relação a várias resoluções e *color depths*.

Uma paleta é criada contendo 256 cores diferentes. Cada uma é definida usando o padrão de definição de cor de 3 bytes que é usada no modo *true color*: 256 possíveis intensidades para cada vermelho (R), verde (G) e azul (B). Então, para cada *pixel* é permitido escolher uma das 256 cores na paleta, que pode ser considerada um "número de cor". Assim a faixa total de cores pode ser usada nas imagens, mas cada imagem pode apenas usar 256 das mais de 16 milhões de diferentes cores possíveis. Quando cada *pixel* é mostrado, a placa de vídeo consulta os valores reais de vermelho, verde e azul na paleta baseado no "número da cor" que foi designado para o *pixel*.

A paleta de cores é um excelente compromisso: por exemplo, ela permite usar apenas 8 bits para especificar cada cor em uma imagem, mas permite ao criador da imagem decidir qual seriam as 256 cores da imagem. Desde que virtualmente nenhuma imagem contém uma distribuição de cores regular, isto permite mais precisão numa imagem pelo uso de mais cores que seria possível designando-se para cada *pixel* um valor de 2 bits para azul, um valor de 3 bits para verde e outro valor de 3 bits para vermelho.

A função do controlador de vídeo é gerar sinais adequados ao monitor de vídeo. Estes sinais são baseados nas informações contidas no *frame buffer* e no tipo de monitor.

O *frame buffer* é responsável por armazenar a imagem que será mostrada no monitor de vídeo. Assim, ele é acessado no momento em que se grava os dados da imagem e também no instante em que estes dados serão processados para serem enviados ao monitor. Técnicas são utilizadas para se evitar os problemas que podem ser causados por conflitos de acesso ao *frame buffer*. Entre elas, destaca-se o uso de *frame buffer dual-port*, o *frame buffer* duplo e multiplexação temporal do *frame buffer*.

O *frame buffer dual-port* é uma memória que possui uma porta para escrita e outra para leitura que podem ser utilizadas ao mesmo tempo, diminuindo drasticamente o conflito de acesso. A grande desvantagem é o alto preço deste tipo de memória.

Tabela 3 - Padrão gráfico x *color depth* x memória

Padrão Gráfico	**Resolução da tela (*pixels* x *pixels*)**	***Color Depth* (bits)**	**Memória (KB)**
VGA (*Video Graphics Array*)	640 x 480	8	300
		16	600
		24	900
SVGA (Super VGA)	800 x 600	8	469
		16	938
		24	1.406
XVGA (*eXtended* VGA)	1024 x 758	8	758
		16	1.516
		24	2.274
CIF (*Common Intermediate Format*)	352 x 240 (NTSC)	8	83
		16	165
		24	248
	352 x 288 (PAL)	8	99
		16	198
		24	297
QCIF (*Quarter* CIF)	176 x 120 (NTSC)	8	21
		16	41
		24	62
	176 x 144 (PAL)	8	25
		16	50
		24	74
4CIF	704 x 480 (NTSC)	8	330
		16	660
		24	990
	704 x 576 (PAL)	8	396
		16	792
		24	1.188

O *frame buffer* duplo ou *double buffering* é uma técnica que utiliza duas memórias. Uma memória é utilizada para gravar os dados e a outra é usada para enviar os dados ao monitor, permitindo que as duas memórias sejam acessadas ao mesmo tempo.

A multiplexação temporal do *frame buffer* é realizada utilizando-se os intervalos de tempo em que o monitor não está necessitando de dados, tais como os intervalos de retraço horizontal e vertical dos monitores CRT, ou o tempo de resposta dos monitores LCD.

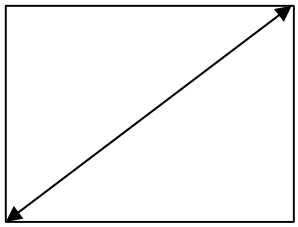


Figura 15 - Diagonal que define o tamanho da tela do monitor de vídeo

Quanto aos monitores, os fatores que os definem são basicamente o tamanho da tela, o *dot pitch* (distância entre os *pixels* que compõe a tela), as resoluções suportadas e a taxa máxima de atualização da imagem (*refresh rate*).

O tamanho da tela do monitor é dado pela sua diagonal (figura 15). Os tamanhos de tela de monitor usuais na atualidade são 14”, 15”, 17”, 19”, 20” e 21”.

O tamanho da tela e o *dot pitch* determinam a resolução máxima da tela. Quanto menor o *dot pitch* mais nítida será a imagem da tela. O *dot pitch* de computadores pessoais geralmente variam entre 0,15 mm e 0,30 mm.

Refresh rate, taxa de atualização ou freqüência de varredura é o número de vezes que a imagem da tela do monitor é atualizada por segundo. *Refresh rates* normalmente são especificadas para operação não entrelaçada, pois é o que os sistemas de vídeo modernos usam. O entrelaçamento era usado em monitores CRT permitindo que a *refresh rate* fosse o dobro que normalmente seria por mostrar linhas alternadas em cada atualização de tela. Ou seja, no modo entrelaçado, metade da tela é desenhada a cada vez.

O entrelaçamento deixou de ser utilizado por causar redução da resolução vertical e porque um sinal entrelaçado necessita de um processo de desentrelaçamento computacionalmente caro necessário para ser adequado aos monitores de tela plana, tais como os monitores LCD.

Suporte para uma dada *refresh rate* requer duas coisas: uma placa de vídeo capaz de produzir as imagens de vídeo na quantidade de vezes por segundo da *refresh rate* e um monitor capaz de manusear e mostrar estes sinais na mesma velocidade. Ela tem grandes efeitos em monitores CRT por causa do efeito *flicker* e do desconforto e da tensão nos olhos causada por ele. *Flicker* é uma instabilidade percebida na tela do monitor, e é menos perceptível nos monitores LCD porque eles não têm uma atualização *per se* (a imagem precisa ser atualizada apenas quando ela é mudada, pois cada *pixel* é ativado ou desativado quando necessário, enquanto que o feixe de elétrons de um monitor CRT esquadrinha e atualiza a tela continuamente). O modo entrelaçado era usado porque causa menos *flicker* do que o modo não entrelaçado.

O *flicker* é mais perceptível quanto maior for o tamanho do monitor. Em monitores CRT de 14” poucas pessoas notam *flicker* com *refresh rate* entre 60 e 72 Hz. Em monitores maiores (17” e 19”), a maioria das pessoas não percebe *flicker* quando a *refresh rate* é de 85 Hz ou mais. Por isso as freqüências de varredura são mais ou menos padronizadas. Valores comuns são 56, 60, 65, 70, 72, 75, 80, 85, 90, 95, 100, 110 e 120 Hz.

A *refresh rate* é controlada pelo sinal de sincronismo vertical gerado pelo controlador de vídeo. Este sinal marca o ponto no qual a atualização da imagem (quadro) corrente termina e o novo quadro começa. Existem ‘M’ períodos de sincronismo horizontal para cada período de

sincronismo vertical, onde ‘M’ é o número de linhas de *pixel* horizontais numa tela definido pela resolução utilizada.

O sinal de sincronismo horizontal marca o ponto no qual uma linha termina e uma nova linha começa. Existem ‘N’ períodos de *clock* para cada período de sincronismo horizontal, onde ‘N’ é o número de *pixels* RGB em uma linha.

Os monitores CRT necessitam retornar o feixe de elétrons para o início da próxima linha. Este procedimento é chamado de retraço horizontal. A figura 16 representa o caminho que o feixe de elétrons percorre na tela de um monitor CRT. A linha horizontal (1) corresponde ao desenho de uma linha da imagem e a linha transversal (2) corresponde ao retraço horizontal.

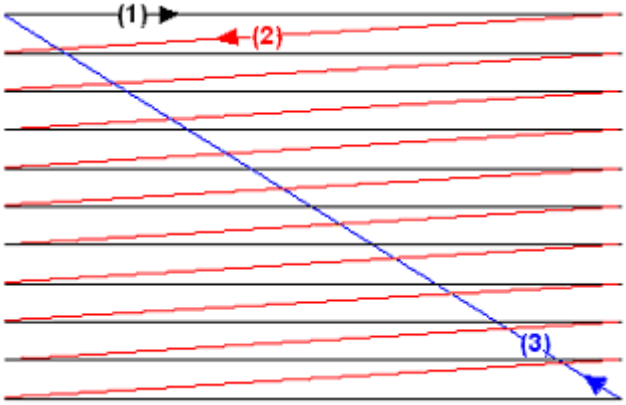


Figura 16 - Traçado do feixe de elétrons sobre a superfície do CRT [24]

O período do sinal de sincronismo vertical é constituído dos ‘M’ períodos de sincronismo horizontal acrescido de ‘M – 1’ tempos de retraço horizontal e de um tempo de retraço vertical. O retraço vertical, representado pela linha 3 na figura 16, é o procedimento realizado para posicionar o feixe de elétrons no início da tela.

Tabela 4 - Resolução da tela x *clock* para CRT

Padrão Gráfico	**Resolução da tela (*pixels* x *pixels*)**	***Refresh rate* (Hz)**	**Sincronismo Horizontal (KHz)**	***Clock* (MHz)**
VGA (Video Graphics Array)	640 x 480	60	34,55	22,11
		72	41,46	26,53
		75	43,19	27,64
SVGA (Super VGA)	800 x 600	72	51,83	41,46
		75	53,99	43,19
		80	57,58	46,07
XVGA (eXtended VGA)	1024 x 758	75	68,21	69,84
		80	72,75	74,50
		85	77,30	79,15
CIF (Common Intermediate Format)	352 x 240 (NTSC)	56	16,12	5,67
		60	17,27	6,08
		65	18,71	6,58
	352 x 288 (PAL)	56	19,34	6,81
		60	20,72	7,29
		65	22,45	7,90
QCIF (Quarter CIF)	176 x 120 (NTSC)	56	8,05	1,42
		60	8,63	1,52
		65	9,35	1,65

	176 x 144 (PAL)	56	9,67	1,70
		60	10,36	1,82
		65	11,22	1,97
4CIF	704 x 480 (NTSC)	72	41,46	29,19
		75	43,19	30,40
		80	46,06	32,43
	704 x 576 (PAL)	72	49,75	35,03
		75	51,83	36,48
		80	55,28	38,92

Portanto, a freqüência do sinal de sincronismo horizontal e do *clock* gerado pelo controlador de vídeo para o monitor LCD dependem do padrão gráfico e da taxa de atualização utilizados, mas para o monitor CRT deve-se considerar também os intervalos de tempo de retraço horizontal e vertical (que dependem do monitor CRT a ser utilizado). A tabela 4 mostra diversas opções para um monitor CRT.

O monitor de vídeo pode ser um monitor CRT (*Cathode Ray Tube*) ou um monitor de tela plana. CRT é um tubo de vácuo no qual imagens são produzidas quando um feixe de elétrons atinge uma superfície fosforescente. O feixe de elétrons é controlado pela deflexão realizada por um campo elétrico ou magnético de forma a varrer a superfície. A figura 17 ilustra o funcionamento de um monitor CRT.

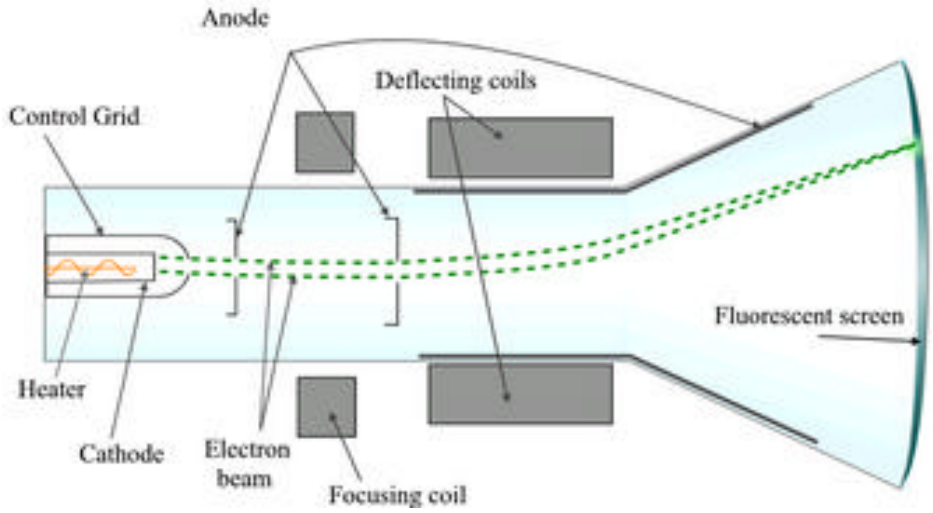


Figura 17 - Estrutura de funcionamento do monitor CRT

Para uma tela colorida o CRT possui três diferentes materiais fosforescentes que emitem as três cores primárias – vermelho (R), verde (G) e azul (B) – e também um feixe de elétron para cada uma destas cores. A intensidade das cores é definida pela quantidade de energia de seu respectivo feixe de elétrons.

Dentre os monitores de tela plana, o mais usado pelos sistemas computacionais é o LCD (*Liquid Crystal Display*). Um monitor LCD é constituído basicamente de um painel de cristal líquido e de um circuito de controle. O painel LCD é um dispositivo fino e achatado constituído de *pixels* coloridos ou monocromáticos colocados na frente de uma fonte ou um refletor de luz. Cada *pixel* é constituído de uma coluna de moléculas de cristal líquido suspensa entre dois eletrodos transparentes e dois filtros polarizados com eixos de polaridade

perpendiculares um ao outro. Sem os cristais líquidos entre eles, a luz passaria por um filtro e seria bloqueada pelo outro. Antes de ser aplicada uma carga elétrica, as moléculas de cristal líquido estão no estado relaxado que não altera a polarização da luz. A aplicação da carga elétrica em suas moléculas altera o alinhamento dos cristais mudando a polarização da luz que passa por eles, permitindo que a luz que passou por um filtro também passe pelo outro. Assim, o cristal se comporta como uma persiana, permitindo ou não que a luz passe.

Todos os painéis obedecem aos mesmos princípios fundamentais, o que varia entre as tecnologias de construção mais adotadas é justamente como as moléculas de cristal liquido são arranjadas entre o refletor de luz e o filtro.

Como pode ser visto na figura 17, cada *pixel* do LCD é dividido em três células, ou sub-*pixels*, que são coloridos de vermelho (R), verde (G) e azul (B) por filtros adicionais. Cada sub-*pixel* pode ser controlado independentemente permitindo milhões de cores possíveis para cada *pixel*.

Tabela 5 - Resolução da tela x *clock* para LCD

Padrão Gráfico	**Resolução da tela (*pixels* x *pixels*)**		***Refresh rate*** **(Hz)**	**Sincronismo Horizontal (KHz)**	***Clock*** **(MHz)**
VGA (*Video Graphics Array*)	480	640	56	26,88	17,20
			60	28,80	18,43
			75	36,00	23,04
SVGA (Super VGA)	600	800	56	33,60	26,88
			60	36,00	28,80
			75	45,00	36,00
XVGA (*eXtended* VGA)	758	1024	56	42,45	43,47
			60	45,48	46,57
			75	56,85	58,21
CIF (*Common Intermediate Format*)	240	352	56	13,44	4,73
			60	14,40	5,07
			75	18,00	6,34
	288	352	56	16,13	5,68
			60	17,28	6,08
			75	21,60	7,60
QCIF (*Quarter* CIF)	120	176	56	6,72	1,18
			60	7,20	1,27
			75	9,00	1,58
	144	176	56	8,06	1,42
			60	8,64	1,52
			75	10,80	1,90
4CIF	480	704	56	26,88	18,92
			60	28,80	20,28
			75	36,00	25,34

	576	704	56	32,26	22,71
			60	34,56	24,33
			75	43,20	30,41

A tecnologia LCD tem algumas desvantagens em comparação com a tecnologia CRT. Por exemplo, enquanto CRT's são capazes de mostrar múltiplas resoluções de vídeo, com a mesma qualidade, monitores LCD normalmente produzem as imagens mais rapidamente e as melhores imagens na resolução nativa. Além disso, o monitor LCD tem de fazer uma interpolação para mostrar imagens com menor resolução do que a nativa. A resolução nativa de um LCD pode ser entendida como a máxima resolução de um monitor CRT.

Como já foi dito anteriormente, um monitor LCD não sofre com problemas de atualização de tela, pois cada *pixel* pode ser ativado ou desativado quando necessário. A desvantagem aqui é o tempo de resposta do *pixel*: se um *pixel* não é atualizado rápido o bastante, aparece um "fantasma na tela" – a imagem atualizada é sobreposta à imagem anterior. Os LCD's mais novos possuem tempo de resposta baixo o bastante (menor que 16 ms) para não causar problemas. Assim, a *refresh rate* dos monitores LCD dependem do seu tempo de resposta. Para um tempo de resposta de 16 ms, teremos uma refresh rate máxima de 60 Hz. A tabela 5 apresenta diversas opções de freqüência de sincronismo horizontal e de *clock*.

4.2. Sincronismo

Para seu correto funcionamento, os monitores CRT precisam dos seguintes sinais de sincronismo:

- **Sinal de *clock* de vídeo:** é o sinal que determina o envio dos dados no formato RGB para o vídeo.
- **Freqüência de varredura:** pode ser vertical ou horizontal, correspondendo ao sinal de sincronismo vertical e de sincronismo horizontal utilizado pelos monitores CRT.
- **Sinal de sincronismo horizontal:** este sinal marca o ponto no qual uma linha atual termina e uma nova linha começa. Existem 'N' períodos de *clock* por cada período de sincronismo horizontal, onde 'N' é o número de *pixels* RGB em uma linha.
- **Sinal de sincronismo vertical:** este sinal marca o ponto no qual o quadro atual termina e o novo quadro começa. Existem 'M' períodos de sincronismo horizontal por cada período de sincronismo vertical, onde 'M' é o número de linhas de *pixel* horizontais numa tela.

As relações entre as formas de onda dos sinais de sincronismo são ilustradas na figura 18.

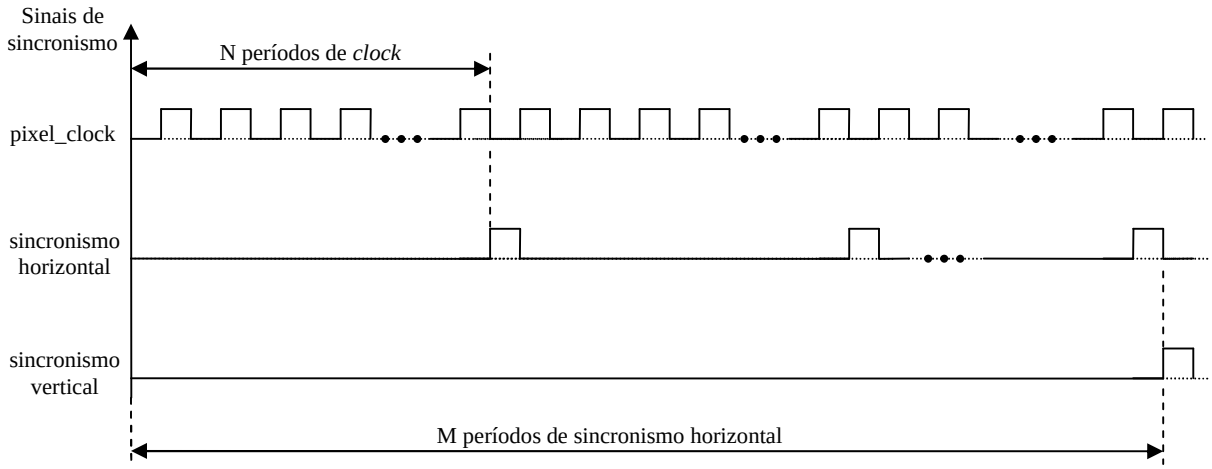


Figura 18 - Formas de onda dos sinais de sincronismo

No caso de um monitor LCD atualizar constantemente sua tela, os sinais de sincronismo necessários são os mesmos do monitor CRT. Entretanto, os monitores LCD não precisam dos intervalos de tempo para retraço vertical e horizontal.

A maioria dos monitores LCD possuem entradas analógicas. Neste caso, o módulo IP apresentado neste trabalho precisaria de um conversor digital/analógico para adequar suas saídas.

5. IP CONTROLADOR DE VÍDEO

A metodologia proposta foi testada através do desenvolvimento de um núcleo IP com a função de controlador de vídeo para monitores LCD que utiliza os mesmos sinais de sincronismo de um monitor CRT. Este módulo é constituído de três partes. Uma delas, chamada de Interpretador, é responsável por receber os comandos enviados ao controlador, interpretá-los e executá-los.

Outra parte, chamada de Seqüenciador, gera a saída do controlador, ou seja, os sinais de sincronismo e os *pixels* da imagem a ser mostrada no monitor de vídeo.

A terceira parte é responsável pelo armazenamento de dados, tais como a imagem, o texto, o hardware cursor fixo e o programável. A figura 19 ilustra o controlador de vídeo através de um diagrama de blocos.

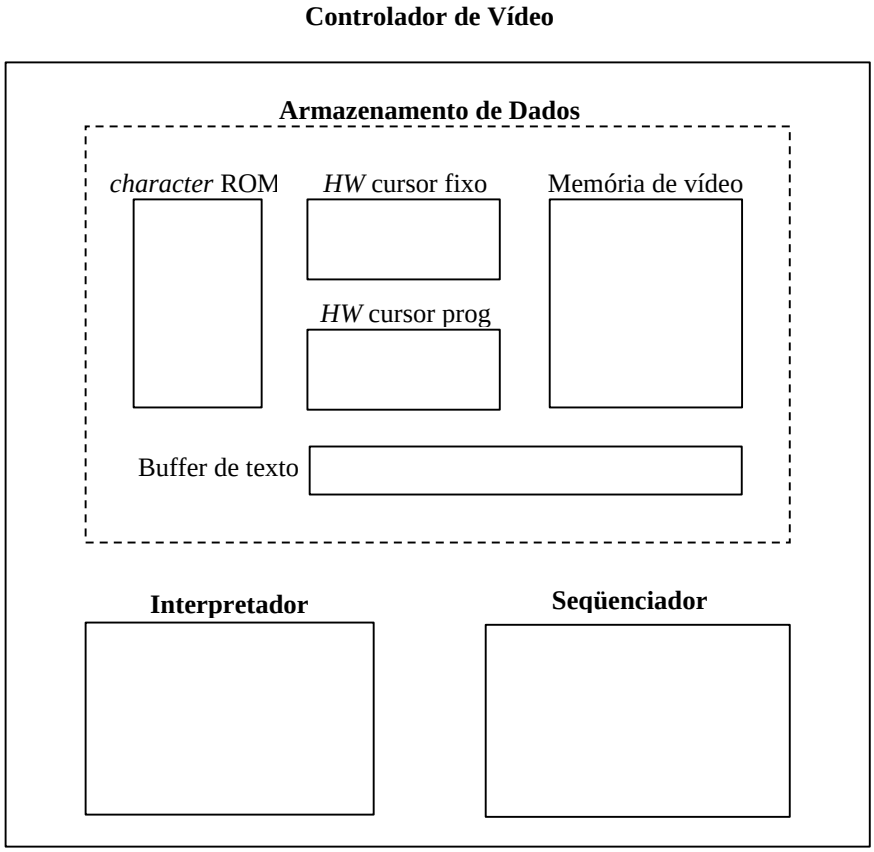


Figura 19 - Diagrama de blocos do controlador de vídeo

Este controlador LCD possui saídas digitais.

5.1. Requisitos funcionais

A especificação de requisitos não faz parte do escopo deste trabalho, podendo ser usada qualquer técnica que consiga capturar os requisitos do sistema. A tabela 6 apresenta os requisitos funcionais deste controlador.

Tabela 6 - Requisitos funcionais do controlador de vídeo

Tamanho da tela e sincronismo vertical	14" @60 Hz
	15" @60 Hz
	17" @75 Hz
Resolução	QCIF NTSC (176 x 120 *pixels*)
	CIF NTSC (352 x 240 *pixels*)
	VGA (640 x 480 *pixels*)
Color depth	1 bit (monocromático)
	8 bits (256 cores)
	16 bits (65.536 cores)
	24 bits (16.777.216 cores)
Paleta	256 cores -> 16.777.216 cores
Sinais de sincronismo	Para monitor LCD
Texto	Possibilidade de escrever texto na tela
Hardware cursor	Fixo e programável: 64x64x2 bits

5.2. Comandos do controlador de vídeo

Os estímulos de entrada do modelo do controlador de vídeo sob teste são fornecidos pelo *testbench* através de um arquivo de entrada constituído por comandos e seus respectivos parâmetros. Estes comandos são apresentados a seguir.

Reset
O comando RESET estabelece o uso de valores *default* pelo controlador. Ele não possui parâmetros, pois estes valores são definidos pelo próprio controlador.

Freqüência de sincronismo vertical
O comando SYNC estabelece a freqüência de sincronismo vertical. O parâmetro deste comando é a freqüência. Inicialmente será implementada a freqüência de 60 Hz.

Resolução da tela
O comando RESOL estabelece a resolução da tela e seus parâmetros são as resoluções horizontal e vertical.

Os padrões gráficos a serem implementados serão QCIF NTSC, CIF NTSC, VGA. Posteriormente, as resoluções SVGA e XVGA também poderão ser implementadas. A resolução *default* é a VGA.

Color Depth
O comando COLOR estabelece a *color depth* do *pixel* da tela e seu único parâmetro é a *color depth* propriamente dita. As *color depth*s a serem implementadas serão de 1, 8, 16 e 24 bits. O valor *default* é 8 bits.

Paleta de cores
O comando PALETA estabelece a paleta de cores a ser usada e carrega a paleta escolhida na memória. Os parâmetros deste comando são o tipo da paleta e a paleta propriamente dita. A

paleta a ser implementada é a paleta que armazena 256 cores com *color depth* de 24 bits (figura 20). O *default* é não usar a paleta.

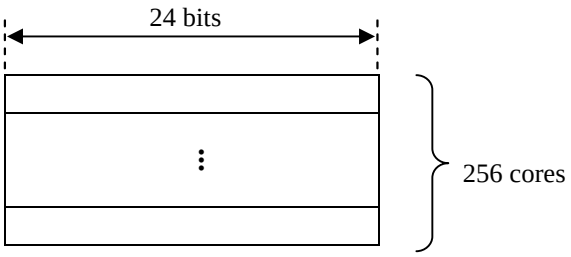


Figura 20 - Paleta de cores

Tipo de monitor
O comando MONITOR estabelece o tipo de monitor para o qual o controlador de vídeo deve gerar os sinais. Esta versão do controlador de vídeo foi desenvolvida para produzir os dados necessários ao funcionamento de um monitor LCD, usando os mesmos sinais de sincronismo de um monitor CRT, mas sem levar em consideração os tempos de retraço horizontal e vertical. Eventualmente, o funcionamento simultâneo de dois monitores, sendo um do tipo LCD e outro do tipo CRT, poderá ser implementado. O parâmetro deste comando é o tipo de monitor.

Escrever texto na tela
O comando TEXTO escreve uma linha de texto com até 10 caracteres na tela do monitor de vídeo. Seus parâmetros são as posições horizontal e vertical do texto na tela, o texto propriamente dito, a cor dos caracteres do texto e a cor de fundo.

A posição na tela é definida considerando-se que a tela é constituída de caracteres em vez de *pixels*, sendo que cada caractere corresponde a uma matriz de 8 x 8 *pixels*. O texto é enviado ao controlador e armazenado no formato ASCII. Ao produzir o *frame*, o controlador deve converter o texto em ASCII para os *pixels* correspondentes e sobrepor a área da imagem com estes *pixels*.

Hardware cursor
O comando CURSOR estabelece o *hardware cursor* a ser usado e define sua posição na tela. Os parâmetros deste comando são o *hardware cursor* a ser usado (fixo ou programável) e a posição do *hardware cursor* na tela (horizontal e vertical).

A imagem do cursor fixo é definida pelo próprio controlador, ou seja, já se encontra armazenada dentro dele. A do cursor programável não está definida e deve ser enviada ao controlador.

Serão implementados dois *hardware cursors*, um fixo e um programável, sendo que ambos serão formados por uma matriz de 64 x 64 *pixels* com 2 bits de *color depth*. A posição do cursor na tela é estabelecida definindo-se a posição horizontal e a posição vertical. Dependendo desta posição os *pixels* da imagem do cursor devem sobrepor a imagem original e/ou os *pixels* de texto.

O *hardware cursor default* é o fixo.

Transferência de *pixels*
O comando TRANSFERENCIA carrega um *pixel* para a memória de vídeo ou a imagem do *hardware cursor* para a área de memória do *hardware cursor* programável.

Um dos parâmetros deste comando é o local de armazenamento. Se este local é a memória de vídeo os outros parâmetros são a posição do *pixel* na tela e o *pixel* no formato RGB. Caso o local seja a memória do *hardware cursor* programável, os parâmetros serão o *pixel* do cursor com *color depth* de 2 bits e a posição deste *pixel* na respectiva memória.

A figura 21 mostra os principais comandos e seus respectivos parâmetros que foram usados no arquivo de comandos.

```
RESET

COLOR
24
CURSOR
FIXO 10 10
TEXTO
"012" 111111111111111111111111 000000000000000000000000 1 1
TRANSFERENCIA
VIDEO_MEM 111111100000000000000000 0 0
```

Figura 21 - Comandos usados no arquivo de comandos

5.3. Verificação funcional

Não serão realizados testes exaustivos para comprovar as funcionalidades de cada modelo do controlador de vídeo. Somente algumas funcionalidades serão verificadas.

O modelo sob teste é considerado correto se conseguir produzir os sinais de sincronismo e os *pixels* necessários para “desenhar” a tela do monitor de vídeo (*frame*). Este *frame* é constituído basicamente da imagem enviada ao controlador pelo *testbench*. A imagem que foi utilizada pelo *testbench* é totalmente vermelha e de dimensões 40 x 32 pixels conforme apresentada na figura 22.

Figura 22 - Imagem original do teste

Esta imagem foi escolhida com uma só cor para facilitar a depuração de código. Por outro lado, o tamanho escolhido para a imagem (40 x 32 pixels) teve a intenção de diminuir o tempo de simulação sem prejudicar os testes, pois uma tela na resolução pretendida (VGA) tem 640 x 480 pixels, o que resultaria num tempo de simulação maior.

Um texto também foi enviado ao controlador juntamente com a posição da tela onde deve aparecer. O texto de teste é constituído de três caracteres. O primeiro caractere deve aparecer na imagem como um quadrado preto. Um quadrado branco deve representar o segundo caractere. E a imagem correspondente ao terceiro caractere deve ser representada por um quadrado metade negro e metade branco. A imagem deste "texto" é apresentada na figura 23.

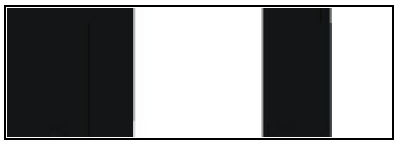


Figura 23 - Imagem correspondente ao texto que deve ser gerado pelo controlador de vídeo

Não foi utilizado um texto com caracteres "reais" devido à maior dificuldade na depuração de código e na verificação visual do arquivo de saída correspondente à imagem. Pelo mesmo motivo também não foi utilizado um cursor "real".

Outro dado enviado ao controlador pelo *testbench* é a posição do cursor. O cursor usado foi o *hardware cursor* fixo, ou seja, a imagem do cursor é pré-definida. Um quadrado cinza, mostrado na figura 24 corresponde à imagem do *hardware cursor* fixo.

Figura 24 - Imagem que corresponde *hardware cursor* fixo

A imagem, o texto e sua posição na tela, serão enviados ao controlador através de comandos, conforme apresentados na seção 5.2.

Assim, antes da imagem da figura 22 ser enviada ao controlador, ela é convertida do formato PNG para o formato RGB pelo comando *convert* presente no Red Hat Linux 9.0. Depois ela é transformada em comandos "TRANSFERENCIA" por um programa criado pelo autor deste trabalho. Estes comandos fazem parte de um arquivo de comandos (figura 21), juntamente com os outros comandos que foram usados pelo *testbench* para estimular o modelo sob teste.

Durante a simulação, além de gerar os estímulos, o *testbench* também captura as saídas dos modelos e produz arquivos do tipo texto com elas. Um destes arquivos (pixel_RGB.out) é constituído de *pixels* da imagem que seria apresentada na tela do monitor de vídeo. Esta imagem deve ser formada pela imagem original recebida pelo controlador, sobreposta pelos *pixels* correspondentes ao texto – caso ele tenha sido enviado ao controlador – e pela imagem do cursor (figura 25). A imagem do cursor deve sobrepor tanto a imagem original quanto a imagem do texto.

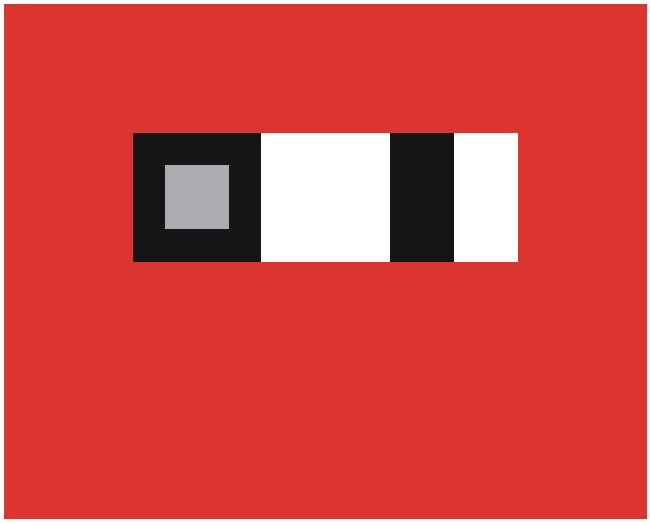


Figura 25 - Imagem resultante da simulação

Para realizar uma inspeção visual e verificar a funcionalidade de cada modelo, o arquivo *pixel_RGB.out* foi convertido no formato RGB por um programa criado pelo autor deste trabalho. Depois ele é convertido para o formato PNG usando novamente o comando *convert*, resultando na figura 25. A figura 26 representa todo este processo.

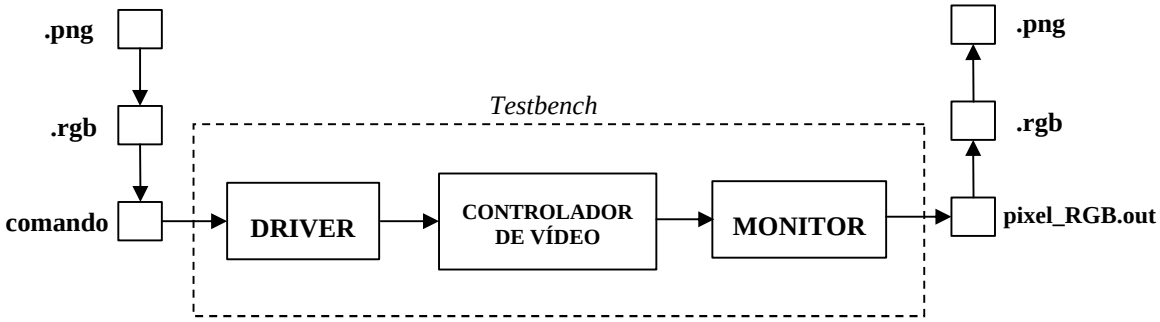


Figura 26 - Arquivos de estímulos e de saída do *testbench*

Além do pixel_RGB.out, o *testbench* cria os arquivos correspondentes aos sinais de sincronismo vertical (vsync.out), horizontal (hsinc.out) e *clock* dos *pixels* (pixel_clock.out) para cada modelo simulado. Estes arquivos foram comparados aos arquivos produzidos pela simulação do Modelo de Referência. Para que o *frame* corresponda à imagem esperada, os sinais de sincronismo e os *pixels* devem ser gerados na ordem correta e, dependendo do nível de abstração do modelo, também pode ser necessário gerá-los no tempo certo.

5.4. Modelos do controlador de vídeo

Os modelos do controlador foram construídos através da metodologia proposta e usando os passos a seguir:

1. A partir dos requisitos funcionais, descrever o Modelo de Referência em SystemC, mas usando somente as características correspondentes ao C/C++.

2. Conversão manual do Modelo de Referência para o Modelo Comportamental.Este modelo começa a usar as características específicas do SystemC, tais como módulos e processos.
3. Conversão automática do Modelo Comportamental para o Modelo RTL usando a ferramenta Cynthesizer da Forte Design Systems.
4. Tradução automática do Modelo RTL para o Modelo HDL também usando o Cynthesizer. A linguagem do Modelo HDL será o Verilog.
5. Síntese lógica do Modelo HDL usando o aplicativo Quartus II da Altera.

A seguir são apresentados os modelos.

5.4.1. Modelo de Referência

Nesta representação, o controlador de vídeo é modelado por uma função e acionado através da chamada desta função. De forma análoga são representados o Interpretador e o Seqüenciador. O armazenamento de dados é realizado através de variáveis globais. Os trechos de código da figura 27 ilustram esta modelagem.

```
// Declaracao de variaveis globais do controlador de vídeo
int ROM[NUM_MAX * M];                          // (NUM_MAX * M) -> Nr de character lines da ROM
int video_mem[X][Y];                           // (X, Y) -> Resolucao maxima da tela
int cursor_prog[DIMENSAO_CURSOR][DIMENSAO_CURSOR];   // Memoria HW cursor programavel
int cursor_fixo[DIMENSAO_CURSOR][DIMENSAO_CURSOR];   // Memoria Hardware cursor fixo
int clut[CORES_PALETA];                              // Paleta de cores

void controlador_video(int comando, int parametro[]) {
        interpretador(comando, parametro);
        sequenciador();
}
```

Figura 27 - Trechos de código do Modelo de Referência

5.4.2. Modelo Comportamental

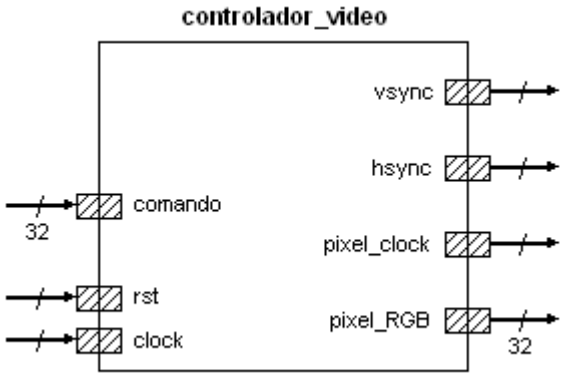


Figura 28 - Modelo Comportamental

No Modelo Comportamental (figura 28), o controlador de vídeo é representado por um módulo, ou seja, este modelo já usa os recursos específicos do SystemC. O Interpretador e o

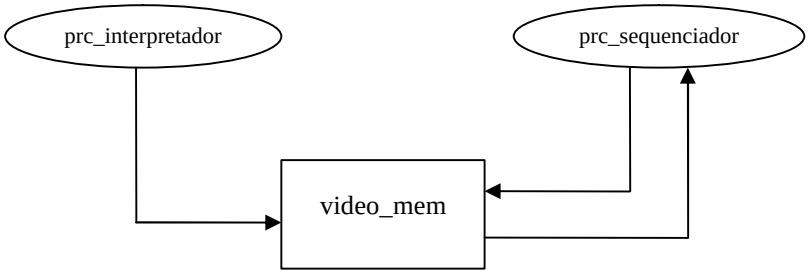
Seqüenciador são modelados como processos do módulo controlador de vídeo e o armazenamento de dados é representado por variáveis globais deste módulo. Os trechos de código da figura 29 ilustram estas representações.

Neste modelo, devido à natureza concorrente dos processos, cada variável que é usada por mais de um processo poderá ser acessada simultaneamente. Para evitar este conflito, o acesso a cada uma destas variáveis deve ser realizado através de um processo específico. Este é o caso da memória de vídeo e do buffer de texto, como é mostrado nos trechos de código da figura 29 e é ilustrado na figura 30.

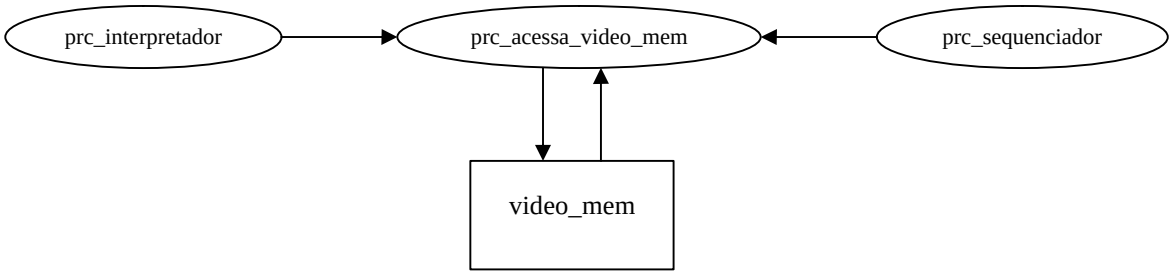
```
SC_MODULE (controlador_video) {
  // ports
  sc_in<int> comando_c;
  sc_in<int> parametro_c0, parametro_c1, parametro_c2;
  sc_in<int> parametro_c3, parametro_c4;
  sc_in<char> parametro_c5;
  sc_in_clk clock;
  sc_in<bool> rst;
  sc_out<bool> vsync_c, hsync_c, pixel_clock_c;
  sc_out<int> pixel_RGB_c;

  // global variables
  sc_uint<M_> ROM[NUM_MAX];                  // (NUM_MAX * M) -> Nr de character lines da ROM
  sc_uint<24> video_mem[X_][Y_];             // (X, Y) -> Resolucao maxima da tela
  int cursor_prog[DIMENSAO_CURSOR][DIMENSAO_CURSOR];   // Memoria HW cursor programavel
  int cursor_fixo[DIMENSAO_CURSOR][DIMENSAO_CURSOR];   // Memoria Hardware cursor fixo
  int clut[CORES_PALETA];                                // Paleta de cores
  ...
    // processes instatiation
    SC_CTHREAD(prc_interpretador, clock.pos());                  // processo Interpretador
    watching(rst.delayed() == 0);

    SC_CTHREAD(prc_sequenciador, clock.pos());                   // processo Sequenciador
    watching(sequenciador_enable.delayed() == 0);

    SC_CTHREAD(prc_acessa_video_mem, clock.pos());       // processo que acessa a memória de vídeo
    watching(rst.delayed() == 0);

    SC_CTHREAD(prc_acessa_buffer_textos, clock.pos());   // processo que acessa o buffer de texto
    watching(rst.delayed() == 0);
  ...
}
```

Figura 29 - Trechos de código do Modelo Comportamental

Na realidade, esta mudança não seria necessária neste modelo, mas a síntese lógica apresenta erros se ela não for realizada.



(a) Acesso à memória de vídeo por dois processos



(b) Acesso à memória de vídeo realizada por um único processo

Figura 30 - Mudança da forma de acesso à memória de vídeo do controlador

5.4.3. Modelo RTL

Este modelo foi gerado automaticamente pelo Cynthesizer a partir do Modelo Comportamental. O Cynthesizer aproveita a estrutura externa do Modelo Comportamental (figura 30). O que é diferente no Modelo RTL é a representação da estrutura interna.

Desta forma, o controlador de vídeo continuou a ser representado como um módulo, mas, por exemplo, as variáveis que eram acessadas através de processos no Modelo Comportamental, agora também são representadas como módulos. Os trechos de código da figura 31 ilustram a estrutura externa do módulo controlador de vídeo e o módulo que corresponde à memória de vídeo para este modelo.

```
/* Declaration of the synthesized module. */
struct controlador_video : public sc_module {
  sc_in<sc_int<32> > comando_c;
  sc_in<sc_int<32> > parametro_c0;
  sc_in<sc_int<32> > parametro_c1;
  sc_in<sc_int<32> > parametro_c2;
  sc_in<sc_int<32> > parametro_c3;
  sc_in<sc_int<32> > parametro_c4;
  sc_in<sc_int<8> > parametro_c5;
  sc_in<bool > clock;
  sc_in<bool > rst;
  sc_out<bool > vsync_c;
  sc_out<bool > hsync_c;
  sc_out<bool > pixel_clock_c;
  sc_out<sc_int<32> > pixel_RGB_c;
   …
}

struct controlador_video_RAM_1280X24_3 : public sc_module {
  sc_in<sc_uint<24> > DIN;
  sc_in<sc_uint<1> > RW;
  sc_in<sc_uint<11> > in1;
  sc_out<sc_uint<24> > out1;
  sc_in<bool > clk;
  controlador_video_RAM_1280X24_3( sc_module_name name );
  SC_HAS_PROCESS(controlador_video_RAM_1280X24_3);
  sc_uint<24> mem[1280];
  sc_uint<11> in1_5;
  sc_uint<1> RW_5;
  sc_uint<24> DIN_5;
  sc_signal<sc_uint<24> > DIN_6;
  sc_signal<sc_uint<1> > RW_6;
  sc_signal<sc_uint<11> > in1_6;
  void thread1();
  void thread2();
};
```

Figura 31 - Trechos de código do Modelo RTL

A figura 32 representa o módulo de Memória de Vídeo.

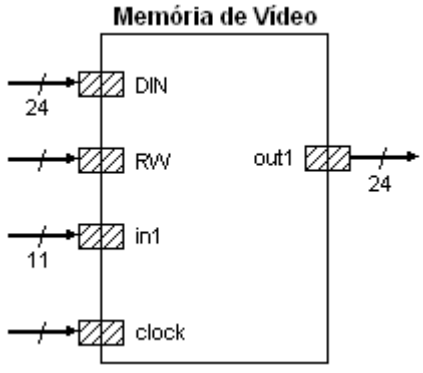


Figura 32 - Estrutura do módulo da memória de vídeo

5.4.4. O Interpretador e o Seqüenciador foram desmembrados em diversos processos através da síntese realizada Cynthesizer.**Modelo HDL**

Este modelo corresponde ao Modelo RTL com a diferença de ser descrito em Verilog. A figura 33 ilustra a estrutura externa do módulo controlador de vídeo e a "variável" que corresponde à memória de vídeo para este modelo.

```
module controlador_video(comando_c, parametro_c0, parametro_c1, parametro_c2, parametro_c3, parametro_c4, parametro_c5, clock, rst, sequenciador_enable, interpretador_enable, vsync_c, hsync_c, pixel_clock_c, pixel_RGB_c);

    input [31:0] comando_c;
    input [31:0] parametro_c0;
    input [31:0] parametro_c1;
    input [31:0] parametro_c2;
    input [31:0] parametro_c3;
    input [31:0] parametro_c4;
    input [7:0] parametro_c5;
    input clock;
    input rst;
    output vsync_c;
    output hsync_c;
    output pixel_clock_c;
    output [31:0] pixel_RGB_c;
    ...

endmodule
```

```
module controlador_video_RAM_1280X24_3(DIN, RW, in1, out1, clk);

    input [23:0] DIN;
    input RW;
    input [10:0] in1;
    input clk;
    output [23:0] out1;
    ...

endmodule
```

Figura 33 - Trechos de código do Modelo HDL

5.4.5. Considerações

Memória de vídeo

Para facilitar a depuração de código e diminuir o tempo de simulação, a imagem usada para teste tem tamanho 40 x 32 *pixels* com *color depth* de 24 bits. Conseqüentemente, a memória de vídeo é constituída de 1280 posições de 24 bits cada, ou seja, 3840 bytes.

Não houve problema em alocar esta quantidade de memória internamente na FPGA Altera Ciclone II EP2C70F672C6. Entretanto, para um sistema VGA com 24 bits de *color depth*, seria preciso uma memória de 900 KB, tornando necessário o uso de uma memória externa.

Esta memória precisaria ser modelada em RTL para que os efeitos de seu uso, tais como atraso e latência, pudessem ser simulados juntamente com o controlador de vídeo.

O Cynthesizer inclui um Gerador de Modelos de Memória (bdw_memgen) para modelar memórias através da criação de uma biblioteca. As memórias geradas desta forma fornecem um modelo executável para simulação C++ em RTL e definem a interface em Verilog que deve ser usada na implementação final. Este gerador não foi usado para a implementação do modelo comportamental.

Durante a fase de alocação da síntese comportamental, o Cynthesizer tenta mapear *unflattened arrays* para alguma memória existente em sua biblioteca. Como uma memória apropriada não está disponível nesta biblioteca, o Cynthesizer gerou seu próprio modelo de memória.

Exigências para a síntese comportamental do Cynthesizer
Para que um projeto seja sintetizado de maneira apropriada, ele deve seguir certas convenções nos códigos utilizados como entrada do Cynthesizer [21]:

- O projeto deve ser representado como um SC_MODULE com um ou mais SC_CTHREAD's, ou seja, a estrutura externa do módulo deve ter precisão de pino.
- O comportamento da inicialização deve ser implementado através do comportamento do *reset* dentro de um SC_CTHREAD.
- A funcionalidade algorítmica deve ser implementada como um *loop* infinito dentro de um SC_CTHREAD.
- O projeto de *I/O* deve ser implementado dentro de um SC_CTREAD com precisão de ciclo.

Modelos x Linhas de código
Pode-se notar no quadro 1 que a quantidade de linhas de código dos modelos de Referência e Comportamental são muito próximas. O mesmo ocorre com os Modelo RTL e HDL. Este fato indica que as conversões entre estes modelos não são difíceis e, provavelmente, não introduzem muitos erros.

Modelo	Quantidade de linhas de código
de Referência	681
Comportamental	731
RTL	4591
HDL	3365

Quadro 1 - Comparação de número de linhas de código por modelo

Por outro lado, a diferença entre a quantidade de linhas de código do Modelo HDL para o Modelo Comportamental foi de aproximadamente 400%. Isto mostra o salto de complexidade, detalhe e informação entre estes dois modelos. Também denota que esta conversão seria bem mais demorada e propensa a erros se fosse feita manualmente. Este é um importante indicativo dos benefícios da síntese comportamental.

Também se nota que SystemC precisa de uma quantidade 36% maior de linhas de código em RTL do que Verilog, indicando que não há grandes diferenças entre estas linguagens no nível RTL.

5.5. Resultados da síntese lógica

Os arquivos do Modelo HDL em Verilog foram utilizados como entrada do Quartus II para que seja realizada a síntese lógica correspondente a uma FPGA Altera Ciclone II EP2C70F672C6. Os resultados desta síntese são apresentados a seguir.

Recursos usados
O quadro 2 mostra um resumo dos recursos da FPGA Altera Ciclone II EP2C70F672C6 usados para implementar o IP controlador de vídeo.

	Usado	**Total**
Elementos lógicos (células)	53608 (78%)	68416
Pinos	239 (57%)	422
Registros	736	
Freqüência de *clock* máxima	62,70 MHz (15.950 ns)	

Quadro 2 – Recursos da FPGA Altera Ciclone II EP2C70F672C6 utilizados

Timing Closure Floorplan
O *Timing Closure Floorplan* da ferramenta Quartus II pode ser usado para designar recursos da FPGA ou para visualizar os resultados da síntese. Ele permite ver a disposição dos elementos lógicos, estimar a temporização, o congestionamento do roteamento, e outras características do projeto.

A seguir são explicadas algumas das opções de visualização [26]:

- ***Package Top*:** mostra a visão de cima do encapsulamento do dispositivo (figura 34).
- ***Package Bottom*:** mostra a visão de baixo do encapsulamento do dispositivo (figura 35).
- ***Interior Cells*:** permite mostrar as células lógicas do dispositivo (figura 36).
- ***Interior* LABs:** permite mostrar os conjuntos de células lógicas fisicamente agrupadas, chamadas *Logic Array Blocks*.

As figuras 34, 35 e 36 são resultantes da síntese do IP controlador de vídeo.

É importante ressaltar que não houve preocupação em otimizar a quantidade de pinos de entrada e de saída do IP controlador de vídeo.

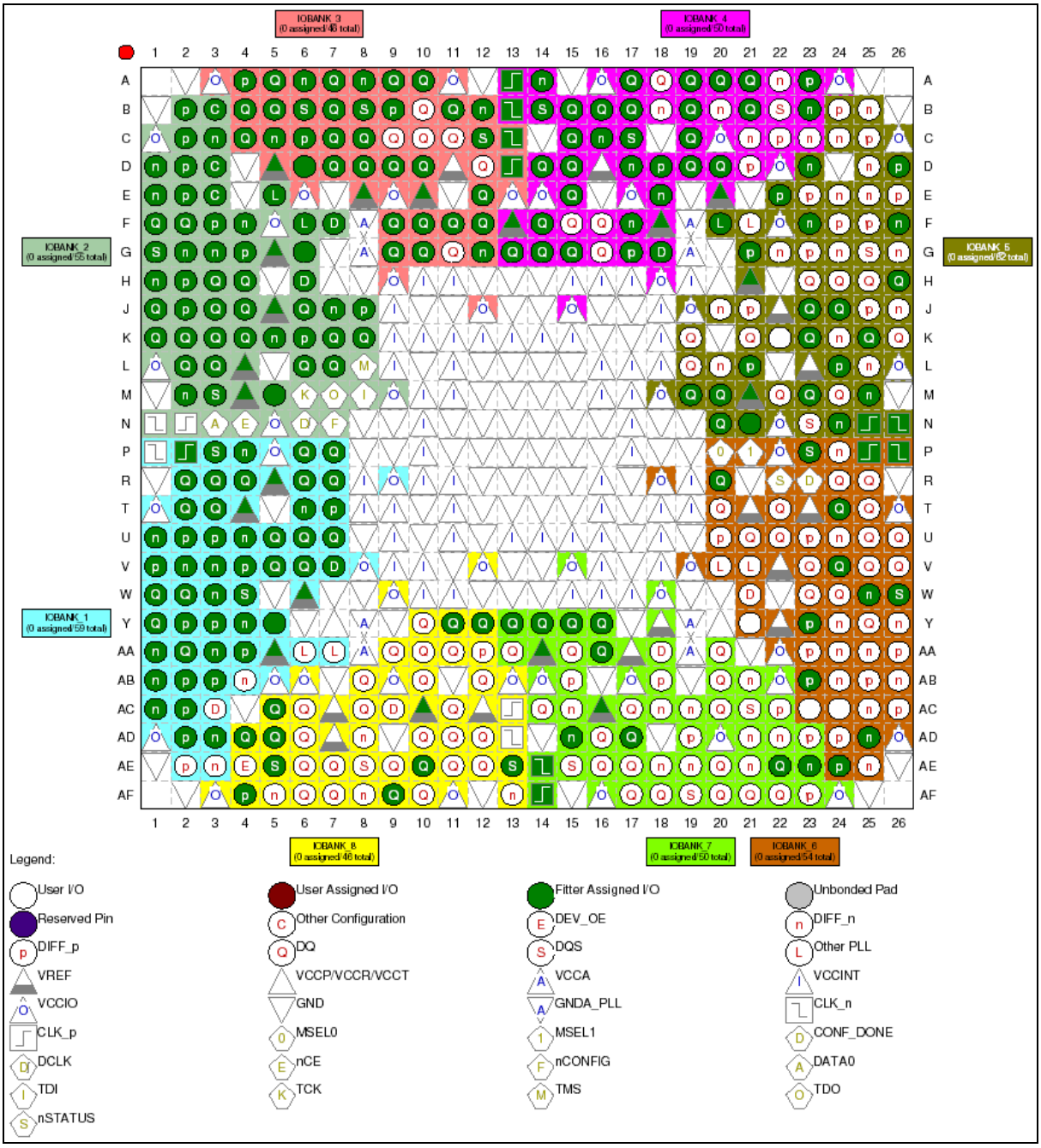


Figura 34 - *Timing Closure Floorplan View – Package Top*

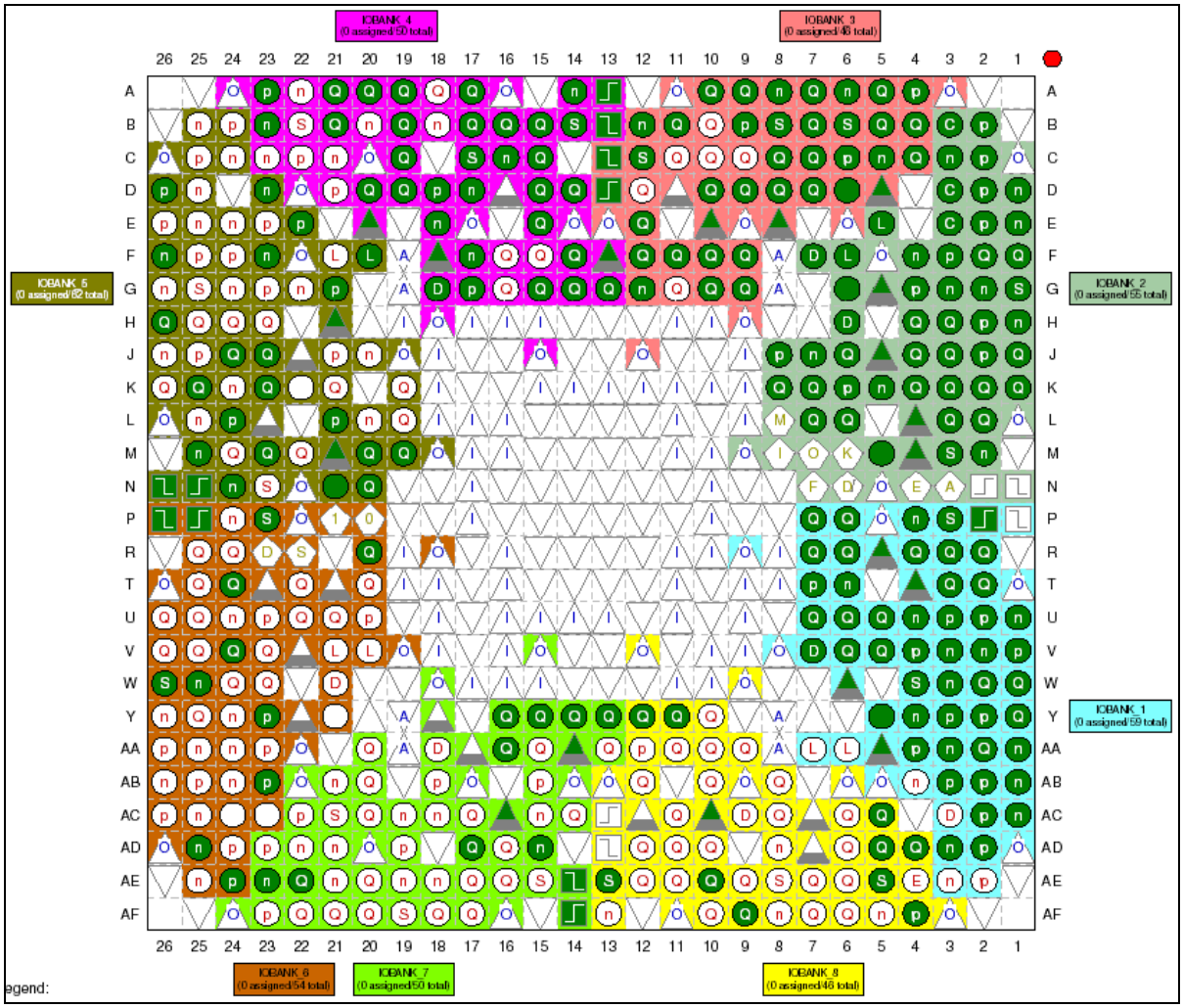


Figura 35 - *Timing Closure Floorplan View – Package Bottom*

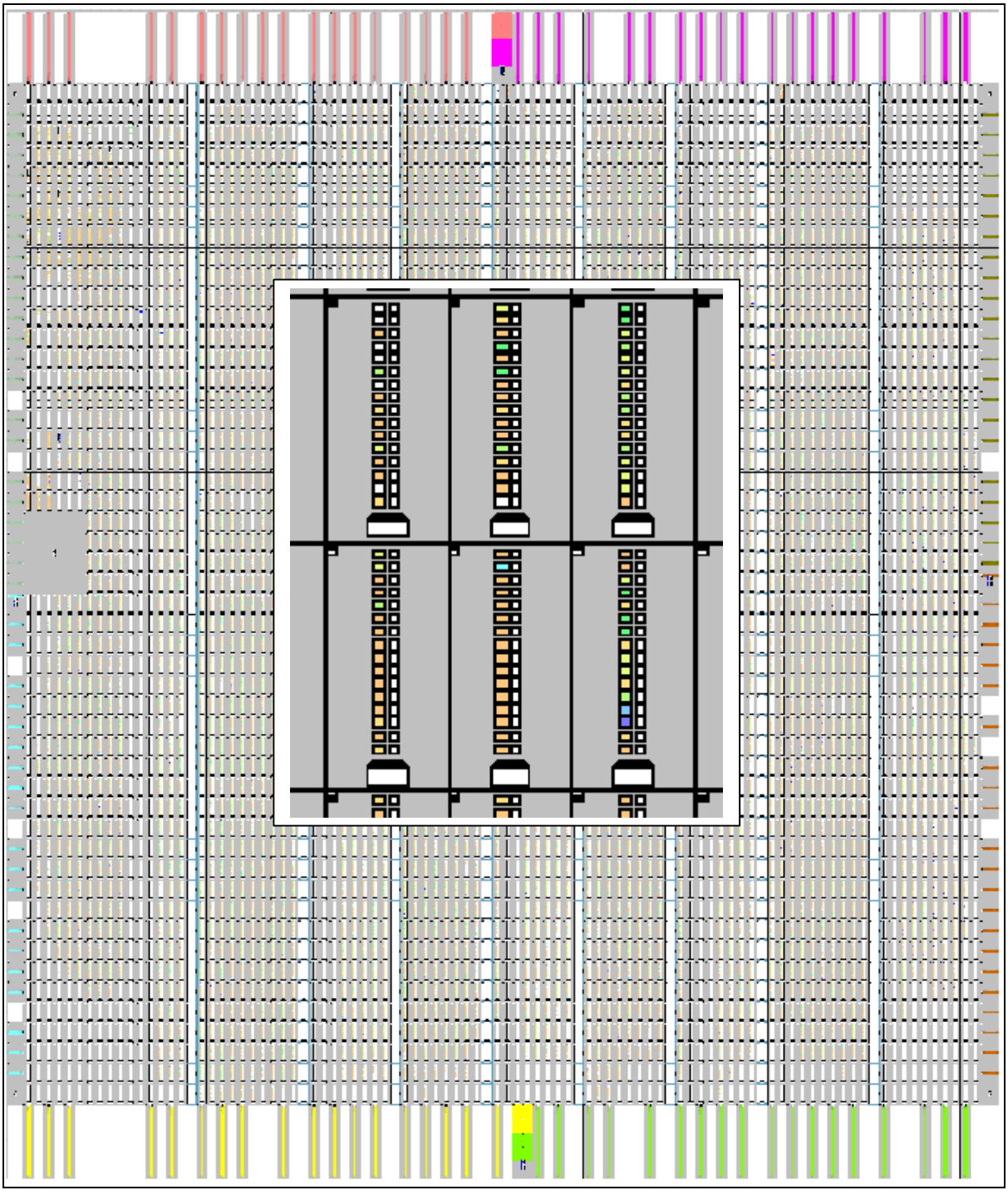


Figura 36 - *Timing Closure Floorplan View – Interior Cells*

A figura 36 procura mostrar as células lógicas usadas para implementar o IP controlador de vídeo na FPGA Altera Ciclone II EP2C70F672C6. Ao centro desta figura pode-se visualizar uma ampliação de parte da FGPA. Na figura 27 é mostrada a legenda referente a esta ampliação. As células são coloridas com base na distância máxima para qualquer *fanout*.

Figura 37 – Legenda da figura 36

6. CONCLUSÕES E TRABALHOS FUTUROS

Neste trabalho foi proposta uma metodologia de projeto de sistemas digitais para o projeto de módulos IP através do refinamento da especificação do projeto em quatro modelos, representados em três níveis de abstração, que utilizam, principalmente, SystemC como ambiente de desenvolvimento. Os modelos para cada ponto do fluxo de projeto que engloba a metodologia foram definidos (Modelo de Referência, Comportamental, RTL e HDL) e foram apresentadas alternativas para a automação de alguns passos.

A metodologia foi testada através do projeto de um núcleo IP controlador de vídeo que usou a síntese comportamental do Cynthesizer. Esta ferramenta utilizou o Modelo Comportamental para gerar os modelos RTL e HDL. A diferença entre a quantidade de linhas de código do Modelo HDL para o Modelo Comportamental foi de quase 400%, o que mostra o salto de complexidade, detalhe e informação entre estes modelos. Esta diferença indica que o uso do Cynthesizer possibilitou um ganho de produtividade significativo ao automatizar a criação de dois dos quatro modelos previstos.

O aumento da quantidade de modelos e níveis de abstração foi compensando pelo aumento da produtividade. Por exemplo, ao se ater aos aspectos funcionais, o Modelo de Referência aumentou a velocidade de simulação e diminuiu o esforço de depuração.

Esta metodologia de análise, projeto, implementação, ciclos de teste e refinamentos foi interessante por minimizar riscos associados ao projeto, diminuindo seu tempo de desenvolvimento. O benefício fundamental do uso de uma metodologia de projeto *top-down* como esta é que as decisões foram feitas no tempo apropriado, usando o mínimo esforço possível para o projeto ser alterado quando fosse necessário. Cada vez que uma decisão foi tomada, ela pôde ser testada num contexto do modelo global do sistema para verificar seus efeitos.

Um estudo comparativo entre implementações usando métodos tradicionais de projeto de sistemas digitais e a metodologia proposta é necessário para se obter um resultado mais preciso sobre a eficiência da metodologia quanto ao tempo de desenvolvimento, área utilizada na implementação final e outros quesitos que possam ser interessantes nesta comparação.

Apesar de ter sido testada no desenvolvimento de um único IP, esta metodologia possibilita o projeto de sistemas completos. No entanto, para sistemas que necessitem de vários IP’s, pode ser necessário modelar a comunicação entre eles através de transações.

Num modelo em nível de transação (TLM), os detalhes de comunicação entre os módulos são separados dos detalhes funcionais de cada módulo. Este ponto merece estudos mais profundos e, possivelmente, uma adaptação da metodologia proposta, pois estudos têm demonstrado que os TLM’s aceleram a simulação e permitem explorar e validar alternativas em projetos com grande necessidade de comunicação entre módulos [8].

Como mais uma vantagem, SystemC também dá suporte a refinamento independente de funcionalidade e comunicação, que é crucial para o desenvolvimento eficiente de TLM’s.

A metodologia também poderia ser melhor desenvolvida para garantir a reutilização dos IP’s gerados através de seu uso e para possibilitar a reutilização do *testbench* escrito em SystemC para o Modelo HDL através da co-simulação de linguagens. Ou seja, simular o *testbench* descrito em SystemC com o Modelo HDL escrito em Verilog ou VHDL. Isto tornaria a

metodologia e os sistemas projetados por ela mais robustos e confiáveis, além de possibilitar maior diminuição do tempo de desenvolvimento.

Outra melhoria, interessante por possibilitar o aumento da produtividade dos projetistas de hardware, seria identificar as construções em SystemC relacionadas ao Modelo Comportamental melhor adaptadas para uso dos recursos do Cynthesizer e para a síntese lógica.

A integração de aplicativos de síntese comportamental com ferramentas de síntese lógica e de layout também se mostra necessária no processo de automação de projetos. O Cynthesizer, em particular, mostrou a necessidade de suporte em ambiente gráfico para facilitar seu uso e diminuir ainda mais o tempo de projeto.

A automação do desenvolvimento de sistemas digitais iniciando em altos níveis de abstração se revelou um aspecto fundamental para permitir o aumento crescente de produtividade exigido pelo mercado. Os resultados do uso da metodologia proposta indicam que ela contribui neste sentido e SystemC se mostrou adequada para esta tarefa.

7. REFERÊNCIAS BIBLIOGRÁFICAS

[1] Bergamaschi, R. A.; Cohn, J., “The A to Z of SoCs”, Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design, pp. 790-798, 2002.

[2] Wolf, W., *Computer as Components: Principles of Embedded Computing System Design*, Academic Press, 2001.

[3] Bergamaschi , R. A.; Lee, W. R., “Designing systems-on-chip using cores”, Proceedings of the ACM/IEEE Design Automation Conference, pp. 420-425, 2000

[4] Keutzer, K., “A Disciplined Approach to the Development of Platform Architectures”, Synthesis and System Integration of Mixed Technologies, Nara, Japan, October 18 - 19, 2001.

[5] Chang, H.; Cooke, L.; Hunt, M.; Martin, G.; McNelly, A.; Todd, L., *Surviving the SoC Revolution – A Guide to Platform-Based Design*, Kluwer Academic Publishers, Boston, 1999.

[6] Bhasker, J., *A SystemC Primer*, Star Galaxy Publishing, 2002.

[7] Arnout, G., “SystemC standard”, Proceedings of Asia and South Pacific Design Automation Conference, 2000, pp. 573-577.

[8] Black, D. C.; Donovan, Jack, S*ystemC: from the ground up*, Kluwer Academic Publishers, Boston, 2004

[9] Claasen, T., “System on a chip: Changing IC design today and in the future”, IEEE Micro, vol. 23, no. 3, pp. 20–26, May/Jun. 2003.

[10] Savage, W.; Chilton, J.; Camposano, R., “IP reuse in the system on a chip era”, in Proc. 13th International Symposium on System Synthesis, Pages: 2-7, 2000.

[11] Keating, M.; Bricaud, P., *Reuse Methodology Manual for System-on-a-Chip Designs*, Second Edition; Kluwer Academic Publishers; 2002.

[12] Medeiros, S. Q., “Utilizando aspectos no projeto de sistemas de hardware desenvolvidos com SystemC”, Jul. 2005, Disponível em <http://www.consiste.dimap.ufrn.br/~sergio/qualificacao.pdf>. Acesso em jul-06.

[13] Martin, G, “SystemC and the Future of Design Languages: Opportunities for Users and Research”, XVI Symposium on Integrated Circuits and Systems Design - São Paulo, SP, Brasil – 2003.

[14] Sanguinetti, J.; Pursley, D., “High-level Modeling and Hardware Implementation with General Purpose Languages and High-level Synthesis”, Ninth IEEE/DATC Electronic Design Processes Workshop, April 2002.

[15] Cesfirio, W.; Baghdadi, A.; Gauthier, L.; Lyonnard, D.; Nicolescu, G.; Paviot, Y.; Yoo, S.; Jerraya, A.A.; Diaz-Nava, M.; “Component-Based Design Approach for Multicore SoCs”, Proceedings of the 39th conference on Design automation, pp. 789-794, 2002.

[16] Varma. A., “Modeling and Synthesis with SystemC”, Bradley Department of Electrical Engineering, Master of Science thesis, Virginia Tech, 2001.

[17] Jantsch, A; Kumar, S; Hemani, A, “Technical report”, Royal Institute of Technology, Mar. 2000.

[18] Gajski, D.; Gerstlauer, A., “System-level Abstraction Semantics”, ISSS'02, Japan, Oct. 2002.

[19] Bhatnagar, H. *Advanced ASIC Chip Synthesis: Using Synopsys Design Compiler and Primetime*, Kluwer Academic Publishers, 2001

[20] Synopsys Inc., Systemc Studio. Architecture Design and Analysis. Disponível em <http://www.synopsys.com/products/cocentric_studio/cocentric_studio.html>. Acesso em jul-06.

[21] Forte Designs System, Cynthesizer Methodology Guide for Cynthesizer 2.4.2, 2005

[22] Celoxica Limited. Agility Compiler. SystemC Behavioral Design and Synthesis. Disponível em <http://www.celoxica.com/products/agility/default.asp>. Acesso em jul-06.

[23] Araújo, G., "The Brazil IP Network", 2002. Disponível em <http://www.mct.gov.br/upd_blob/2380.pdf>. Acesso em jul-06.

[24] The Computer Language Co. Inc, *Computer Desktop Encyclopedia*, 2001.

[25] Altera Corporation, *Quartus II Introduction for Verilog Users*, Altera Quartus II 6.0 software, 2006.

[26] Altera Corporation, *Quartus II Help Version 6.0*, 2006.

[27] Forte Design Systems Cynthesizer, Disponível em <http://www.forteds.com/index.asp?bhcp=1>. Acesso em out-06.

[28] McFarland, M. C.; Parker, A. C.; Camposano, R., "The high-level synthesis of digital systems", Proceedings of the IEEE, pp. 301-318, February 1990.

[29] Altera Corporation. Disponível em <www.altera.com>. Acesso em out-06.

[30] Foley, J. D.; Dam, A., Feiner; S. K.; Hughes; J. F., Computer Graphics: Principles and Practice, Addison-Wesley Publishing Company, Inc, USA, second ed., 1990.

[31] Kane, G., The CRT Controller Handbook, OSBORNE/McGraw-Hill, Berkeley, California, 1980.

[32] Davis, S., Computer Data Displays, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1969.