
Universidade Federal de Campina Grande
Centro de Ciências e Tecnologia
Programa de Pós-Graduação em Engenharia Elétrica

# Um Fluxo de Prototipagem Rápida em FPGA para Algoritmos de Processamento de Vídeo

*Dissertação apresentada à Coordenação do Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, em cumprimento às exigências para obtenção do Grau de Mestre em Ciências no Domínio da Engenharia Elétrica.*

Área de Concentração: Microeletrônica

Candidato:
**Marcos Eduardo do Prado Villarroel Zurita**
Orientadores:
**Raimundo Carlos Silvério Freire**
**Cleonilson Protásio de Souza**

Campina Grande
2009

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

Z96f Zurita, Marcos Eduardo do Prado Villarroel

Um Fluxo de prototipagem rápida em FPGA para algoritmos de processamento de vídeo / Marcos Eduardo do Prado Villarroel Zurita — Campina Grande, 2009.

100 f.

Dissertação (Mestrado em Engenharia Elétrica)- Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática.

Referências.

Orientadores: Prof. Dr. Raimundo Carlos Silveira Freire; Cleonilson Protásio de Souza.

1. Fluxo de Projeto 2. Síntese de Alto Nível 3. Processamento de Vídeos 4. FPGA I. Título.

CDU 621.6 (043)

**UM FLUXO DE PROTOTIPAGEM RÁPIDA EM FPGA PARA ALGORITMOS DE PROCESSAMENTO DE VÍDEO**

**MARCOS EDUARDO DO PRADO VILLARROEL ZURITA**

Dissertação Aprovada em 12.06.2009

**RAIMUNDO CARLOS SILVÉRIO FREIRE, Dr., UFCG**
**Orientador**

**CLEONILSON PROTÁSIO DE SOUZA, D.Sc., CEFET-MA**
**Orientador (Ausência Justificada)**

**ELMAR UWE KURT MELCHER, Dr., UFCG**
**Componente da Banca**

**VINCENT PATRICK MARIE BOURGUET, Dr., Visitante – UFCG**
**Componente da Banca**

CAMPINA GRANDE - PB
JUNHO - 2009

*"Não podemos resolver nossos problemas pensando da mesma forma que quando os criamos"*
*Albert Einstein*

# Agradecimentos

Acima de tudo agradeço a Deus por todas as oportunidades e inúmeras graças alcançadas ao longo deste mestrado.

De forma especial agradeço:

Aos meus pais, Carlos A. Javier Zurita e Rosa Maria V. Zurita, a quem devo muitos dos meus conhecimentos pois muito lutaram pela minha formação profissional e pessoal.

A minha noiva Silvana Lopes de Carvalho por sua dedicação, companheirismo, paciência e pela imensa ajuda na revisão e correção dos textos desta dissertação.

Ao professor Raimundo C. S. Freire, meu orientador, por sua orientação e por todos os conhecimentos e experiência de inestimável valor que me tem passado ao longo dos últimos anos.

Ao professor Cleonilson Protásio de Souza, por sua fundamental participação neste projeto, auxiliando-me diversas vezes na escrita desse documento.

Aos professores Elmar Uwe Kurt Melcher e Edna Natividade Barros, que me integraram ao Brazil-IP, LINCS e ao programa de cooperação tecnológica entre o governo brasileiro e a *École Nationale Supérieure des Mines de Saint-Étienne*, na França, permitindo-me ampliar meus conhecimentos em microeletrônica, essenciais a este trabalho.

Aos engenheiros Francisco Osman Oliveira Gomes, Wilson Rosas Vasconcelos Neto e Isaac Maia Pessoa, pelo seu apoio e companheirismo.

A equipe da Divisão de Vídeo da STMicroelectronics, com a qual trabalhei: Xavier Cauchy, Anthony Philippe, Thomas Kunlin, Isabelle Faugeras, Philippe Damalix, Bruno Théry e Fritz Lebowsky, que tanto me apoiaram e auxiliaram no desenvolvimento e avaliação do fluxo de projeto proposto.

Aos professores Bernard Dhalluin, Jean Pierre Bonnot e Nouria Chaouy por me auxiliarem na busca e colocação em um estágio na STMicroelectronics.

A François Englebert, da CoWare, e Thomas Bollaert, da Mentor Graphics, pelo auxílio no fornecimento de informações técnicas a respeito de seus produtos.

Gostaria de agradecer também ao CNPq por subsidiar financeiramente meus estudos no curso de Mestrado em Tecnologia de Microeletrônica e Gerenciamento de Fabricação (*Master's In Microelectronics Technology & Manufacturing Management*) da *École Nationale Supérieure des Mines de Saint-Étienne*, na França.

E, por fim, agradeço a todos aqueles que de alguma forma tenham me auxiliado no decorrer deste mestrado.

# Prefácio

Assim como a complexidade dos sistemas digitais tem avançado continuamente ao longo dos anos, a dos dispositivos de lógica programável tem seguido o mesmo ritmo. Os FPGAs contêm atualmente milhões de portas e suas freqüências de operação podem chegar a 600 MHz. Isso os torna capazes de serem utilizados até mesmo na codificação e decodificação de vídeo em tempo real. Ao mesmo tempo, as constantes reduções no custo desses dispositivos aliado ao reduzido tempo de prototipagem em relação ao ASIC equivalente, tornam essa tecnologia FPGA atrativa para prototipagem de sistemas VLSI, auxiliando projetistas a melhor ajustarem e validarem soluções antes da sua fabricação em ASIC, ou mesmo as empregando como solução final para o mercado. Contudo, a complexidade dos sistemas digitais parece aumentar mais rapidamente do que o avanço das técnicas e ferramentas de projeto baseadas em RTL, criando um caminho tortuoso entre a modelagem do sistema (geralmente, um algoritmo C++) e o protótipo final.

Este trabalho propõe um fluxo alternativo de projeto para acelerar a prototipagem em FPGA de algoritmos de processamento de vídeo. Baseada nas recentes ferramentas de síntese de alto nível (HLS) disponíveis no mercado, a metodologia estabelece regras gerais para orientar o projetista a implementar em FPGA sistemas descritos em C ou C++, com esforços e tempo de desenvolvimento reduzidos, atingindo ainda assim resultados minimamente satisfatórios.

Um estudo de caso utilizando a metodologia proposta é apresentado ao final desse documento, discutindo as dificuldades encontradas e apresentando os resultados práticos obtidos na sua implementação.

Como contribuição principal deste trabalho pode-se citar a avaliação da síntese de alto nível no desenvolvimento de sistemas de processamento de vídeo. Os estudos e resultados obtidos indicam um notável avanço das ferramentas de síntese de alto nível atualmente disponíveis no mercado, o que já as torna capazes de serem empregadas no desenvolvimento de sistemas complexos e exigentes, como os sistemas de processamento de vídeo.

Adicionalmente, o Catapult, ferramenta de síntese utilizada no fluxo proposto, mostra-se hoje entre as mais poderosas do mercado, sendo adotada por multinacionais como a Panasonic, Siemens, STMicroelectronics, Nokia e a Alcatel. Apesar disso, nenhuma licença desse programa está atualmente atribuída ao Brasil. Desta forma, o trabalho aqui apresentado também contribui fornecendo à academia informações e avaliações independentes sobre esse utilitário.

**Palavras-chave:** Fluxo de Projeto, Síntese de Alto Nível, Processamento de Vídeo, Prototipagem Rápida, FPGA.

# Abstract

Following the constant increasing complexity of SoC devices, programmable logic devices as FPGA's now contain millions of logic gates and can operate at speeds close to 600 MHz, enabling their use even for real time video coding and decoding. This increasing of capacity combined with fast prototyping time and relative low-cost for short production cycles makes FPGA's a very attractive technology for systems validation or even as final market solution. However, the complexity of digital systems seems to increase faster than the advance of design techniques and tools based on RTL, creating a very tortuous path from the system modeling (usually a C++ algorithm) to the final prototype.

This document proposes an alternative methodology for accelerating the FPGA prototyping of video processing subsystems for demonstrative purposes. Based on a High-Level Synthesis (HLS) design tool, the alternative design flow establishes the general rules to guide the designer from an algorithm system level C description to its FPGA prototyping, keeping satisfactory results with a minimal effort and development time.

A case study using the proposed methodology is presented at the end of this document, discussing the difficulties encountered and showing the practical results obtained during its implementation.

As a main contribution of this work we can cite the evaluation of high-level synthesis for the development of video processing systems. The studies and obtained results indicates a remarkable progress of high-level synthesis tools currently available on the market, which already makes them capable of being employed in the development of complex systems, as video processing systems.

Additionally, the Catapult, the employed synthesis tool in the proposed flow, is now the most powerful synthesis tool on the market, being used by huge companies such as Panasonic, Siemens, STMicroelectronics, Nokia and Alcatel. Nevertheless, no Catapult licensing is currently assigned to Brazil. Thus, the presented work also contributes providing information and independent evaluations about its utility to the academy.

**Keywords:** Design Flow, High-level Synthesis, Video Processing, Rapid Prototyping, FPGA.

# Lista de Figuras

## Abreviações

| | |
|---|---|
| **AMS** | *Analog Mixed-Signal* |
| **ANSI** | *American National Standards Institute* |
| **ASIC** | *Application Specific Integrated Circuit* |
| **bps** | *bits* por segundo |
| **Bps** | *Bytes* por segundo |
| **CLB** | *Configurable Logic Block* |
| **CMOS** | *Complementary Metal-Oxide Semiconductor* |
| **DCT** | *Discrete Cosine Transform* |
| **DDR** | *Double-Data-Rate* |
| **DMA** | *Direct Memory Access* |
| **DMP** | *Digital Media Processor* |
| **DSL** | *Digital-system Specification Language* |
| **DSP** | *Digital Signal Processor* |
| **EDIF** | *Electronic Design Interchange Format* |
| **ESL** | *Electronic System Level* |
| **ESLD** | *Electronic System Level Design* |
| **EXU** | *EXecution Unit* |
| **FBF** | *FPGA Bit Files* |
| **fps** | *frames* por segundo |
| **FPGA** | *Field Programmable Gate Array* |
| **FPOA** | *Field Programmable Object Array* |
| **FW** | *Firmware* |
| **GOPS** | *Giga Operations Per Second* |
| **GPIO** | *General Purpose Input/Output* |
| **GPP** | *General Purpose Processors* |
| **GPS** | *Global Position System* |
| **GUI** | *Graphical User Interface* |
| **HDVICP** | *High Definition Video Imaging Co-Processor* |
| **HDL** | *Hardware Description Language* |

| | |
|---|---|
| **HL** | *High-Level* |
| **HLS** | *High Level Synthesis* |
| **HVD** | *Home Video and Display division* |
| **HW** | *Hardware* |
| **IDSP** | *Image Digital Signal Processor* |
| **IEEE** | *Institute of Electrical and Electronics Engineers* |
| **IOB** | *Input/Output Block* |
| **IRAM** | *Internal RAM* |
| **IP** | *Intellectual Property* |
| **JHDL** | *Java Description Language* |
| **JPEG** | *Joint Photographic Experts Group* |
| **LUT** | *Look Up Table* |
| **LVDS** | *Low-Voltage Differential Signaling* |
| **MAC** | *Multiply ACcumulate* |
| **MAX** | *Multimedia Acceleration eXtensions* |
| **MEMS** | *Micro-Electro-Mechanical Systems* |
| **mDDR** | *mobile DDR RAM* |
| **MJCP** | *MMPEG JPEG Co-Processor* |
| **MP** | *Media-Processor* |
| **MPEG** | *Moving Picture Experts Group* |
| **MPSoC** | *Multi-Processor System-on-Chip* |
| **MMP** | *Multimedia Processor* |
| **MMX** | *Multi-Media eXtensions* |
| **NoC** | *Network-on-Chip* |
| **OCP-IP** | *Open Core Protocol International Partnership* |
| **pps** | *pixels* per Segundo |
| **RF** | *Radio frequency* |
| **RTL** | *Register Transfer Level* |
| **SAD** | *Soma Absoluta das Diferenças* |
| **SD** | *Secure Digital* |
| **SDRAM** | *Synchronous Dynamic Random Access Memory* |

| | |
|---|---|
| **SDTV** | *Standard Definition Television* |
| **SIMD** | *Single Instruction Multiple Data* |
| **SoC** | *System on Chip* |
| **SoPC** | *System on a Programmable Chip* |
| **SSE** | *Streaming SIMD Extensions* |
| **SW** | *Software* |
| **STM** | STMicroelectronics |
| **TCAB** | *Tightly Coupled Accelerator Blocks* |
| **TCP/IP** | *Transmission Control Protocol/Internet Protocol* |
| **TLM** | *Transaction Level Modeling* |
| **ULA** | Unidade Lógica Aritmética |
| **UML** | *Unified Modeling Language* |
| **USB** | *Universal Serial BUS* |
| **Verilog-AMS** | *Verilog for Analog Mixed Signal* |
| **VHDL** | *VHSIC Hardware Description Language* |
| **VHDL-AMS** | *VHDL for Analog Mixed-Signal* |
| **VIS** | *Visual Instruction Set* |
| **VLD** | *Variable Length Decoder* |
| **VLSI** | *Very Large Scale Integration* |
| **VSP** | *Video Signal Processor* |
| **XRAM** | *eXternal RAM* |

# Definições

Alguns termos utilizados neste trabalho possuem significados diferentes, ou mesmo divergentes entre si, em diversas obras e documentos da área. A fim de melhorar a leitura do texto, evitando ambigüidades, são apresentadas a seguir algumas das definições adotadas neste trabalho:

**Abstração:** descrição de um objeto usando um modelo no qual parte dos detalhes de baixo nível é ignorada.

**Agendamento:** sincronização e ordenamento de tarefas funcionais.

**Arquitetura:** a forma como aspectos de um projeto se relacionam entre si estrutural ou funcionalmente.

**Arquitetura Heterogênea:** arquitetura composta por componentes físicos (em circuito) e lógicos (em programa), na qual os componentes lógicos operam por meio de um ou mais processadores embarcados conectados aos componentes físicos. Referida em inglês por *hardware/software architecture*.

**Código Estaticamente Determinável:** código cujas propriedades de execução podem ser totalmente determinadas no momento de sua compilação.

**Dispositivos de Lógica Reconfigurável:** dispositivos capazes de implementar circuitos lógicos de forma reconfigurável mediante a associação por interconexão de unidades lógicas básicas;

**Dispositivos de Lógica Processada:** dispositivos cujo núcleo é composto por um ou mais microprocessadores;

**Dispositivos de Lógica Mista:** dispositivos cujo núcleo é composto por dispositivos de lógica reconfigurável e processada;

**Exploração da Arquitetura:** a exploração de múltiplas arquiteturas de implementação possíveis na busca por encontrar uma que melhor corresponda aos requisitos pré-estabelecidos.

**Exploração da Micro-Arquitetura:** a exploração de múltiplas possíveis micro-arquiteturas de implementação na busca por encontrar uma que melhor corresponda aos requisitos pré-estabelecidos.

**Modelo Funcional:** o modelo de um sistema em qualquer nível de abstração que não inclua informação de tempo.

**Propriedade Intelectual (IP):** sistema ou subsistema pré-concebido, em qualquer nível de abstração, incluindo componentes de circuito e programa, caso existam, que pode ser reusado em vários projetos ou partes de um projeto.

**Projeto Baseado em Plataforma:** estilo de projeto em que grandes parcelas do sistema são projetadas em função de características especificas e pré-determinadas de uma plataforma.

**Projeto Heterogêneo:** abordagem de projeto no qual parte das funcionalidades do sistema é implementada por circuitos eletrônicos dedicados e parte por programas contidos em um ou mais microprocessadores embarcados, sendo o desenvolvimento de ambas as partes (circuito e programa), realizado paralelamente (referido no inglês por *hardware/software co-design*).

**Prototipagem Rápida:** implementação do protótipo funcional de um sistema eletrônico em tempo significativamente inferior ao necessário para produção do ASIC equivalente, através do uso de dispositivos alternativos.

**Protótipo:** produto que ainda não foi comercializado, mas que foi concluído, podendo ainda estar em fase de testes ou de aprimoramento.

**Recursão:** execução de uma função por invocação da própria função.

**Síntese de Alto Nível:** também é conhecida como síntese comportamental ou síntese algorítmica, é o processo de conversão de uma especificação em nível algorítmico (comportamental) em uma descrição de circuito eletrônico para prototipagem VLSI.

**Validação**: Confirmação por examinação e provisão de evidências objetivas de que os requisitos particulares para uma intenção específica de uso foram satisfeitos (definição do IEEE [1]).

**Verificação**: Confirmação por examinação e provisão de evidências objetivas de que requisitos específicos foram satisfeitos (definição do IEEE [1]).

# Sumário

# 1. Introdução

Assim como a complexidade dos sistemas digitais tem avançado continuamente ao longo dos anos, a dos dispositivos de lógica programável tem seguido o mesmo ritmo. Os FPGAs contêm atualmente milhões de portas e suas freqüências de operação podem chegar a 600 MHz [2][3], o que os torna capazes de serem utilizados até mesmo na codificação e decodificação de vídeo em tempo real [4][5], tarefa que até pouco tempo atrás era exclusivamente executada por circuitos integrados dedicados. Todos esses fatores, aliados ao reduzido tempo de produção (relativamente ao tempo para produzir um ASIC equivalente), tornam a tecnologia FPGA muito atrativa para prototipagem de sistemas VLSI, auxiliando projetistas a melhor ajustarem e validarem soluções antes da sua fabricação em ASIC, ou mesmo, servindo como solução final [6].

Ferramentas poderosas de projeto em alto nível, que permitam a transição de projetos baseados em código RTL para projetos baseados em código C, se fazem hoje necessárias, de sorte a acompanhar a crescente complexidade dos projetos de sistemas digitais [7]. Elas representam uma mudança no paradigma de concepção de SoCs e são apontadas como a melhor solução para superar o atual problema de produtividade na fase de projeto de novos produtos [8][9].

A distância entre a produtividade por engenheiro por ano e o aumento constante da complexidade dos circuitos integrados, mesmo levando-se em conta valores conservativos de portas por tecnologia, levam a uma expansão do número de engenheiros necessários nas tecnologias vindouras [10], conforme pode ser visto na Tabela 1 [11].

Por outro lado, o aumento da complexidade dos circuitos integrados também tem solicitado equipes de projetistas cada vez maiores no seu desenvolvimento, aumentando cada vez mais a complexidade da logística de projeto, bem como a necessidade de se encontrar novas soluções que melhor se adequem a essa nova realidade. Atualmente, o desenvolvimento de um SoC na indústria, envolve, tipicamente, dezenas de equipes em diferentes especialidades, englobando mais de uma centena de engenheiros [10].

**Tabela 1 - Número de engenheiros necessários para projetar circuitos VLSI empregando as novas tecnologias de micro-fabricação**

| Ano | Tecnologia | Portas/Chip (50 mm2) | Portas/Engenheiro/Ano | Homens-ano por chip |
|---|---|---|---|---|
| 1991 | 0,7 µm | 50 k | 4 k | ~10 |
| 1994 | 0,5 µm | 250 k | 6 k | ~40 |
| 1996 | 0,35 µm | 750 k | 9 k | ~80 |
| 1998 | 0,25 µm | 1,5 M | 40 k | ~40 |
| 2000 | 0,18 µm | 2,2 M | 56 k | ~40 |
| 2002 | 0,13 µm | 4 M | 91 k | ~43 |
| 2004 | 90 nm | 7,5 M | 125 k | ~60 |
| 2006 | 65 nm | 15 M | 200 k | ~75 |

Toda essa estrutura, necessária a projetos de grande envergadura, eleva a confiabilidade na sua execução e coerência, mas cria, em contrapartida, um tortuoso caminho a ser percorrido entre o modelo do sistema (usualmente um algoritmo em C++) e sua prototipagem final [12].

Com tudo isso, torna-se clara a necessidade de um novo avanço no aumento da produtividade. Parte desse avanço pode estar nas soluções de Projeto em Nível de Sistema Eletrônico (ESLD - *Electronics System Level Design*), tais como ferramentas de projeto heterogêneo (*HW/SW co-design*) e de Síntese de Alto Nível. A larga difusão projetos baseados em arquiteturas heterogêneas ocorreu alguns anos atrás, em boa parte graças ao SystemC e a Modelagem em Nível de Transações (TLM) [13][14]. A síntese de alto nível, no entanto, apenas recentemente alcançou uma difusão significativa na indústria [8].

No restante deste capítulo será apresentada uma breve revisão sobre linguagens de descrição de circuitos que são referenciadas ao longo deste trabalho. No capítulo 2, é feito um estudo sobre as bases da prototipagem rápida, citando as principais soluções aplicáveis ao processamento de vídeo digital, bem como uma análise sobre as metodologias de projeto de sistemas VLSI, citando importantes conceitos atualmente na área. Um estudo do estado-da-arte da síntese de alto nível, pilar fundamental deste trabalho, é apresentado no capítulo 3. Adicionalmente, um resumo sobre trabalhos cujos objetivos se assemelham a esta proposta, também é realizada. No capítulo 4, o fluxo de projeto proposto é apresentado, detalhando regras e características de sua implementação. Um estudo de caso do fluxo proposto é então relatado no capítulo 5, apresentando resultados e comparando-os aos obtidos através do fluxo RTL clássico. Por fim, o capítulo 6 contém as conclusões e perspectivas futuras do trabalho desenvolvido.

## 1.1. Linguagens de Descrição de Circuitos

Uma linguagem de descrição de circuitos (HDL – *Hardware Description Language*) é uma linguagem estruturada voltada para a descrição formal dos circuitos eletrônicos. Ela é capaz de descrever a operação do circuito, sua organização, detalhes de concepção, bem como os testes para verificar a sua operação por meio de simulações [15]. Trata-se de uma listagem textual, formal, baseada em padrões estruturais, temporais, comportamentais de um sistema eletrônico. Ao contrário de uma linguagem de programação de *software*, a sintaxe e a semântica de uma HDL incluem notações explícitas para exprimir o tempo e o paralelismo, atributos primários de um circuito eletrônico. Linguagens, cujo único objetivo é o de exprimir as conexões do circuito em uma hierarquia dos blocos, são denominadas linguagens de *Netlist*.

HDLs são empregadas para descrever características implementáveis de uma determinada parte de um circuito. Um programa de simulação interpreta a semântica fundamental descrita e simula o sistema, levando em conta o andamento do tempo no circuito [16][17][18][19].

Nos simuladores dedicados a HDLs puramente digitais, o tempo avança de maneira discreta. Geralmente, a cada transição do relógio do sistema e a cada avanço, o nível lógico de cada um dos nós do circuito é recalculado com base na álgebra booleana. Já nos simuladores concebidos para HDLs que comportam circuitos analógicos ou mistos, o avanço do tempo na simulação é baseado em parâmetros informados pelo projetista. Este avanço ainda ocorre de maneira discreta, porém, com intervalos curtos o suficiente para que a simulação se assemelhe a um evento em tempo contínuo, calculando para cada nó das partes analógicas do circuito, os valores de tensão e corrente através da teoria de circuitos. As partes digitais, se existirem, podem ser calculadas com base na álgebra booleana (simuladores mistos ou AMS), ou também pela teoria de circuitos. Esta última solução, embora mais exata, costuma envolver tempos de simulação bastante elevados, sendo, portanto, menos empregada na indústria.

Desde o nascimento deste conceito de modelagem, várias linguagens de descrição de circuitos já foram criadas [15][20][21]. Dentre elas destacam-se por sua relevância:

- Verilog;
- VHDL;
- System-C;
- SystemVerilog;
- Spice;
- VHDL-AMS;
- Verilog-AMS.

As quatro primeiras são dedicadas exclusivamente à modelagem de sistemas digitais, enquanto as três últimas são capazes de descrever também sistemas analógicos [20][21].

Contudo, com o advento da síntese de alto nível, linguagens antes exclusivamente utilizadas para a criação de aplicativos, passaram também a poderem ser empregadas na descrição de circuitos eletrônicos, conforme é abordado em detalhes no capítulo 3.

# 2. Prototipagem Rápida de Sistemas de Vídeo Digital

## 2.1. Introdução

O termo "prototipagem rápida", em microeletrônica, é comumente associado a plataformas baseadas em FPGA. No entanto, conceitualmente, a definição adotada para este termo se estende também a outros dispositivos capazes de implementar o sistema em questão em um intervalo reduzido de tempo, relativamente ao tempo consumido pela prototipagem em silício.

Conceitualmente, a prototipagem rápida de sistemas de vídeo digital difere da prototipagem tradicional essencialmente por dois aspectos básicos:

- A plataforma de prototipagem;
- A metodologia de projeto.

Na prototipagem tradicional a plataforma consiste usualmente em um circuito integrado de aplicação específica (ASIC) [22]. A prototipagem desse componente envolve custos elevados e um complexo processo de fabricação que se estende por vários meses [23]. Adicionalmente, o fluxo de projeto de um ASIC também é mais demorado que o equivalente para prototipagem rápida, por envolver etapas adicionais (tais como a inclusão de subsistemas de teste [24] e o leiaute do silício [25]) e processos de verificação mais exigentes e extensos [26]. Em contrapartida, uma vez fabricado, o dispositivo é capaz de operar em freqüências superiores às das plataformas de prototipagem rápida e consumindo, geralmente, menos energia estática e dinâmica [27]. Além disso, seu custo unitário é significativamente menor quando produzido em larga escala [28][29].

Na prototipagem rápida o ASIC é substituído por um dispositivo alternativo capaz de desempenhar o mesmo papel, porém com custo e tempo de prototipagem reduzidos. De acordo com o tipo desse dispositivo, as plataformas existentes para prototipagem rápida de vídeo podem ser classificadas em três categorias, a saber:

- Plataformas de lógica reconfigurável - têm como núcleo um ou mais dispositivos capazes de implementar circuitos lógicos de forma reconfigurável mediante a associação por interconexão de unidades lógicas básicas;
- Plataformas de lógica processada - têm como núcleo um ou mais microprocessadores;
- Plataformas de estrutura mista - compostas por dispositivos de lógica reconfigurável e processada.

Um ponto importante, comum entre todas as categorias, é a sua capacidade de reutilização, ou seja, a mesma plataforma pode ser utilizada várias vezes na prototipagem de um ou mais projetos de maneira independente. Esta característica reduz drasticamente os custos de prototipagem do projeto em relação a um ASIC, ao mesmo tempo em que permite que parte da verificação do sistema seja feita através da própria plataforma, pois, de maneira geral, opera mais rápido do que simulações equivalentes em computador [30]. Alguns fabricantes oferecem também versões não reutilizáveis de seus dispositivos, configuráveis ou programáveis uma única vez. Estas, porém, não são comumente empregadas durante a fase de desenvolvimento do sistema.

A ausência de algumas etapas de projeto dos ASICs e a redução do tempo de verificação também tornam o fluxo de projeto de prototipagem rápida mais curto do que o equivalente para ASIC [31][26]. Uma das desvantagens das plataformas de prototipagem rápida em relação aos ASICs reside no fato de que estas, por não serem especificamente desenhadas para o sistema prototipado, freqüentemente necessitam de componentes complementares externos, que estariam embarcados no ASIC equivalente, aumentando assim a complexidade do projeto. Podem também ser subutilizadas, quando oferecem mais recursos de que o sistema necessita. A não especificidade da plataforma e a sua natureza reconfigurável (ou programável) também impõem limitações a sua freqüência de operação, que, por conseguinte, não atinge os mesmos valores que os alcançados pelo homólogo ASIC. Além disso, seu custo unitário também é, de maneira geral, superior ao dos ASICs, quando produzidos em larga escala [28][29].

Uma revisão sobre as principais plataformas e metodologias de projeto para prototipagem rápida, destinadas ou aplicáveis ao processamento de vídeo, são discutidas em detalhes nas seções 2.2 e 2.3, respectivamente.

## 2.2. Plataformas de Prototipagem Rápida para Sistemas de Processamento de Vídeo Digital

Ao longo dos últimos anos, um vasto número de soluções têm sido propostas visando ou prestando-se à prototipagem rápida de sistemas de processamento de vídeo.

No inicio da década de 90, as principais plataformas de prototipagem rápida existentes, isto é, processadores, DSPs e FPGAs, não dispunham de desempenho suficientemente satisfatório para a maioria das aplicações de vídeo em tempo real [32]. Por essa razão, grande parte das soluções propostas envolvia a concepção de dispositivos dedicados, com arquitetura e características apropriadas à natureza e exigências desse tipo de processamento. Tais soluções requeriam muitas vezes que seus idealizadores tivessem que conceber suas próprias ferramentas de desenvolvimento e suporte [33].

Nos últimos anos da década de noventa, o aumento do desempenho dos DSPs e FPGAs fez com que muitos começassem a adotá-los para tais aplicações, ainda que sob restrições modestas de resolução e cadência de vídeo. Houve uma redução gradual na preocupação em se propor novos dispositivos de prototipagem e, em contrapartida, um aumento no interesse de se propor novas arquiteturas empregando as plataformas de prototipagem existentes, capazes de implementar de maneira rápida e eficiente os sistemas de processamento de vídeo em desenvolvimento.

Nesta seção são apresentadas algumas das principais soluções existentes, assim como os trabalhos que de alguma forma tenham contribuído significativamente para o desenvolvimento das soluções atuais.

### 2.2.1. Plataformas Baseadas em Processadores

Considerando as principais arquiteturas de processadores atualmente existentes aplicáveis ao processamento de vídeo, três principais categorias podem ser definidas:

- processadores de propósito geral;
- processadores digitais de sinais (DSPs);
- processadores de mídia.

É importante salientar que esta revisão resume-se a dispositivos destinados, ou diretamente aplicáveis, à prototipagem rápida de sistemas de processamento de vídeo. Por esta razão, apenas soluções reconfiguráveis e/ou programáveis são consideradas neste estudo.

**GPP** - *General Purpose Processors* (Processadores de Propósito Geral) – entende-se por processadores de propósito geral aqueles projetados para uma grande variedade de tarefas computacionais [34]. Sua maior utilização se dá atualmente em computadores pessoais, dos quais a Intel e a AMD são os principais líderes de mercado [35]. O desempenho desses processadores, durante muito tempo, não era adequado para que fossem utilizados na prototipagem de sistemas de processamento de vídeo [36]. Por esta razão, a partir de meados da década de noventa, muitos processadores passaram a incluir em sua arquitetura estruturas de suporte ao processamento de multimídia comandadas por instruções específicas adicionadas ao seu conjunto de instruções [37]. Os processadores PA-RISC da HP foram os pioneiros, introduzindo a extensão MAX-1 (*Multimedia Acceeration eXtensions*), em janeiro de 1994 [38]. Em seguida, a Sun adicionou o VIS (*Visual Instruction Set - Conjunto de Instruções Visuais*) aos seus processadores Sparc [39], e em janeiro de 1997, a Intel passou a acrescentar a extensão MMX (*Multi-Media eXtensions*) aos seus processadores ix86 [40].

Atualmente, a maioria dos processadores dispõe de instruções dedicadas ao processamento de multimídia. Os modernos processadores da Intel e da AMD, por exemplo, implementam a tecnologia SSE4 (*Streaming SIMD Extensions 4*), uma evolução da tecnologia MMX, capaz de realizar operações de multimídia com elevado desempenho, além de possuir recursos de aceleração de processamento de textos [41]. Aplicações de processamento de vídeo envolvendo processadores de propósito geral com as recentes extensões SSE podem ser encontradas em [42][43][44].

**PDSP / DSP** – *Programmable Digital Signal Processors* (Processador Digital de Sinais Programável) ou simplesmente DSPs - são processadores projetados especificamente para o processamento digital de sinais. Foram introduzidos no mercado no início dos anos 80, pela NEC, Texas Instruments e AT&T [45]. Inicialmente, os DSPs distinguiam-se dos processadores de propósito geral apenas por duas características [46]: sua elevada capacidade de processamento aritmético e por sua elevada taxa de acesso à memória. Com o advento de novas aplicações, notadamente os sistemas de processamento de multimídia [47], tanto os DSPs quanto os processadores de propósito geral, buscaram aprimorar seus recursos de forma a tornarem-se aplicáveis a estas novas aplicações. Os DSPs passaram a adotar novas arquiteturas como paralelismo VLIW [48] e organização de memória *cache* hierárquica, enquanto os processadores de propósito geral adotaram tecnologias como paralelismo em nível de instruções (*superscalar*) e paralelismo em nível de dados (tecnologias SIMD ou MMX [49]). Conseqüentemente, os DSPs e os processadores de propósito geral tornaram-se muito semelhantes uns aos outros. Um DSP VLIW, recentemente proposto por Agarwala et al. [50], ilustra bem este ponto. O digrama simplificado de um DSP é apresentado na Figura 1 [51].

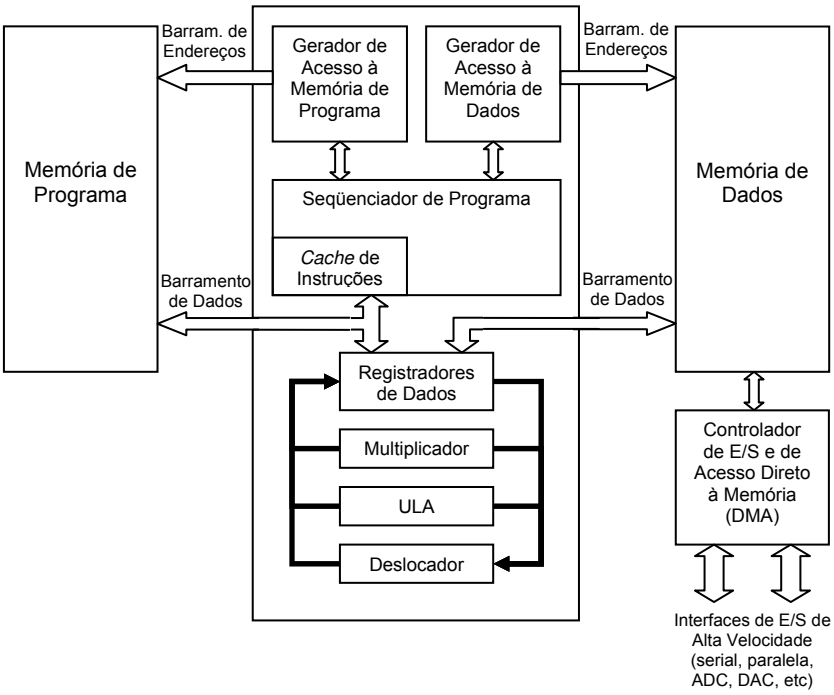


**Figura 1 - Digrama simplificado de um DSP [51].**

Huang et al. [52] e Normile et al. [53] foram uns dos primeiros a empregar os DSPs comerciais como plataforma de prototipagem rápida de sistemas de processamento de vídeo. Entretanto, devido ao baixo desempenho desses componentes na época, mesmo aplicações de resolução e cadência de vídeo limitadas exigiam o emprego de múltiplos DSPs em paralelo [52][53]. Um exemplo do uso pioneiro de DSPs na prototipagem de um codificador de vídeo pode ser encontrado em [54]. Para este projeto, 12 DSPs TMS320C30 da *Texas Instruments* e CIs codificadores DCT IMSA121 foram utilizados para implementar um codificador H.261 [55] com resolução de 352x288 *pixels* à 15 quadros por segundo, sem estimação de movimento. Atualmente, vários DSPs comerciais oferecem desempenho suficiente para que possam ser utilizados singularmente em aplicações de processamento de vídeo em tempo real [56][57][58].

**MP** – *Media Processors* (Processadores de Mídia) – também referidos por MMP (*Multi-Media Processors* – Processadores de Multimidia) ou por DMP (*Digital Media Processors* Processadores Digitais de Mídia) - são DSPs de propósito específico. Geralmente utilizam arquiteturas VLIW e SIMD, permitindo o paralelismo em nível de instruções e dados [59][60]. Também oferecem suporte ao controle de diversas funções internas a partir de uma única instrução e operações para pacotes de dados, o que pode ser eficientemente utilizado no processamento de imagem [42]. Podem conter ainda estruturas específicas às aplicações de multimídia como: redimensionamento de imagem; decodificadores de códigos de comprimento variável (VLD); ou mesmo um co-processador de vídeo de alta definição (HDVICP) e de decodificação de padrões MPEG e JPEG (MJCP - Mpeg Jpeg Co-Processor), como no caso dos processadores de mídia da série TMS320DM36x [61] da Texas Instruments. Dispõem, de maneira geral, de interfaces de alta velocidade para memórias SDRAM, DDR, mDDR ou Flash [62] e suporte a DMA, de forma a comportar as intensas operações de acesso à memória, tipicamente presentes em sistemas de processamento de vídeo [42][63][64]. Também incluem interfaces para áudio, vídeo, interfaces de E/S convencionais e até mesmo interfaces de rede, USB e de suporte a meios de armazenamento de mídia, tais como cartões SD ou Flash, por exemplo. O TMS320DM365 [65] da Texas Instruments, ilustra bem esta categoria de processadores, oferecendo desempenho suficiente para implementar um decodificador H.264 [66] em alta

resolução (1920x1080 *pixels*) à 10 quadros por segundo [67]. Seu diagrama de blocos é apresentado na Figura 2 [65].

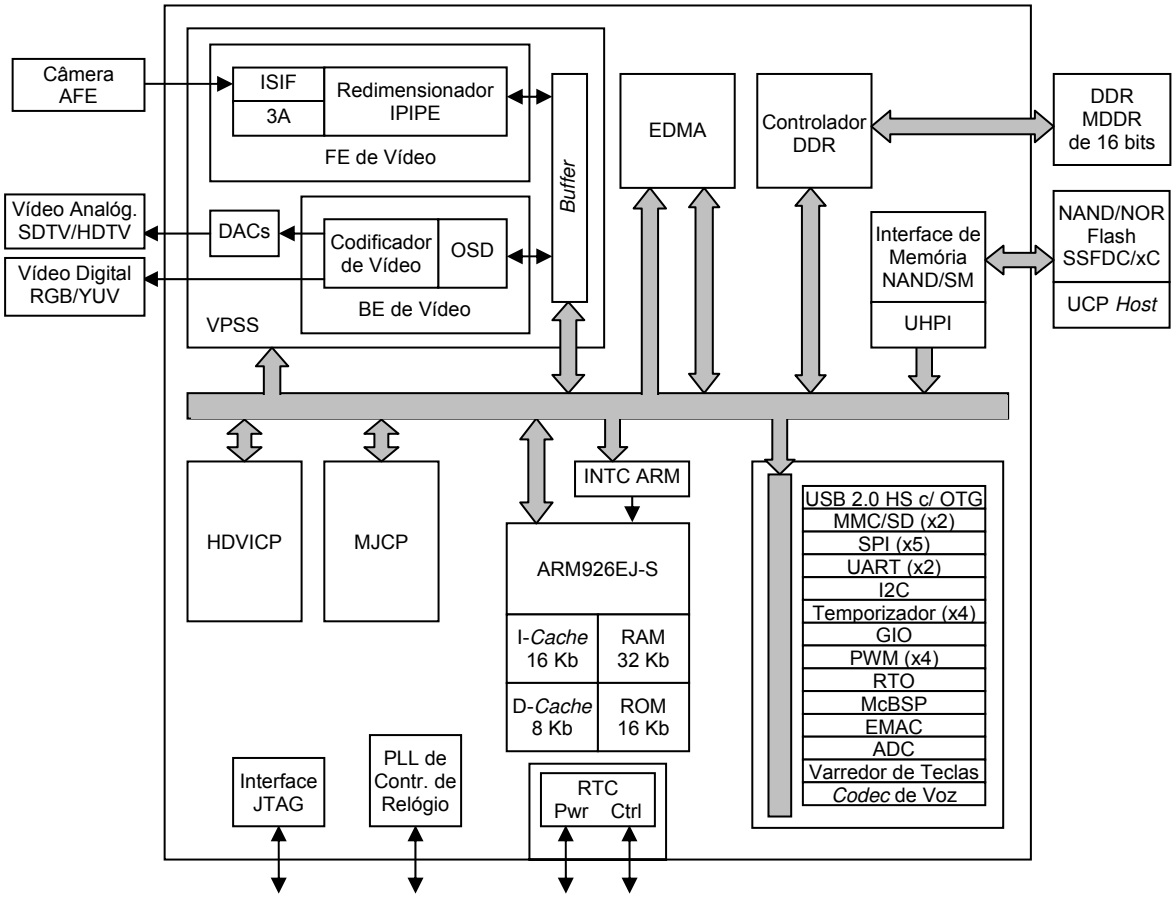


**Figura 2 - Digrama de blocos do processador de mídia TMS320DM365 [65].**

## Outras Arquiteturas Relevantes

**VSP** – *Vídeo Signal Processor* (Processador de Sinais de Vídeo) – também referido por IDSP [68] (*Image Digital Signal Processor* - Processador Digital de Sinais de Imagem). Trata-se de um DSP de propósito específico, precursor dos processadores de mídia (MP). São voltados ao processamento de sinais vídeo. Esta classe de DSPs geralmente incorpora conceitos de paralelismo em suas arquiteturas, tais como VLIW, e são otimizados para executar tarefas de processamento de imagem. Ainda na década de noventa, VSPs com desempenho de até 1,5 bilhões de operações por segundo (GOPS) foram relatados [69][70][71][72]. O MC149570 [73], fabricado pela Freescale, é um exemplo comercial de VSP.

**PADDI** [74][32] - uma solução objetivando a prototipagem em uma arquitetura programável, multi-processada, dedicada ao processamento de imagem e vídeo, proposta por Chen [75]. A arquitetura alvo, desenvolvida com base na análise estatística dos requerimentos de sistemas de processamento de imagem, vídeo e voz [76], era programada através de uma linguagem de fluxo de dados de alto nível, chamada Silage [77], com as operações do sistema de vídeo projetado e, então, sintetizada e mapeada em FPGA.

Apesar de empregar arquitetura, linguagem e ferramentas de projeto dedicadas, muitas das idéias contidas nesta abordagem foram posteriormente adotadas no desenvolvimento de outras soluções [33], dentre as quais a versão aprimorada da solução, PADDI-2, proposta por Yeung [78].

**Plêiades** [79] – foi uma nova arquitetura voltada para a prototipagem rápida de aplicações de multimídia proposta por Rabaey, com base nas mesmas idéias que guiaram a concepção das arquiteturas PADDI e PADDI-2. Ela era composta por uma matriz reconfigurável heterogênea de matrizes reconfiguráveis de unidades de execução (EXUs – EXecution Units).

**REMARC** [80], um co-processador multimídia reconfigurável acoplado a um processador RISC principal, é composto por uma unidade de controle central acoplada a uma memória e 64 unidades lógicas, de 16 bits, chamadas nanoprocessadores. Um ambiente de programação dedicado foi desenvolvido, integrando um compilador assembler e um compilador C, com suporte a programação paralela. Tendo como principais aplicações alvo a compressão e a descompressão de vídeo e processamento de imagens, a eficiência da arquitetura foi comprovada com o desenvolvimento de um codificador e decodificador MPEG-2 que foram capazes de operar em tempo real a 30 quadros por segundo a uma resolução de 352x224 *pixels*.

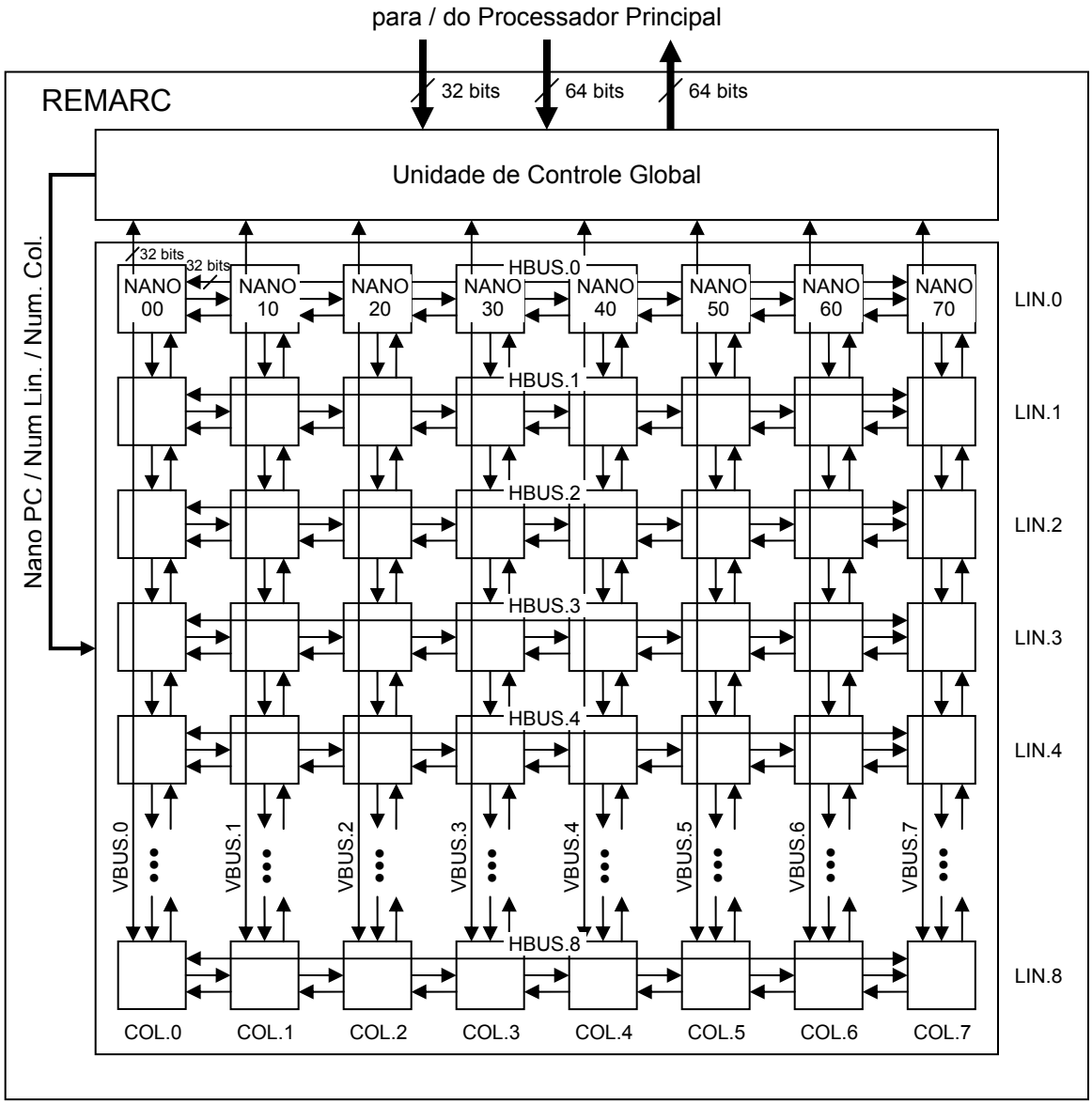


**Figura 3 - O co-processador multimida REMARC**

Outros dispositivos de arquitetura reconfigurável de propósito geral também foram propostos por Hartenstein [81][82] e Mirsky [83].

**MPSoC** [84], não se trata exatamente de uma arquitetura, mas um conceito de arquitetura. Baseando-se no conceito de SoC e em arquiteturas multiprocessadas, o MPSoC (*Multi-Processor System-on-Chip* - Sistema Multiprocessado em Chip), é basicamente um SoC composto por múltiplos processadores, sendo estes geralmente voltados para aplicações embarcadas. Por envolver conceitos de multiprocessamento, tal arquitetura

conduz a um aumento ainda maior da complexidade já existente nos SoCs. Dessa forma, a concepção de MPSoCs traz consigo grandes desafios aos projetistas, tais como problemas de comunicação e consumo [85]. Soluções para a redução do consumo de energia apontam que este problema pode ser tratado em nível de dispositivo [86], de comunicação [87][88][89] e de programa [90][91]. O ponto mais crítico, no entanto, é apontado como sendo a comunicação entre os módulos, principalmente processadores [92]. Por esta razão, diversas arquiteturas de comunicação dedicadas foram propostas [93], sendo a Rede em Chip (NoC – *Network-on-Chip*) a que tem recebido maior aceitação por apresentar maior flexibilidade (em relação as topologias de conexão) e escalabilidade [94]. A principal desvantagem desse conceito é que a complexidade do sistema faz com que cada MPSoC necessite de seu próprio conjunto de ferramentas (compiladores, simuladores, etc.), além de tornar seu processo de teste e verificação bem mais complexos, demorados e caros. Uma solução para este problema parece estar no reuso de projetos, através da abordagem de projeto baseado em plataforma [95].

### 2.2.2. Plataformas Baseadas em Dispositivos de Lógica Reconfigurável e Dispositivos Mistos

**FPGA** (*Field Programmable Gate Array* – Arranjo de Portas Programável em Campo) - foi um conceito totalmente novo de componente eletrônico cuja criação é atribuída a Ross Freeman (co-fundador da Xilinx Inc.), em 1984 [96], tornando-se comercial dois anos depois [97]. Inicialmente, o dispositivo era composto basicamente por três componentes fundamentais: blocos de entrada e saída (IOB – *Input/Output Block*), blocos lógicos configuráveis (CLB – *Configurable Logic Block*) e matrizes de interconexão [98]. Cada CLB pode ser programado de forma a implementar pequenos circuitos lógicos e cada IOB é conectado a um terminal externo do FPGA, podendo se comportar como porta de entrada, saída, bidirecional ou mesmo como um pino não conectado (alta impedância). As matrizes de interconexão são capazes de conectar CLB's e IOB's formando então o sistema completo. O diagrama simplificado de uma FPGA e seus componentes básicos é mostrado na Figura 4 [99].

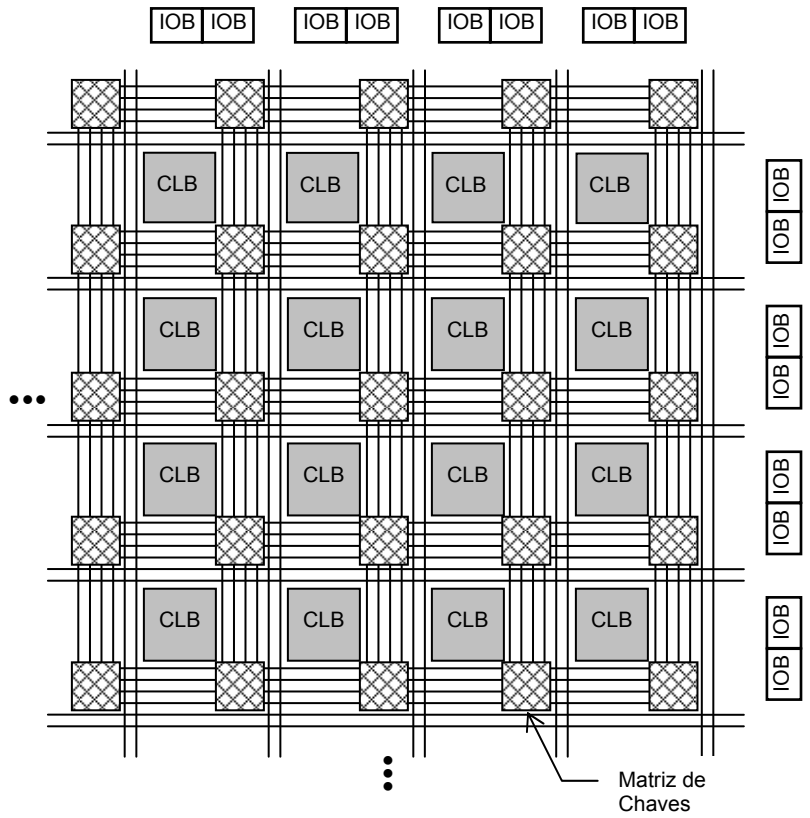


**Figura 4 - Estrutura básica de um FPGA: blocos de lógica configurável, blocos de entrada e saída e matrizes de interconexão [99].**

A estrutura do FPGA e a característica reconfigurável de todos os seus componentes básicos o tornam capaz de implementar literalmente qualquer sistema digital (respeitando-se a quantidade disponível de recursos lógicos do componente). De fato, após ser configurado, um FPGA comporta-se exatamente como um circuito integrado de aplicação específica (ASIC) [100], podendo ser então utilizado em substituição a este, como plataforma de prototipagem rápida ou simplesmente como plataforma de validação do ASIC a ser fabricado. Estas características, aliadas à sua rápida comercialização no mercado [97], fizeram com que os FPGAs fossem rapidamente adotados pela indústria, consolidando-se como plataforma de desenvolvimento padrão no desenvolvimento de ASICs [101]. Atualmente, a Xilinx e a Altera são os líderes de mercado em FPGAs de granularidade fina baseadas em SRAM. Outros fabricantes oferecem soluções alternativas tais como FPGAs baseados em memória não volátil (Lattice Semiconductors e Actel) [102][103], com microcontroladores embarcados (Atmel) [104], voltados para aplicações portáteis (QuickLogic e Actel) [105][106], e de alto desempenho (Achronix). A Achronix é

atualmente o fabricante dos FPGAs de maior desempenho do mundo, atingindo velocidades de até 1.5GHz [107].

**FPOA** – *Field Programmable Object Array* (Arranjo de Objetos Programável em Campo) - são dispositivos de lógica reconfigurável criados pela MathStar™ objetivando acelerar a prototipagem de sistemas de alto desempenho. Ao contrário dos FPGAs, cujos blocos de construção elementares estão no nível de portas lógicas, os blocos de construção elementares nos FPOAs são estruturas de complexidade mais elevada, chamados "objetos". Isto dá aos FPOAs a capacidade de poderem implementar sistemas utilizando um nível de abstração mais elevado em sua programação. FPOAs atuais são compostas por 400 objetos reconfiguráveis, sendo 380 em seu núcleo e 20 na periferia.

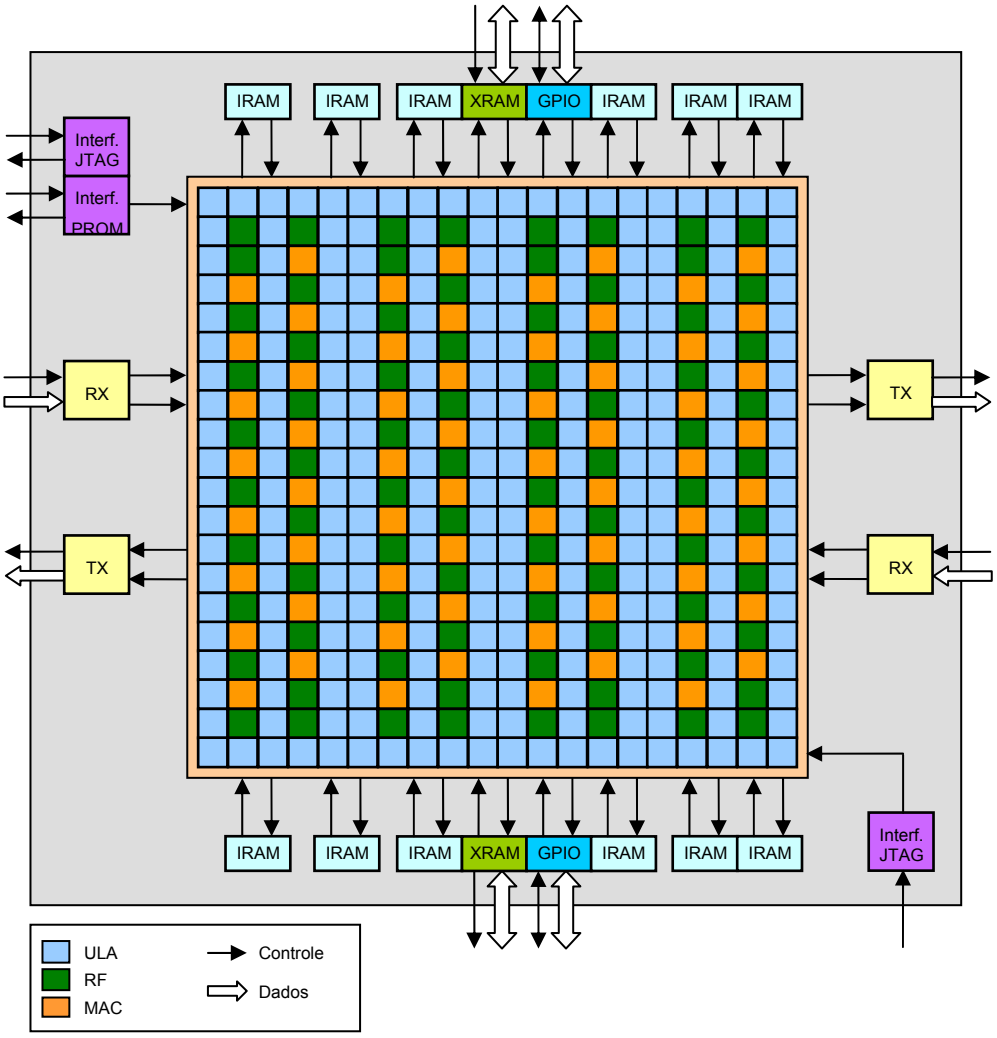


**Figura 5 - Diagrama de um FPOA [108].**

Seu núcleo reúne 256 ULAs, 64 multiplicadores acumuladores (MAC – *Multiply ACcumulator*) e 80 bancos de registradores (RF – *Register File*); enquanto a periferia é composta por 12 blocos de memória RAM de uso interna (IRAM – *Internal RAM*), 2 blocos de memória RAM como buffer de memória externa (XRAM – *eXternal RAM*), 2 portas bidirecionais de propósito geral (GPIO – *General Purpose Input/Output*) e 4 interfaces LVDS, sendo duas de transmissão (TX) e duas de recepção (RX). O diagrama simplificado de um FPOA é apresentado na Figura 5 [108].

Os objetos dos FPOAs têm natureza semi-autônoma, isto é, cada ULA, por exemplo, possui uma memória de programa de oito instruções que podem conter tanto instruções de operações aritméticas quanto de comunicação. Uma estrutura de controle, manipulável em nível de bit, interconecta os objetos e guia a execução do programa enquanto os dados são transferidos através de uma estrutura proprietária de interconexão que interliga todos os objetos. As principais características dessa estrutura de interconexão são a transferência sempre síncrona dos dados e a freqüência operação determinística, podendo chegar a 1 GHz no modelo MOA2400D-10 [109]. Esta é a característica mais importante do FPOA em relação a outras arquiteturas reconfiguráveis, pois elimina problemas de projeto como "condição de corrida" dispensando completamente a necessidade de ferramentas de "encerramento temporal" (*time closure*), tipicamente presentes nos fluxos de projeto de ASICs e FPGAs. Além disso, objetos não utilizados podem ser desligados reduzindo o consumo do componente. Outro ponto interessante é que a migração de um sistema de um dado FPOA para outro mais recente, ou de maior capacidade, não requer re-projeto, uma vez que os objetos que serão arranjados serão sempre um subconjunto dos objetos do novo dispositivo.

As principais ferramentas de suporte a projeto com FPOA são atualmente o COAST da própria MathStar e o Visual Elite™ da Mentor Graphics, aceitando descrições em SystemC ou em diagrama esquemático [110].

Sistemas de processamento de vídeo prototipados em FPOAs podem ser encontrados em [111][112][113]. Em [112] Riley e Moll descrevem o projeto de decodificadores de vídeo MPEG-2 e H.264 utilizando FPOAs comerciais.

As principais desvantagens dessa arquitetura são as fases de projeto, simulação e mapeamento para o dispositivo que são relativamente longas [114]. O número de ferramentas de desenvolvimento e suporte ao dispositivo também é bastante limitado. A ferramenta de mapeamento para o dispositivo segue um processo manual, o que pode consumir um tempo significativo. Por se tratar de um dispositivo de granularidade média, o FPOA não oferece o mesmo grau de flexibilidade de um FPGA. Em termos de estrutura, a memória embarcada no componente é bem reduzida, o que praticamente impõe o uso de memórias complementares externas. Além disso, a quantidade total de pinos de interfaceamento é limitada a 128 (96 pinos de entrada e saída de propósito geral e 32 pinos de interface LVDS, excluindo-se as portas de interface com memórias externas), o que pode ser bastante restritivo para muitas aplicações.

**SoPC** [115] - Com o advento dos DSPs e processadores embarcados nos FPGAs, ocorrido no final dos anos noventa [116], nova classe de arquitetura para prototipagem rápida surgiu e ganhou popularidade na indústria, a arquitetura SoPC. O Sistema em Chip Programável (SoPC - *System-On-a-Programmable-Chip*) é um conceito que designa um sistema embarcado completo em um dispositivo de lógica programável do tipo FPGA, podendo conter memória, um ou mais processadores, periféricos de interface analógica e demais componentes necessários à aplicação desejada [117]. Todos esses dispositivos se conectam geralmente por um barramento ou sistema de chaveamento, comumente controlado por um processador embarcado [118][119]. Um exemplo de SoPC completo é o sistema de piloto automático para aeronaves, desenvolvido por Christophersen et al. [120], apresentado na Figura 6 [31].


**Figura 6 – Sistema de piloto automático de avião baseado em SoPC. O lado superior da placa contém um FPGA com processador NIOS embarcado, um processador DSP e memória. O lado inferior contém um receptor GPS, um conversor A/D, giroscópios e acelerômetros nos três eixos, um sensor de velocidade do ar e um sensor de altitude [31].**

## Outras Arquiteturas Relevantes

**CHESS** [121] - desenvolvida por Marshall et al. nos laboratórios da HP, era uma matriz reconfigurável composta por ULAs (Unidades Lógicas Aritméticas) e blocos de chaveamento, alternadamente, na forma de um tabuleiro de xadrez, conforme a Figura 7 [33]. Além disso, a matriz também é permeada por colunas de memória RAM embarcada, de forma a suportar aplicações com elevados requisitos de memória, tal como aplicações de vídeo. Os blocos de chaveamento também foram projetados para poderem ser convertidos em memórias RAM de 16x4 bits ou em LUT's de quatro entradas e quatro saídas, se necessário. Para a programação da matriz CHESS, um compilador capaz de aceitar códigos em JHDL [122] e gerar diretamente a *netlist* CHESS foi desenvolvido.

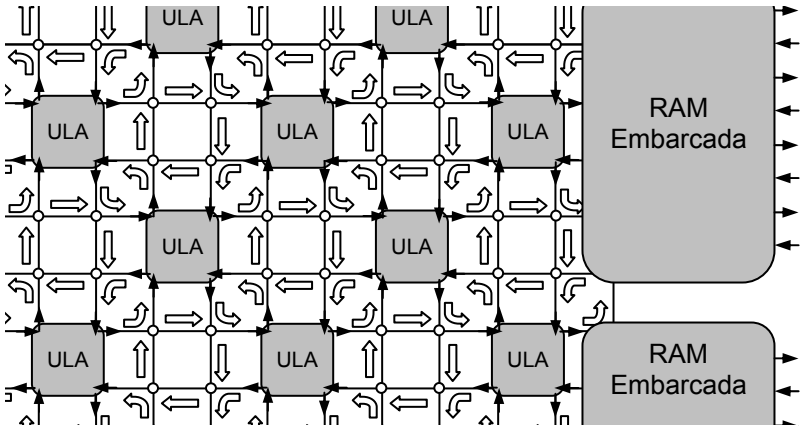


**Figura 7 - CHESS: ULAs e blocos de chaveamento dispostos em um tabuleiro de xadrez [33].**

## 2.3. Metodologias de Projeto

Após a exposição das principais plataformas de prototipagem atualmente existentes na seção anterior, esta seção concentra-se apenas nas metodologias de concepção voltadas às plataformas baseadas em FPGA, dispositivo de prototipagem utilizado neste trabalho. A metodologia SoPC também será abordada por ter o FPGA como seu dispositivo fundamental.

Fluxo de projeto FPGA pode ser definido como a seqüência de passos que vai da especificação do sistema até sua prototipagem final em FPGA. O fluxo completo envolve diferentes metodologias a cada passo (p. ex.: metodologia de modelagem, metodologias de verificação, etc.). De maneira geral, pode-se classificar os fluxos existentes em três categorias, conforme a seqüência em que dão evolução ao projeto. São elas:

- ***Top-down*** - nesta abordagem o projeto inicia com uma formulação geral das características finais do sistema desejado, feita de maneira abstrata, ou seja, sem detalhes de como será implementado. À medida que o projeto avança, o sistema vai sendo refinado, dividido em subsistemas, os subsistemas em sub-subsistemas, e assim por diante, até que seja descrito em termos de componentes elementares.

- ***Bottom-up*** - nesta abordagem o projeto inicia a partir da descrição detalhada dos componentes elementares do sistema. Esses componentes são então conectados para formar subsistemas, que por sua vez são conectados para formarem subsistemas maiores e assim, sucessivamente, até compor o sistema completo.
- ***Middle-out*** – é uma mistura das abordagens *bottom-up* e *top-down*. Nela o projeto inicia em um nível intermediário de abstração e a partir dele se avança nos dois sentidos, isto é, refinando a descrição (*top-down*) e compondo subsistemas maiores a partir dos subsistemas descritos no nível intermediário (*bottom-up*).

Para a análise das metodologias de projeto, propõe-se fazê-la tomando como base dois eixos principais:

I. Os aspectos do protótipo que se propõe a otimizar;
II. Os aspectos próprios da metodologia;

Uma breve exposição sobre cada um desses eixos de análise será feita a seguir:

**I. Aspectos do Protótipo**: são características físicas ou comportamentais do protótipo que podem ser quantificadas ou avaliadas de forma objetiva. Os principais aspectos de um sistema prototipado em FPGA são:

1. **Área** – área de silício necessária para acomodar o sistema projetado. Em um FPGA a área está diretamente relacionada aos recursos consumidos (blocos de memória, multiplicadores, registradores, portas lógicas, etc).
2. **Consumo** – consumo energético do dispositivo. As duas principais medidas de consumo de um dispositivo são o consumo médio estático e consumo médio dinâmico. Em sistemas síncronos o consumo estático depende unicamente das perdas parasíticas do dispositivo, não sendo, portanto, influenciado pelo projeto. Por outro lado, o consumo dinâmico depende da atividade do dispositivo, estando assim diretamente relacionado à maneira como o sistema é projetado.

3. **Latência** – A latência é o intervalo de tempo entre o momento em que um estímulo (sinal ou dado) é injetado no sistema e o momento em que o sistema produz a resposta a esse estímulo. Sua medida é dada em unidades de tempo.
4. **Vazão** – A vazão diz respeito à taxa na qual os estímulos são processados pelo sistema, ou seja, a quantidade de estímulos por unidade de tempo. Em sistemas de processamento de vídeo a vazão é comumente medida em *bits*/segundo (bps), *bytes*/segundo (Bps), *pixels*/segundo (pps) ou quadros/segundo (fps).
5. **Freqüência** – medida direta da máxima freqüência interna de operação do dispositivo. Apesar de ser um parâmetro importante do sistema, a maximização da freqüência de operação não costuma ser objeto de interesse direto nos sistemas de processamento digitais. Ao invés disso, o interesse é mais comumente focado na minimização da latência e/ou maximização da vazão, sendo a elevação da freqüência uma conseqüência indesejada desse objetivo.
6. **Confiabilidade** – basicamente, o conceito de confiabilidade refere-se à capacidade de um sistema de operar livre de falhas mesmo quando sob situações adversas. Este conceito envolve características físicas do dispositivo e também características de projeto, tais como autoteste e redundância. Sistemas vitais como marca-passos, piloto automático e de controle de navegação de aeronaves são tipicamente casos em que a confiabilidade é o foco principal. A confiabilidade de um sistema é definida em termos de probabilidade de falhas mediante uma dada situação.
7. **Segurança** – esta característica diz respeito à dificuldade de se extrair ou adulterar informações contidas ou tratadas pelo dispositivo ou comandá-lo de forma indesejada. O projeto de um sistema seguro envolve, normalmente, técnicas de criptografia e de combate à engenharia reversa. Sistemas de comunicação militar, de cartões de crédito e sistemas de identificação são exemplos em que a abordagem de projeto de segurança é necessária. O grau de segurança de um sistema pode ser avaliado por sua robustez a ataques por meios, técnicas e ferramentas conhecidas.

Existem ainda outros aspectos do protótipo, tais como: tensão de alimentação, tensões das interfaces lógicas, capacidade de dreno de corrente das saídas, temperatura de

operação, entre outros, que não foram citados por se tratarem de características inerentes ao FPGA escolhido ou que são dependentes de parâmetros externos ao escopo do projeto do sistema (as tensões das interfaces lógicas dependem do projeto da plataforma, por exemplo).

Os aspectos do protótipo que se deseja otimizar são definidos implícita ou explicitamente ainda na fase de especificação do sistema. Ter o baixo custo como requisito principal traduz-se como "projeto eficiente em área", especificar que o sistema é para aplicações embarcadas traduz-se como "projeto de baixo consumo", e assim por diante.

Uma vez que tais características fazem parte da especificação do sistema (ponto de partida do fluxo) e do protótipo (passo final do fluxo) conclui-se que devem estar presentes também durante todo o fluxo de projeto de forma a assegurar a coerência entre os níveis. Em outras palavras, os aspectos desejados do protótipo têm influência sob todo o projeto.

Dessa forma, durante a fase e a especificação do sistema é importante destacar de forma clara quais são os aspectos do protótipo que se deseja otimizar. Isto deve ser feito de forma hierárquica, ou seja, devem-se estabelecer as características a serem otimizadas em ordem decrescente de relevância. Essas características são as guias-mestre do projeto e estabelecem critérios importantes à tomada de decisões durante as etapas futuras do fluxo.

**II. Aspectos da Metodologia:** caracterizam a metodologia adotada. São aspectos relacionados unicamente ao desenvolvimento do sistema em si. Interessam unicamente à equipe de desenvolvimento e sua hierarquia:

1. **Produtividade** – expressa o potencial de produção do fluxo adotado. É medida em transistores/homem/mês, portas/homem/ano, etc.
2. **Flexibilidade** – é o grau de facilidade que o sistema projetado, através do fluxo adotado, apresenta a alterações no projeto.
3. **Reusabilidade** – É o grau de facilidade ou de potencialidade que um sistema, ou parte dele, possui para ser reusado. Está relacionado à alta coesão e baixo acoplamento com outros módulos do sistema [123].

4. **Verticalidade** – está ligada ao número de passos que compõe o fluxo de projeto, sendo a execução de cada passo dependente da execução do passo anterior. Um fluxo com elevado grau de verticalidade é composto por vários passos.
5. **Horizontalidade** – está ligada à capacidade de distribuição (paralelização) das tarefas que compõem cada passo do fluxo de projeto.
6. **Dificuldade** – está relacionada ao grau de conhecimento exigido para a execução do projeto.
7. **Confiabilidade** – expressa o quão imune a erros de projeto é a metodologia adotada. Este aspecto está fortemente ligado ao grau de automação do fluxo e às metodologias de verificação empregadas.

Dessa forma, um determinado fluxo de projeto pode estar focado na otimização de um ou mais aspectos da metodologia. Estes aspectos não estão diretamente relacionados aos aspectos do protótipo descritos anteriormente, muito embora possam influenciá-los de maneira indireta.

Três importantes conceitos cujas metodologias de projeto estão intimamente voltadas ao aumento da produtividade de projeto e, portanto, em comunhão com o trabalho proposto, são:

- Conceito de Projeto Heterogêneo;
- Conceito SoPC;
- Conceito de Síntese de Alto Nível (HLS);

Uma breve abordagem sobre o fluxo de projeto VLSI clássico será apresentada na seção 2.3.1, introduzindo importantes noções de projeto e facilitando a compreensão das seções seguintes. Os conceitos de Projeto Heterogêneo e SoPC são abordados nas seções 2.3.2 e 2.3.3, respectivamente, sendo o conceito de Síntese em Nível de Sistema Eletrônico abordado especificamente na seção 3.1 do capítulo 3, por se tratar de elemento fundamental da metodologia proposta neste trabalho.

### 2.3.1. Fluxo de Projeto VLSI Clássico

O projeto de um circuito integrado, parte dele ou protótipo FPGA devem seguir um fluxo preciso cujo objetivo principal é maximizar a confiabilidade do projeto elevando as chances de sucesso na sua prototipagem. O fluxo de projeto típico de um circuito integrado VLSI é esboçado de forma simplificada na Figura 8 [4].

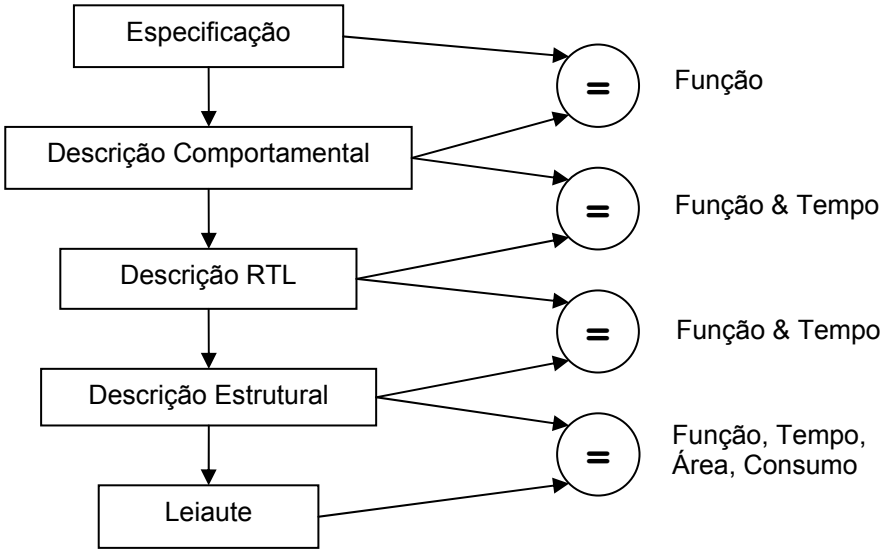


**Figura 8 - Fluxo de projeto clássico de um circuito integrado VLSI**

Cada um dos nós do fluxo representado na Figura 8 descreve o sistema alvo com diferentes níveis de detalhamento. Percorrendo-se o fluxo de cima para baixo, parte-se do menor ao maior nível de detalhamento, o que implica numa elevação da complexidade. Quanto menor o nível de detalhamento na descrição do sistema, menor é a complexidade e maior é o nível de abstração dessa descrição, ou seja, a descrição do projeto parte do nível mais abstrato (e menos complexo) ao menos abstrato (e mais complexo). Ao mesmo tempo, quanto mais se desce no diagrama, mais a descrição do projeto se assemelha ao seu nível final: o protótipo em silício ou FPGA. Uma vez que a perda de abstração é gradual a cada nível que se desce, é possível realizar testes entre níveis de descrição consecutivos de forma a verificar que a descrição menos abstrata ainda corresponde à descrição anterior, mais abstrata [4]. Os tipos de testes que podem ser realizados entre cada nível de descrição são indicados do lado direito do diagrama.

Conforme a Figura 8, o projeto inicia-se com a Especificação do sistema a ser projetado. Nesta fase, o projeto é descrito no mais elevado nível de abstração, isto é, descrevendo-se apenas o que ele deve fazer e a quais restrições deve atender, sem se preocupar em "como fazê-lo". A especificação é tipicamente feita em linguagem natural (português, inglês, francês, etc.).

A especificação em linguagem humana precisa ser refinada em uma descrição que possa ser simulada. A este nível de descrição dá-se o nome de Descrição Comportamental que, por sua vez, possui um nível de abstração mais baixo do que a especificação e, portanto, mais complexa. Nesta etapa inicia-se a descrição de como o sistema deve ser implementado. Geralmente esta descrição é representada por uma linguagem de alto nível, tipicamente C, C++, Simulink ou Matlab. A forma como o sistema deve comportar-se é descrita, mas ainda não há noções de circuito tais como: sinal de relógio, registradores, portas de entrada e saída e, tempos de atraso etc.

No nível de Descrição RTL (*Register Transfer Level*), o sistema descrito pelas etapas anteriores passa a ser representado por uma linguagem de descrição de hardware (HDL - *Hardware Description Language*), tal como Verilog, VHDL, System-C ou SystemVerilog. Neste nível, agregam-se noções de circuito tais como: sinal de relógio, registradores, portas de entrada e saída, etc. Simulações neste nível já envolvem formas de onda, mas os tempos de atraso das conexões e dos componentes do sistema ainda não são considerados. No fluxo clássico de projeto RTL, esta descrição é completamente manual ou envolve o mínimo de automação.

Em seguida tem-se a Descrição Estrutural do sistema, que é geralmente obtida de forma automática por ferramentas de síntese a partir da Descrição RTL. Neste nível, o sistema é descrito em termos de blocos lógicos elementares (portas lógicas e registradores) e blocos básicos (somadores, multiplicadores, multiplexadores, memórias e portas de entrada e saída) interligados entre si. Esta descrição pode ser feita tanto visando à prototipagem em FPGA, quanto à em silício, e com ela já se torna possível realizarem-se simulações envolvendo os tempos de atraso dos componentes do sistema, mas os atrasos devidos às conexões ainda não podem ser levados em conta, uma vez que elas ainda não se encontram fisicamente descritas.

Completando o fluxo de projeto tem-se o Leiaute / Síntese FPGA, o nível mais baixo de abstração, portanto, o mais detalhado e complexo. Neste nível, o sistema é descrito tal qual sua implementação, seja em silício, seja em FPGA. No caso da implementação em silício esta descrição é feita em termos de transistores, apesar de estes estarem geralmente reunidos em pequenas células básicas (*standard cells* – células padrões) e por fios precisamente posicionados e dispostos em camadas no *chip* [124][125]. Embora ainda seja possível se descer até o nível de descrição físico-química do circuito integrado, isto na prática não costuma ser abordado pelos projetistas VLSI. No caso da implementação em FPGA, a descrição precisa exatamente quais elementos da FPGA em questão serão utilizados e a maneira como a matriz de conexões entre os elementos será configurada [1][126], assemelhando-se em muitos aspectos à descrição de Leiaute em silício.

Um detalhe importante deste fluxo de projeto é a capacidade de automatização. Esta capacidade é tanto maior quanto menor é o nível de abstração. Enquanto a Especificação e sua tradução para o nível mais baixo (a Descrição Comportamental) são feitas geralmente com pouco ou nenhum auxilio de máquina, as etapas finais como a síntese da Descrição Estrutural para a Descrição em Layout é quase que totalmente automatizada [125]. Esta característica tornou possível o avanço da complexidade dos sistemas digitais até os níveis dos dias de hoje. O QX9775, processador de quatro núcleos da Intel, por exemplo, integra 820 milhões de transistores em uma única pastilha de silício [127], algo que na prática seria pouco provável de ser alcançado sem a automatização da síntese dos níveis de mais baixa abstração do fluxo de projeto.

Ao mesmo tempo, deve-se observar que quanto mais elevado o nível de abstração, maior a facilidade de reuso do código, uma vez que, quanto menor o nível de abstração, mais atrelada está sua descrição às particularidades da tecnologia a qual o sistema será implementado. Em suma, quanto mais elevado o nível de abstração, mais fácil é descrever o sistema, e mais reusável é esta descrição. Por este motivo, a indústria tem buscado cada vez mais elevar o nível de abstração em que os sistemas VLSI são projetados, deixando os níveis mais baixos a cargo das ferramentas de síntese, que avançam no mesmo sentido. Atualmente, já existem metodologias e ferramentas capazes de sintetizar sistemas em silício ou FPGA a partir de suas descrições em nível comportamental [16][17][18][19]. São as

chamadas ferramentas de Síntese em Nível de Sistemas Eletrônicos (ESL - *Electronic System Level*) ou ferramentas de Síntese de Alto Nível (HLS – *High Level Sythesis*), que são explicadas em maiores detalhes na seção 3.1.

### 2.3.2. Conceito de Projeto Heterogêneo

O conceito de Projeto Heterogêneo foi introduzido por Wayne Wolf, sob o título *Hardware-Software Co-design* [128], em 1994, e designa a abordagem de projeto no qual parte das funcionalidades do sistema são implementadas por circuitos eletrônicos dedicados e parte por programas contidos em um ou mais microprocessadores embarcados, sendo o desenvolvimento de ambas as partes (circuito e programa), realizado paralelamente [129][130][131][132].

Formalmente, a definição de Projeto Heterogêneo, segundo [129], é "o estudo do projeto de sistemas de computação embarcada". Conforme este conceito, esse estudo compreende três principais tarefas:

- co-especificação - estabelecimento de especificações que descrevam ambos os elementos de circuito e programa, bem como os relacionamentos entre eles;
- co-síntese - síntese automática ou semi-automática do circuito e do programa para atender às especificações;
- co-simulação - simulação simultânea dos elementos de circuito e de programa, geralmente em diferentes níveis de abstração.

Wayne Wolf, em uma recente publicação sobre a evolução dessa metodologia [133], define que sua arquitetura alvo pode ser generalizada como no diagrama da Figura 9.

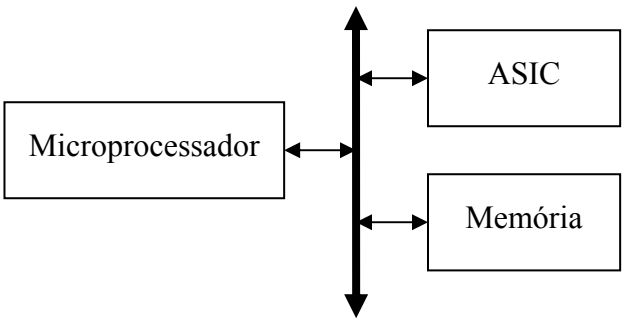


**Figura 9 - Arquitetura alvo da metodologia de Projeto Heterogêneo [133]**

Embora essa metodologia visasse inicialmente sistemas microprocessados compostos por dispositivos fisicamente distintos (processador, memória e ASIC), a evolução dos FPGAs fez com que o sistema pudesse também ser completamente integrado em um único dispositivo. Neste caso, o ASIC da Figura 9 passa a ser um componente IP. Atualmente diversos FPGAs integram microprocessadores e blocos de DSP em suas estruturas.

Abordagens de Projeto Heterogêneo incluem o desenvolvimento de algoritmos para co-síntese [134], simulação [135] e técnicas baseadas em fluxo de dados [136]. O fluxo de Projeto Heterogêneo pode ser resumido conforme a Figura 10 [30] (verificação entre os níveis não explicitada).

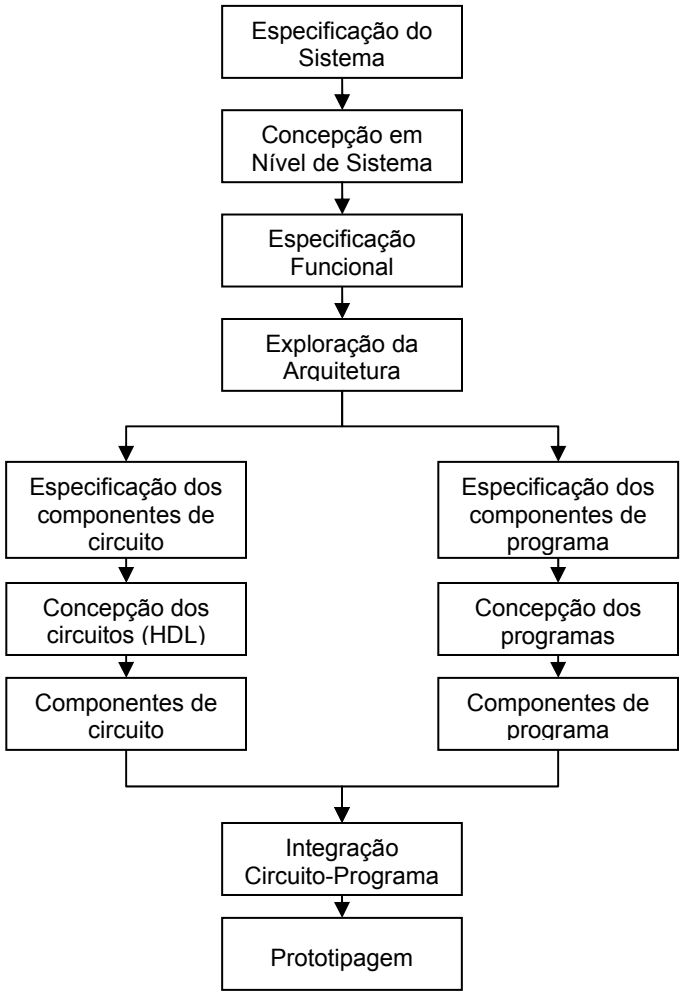


**Figura 10 - Fluxo de Projeto Heterogêneo [30]**

As metodologias e ferramentas de Projeto Heterogêneo partem geralmente da etapa de exploração arquitetural a fim de conduzir os projetistas a uma implementação eficaz de cada uma das partes do sistema.

Desta forma, os projetistas consideram o compromisso entre a implementação em circuito e em programa de cada um dos componentes de um sistema que devem operar conjuntamente. Tal implementação deve respeitar o comportamento desejado dentro de um conjunto de metas previamente especificadas. Para tanto, a fase de exploração da arquitetura se baseia em ferramentas de estimativa de desempenho de cada parte do sistema. Em função dessas estimativas, a ferramenta gera ou propõe um conjunto de soluções de implementação para cada bloco funcional. Os diferentes módulos são em seguida concebidos em função das decisões tomadas durante a etapa de particionamento. A descrição do programa e do circuito pode ser feita de maneira manual (programação em código Assembly, RTL, etc) ou com o auxílio de ferramentas adaptadas (compiladores e, ou, ferramentas de síntese).

No que diz respeito à implementação das partes em circuito, o tempo de desenvolvimento pode ser significativamente reduzido através da reutilização de blocos IP (*Intellectual Property* – Propriedade Intelectual) já existentes. A comunicação entre os componentes de circuito e o programa também deve ser cuidadosamente estudada. Várias soluções foram propostas para realizar a essa comunicação, tais como os trabalhos de [137][138][139][140].

A última etapa consiste em validar funcionalmente o sistema obtido, ainda que as validações funcionais por simulação e co-simulação tenham sido feitas durante as etapas do processo de refinamento do sistema. De fato, os tempos de simulação de um circuito digital dependem principalmente do nível de abstração de sua descrição.

Um avançado conjunto de ferramentas de projeto e suporte ao projeto de sistemas heterogêneos é o CoFluent Studio [141] fornecido pela CoFluent Design.

### 2.3.3. Conceito de Projeto SoC

Um SoC é um sistema que integra circuitos eletrônicos de diversas funcionalidades, tais como: microprocessadores, memória, conversores A/D e D/A, moduladores, demoduladores, controladores de mídia, sensores, etc, em um único circuito integrado, na forma de um sistema eletrônico completo voltado para uma aplicação específica, que de outra forma seriam integrados como dispositivos fisicamente distintos em uma placa de circuito impresso.

As principais vantagens desse conceito são: melhor desempenho do sistema, menor espaço requerido para montagem, maior confiabilidade do sistema e a redução dos custos finais do produto alvo.

Por outro lado, a integração de tantos componentes numa única pastilha de silício tem desvantagens que precisam ser levadas em conta já durante a fase inicial do projeto, tais como: elevada área de silício, custos de projeto e prototipagem elevados, tempos de projeto e prototipagem mais longos do que um sistema tradicional, maior complexidade na depuração, menor rendimento na fabricação (devido à grande área de silício ocupada) e a integração de IPs de várias fontes diferentes, possivelmente independentes e ligadas a tecnologias de fabricação distintas (*low-power*, *high-speed*, RF, MEMS, etc).

Como solução à parte dos problemas da arquitetura SoC, criou-se o conceito de SoPC (explicado na seção 2.2.2), que pode ser tido simplesmente como um "SoC Programável" baseado em FPGA.

Um projeto direcionado a uma plataforma SoPC, tipicamente um sistema que integra componentes em circuito e em programa, herda, por esta razão, parte de seu fluxo de projeto e ferramentas da metodologia de Projeto Heterogêneo. Além disso, a quase inexistência de blocos analógicos nos FPGAs, atualmente existentes no mercado, faz com que, freqüentemente, parte dos dispositivos do sistema acabe ficando do lado de fora do FPGA, o que leva comumente a adoção do conceito de Projeto Baseado em Plataforma.

Assim como em outras arquiteturas, o fluxo de projeto de um SoPC exige ferramentas de projeto capazes de lidar com a união entre o projeto de circuitos e

programas (conceito de Projeto Heterogêneo). A popularidade dos SoPCs impulsionou o desenvolvimento de uma nova geração de ferramentas de projeto integrado por parte dos principais fabricantes de FPGA como o Altera SoPC Builder [142] e o Xilinx System Generator [143], e o suporte em ferramentas de terceiros como o Accelchip DSP Synthesis [144] (posteriormente adquirido pela Xilinx [145]), Synplicity Synplify DSP [146] e CoWare SPW (atualmente CoWare Signal Processing Designer [147]). O fluxo de projeto típico de um SoPC, utilizando ferramentas de CAD comerciais, pode ser generalizado pelo diagrama da Figura 11 [117].

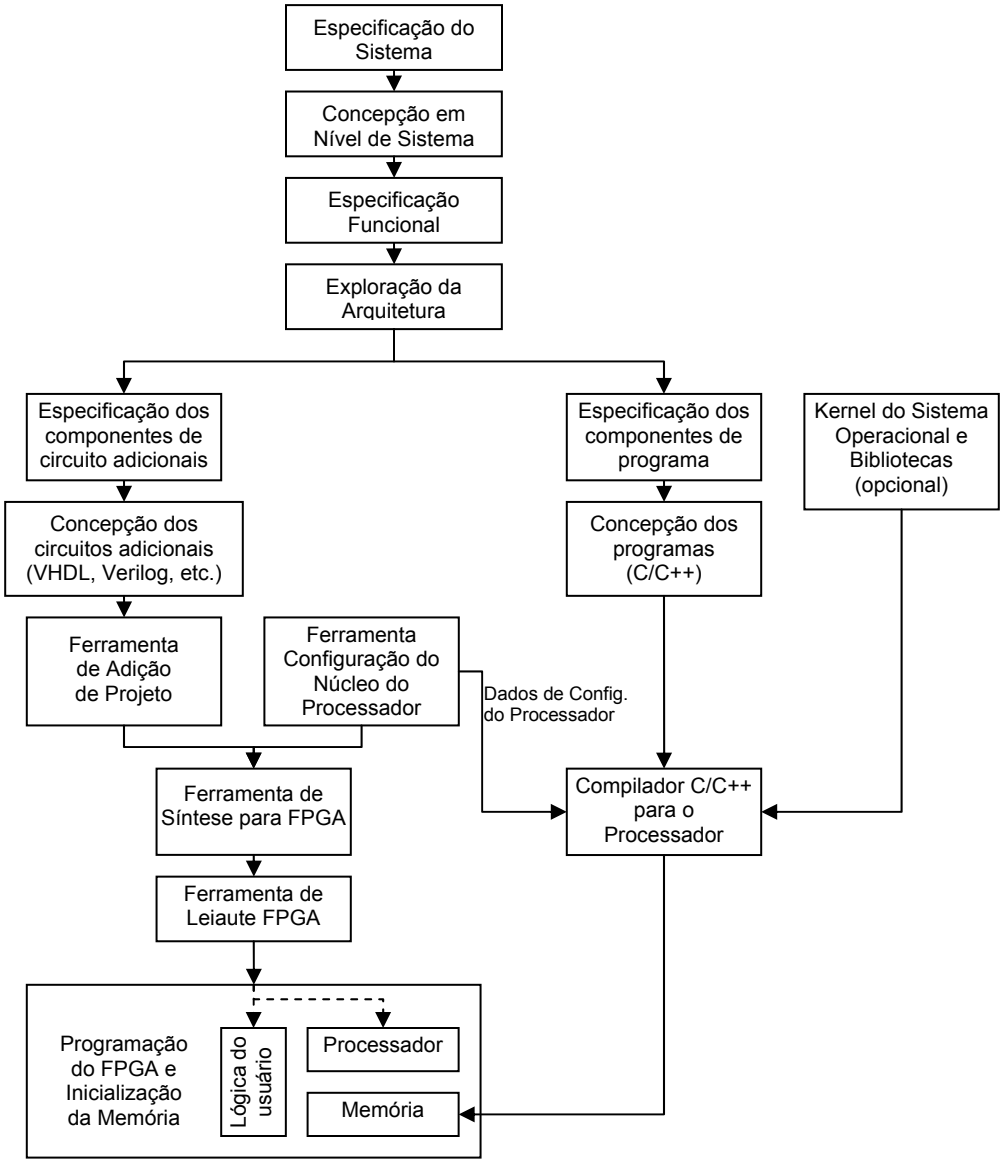


**Figura 11 - Fluxo de projeto típico de um SoPC [117].**

# 3. Estado-da-Arte

## 3.1. Síntese de Alto Nível (HLS)

### 3.1.1. Introdução

Síntese de alto nível é o processo de conversão de uma especificação em nível algorítmico (comportamental) em uma descrição de circuito eletrônico para prototipagem VLSI (em silício ou FPGA). Ela também é conhecida como síntese comportamental ou síntese algorítmica.

Recentemente, o termo ESL (*Electronic System Level*), criado por Gary Smith em meados de 2002 [148], tem ganhado grande popularidade. Apesar de estar sendo utilizado pela indústria para identificar ferramentas de síntese e desenvolvimento em alto nível, até o presente momento, não existe uma definição formal e clara para ele, sendo as existentes conflitantes e vagas [149][150][151]. Bailey et. al., em seu recente livro sobre ESL [151] discute essa problemática e ensaia defini-lo como sendo: *"a utilização das abstrações apropriadas para ampliar a compreensão sobre um sistema e aumentar a probabilidade de sucesso na implementação da funcionalidade de maneira efetiva em custo, atingindo as limitações necessárias."*. Essa definição, no entanto, ainda parece bastante vaga, de sorte que seu emprego será evitado nesse trabalho, dando-se preferência aos termos como "ferramenta de síntese de alto nível", "ferramenta de projeto de alto nível" ou HLS, quando cabíveis.

Um grande benefício deste tipo de síntese é a praticidade no reuso de IPs, tornando-os significativamente independentes das características particulares de implementação (tecnologia, freqüência de operação, etc.) [8][9][13][16]. Descrições em nível funcional são fáceis de serem ressintetizadas e modificadas para adaptarem-se às novas especificações. Ao mesmo tempo, o tamanho de uma descrição em alto nível (C++, por exemplo) é, em média, cerca de dez vezes menor que a equivalente em RTL [8]. Isto reduz drasticamente o tempo de modificação do código bem como as chances de introdução de erro no sistema final por falha humana no projeto [8][9][152].

Outro benefício é o elevado grau de automação das ferramentas de síntese de alto nível. Isto permite uma boa exploração da micro-arquitetura sintetizada. Com isso, ocorre uma mudança de paradigma, conforme apresentado na Figura 12, e a freqüência de relógio torna-se um parâmetro e não mais uma restrição, como no projeto RTL padrão. Assim, a freqüência pode ser praticamente descorrelacionada dos requisitos de vazão de dados e latência do sistema [6].

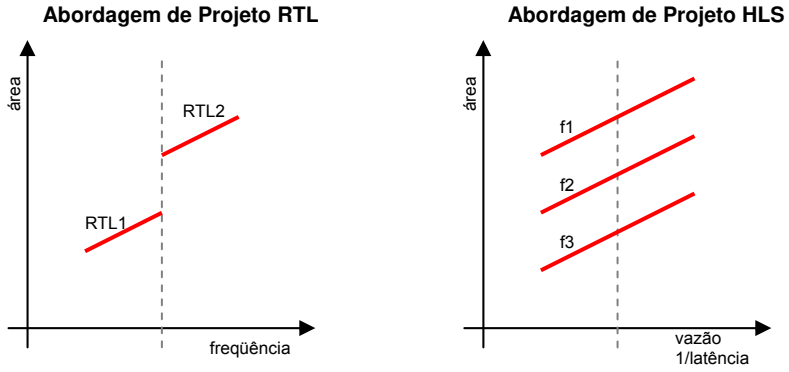


**Figura 12 - Benefícios da exploração da micro-arquitetura nas ferramentas de HLS: a freqüência torna-se um parâmetro do projeto [6]**

Isto significa que, priorizando pelos requisitos funcionais (vazão e latência), o projetista pode explorar várias soluções diferentes que satisfazem às especificações, porém, com diferentes freqüências de operação. Graças à descorrelação da freqüência de operação, um projeto de baixa freqüência não será necessariamente penalizado pelo consumo de área de silício, se for aplicado em uma solução de alta freqüência. Aliando-se ainda a capacidade de exploração dos recursos de particionamento e gerenciamento de memória, é possível se obter soluções com um elevado grau de otimização, atingindo desempenhos próximos a soluções obtidas pelo fluxo tradicional (RTL codificado à mão) [153].

Na Figura 13 estão representadas três possíveis soluções de implementação de uma tarefa hipotética composta por três estágios. Cada estágio necessita de um pulso de relógio para ser executado, consumindo um registro que é atualizado ao seu término. Na primeira solução, a implementação mais simples é adotada, isto é, a arquitetura seqüencial, onde uma nova tarefa só é iniciada após o término da anterior. Para esta solução, a execução contínua dessa tarefa gera uma saída a cada três pulsos de relógio. Entretanto, usa-se apenas

um registro e permite-se que haja reaproveitamento de recursos eletrônicos entre os estágios, já que todos os componentes empregados em uma execução da tarefa estão completamente livres para serem utilizados na execução seguinte. Na segunda e terceira soluções, uma arquitetura em *pipeline* foi adotada, havendo na segunda solução uma sobreposição do último estágio da tarefa e na terceira solução a sobreposição dos dois últimos estágios. A segunda solução consome dois registros, uma vez que, em operação contínua, sempre haverão duas tarefas sendo executadas simultaneamente.

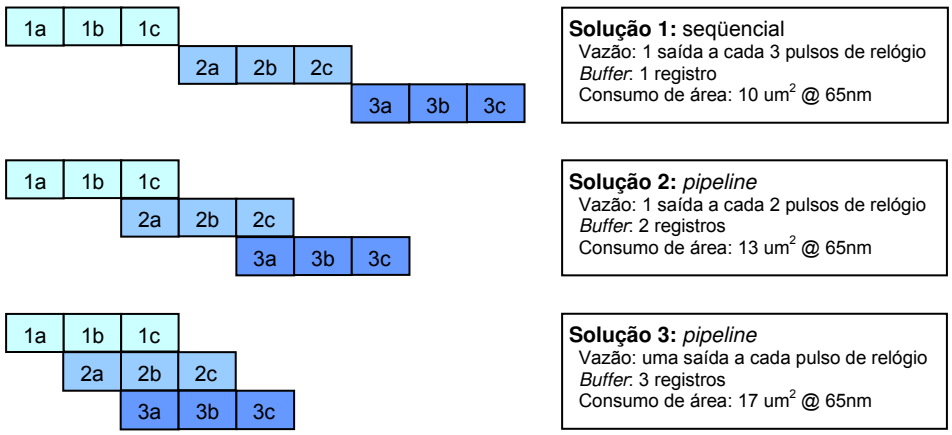


**Figura 13 - Exploração da arquitetura através das ferramentas de síntese em alto nível [154]**

Da mesma forma, a terceira solução consome três registros, e permite pouco ou nenhum reaproveitamento de recursos eletrônicos entre os estágios, já que todos os estágios estarão sendo sempre executados simultaneamente. Contudo, esta última solução produz uma saída a cada pulso do relógio, isto é, três vezes maior que a primeira solução, sem requerer três vezes mais recursos na sua implementação, podendo gerar a mesma vazão que a primeira, mas com uma freqüência de relógio três vezes menor. O tempo gasto para se obter todas essas soluções é dramaticamente reduzido graças à automação oferecida pelas ferramentas de síntese. O projetista pode então tomar a solução mais adequada aos requisitos da aplicação (área e consumo) sem a necessidade de ter que reescrever nenhum código, reconfigurando apenas os parâmetros da síntese [17][18][19][155].

Atualmente também é possível estimar o consumo de potência diretamente a partir do RTL gerado automaticamente por essas ferramentas [156]. Além disso, experiências

mostram que reduções muito mais significativas no consumo de potência de um sistema podem ser alcançadas com otimizações no nível da arquitetura do que no nível de *back-end* do projeto [157][158][159].

Finalmente, pode-se dizer que um dos maiores benefícios da síntese de alto nível está na redução drástica do tempo de simulação do sistema. A simulação de um modelo C ou C++, por exemplo, é centena de vezes mais rápida que a co-simulação do modelo equivalente em RTL [8]. Isto torna possível a validação de sistemas digitais complexos em um tempo muito menor do que pelo fluxo de projeto tradicional, reduzindo o tempo de lançamento da solução no mercado e minimizando custos no seu desenvolvimento.

### 3.1.2. Bases da Síntese de Alto Nível

O processo utilizado pela maioria das ferramentas de síntese de alto nível para transformar códigos em nível algorítmico para descrições RTL pode ser generalizado pelo diagrama apresentado na Figura 14.

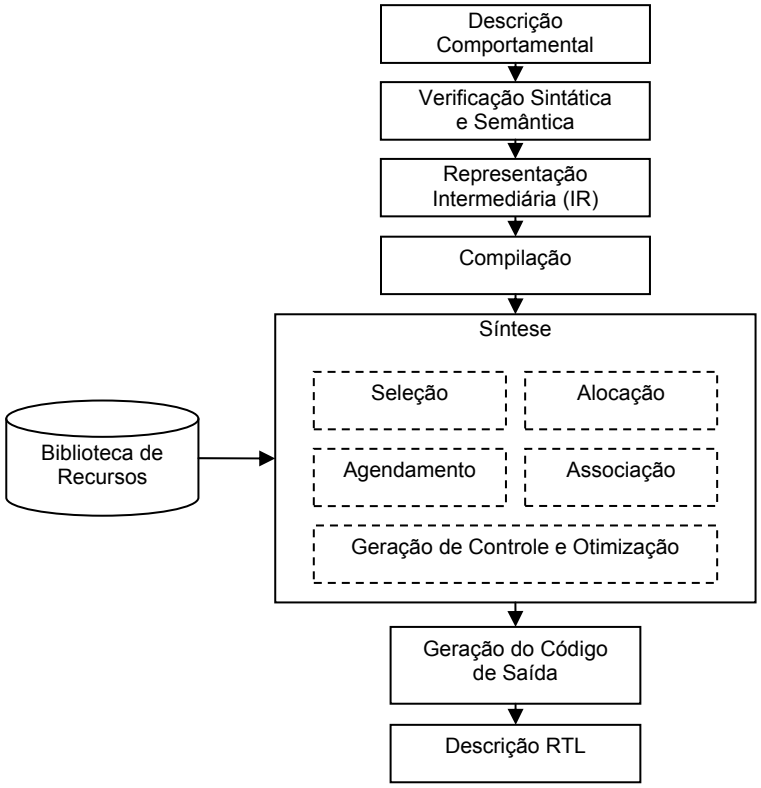


**Figura 14 - Fluxo típico de síntese internamente empregado pelas ferramentas de HLS.**

Na síntese de alto nível a primeira etapa consiste em realizar a verificação sintática e semântica da descrição algorítmica de entrada. A etapa seguinte é a compilação, onde o código comportamental é traduzido em uma representação intermediária (IR – *Intermediate Representation*), própria da ferramenta, capaz de representar tanto o fluxo de dados como o de controle [160].

Atualmente, três modelos são empregados internamente para a representação intermediária nas ferramentas de síntese de alto nível: os Gráficos de Fluxo de Dados (DFG - *Data Flow Graph*) [161], os Gráficos de Fluxo de Dados e de Controle (CDFG - *Control and Data Flow Graph*) [162][163] e ainda os Gráficos de Tarefas Hierárquicas (HTG – *Hierarchical Task Graph*) [164][165], sendo as duas primeiras as mais utilizadas. Tais modelos representam a ordem imposta pela produção e consumo de dados, ou seja, as dependências de dados e controle.

Ainda na etapa de compilação são realizadas importantes tarefas como a eliminação de código morto e o tratamento dos laços de repetição presentes no código. Basicamente, esse tratamento consiste em transformações como "desdobramento" (*loop unrolling*), "fusão de laços" (*loop merging*) e *pipelining* [166]. Estas tarefas afetam drasticamente arquitetura e o desempenho do sistema a ser sintetizado. A partir do modelo de representação intermediário, a síntese é então realizada em etapas. Segundo Gupta [167], essas etapas geralmente são:

- **Etapa de Seleção:** consiste em escolher os tipos de recursos, a partir da biblioteca, que executarão as operações presentes no modelo. Na biblioteca existem vários recursos de diferentes tipos que podem executar cada operação. A seleção é feita levando-se em conta compromissos como desempenho, consumo e área ocupada, de acordo critérios de síntese especificados pelo projetista [168].
- **Etapa de Alocação:** determina para cada tipo de operador ou operação, o número de recursos que serão utilizados na arquitetura final. Recursos consistem em unidades funcionais, tais como somadores e multiplicadores, registros e componentes de interconexão (multiplexadores e barramentos) [160].

- **Etapa de Agendamento:** sua função é agendar o tempo de execução de cada uma das operações, em termos de ciclos de relógio. A seqüência entre as operações agendadas é limitada pelas dependências dos dados, e possivelmente pelo controle, entre as operações, e pelas limitações impostas pelo projetista [169]. Segundo Lin [170], o agendamento pode ser limitado por tempo ou por recursos. No agendamento limitado por tempo prioriza-se a redução do número de etapas de controle em função de uma determinada quantidade de recursos, ao passo que no agendamento limitado por recursos prioriza-se minimizar a quantidade de recursos em função de um número determinado de ciclos de relógio [171].
- **Etapa de Associação:** associa a cada operação, variável, transferência de dados e controle no projeto, os recursos de circuito que foram especificamente reservados na etapa de alocação [172].
- **Etapa de Geração de Controle e Otimização:** a etapa de síntese de controle gera uma unidade de controle que implementa o agendamento, conforme previsto anteriormente. Esta unidade gera os sinais de controle que comandam o fluxo dos dados através dos componentes [168]. A otimização busca minimizar o tamanho da unidade de controle, reduzindo seu o consumo e área ocupada.

Dependendo da ordem com que essas operações são efetuadas as características do código sintetizado mudam [168]. Em uma síntese priorizando a otimização da latência, a alocação será realizada após a etapa de agendamento, por exemplo. Do contrário, se a síntese priorizar a redução de recursos, será o agendamento que terá lugar após a etapa de alocação.

Uma vez concluída a etapa de síntese, o código de saída é gerado, juntamente com *testbenches* e relatórios sobre a síntese, contendo, geralmente, informações sobre a quantidade de recursos de circuito consumidos, bem como estimativas de vazão e latência. Algumas ferramentas dão ainda a opção de síntese RTL automática, gerando também os arquivos de *Netlist* para a plataforma FPGA desejada [155][18], ou mesmo para síntese em silício, segundo a tecnologia de fabricação informada [155][19].

Entretanto, a execução do fluxo descrito na Figura 14 enfrenta, de maneira geral, um grande problema: o algoritmo a ser sintetizado precisa ser estaticamente determinável. Um código estaticamente determinável é aquele cujas propriedades de execução podem ser totalmente determinadas no momento de sua compilação. Esta exigência vem da necessidade de se poder determinar, durante a síntese, quantos e quais recursos serão utilizados para implementar o modelo descrito.

Programas escritos para rodar em um microcomputador, normalmente dispõem de uma vasta quantidade de memória no sistema, o que libera os programadores de se preocuparem com ela e permite o uso de diversos artifícios algorítmicos que implicam direta ou indiretamente em um consumo variável de memória ao longo da execução do código, em virtude do seu comportamento ou de fatores externos. Se durante sua execução, um programa necessitar de mais memória ou recursos do que os disponíveis no sistema, um erro em tempo de execução é gerado e a tarefa não é realizada. Em um circuito, tal comportamento não é visto com a mesma tolerância.

Além disso, em um FPGA os recursos de circuito são limitados, e uma implementação em silício não pode simplesmente assumir "o pior caso" e consumir dezenas de vezes mais área de silício do que o realmente necessário. Adicionalmente, códigos não estaticamente determináveis podem ocultar dependências de dados e controle difíceis de se determinar e que impedem a exploração do paralelismo na síntese.

Um código estaticamente determinável possui as seguintes características:

1. **Não aloca memória dinamicamente:** alocação dinâmica de memória não é sintetizável pois significaria criação dinâmica de circuitos de memória.

2. **Não possui recursividade:** Quando uma função é invocada, necessita-se de mais memória para armazenar os registros e varáveis correspondentes à sua nova chamada. Por essa razão, a recursão utiliza indiretamente alocação dinâmica de memória, não sendo sintetizável pela mesma razão. Algumas ferramentas contornam parcialmente essa característica limitando a profundidade máxima das recursões [173][174].

3. **Não possui ponteiros para funções:** a natureza dinâmica deste tipo de ponteiro torna complexa a análise de dependência de dados e controle no algoritmo, além de poder recair em recursão implícita, não sintetizável.

4. **Suas interfaces são completamente definidas:** uma vez sintetizadas, as interfaces das funções tornar-se-ão interconexões entre subsistemas, sistemas, ou entre o sistema e o exterior. Dessa forma, interfaces de função com argumentos variáveis implicariam em interconexões e pinos criados dinamicamente, não sendo portanto sintetizáveis.

O HardwareC [175][176] é uma linguagem de alto nível concebida com base nessas quatro características: ela não suporta ponteiros, recursão e alocação dinâmica de memória. Por conseguinte é totalmente sintetizável.

Uma solução ao problema de síntese de códigos contendo alocação dinâmica de memória e ponteiros foi proposta por Semeria et al. [177] e implementada na ferramenta SpC [178].

### 3.1.3. Evolução das Ferramentas

A síntese de circuitos a partir de descrições de alto nível teve início ainda em meados dos anos 80, através de trabalhos pioneiros como os de Hilfinger [77], Rosenstiel et al. [179], De-Michelli et al. [180], McFarland et al.[181] e Stroud et al. [182]. Literaturas pioneiras tratando desse assunto podem ser encontradas em [152][183][184][185][186]. Uma breve abordagem das principais soluções de síntese de alto nível propostas ao longo dos últimos 20 anos é discutida a seguir.

Ainda no final da década de 80, a AT&T Bell Labs concebeu o **Cones** [182], uma ferramenta de síntese automatizada capaz de transformar modelos comportamentais escritos em uma linguagem baseada em C [187], previamente verificados em simuladores ESIM [188] ou MOTIS [189] e gerar implementações em nível de portas. Nesse sistema, o modelo C descrevia o comportamento do circuito, de lógica seqüencial, durante cada ciclo de relógio.

O **HYPER** [190], conjunto de ferramentas de síntese de alto nível concebidas por Rabaey [191], no início da década de 90 foi adotado por Chen [75] e Yeung [78] como principal ferramenta de síntese para o PADDI [74], um dispositivo de arquitetura multi-processada programável, proposto por eles, objetivando a aceleração da prototipagem de sistemas de vídeo (discutida na seção 2.2.1).

Em 1993, Ernst et al. desenvolveram uma ferramenta chamada **COSYMA** [192][193], que extraia o núcleo computacional do sistema modelado em C*, um superconjunto de C com restrições de processos e noções de temporização, e o implementava como um circuito co-processador para acelerar a execução do software.

O **OCAPI** [194], um ambiente completo de simulação, verificação e síntese de alto nível desenvolvido pelo IMEC desde meados da década de 90, almejando a integração em um ambiente de síntese automática a partir de modelos MatLab, sintetizava modelos descritos em C, utilizando bibliotecas de ponto fixo, em modelos VHDL sintetizáveis. O OCAPI-xl [195], uma versão mais recente da ferramenta, foi licenciada pela CoWare [196] para integração no CoWare Platform Architect [197], sua ferramenta de síntese de alto nível.

O **SPARK,** um ambiente de síntese comportamental desenvolvido por Gupta et al. [167], é capaz de sintetizar RTL para ambas as funções de controle e fluxo de dados a partir de descrições comportamentais em ANSI C. Utilizando recursos de biblioteca de circuitos recursos e restrições temporais configuráveis pelo usuário, a ferramenta sintetiza tarefas comportamentais como agendamento, alocação de recursos, e composição sistêmica [198]. A ferramenta de síntese utiliza uma abordagem paralela, nomeada PHLS (*Parallelizing High-Level Synthesis*), que não só permite a otimização da arquitetura de hardware, como também realiza otimizações pré-síntese do código C, tais como as descritas na seção 3.1.2. Experimentos com o ambiente foram realizados utilizando vários algoritmos de processamento de imagem (MPEG-1, MPEG-2, GIMP, e Susan) e posteriormente sintetizados via Synopsys Design Compiler [199], demonstrando que tais otimizações de código alcançaram reduções de 50% no tempo de atraso das funções, com um custo de área inferior a 20% [200][201]. A tecnologia foi licenciada pela Poseidon Design Systems para utilização na sua ferramenta de síntese comportamental, o **Triton Builder** [202][203].

Uma interessante ferramenta de síntese de alto nível voltada para uma variação da arquitetura paralela RAW, na qual um ou mais unidades de processamento podem ser substituídas por blocos de circuito dedicados, foi criada por Babb et al. [204]. A ferramenta, baseada no compilador **SUIF** [205], toma como entrada códigos comportamentais escritos em C ou Fortran, gerando como saída códigos RTL em Verilog.

O **Bach** [206], uma ferramenta de síntese desenvolvida pela Sharp, gerava códigos VHDL, de forma automática, a partir de algoritmos codificados em Bach C, uma linguagem também desenvolvida por eles, composta por um subconjunto limitado do ANSI C, mas com extensões baseadas em CSP [207] e OCCAM [208], para explicitar paralelismo, comunicações entre processos paralelos e largura de bits dos dados e operações aritméticas.

O **Transmogrifier**, uma ferramenta de síntese de alto nível, desenvolvida academicamente em 1994 por Galloway [209], utilizava um subconjunto restrito da linguagem C, batizada Transmogrifier C, para gerar códigos Netlist de circuitos de lógica sequencial. A ferramenta foi utilizada em projetos acadêmicos da Universidade de Toronto, gerando implementações funcionais em FPGA, sendo abandonada após cerca de cinco anos [210].

No final dos anos 90, duas ferramentas comerciais, o **C2HDL** [211], da C-Level Design, e o **AR|T Builder** [212], da Frontier Design, forneciam suporte à síntese de modelos algorítmicos em C para modelos em Verilog ou VHDL. Ambas as ferramentas dispunham de capacidade limitada ao agendamento e compartilhamento de recursos além de uma série de restrições aos modelos de entrada em C, no caso do AR|T Builder. Por outro lado, o C2HDL suporta todas as instruções ANSI C, com exceção das bibliotecas, mas ponteiros eram considerados apenas como endereços aos dados armazenados nas memórias, o que exigia a alocação das memórias, para armazenar as diversas variáveis, e unidades de endereçamento. Posteriormente, a C-Level Design foi adquirida pela Synopsys que passou a integrar o C2HDL como parte de sua ferramenta de síntese de alto nível, o Behavioral Compiler [213]. O AR|T Builder, no entanto, não alcançou muito sucesso entre os projetistas levando ao encerramento das atividades da Frontier Design, anos mais tarde.

O **Cyber** [214], outra ferramenta desenvolvida nos anos noventa, foi concebida NEC e empregava um subconjunto da linguagem C, nomeada BDL (*Behavior Description Language*), como linguagem de entrada para sintetizar modelos em RTL. A ferramenta foi utilizada internamente pela empresa na produção de alguns componentes, incluindo um complexo adaptador de interface de rede comercial, que, segundo eles, apresentou desempenho de cerca de 30% superior ao equivalente projetado à mão em arquitetura microprocessada. Uma versão evoluída dessa ferramenta, juntamente com outros aplicativos de verificação e simulação, também desenvolvidos pela NEC, foram reunidos em um aplicativo nomeado **CyberWorkBench** [6][215], que é atualmente utilizado industrialmente pela empresa no projeto de produtos comerciais.

Outro importante trabalho desenvolvido pelo IMEC, ainda nos anos noventa, foi o **CoWare** [216], um avançado ambiente de desenvolvimento de projetos em alto nível voltado à sistemas de arquitetura heterogênea (circuito/programa), que podiam ser simulados e sintetizados a partir de especificações também heterogêneas. Um dos principais pontos de foco desse ambiente de desenvolvimento era o suporte à implementação de interfaces de comunicação inter-processadores. A integração entre os componentes de circuito e programa era feita por uma ferramenta interna chamada Symphony, que gerava de maneira interativa os circuitos e trechos de código adicionais, necessários à comunicação. No entanto, o ambiente de síntese, que aceitava modelos algorítmicos em C/C++ como entrada, baseava-se nos compiladores DFL [217] e CATHEDRAL [218], o que impunha importantes limitações como: o suporte limitado ao uso de ponteiros e a incapacidade de gerar *threads* múltiplas. Mais tarde, os criadores do CoWare fundaram uma empresa, de mesmo nome, lançando no mercado uma evolução dessa ferramenta, batizada de **N2C** [219].

No **PAM-Blox** [220], desenvolvido por Mencer et al., uma metodologia *bottom-up* é apresentada na qual uma biblioteca de componentes pode ser definida e utilizada como objetos C++ para construir sistemas para a arquitetura Pamette. No PAM-Blox-II [221], uma versão mais recente da ferramenta, a flexibilidade da síntese foi extendida de maneira a comportar também outras arquiteturas.

Um ambiente de projeto semelhante também foi desenvolvido por Gokhale e Minnich [222], com base na linguagem dbC (*Data-parallel Bit-serial C*) [223], para implementação na arquitetura Splash. Para arquiteturas heterogêneas do tipo SoPC, as ferramentas de síntese Garp [224], bem como a Nimble [225] geram automaticamente co-processadores redirecionáveis (*retargetable co-processors)* para acelerar os laços de repetição presentes no código, a partir de ANSI C.

O compilador **CASH** [226], concebido por Budiu et al., possui a particularidade de gerar circuitos assíncronos a partir da identificação do paralelismo em nível de funções de algoritmos em ANSI C.

Outro trabalho notável foi desenvolvido por Semeria e De Micheli, uma ferramenta de síntese e otimização de código C com refinado suporte a ponteiros, batizada **SpC** [178]. Baseada no compilador SUIF [205], o utilitário oferece ainda recursos à otimização da localização dos dados em múltiplas memórias, registros ou até mesmo conexões.

O **AccelFPGA** [227], uma ferramenta de síntese poroposta por Banerjee et al., é capaz de gerar modelos RTL sintetizáveis em Verilog ou VHDL e testbenches para simulação a partir de descrições de alto nível em MatLab. Outros esforços em utilizar modelos em MatLab para síntese RTL podem ser encontrados em [228][229], onde Pascal Urard descreve uma metodologia empregada dentro da ST Microelectronics para produzir circuitos integrados de alto desempenho, empregando métodos de verificação formal baseados em simulação simbólica [230].

Há ainda ferramentas de síntese baseadas em JHDL (*Java Description Language*), como as que foram concebidas por Hutchings [231] e Kuhn [232], e em UML, como o **Chip Fryer** [233], desenvolvido por Thomson et al. e o **MODCO** [234] de Coyle e Thornton, e os utilitários de Kangas et. al. [235], Björklund e Lilius [236], Nguyen et al. [237] e Riccobene et al. [238]. Uma solução atualmente comercializada para a síntese em RTL de modelos UML é a ferramenta **FalconML**, da Axilica [239][240].

Finalmente, o **Trident Floating Point Compiler** [241], uma ferramenta de síntese de alto nível de código aberto [242], desenvolvida academicamente por Tripp et al., é voltada à síntese em VHDL de algoritmos em ponto flutuante descritos em C.

Outras soluções relevantes incluem ainda o **Flamel**, desenvolvido por Trickey [243], para a síntese de códigos em Pascal para modelos em nível de portas; o **BECOME,** proposto por Wei et al. [244]; **Mistral** [245], de Van Nieuwenhoven et al.; **GAUT** [246], concebido por Martin et al, e o **Cathedral** [218][247], de De Man et al., ambos voltados à para processadores de sinal dedicados; **AMICAL** [248]**,** de Park et al.; o **CADDY** [249], desenvolvido ainda na década de oitenta por Camposano e Rosentiel, que sintetizava descrições comportamentais em DSL (*Digital System Specification Language*) [250] para uma *netlist* hierárquica chamada STRUDEL (*STRUcture Description Language*) [251]; o **CALLAS** [252], desenvolvido pela Siemens; ou **HERCULES/HEBE** [180][253], empregado pela Olympus [176], baseado em HardwareC [175]; e o compilador **Streams-C** [254], desenvolvido por Gokhale et al.

### 3.1.4. Soluções Comerciais Atuais

Atualmente, diversos fabricantes oferecem no mercado ferramentas competitivas de síntese e suporte à síntese de alto nível. Uma breve revisão sobre as mais importantes soluções comerciais hoje disponíveis é apresentada a seguir:

O **Catapult** [174] da Mentor Graphics, originário do Monet, antiga ferramenta de síntese, do mesmo fabricante, oferece um refinado suporte à descrições C/C++, impondo poucas restrições ao projetista em termos de recursos algorítmicos. Variáveis de ponto flutuante, ponteiros, recursão, alocação dinâmica de memória e laços de repetição dinâmicos (*unbonded loops*) e outras estruturas, normalmente proibidas por outras ferramentas, são comportadas, dentro de certos limites, nas versões mais recentes do compilador.

Dispõe de um bom suporte à exploração de múltiplas micro-arquiteturas, interfaces e mapeamento de memória, sem que seja necessária qualquer alteração no código de origem. Bibliotecas especializadas para diversos FPGAs da Xilinx e Altera, bem como para

diversas tecnologias de fabricação ASIC, garantem uma boa otimização na utilização dos recursos da plataforma alvo, em virtude das metas de síntese informadas pelo projetista. Além disso, a ferramenta de suporte Catapult C Library Builder permite a criação de novas plataformas alvo, ou mesmo a modelagem de dispositivos dedicados para serem empregados na síntese. A verificação do código sintetizado contra o modelo comportamental é feita através da utilização de um *testbench*, escrito pelo projetista em C++. A partir desse código a ferramenta gera automaticamente toda a estrutura de verificação, permitindo que todo o sistema possa ser co-simulado. Outro importante recurso do Catapult é o suporte a projetos hierárquicos, que o torna capaz de explorar o paralelismo também entre funções do código, levando a incrementos notáveis no desempenho do sistema gerado.

O **Cynthesizer** [18] da Forte Design Systems, é uma ferramenta de síntese baseada em SystemC não temporizado. Possui tanto bibliotecas de suporte síntese ASIC quanto a FPGAs Xilinx e Altera. Dispõe de um eficiente processo de verificação e simulação no qual o mesmo *testbench* utilizado na simulação comportamental pode ser empregado em todas as fases do projeto. Para isso, os protocolos de sincronização de cada um dos sinais das interfaces devem ser informados pelo projetista. A síntese das micro-arquiteturas e interfaces segue especificações passadas na semântica da descrição comportamental sob a forma de diretivas, o que reduz a flexibilidade da exploração arquitetural. Além disso, arquivos de diretivas de síntese, de regras do projeto, *testbench* e *makefile*, precisam ser codificados manualmente, o que limita a praticidade da ferramenta.

Embora o SystemC não seja uma linguagem puramente comportamental, outras ferramentas de síntese de alto nível também basearam-se nela para descrever e integrar sistemas, tais como o **Platform Architect** [197] da CoWare, o **AutoPilot** [255] da AutoESL e o **C-to-Silicon** [19] da Cadence.

O **PICO** [173], originalmente concebido nos laboratórios da Hewlett Packard e atualmente comercializado pela Synfora, é uma ferramenta de síntese de circuitos capaz de gerar descrições RTL a partir de modelos algorítmicos em ANSI C. Uma metodologia recursiva de composição do modelo sintetizado chamada TCAB (*Tightly Coupled Accelerator Blocks*), integrada na sua versão mais recente, o PICO Extreme, a torna capaz

de implementar sistemas grandes e complexos em tempo reduzido. A metodologia se baseia no conceito de que procedimentos em C são semanticamente equivalentes a módulos Verilog ou entidades VHDL. Com isso, recursos como hierarquia e recursão podem ser empregados para elevar o desempenho do sistema. O PICO oferece ainda suporte à otimização da síntese por área, desempenho ou consumo, além de recursos a integração de interfaces OCP-IP [256].

O **Agility DK Design Suíte** [257], originário do Agility Compiler da Celoxica, era inicialmente uma ferramenta HLS baseada em SystemC, posteriormente alterada para Handel-C [258]. Atualmente o suporte a códigos em ANSI C/C++ restringe-se apenas ao ambiente de simulação e depuração, não sendo ainda suportados para síntese. Conta com recursos à exploração de algoritmos complexos e micro-arquiteturas ainda nas fases preliminares do fluxo de projeto, além de ferramentas de análise e estimativa de desempenho do sistema. Outra interessante ferramenta comercializada pelo mesmo fabricante é o **Agility MCS** [259], um compilador de códigos em MatLab para ANSI C que, em conjunto com ferramentas de síntese de alto nível baseadas em C, pode dar origem a um fluxo de síntese de alto nível baseado em MatLab.

O **SystemVue** [260], da Agilent Technologies, é uma plataforma completa de desenvolvimento, modelagem, simulação, verificação e prototipagem de arquiteturas mistas voltada a sistemas de telecomunicações. Sua versão mais nova, o **SystemVue 2008**, inclui recursos à síntese de alto nível através de um ambiente capaz de gerir modelos em C++, MatLab (linguagem m), Verilog, VHDL, ou ainda modelos gráficos, descritos em diagramas de blocos. A síntese do sistema modelado pode ser direcionada para implementação em FPGAs, DSPs, ASICs ou mesmo através de componentes analógicos ou de RF. Sua biblioteca inclui centenas de blocos extensíveis de subsistemas de comunicação, processamento de sinais, RF, ponto flutuante e interfaces padronizadas, como TCP/IP. Adicionalmente, o custo de sua licença anual é dezenas de vezes inferior ao de outras ferramentas como Catapult e PICO (cerca de US$ 14.000 contra US$ 350.000, em média).

Utilizando fluxo de síntese baseado em MatLab e SimuLink, tem-se também o **Symplify DSP HLS** [261] da Symplify/Synopsys, voltado a síntese de arquiteturas para

processamento digital de sinais em FPGA ou ASIC, com uma eficiente ferramenta de síntese em *Netlist* para diversos dispositivos da Altera e Xilinx.

Outros interessantes esforços baseiam-se no paralelismo implícito, como o **Impulse C Compiler** [262], que oferece um bom suporte ao fluxo de dados mediante um subconjunto de instruções C, e o **Mitrion-C** [263], que assume uma arquitetura paralela fundamental, chamada Mitrion Virtual Processor, voltada ao processamento paralelo para supercomputação [264]. Uma vez modelado, o sistema pode ser sintetizado em um co-processador reconfigurável baseado em FPGA, como os fornecidos pela XtremeData [265][266], de pinagem compatível com processadores AMD e Intel atuais, soquetado diretamente em uma placa mãe ou sistema de múltiplos núcleos.

Oferecendo um bom suporte ao C/C++ padrão, pode-se citar ainda o **C2R** Compiler [267] da Cebatech, o **C-to-Silicon** [19] da Cadence e o recente **Triton Builder** [202] da Poseidon Design Systems, orginário do compilador SPARK [268], desenvolvido academicamente por Gupta, et al. [167].

Finalmente, para a síntese de modelos em SystemVerilog e UML, tem-se o **Bluespec Compiler** [269], da Bluespec e o **FalconML** [240], da Axilica, respectivamente.

Um resumo das ferramentas de síntese comerciais atualmente no mercado é apresentado na Tabela 2.

**Tabela 2 - Ferramentas de síntese de alto nível comerciais atualmente no mercado**

| Ferramenta | Ref. | Entrada | Saída | Desenvolvedor | Comentário |
|---|---|---|---|---|---|
| Catapult | [174] | ANSI C/C++ | Verilog, VHDL, SystemC, Netlist. | Mentor Graphics | Suporte a projeto hierárquico; bom suporte a ponteiros e poucas restrições ao código de entrada. |
| Cynthesizer | [18] | SystemC | Verilog, SystemC, Netlist | FORTE | Processo de verificação eficiente; exploração arquitetural limitada. |
| PICO | [173] | C | Verilog, SystemC, Netlist | Synfora | Exploração nativa do paralelismo implícito entre funções. |
| Agility DK | [257] | Handel-C | Verilog, VHDL, SystemC, Netlist | Agility | Bom suporte a interfaces heterogêneas. |
| C-to-Silicon (C2S) | [19] | C/C++, SystemC | Verilog | Cadence | Síntese incremental e interativa. |
| Symplify DSP HLS | [261] | MatLab-m, SimuLink | Verilog, VHDL, Netlist. | Synplify / Synopsys | Voltado a arquiteturas DSP em FPGA ou ASIC. |
| SystemVue 2008 | [260] | C/C++, MatLab-m, Verilog, VHDL, GUI** | Verilog, VHDL, Netlist. | Agilent | Ambiente completo de projeto de sistemas mistos incluindo suporte a blocos analógicos e RF. |
| Platform Architect | [197] | SystemC | Verilog, VHDL, SystemC | CoWare | Bom suporte a arquiteturas heterogêneas; módulos RTL externos podem ser integrados durante a síntese. |
| Bluespec Compiler | [269] | SystemVerilog | Verilog, SystemC | Bluespec | Bom controle da arquitetura e micro-arquitetura gerada. |
| FalconML | [240] | UML | Verilog, VHDL, SystemC | Axilica | Recursos ao controle de portas e temporização ainda em UML. |
| Triton Builder | [202] | ANSI C | Verilog, VHDL | Poseidon | Voltado a arquiteturas heterogêneas. |
| C2R Compiler | [267] | ANSI C | Verilog | CebaTech | Bom suporte ao paralelismo e controle de interfaces. |
| Mitrion | [263] | Mitrion-C | Netlist, FBF* | Mitrionics | Voltado ao projeto de co-processadores dedicados para supercomputação. |
| ImpulseC | [262] | C | Verilog, VHDL, Netlist, FBF* | Impulse | Suporte a SoPC e outras arquiteturas heterogêneas; suporte a geração de co-processadores. |
| AutoPilot | [255] | C/C++, SystemC | Verilog, VHDL, EDIF | AutoESL | Suporte a ponto flutuante; Ferramenta recente, pouca informação disponível. |
| Simulink HDL Coder | [270] | Simulink, Matlab-m, Stateflow charts | Verilog, VHDL | MathWorks | Suporte à modelagem totalmente gráfica; Código gerado elegante e inteligível. |

*FBF (*FPGA Bit Files*) – Arquivos binários de configuração específicos para cada FPGA
**GUI (*Graphical User Interface*) – Interface de entrada gráfica baseada em diagrama de blocos

## 3.2. Abordagens Existentes

O fluxo proposto neste trabalho concentra-se em otimizar a produtividade da metodologia de projeto, tendo como guias-mestre a vazão e a latência do sistema. Por esta razão, são exploradas nesta seção, apenas metodologias que também se enquadrem nessas características.

**Fluxo de Vitabile** - Uma abordagem baseada unicamente no estabelecimento de regras de codificação em alto nível é proposta em [271]. Essa metodologia propõe 10 regras a serem seguidas pelos projetistas na escrita do código C, que modela o sistema projetado de forma que o mesmo possa ser facilmente transcrito em Handel-C [258] (linguagem para síntese de alto nível adotada) e corretamente sintetizado pela ferramenta, Celoxica DK (atualmente Agility DK [272]). A abordagem proposta foi aplicada em um sistema de processamento de vídeo para a detecção e reconhecimento de sinais de trânsito, sendo prototipada em FPGA e alcançando desempenho de aproximadamente um quarto da freqüência real de vídeo no processamento de quadros de 320x240 pixels. Outra implementação utilizando as primeiras versões da linguagem Handel-C e sua ferramenta de síntese (DK), no projeto de um bloco DCT 8x8 (*Discrete Cosine Transform* – Transformada do Co-seno Discreta) alcançou um desempenho de aproximadamente 75% do obtido pela implementação direta em VHDL, consumindo, no entanto, com um consumo de portas lógicas mais de três vezes superior [258].

**Fluxo Huet** - Uma abordagem baseada no refinamento automático de modelos em alto nível de abstração foi proposta por Huet [273]. Nela, o protótipo implementado é composto por circuitos e códigos (plataforma tipo SoPC), contendo no circuito um microprocessador capaz de executar o código gerado de maneira a comportar-se da mesma forma que o modelo algoritmo de origem. Essa metodologia consiste em quatro principais passos ligando três diferentes modelos do sistema: Modelo Funcional da Arquitetura, Modelo da Plataforma e Modelo de Macro-Arquitetura (Figura 15), utilizando como ferramentas de base: o Cofluent Studio™ [141], para a integração circuito/programa; e o GAUT [274], um utilitário de síntese de alto nível de processadores digitais de sinais (DSPs), para a síntese do circuito. A metodologia foi aplicada no desenvolvimento de um

sistema de comunicação sem fio, alcançando valores de desempenho pouco inferiores aos obtidos pela metodologia tradicional de prototipagem unicamente em circuito (sem partes programadas).

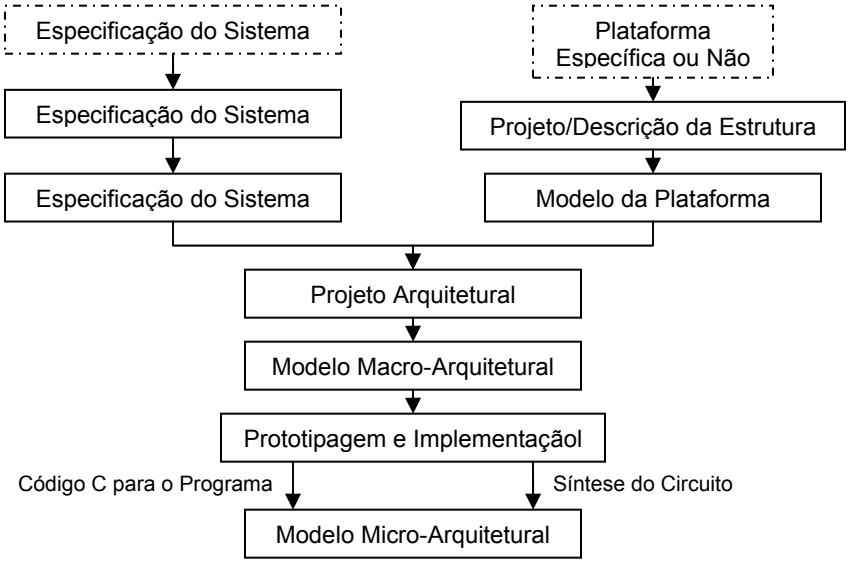


**Figura 15 - Fluxo de prototipagem rápida proposto por Huet.**

**Benkrid** et al. propõem, em [275], um ambiente de modelagem de alto nível de sistemas de processamento de vídeo. O aplicativo proposto é capaz de sintetizar em FPGA modelos baseados em operadores de álgebra de imagem [276] descritos em linguagem C++. O processo de síntese empregado pela ferramenta consiste em transcrever o sistema modelado de C++ para uma linguagem intermediaria interna, HIDE [277], baseada em Prolog [278], na qual o sistema é mapeado em uma arquitetura pré-definida composta por componentes escalonáveis e parametrizáveis. Um transcritor dedicado (HIDE2EDIF) gera, então, a *netlist* EDIF para ser mapeada no FPGA selecionado através de aplicativos próprios do fabricante.

### 3.2.1. Outros Trabalhos Relevantes

Além dos fluxos abordados na seção anterior, podem-se citar ainda dois trabalhos relevantes ao estudo, apesar de não serem voltados a síntese em FPGA. São eles:

**Fluxo de Engels** - Engels et al. propõem uma metodologia composta por seis passos para a prototipagem de sistemas de codificação de vídeo em DSPs comerciais com a qual

implementaram um codificador à baixa resolução (160x240 *pixels*, 30 quadros por segundo) com um esforço de aproximadamente 6 homens/mês [279].

**Fluxo de Madisetti** - uma metodologia sistemática para o agendamento de algoritmos de processamento digital de sinais em sistemas multiprocessados síncronos para prototipagem rápida [280].

# 4. Fluxo de Prototipagem Rápida para Vídeo

## 4.1. Introdução

Conforme exposto anteriormente, apesar do grande avanço das ferramentas de síntese de alto nível, ainda existem fatores que limitam, ou até mesmo impedem, que o projeto de um sistema em nível algorítmico possa ser diretamente sintetizado sem antes ter passado por alguma recodificação.

À medida que o sistema se torna mais complexo, a recodificação torna-se mais trabalhosa, demorada e susceptível a erros. Em particular, os sistemas de processamento digital de vídeo raramente escapam a essa regra. Eles são especialmente conhecidos pela sua complexidade, elevados requisitos de vazão, processamento de grandes quantidades de dados e manipulação de grandes blocos de memória [281]. O processamento de vídeo de alta definição em tempo-real, que requer 3 bilhões de operações por segundo, e a geração sintética de vídeo que pode chegar a 1 trilhão de operações por segundo [282], são exemplos que ilustram bem essas características. Por essa razão, projetos de sistemas de processamento de vídeo costumam envolver grandes equipes e consumir tempos elevados no seu desenvolvimento.

Propõe-se a concepção de uma metodologia para acelerar a prototipagem em FPGA de algoritmos de processamento de vídeo, estabelecendo regras gerais de projeto que conduzam o projetista a sintetizar o sistema de forma satisfatória, com esforços e tempo reduzidos, em relação ao fluxo RTL clássico. O fluxo de projeto que aqui se propõe não almeja a substituir o fluxo de projeto atualmente empregado na indústria, e sim, tornar-se uma alternativa para a geração de protótipos funcionais em FPGA, embora se acredite que ela possa ser aprimorada para ser diretamente aplicada a soluções comerciais.

Partindo de modelos algorítmicos em C/C++, o fluxo proposto faz uso de uma ferramenta de síntese de alto nível para gerar as descrições RTL, necessárias para a prototipagem, no menor tempo possível. Para tanto, escolheu-se o Catapult C Compiler da Mentor Graphics, abordado na seção 3.1.4. A decisão baseou-se principalmente na

linguagem de entrada (C/C++), no seu elevado suporte à exploração arquitetural, nas reduzidas limitações impostas ao código do modelo comportamental, nos recursos integrados de verificação semi-automática, na boa qualidade do RTL gerado além do nível de maturidade e na confiabilidade da ferramenta.

Mesmo sofrendo poucas restrições de codificação por parte da ferramenta de síntese, o modelo algorítmico ainda precisa ser adaptado para que possa ser corretamente sintetizado. Por essa razão, a definição do novo fluxo de projeto começa com a identificação das adaptações necessárias para a estes códigos, explorando as particularidades existentes nos sistemas de vídeo, bem como as possibilidades oferecidas pelas ferramentas de síntese de alto nível.

O refinamento do fluxo de projeto e a definição da metodologia são explorados ao longo das seções seguintes.

## 4.2. Visão Geral do Fluxo Proposto

Objetivando a prototipagem rápida de algoritmos de processamento de vídeo, propõe-se o fluxo apresentado na Figura 16. Relativamente ao exposto na seção 2.3, pode-se dizer que o fluxo segue uma abordagem do tipo *top-down*, que a metodologia de projeto aqui descrita prioriza a otimização da vazão e latência do protótipo, e foi concebida visando a elevação da produtividade e a redução da dificuldade envolvida. Sua horizontalidade está diretamente relacionada ao número de partições do sistema tratado, conforme será explicado adiante.

No intuito de facilitar a compreensão da metodologia, o fluxo apresentado na Figura 16 não detalha a etapa de "Síntese de Cada Partição" e o "Fluxo de Otimização do Sistema", que são particularmente abordados na seção 4.3.

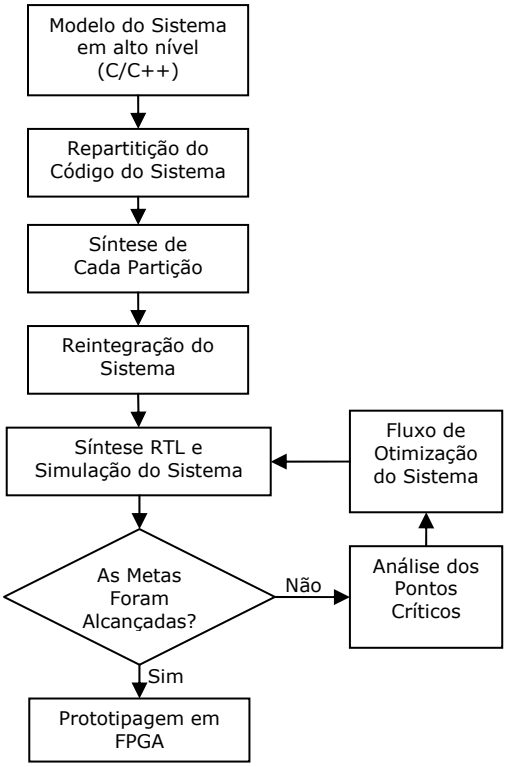


**Figura 16 - Diagrama do fluxo de projeto proposto.**

O primeiro passo é a "Repartição do Código do Sistema". Este passo não é uma reestruturação do código, mas apenas uma análise e divisão do algoritmo, repartindo-o em subsistemas menores, isto é, em suas funções mais importantes, sem qualquer modificação na arquitetura ou recodificação do modelo. Se um dado projeto é composto por funções *main_func_A( )* e *main_func_B( )*, diretamente responsáveis pelo processamento de vídeo, e funções auxiliares menores *aux_cpy_memory( )* e *aux_clr_memory( )*, este será segmentado em duas partes principais em função de *main_func_A( )* e *main_func_B( )*, mantendo-se a funções auxiliares associadas às principais, porém, sem que recebam especial atenção nos primeiros passos do fluxo.

Esta repartição do algoritmo não é necessariamente feita separando o código em arquivos distintos, mas apenas identificando as partições que serão tratadas em separado nos passos seguintes do fluxo. Contudo, por vezes, esta prática pode ser útil por questões de organização ou quando partições distintas precisam ser tratadas por diferentes projetistas de uma equipe. Tratar cada partição separadamente permite facilmente identificar, corrigir e

otimizar pontos problemáticos, reduzindo o tempo gasto na otimização do sistema, já que segmentos menores de código são mais facilmente processados pela ferramenta de síntese, bem como pelo projetista. Além disso, a abordagem através de partições do modelo também auxilia a utilização dos recursos de síntese hierárquica suportados por algumas ferramentas como Catapult e PICO, permitindo uma melhor exploração do paralelismo implícito no código, levando a ganhos substanciais no desempenho do RTL gerado. A etapa de "Síntese de Cada Partição" será abordada em detalhes na seção 2.3.

A desvantagem de repartir o sistema é que, algumas vezes, isto exige que *testbenches* dedicados para cada partição precisem ser criados, além de requerer que ao final do processo de síntese o sistema precise ser reintegrado, adicionando outro passo no fluxo de projeto: "Reintegração do Sistema".

Após a etapa de "Reintegração do Sistema", o RTL correspondente ao sistema completo precisa ser verificado em simulação. A verificação do RTL contra o modelo algorítmico original pode ser realizada fazendo-se uso das metodologias tradicionais de verificação. Entretanto, a ferramenta de síntese adotada na execução desse fluxo, Catapult, provê esse recurso de maneira quase que totalmente automatizada gerando o *testbench* RTL e todos os arquivos de suporte à simulação, diretamente a partir do *testbench* em C++ fornecido pelo projetista no início da síntese. Da mesma forma, a simulação do RTL gerado pode ser feita empregando-se as ferramentas e métodos tradicionais do fluxo RTL clássico, ou através do próprio compilador, caso este suporte seja disponível. Se os resultados da simulação forem satisfatórios, o RTL pode então ser mapeado na plataforma FPGA desejada e em seguida prototipado, encerrando o fluxo de projeto. Caso contrário, o sistema precisará ser otimizado.

A otimização começa com uma "Análise dos Pontos Críticos do Sistema", tais como uma partição específica com vazão insatisfatória ou a lógica de cola (*glue logic*) entre duas ou mais partições. Em nível comportamental, a lógica de cola pode ser uma função intermediária que faz interface entre duas partições (funções principais) ou mesmo uma simples operação matemática. Identificados os pontos críticos, executa-se o "Fluxo de Otimização do Sistema", descrito em detalhes na seção 2.3.

Para cada otimização feita, recomenda-se re-sintetizar a partição em questão, analisando-se os novos resultados e comparando-os aos anteriormente existentes. Se os novos resultados estiverem dentro das metas esperadas, ressintetiza-se o sistema completo, avaliando-se os ganhos de desempenho obtidos. Finalmente, após todas as otimizações necessárias, o sistema pode ser então sintetizado e prototipado na plataforma FPGA.

## 4.3. Síntese das Partições do Código

A segunda etapa do fluxo representado na Figura 16, "Síntese de Cada Partição", é, de fato, o cerne do fluxo proposto. É nesta etapa que todo o algoritmo C/C++ é traduzido em RTL sintetizável. O fluxo correspondente a esta etapa é apresentado na Figura 17.

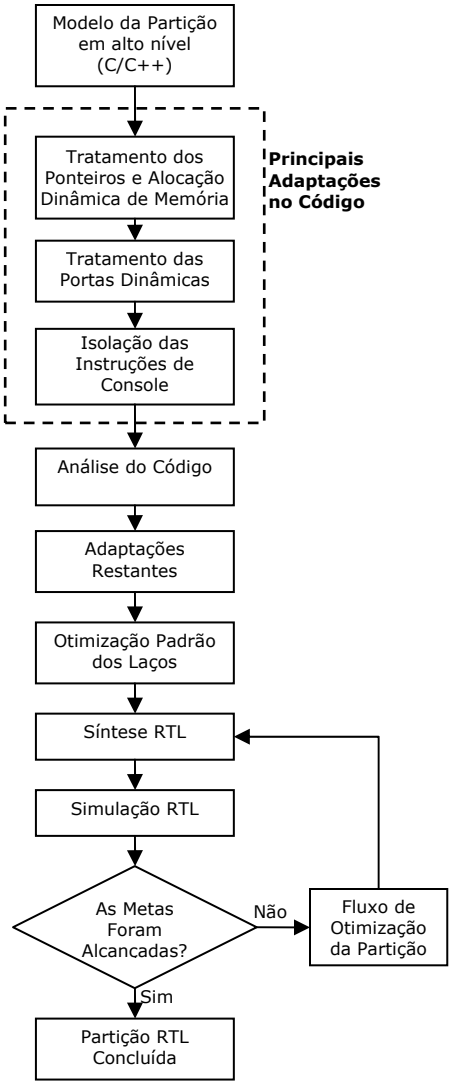


**Figura 17 - Fluxo de síntese de cada partição do código**

As etapas correspondentes ao bloco “Principais Adaptações no Código” reúnem o conjunto das mais freqüentes e importantes modificações que precisam ser aplicadas ao código para adaptá-lo à síntese em RTL. É importante observar que esse conjunto de modificações é melhor adaptado as algoritmos de processamento de vídeo, tais como algoritmos de redução do ruído descritos em [283][284][285][286]. Outros tipos de sistemas podem ter um conjunto diferente de modificações prioritárias.

Dessa forma, todas as regras descritas a seguir devem ser aplicadas no código C/C++ das partições identificadas na etapa de “Repartição do Código” (Figura 16). Elas são diretamente derivadas das características de um código estaticamente determinável, descritas na seção 3.1.2. Uma exceção a esta regra são os segmentos de depuração, comumente empregados na codificação algorítmica. Tais segmentos são tipicamente identificados por diretivas do tipo “#ifdef DEBUG_ENABLE *instruções* #endif”. Desde que os *flags* de ativação dessas diretivas estejam desabilitados, a ferramenta de síntese ignorará tais segmentos, tal qual um compilador comum o faria. Por esta razão, as instruções contidas por estas diretivas não são compreendidas pelas regras que se seguem, podendo o projetista as deixar tais quais foram escritas. Para o restante do código, as regras a serem respeitadas são:

- **Eliminar ponteiros dinamicamente alocados** – ponteiros dinamicamente alocados, se existirem, devem ser eliminados. Algoritmos de processamento de imagem e vídeo são freqüentemente descritos para poderem aceitar diferentes resoluções e formatos de tela. Conseqüentemente, a quantidade de memória alocada para conter os quadros de vídeo depende diretamente das características do vídeo a ser processado. A alocação dinâmica de memória através de ponteiros torna-se, então, uma solução natural e elegante, extremamente útil na codificação do modelo comportamental, mas inconveniente para ser sintetizada em circuito. A alocação dinâmica de memória não é suportada porque sua síntese exigiria a criação dinâmica de circuitos, o que, até o momento, não é fisicamente realizável. Ponteiros dinamicamente alocados são, comumente, a principal obstrução à síntese direta de algoritmos de processamento de vídeo. Geralmente, a maneira mais fácil de satisfazer esta condição é substituir o termo

dinâmico de alocação de memória da instrução *malloc( )* por uma constante, cujo valor corresponda à quantidade de memória requerida pela máxima resolução do vídeo a ser processado pelo sistema. Outra solução consiste em substituir todos os ponteiros dinamicamente alocados no código por arranjos de tamanho fixo igualmente apropriado. Neste último caso deve-se ter o cuidado de remover as instruções *malloc( )* não mais utilizadas.

- **Atenção às instruções *free( )*** – se instruções de alocação de memória (*malloc( )*) forem removidas do código, as respectivas instruções de liberação (*free( )*) precisam ser removidas para evitar erros na síntese. Apesar de essa regra parecer evidente, essa falha, caso ocorra, pode levar a problemas difíceis de se detectar no sistema.
- **Eliminar instruções *memcpy( )*** – essa instrução é largamente utilizada em operações com memória comumente presentes em algoritmos de processamento de vídeo. Sua implementação no C++ padrão, no entanto, envolve internamente operações com ponteiros não sintetizáveis. A solução proposta é a sua substituição por uma equivalente descrita pelo projetista segundo as regras aqui apresentadas.
- **Portas de tamanho estático** – algoritmos de processamento de vídeo possuem, freqüentemente, funções nas quais os quadros de vídeo são referenciados através de ponteiros na interface (ponteiros entre os parâmetros da função). O circuito equivalente desejável para essa função deveria ter interfaces de memória correspondentes aos ponteiros informados. No entanto, algumas ferramentas de síntese não comportam tais interfaces ou interpretam esses ponteiros como portas do tipo *streaming*. A melhor maneira de gerar a interface de maneira correta é substituindo os ponteiros dos parâmetros das funções por arranjos de tamanho fixo apropriado.
- **Descrever a função *abs( )*** – a natureza polimórfica da função abs() presente na biblioteca stdlib.h do C++ padrão a torna difícil de ser sintetizada sem o risco de haver erros de interpretação na síntese. Por essa razão, o Catapult não a implementa, deixando a encargo do projetista provê-la, caso necessite. Isso pode ser facilmente contornado no código através de funções ou macros locais.

- **Eliminar instruções de console** - instruções de console como *fprintf( )*, *printf( )*, *scanf ( )*, *cout* e *cin*, não são sintetizáveis. Por esta razão, não podem estar presentes fora dos segmentos de depuração anteriormente descritos. Caso existam devem ser eliminadas ou incluídas dentro de diretivas de depuração.

O passo seguinte do fluxo é a "Análise do Código". Essa etapa consiste basicamente em uma primeira tentativa de síntese. O objetivo é verificar a necessidade de se realizar outras adaptações no código e identificá-las, caso exististam. A verificação e identificação das adaptações restantes são feitas pela simples análise do relatório de síntese gerado pela ferramenta. Adaptações necessárias são indiretamente reportadas como instruções não sintetizáveis. As adaptações restantes, se existirem, serão então executadas na etapa subsequente, "Adaptações Restantes".

Uma vez que o código torna-se sintetizável, inicia-se a etapa de exploração arquitetural. Em algumas ferramentas como Catalpult, PICO e ImpulseC, o suporte a essa exploração é diretamente provido pela ferramenta via análises gráficas e configurações em tempo de síntese, o que facilita bastante esse processo. Em outros compiladores, como o Cynthesizer, a exploração arquitetural é feita por meio de diretivas no código do algoritmo, o que torna o processo mais lento e menos eficiente. Nesse fluxo, essa exploração resume-se, num primeiro momento, à "Otimização Padrão dos Laços" de repetição do algoritmo. Mais uma vez, tirando-se proveito das características comumente encontradas em algoritmos de processamento de vídeo, é possível definir algumas regras gerais:

1. **Habilitar a fusão de laços** – a fusão de laços é um recurso normalmente suportado pelas ferramentas de síntese. A habilitação desse recurso autoriza a ferramenta a explorar por si própria a potencialidade de fusão dos laços presentes no código, conforme esboçado na Figura 18. Essa capacidade está diretamente relacionada à semelhança entre os laços e à dependência entre os dados no algoritmo.

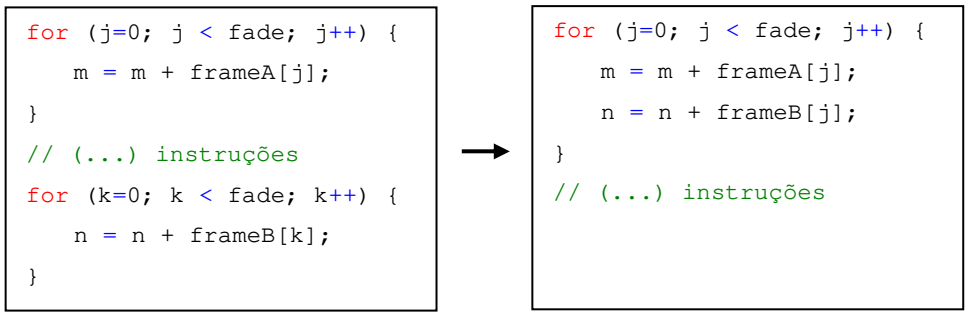


**Figura 18 - Fusão de laços**

2. **Identificar laços de varredura de quadro** - alguns algoritmos processam o vídeo quadro a quadro. Nestes casos, a estrutura de processamento dos quadros costuma ser similar a estrutura representada na Figura 19.
3. **Desdobrar todos os laços internos** – todos os laços de repetição contidos no interior dos laços de varredura do quadro devem ser desdobrados. De maneira geral os melhores resultados, em termos de desempenho, são obtidos desdobrando-se totalmente cada laço. Contudo, tal solução consome mais recursos de circuito e nem sempre é realizável devido a possíveis dependências de dados entre as iterações ou mesmo entre as instruções neles contidas. Recomenda-se iniciar o processo desdobrando o laço em dois ou quatro, caso não se visualize a possibilidade de um valor mais elevado. Se a síntese não puder implementá-lo, o desdobramento deve ser reduzido, ou mesmo eliminado.

```
for (j=0; j < fade; j++) { //laço externo à varredura do quadro
   // (...) instruções internas
}
for (y=0; y < fSizeY; y++) {      // laço de varredura y
   for (x=0; x < fSizeX; x++) {   // laço de varredura x
      // (...) instruções internas
      // (...) laços internos
   }
}
```

**Figura 19 - Núcleo de processamento típico de quadros de vídeo**

4. **Colocar os laços de varredura "y" em *pipeline* mantendo os laços "x" inalterados** – a melhor vazão é obtida maximizando-se a capacidade de

*pipeline*, isto é, disparando cada nova iteração do laço no menor intervalo de tempo possível. No processamento de vídeo, os mais comuns obstáculos a essa otimização costumam ser os múltiplos acessos à memória ao longo do laço, ou cálculos complexos realizados em cada iteração. Assim como na regra anterior, recomenda-se utilizar um valor intermediário de inicialização do *pipeline* e aumentá-lo caso não seja implementável. O valor intermediário é obtido pela média do número de ciclos de relógio consumidos em cada iteração do laço.

5. **Desdobrar todos os laços externos aos laços de varredura de quadro** – com o intuito de maximizar o paralelismo presente no algoritmo, essa definição inicial geralmente conduz a bons resultados na síntese, sem que para isso seja necessária uma análise mais cuidadosa, poupando tempo e esforço humano. Naturalmente, alguns desses laços podem ser melhor otimizados em *pipeline*, ou mesmo não podem ser otimizados. Uma melhor exploração desses laços pode ser feita em etapas futuras do fluxo, caso ainda seja necessária alguma melhora no desempenho. O critério de desdobramento é o mesmo adotado no item 3.

Com a otimização padrão dos laços realizada, a partição pode, enfim, ser sintetizada e simulada (etapas de "Sintese RTL" e "Simulação RTL"). Mediante as metas pré-estabelecidas (vazão de 5.2 Mpixels/s, consumindo menos que 2000 multiplexadores, por exemplo), a análise dos resultados da simulação e os relatórios gerados definirão se a partição sintetizada ainda precisará ser otimizada ou não.

O "Fluxo de Otimização da Partição", representado na Figura 17, é apresentado em detalhes na Figura 20. Trata-se de uma abordagem sistemática para alcançar os objetivos de desempenho desejados em tempo reduzido. Para tanto, a metodologia parte da solução mais fácil (de menor esforço humano) e potencialmente suficiente para resolver o problema, a "Otimização dos Laços". Se após essa etapa as metas ainda não tiverem sido alcançadas, aplica-se outra potencial solução, a "Otimização do Código das Partições", que normalmente requer mais esforço humano. Finalmente, se após as otimizações das etapas anteriores as metas ainda não tiverem sido atingidas, aplica-se a solução potencialmente menor (porém não desprezível), a "Otimização dos Recursos".

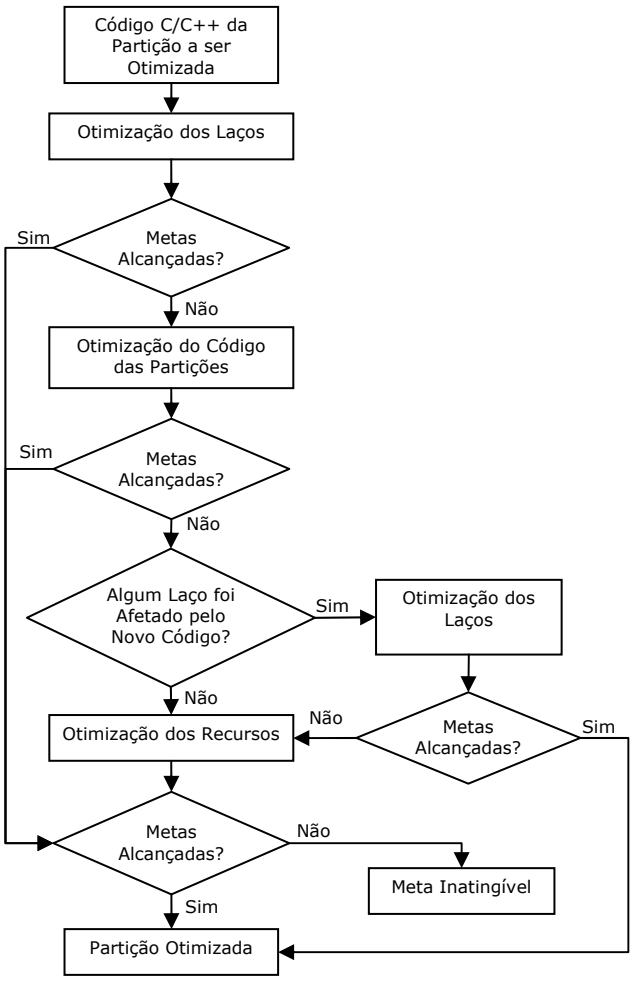


**Figura 20 - Fluxo de otimização das partições**

O passo de "Otimização dos Laços" (Figura 20) é um ajuste fino das otimizações inicialmente executadas. Este ajuste fino consiste numa exploração arquitetural mais criteriosa do que a realizada na etapa de "Otimização Padrão dos Laços", no fluxo de síntese de cada partição (Figura 17). Para isso, devem-se buscar para cada um dos laços do código da partição os melhores valores de desdobramento e *pipeline*. Para tanto, recomenda-se iniciar o processo empregando o máximo desdobramento possível nos laços internos aos laços de varredura do quadro, reduzindo-os caso a síntese não consiga implementá-los. Deve-se ainda explorar o paralelismo nos laços de varredura "x", encontrando os melhores desdobramentos possíveis para cada um deles e, em seguida, redefinir os melhores intervalos de inicialização do *pipeline* dos laços de varredura "y",

diminuindo-os, caso não seja implementável. Por fim, deve-se otimizar cada um dos laços externos aos laços de varredura do quadro, que inicialmente foram apenas desdobrados. Essa otimização consiste em verificar se existem laços que podem ser melhor explorados por *pipeline* (ao invés do desdobramento inicialmente utilizado) e encontrando os melhores valores de desdobramento e *pipeline* para cada um desses laços.

A segunda solução "Otimização do Código das Partições" tem início com a identificação dos pontos críticos no código. Em cada partição, os pontos críticos podem ser funções auxiliares, linhas de instrução ou segmentos de código consumindo grandes fatias de tempo no processamento de vídeo. Esses pontos críticos podem ser determinados através da análise do agendamento de execução das tarefas, recurso presente em todas as ferramentas comerciais estudadas no capítulo anterior, ou através da análise dos relatórios de síntese e pré-síntese, igualmente providos por todas as ferramentas estudadas. Laços também podem esconder pontos críticos como múltiplos acessos desnecessários à memória, operações tipicamente dispendiosas em tempo. Em nível comportamental, tais acessos à memória são referências a ponteiros ou arranjos. Um exemplo típico é esboçado na Figura 21, onde a posição "2" da memória "quadro_B" é acessada duas vezes dentro da mesma iteração do laço.

```
for (y=0; y < fSizeY; y++) {      // laço de varredura y
   for (x=0; x < fSizeX; x++) {   // laço de varredura x
     m = a * quadro_A[posicao_1] + b * quadro_B[posicao_2];
     // (...) outras instruções
     q = a * quadro_A[posicao_2] + b * quadro_B[posicao_2];

   }
```

**Figura 21 - Múltiplo acesso à posição "2" da memória "quadro_B".**

A maior parte das ferramentas de síntese é capaz de identificar automaticamente múltiplos acessos desnecessários à memória, eliminando-os sem que seja preciso intervir no código. Entretanto, algumas vezes o acesso múltiplo ao mesmo endereço não é explícito, estando o endereço "mascarado" por operações matemáticas com valores dinâmicos ou por múltiplas variáveis que levam sempre ao mesmo resultado. Tais estruturas podem enganar a etapa de otimização dos compiladores, conduzindo a soluções não otimizadas.

Uma vez identificados os pontos críticos, estes devem ser otimizados reescrevendo-se o trecho de código correspondente. Múltiplos acessos desnecessários à memória, se existentes, podem ser facilmente corrigidos armazenando-se o conteúdo do endereço da memória em questão em uma variável e utilizando-se essa variável ao longo do código em substituição ao arranjo ou ponteiro. É importante ressaltar que se o código modificado afetar algum laço de repetição, isto é, se a declaração do laço ou instruções nele contidas forem alteradas ou eliminadas, o passo "Otimização dos Laços", previamente descrito, deverá ser refeito de forma a assegurar a coerência e máxima otimização do segmento.

A "Otimização dos Recursos" é o último passo proposto na solução do problema de otimização. A restrição do tipo e quantidade de recursos do FPGA para serem alocados na síntese é um importante artifício geralmente suportado pelas ferramentas síntese por diferentes meios (especificações em ambiente gráfico, arquivos de especificação ou através de diretivas no algoritmo). Normalmente, os possíveis ganhos de desempenho nesta etapa não são muito significativos, no entanto, se a meta de otimização, neste ponto, estiver relacionada à redução dos recursos consumidos do FPGA, melhorias consideráveis poderão ser alcançadas.

Se após todas essas otimizações as metas de projeto ainda não forem alcançadas, a solução pode ser buscada através de modificações mais profundas no código ou na reestruturação na arquitetura do algoritmo. Esta situação é extremamente importante para o projeto, pois pode ser a indicação precoce de gargalos ou pontos sinteticamente problemáticos no modelo, podendo levar a melhorias no algoritmo, modelo, plataforma ou mesmo no dispositivo de prototipagem.

# 5. Estudo de Caso

O fluxo de projeto proposto foi testado e avaliado dentro da divisão de vídeo da STMicroelectronics em Grenoble, na França. Neste intuito, três diferentes algoritmos empregados no processamento de vídeo foram adotados: um Estimador de Ruído, um Preditor e Estimador de Movimento e um Atenuador. Todos os algoritmos utilizados foram extraídos de sistemas comerciais desenvolvidos pela empresa, de maneira a reforçar o aspecto prático da aplicabilidade do fluxo.

Os modelos utilizados podem ser combinados para a criação de um sistema de filtragem digital de vídeo, conforme descrito por Dubois e Sabri em [286]. Um diagrama do sistema completo envolvendo cada um dos blocos avaliados é apresentado na Figura 22.

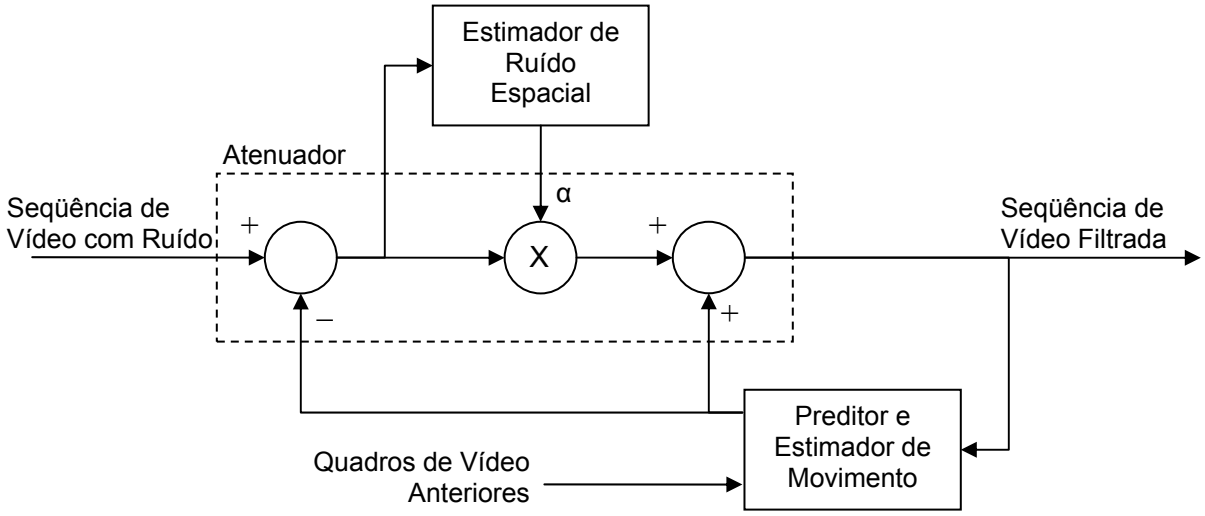


**Figura 22 - Filtro digital de vídeo com compensação de movimento [286]**

Estabeleceu-se como meta que todos blocos sintetizados deveriam processar vídeos em cores (padrão YUV), com definição padrão de televisão (SDTV), isto é, 720 × 480 *pixels*, a uma cadência mínima de 12 quadros por segundo, o que corresponde a metade da taxa de atualização de uma TV padrão, e já aceitável em termos visuais. Versões comerciais do produto equivalente operam a 15 ou 30 quadros por segundo, sob as mesmas condições de resolução e cor.

O primeiro algoritmo avaliado foi o Atenuador. Conforme pode-se notar na Figura 22, trata-se de um subsistema responsável por combinar as informações do Estimador de Ruído Espacial e do Preditor e Estimador de Movimento, e aplicá-las ao quadro de vídeo atual, atenuando o ruído e gerando o quadro seguinte. Apesar de sua simplicidade, o Atenuador está diretamente no caminho do fluxo de dados, o que desperta especial atenção em relação a síntese de suas interfaces de entrada e saída, bem como sua latência e vazão, tanto para não obstruir a fluidez do vídeo quanto para atender aos limites de temporização do barramento de interconexão do SoC no qual é normalmente inserido. Sua síntese foi realizada com sucesso, sem que fossem necessárias alterações no algoritmo além das previstas na etapa de *Preparação do Código*, descrita no capítulo anterior, com especial atenção aos ponteiros na interface da função principal, conforme previsto no fluxo. Sua exploração arquitetural ateve-se a *Otimização Padrão dos Laços*, atingindo de imediato uma vazão 45 vezes superior a meta pré-estabelecida (542 quadros por segundo contra 12, conforme apresentado na Tabela 3).

O segundo algoritmo tratado foi o Estimador de Ruído. Analizando cada quadro de vídeo por regiões, o papel deste sub-sistema é estimar a quantidade de ruído presente em cada um delas e devolver essa informação ao Atenuador. O algoritmo em questão implementa o estimador descrito por Bosco *et al.* [285]. Segundo ele, o ruído médio de uma região *R* de um quadro *f(x,y)* de vídeo pode ser estimado por:

$$\alpha = \sqrt{\frac{1}{N-1}\sum_{(x,y)\in R}\left(f(x,y) - média\right)^2} \qquad \textbf{(Equação 1)}$$

onde: *N* é o número de amostras na região *R,* cujo valor médio dos *pixels* é dado por *média*. O valor de $\alpha$ representa a estimativa do ruído médio presente em R, e é utilizado pelo atenuador para avaliar a intensidade do filtro aplicado sobre o local. É importante notar que a Equação 1 estima o ruído presente apenas em um componente de cor. Em um vídeo em cores a estimação deve, a principio, ser feita nos três componentes. No entanto, operando-se no espaço de cores de luminância e crominâncias (YUV), pode-se tirar vantagem do fato de que a maior parte da informação contida na imagem estão no componente de luminância (Y). Com isso, o Estimador de Ruído pode ser simplificado para

processar apenas o componente Y do vídeo, estendendo a estimativa para os demais componentes na mesma região. Apesar dessa simplificação, já presente no algoritmo tratado, a natureza do cálculo realizado (Equação 1), apresenta dependência intrínseca entre os dados, o que limita a capacidade de exploração de paralelismo e desdobramento dos laços envolvidos, além disso, sua implementação envolve operações com memória para permitir o processamento do quadro de vídeo por regiões. A exploração arquitetural realizada no Estimador de Ruído foi principalmente feita sobre o laço principal, responsável pela varredura das regiões do quadro de vídeo. Tirando proveito da independência de dados entre os quadros, foi possível disparar seu processamento em *pipeline*, conduzindo a um sistema sintetizado capaz de operar com uma vazão mais de 12 vezes superior a meta (135 quadros por segundo contra 12, conforme apresentado na Tabela 3). Mais uma vez a *Otimização Padrão dos Laços*, foi bastante e suficiente para atingir os resultados expostos.

Finalmente, aplicou-se o fluxo proposto ao algoritmo mais complexo dos três, o Preditor e Estimador de Movimento. O sub-sistema implementa basicamente o os blocos de predição e estimação de movimento baseado na soma absoluta das diferenças (SAD), conforme descrito em pela Equação 2 [287][285]:

$$SAD(x,y,r,s) = \sum_{i=0}^{i=15}\sum_{j=0}^{j=15}\left|A_{(x+i,y+j)} - B_{((x+r)+i,(y+s)+j)}\right| \qquad \textbf{(Equação 2)}$$

onde *(x,y)* é a coordenada do bloco de imagem analisado dentro do quadro de vídeo e *(r,s)* é o vetor de movimento relativo entre os bloco no quadro corrente *A* e o mesmo bloco no quadro de referência *B*. Os vetores de movimento calculados são então comparados a um valor limite. Caso o valor seja superior ao limite o movimento é detectado. Regiões da imagem em que foram detectados movimento não são confiáveis para serem comparadas com quadros anteriores, conseqüentemente o Atenuador não aplicará nelas a filtragem sugerida pelo Estimador de Ruído. A elevação da complexidade envolvida nesse subsistema, em relação aos dois anteriores, é notável. O estimador precisa varrer dois quadros de imagem ao mesmo tempo e executar a varredura em regiões. Além disso, as considerações feitas no Estimador de Ruído que permitiram a simplificação do algoritmo não podem ser feitas nesse subsistema. O detector de movimento precisa levar

em conta todos os componentes do espaço de cores da imagem. Isso tudo implica num tráfego muito mais elevado de dados da memória para o bloco: é necessário colher componentes Y, U e V de dois quadros de vídeo, enquanto apenas o componente Y de um quadro é necessário no Estimador de Ruído. Considerando-se que os quadros de vídeo estavam armazenados em memórias fisicamente distintas, cada uma contendo os três componentes de cor da imagem, o Estimador de Movimento precisava fazer ao menos três acessos a cada memória por *pixel* varrido. Ao mesmo tempo, a memória que continha o quadro de referência era um componente externo ao FPGA, o que impôs tempos de latência na comunicação bem superiores aos encontrados no Estimador de Ruído. Dessa forma, a exploração arquitetural desse algoritmo requeriu um esforço superior ao dispendido na síntese dos dois primeiros. Assim como no sub-sistema anterior, o principal ganho de desempenho foi obtido pelo emprego de *pipeline* no laço principal de varredura dos quadros. A otimização das partições do código foi necessária, e nela encontrou-se um acesso múltiplo à memória desnecessário, que não foi detectado pela ferramenta. Utilizou-se ainda de otimização de todos os laços de repetição do código e de otimização dos recursos. Após a realização de todas as otimizações previstas no fluxo, salvo a reestruturação do código, foi possível atingir a meta de 12 quadros por segundo, conforme exposto na Tabela 3.

Todos os algoritmos foram sintetizados e prototipados em uma plataforma TB-5V-LX330-EX [288] da Inrevium (Tokyo Electron Device Limited), contendo um FPGA Xilinx Virtex-5 com 330 mil células lógicas (XC5VLX330-2FFG1760). A geração das *netlists* de configuração do FPGA foi feita utilizando-se o Synplify Pro [289] da Synplicity, a partir dos códigos RTL sintetizados pelo Catapult. Esta ferramenta mostrou-se superior ao ISE, ferramenta nativa do fabricante do FPGA, consumindo aproximadamente um décimo do tempo na síntese e gerando *netlists* de desempenho cerca de 30% superior.

Uma comparação entre os resultados obtidos pelo fluxo RTL clássico e a abordagem proposta é apresentada na Tabela 3 A vazão foi calculada em termos de quadros por segundo (fps) para um vídeo em cores (padrão YUV), 24 bits por *pixel*, na resolução padrão de TV (SDTV), isto é, 720 × 480 *pixels*. Os valores da abordagem RTL clássica

apresentados referem-se aos tempos e resultados obtidos por uma equipe de cinco engenheiros da empresa no desenvolvimento da versão comercial do mesmo sistema.

**Tabela 3 - Comparação entre os resultados obtidos pelo fluxo RTL clássico e o fluxo proposto**

| | **Atenuador** | **Estimador de Ruído** | **Preditor e Edtimador de Movimento** |
|---|---|---|---|
| | **Abordagem RTL Clássica** | | |
| **Max. Freqüência de Relógio (MHz)** | 300 | 220 | 100 |
| **Vazão em quadros por segundo (fps)** | 740 | 232 | 30 |
| **Tempo de desenvolvimento** | 28 dias | 34 dias | 45 dias |
| **Engenheiros envolvidos** | 5 | 5 | 5 |
| | **Abordagem Proposta** | | |
| **Max. Freqüência de Relógio (MHz)** | 220 | 113 | 80 |
| **Vazão em quadros por segundo (fps)** | 542 | 135 | 12 |
| **Tempo de desenvolvimento** | 3 dias | 4 dias | 6 dias |
| **Engenheiros envolvidos** | 2 | 2 | 2 |

De uma maneira geral, é possível notar que os resultados obtidos através do modelo proposto são inferiores aos obtidos pelo fluxo RTL clássico. Entretanto, o tempo de desenvolvimento foi drasticamente reduzido para todos os modelos avaliados, mesmo envolvendo menos da metade dos engenheiros, o que atende perfeitamente aos objetivos da metodologia proposta neste trabalho.

# 6. Conclusões e Perspectivas Futuras

Uma metodologia para acelerar a prototipagem em FPGA de algoritmos de processamento de vídeo foi estabelecida. Para isso, formalizou-se um fluxo alternativo de projeto, contendo métodos e regras para conduzir o projetista a sintetizar um protótipo do sistema de forma satisfatória, com esforços e tempo notavelmente reduzidos em comparação aos necessários empregando o fluxo de projeto RTL clássico.

Partindo do algoritmo em C/C++, o novo fluxo de projeto explora as particularidades dos sistemas de processamento de vídeo, bem como as possibilidades oferecidas pelas ferramentas de síntese de alto nível.

Uma vez desenvolvido o fluxo proposto foi testado e avaliado com sucesso dentro da divisão de vídeo da STMicroelectronics em Grenoble, na França. A avaliação envolveu algoritmos extraídos de sistemas comerciais desenvolvidos pela empresa, reforçando o aspecto prático da aplicabilidade da metodologia.

Mesmo apresentando características de desempenho inferiores às obtidas pelo fluxo clássico de projeto RTL, o protótipo gerado através dessa metodologia atendeu a todas as metas pré-estabelecidas, produzindo um protótipo funcional do sistema em FPGA em aproximadamente um oitavo do tempo do fluxo clássico de projeto RTL, empregando para isso menos da metade dos engenheiros.

Apesar da proposta visar, num primeiro momento, apenas algoritmos de tratamento de vídeo, espera-se que a metodologia utilizada na determinação do fluxo de síntese também possa ser aplicada, em trabalhos futuros, na determinação de fluxos dedicados a outras classes de algoritmos, gerando assim meios de se extrair o máximo desempenho possível das ferramentas de síntese de alto nível para cada aplicação.

Espera-se também que o fluxo proposto possa ser futuramente integrado à ferramentas de modelagem de mais alto nível, tal como o Simulink, que permite a modelagem gráfica (em diagrama de blocos) e a geração do algoritmo equivalente em C,

linguagem que alimenta o fluxo. Finalmente, acredita-se que uma evolução desta metodologia poderá também ser aplicada diretamente à síntese em silício.

# 7. REFERÊNCIAS


[1] IEEE, *"IEEE Std 1012-2004 Standard for Software Verification and Validation"*, Padrão IEEE, 2004.

[2] Altera Corporation, *"Stratix IV Device Handbook"*, Vol. 1, disponível em http://www.altera.com/products/devices/stratix-fpgas/stratix-iv/stxiv-index.jsp, Março de 2009.

[3] Xilinx Corporation, *"Virtex-6 Family Overview"*, Disponível em <http://www.xilinx.com/products/v6s6.htm>, Acessado em Fevereiro de 2009.

[4] Rocha, A. K., Lira, P., Ju, Y. Y., Barros, E., Melcher E., Araújo G., *"Silicon Validated IP Cores Designed by the Brasil-IP Network"*, IP/SOC 2006.

[5] Kang, H-Y., Jeong, K-A., Bae, J-Y., Lee, Y-S., Lee, S-H., *"MPEG4 AVC/H.264 Decoder with Scalable Bus Architecture and Dual Memory Controller"*, Proceedings of the International Symposium on Circuits and Systems (ISCAS'04), Vancouver, Canada: Vol 2, pp. 145, Maio de 2004.

[6] Fullhan Microelectronics, "FH8602 H.264 HD Decoder", Product Brochure, Setembro de 2005.

[7] Coussy, P., Morawiec, A., *"High-Level Synthesis, From Algorithm to Digital Circuit"*, Springer, Agosto de 2008.

[8] Wakabayashi, K., Okamoto, T., *"C-Based SoC Design Flow and EDA Tools: An ASIC and System Vendor Perspective"*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 19, No. 12, pp. 1507-1522, Dezembro de 2000.

[9] Kundu S., Lerner S., Gupta R., *"Validating High-Level Synthesis"*, 20th International Conference, CAV 2008 Princeton, NJ, USA, pp. 459-472, Julho de 2008.

[10] Hosseini, A., Parikh, A., Chin, H. T., Urard, P., Girczyc, E., Bloch, S., *"Building a Standard ESL Design And Verification Methodology: Is it Just a Dream?"*, Proceedings of the 43rd annual conference on Design automation (DAC '06), pp. 370-371, San Francisco, CA, Estados Unidos, 2006.

[11] International Technology Roadmap for Semiconductors (ITRS), *"Design"*, San Jose, CA, Estados Unidos, 2007.

[12] Makhijani, H., Meier, S., *"A High Level Design Solution for FPGA's"*, Proceedings of IEEE WESCON/94, California, Estados Unidos, 1994.

[13] Klingauf, W., Gädke, H., and Günzel, *"TRAIN: A Virtual Transaction Layer Architecture for TLM-Based HW/SW Codesign of Synthesizable MPSoC"*, *Proceedings of the Conference on Design, Automation and Test in Europe: Proceedings,* Belgium, pp. 1318-1323, Março 2006.

[14] Viaud, E. Pêcheux, F., *"A New Paradigm and Associated Tools for TLM/T Modeling of MPSoCs"*, IEEE Research in Microelectronics and Electronics, Ph.D, Otranto, pp. 217-220, 2006.

[15] Huss, S.A., *"Advances in Design and Specification Languages for Embedded Systems"*, Springer, 2007.

[16] Mentor Graphics Corporation, *"Catapult Synthesis - Automated Verification Extension Datasheet"*, 2006.

[17] Synfora Inc., *"PICO Extreme – The Most Advanced Algorithmic Synthesis Enabling the Most Efficient Hardware for Complex Algorithms"*, Product Datasheet, 2009.

[18] Forte Design Systems, *"Cynthesizer Datasheet"*, Product Brochure, disponível em http://www.forteds.com/products/cynthesizer.asp, 2008.

[19] Cadence Design Systems Inc, *"Cadence C2Silicon Compiler Datasheet"*, Product Brochure, 2008.

[20] Pecheux, F., Lallement, C., Vachoux, A., *"VHDL-AMS and Verilog-AMS as Alternative Hardware Description Languages for Efficient Modeling of Multidiscipline Systems"*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Volume 24, Issue 2, pp. 204-225, Fevereiro de 2005.

[21] Nikitin P.V., Richard Shia C.-J., *"VHDL-AMS Based Modeling and Simulation of Mixed-Technology Microsystems: A Tutorial"*, the VLSI Journal Integration, Volume 40, Issue 3, pp. 261-273, Abril 2007.

[22] Smith, M.J.S., *"Application-Specific Integrated Circuits"*, MA, Addison-Wesley, 1997.

[23] Quirk, M., J. Serda., *"Semiconductor Manufacturing Technology"*, Prentice-Hall, Upper Saddle River, NJ, 2001.

[24] Crouch, A.L., *"Design-for-Test for Digital IC's and Embedded Core Systems"*, Englewood Cliffs, NJ, Prentice-Hall, 1999.

[25] Weste, N.H.E., Harris, D., *"CMOS VLSI Design A Circuits and Systems Perspective"*, Addison-Wesley, 2005.

[26] Chinnery, D., Keutzer, K., *"Closing the Gap Between ASIC & Custom Tools and Techniques for High-Performance ASIC Design"*, Norwell, MA, Kluwer, 2002.

[27] Kuon, I., Rose, J., *"Measuring the Gap between FPGAs and ASICs"*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 26, No. 2, pp. 203-215, Sonoma, CA, Estados Unidos, Fevereiro de 2007.

[28] Wu, K.-C., Tsai, Y.-W., *"Structured ASIC, Evolution or Revolution?"*, Proceedings of the International Symposium on Physical Design (ISPD), Abril de 2004.

[29] Keutzer, K., Malik, S., Newton, A.R., *"From ASIC to ASIP: The next design discontinuity"*, in Proceedings of International Conference Computer Design, pp. 84-90, Setembro de 2002.

[30] Sasongko, A., *"Prototypage basé sur une plateforme reconfigurable pour vérif des systèmes monopuces"*, Tese de Doutorado, Université Joseph-Fourier, Grenoble, França, 15 de Outubro de 2004.

[31] Hamblen, J.O., Hall, T.S., *"Rapid Prototyping of Digital Systems: Sopc Edition"*, Segunda Edição, Springer, 2008.

[32] Chen, D.C., *"Programmable Arithmetic Devices for High Speed Digital Signal Processing"*, Tese de Doutorado, University of California, Berkeley, Estados Unidos, 1992.

[33] Hartenstein, R., *"A decade of reconfigurable computing: a Visionary Retrospective"*, In Proceedings of the Conference on Design, Automation and Test in Europe, pp. 642-649, Munich, Germany, 2001.

[34] Vahid, F., Givargis, T., *"Embedded Systems Design: A Unified Hardware/Software Introduction"*, John Wiley & Sons, 2002.

[35] International Data Corporation, *"IDC Reveals Precipitous Decline of the Worldwide PC Microprocessor Market in 4Q08 Will Continue Through 1H09"*, Disponível em http://www.idc.com/getdoc.jsp?containerId=prUS21672009, 11 de Fevereiro de 2009.

[36] Bove, V.M.Jr, Watlington, J.A., *"Cheops: A Reconfigurable Data-Flow System for Video Processing"*, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 5, No. 2, pp. 140-149, 5 de Abril de 1995.

[37] Diefendorff, K., Dubey, P.K., *"How Multimedia Workloads Will Change Processor Design"*, IEEE Computer, Vol. 30, No. 9, pp. 43-45, Setembro de 1997.

[38] Lee R., *"Accelerating Multimedia with Enhanced Microprocessors"*, IEEE Micro, vol. 15, no. 2, pp.22-32, Abril de 1995.

[39] Trembley M., O'Connor M., Narayanan V., He L., *"VIS Speeds New Media Processing"*, IEEE Micro, vol. 16, no. 4, pp. 10-20, Agosto de 1996.

[40] Peleg A. and Weiser U., *"MMX Technology Extension to the Intel Architecture"*, IEEE Micro, vol. 16 no. 4, pp. 42-50, August 1996.

[41] Ramanathan, R.M., *"Extending the World's Most Popular Processor Architecture"*, Intel Architecture White Paper, 2007.

[42] Zhou, X., Li, E.Q., Chen, Y.-K., *"Implementation of H.264 Decoder on General-Purpose processors with Media Instructions"*, Proceedings of SPIE Conference on Image and Video Communications and Processing 2003, 2003.

[43] Schösser, A., Geiß, R., *"Graph Rewriting for Hardware Dependent Program Optimizations"*, Applications of Graph Transformations with Industrial Relevance, pp. 233-248, SpringerLink, 10.1007/978-3-540-89020-1_17, 2008.

[44] Shahbahrami, A., Juurlink, B., *"Optimization of Content-Based Image Retrieval Functions"*, Tenth IEEE International Symposium on Multimedia, 2008. ISM 2008, pp. 607-612, Berkeley, CA, Estados Unidos, 15-17 de Dezembro de 2008.

[45] Hu, Y.H., *"Programmable Digital Signal Processors: architecture, programming, and applications"*, CRC Press, 2001.

[46] Lee, E.A., *"Programmable DSP architectures: Part I"*, IEEE ASSP Magazine, Vol. 5, No. 4, pp. 4–19, Outubro de 1988.

[47] Kuroda, I., Nishitani, T., *"Multimedia Processors"*, Proceedings of IEEE, Vol 86, No. 6, pp. 1203–1221, Junho de 1998.

[48] Hoare, R., Jones, A.K., Kusic, D., Fazekas, J., Foster, J., Tung, S., McCloud, M., *"Rapid VLIW Processor Customization for Signal Processing Applications Using*

*Combinational Hardware Functions"*, EURASIP Journal on Applied Signal Processing, pp. 67-67, 2005.

[49] Peleg, A., Weiser, U., *"MMX Technology Extension to the Intel Architecture"*, IEEE Micro, pp. 42–50, Agosto de 1996.

[50] Agarwala, S., Anderson, T., Hill, A., Ales, M.D., Damodaran, R., Wiley, P., Mullinnix, S., Leach, J., Lell, A., Gill, M., Rajagopal, A., Chachad, A., Agarwala, M., Apostol, J., Krishnan, M., Bui, D., An, Q., Nagaraj, N.S., Wolf, T., Elappuparackal, T.T., *"A 600-MHz VLIW DSP"*, IEEE Journal of Solid-State Circuits (JSSC), Vol. 37, No. 11, pp. 1532–1544, Novembro de 2002.

[51] Smith, S.W., *"The Scientist and Engineer's Guide to Digital Signal Processing"*, 2a Edição, California Technical Publishing, San Diego, California, Estados Unidos, 1999.

[52] Huang, Z., Wang, T., *"MDPCM picture coding"*, Proceedings of the IEEE International Symposium on Circuits and Systems, Vol. 4, pp. 3253–3255, New Orleans, Louisiana, Estados Unidos, 1-3 de Maio de 1990.

[53] Normile, J., Wright, D., *"Image Compression Using Coarse Grain Parallel Processing"*, Proceedings of the 1991 International Conference on Acoustics, Speech, and Signal Processing, Vol. 2, pp. 1121–1124, Toronto, Ontario, Canada, 14-17 de Abril de 1991.

[54] Sa, L., Silva, V., Cruz, L., Faria, S., Amado, P., Navarro, A., Lopes, F., Silvestre, J., *"DSP-based Hardware for Real-time Video Coding"*, Proceedings of the SPIE Conference on Image Processing and Interchange: Implementation and Systems, Vol. 1659, pp. 99–104, San Jose, California, 12–14 de Fevereiro de 1992.

[55] CCITT, *"Video Codec for Audio Visual Services at px64 kbits/s"*, CCITT Recommendation H.261, Disponível em http://www.itu.int/rec/T-REC-H.261, Março de 1993.

[56] Texas Instruments, *"3Q2008 - Video and Image Guide"*, Guia de Produtos, Texas Instruments Incorporated, Dallas, Texas, Estados Unidos, Disponível em http://www.ti.com/home_a_vi, 2008.

[57] Analog Devices, *"Embedded Processing and DSP"*, Analog Devices, Guia de Seleção de Produtos, Disponível em http://www.analog.com/en/embedded-processing-dsp/sharc/content/index.html, 6 de Maio de 2009.

[58] Freescale Semiconductor Inc., *"DSP56301 - 24-Bit Digital Signal Processor"*, Dados Técnicos de Componente, Freescale Semiconductor, Rev. 10, Disponível em http://www.freescale.com/files/dsp/doc/data_sheet/DSP56301DS.pdf , Julho de 2006

[59] Foley, P. ,*"The Mpact Media Processor Redefines the Multimedia PC"*, in Proceedings of Compcon. New York: IEEE Computer Science Press, pp. 311–318, 25-28 de Fevereiro 1996.

[60] Rathnam , S., Slavenburg, G., *"An Architectural Overview of the Programmable Multimedia Processor"*, Digest of Papers Compcon '96. 'Technologies for the Information Superhighway', pp. 319-326, Santa Clara, CA, Estados Unidos, 25-28 de Fevereiro de 1996.

[61] Texas Instruments Inc., *"TMS320DM36x Digital Media System-on-Chip (DMSoC) ARM Subsystem User's Guide"*, Disponível em http://www.ti.com/litv/pdf/sprufg5, Abril de 2009.

[62] Oshima, Y., Sheu, B.J., Jen, S.H., *"High-speed Memory Architectures for Multimedia Applications"*, IEEE Circuits and Devices Magazine, vol. 13, No. 1, pp. 8–13, Janeiro de 1997.

[63] Kuroda, I., Nishitani, T., *"Multimedia Processors"*, Proceedings of the IEEE, Vol. 86, No. 6, pp. 1203-1221, Junho de 1998.

[64] Gries, M., *"The Impact of Recent DRAM Architectures on Embedded Systems Performance"*, Proceedings of the 26th Euromicro Conference, Vol. 1, pp. 282-289, Maastricht, Holanda, 5-7 de Setembro de 2000.

[65] Texas Instruments Inc., *"TMS320DM365 - Digital Media System-on-Chip (DMSoC)"*, Product Preview Datasheet, Texas Inst., Dallas, Texas, Estados Unidos, Disponível em http://focus.ti.com/docs/prod/folders/print/tms320dm365.html, 2009.

[66] International Telecommunication Union, *"ITU-T H.264 - Advanced Video Coding for Generic Audiovisual Services"*, ITU-T Recommendation, Novembro de 2007.

[67] Texas Instruments Inc., *"New TMS320DM365 Digital Media Processor Provides Pixel-Perfect HD H.264"*, Disponível em http://www.ti.com/corp/docs/landing/davinci/firstproducts.html, 6 de Maio de 2009.

[68] Minami, T., Kasai, R., Yamauchi, H., Tashiro, Y., Takahashi, J., Date, S., *"A 300-MPOS Video Signal Processor with a Parallel Architecture"*, IEEE Journal of Solid-State Circuits, Vol. 26, No. 12, pp. 1868-1875, Dezembro de 1991.

[69] Inoue, T., Goto, J., Yamashina, M., Suzuki, K., Nomura, M., Koseki, Y., Kimura, T., Atsumo, T., Motomura, M., Shih, B.S., Horinchi, T., Hamatake, N., Kumagai, K., Enomoto, T., Yamada, H., Takada, M., *"A 300MHz 16b BiCMOS Video Signal Processor"*, IEEE Journal of Solid-State Circuits, Vol. 28, No. 12, pp. 1321–1330, Dezembro de 1993.

[70] Aono, K., Toyokura, M., Araki, T., Ohtani, A., Kodama, H., Okamoto, K., *"A Video Digital Signal Processor with a Vector-pipeline Architecture"*, IEEE Journal of Solid-State Circuits, vol. 27, no. 12, pp. 1886–1894, Dezembro de 1992.

[71] Goto, J., Ando, K., Inoue, T., Yamashina, M., Yamada, H., Enomoto, T., *"250-MHz BiCMOS Super-high-speed Video Signal Processor (S-VSP) ULSI"*, IEEE Journal of Solid-State Circuits, Vol. 26, No. 12, pp. 1876–1884, Dezembro 1991.

[72] Tamitani, I., Harasaki, H., Nishitani, T., *"A Real-time HDTV Signal Processor: HDVSP"*, IEEE Transactions on Circuits and Systems for Video Technology, vol. 1, no. 1, pp. 35–41, Março de 1991.

[73] Freescale Semiconductor Inc., *"MC149570 - Multi-standard Video Processor"*, Dados Técnicos de Produto, Rev. 2, 1999.

[74] Chen, D. C., Rabaey, J. M., *"PADDI: Programmable Arithmetic Devices for Digital Signal Processing"*, VLSI Signal Processing IV, pp. 240-249. Novembro de 1990.

[75] Chen, D.C., Guerra, L.M., Ng, E.H., Potkonjak, M., Schultz, D.P., Rabaey, J.M., *"An Integrated System for Rapid Prototyping of High Performance Algorithm Specific Data Paths"*, In Proceedings of the International Conference on Application Specific Array Processors, pp. 134-148, Berkeley, CA, USA, Aug. 4-7, 1992.

[76] Chen, D.C., Rabaey, J.M, *"A Reconfigurable Multiprocessor IC for Rapid Prototyping of Algorithmic-Specific High-Speed DSP Data Paths"*, IEEE Journal of Solid-State Circuits, Vol. 27, No 12, pp. 1895-1904, Dezembro de 1992.

[77] Hilfinger, P., *"A High Level Language and Silicon Compiler for Digital Signal Processing"*, In Proc. IEEE Custom Integrated Circuits Conference, pp. 240-243, Maio de 1985.

[78] Yeung, A.K.W., Rabaey, J.M., *"A Reconfigurable Data-Driven Multiprocessor Architecture for Rapid Prototyping of High Throughput DSP Algorithms"*, Proceeding of the Twenty-Sixth Hawaii International Conference on System Sciences, Vol.1, pp. 169-178, Jan 5-8, 1993.

[79] Rabaey, J.M., *"Reconfigurable Processing: The Solution to Low-Power Programmable DSP"*, Proceedings of the 1997 IEEE international Conference on Acoustics, Speech, and Signal Processing (ICASSP '97), Vol. 1, pp. 275, Munich, Alemanha, 21-24 de Abril de 1997.

[80] Miyamori, T., Olukotun, K., *"REMARC: Reconfigurable Multimedia Array Coprocessor"*, Proceedings of ACM/SIGDA FPGA'98, Monterey, Feb. 1998.

[81] Hartenstein, R.W., Kress, R., *"A Datapath Synthesis System for the Reconfigurable Datapath Architecture"*, Asia and South Pacific Design Automation Conference, ASPDAC'95, Nippon Convention Center, Makuhari, Chiba, Japão, 29 Agosto a 1˚ de Setembro de 1995.

[82] Hartenstein, R., Herz, M., Hoffmann, T., and Nageldinger, U., *"KressArray Xplorer: A New CAD Environment to Optimize Reconfigurable Datapath Array Architectures"*, In Proceedings of the 2000 Conference on Asia South Pacific Design Automation, pp. 163-168, Yokohama, Japão, 25-28 de Janeiro de 2000.

[83] Mirsky, E., DeHon, A., *"MATRIX: A Reconfigurable Computing Architecture with Configurable Instruction Distribution and Deployable Resources"*, IEEE Symposium on FPGAs for Custom Computing Machines, pp. 157-166, Napa Valley, California, Estados Unidos, 17-19 de Abril de 1996.

[84] Loghi, M., Angiolini, F., Bertozzi, D., Benini, L., Zafalon, R., *"Analyzing On-Chip Communication in a MPSoC Environment"*, Proceedings of Design, Automation and Test in Europe Conference and Exhibition, Vol. 2, pp.752-757, 16-20 de Fevereiro de 2004.

[85] Jerraya A.A., Wolf, W., *"Why MPSoCs?"*, Multiprocessors Systems-on-Chips, Morgan Kaufmann, pp. 1-18, 2005.

[86] Irwin, M.J., Benini, L., Vijaykrishnan, N., Kandemir, M., *"Techniques for Designing Energy-Aware MPSoCs"*, Multiprocessors Systems-on-Chips, Morgan Kaufmann Publishers, pp. 21-48, 2005.

[87] Wolf, W., *"The Future of Multiprocessor Systems-on-Chips"*, Proceedings of the 41st Annual Conference on Design Automation, pp. 681-685, San Diego, CA, Estados Unidos, 07-11 de Junho de 2004.

[88] Benini, L., De Micheli, G., *"Powering networks on chips"*, Proceedings of the 14th International Symposium on System Synthesis, pp. 33-38, 2001.

[89] Dey, S., Lahiri, K., Raghunathan, A., *"Design of Communication Architectures for High-Performance and Energy Efficient Systems-on-Chips"*, Multiprocessors Systems-on-Chips, pp. 187-222, Morgan Kaufmann Publishers, 2005.

[90] Soteriou, V., Eisley, N., Peh, L-S., *"Software-directed Power-aware Interconnection Networks"*, ACM Transactions on Architecture and Code Optimization, Vol. 4, No. 1, Art. 5, Março de 2007.

[91] J. Hu, G. Chen, M. Kandemir, and N. Vijaykrishnan, *"Software Power Optimisation"*, System On Chip: Next Generation Electronics, pp. 289-316, 2006.

[92] Wolf, W., *"Multiprocessor Systems-on-Chips"*, IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures, pp. 4-8, 2-3 de Março de 2006.

[93] Bertozzi, D., Jalabert, A., Murali, S., Tamhankar, R., Stergiou, S., Benini, L., De Micheli, G., *"NoC Synthesis Flow for Customized Domain Specific Multiprocessor Systems-on-Chip"*, IEEE Transactions on Parallel and Distributed Systems, Vol. 16, No. 2, pp. 113–129, Fevereiro de 2005.

[94] Zeferino, C.A., Kreutz, M.E., Carro, L., Susin, A.A., *"A Study on Communication Issues for Systems-on-Chip"*, Proceedings of 15th Symposium on Integrated Circuits and Systems Design, pp. 121-126, Setembro de 2002.

[95] Goering, R., *"Platform-based design: A choice, not a panacea"*, EE Times, Disponível em http://www.eetimes.com/reshaping/platformdesign/ OEG20020911S0061, 11 de Setembro de 2002.

[96] Xilinx Inc., *"Celebrating 20 Years of Innovation"*, Xilinx Xcell Journal, Vol. 48, pp. 14-16, 2004.

[97] Xilinx Inc., *"Our History"*, Disponível em http://www.xilinx.com/company/history.htm, Maio 2009.

[98] Carter, W.S., Duong, K., Freeman, R.H., Hsieh, H.C., Ja, J.Y., Mahoney, J.E., Ngo, L.T., Sze, S.L., *"A User Programmable Reconfigurable Logic Array"*, IEEE 1986 Custom Integrated Circuits Conference, pp. 233-235, 1986.

[99] Brown, S., Rose, J., *"FPGA and CPLD Architectures: A Tutorial"*, IEEE Design & Test of Computers, Vol. 13, No. 2, pp. 42-57, 1996.

[100] Vuillemin, J.E., Bertin, P., Roncin, D., Shand, M., Touati, H.H., Boucard, P., *"Programmable Active Memories: Reconfigurable Systems Come of Age"*, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 4, No. 1, pp. 56-69, Março de 1996.

[101] S. Brown, *"FPGA Architectural Research: a Survey"*, IEEE Design & Test of Computers, Vol. 13, No. 4, pp. 9–15, 1996.

[102] Lattice Semiconductor Corp., *"LatticeXP2 Family Handbook"*, Especificação Técnica de Produto, Ver. 2.3, Disponível em http://www.latticesemi.com/products/fpga/xp2/index.cfm, Janeiro de 2009.

[103] Actel Corporation, *"ProASIC3"*, Resumo de Dados do Produto, Disponível em http://www.actel.com/documents/PA3_E_PIB.pdf, Fevereiro de 2009.

[104] Atmel Corporation, *"Coprocessor Field Programmable Gate Arrays - AT6000(LV) Series"*, Especificação Técnica de Produto, Disponível em http://www.atmel.com/, Outubro de 1999.

[105] QuickLogic Corporation, *"Eclipse™ II Family Data Sheet"*, Especificação de Dados Tecnicos de Produto, Rev. R, Disponível em http://www.quicklogic.com/images/eclipse2_family_DS.pdf, 31 de Agosto de 2007.

[106] Actel Corporation, *"Actel IGLOO® Low-Power Flash FPGAs"*, Resumo de Dados do Produto, Disponível em http://www.actel.com/documents/IGLOO_PIB.pdf, Abril de 2009.

[107] Achronix Semiconductor Corporation, *"Speedster™ FPGA Family"*, Resumo de Dados de Produto, Ver. 3.5, Disponível em http://www.achronix.com/serve_doc.php?id=Speedster_Product_Brief_PB001.pdf, 2009.

[108] MathStar Inc., *"Arrix Family FPOA Architecture Guide"*, Guia de Produtos, Disponível em http://mathstartest.com/Products.php, 18 de Maio de 2007.

[109] MathStar Inc., *"Arrix FPOA Overview"*, Ver. 1.7, Dados de Produto, Disponível em http://mathstartest.com/Products.php, Abril de 2008.

[110] MathStar Inc., *"Arrix FPOA Design Flow"*, Ver. 1.6, Resumo de Produto, Disponível em http://mathstartest.com/Tools.php, Abril de 2008.

[111] MathStar Inc., *"MPEG-2 Multi-stream Decoder for Arrix FPOA"*, Ver. 2.7, Resumo de Produto, Disponível em http://mathstartest.com/Video.php, Abril de 2008.

[112] Riley, S., Moll, J.L., *"Implementing the High Profiles of MPEG-2 & MPEG-4/ H.264 on an FPOA Architecture"*, TechOnline, On-Demand Webinar, Disponível em http://www.techonline.com/learning/webinar/201802955/, 7 de Maio de 2009.

[113] Chiang P., Riley, S., "*Using a Field Programmable Object Array (FPOA) to Accelerate Image Processing"*, Real-Time Image Processing Conference, San Jose, CA, Estados Unidos, DOI:10.1117/12.649314, 2006.

[114] Tripp, J.L., Frigo, J., Graham, P., *"A Survey of Multi-Core Coarse-Grained Reconfigurable Arrays for Embedded Applications"*, Los Alamos National Labs http://www.ll.mit.edu/HPEC/agendas/ proc07/Day3/08_Tripp_Abstract.pdf, 2 de Maio de 2008.

[115] Dauman, A., Seynhaeve, D., *"A methodology for DSP-based FPGA design"*, Disponível em http://www.eetimes.com/news/design/showArticle.jhtml?articleID= 61800027, 05 de Maio de 2005.

[116] Snyder, C., *"FPGA Processor Cores Get Serious"*, Cahners Microprocessor Report, Disponível em http://www.mpronline.com/, Setembro de 2000.

[117] Hall, T.S., Hamblen, J.O., *"System-on-a-Programmable-Chip Development Platforms in the Classroom"*, IEEE Transactions on Education, Vol. 47, No. 4, pp. 502-507, Novembro de 2004.

[118] Altera Corporation, *"Avalon Interface Specifications"*, Ver. 1.2, Altera Corporation, Disponível em http://www.altera.com/literature/manual/mnl_avalon_spec.pdf, Abril de 2009.

[119] Xilinx Inc, *"Processor Local Bus (PLB) v4.6 (v1.00a)"*, Product Specification, Disponível em http://www.xilinx.com/support/documentation/ip_documentation/plb_v46.pdf, 9 de Agosto de 2007.

[120] Christophersen, H.B., Pickell, R.W., Neidhoefer, J.C., Koller, A.A., Kannan, S.K., Johnson, E.N., *"A Compact Guidance, Navigation, and Control System for Unmanned Aerial Vehicles"*, Journal of Aerospace Computing, Information, and Communication, pp. 1542-9423, vol.3 no.5, 2006.

[121] Marshall, A., Stansfield, T., Kostarnov, I., Vuillemin, J., and Hutchings, B., *"A Reconfigurable Arithmetic Array for Multimedia Applications"*, Proceedings of the 1999 ACM/SIGDA Seventh international Symposium on Field Programmable Gate Arrays, Monterey, California, Estados Unidos, 21-23 de Fevereiro de 1999.

[122] Hutchings, B.L., Nelson, B.E., *"Using General-purpose Programming Languages for FPGA Design"*, Proceedings of 37th Design Automation Conference, pp. 561-566, Los Angeles, Junho de 2000.

[123] Fernandes, J.H.C., *"Introdução à Engenharia de Software e Sistemas"*, DIMAp-UFRN, Disponível em http://www.cic.unb.br/~jhcf/MyBooks/, Janeiro de 2004.

[124] Cong, J., Preas, B., *"A New Algorithm for Standard Cell Global Routing"*, IEEE International Conference on Computer-Aided Design, pp. 176-179, Novembro de 1988.

[125] Guruswamy, M., Maziasz, R.L., Dulitz, D., Raman, S., Chiluvuri, V., Fernandez, A., Jones, L.G., *"CELLERITY: A Fully Automatic Layout Synthesis System for Standard Cell Libraries"*, Proceedings of the 34th Design Automation Conference, pp. 327-332, 1997.

[126] Xilinx Corporation, *"Virtex-5 FPGA User Guide"*, UG190 (v4.5), Disponível em http://www.xilinx.com/support/documentation/virtex-5.htm, Janeiro de 2009.

[127] Intel Pressroom, *"The Evolution of a Revolution"*, Disponível em http://download.intel.com/pressroom/kits/IntelProcessorHistory.pdf, 3 de Março de 2009.

[128] Wolf, W.H., *"Hardware-Software Co-design of Embedded Systems"*, Proceedings of the IEEE, Vol. 82, No. 7, pp. 967-989, Julho de 1994.

[129] De Micheli, G., Ernst, R., Wolf, W., *"Readings In Hardware/Software Co-Design"*, The Morgan Kaufmann Series in Systems on Silicon, Number ISBN:1-55860-702-1, Elsevier, 2002.

[130] Sander, I., Jantsch, A., *"System Modeling and Transformational Design Refinement in ForSyDe"*, IEEE transaction on : Computer-Aided Design of Integrated Circuits and Systems, Vol. 23, No. 1, pp. 17-32, Janeiro de 2004.

[131] Dumoulin, C., Dekeyser, J-L., *"UML Framework for Intensive Signal Processing Embedded Applications"*, Relatório de Pesquisa, Laboratoire d'Informatique Fondamentale de Lille, Université de Lille 1, França, Julho de 2002.

[132] Moullec, Y.Le, Diguet, J-Ph., Gourdeaux, T., Philippe, J-L., *"Design Trotter: System-Level Dynamic Estimation Task a 1st step towards platform architecture selection"*, Journal of embedded computing (JEC), Vol. 1, No. 4, Dezembro de 2005.

[133] Wolf, W.H., *"A Decade of Hardware/Software Codesign"*, IEEE Computer, Vol. 36, No. 4, pp. 38-43, Abril de 2003.

[134] Yanbing Li, Wolf, W.H., *"Hardware/Software Co-Synthesis with Memory Hierarchies"*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 18, No. 10, pp. 1405-1417, Outubro de 1999.

[135] Buck, J.T., Ha, S., Lee, E.A., Messerschmitt, D.G., *"Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems"*, International Journal of Computer Simulation, special issue on "Simulation Software Development", vol. 4, pp. 155-182, 31 de Agosto de 1992.

[136] Bhattacharyya, S.S., Buck, J.T., Ha, S., Lee, E.A., *"Generating Compact Code From Dataflow Specifications Of Multirate Signal Processing Algorithms"*, IEEE Transactions on Circuits and Systems — I: Fundamental Theory and Applications, Vol. 42, pp. 138-150, 1995.

[137] D. Hommais, *"Une méthode d'évaluation et de synthèse des communications dans les systèmes intégrés matériel-logiciel"*, Tese de Doutorado, Université Pierre et Marie Curie (Paris-IV), Paris, França, Setembro de 2001.

[138] Öberg, J., *"ProGram: A Grammar-Based Method for Specification and Hardware Synthesis of Communication Protocols"*, Tese de Doutorado, Royal Institute of Technology, Department of Electronics, ISSN 1104-8697, Stockholm, Suécia, 1999.

[139] Cesario, W.O., Nicolescu, G., Gauthier, L., Lyonnard, D., Jerraya, A.A., *"Colif: a Multilevel Design Representation for Application-Specific Multiprocessor System-on-Chip Design"*, IEEE Design & Test of Computers, Vol. 18, No. 5, pp. 8-20, Setembro/Outubro de 2001.

[140] Fraboulet, A., Risset, T., *"Efficient on-chip communications for data-flow IPs"*, Proceedings of 15th IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP 2004), pp. 293-303, Galveston, Texas, Estados Unidos, Setembro de 2004.

[141] CoFluent Design, Inc., *"CoFluent Studio™ - ESL Tool for Architecture Exploration and Performance Analysis"*, Product Datasheet, Disponivel em http://www.cofluentdesign.com, 1 de maio de 2009.

[142] Altera Corporation, *"Quartus II Handbook Version 9.0 - Volume 4: SOPC Builder"*, Disponível em http://www.altera.com/literature/hb/qts/qts_qii5v4.pdf, Março de 2009.

[143] Xilinx Inc, *"System Generator for DSP - Getting Started Guide"*, Disponível em http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/sysgen_gs.pdf, 27 de Abril de 2009

[144] Accelchip Incorporated, *"DSP Synthesis Datasheet"*, Application Note, 2005.

[145] Washington, L., *"Xilinx Acquires DSP Design Tool Leader Accelchip"*, Xilinx Press Release, No 613, Xilinx Inc., 13 de Janeiro de 2006.

[146] Synplify Inc., *"Synplify DSP"*, Sunnyvale, Synplicity Inc, Disponivel em http://www.synplicity.com/products/synplifydsp/index.html, 4 de Maio de 2009.

[147] Coware Inc., *"CoWare Signal Processing Designer - Implementing DSP Algorithms for Complex Wireless System Design"*, Product datasheet, CoWare, Disponível em http://www.coware.com/products/signalprocessing.php, 4 de Maio de 2009.

[148] Moretti, G., *"System-level Design Merits a Closer Look"*, EDN Magazine, 21 de Fevereiro de 2002.

[149] Maniwa, T., *"Electronic System-Level (ESL) Tools: A bolt-on to RTL or a new methodology?"*, Chip Design Magazine, pp. 17–21, Abril-Maio de 2004.

[150] Smith, G., Nadamuni, D., Balch, L., Wu, N., *"Market trends: Electronic Design Automation"*, Gartner/Dataquest, ID No. G00136302, 5 de Dezembro de 2005.

[151]Bailey, B., Martin, G., Piziali, A., *"ESL Design and Verification: A Prescription for Electronic System-Level Methodology"*, Primeira edição, Morgan Kaufmann Publishers, 2007.

[152]Ku D., Micheli. G.D., *"High-level Synthesis of ASICs Under Timing and Synchronization Constraints"*, Kluwer Academic Publishers, 1992.

[153]Shin, D., Gerstlauer, A., Dömer, R., Gajski, D.D, *"An Interactive Design Environment for C-based High-Level Synthesis"*, IFIP International Federation for Information Processing, Vol. 231, pp. 135-144, Springer, 2007.

[154]Mentor Graphics Corporation, *"Catapult Synthesis User's and Reference Manual, System-Level and Block-Level Products"*, Release 2007b_update1, Maio de 2008.

[155]Mentor Graphics, *"Catapult C Synthesis - C++ to Hardware Concepts - Release 2005a Update1"*, Setembro de 2005.

[156]Ravi, S., Raghunathan, A., Chakradhar, S., *"Efficient RTL Power Estimation for Large Designs"*, Anais do 16th International Conference on VLSI Design, pp. 431-439, Janeiro de 2003.

[157]Chen, D., Cong, J., Fan, Y., Zhang, Z., *"High-Level Power Estimation and Low-Power Design Space Exploration for FPGAs"*, *Proceedings of the 12th Asia and South Pacific Design Automation Conference (ASP-DAC 2007)*, Yokohama, Japan, pp. 529-534, Janeiro de 2007.

[158]Nemani, M., Najm, F. N., *"Towards a High-Level Power Estimation Capability [Digital ICs]"*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, pp. 588-598, Junho de 1996.

[159]Gupta, S., Najm, F. N., *"Power Modeling for High-Level Power Estimation"*, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, pp. 18-29, 2000.

[160]Gajski, D.D., Ramachandran, L., *"Introduction to High-level Synthesis"*, IEEE Design & Test of Computers, Vol. 11, No. 4, pp. 44-54, 1994.

[161]Chavet, C., *"Synthèse automatique d'interfaces de communication matérielles pour la conception d'applications du domaine du traitement du signal"*, Tése de Doutorado, Université de Bretagne Sud, Bretagne, França, 26 de Outubro de 2007.

[162]Orailoglu, A., Gajski, D.D., *"Flow Graph Representation"*, Proceedings of the 23rd ACM/IEEE conference on Design automation (DAC'86), pp. 503–509, Piscataway, NJ, Estados Unidos, 1986.

[163]Berrebi, E., Kission, P., Vernalde, S., De Troch, S., Herluison, J. C., Fréhel, J., Jerraya, A. A., and Bolsens, *"Combined Control Flow Dominated and Data Flow Dominated High-level Synthesis"*, In Proceedings of the 33rd Annual Conference on Design Automation, pp. 573-578, Las Vegas, Nevada, United States, June 03-07, 1996.

[164]Girkar, M.B., Polychronopoulos, C.D, *"Hierarchical Task Graph as a Universal Intermediate Representation"*, International Journal of Parallel Programming, Vol. 22, No. 5, pp. 519-551, 1994.

[165]Girkar, M.B., Polychronopoulos, C.D., *"The HTG: An Intermediate Representation for Programs Based on Control and Data Dependences"*, University of Illinois at Urbana-Champaign, CSRD Report 1046, Maio de 1991.

[166]Fan, K., Kudlur, M., Park, H., Mahlke, S., *"Increasing Hardware Efficiency with Multifunction Loop Accelerators"*, Proceedings of the 4th international conference on Hardware/software codesign and system synthesis (CODES+ISSS'06), pp. 276-281, Seoul, Korea, 22-25 de Outubro de 2006.

[167]Gupta, S., Gupta, R.K., Dutt, N.D., Nicolau, A., *"SPARK: A Parallelizing Approach to the High-Level Synthesis of Digital Circuits"*, Kluwer Academic Publishers, 2004.

[168]De Micheli, G., *"Synthesis and Optimization of Digital Circuits"*, McGraw-Hill, 1994.

[169]Jerraya, A.A., Ding, H., Kission, P., Rahmouni. M., *"Behavioral Synthesis and Component Reuse with VHDL"*, Springer, 1997.

[170]Lin, Y-L., *"Recent Developments in High-level Synthesis"*, ACM Transactions on Design Automation of Electronic Systems (TODAES), Vol. 2, No. 1, pp. 2-21, Janeiro de 1997.

[171]McFarland, M.C., Parker, A.C., Camposano, R., *"The high-level Synthesis of Digital Systems"*, Proceedings of the IEEE, Vol. 78, No. 2, pp. 301-318, Fevereiro de 1990.

[172]Gajski, D.D., Dutt, N.D., Wu, A.C-H., Lin, S.Y-L., *"High-Level Synthesis: Introduction to Chip and System Design"*, Kluwer Academic, 1992.

[173] Synfora Inc., *"PICO Extreme - The most advanced algorithmic synthesis enabling the most efficient hardware for complex algorithms"*, Dados de Produto, Disponível em http://www.synfora.com/products/picoExtreme.html, 2008.

[174] Mentor Graphics Corp., *"Catapult C Synthesis - Generate Correct-by-Construction, High-Quality RTL, 10-100x Faster"*, Disponível em http://www.mentor.com/products/esl/high_level_synthesis/catapult_synthesis/, 20 de Maio de 2009.

[175] Ku, D., De Micheli, G., "*HardwareC -- a Language for Hardware Design (Version 2.0"*, Stanford University Technical Report, CSL-TR-90-419, Stanford, CA, USA, Abril, 1990.

[176] De Micheli, G., Ku, D., Mailhot, F., Truong, T., *"The Olympus Synthesis System"*, IEEE Design and Test of Computers, vol. 7, no. 5, pp. 37–53, Oct. 1990.

[177] Séméria, L., Sato, K., De Micheli, G., *"Memory Representation and Hardware Synthesis of C Code with Pointers and Complex Data Structures"*, Proceedings of Synthesis And System Integration of MIxed Technologies Workshop (SASIMI'00), pp. 43-48, Kyoto, Abril de 2000.

[178] Semeria, L., De Micheli, G., *"Resolution, Optimization, and Encoding of Pointer Variables for the Behavioral Synthesis from C"*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 20, No. 2, pp. 213-233, Fevereiro de 2001.

[179] Rosenstiel, W. and Camposano, R., *"Synthesizing Circuits from Behavioral Level Specifications"*, Proceedings of the 7th International Conference on Computer Hardware Description Languages and their Applications, pp. 391-402, Holanda, Agosto de 1985.

[180] De-Michelli, G., Ku, D., *"HERCULES - A System for High-Level Synthesis"*, 25th ACM/IEEE Design Automation Conference (DAC'88), pp. 483-488, Anaheim, California, Estados Unidos, 12-15 de Junho de 1988.

[181] McFarland, M.C., Parker, A.C., Camposano, R., *"Tutorial on High-level Synthesis"*, Proceedings of the 25th ACM/IEEE conference on Design automation, pp.330-336, Atlantic City, New Jersey, Estados Unidos, 12-15 de Junho de 1988.

[182] Stroud, C.E., Munoz, R.R., Pierce, D.A., *"Behavioral Model Synthesis with Cones"*, IEEE Design & Test of Computers, Vol. 5, No. 3, pp. 22–30, Junho de 1988.

[183] Composano, R., Wolf, W., *"High-Level VLSI Synthesis"*, Kluwer Academic Publishers, Boston, 1991.

[184] Gajski, D., Vahid, F., Narayan, S., Gong, J., *"Specification and Design of Embedded Systems"*, Prentice Hall, 1994.

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

[186] Knapp, D., *"Behavioral Synthesis: Digital System Design Using the Synopsys Design Compiler"*, Prentice Hall, Upper Saddle River, NJ, 1996.

[187] Bye, C.T., Lightner, M.R., Ravenscroft, D.L., *"A Functional Modeling and Simulation Environment based on ESIM and C"*, Proceedings of International Conference on Computer-Aided Design (ICCAD'84), pp.51-53, Novembro de 1984.

[188] Frey, E.J., *"ESIM: A Functional Level Simulation Tool"*, Proceedings of International Conference on Computer-Aided Design, pp. 48-50, 1984.

[189] Chawla, B.R., Gummel, H.K., Kozak, P., *"MOTIS-An MOS Timing Simulator"*, IEEE Transactions on Circuits and Systems, Vol. 22, No. 12, pp. 901-910, Dezembro de 1975.

[190] Chu, C.-M., Potkonjak, M., Thaler, M., Rabaey, J., *"HYPER: an Interactive Synthesis Environment for High Performance Real Time Applications"*, Proceedings of IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD'89), pp. 432-435, Cambridge, MA, Estados Unidos, 2-4 de Outubro de 1989.

[191] Rabaey, J.M., Chu, C., Hoang, P., Potkonjak, M., *"Fast prototyping of Datapath-Intensive Architectures"*, IEEE Design & Test of Computers, Vol. 8, No. 2, pp. 40-51, Jun 1991.

[192] R.Ernst, J.Henkel and T. Benner, *"Hardware-Software Co-synthesis for Micro-controllers"*, IEEE Design & Test, pp. 64-75, December 1993.

[193] Ernst, R., Henkel, J., Benner, Th., Ye, W., Holtmann, U., Herrmann, D., Trawny, M.,, *"The COSYMA Environment for Hardware/Software Cosynthesis of Small Embedded Systems"*, Microprocessors and Microsystems, Vol. 20, No. 3, pp.159-166, Maio de 1996.

[194] Schaumont, P., Vernalde, S., Rijnders, L., Engels, M., Bolsens, I., *"A Programming Environment for the Design of Complex High Speed ASICs"*, Proceedings of Design Automation Conference (DAC'98), pp. 315-320, San Francisco, CA, Estados Unidos, 15-19 de Junho de 1998.

[195] Vanmeerbeeck, G., Schaumont, P., Vernalde, S., Engels, M., Bolsens, I., *"Hardware/software Partitioning of Embedded System in OCAPI-xl"*, Proceedings of the Ninth International Symposium on Hardware/Software Codesign (CODES 2001), pp. 30-35, Copenhagen, Dinamarca, 25-27 de Abril de 2001.

[196] Clarke, P., *"CoWare licenses Ocapi C/C++ from IMEC"*, EETimes, 7 de Fevereiro de 2000, Disponível em http://www.eetimes.com/story/OEG20000207S0033, Acessado em 22 de Maio de 2009.

[197] CoWare, Inc., *"CoWare Platform Architect - SystemC Platform Capture and Architecture Analysis for Platform-driven ESL Design"*,Dados de Produto, Disponível em http://www.coware.com/products/platformarchitect.php, Acessado em 22 de Maio de 2009.

[198] Center for Embedded Computer Systems, Microelectronic Embedded Systems Laboratory, *"Design Flow Through the SPARK Framework"*, University of California, San Diego. Disponível em http://mesl.ucsd.edu/spark/methodology.shtml, 19 de Maio de 2009.

[199] Synopsys Inc., *"Design Compiler Graphical - Extending Design Compiler topographical technology to predict and alleviate routing congestion"*, Especificação de Produto, Disponível em http://www.synopsys.com, 19 de Maio de 2009.

[200] Scheffer, L., Lavagno, L., Martin, G., *"Electronic Design Automation for Integrated Circuits Handbook"*, Taylor & Francis/CRC Press, 2006.

[201] Gupta, S., Dutt, N., Gupta, R., Nicolau, A., *"A High-level Synthesis Framework for Applying Parallelizing Compiler Transformations"*, Proceedings of 16th International Conference on VLSI Design, pp. 461–466, 4-8 de Janeiro de 2003.

[202] Poseidon Design Systems Inc., *"Triton Builder""*, Disponível em http://www.poseidon-systems.com/tbuilder.htm, 19 de Maio de 2009.

[203] Gupta, S., Curriculum Vitae, Disponível em http://www.4bearsonline.com/sumitg/SumitResume.pdf, 19 de Maio de 2009.

[204] Babb, J., Rinard, M., Moritz, C.A., Lee, W., Frank, M., Barua, R., Amarasinghe, S., *"Parallelizing Applications Into Silicon"*, Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines '99 (FCCM '99), pp. 70-80, Napa Valley, CA, Estados Unidos, Abril de 1999.

[205] Wilson, R.P., French, R.S., Wilson, C.S., Amarasinghe, S.P., Anderson, J.M., Tjiang, S.W.K., Liao, S-W., Tseng, C-W., Hall, M.W., Lam, M.S., Hennessy, J.L., *"SUIF: An Infrastructure for Research on Parallelizing and Optimizing Compilers"*, ACM SIGPLAN Notices, Vol. 29, No. 12, pp. 31-37, Dezembro de 1994.

[206] Kambe, T., Yamada, A., Nishida, K., Okada, K., Ohnishi, M., Kay, A., Boca, P., Zammit, V., Nomura, T., *"A C-based Synthesis System, Bach, and its Application"*, Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC'2001), pp. 151-155, Yokohama, Japão, 30 de Janeiro a 2 de Fevereiro de 2001.

[207] Hoare, C.A.R., *"Communicating Sequential Processes"*, Prentice-Hall, 1985.

[208] SGS-THOMSON Microelectronics Limited, *"occam 2.1 reference manual"*, 12 de Maio de 1995.

[209] Galloway, D., *"The Transmogrifier C Hardware Description Language and Compiler for FPGAs"*, Proceedings of the IEEE Symposium on Fpga's For Custom Computing Machines (FCCM'95), pp. 136-144, Napa, California, Estados Unidos, 19-21 de Abril de 1995.

[210] University of Toronto, *"Transmogrifier C"*, Disponível em http://www.eecg.toronto.edu/EECG/RESEARCH/tmcc/tmcc/, Acessado em 22 de Maio de 2009.

[211] Goering, R., *"EDA startup spins C-based HDL"*, EETimes, 9 de Novembro de 1999, Disponível em http://www.eetimes.com/story/OEG19991109S0029, Acessado em 22 de Maio de 2009.

[212] Clarke, P., *"DATE99: Frontier ports A/RT Builder to NT"*, EETimes, 8 de Março de 1999, Disponível em http://www.eetimes.com/story/OEG19990308S0027, Acessado em 22 de Maio de 2009.

[213] Synopsys, Inc., *"Apple Computer Designs Interactive TV Set-Top Box ASIC With Behavioral Compiler"*, Disponível em http://www.synopsys.com/ip/capsulemodule/apple_synop_a4.pdf, 21 de Maio de 2009.

[214] Wakabayashi, K., *"C-based Synthesis with Behavioral Synthesizer, Cyber"*, Proceedings of Design, Automation and Test in Europe Conference and Exhibition, pp. 390-393, Munich, Alemanha, 9-12 de Março de 1999.

[215] Wakabayashi, K., *"CyberWorkBench Integrated Design Environment Based on C-based Behavior Synthesis and Verification"*, IEEE VLSI-TSA International Symposium on VLSI Design, Automation and Test (VLSI-TSA-DAT), pp. 173-176, 27-29 de Abril de 2005.

[216] Bolsens, I., De Man, H.J., Lin, B., Van Rompaey, K., Vercauteren, S., Verkest, D., *"Hardware/Software Co-Design of Digital Telecommunication Systems"*, Proceedings of the IEEE, Vol. 85, No. 3, pp. 391-418, Março de 1997.

[217] Willekens, P., Devisch, D., Van Canneyt, M., Conflitti, P., Genin, D., *"Algorithm Specification in DSP Station Using Data Flow Language"*, DSP Applications, vol. 3, no. 1, pp. 8 – 16, Janeiro de 1994.

[218] De Man, H., Rabaey, J., Six, P., Claesen, L., *"Cathedral-II: A Silicon Compiler for Digital Signal Processing"*, IEEE Design & Test of Computers, Vol. 3, No. 6, pp. 13-25, Dezembro de 1986.

[219] Clarke, P., *"CoWare'S N2C adds support for SystemC"*, EETimes, 4 de Junho de 2001, Disponível em http://www.eetimes.com/conf/dac/showArticle.jhtml?articleID=17407432&kc=2443, Acessado em 22 de Maio de 2009.

[220] Mencer, O., Morf, M., Flynn, M.J., *"PAM-Blox, High Performance FPGA Design for Adaptive Computing"*, Proceedings. IEEE Symposium on FPGAs for Custom Computing Machines, pp. 167-174, Napa Valley, CA, Estados Unidos, 15-17 de Abril de 1998.

[221] Mencer, O., *"PAM-Blox II Design and Evaluation of C++ Module Generation for Computing with FPGAs"*, Proceedings. 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, pp. 67-76, 2002.

[222] Gokhale, M., Minnich, R., *"FPGA Computing in a Data Parallel C"*, Proceedings. IEEE Workshop on FPGAs for Custom Computing Machines (FCCM'93), pp. 94-101, Napa, CA, Estados Unidos, 5-7 de Abril de 1993.

[223] Schlesinger, J., Gokhale. M., *"Dbc Reference Manual"*, Relatório Técnico 92-068, Supercomputing Research Center, Outubro de 1992.

[224] Callahan, T.J., Wawrzynek, J., *"Instruction Level Parallelism for Reconfigurable Computing"*, Proceedings of International Workshop on Field Programmable Logic, pp. 248-257, Tallinn, September 1998.

[225] Yanbing Li, Callahan, T., Darnell, E., Harr, R., Kurkure, U., Stockwood, J., *"Hardware-Software Co-Design of Embedded Reconfigurable Architectures"*, Proceedings of the 37th Design Automation Conference (DAC'00), pp. 507-512, Junho de 2000.

[226] Budiu, M., Goldstein, S.C., *"Compiling Application-specific Hardware"*, Proceedings of the 12th International Conference on Field Programmable Logic and Applications (FPL), Lecture Notes in Computer Science, vol. 2438, pp. 853–863, Montpellier, França, Setembro de 2002.

[227] Banerjee, P., Haldar, M., Nayak, A., Kim, V., Saxena, V., Parkes, S., Bagchi, D., Pal, S., Tripathi, N., Zaretsky, D., Anderson, R. and Uribe, J. R., *"Overview of a Compiler for Synthesizing MATLAB Programs Onto FPGAs"*, In IEEE Transactions on VLSI Systems, vol. 12, no. 3, pp. 312-324, Março de 2004.

[228] Urard, P., Paumier, L., Viollet, M., Lantreibecq, E., Michel, H., Muroor, S., Coates, B., Gupta, B., *"A generic 350 Mb/s turbo-codec based on a 16-states SISO decoder"*, Digest of Technical Papers of IEEE International Solid-State Circuits Conference, pp. 424-536, Vol.1, Feb 15-19, 2004.

[229] Urard, P., Yeo, E., Paumier, L., Georgelin, P., Michel, T., Lebars, V., Lantreibecq, E., Gupta, B., *"A 135Mb/s DVB-S2 compliant codec based on 64800b LDPC and BCH codes"*, Digest of Technical Papers of IEEE International Solid-State Circuits Conference 2005, pp. 446-609, Vol.1, Feb 10-10, San Francisco, CA, 2005.

[230] P. Georgelin, *"Formal Verification of Synchronous Digital Systems, Based on Symbolic Simulation"*, PhD Thesis, Université Joseph Fourier, France, 18 Oct., 2001.

[231] Hutchings, B., Bellows, P., Hawkins, J., Hemmert, S., Nelson, B., Rytting, M., *"A CAD Suite for High-Performance FPGA Design"*, Proceedings of Seventh Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 12-24, Napa Valley, CA, Estados Unidos, 21-23 de Abril de 1999.

[232] Kuhn, T. and Rosenstiel, W., *"Java Based Object Oriented Hardware Specification and Synthesis"*, In Proceedings of ASP-DAC, pp. 579-581, Jan. 2000.

[233] Thomson, R., Chouliaras, V., Mulvaney. D., *"From UML to Structural Hardware Designs"*, Proceedings of 4th UML-SoC Workshop at 44thDAC Conference, San Diego, California, Estados Unidos, 2007.

[234] Coyle, F. P. and Thornton, M. A., *"From UML to HDL: a Model Driven Architectural Approach to Hardware-Software Co-Design"*, In Proceedings of Information Systems: New Generations Conference (ISNG), pp.88-93, Apr. 2005.

[235] Kangas, T., Kukkala, P., Orsila, H., Salminen, E., Hännikäinen, M., Hämäläinen, T. D., Riihimäki, J., and Kuusilinna, K., *"UML-based Multiprocessor SoC Design Framework"*, ACM Transactions on Embedded Computing Systems (TECS), Vol. 5, No. 2, pp. 281-320, Maio de 2006.

[236] Bjarklund, D. and Lilius. J., *"From UML Behavioral Descriptions to Efficient Synthesizable VHDL"*, In Proceedings of IEEE Norchip Conference, Nov. 2002.

[237] Nguyen, K.D., Zhenxin Sun, Thiagarajan, P.S., Wong, W-F., *"Model-driven SoC Design Via Executable UML to SystemC"*, Proceedings of 25th IEEE International Real-Time Systems Symposium, pp. 459-468, 5-8 de Dezembro de 2004.

[238] Riccobene, E., Scandurra, P., Rosti, A., Bocchio, S., *"A SoC Design Methodology Involving a UML 2.0 Profile for SystemC"*, Proceedings of Design, Automation and Test in Europe, Vol. 2, pp. 704-709, 7-11 de Março de 2005.

[239] Moyers, S., Thomson, R., Chouliaras, V., Mulvaney, D., *"UML-based Design of a JPEG-LS IP via Axilica FalconML"*, Proceedigns of the Intellectual Property-based Electronic System Conference and Exchibition (IP08), Grenoble, França, 3-4 de Dezembro de, 2008.

[240] Axilica Limited, *"FalconML Delivers behavioural synthesis from UML of FPGA or ASIC"*, Disponível em http://www.axilica.com/, Acessado em 23 de Maio de 2009.

[241] Tripp, J.L., Peterson, K.D., Ahrens, C., Poznanovic, J.D., Gokhale, M.B., *"Trident: an FPGA compiler framework for floating-point algorithms"*, International Conference on Field Programmable Logic and Applications, pp. 317-322, 24-26 de Agosto de 2005.

[242] Trident Compiler, Dispível em http://sourceforge.net/projects/trident, Acessado em 23 de Maio de 2009.

[243] Trickey, H., *"Flamel: A High-Level Hardware Compiler"*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 6, No. 2, pp. 259-269, Março de 1987.

[244] Wei, R., Rothweiler, S., and Jou, J., *"BECOME: Behavior Level Circuit Synthesis Based on Structure Mapping"*, Proceedings of the 25th ACM/IEEE Conference on Design Automation (DAC'88), pp. 409-414, Atlantic City, New Jersey, Estados Unidos, 12-15 de Junho de 1988.

[245] Van Nieuwenhoven, K., De Moortel, J., Genin, D., Note, S., *"Mistral 2 a True Architectural Synthesis Tool: from a Behavioural Specification down to a Register Transfer Level Description"*, DSP Applications and Multimedia, Outubro de 1994.

[246] Martin, E., Sentieys, O., Dubois, H., Philippe, J.L., *"GAUT: An Architectural Synthesis Tool for Dedicated Signal Processors"*, Proceedings of European Design Automation Conference (Euro DAC'93), with EURO-VHDL'93, pp. 14-19, Hamburg, Alemanha, 20-24 de Setembro de 1993.

[247] Note, S., Geurts, W., Catthoor, F., De Man, H., *"Cathedral-III: Architecture-driven High-level Synthesis for High Throughput DSP Applications"*, 28th ACM/IEEE Design Automation Conference, pp. 597-602, 1991.

[248] Park, I., O'Brien, K., Jerraya, A.A., *"AMICAL: Architectural Synthesis Based on VHDL"*, Proceedings of the IFIP WG10.2/WG10.5 Workshops on Synthesis for Control Dominated Circuits, pp. 219-234, 1993.

[249] Camposano, R., Rosentiel, A., *"Synthesizing Circuits from Behavioral Descriptions"*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 8, No. 2, pp. 171-180, Fevereiro de 1989.

[250] Camposano, R., Weber, R., *"DSL-Eine Sprache zur Spezifikation digitaler Schaltungen"*, Rep. 24/84. Fakultaet fuer Informatik, Univ. Karlsruhe, 1984.

[251] Camposano, R., Treff, L., *"STRUDEL-Eine Sprache zur Spezifikation der Struktur digitaler Schaltungen"*, Rep. 7/84, Fakultaet fuer Informatik. Univ. Karlsruhe, 1984.

[252] Stoll, A., Duzy, P., *"High-level Synthesis from VHDL with Exact Timing Constraints"*, Proceedings of the 29th ACM/IEEE conference on Design Automation (DAC'92), pp. 189-193, Anaheim, California, Estados Unidos, Setembro de 1992.

[253] Ku, D., De Micheli, G., *"High-level Synthesis and Optimization Strategies in Hercules and Hebe"*, Proceedings of the European ASIC Conference (EURO ASIC'90), pp. 111–120, Paris, France, Maio de 1990.

[254] Gokhale, M., Stone, J., Arnold, J., Kalinowski, M., *"Stream-oriented FPGA Computing in the Streams-C High Level Language"*, IEEE Symposium on Field-Programmable Custom Computing Machines, pp.49-56, Napa Valley, CA, Estados Unidos, 17-19 de Abril de 2000.

[255] AutoESL Design Technologies, http://www.autoesl.com/, Acesso em 23 de Maio de 2009.

[256] OCP-IP, *"Open Core Protocol Specification"*, Ver. 2.1, Disponível em http://www.ocpip.org/socket/ocpspec/, Acessado em 23 de Maio de 2009.

[257] Agility Design Solutions Inc., *"DK Design Suite - Rapid implementation from C to FPGA"*, Ver. 1.0, Disponível em http://agilityds.com/products/c_based_products/default.aspx, 2008.

[258] Loo, S.M., Wells, B. E., Freije, N., Kulick, J., *"Handel C for Rapid Prototyping of VLSI Coprocessors for Real Time Systems"*, 34th Southeastern Symposium on System Theory (SSST 2002), IEEE Proceedings, 2002.

[259] Agility Design Solutions Inc., *"Agility MCS: MatLab to C Synthesis - Automatic Generation of ANSI C-Code from MatLab Source"*, Ver. 1.0, Disponível em http://agilityds.com/products/matlab_based_products/mcs/default.aspx, 2008.

[260] Agilent Technologies Inc., *"Agilent EEsof EDA SystemVue 2008 - Configuration and Ordering Guide"*, Disponível em http://www.agilent.com/find/eesof-systemvue, Acessado em 23 de Maio de 2009.

[261] Synopsys, Inc., *"Synplify DSP ESL Synthesis - High-performance DSP Implementation for FPGAs and ASICs"*, Dados de Produto, Disponível em http://synopsys.com/Tools/SLD/Algorithmic Synthesis/Pages/default.aspx, 2008.

[262] Impulse Accelerated Technologies, Inc., *"Accelerate C in FPGA - Fast Prototyping, Fast Optimization, Fast Applications"*, Disponível em http://www.impulseaccelerated.com/, 20 de Maio de 2009.

[263] Mitrionics AB, *"Mitrion Product Brief"*, Disponível em http://www.mitrionics.com/?page=products_platform, 20 de Maio de 2009.

[264] Dellson, A., *"Progamming FPGAs for High Performance Computing Acceleration"*, Xilinx XCELL Journal, No. 55, pp. 92-94, 1 de Dezembro de 2005.

[265] Urban, K., *"In-Socket Accelerators -- When to Use Them"*, XtremeData Inc. White Paper, Disponível em http://www.xtremedatainc.com, 5 de Junho de 2008.

[266] XtremeData Inc, *"Star-P Parallel FPGA Solution"*, Folha de dados do Produto, Disponível em http://www.xtremedatainc.com, 12 de Maio de 2009.

[267] CebaTech, Inc., *"CebaTech C2R Compiler"*, Dados de Produto, Disponível em http://www.cebatech.com/c2r_compiler, 20 de Maio de 2009.

[268] Gupta, S., Dutt, N.D., Gupta, R.K., Nicolau, A., *"SPARK: A High-Level Synthesis Framework for Applying Parallelizing Compiler Transformations"*, International Conference on VLSI Design, pp. 461-466, 4-8 de Janeiro de 2003.

[269] Bluespec, Inc., *"Bluespec Compiler (BSC)"*, Disponível em http://www.bluespec.com/products/bsc.htm, 21 de Maio de 2009.

[270] The Mathworks Inc., *"Simulink HDL Coder 1.5 - Generate HDL code from Simulink models and MATLAB code"*,Disponível em http://www.mathworks.com/products/slhdlcoder/?s_cid=HP_FP_SL_HDLCoder, Acessado em 23 de Maio de 2009.

[271] Vitabile, S., Gentile, A., Siniscalchi, S. M. and Sorbello, F., *"Efficient Rapid Prototyping of Image and Video Processing Algorithms"*, Proceedings of Euromicro Symposium on Digital System Design 2004, pp. 452- 458, Aug. 31 - Sept. 3, 2004.

[272] Agility Design Solutions Inc., *"Agility DK Design Suite Version 5.0 - Sofware Product Description"*, Product Datasheet, Abril de 2008.

[273] Huet, S., Casseau, E., Pasquier, O., *"Design Exploration and HW/SW Rapid Prototyping for Real-time System Design"*, The 16th IEEE International Workshop on Rapid System Prototyping, June 8-10, pp. 240 – 242, Montreal, Canada, 2005.

[274] Corre, G., Julien, N., Senn, E., Martin, E., "*Intégration de la synthèse mémoire dans l'outil de synthèse d'architecture GAUT low power"*, Proceedings of Journées

Francophone Adéquation Algorithme Architecture (JFAAA'02), Monastir, Tunisia, Dezembro de 2002.

[275] Benkrid, K., Crookes, D., Smith, J., Benkrid, A., *"High Level Programming for Real Time FPGA Based Video Processing"*, Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing, 2000. ICASSP '00, Vol. 6, pp. 3227-3230, Istanbul, Turquia, 5-9 de Junho de 2000.

[276] Ritter, G.X., Wilson, J.N., Davidson, J.L., *"Image Algebra: an overview"*, Computer Vision, Graphics and Image Processing, Vol. 49, No. 3, pp. 297-331, Março de 1990.

[277] Crookes, D., Alotaibi, K., Bouridane, A., Donachy, P., Benkrid, A., *"An Environment for Generating FPGA Architectures for Image Algebra-based Algorithms"*, Proceedings of IEEE International Conference on Image Processing (KIP-98), Vol. 3, pp 990-994, 1998.

[278] Clocksin, W.F., Melish, C.S., *"Programming in Prolog"*, 5ª Edição, Springer, 2003.

[279] Engels, M., Meng, T., *"Rapid Prototyping of a Real-Time Video Encoder"*, Proceedings of Fifth International Workshop on Rapid System Prototyping, pp. 8-15, Grenoble, France, Jun 21-23, 1994.

[280] Madisetti, V.K., Curtis, B.A., *"A Quantitative Methodology for Rapid Prototyping and High-Level Synthesis of Signal Processing Algorithms"*, IEEE Transactions on Signal Processing, Vol. 42, No. 11, pp. 3188-3208, Nov 1994.

[281] Pirsch, P., Demassieux, N., Gehrke, W., *"VLSI Architectures for Video Compression - A Survey"*, Proceedings of the IEEE, Vol. 86, No. 2, pp. 220-246, Fevereiro de 1995.

[282] Bertozzi, D., Jalabert, A., Murali, S., Tamhankar, R., Stergiou, S., Benini, L., De Micheli, G., *"NoC Synthesis Flow for Customized Domain Specific Multiprocessor Systems-on-Chip"*, IEEE Transactions on Parallel and Distributed Systems, Vol. 16, No. 2, pp. 113–129, Fevereiro de 2005.

[283] Bosco, A., Findlater, K., Battiato, S., Castorina, A., *"A Noise Reduction Filter For Full-Frame Data Imaging Devices"*, IEEE Transactions on Consumer Electronics, Vol. 49, No. 3, pp. 676 - 682, Agosto de 2003.

[284] Andreadis, I. and Louverdis, G., *"Real-time Adaptive Image Impulse Noise Suppression"*, IEEE Transactions Instrumentation and Measurement, pp. 798-806, Junho de 2004.

[285] Bosco, A., Mancuso, M., Battiato, S. and Spampinato, G., *"Temporal Noise Reduction of Bayer Matrixed Video Data"*, Proceedings of IEEE International Conference on Multimedia and Exposition, Vol. 1, pp. 681-684, Augosto de 2002.

[286] Dubois, E., Sabri, S., *"Noise Reduction in Image Sequences Using Motion-Compensated Temporal Filtering"*, IEEE Transactions on Communications, Vol. 32, No. 7, pp. 826- 831, Julho de 1984.

[287] Vassiliadis, S., Hakkennes, E.A., Wong, J.S.S.M., Pechanek, G.G., *"The Sum-Absolute-Difference Motion Estimation Accelerator"*, Proceedings of the 24th Euromicro Conference, Vol. 2, pp. 559-566, Vasteras, Suécia, 25-27 de Agosto de 1998.

[288] Tokyo Electron Device Limited, *"TB-5V-LX330-EX - Virtex-5 LX330 Evaluation Platform for ASIC Prototyping"*, Disponível em http://www.inrevium.jp/eng/x-fpga-board/sepideh5x.html, Acessado em 24 de Maio de 2009.

[289] Synplicity Inc, *"Synplify Pro"*, Disponível em http://www.synplicity.com/products/synplifypro/, Acessado em 24 de Maio de 2009.