UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

WAGSTON TASSONI STAEHLER

Projeto de Sistemas Digitais Complexos: uma Aplicação ao Decodificador H.264

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação

Prof. Dr. Altamiro Amadeu Susin
Orientador

Porto Alegre, dezembro de 2006.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Staehler, Wagston Tassoni

Projeto de Sistemas Digitais Complexos: uma Aplicação ao Decodificador H.264 / Wagston Tassoni Staehler – Porto Alegre: Programa de Pós-Graduação em Computação, 2006.

108 f.:il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2006. Orientador: Altamiro Amadeu Susin.

1. Codificação de vídeo 2. Padrão H.264 3. Projeto de sistemas digitais I. Susin, Altamiro Amadeu. II. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. José Carlos Ferraz Hennemann
Vice-Reitor: Prof. Pedro Cezar Dutra Fonseca
Pró-Reitora de Pós-Graduação: Profa. Valquiria Linck Bassani
Diretor do Instituto de Informática: Prof. Flávio Rech Wagner
Coordenador do PPGC: Prof. Carlos Alberto Heuser
Bibliotecária-Chefe do Instituto de Informática: Beatriz Regina Bastos Haro

AGRADECIMENTOS

Aproveito a oportunidade para agradecer a Deus.

A Moisés, Beatriz e Jennifer, o meu muito obrigado, por serem uma família tão carinhosa e atenciosa, pelo incentivo e apoio.

A Fernanda, que está sempre ao meu lado e que faz com que meus dias sejam sempre plenos de inspiração e alegrias.

A Altamiro Susin, meu orientador, pela atenção, pela amizade, pela experiência e conhecimento compartilhados.

A Letícia, Borin, Ana, Agostini, Vagner, Marcel e Eduardo pelas discussões que levaram ao progresso de minha pesquisa.

À equipe TV Digital e aos colegas do LaPSI, pelo companheirismo e pelo dia-a-dia tão agradável.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

AD	Conversor Analógico-Digital
AHB	*Advanced High Performance Bus*
AMBA	*Advanced Microcontroller Bus Architecture*
APB	*Advanced Peripheral Bus*
ARM	*Advanced RISC Machines*
ASIC	*Application-Specific Integrated Circuit*
ASIP	*Application-Specific Instruction-Set Processor*
AVC	*Advanced Video Coding*
AVI	*Microsoft Audio Video Interleave*
BFM	*Bus Functional Model*
CABAC	*Context-based Adaptive Binary Arithmetic Coding*
CAD	*Computer Aided Design*
CAVLC	*Context-Adaptive Variable Length Coding*
CIF	*Common Intermediate Format*
CODEC	Par Codificador/Decodificador
DA	Conversor Digital-Analógico
DC	*Direct Current* – Corrente Contínua
DRAM	*Dynamic Random Access Memory*
EDK	*Embedded Development Kit*
FPGA	*Field-Programmable Gate Array*
GME	Grupo de Microeletrônica do Instituto de Informática - UFRGS
HDL	*Hardware Description Language*
HDTV	*High Definition Digital Television*
I/O	*Input/Output*
IBM	*International Business Machines*
IEEE	*The Institute of Electrical and Electronics Engineers*
IIP	*Implementation IP*

IP	*Intellectual Property*
ISE	*Xilinx Integrated Software Environment*
ISO	*International Organization for Standardization*
ITRS	*International Technology Roadmap for Semiconductors*
ITU-T	*International Telecommunication Union - Telecommunication Standardization Sector*
JTAG	*Joint Test Action Group*
LaPSI	Laboratório de Processamento de Sinais e Imagens - UFRGS
LED	*Light Emitter Diode*
LGPL	*Lesser General Public License*
MC	*Motion Compensation*
ME	*Motion Estimation*
MPEG	*Moving Picture Experts Group*
NoC	*Network-on-Chip*
OCP-IP	*Open Core Protocol International Partnership*
OPB	*On-chip Peripheral Bus*
PC	*Personal Computer*
PLB	*Processor Local Bus*
RISC	*Reduced Instruction Set Computer*
RMM	*Reuse Methodology Manual*
RTL	*Register Transfer Level*
SBTVD	Sistema Brasileiro de Televisão Digital
SDTV	*Standard Definition Television*
SoC	*System on a Chip*
SRAM	*Static Random Access Memory*
UART	*Universal Asynchronous Receiver/Transmitter*
UFRGS	Universidade Federal do Rio Grande do Sul
USB	*Universal Serial Bus*
VC	*Virtual Component*
VHDL	*VHSIC Hardware Description Language*
VHSIC	*Very-High-Speed Integrated Circuit*
VIP	*Verification IP*
VSIA	*Virtual Socket Interface Alliance*
XPS	*Xilinx Platform Studio*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

A evolução dos processos de fabricação de circuitos microeletrônicos coloca um número cada vez maior de dispositivos à disposição do projetista de circuitos integrados. Mais e mais funcionalidades são adicionadas aos equipamentos eletrônicos com um aumento correspondente no esforço de projeto. Aplicações de multimídia e comunicação digital, por exemplo, são muito populares e integram funções cada vez mais complexas. As janelas de mercado diminuem com a grande competição por novos produtos. Este cenário desafia os projetistas: são necessárias novas metodologias.

Para aumentar a produtividade de uma equipe de projeto, é imprescindível a utilização de um nível de abstração mais alto. O mesmo sistema pode ser descrito por um número menor de primitivas se a linguagem de descrição possuir primitivas semanticamente mais ricas. Este é o chamado projeto baseado em reuso, onde módulos são desenvolvidos para responderem necessidades mais genéricas e serem reconfiguráveis. Além disso, devem seguir algum padrão de interface de comunicação.

Aplicações multimídia são muito complexas. Na área de compressão de vídeo, por exemplo, há uma grande quantidade de processamento para permitir a compressão dos dados. Áudio e vídeo geram uma grande quantidade de dados. É imperativo comprimir os dados de maneira a permitir o seu armazenamento/transmissão através de recursos limitados. H.264 é a evolução dos padrões de compressão de vídeo digital, como H.263 ou MPEG-2, e a sua implementação só é possível graças ao progresso da microeletrônica. O desenvolvimento de um decodificador H.264 é um exemplo de um projeto de sistema digital complexo, visto como uma composição de módulos que executam as diferentes operações sobre o sinal.

O foco deste trabalho é a metodologia para a construção de sistemas digitais a partir de funções já prontas, em um fluxo de projeto que permita o projeto e o teste baseados em reuso. O caso de estudo, o decodificador H.264, é analisado como um sistema composto por alguns módulos e o resultado é uma metodologia SoC apropriada para ele. Este trabalho levará a uma descrição de como o decodificador foi desenvolvido, uma vez que as técnicas de processamento e os desafios de implementação tenham sido completamente compreendidos.

Palavras-Chave: codificação de vídeo, padrão H.264, metodologia de projeto baseado em reuso, System-on-Chip.

Complex Digital Systems Design: An H.264 Decoder Case Study

ABSTRACT

The evolution of the manufacturing process of microelectronic circuits offers an ever increasing number of devices to the chip designer. More and more functionalities are added to the electronic equipments with a corresponding increase in design effort. Multimedia and digital communication applications, for example, are very popular and integrate each time more complex functions. The time-to-market reduces with the competition for new products. This scenario challenges the circuit designers: new methodologies are needed.

To increase the productivity of a design team, higher level of abstraction must be used. The same system can be described with less number of primitives if the description language has primitives semantically richer. One primitive can call a pre-designed module giving a hierarchical design process. This is the so called reuse based design, because modules are developed to respond general needs and made reconfigurable and they must follow some standards of communication interfaces.

Multimedia applications are very complex. For video compression, for example, we need a big amount of processing in order to realize data compressing. Audio and video generate a big amount of data. It is imperative to compress the data to allow its storage/transmission through limited resources. H.264 is the evolution of video compression standards, like H.263 or MPEG-2, and its implementation is only possible due to microelectronics progress. Its development is an example of a complex digital system design, and can be seen as a composition of modules that execute the different signal operations.

The focus here is the methodology for building digital systems from functions already developed, in a design flow that facilitates reuse-based design and test. The case study, an H.264 decoder, is analyzed as a system made of several modules and the result is a SoC methodology fashioned for it. This work presents a description of how the decoder was developed, after the complete understanding of all the involved processing techniques and design implementation challenges.

Keywords: Video coding, H.264 standard, reuse based design methodology, System-on-Chip.

1 INTRODUÇÃO

Nos tempos atuais podemos encontrar *chips* em praticamente todos os produtos eletrônicos, como aplicações de entretenimento, telecomunicações, engenharia biomédica, controle de equipamentos industriais, produção, etc. O projeto de sistemas digitais está se fazendo presente cada vez mais como um meio, atendendo as diversas necessidades das demais áreas do conhecimento. O progresso da física de semicondutores e também das técnicas de fabricação de circuitos integrados vem oferecer cada vez mais recursos aos projetistas de sistemas eletrônicos abrindo um leque de novas possibilidades. Mais transistores significa mais inteligência agregada ao sistema e, conseqüentemente, sistemas muito mais complexos capazes de realizar tarefas anteriormente impossíveis.

Na realidade, conforme pode ser visto no gráfico da Figura 1.1 extraído do *International Technology Roadmap for Semiconductors* (ITRS, 2001), as estatísticas atestam que ainda não é possível fazer uso eficiente de todos os transistores que a tecnologia oferece. Na figura, a reta traçada entre os pontos representados por pequenos losangos demonstra o crescimento do número de transistores por *chip*, e a reta traçada entre os pontos representados por pequenos "x" apresenta o crescimento da produtividade dos engenheiros, ou seja, a quantidade de transistores utilizados no projeto por engenheiro por mês. Este fenômeno é conhecido por *gap* de produtividade, ou seja, o número de transistores por circuito integrado cresce mais rapidamente do que nossa capacidade de utilizá-los eficientemente. Desta forma, impera a necessidade de encontrar uma forma de explorar este universo que se abre: cabe aos desenvolvedores de metodologias de projeto e ferramentas de CAD e também aos projetistas usufruírem desta disponibilidade oferecendo cada vez mais inteligência embarcada.

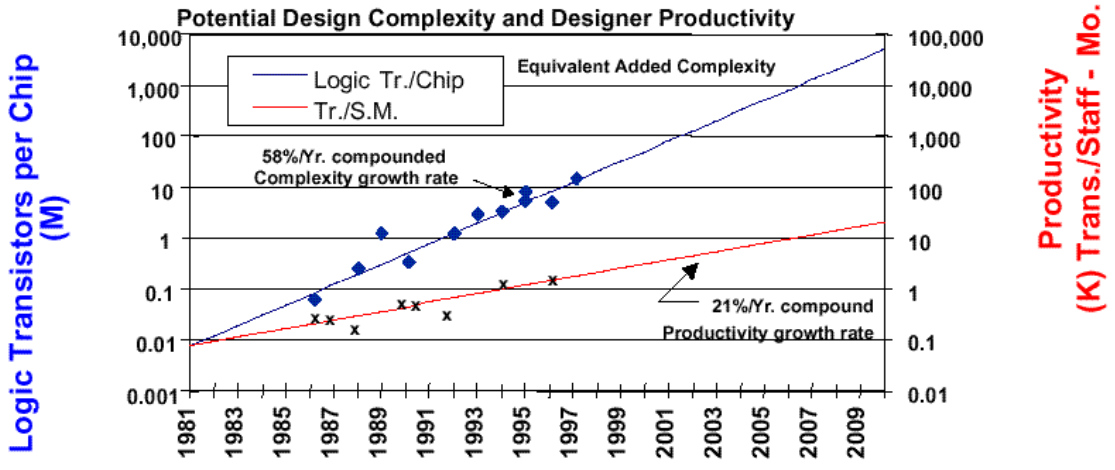

Figura 1.1: *Gap* de Produtividade

É preciso, acima de tudo, uma mudança na forma de projetar. Para atingir altos níveis de complexidade, é necessário trabalhar em um nível de abstração maior, seja através de ferramentas de CAD e de síntese automática, seja na realização de projetos baseados na reunião de diversos sub-sistemas e no reuso. Em suma, o que era feito sobre placas de circuito impresso através da soldagem de diversos *chips* com diversas funcionalidades, hoje pode ser realizado dentro de um só *chip*, produzindo o chamado *system-on-a-chip* (SoC).

A metodologia de projeto a ser adotada deve permitir a realização de projetos complexos, com um tempo de projeto cada vez menor. Esta metodologia deve primar pela modularidade, subdividindo o sistema até que seja possível ser desenvolvido por grupos menores ao invés de uma grande equipe. Já os módulos, se enquadrados em padrões de interface e de comunicação, podem ser chamados de IPs (*Intellectual Property*). Os IPs são concebidos para resolver problemas mais genéricos, de maneira que possa ser comercializado reduzindo, assim, seu custo de desenvolvimento. A integração de IPs permite a construção de SoCs. O ganho desta nova perspectiva advém do aumento do nível de abstração e do reuso, constituindo o sistema a partir de sub-módulos que podem ser previamente projetados ou advindos de terceiros. Seguindo esta tendência mundial já existem diversas iniciativas, como (Design and Reuse, 2006), (OpenCores, 2006), (Inicore, 2006) ou (BrazilIP, 2006), entre outras, aonde é possível adquirir diversos componentes IPs para as mais diversas aplicações.

Os IPs são componentes de *hardware* que respeitam padrões internacionais de interface e comunicação e as vantagens de produzi-los e utilizá-los na construção de um sistema são várias. Primeiro, na etapa de integração o processo é bastante facilitado levando-se em conta que a validação individual já foi realizada e que a conexão entre dois IPs é realizada de maneira trivial, semelhante a equipamentos *plug-and-play*. Segundo, no caso de evolução dos módulos e, conseqüentemente, do sistema, pode-se apenas efetuar a substituição do módulo envolvido. E por terceiro, observando-se o fato de que os IPs atendem necessidades mais genéricas eles poderão ser reutilizados em diversos outros projetos, dividindo o seu custo de desenvolvimento.

Este trabalho de dissertação visa apresentar uma metodologia de projeto segundo o paradigma SoC voltado ao desenvolvimento de um decodificador H.264. O padrão H.264 é o estado-da-arte em compressão de vídeo, sua eficiência é o resultado de anos de evolução da tecnologia de processamento de imagens e sua implementação é possível, hoje em dia, graças aos avanços da microeletrônica.

Na área de aplicações multimídia, são lançados equipamentos cada vez mais sofisticados, fazendo uso de algoritmos arrojados no campo da compressão de vídeo. Para permitir a transmissão ou o armazenamento do vídeo de alta qualidade utilizando uma quantidade reduzida de recursos, algoritmos altamente eficientes são implementados utilizando a capacidade computacional disponível, levando a uma redução de até cem vezes no volume de informação veiculado. Por sua grande complexidade, o decodificador H.264 se apresenta como uma experiência adequada ao estudo de metodologias de projeto SoC.

Este decodificador é naturalmente, por suas diversas funcionalidades, subdividido em seis grandes partes. Estas, por sua vez, podem ser novamente subdivididas aplicando-se a mesma metodologia recursivamente. Um algoritmo de descompressão de vídeo apresenta conceitos e algoritmos largamente utilizados no campo de processamento digital de sinais, o que justifica a geração de IPs para alguns dos

módulos que serão especificados permitindo o seu futuro reuso. Além do mais, para o próprio caso do decodificador H.264, existem múltiplos modos de operação atendendo diversas aplicações com diferentes requisitos de *hardware*. Isto justifica a realização de módulos configuráveis, ou seja, que possam ser inseridos nas diversas plataformas (desde um celular com requisitos mínimos de imagem até um cinema digital) sem a necessidade de reprojeto.

Desta forma, no Capítulo 2 segue um estudo sobre as metodologias apresentadas no internacionalmente aceito RMM (*Reuse Methodology Manual*), e de que modo elas poderão ser aplicadas ao gênero de projeto descrito acima. Assim como já houve outros algoritmos de compressão que evoluíram, deve-se levar em conta que este padrão também evoluirá. Através da utilização de padrões internacionais de comunicação e interface entre os módulos, pretende-se obter um decodificador de fácil evolução, sendo suficiente a alteração dos componentes implicados na modificação proposta. E ainda, de acordo com a funcionalidade, poderá haver interesse em se fazer com que o componente também se torne um IP.

Neste sentido, será apresentada no Capítulo 3 uma discussão sobre as diferentes formas em que um IP pode ser fornecido e suas vantagens e desvantagens. Também será abordada a estratégia de sub-divisão, ou seja, com que critérios serão definidos os IPs e com que metodologia devem ser desenvolvidos.

Outro fator importante é qual padrão de comunicação e de empacotamento adotar. No Capítulo 4 serão apresentados os padrões existentes e suas principais características.

No Capítulo 5, o padrão H.264 será apresentado. O próprio decodificador pode ser visto como um IP dentro de um sistema completo de recepção de TV Digital, conforme será descrito na parte de contextualização neste capítulo. É implementado um grande módulo – Predição Intra-Quadro – e é realizada a integração dele com os módulos mais próximos, resultando em uma primeira versão do decodificador H.264. A partir do estudo apresentado na primeira parte da dissertação, é feita uma análise do projeto objeto de estudo, e posteriormente são oferecidas recomendações para a bem-sucedida conclusão do decodificador. De maneira a concluir o decodificador por inteiro, outros módulos, ainda em desenvolvimento, devem ser integrados.

2 METODOLOGIAS DE PROJETO DE SOC

Projeto baseado em reuso é a alternativa que pode melhorar a eficiência de projetos de circuitos integrados complexos, permitindo reduzir o consumo de recursos humanos e técnicos. Em muitos casos, os módulos reutilizados foram produzidos na própria empresa. Entretanto, nem sempre as empresas podem dedicar recursos para o desenvolvimento e manutenção de todos os módulos necessários a uma completa solução *system-on-a-chip*, levando-as à busca junto a desenvolvedores de IP (VSIA, 2006b).

O que acontece na prática é a busca por um acesso a uma maior variedade de módulos prontos, que possibilite a conclusão de projetos e o cumprimento de prazos. Assim, surgem duas classes de profissionais: a de desenvolvedor e a de integrador de módulos. Neste capítulo pretende-se discutir os aspectos referentes ao trabalho do integrador, ou seja, sob o ponto-de-vista de sistema.

Serão apresentados conceitos relativos ao projeto de SoCs, fluxo de projeto do sistema, integração de *cores* e ainda informações referentes à verificação a nível de sistema.

O termo *core* (núcleo, em inglês) será usado de maneira equivalente a módulo de *hardware*, ou seja, é uma sub-parte constituinte do sistema. Também pode ser chamado simplesmente de bloco, porém como bloco é um termo utilizado para descrever uma porção da imagem de um quadro em uma seqüência de vídeo, este termo não será utilizado neste contexto. O termo IP (*Intellectual Property*) se aplica a um *core* quando ele atende a padrões de interface e de documentação, tornando-se um produto.

2.1 Projeto de SoC Genérico

Um SoC pode conter diversos núcleos. A Figura 2.1 ilustra a estrutura de um SoC contendo os seguintes itens (Keating, 2002):

- Microprocessador e seu subsistema de memória;
- Barramentos *on-chip* (de alta velocidade e de baixa velocidade), que permite a comunicação entre os diversos módulos;
- Controlador para memória externa;
- Controlador para comunicações;
- Decodificador de vídeo;
- Controlador de tempo e de interrupções;
- Interface de I/O de propósito geral;

- Interface UART.

Como se pode observar, este sistema tem características de um sistema computacional genérico. Ele apresenta estruturas e objetos encontrados em muitos projetos SoCs reais. Poderiam ser adicionados outros elementos já disponíveis como IPs, como, por exemplo, um processador 8051 ou PowerPC incluindo memórias internas SRAM ou DRAM, memórias externas SRAM ou DRAM ou Flash, mais controladores de porta USB, Ethernet, IEEE 1394, mais conversores DA ou AD, eletro-mecânicos ou eletro-ópticos, um decodificador de vídeo MPEG2 ou AVI, e um controlador de I/O que gerenciasse LEDs ou botões de controle. O mais importante é que a construção de um sistema como esse apresenta todos os elementos básicos e desafios de um projeto SoC, que seriam o desenvolvimento e verificação de blocos, bem como a sua integração em um único circuito integrado.

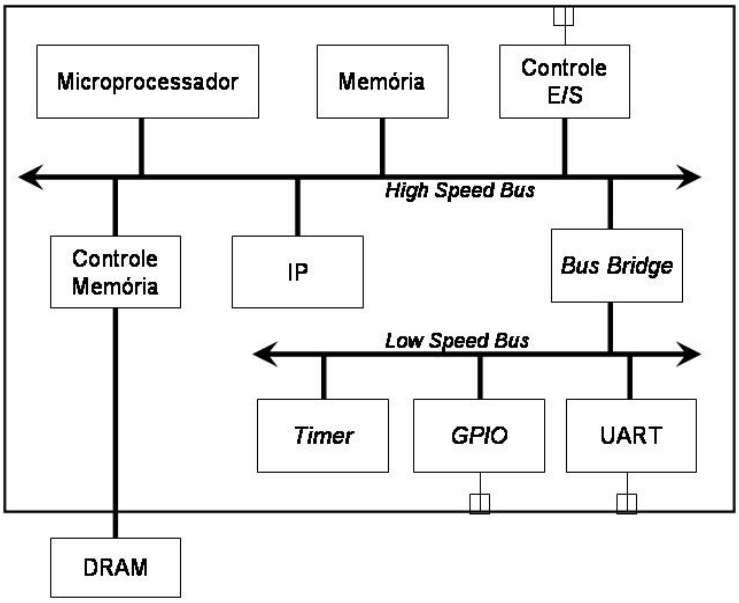

Figura 2.1: Projeto de SoC Genérico (Keating, 2002)

2.2 Fluxo de Projeto do Sistema

De modo a permitir esta transição da forma de projetar para o ponto de vista de SoC, os fluxos de projeto estão mudando de duas principais maneiras:

- Está passando do modelo *waterfall* para o modelo espiral;
- Está passando de uma metodologia *top-down* para uma combinação de *top-down* com *bottom-up*.

O modelo *waterfall* é o modelo tradicional de projeto de ASIC, pelo qual as transições de uma fase de projeto para outra ocorrem sem nunca retornar às atividades da etapa anterior e sem muitas interações entre equipes responsáveis por fases diferentes. Como, normalmente, o desenvolvimento do *software* não é possível sem um modelo do *hardware* a ser usado para depuração, ocorre que o desenvolvimento de *hardware* e *software* acaba sendo serializado. Este modelo pode ser visto na Figura 2.2.

Este modelo de desenvolvimento foi bastante utilizado, produzindo *chips* relativamente complexos que funcionavam bem na primeira vez, mas que causavam

problemas ao serem integrados em sistemas maiores. Neste modelo, a ocorrência de situações onde grupos trabalhando em estágios mais avançados detectem problemas e devolvam o projeto a etapas anteriores é dificultada pelo seu custo. Por exemplo, a equipe de projeto poderia constatar que o algoritmo não era implementável e devolver o projeto para a equipe de especificação, ou a equipe de síntese poderia devolver o código RTL à equipe de projeto porque o código atual não atendia aos requisitos temporais.

Deste modo, para permitir complexidades cada vez maiores obedecendo às pressões de mercado, se faz necessária uma metodologia que permita a realização de várias etapas de forma concorrente e de modo que se desenvolvam, efetivamente, apenas os módulos que já não estejam disponíveis (VSIA, 2006b). Este é o chamado modelo de desenvolvimento em espiral onde são mantidos fluxos de projeto paralelos capazes de interagirem entre si. De fato, a analogia se faz comparando-se uma descida em *waterfall* (cachoeira, em inglês) com uma descida em espiral. Na cachoeira, o caminho seguido é reto e o retorno é bastante caro. No modelo espiral, há um caminho quase horizontal percorrendo etapas que ocorrem paralelamente e pode haver voltas bastante próximas enquanto há um aprimoramento de determinada parte do projeto.

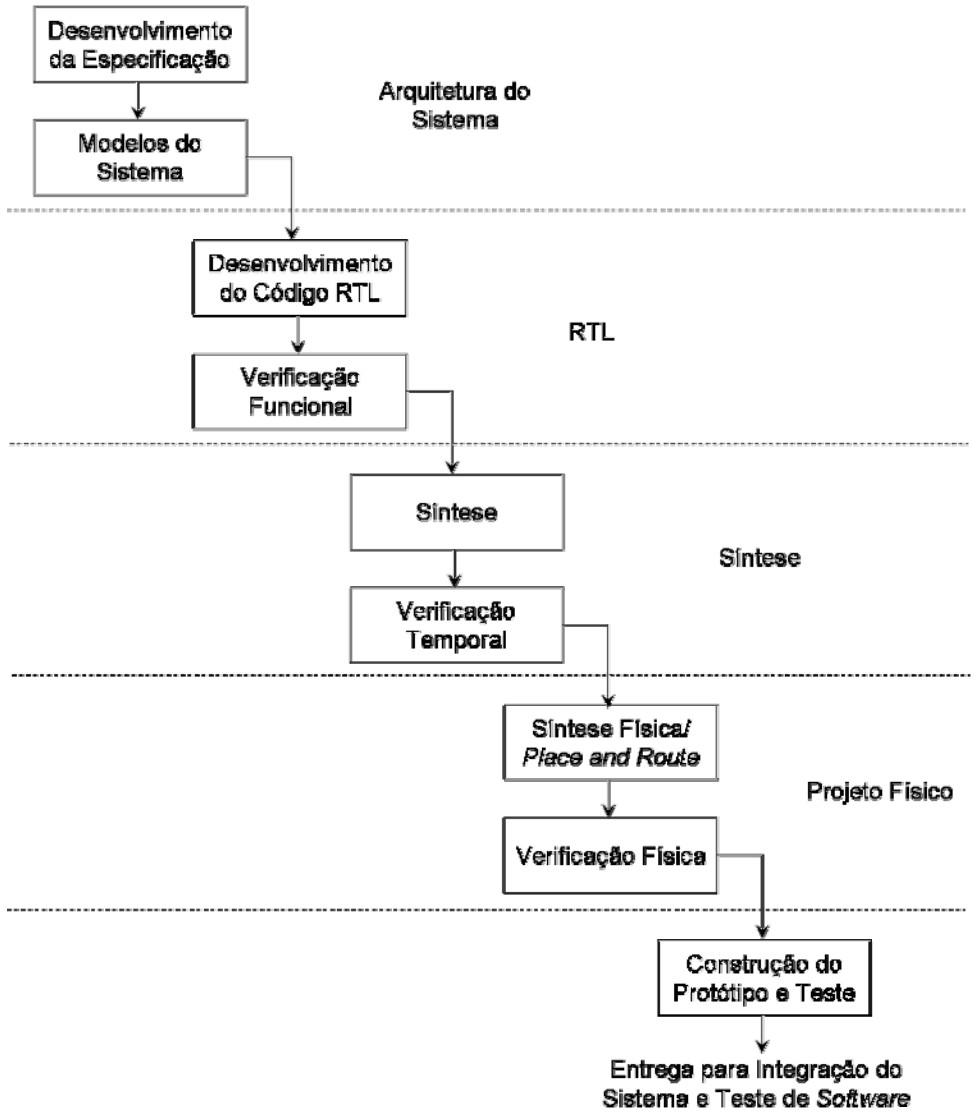

Figura 2.2: Modelo de Desenvolvimento *Waterfall* (Keating, 2002)

Na Figura 2.3 temos o modelo de fluxo de projeto para *chip* complexo sugerido no *Architecture Document* da VSIA (VSIA, 2006b). Deve ser acrescentada uma coluna à direita na qual teríamos o fluxo de projeto de *software*, igualmente em paralelo com os demais fluxos. Durante a etapa de especificação do sistema de *hardware*, é importante o desenvolvimento de um modelo executável que proverá as informações necessárias para que se dê início ao desenvolvimento do *software*. Após a conclusão destas duas partes, haverá ainda a etapa de integração *hardware/sofware* e verificação do sistema.

Etapa de Projeto \ Domínio do Projeto	Projeto do Sistema	Projeto Lógico	Projeto do Teste	Projeto Físico
Especificação do Sistema	Projeto do Algoritmo e da Arquitetura e Particionamento	Função Alvo e Desempenho	Método de Teste e *Testbench*	Potência Necessária, carga e packaging
Detalhamento do Sistema e dos Blocos	Seleção de IPs e Simulação do Sistema	Desenvolvimento e Síntese do HDL	Estruturas de Teste, Vetores e ATG	Floorplan, estimativa do tamanho do bloco, carga
Integração do Chip		Refinamento e Interconexão dos Blocos	Inserção da Estrutura de Teste	*Timing*, Tamanho, Potência
Requisição de Modificação		Modificação e Verificação dos Blocos	Reordenamento Scan Chain	Atualizações post-layout para *timing*, tamanho e potência
Implementação do Chip			Criação do Teste de Produção	Protótipo em Silício e Certificação

Figura 2.3: Modelo de Fluxo de Projeto de *Chip* Complexo (VSIA, 2006b)

Ainda na forma tradicional de desenvolvimento de projeto, adotava-se claramente a estratégia *top-down*, que começa com a especificação e decomposição e culmina na integração e verificação. Esta acaba sendo apenas uma idealização da forma como gostaríamos que o desenvolvimento procedesse. O que ocorre, na prática, são idas e vindas sobre este processo, pois chegando em etapas mais avançadas pode-se constatar que determinados requisitos não são atendidos e deve-se retornar às etapas anteriores de maneira a realizar correções.

A forma atual de dar seguimento ao projeto se trata de uma combinação de *top-down* com *bottom-up*. Isto significa que determinados módulos críticos, que constituirão futuros gargalos em termos de desempenho, área ou potência, passam a ser projetados inicialmente, ao mesmo tempo em que são realizados o refinamento do sistema e a especificação dos demais blocos.

2.2.1 Processo de Projeto do Sistema

Produzir a arquitetura ótima em termos de custo e desempenho envolve um grande número de decisões como, por exemplo, quais partes irão compor o *software* e quais

irão compor o *hardware*, qual processador será utilizado, qual o tipo de barramento e sua velocidade, ou qual o tipo de arquitetura de memória fornecerá um melhor equilíbrio entre potência, área e velocidade (Keating, 2002).

Na maioria dos projetos, não é possível realizar uma simples análise e determinar uma arquitetura para o sistema que cumpra todos os objetivos. É necessária uma modelagem de algumas alternativas de arquiteturas, para que sejam feitos a análise e o refinamento delas até se obter a arquitetura apropriada.

Segundo Keating e Bricaud (Keating, 2002), o processo de especificação e refinamento da arquitetura do sistema pode ser dividido nas seguintes etapas:

1. Criação da especificação do sistema: identificação dos objetivos do sistema, funções requeridas, desempenho, custo e tempo de desenvolvimento do sistema.
2. Desenvolvimento de um modelo comportamental: criação de um modelo comportamental de alto nível para o sistema global. Este modelo será utilizado para testar os algoritmos básicos e mostrar que eles cumprem os requisitos destacados na especificação. Poderá ser utilizado como referência durante as etapas seguintes do projeto servindo, principalmente, como modelo de execução para teste e verificação.
3. Refinamento e teste do modelo comportamental: é desenvolvido um ambiente de verificação para o modelo de alto nível de maneira a refiná-lo e testá-lo. Este ambiente proporciona um refinamento do modelo de alto nível, verificando a funcionalidade e desempenho do algoritmo. Se preparado adequadamente, poderá também ser utilizado posteriormente para verificar os modelos para *hardware* e *software*, como um modelo RTL sendo verificado através de *hardware*/*software* co-simulação.
4. Particionamento entre *software* e *hardware* (decomposição): após o refinamento do modelo de alto nível, é feita a divisão das funcionalidades do sistema em *hardware* e *software*. Esta etapa é comumente um processo manual baseado no julgamento e na experiência de bons projetistas. Uma boa biblioteca contendo *cores* caracterizados e pré-verificados, assim como rotinas reutilizáveis de *software*, é essencial para a identificação do tamanho e desempenho das diversas funções. Finalizando esta etapa, é preciso explicitar as interfaces entre o que é *hardware* e o que é *software*, assim como o protocolo de comunicação a ser utilizado.
5. Especificação e desenvolvimento do modelo arquitetural de *hardware*: a partir da definição dos requisitos de *hardware*, é necessário especificar uma arquitetura detalhada de *hardware*. Isto implica em determinar quais são os seus blocos constituintes e como eles se comunicam. A arquitetura da memória, a estrutura do barramento e a largura do barramento são fatores críticos. A arquitetura final é obtida em um processo de desenvolvimento, teste e modificação dos modelos arquiteturais até o cumprimento dos requisitos do sistema.
6. Refinamento e teste do modelo arquitetural de *hardware* (co-simulação): buscando evitar que o *software* apenas comece a ser desenvolvido após a conclusão do *hardware*, o modelo arquitetural passa a ser utilizado durante o

processo de co-simulação de *hardware* e *software*. Ele já proporciona precisão suficiente para o seu desenvolvimento e depuração.

7. Especificação dos blocos de implementação: a saída obtida durante a atividade de exploração arquitetural produzirá a especificação de *hardware*, que contém uma especificação detalhada de funcionalidade, desempenho e interfaces para o sistema de *hardware* e de seus blocos componentes.

A partir da especificação e do modelo da arquitetura, já é possível iniciar o detalhamento e teste do *software*, concorrentemente ao desenvolvimento do *hardware*. Ao final, após estarem prontos tanto a arquitetura quanto o *software*, eles devem ser integrados e é feito o teste de integração *hardware/software*.

2.3 Integração do Sistema a Partir de IPs Reutilizáveis

A integração de módulos desenvolvidos independentemente é uma tarefa crítica. Mesmo com uma criteriosa definição das interfaces, é freqüente o surgimento de conflitos ou incompatibilidades. Essas podem ser de natureza temporal (*timing* do protocolo), elétrica (impedância, fan-out, amplitude de resposta em freqüência), lógica (polaridade do sinal), semântica (interpretação dos padrões), desempenho (tempo de resposta de um bloco). Embora deva ter sido estudado e resolvido anteriormente, podem ainda aparecer problemas com roteamento, área total ou consumo de potência que estejam fora do especificado.

Após o processo de especificação, aonde foi explicitada a sub-divisão do sistema de acordo com suas funcionalidades, deve-se proceder com a seleção dos blocos IPs que irão desempenhá-las e a preparação deles para a integração. Em seguida, deve-se integrá-los em RTL de alto nível, para então passar para a fase de síntese e análise temporal. A partir daí, pode-se seguir com a síntese física, *floorplaning*, posicionamento e roteamento detalhados. Este processo é utilizado quando se trata de soft-IPs, pois os hard-IPs já são lançados em nível de máscara, o que significa que eles deverão ser inseridos diretamente sobre o *chip*.

Os hard-IPs já devem conter mecanismos de verificação embutidos, como portas JTAG ou *full-scan*, incluindo estruturas para facilitar o debug. Estas estruturas devem ser conectadas ao sistema global, assim como seus anéis de alimentação de energia (*power* e *ground*) e de distribuição de *clock*. Os soft-IPs terão seus mecanismos de distribuição de *clock* e testabilidade inseridos durante a etapa de integração ao nível de *chip* (VSIA, 2006b). No capítulo seguinte, poderão ser encontradas maiores informações sobre a distinção entre soft-IP e hard-IP.

Os grandes problemas encontrados durante a fase de integração dos sub-blocos são devidos ao mau funcionamento das interfaces. Pode haver algum sinal de *handshake* invertido, algum mal-entendido referente à funcionalidade do bloco ou a própria existência de imperfeições no projeto. Outras dificuldades podem ocorrer devido à documentação incompleta, interface do IP que não esteja de acordo com a do barramento e modelos de verificação incompletos.

Deve-se planejar as interfaces antecipadamente de maneira a minimizar o esforço durante a integração. As interfaces devem ser o mais simples possível, e devem seguir padrões de barramento da indústria. Deve-se documentar o conhecimento adquirido durante a utilização de determinado IP, isto é, cada problema detectado e resolvido, o

que facilitará futuros projetos semelhantes através de uma nova utilização do mesmo módulo. A documentação dos IPs e de suas interfaces podem evoluir com o uso do IP.

2.4 Verificação ao Nível de Sistema

Verificar a funcionalidade e o sincronismo em nível de sistema é a etapa mais difícil e importante dentro de um projeto SoC (Keating, 2002). Nesta etapa são concentrados de 50% a 80% dos esforços da equipe de desenvolvimento. É um processo que ocorre ao longo de todo o projeto, envolvendo integralmente todas as demais etapas.

Já na etapa de especificação funcional, são explicitados os critérios para conclusão do projeto (quais testes devem ser completados antes de consentir a fabricação). Em seguida, durante o desenvolvimento do modelo comportamental ao nível de sistema, são gerados *testbenches* e um conjunto de testes para verificar o modelo. Quando o sistema de *software* é desenvolvido, ele será testado utilizando-se o próprio modelo comportamental no lugar de aguardar o *hardware* real ficar pronto. Em resumo, ao final tem-se um rico conjunto de testes que foram sendo criados ao longo do projeto.

Pode-se afirmar que a verificação ao nível de sistema consiste das seguintes etapas básicas (Keating, 2002):

1. Verificação dos blocos constituintes individualmente;
2. Verificação das interfaces entre os blocos, primeiramente em termos de tipos de transações e depois em termos de conteúdo das transações;
3. Execução de aplicações gradualmente mais complexas sobre o *chip*;
4. Prototipagem do *chip* e execução do *software* real para a verificação final (podendo fazer uso de um FPGA, inicialmente, e depois de um protótipo ASIC).

3 FLUXO DE PROJETO DE *CORES*

Na construção de um SoC, após a identificação das funcionalidades existentes, são selecionados IPs para desempenhá-las. Para tanto, é importante dispor de uma biblioteca de componentes reutilizáveis com os quais o sistema é construído. Nesta seção, discutiremos metodologias para o desenvolvimento destes componentes, também chamados de *cores* ou IPs, de maneira a constituir esta biblioteca ou enriquecê-la com novos componentes.

Primeiramente é discutido o termo reusabilidade, passando à descrição das características de um IP e, finalmente, ao seu processo de projeto.

3.1 Reusabilidade

Reusabilidade é qualidade de um objeto que o torna utilizável em diferentes contextos. O resultado é a redução do tempo de projeto a partir do redirecionamento para um novo projeto de um módulo que já foi projetado anteriormente em outro. Entretanto, isto implica em se adotar uma perspectiva de projeto para reuso ao projetar pela primeira vez determinado módulo, desenvolvendo-o da forma mais flexível possível. Assim, aumenta-se a probabilidade de reutilizá-lo, mesmo levando-se em conta as diferenças inerentes entre um projeto e outro.

Um projeto visando reuso exige, em um primeiro momento, as mesmas técnicas exigidas a qualquer projeto correto, como boa documentação, código claro e comentado, ambientes de verificação e teste bem projetados e *scripts* coerentes. Além disso, ainda será necessário adicionar certas técnicas próprias do reuso, como (Keating, 2002):

- Projeto visando resolver um problema mais genérico;
- Projeto visando o uso em múltiplas tecnologias, a partir do emprego de parâmetros a serem fixados na utilização;
- Projeto visando à simulação em diversos simuladores;
- Verificação possível independentemente do *chip* ao qual o bloco será inserido;
- Completa documentação, incluindo as aplicações apropriadas e restrições de uso.

Quanto mais forem obedecidas estas técnicas, maior grau de reuso será obtido para o IP. Entretanto, é preciso ter em mente que a adição de flexibilidade aumenta o tempo de projeto e validação de um módulo podendo, inclusive, reduzir o desempenho (em área, potência ou velocidade). Projetar um IP extremamente específico a determinado projeto resultará em um IP altamente eficiente em termos de desempenho, área e consumo de energia, e diminuirá o seu grau de reusabilidade. Deve-se encontrar uma relação satisfatória entre eficiência e reusabilidade, relevando-se uma redução na primeira em

prol da segunda. Um IP que apresente diversas opções de configuração e que seja passível de implementação em diversas tecnologias poderá ser inserido em muitos projetos, reduzindo seus tempos de desenvolvimento e dividindo o seu próprio custo de desenvolvimento, tornando-se um elemento definitivo de competitividade de um projeto. Vê-se que é um tópico altamente relevante da tecnologia atual.

3.2 Características de um IP

De forma a permitir a sua utilização em uma vasta gama de aplicações, fazendo valer o investimento em torná-lo reutilizável, o IP deve apresentar as seguintes características (Keating, 2002):

- Configurabilidade: esta característica permite que seja possível atender os requisitos de diferentes aplicações.
- Interfaces padrões: a adoção de interfaces padronizadas aceitas pela indústria facilita a integração de diferentes IPs.
- Utilização de boas práticas de projeto: como em qualquer outro projeto, no desenvolvimento de um IP é importante adotar boas práticas de projeto (códigos bem escritos e documentados, entradas e saídas registradas, etc.). Isto facilitará a obtenção de um funcionamento correto e da obtenção do tempo de resposta necessário.
- Conjunto de arquivos completo: para uma devida integração do IP em um sistema, deve ser disponibilizado como um conjunto de arquivos contendo código RTL sintetizável, ambiente de verificação e teste (tanto *stand-alone* quanto a nível de sistema), *scripts* de compilação e síntese além de documentação de uso.

O IP ainda pode ser fornecido sob três formas, que se diferenciam em termos de flexibilidade e eficiência (VSIA, 2006b):

- Hard IP: é específico para determinada tecnologia de fabricação. O processo de desenvolvimento chegou até o nível de leiaute do *chip*. É visto como uma caixa-preta, pois não pode ser re-sintetizado. Apresenta os melhores resultados em matéria de eficiência, contudo apresenta flexibilidade mínima; só pode ser implementado na mesma tecnologia. É relativamente fácil de ser inserido no projeto.
- Soft IP: entregue sob a forma de código RTL, por isso é visto como uma caixa-branca. É totalmente flexível, pois basta sintetizá-lo para a tecnologia desejada. Não é possível prever seu desempenho exato previamente. Apresenta maior dificuldade de inserção em um projeto porque ainda precisa passar por uma parte do fluxo de projeto até chegar ao leiaute.
- Firm IP: permite ao usuário configurar as entradas e saídas, mas sem alterar explicitamente a implementação interna. Assim, módulos internos não desejados podem ser desativados. Oferece um meio-termo entre flexibilidade e eficiência, sendo chamado de caixa-cinza.

Idealmente, uma biblioteca deveria ser constituída de componentes IP que ofereçam tanto Soft IP quanto Hard IP para diversas tecnologias. Deste modo, caso o projetista de sistema esteja trabalhando exatamente sobre alguma destas tecnologias e o IP respeite

os seus requisitos, ele poderá inseri-lo diretamente sobre o leiaute do seu *chip*. Caso contrário, ele deverá partir do Soft IP, ou seja, do código RTL, e proceder com o fluxo de projeto até chegar ao nível de leiaute. Hard IPs oferecem uma maior economia no tempo de projeto sendo, inclusive, imprescindíveis quando se trata de projetos de alto desempenho; porém, infelizmente, nem sempre estão disponíveis. Nestas situações, o Soft IP também é capaz de oferecer um ganho razoável no tempo de projeto.

O provedor de IPs deve, então, se preparar para cumprir certos itens essenciais (Gajski, 2000): manter uma biblioteca de IPs, aonde projetistas possam buscar soluções para os seus projetos; acompanhar o IP de uma documentação completa; garantir a qualidade e testabilidade do IP; utilizar de padronizações, que além de facilitar a comparação entre IPs de provedores concorrentes também tornam a etapa de integração mais ágil e segura.

3.3 Qualidade do IP

Uma vez que blocos IPs são pré-projetados e pré-verificados, o projetista pode se concentrar no nível de sistema sem ter que se preocupar com a exatidão e desempenho dos sub-componentes . Porém, na prática, muitas vezes o IP não traz tanto benefício ao projeto quanto se esperaria, devido a problemas de qualidade do IP. Existem duas principais causas para isto. Primeiro, o IP é projetado para um produto específico, sendo ajustado a ele para cumprir requisitos como velocidade e cobertura de teste, e acaba perdendo o sentido de reuso. Segundo, o custo e o tempo necessários a um IP completamente reusável é muito alto, por isso uma completa reusabilidade é raramente alcançada (Sarkar, 2005).

3.4 Visão Geral do Processo de Projeto de *Cores*

O projeto de um IP pode ser resumido nas etapas apresentadas a seguir:

1. Definição de características-chave: o primeiro passo do projeto de um *core* significa definir as suas principais funcionalidades e parâmetros de configuração que permitirão o seu uso em diferentes aplicações. Para IPs baseados em padrões (p. ex. USB 2.0, IEEE 1394), são incluídas qual a versão escolhida e quais funções serão oferecidas ou não.
2. Planejamento e especificação: nesta etapa são realizadas especificações detalhadas para o *core* e para o VIP, além de um planejamento detalhado do andamento do resto do projeto.
3. Projeto e verificação do IP e do VIP: após a conclusão das especificações, aqui é feita a implementação propriamente dita e a sua verificação.
4. Finalização: após a conclusão do projeto, são feitos testes adicionais e são organizados todos os arquivos que compõem o pacote final do IP.
5. Teste alfa e entrega: teste final do produto gerado na etapa anterior e conclusão desta versão do IP, que a partir deste momento fica disponível para uso.

3.5 Conteúdo da Especificação

A especificação é uma etapa de fundamental importância no desenvolvimento de IPs. É uma etapa onerosa, mas o tempo consumido em definir completa e corretamente o bloco desejado evitará futuros atrasos com correções. Lembrando que correção implica em retornar à etapa anterior e repeti-la até obter o resultado necessário.

Os documentos gerados na etapa de especificação e planejamento irão conduzir o projeto durante toda a sua vida útil. Estes documentos são Especificação Funcional, Especificação da Verificação, Especificação do Conjunto de Arquivos e Plano de Desenvolvimento.

A especificação funcional se trata de uma descrição completa da funcionalidade do projeto sob o ponto de vista do usuário, ou seja, do projetista do SoC ou integrador. A partir desta visão, é possível tratar o projeto localmente como uma atividade autônoma. Esta especificação apresenta definições dos pinos, parâmetros, registradores, e ainda apresenta os requisitos físicos e de desempenho.

A especificação da verificação descreve o ambiente que será usado para verificar o IP, incluindo eventuais modelos BFM e monitores que serão desenvolvidos ou adquiridos para compor o ambiente. Também descreve a abordagem de verificação, quando será usado teste direto ou teste aleatório e quais as métricas que determinam se o IP está pronto.

A especificação do conjunto de arquivos define os *scripts* que são fornecidos como parte do pacote final, incluindo *scripts* de instalação, de configuração e de síntese. Neste pacote serão incluídos todos os arquivos referentes ao IP.

As especificações anteriores constituem o conteúdo técnico do projeto, enquanto o plano de desenvolvimento descreve como elas serão produzidas. Um plano inclui os recursos existentes, como pessoal e equipamento, e como eles serão escalonados. Também faz uma estimativa do tempo de duração de cada etapa do projeto.

Em projetos mais complexos, pode-se desenvolver um modelo de alto nível de maneira a permitir o estudo do algoritmo e de possíveis arquiteturas. Este modelo também poderá compor a própria especificação funcional. Além disso, como outras vantagens, o modelo de alto nível permite que já possam ser desenvolvidos *testbenches* para o projeto e também pode servir de base para o projeto de *software*.

3.6 Metodologia de Particionamento

Na Figura 3.1 é apresentado o fluxo básico de projeto de *cores*, onde é considerado que um *core* seja suficientemente complexo para ser subdividido em sub-blocos. As principais etapas são (Bricaud, 1999):

- (*Core*) Especificação Técnica;
- (Sub-bloco) Especificação Técnica;
- (Sub-bloco) RTL;
- (Sub-bloco) Verificação Funcional;
- (Sub-bloco) Verificação da Implementação;
- (*Core*) RTL;

- (*Core*) Verificação Funcional;
- (*Core*) Verificação da Implementação.

Além dessas, também devem ser incluídas as tarefas de produzir os *scripts* de síntese, rodar a síntese, inserir mecanismos de teste (*boundary scan*, por exemplo) e analisar o consumo de potência.

Na prática, é interessante não se fixar em uma metodologia *top-down*, mas permitir que seja uma combinação entre *top-down* e *bottom-up*, seguindo em frente vez ou outra para comprovar que a especificação que está sendo feita será viável nas etapas de implementação. Também desta forma permite-se que determinadas etapas possam ser executadas em paralelo, seguindo similarmente o diagrama da Figura 2.3, proposto para o fluxo de projeto no nível de sistema.

3.7 Finalização e Lançamento

Na etapa de finalização, é necessário agregar um conjunto de informações a serem entregues ao projetista integrador de maneira a facilitar o seu trabalho. Estas informações consistem do código do IP e dos *testbenches* em Verilog e VHDL, *scripts* de instalação e de síntese, documentação e a informação da versão do IP.

Também nesta etapa, é realizado um *chip* protótipo, seja em FPGA seja em ASIC, para comprovar a utilização final do IP. Devem ser realizadas simulações em diversos simuladores, sínteses sobre múltiplas tecnologias e simulação em nível de porta lógica. Tudo isto de maneira a garantir a reusabilidade sobre diversas plataformas e portabilidade dos *scripts*.

Finalmente, a documentação é atualizada e o IP é passado a uma pessoa que não participou do projeto e que realizará o papel de um projetista integrador testando o conjunto gerado pela fase de finalização. Caso tudo corra bem, o componente é lançado.

3.8 Considerações

Como pode ser acompanhado no Capítulo 5, durante a experiência realizada como estudo de caso, toda a descrição do circuito foi realizada em linguagem VHDL. A própria linguagem permite a criação de parâmetros a serem configurados antes do momento da síntese possibilitando alterar o circuito gerado. O projeto, então, se caracteriza como um soft IP, mas ainda necessita da inserção de certos parâmetros para aumentar a sua flexibilidade.

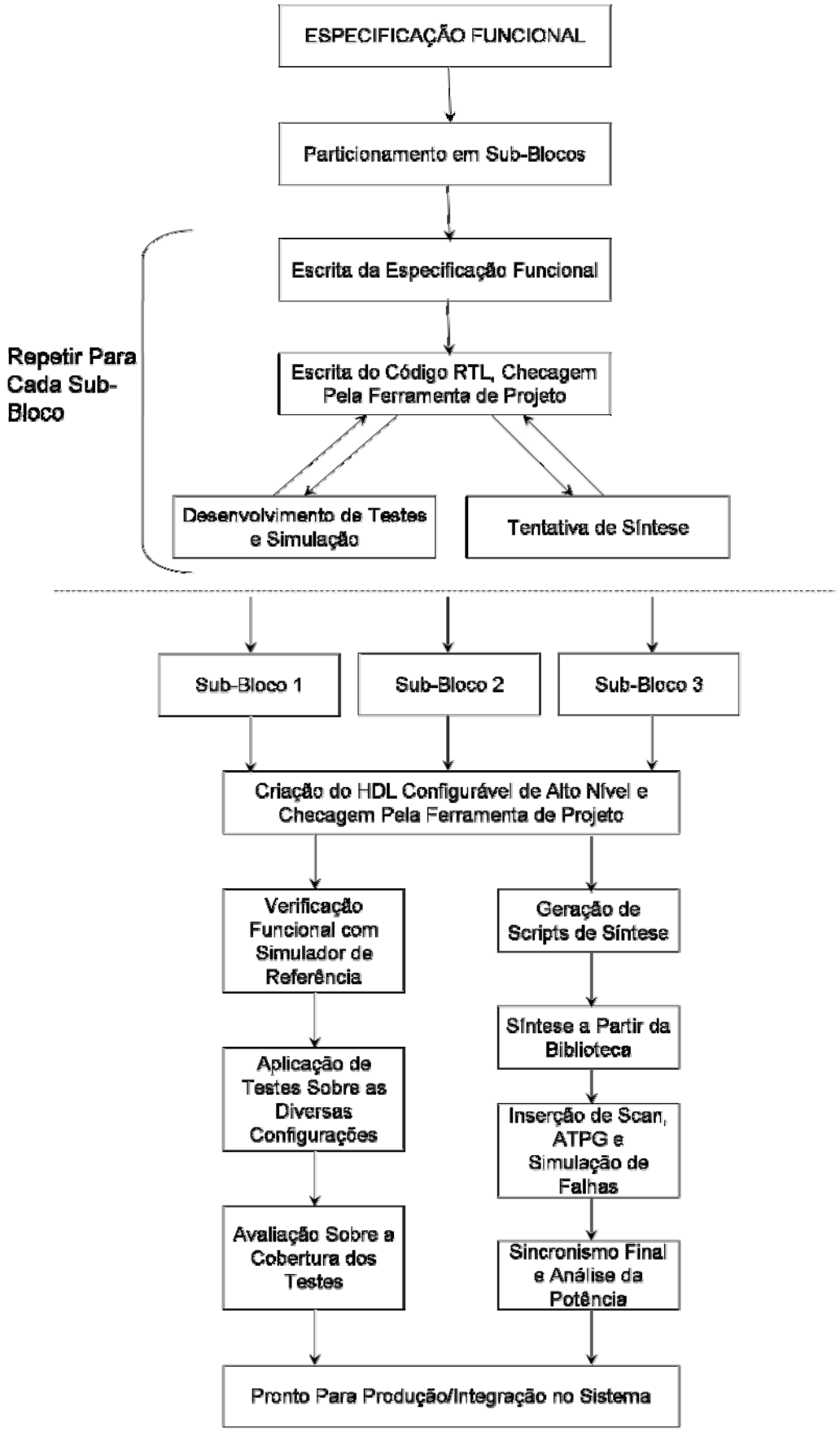

Figura 3.1: Processo de Desenvolvimento de *Cores* (Bricaud, 1999)

4 PADRÕES PARA O DESENVOLVIMENTO E USO DE IPS

Construir um sistema significa mais do que a integração de milhões de transistores. Também é preciso dominar um conjunto de tecnologias e padrões de telecomunicações, de multimídia e de PCs que são complicados e evoluem rapidamente (Hunt, 96).

Neste capítulo são apresentados padrões de desenvolvimento e utilização de IPs em projeto de SoCs. Seguindo o formato canônico ou genérico de projeto SoC, apresentado no Capítulo 2, existem os padrões de barramento AMBA (ARM, 2006) e CoreConnect (IBM, 2006). Existem ainda padrões de barramento de uso mais amplo, como é o caso do OCP-IP (OCP-IP, 2006) e do Wishbone (OpenCores, 2006b), baseados em uma abordagem mestre-escravo que permite também configurações como comunicação ponto-a-ponto.

Todavia, um IP não é formado apenas pelo módulo propriamente dito. Além de apresentar uma interface padronizada que permita a sua fácil integração, ele deve ser acompanhado de um conjunto de informações adicionais que permitam a sua fácil compreensão, configuração e verificação. Para tanto, um IP deve estar acompanhado de uma documentação completa, bem como um ambiente de teste apropriado. Para todos estes arquivos e documentos que acompanham o componente IP existem recomendações de padronização internacional. As maiores iniciativas a respeito, descritas a seguir, são o VSIA (VSIA, 2006) e o OpenCores(OpenCores, 2006).

A padronização de como entregar um IP, incluindo diversos arquivos e documentos, e a padronização de sua interface de comunicação constituem etapas independentes. Deste modo, acabaram por ocorrer naturalmente parcerias entre padrões de um e de outro. As associações mais importantes são entre VSIA e OCP-IP, no campo das grandes indústrias, e entre OpenCores e Wishbone, no campo do código livre.

O uso de um determinado padrão não impede a sua inter-relação com os demais. Uma alternativa para o uso em um barramento de determinado IP formatado segundo outro padrão seria utilizar um IP deste barramento que faça a conversão entre formatos. Por exemplo, pode-se utilizar um módulo IP CoreConnect, cuja funcionalidade seja converter a comunicação deste barramento para o formato Wishbone, permitindo que um IP Wishbone possa ser inserido em um sistema baseado em CoreConnect.

4.1 AMBA

AMBA é um padrão aberto da especificação de um barramento *on-chip* que proporciona uma estratégia para a interconexão e gerência dos blocos funcionais que constituem um SoC. Facilita a operação de integração, uma vez que os blocos são simplesmente conectados a um dos dois barramentos e a estrutura existente permitirá a sua comunicação com os demais módulos, que inclui também um processador ARM. A existência de dois barramentos, um rápido (*Advanced High-performance Bus* – AHB) e um lento (*Advanced Peripheral Bus* – APB), permite que se possa dividir os IPs em dois grupos, melhorando o desempenho do sistema. Assim, aqueles que necessitam de comunicação de alta velocidade não necessitam esperar as transações dos mais lentos, para os quais um barramento mais lento que permita um baixo consumo de energia já é suficiente. Os dois barramentos estão conectados por uma ponte, permitindo a comunicação entre um lado e outro.

A Figura 4.1 apresenta um exemplo da utilização do barramento AMBA com diversos IPs conectados a ele.

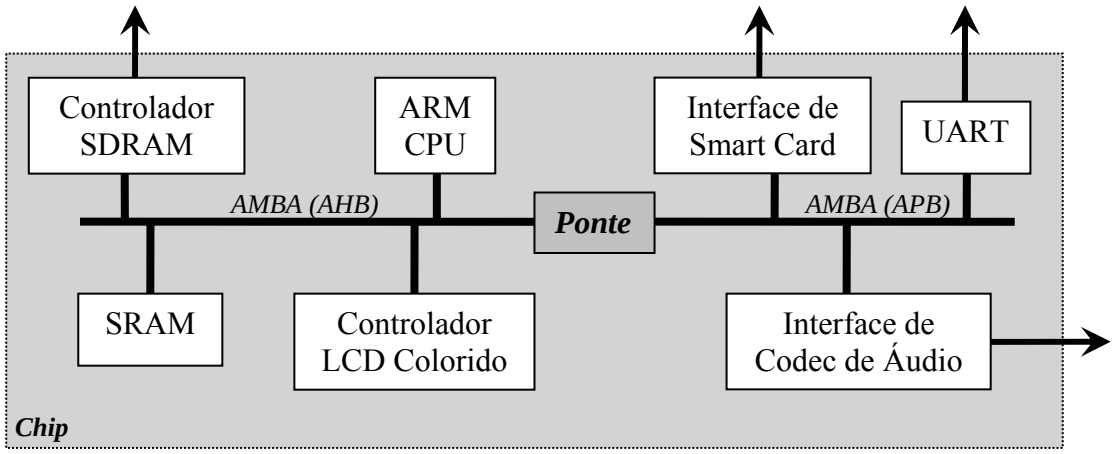

Figura 4.1: Estrutura do Barramento AMBA (ARM, 2006)

4.2 CoreConnect

A topologia do barramento CoreConnect é bastante semelhante ao padrão AMBA. O CoreConnect é igualmente baseado no modelo genérico de projeto SoC. A diferença básica é que este padrão se baseia no processador PowerPC (este sistema foi desenvolvido pela IBM). O CoreConnect tem adicionalmente um anel DCR que conecta os diversos IPs diretamente e pode ser útil em determinadas aplicações.

A Figura 4.2 apresenta um exemplo da utilização do barramento CoreConnect com diversos IPs conectados a ele.

A grande vantagem deste padrão com relação ao AMBA se deve ao fato de um dos maiores fabricantes de FPGA, a Xilinx (Xilinx, 2006), ter adotado a inserção de PowerPCs em seus *chips*. Desta forma, o projetista de SoC terá uma plataforma CoreConnect à disposição, incluindo um processador PowerPC e toda a infra-estrutura para que seus módulos de *hardware* em FPGA se conectem ao sistema.

A empresa Xilinx também oferece fartos recursos no quesito ambiente de projeto. Além do tradicional ISE (Xilinx, 2006c), *software* com o qual podem ser desenvolvidos

projetos de *hardware* em linguagem HDL, também é oferecida a ferramenta XPS (*Xilinx Platform Studio*) (Xilinx, 2006d), com o qual é possível desenvolver um sistema inteiro baseado em CoreConnect. O projetista pode desenvolver o *software*, em linguagem C, para uma aplicação no PowerPC e o *hardware*, em linguagem HDL, a ser implementado fisicamente nas unidades lógicas do FPGA. Quanto ao mecanismo de comunicação, é possível configurar os barramentos através de interfaces e modelos proporcionados pela ferramenta.

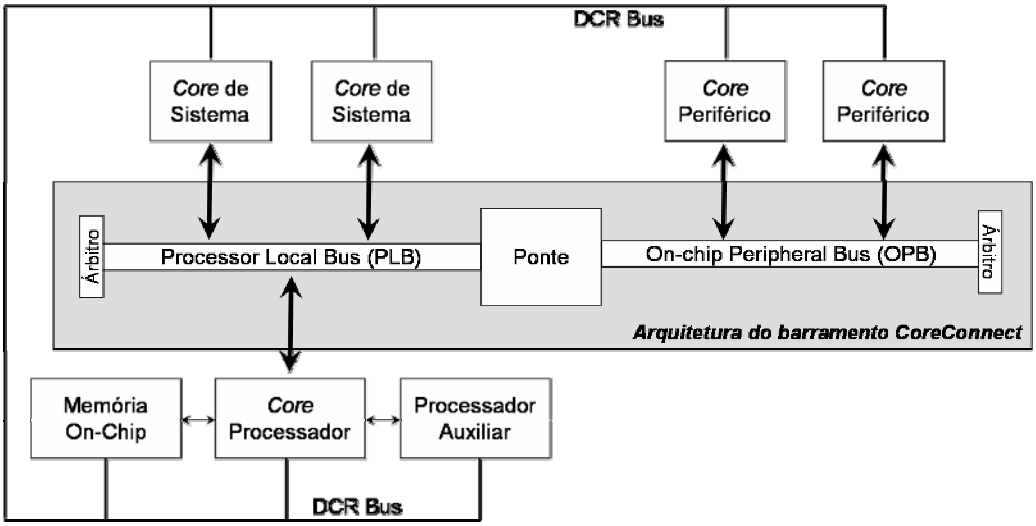

Figura 4.2: Estrutura do Barramento CoreConnect (IBM, 2006)

4.3 VSIA – Virtual Socket Interface Alliance

O principal objetivo da padronização VSIA é especificar/recomendar um conjunto de interfaces, formatos e práticas de projeto de *hardware* e *software*, para a criação de blocos funcionais que permitam a integração, verificação e teste eficientes e precisos, de múltiplos blocos em um único *chip* (VSIA, 2006b).

A partir da definição de um conjunto de características de interface que seja comum, pode haver um intercâmbio de blocos dentro da indústria, facilitando a construção de sistemas maiores em um menor espaço de tempo.

Segundo esta organização, todo projeto pode ser dividido em duas áreas básicas: criação de módulos e integração de módulos, aqui chamados de VC – *Virtual Component*. A parte de criação consiste no desenvolvimento de VCs que poderão ser utilizados e re-utilizados tanto *stand-alone* quanto como componente em um sistema maior. A parte de integração consiste na integração e verificação de múltiplos VCs sobre um único *chip*. Sob esta visão, os projetistas podem se enquadrar em duas classes: a de desenvolvedor de VC ou a de integrador de VC. Os dados que são trocados entre um e outro constituem o domínio do padrão VSIA.

4.4 OCP-IP

Open Core Protocol International Partnership (OCP-IP) define uma interface de alto desempenho, independente de barramento, entre componentes IP visando reduzir tempo de projeto, risco de projeto e custos de manufatura para projetos SoC (OCP-IP, 2006b).

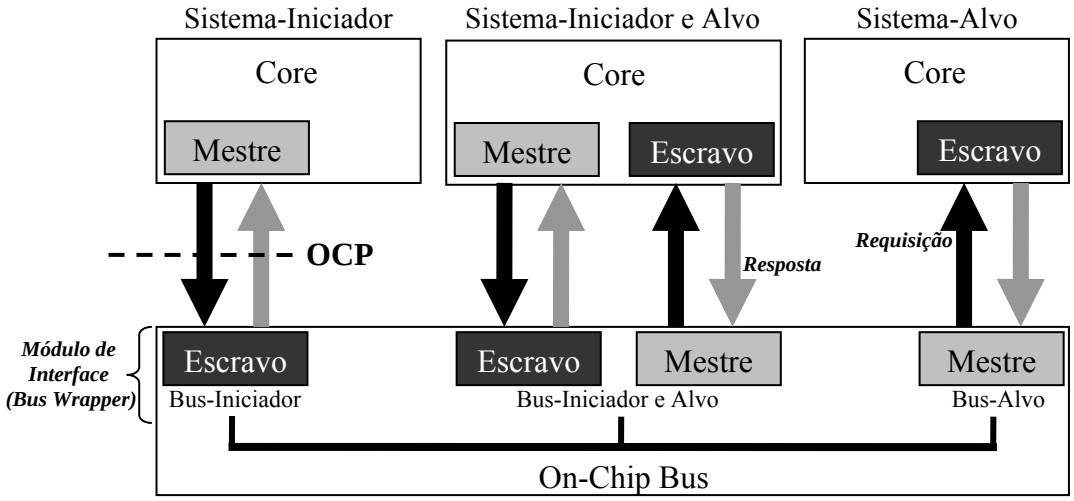

Figura 4.3: Estrutura da Interface OCP

A comunicação é sempre realizada entre duas entidades, uma sendo mestre e uma sendo escrava. Para o caso de uma comunicação ponto-a-ponto, bastam duas instâncias, uma mestre e uma escrava. Quando houver necessidade de um barramento, como o mostrado na Figura 4.3, é preciso criar instâncias complementares a cada IP no barramento.

A Tabela 4.1 apresenta os principais sinais presentes em uma interface OCP.

Tabela 4.1: Sinais Básicos da Interface OCP (OCP-IP, 2006b)

Nome	***Largura***	***Entidade***	***Função***
Clk	1	Ambas	OCP clock
MAddr	Configurável	Mestre	Transferências de endereços
MCmd	3	Mestre	Transferência de comandos
MData	Configurável	Mestre	Escreve informação
MDataValid	1	Mestre	Escreve informação (válido)
MRespAccept	1	Mestre	Mestre aceita resposta
SCmdAccept	1	Escrava	Escrava aceita transferência
SData	Configurável	Escrava	Lê informação
SDataAccept	1	Escrava	Escrava aceita escrita de informação
SResp	2	Escrava	Transfere resposta

Além destes sinais básicos, existem diversos outros sinais opcionais, principalmente sinais de teste.

4.5 OpenCores

A proposta da organização OpenCores (OpenCores, 2006) é desenvolver um grupo de intercâmbio de componentes IP baseado na filosofia de *software* livre. Eles também entendem que os projetos complexos só podem ser finalizados em curto prazo se forem tratados através de uma abordagem SoC baseado no reuso de componentes. Entretanto, visando uma classe de projetos menores tratados por empresas menores, aonde a aquisição de componentes caros ou a utilização de ferramentas comerciais tornam-se proibitivas, OpenCores tem por objetivo distribuir componentes sob licença Lesser GPL e utilizar apenas ferramentas *freeware*.

A licença *Lesser General Public License* – LGPL – (Gnu, 2006) é uma licença GNU, criada inicialmente para código de *software*. A licença GPL (hoje chamada de licença ordinária GPL) obriga que qualquer *software* que utilize código ou biblioteca sob esta licença também seja completamente público e de uso gratuito. Enquanto isso, a LGPL permite que sejam adicionados códigos ou bibliotecas privadas a códigos ou bibliotecas públicas. Caso os componentes de uso público estejam sob a licença LGPL, o produto final pode ser comercializado com a parte privada protegida. Desta forma, se consegue uma adesão mais abrangente por parte da indústria já que todos poderão adotar a mesma biblioteca pública.

Trazendo esta filosofia para o campo de projeto de sistemas digitais, a partir da organização OpenSources, que promove o intercâmbio de componentes de *software*, foi criado o OpenCores, voltado a módulos de *hardware*. E na prática de projetos baseado em reuso, OpenCores oferece uma série de recomendações e padrões de maneira a facilitar o intercâmbio de componentes e a sua posterior integração e verificação. Todos os códigos são abertos e permanecem disponíveis no *site*, e todas as plataformas e ferramentas de desenvolvimento recomendadas são de uso gratuito.

4.6 Wishbone

O padrão de interface de barramento Wishbone (OpenCores, 2006b) é bastante similar ao OCP-IP. É utilizado como padrão de interface de componentes IP pela organização OpenCores por se tratar de um padrão completamente aberto e gratuito.

A comunicação é tratada da mesma maneira que no OCP, sempre através de pares mestre-escravo. Desta maneira é possível utilizá-lo tanto para comunicações ponto-a-ponto quanto via barramento ou ainda para aplicações tipo *dataflow*.

4.7 Considerações

Em questões práticas, o padrão adotado para a implementação apresentada foi o CoreConnect. Em um primeiro momento, esta decisão se deu devido ao fato de que havia à disposição do grupo de pesquisa em TV Digital placas Digilent XUPV2P contendo FPGA da Xilinx que, por sua vez, contém um processador IBM PowerPC embarcado. Vale enfatizar que este padrão não apresenta nenhuma vantagem ou desvantagem com relação aos demais padrões, visto que todos têm por objetivo trazer o projeto em andamento para uma estrutura que, se gostaria, fosse também utilizada por todos os outros projetistas do mundo. Desde modo, o projeto obteria alcance máximo, uma vez que o produto poderia ser integrado e se comunicar com qualquer outro porque estaria utilizando a mesma interface.

A estrutura do *chip* da Xilinx já vem preparada para a realização de projetos *hardware/software* co-design, sendo a parte de *hardware* instanciada sobre o FPGA e a parte de *software* sendo uma aplicação rodando no PowerPC. Diversos módulos de *hardware* em FPGA podem se comunicar entre si e com o PowerPC através justamente da estrutura de barramento CoreConnect. O conjunto de ferramentas providas pela Xilinx oferece suporte completo ao projeto nestes termos.

Já na parte de fechamento do IP, o padrão adotado para a estrutura de diretórios, organização dos *testbenches*, formatação dos códigos, etc., foi o VSIA. As recomendações da entidade VSIA são seguidas pela grande maioria dos projetistas no mundo inteiro, visto que trinta das maiores empresas da área de microeletrônica já se tornaram membros desta organização. Deste modo, a sua utilização permitirá a um eventual IP resultante deste trabalho um alcance mais amplo.

5 REFLEXÕES BASEADAS NA REALIZAÇÃO DE UM PROJETO REAL

A melhor maneira de prover uma análise sobre a importância da utilização da metodologia apresentada e defendida neste trabalho se dá através da apresentação de um projeto que contenha exemplos tanto da utilização do método baseado em reuso de Ips quanto da sua não utilização. Assim é possível realizar uma comparação, e fazer reflexões sobre as suas vantagens e implicações.

O Capítulo 5 é organizado da seguinte maneira: na primeira seção é dada uma visão geral de um sistema de televisão digital e da necessidade de um mecanismo de compressão de vídeo, justificando a utilização do padrão H.264 e de sua implementação em *hardware*. Na seção seguinte, o padrão de compressão H.264 é descrito brevemente. Em seguida, é apresentada a descrição do projeto do sub-módulo de predição intra 4x4, e os dois processos de integração, primeiro em nível de módulo e depois em nível de decodificador. Ao final, é feita uma reflexão sobre a realização deste projeto e é sugerida uma forma de conclusão para o decodificador utilizando-se o IP produzido acrescido dos módulos restantes.

5.1 Projeto do Decodificador H.264

Inicialmente, este caso foi escolhido porque o decodificador H.264 é um projeto suficientemente grande para ser tratado como um sistema digital complexo. É o algoritmo de compressão de vídeo do estado-da-arte, e é intuitivamente subdividido em diversos módulos de acordo com as operações efetuadas sobre o sinal de vídeo. Cada um dos módulos também pode ser subdividido reduzindo-se a sua complexidade, através da aplicação recursiva do método. Este estudo de caso foi orientado a partir de um sub-módulo do decodificador H.264, chamado Intra 4x4. Desta maneira, é apresentado o seu desenvolvimento, a sua integração com os demais sub-módulos obtendo-se o módulo de Predição Intra-Quadro, a integração deste com mais dois outros módulos do decodificador gerando um IP CoreConnect.

5.1.1 Sistema Brasileiro de Televisão Digital – SBTVD

Este projeto foi motivado pela iniciativa de desenvolver um sistema brasileiro de televisão digital, o SBTVD (SBTVD, 2006). Como mostrado na Figura 5.1, um sistema de TV digital é composto basicamente por um codificador de vídeo, um codificador de áudio, um multiplexador e um modulador do lado do transmissor, e seus elementos simétricos do lado do receptor. A TV digital vai permitir o envio de dados adicionais,

como programação, letras de músicas, etc., multiplexados com os sinais de áudio e vídeo. Outra novidade é o canal de interatividade, que vai permitir o envio de dados por parte do usuário, isto é, a sua interação com a empresa distribuidora do sinal de conteúdo. Neste trabalho é dado particular enfoque à tarefa de compressão e descompressão de vídeo.

Existem diversas técnicas matemáticas destinadas a reduzir a quantidade de dados de um sinal digital de vídeo a ser transmitido. No entanto, quando se fala de implementação em *hardware* há um fator limitante que é o número de transistores que podem ser projetados sobre um *chip*. Um algoritmo muito complexo vai exigir seguramente uma grande capacidade de processamento, o que implica em um grande número de transistores. A evolução da microeletrônica e dos processos de fabricação de circuitos integrados vai colocando à disposição um número cada vez maior de transistores, permitindo que sejam implementadas técnicas de compressão mais complexas e inseridos tratamentos para casos pouco prováveis, de modo a obter altas taxas de compressão em qualquer tipo de vídeo.

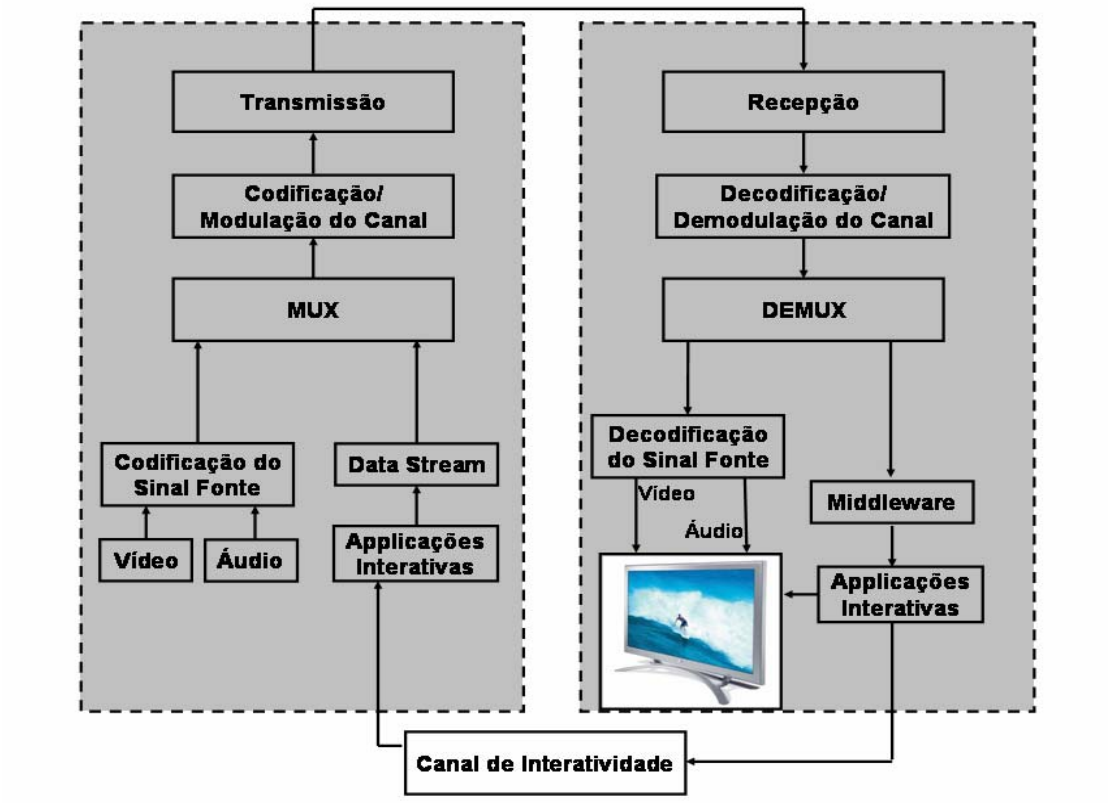

Figura 5.1: Sistema de TV Digital

Determinados algoritmos, que antes não podiam ser implementados ou inseridos no sistema devido à falta de tecnologia, hoje foram incorporados ao padrão H.264. Contudo, ainda não é possível realizar o processamento em tempo real nem do codificador nem do decodificador H.264, perfil Main, em *software*. Codificação e decodificação em tempo real significa realizar o processamento no mesmo tempo em que o vídeo é exibido; por exemplo, caso se trate de um vídeo HDTV (*High Definition Digital Television*), deve-se processar os quadros de resolução 1920x1080 pixels a uma taxa de 30 quadros por segundo, assim como é feita a exibição. E realizando-se o

processamento tanto da codificação quanto da decodificação em *software* rodando no melhor processador de propósito geral da atualidade, não será obtida a mesma taxa daquela de exibição. Por exemplo, para decodificar em tempo real um vídeo de resolução SDTV (720x480, 30 quadros por segundo), em *software*, conforme o padrão H.264 perfil Main, é necessário um processador Pentium IV de 2.4 GHz (VSS, 2006). Este é um dos melhores processadores da atualidade, e não será capaz de decodificar um vídeo de resolução HDTV (1920x1080), tampouco será capaz de realizar a codificação. A única maneira de realizar esta tarefa em tempo real é através da construção de um *hardware* dedicado.

Este projeto também é justificado pela necessidade deste codec em silício para que possa ser embarcado em um sistema de TV digital, uma vez que há a necessidade de torná-lo um produto independente da presença do computador. O processamento de vídeo de alta resolução impõe sérios requisitos de desempenho, tornando o projeto ainda mais complexo.

5.1.2 Visão Geral do Padrão de Compressão de Vídeo H.264

H.264 é um padrão de compressão de vídeo, a própria evolução do MPEG1, MPEG2, H.261 e H.263. Por se tratar de um esforço conjunto entre ITU-T (ITU-T, 2006), responsável pelos H.26x, e ISO (ISSO, 2006), responsável pelos MPEG, comunidades de especialistas em compressão de vídeo, o H.264 apresenta resultados excelentes nas mais diversas aplicações, de vídeo-conferência até difusão de TV de alta resolução. H.264 permite uma redução da ordem de cem vezes na quantidade de informação a ser transmitida. Isto quer dizer que ele é, em geral, duas vezes mais eficiente do que o MPEG2. Para o projeto em questão, há um particular interesse na sua utilização para aplicações de difusão de TV.

Através de uma cooperação entre o Grupo de Microeletrônica (GME) do Instituto de Informática da UFRGS e do Laboratório de Processamento de Sinais e Imagens (LaPSI) do Departamento de Engenharia Elétrica da UFRGS, iniciou-se a implementação deste padrão de compressão de vídeo: um sistema de *hardware* capaz de realizar a codificação segundo o padrão H.264 e outro capaz de realizar a decodificação. Como base para este projeto, foi utilizada a norma H.264 (ITU-T, 2005), documento que descreve os algoritmos e organização dos dados no *bistream*, e o código de referência (ITU-T, 2005b), que é uma descrição em linguagem C do algoritmo para servir como exemplo e para geração de arquivos de teste.

Um codificador H.264, dividido em módulos segundo as diversas operações que são aplicadas sobre o sinal digital de vídeo, pode ser visto na Figura 5.2. Uma descrição completa da norma H.264 é apresentada no Apêndice A, assim nesta seção é dada apenas uma visão geral do algoritmo, suficiente para a descrição do trabalho.

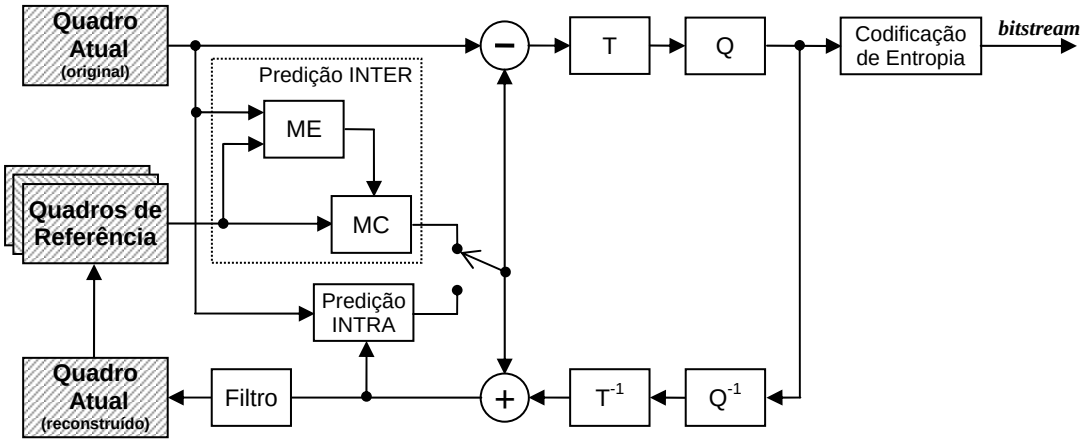

Figura 5.2: Diagrama de Blocos do Codificador H.264

A principal tarefa de um codificador de vídeo é encontrar e eliminar as redundâncias presentes no sinal. O vídeo é organizado em quadros que são divididos em regiões de 16x16 pixels, chamadas macroblocos, que são processadas uma de cada vez. Ao processar um macrobloco, o codificador compara-o com os macroblocos já processados até o momento, tanto de outros quadros como do mesmo quadro, em busca de similaridades. No lugar de transmitir todos os 16x16=256 pixels, o codificador vai enviar apenas uma referência ao macrobloco anterior que mais se assemelha a ele e um macrobloco resíduo que é a diferença entre eles. O resíduo ainda é passado pela transformada e pela quantização para também ser comprimido. No final, é feita a codificação de entropia, baseada na estatística do sinal, e é emitido o *bitstream* que é o sinal de vídeo já codificado (Richardson, 2003).

Caso a busca por um macrobloco semelhante seja feita em outros quadros, o processo é chamado de Predição Interquadro ou Estimação de Movimento (ME). Caso a busca seja feita dentro do mesmo quadro, ela é chamada de Predição Intra-Quadro.

O processo completo de compressão realizado sobre o sinal digital de vídeo apresenta perdas, ou seja, o vídeo reconstruído no decodificador não é idêntico ao original. Para evitar a propagação de erros, as predições são feitas utilizando como referência os macroblocos codificados e decodificados, sendo exatamente os mesmos que o decodificador utilizará para reconstituir o sinal. Por isso, pode-se ver na Figura 5.2 uma realimentação do sinal. Trata-se de um decodificador completo dentro do codificador que provê os valores a serem utilizados como referência para o ME e para a Predição Intra.

Pela ordem de complexidade e também pelo fato do decodificador ser parte constituinte do sistema codificador, o projeto foi iniciado pelo desenvolvimento do decodificador, objeto deste estudo de caso.

O decodificador pode ser visto na Figura 5.3. Primeiramente, o sinal codificado passa pelo decodificador de entropia (composto por CAVLC, ExpGolomb e CABAC) e pelo *parser*. O *parser* é um analisador sintático, isto é, depois que os símbolos são convertidos em elementos sintáticos pelo decodificador de entropia, eles devem ser analisados pelo *parser*, que resgata todas as informações que o controle necessita para encaminhar cada dado ao respectivo módulo com as devidas configurações. No decodificador, também há o compensador de movimento (MC) e o módulo de predição intra-quadro, responsáveis por reconstruir amostras que foram codificadas fazendo referência a amostras de outros quadros ou do mesmo quadro, respectivamente. O

módulo indicado por Q^{-1}/T^{-1} é responsável pela quantização e transformada inversas. E por último, temos o somador e o filtro de redução de efeito de bloco. O filtro é responsável por suavizar quaisquer vestígios do processo que poderia resultar em uma imagem quadriculada, e é a última etapa antes da exibição.

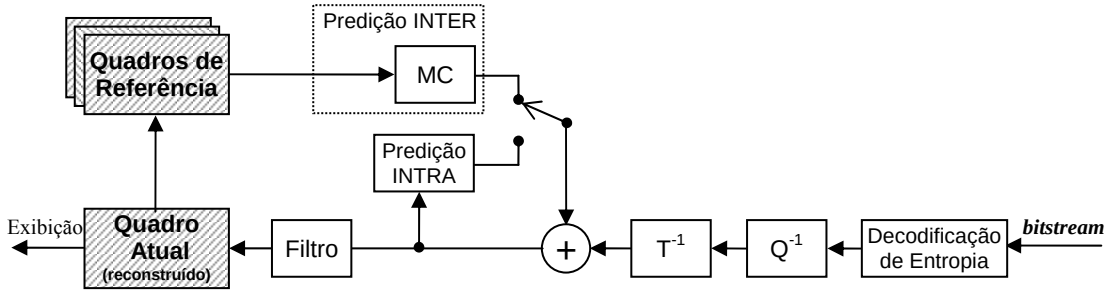

Figura 5.3: Diagrama de Blocos do Decodificador H.264

5.1.3 Sub-Módulo de Predição Intra 4x4

O espaço de amostragem de cores utilizado no H.264 é o YCbCr, onde Y é a informação de luminância e Cb e Cr são componentes de crominância. Como nossa visão é mais sensível ao brilho que a cor, pode-se subamostrar as componentes Cb e Cr reduzindo a quantidade total de dados sem perda na qualidade subjetiva (Richardson, 2003). A Figura 5.4 apresenta três espaços de subamostragem de cor, representando a componente de luminância por 'X' e as componentes de crominância por 'O'. No espaço 4:4:4, temos uma amostra de croma para cada amostra de luma. No espaço 4:2:2, temos uma amostra de croma para cada duas de luma. E no espaço 4:2:0, temos uma amostra de croma para cada quatro amostras de luma. Esta última é utilizada no padrão H.264 perfil Main e, portanto, os mecanismos implementados neste trabalho são baseados nela.

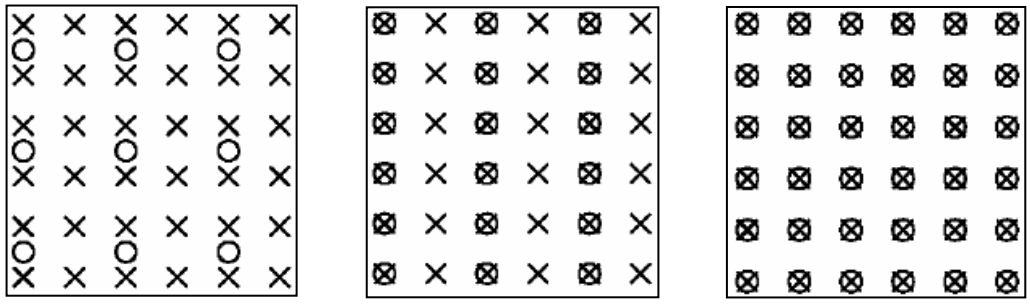

Figura 5.4: Espaços de Subamostragem de Cor

Um quadro de crominância representa, então, um quarto do quadro de luminância, com metade da dimensão vertical e metade da dimensão horizontal. Sendo assim, o macrobloco de crominância é composto por 8x8 amostras no lugar de 16x16. Na predição de luma, também é possível subdividir o macrobloco em blocos de 4x4 pixels e realizar a predição de cada um individualmente, o que provê melhores taxas de compressão em regiões da imagem com mais detalhes. Também é necessário um buffer, que serve para o armazenamento das amostras já processadas que servirão de referência para as predições seguintes. A partir destas informações, temos que o módulo de predição intra-quadro é dividido em quatro partes: predição 16x16 de luma, predição

4x4 de luma, predição de croma e um buffer. Nesta sub-seção, será dado enfoque à predição de amostras de luma agrupadas em regiões de 4x4 pixels.

5.1.3.1 Desenvolvimento do Projeto

O projeto do sub-módulo de predição intra 4x4 foi desenvolvido através das seguintes etapas (Staehler, 2006):

1. Estudo do algoritmo: a partir da leitura da norma H.264 (ITU-T, 2005), foi possível compreender o mecanismo do algoritmo e as técnicas matemáticas envolvidas;
2. Especificação: em seguida, pôde-se elaborar uma arquitetura de *hardware* para um sistema capaz de desempenhar as funções identificadas na etapa anterior;
3. Modelo executável: para afirmar que a especificação estava realmente correta e condizente com a norma, foi desenvolvido um modelo executável em ambiente Matlab para a predição 4x4;
4. Escrita do código VHDL: após a conclusão do modelo, foi escrito o código VHDL descrevendo a arquitetura;
5. Simulação: juntamente à norma H.264 existe um código de referência H.264 (ITU-T, 2005b), que é uma descrição do algoritmo em código C. A partir deste código de referência foi possível extrair arquivos de entrada e de saída para o respectivo módulo durante a execução de vídeos de teste. Através da utilização destes arquivos pôde-se realizar o teste e validação do código VHDL por meio de simulação em ambiente ModelSim (Mentor, 2006);
6. Prototipagem: por último, foi realizada a prototipagem sobre a placa Digilent XUPV2P (Digilent, 2006).

O Apêndice B apresenta uma descrição completa da arquitetura de predição intra 4x4, incluindo as fórmulas matemáticas do algoritmo, código Matlab e descrição VHDL.

A seção seguinte descreve o procedimento de prototipagem.

5.1.3.2 Plataforma de Prototipagem

Nesta sub-seção apresenta-se a metodologia de prototipagem tendo-se por objetivo colocar os módulos projetados em funcionamento diretamente em *hardware*, de forma que o sistema possa executar de maneira independente, apenas recebendo os dados codificados, decodificando-os e exibindo o resultado em um televisor ou monitor. Mesmo que o projeto já esteja validado em uma simulação pós *place&route*, há a necessidade de comprovar o seu correto funcionamento em *hardware*, já que em uma situação real de execução existem diversos fatores atrelados à plataforma de prototipagem que podem não ter sido corretamente levados em conta durante a fase de simulação.

Desta forma, após um estudo sobre a plataforma a ser escolhida para este projeto, o grupo acabou por definir a placa Sundance SMT395E (Sundance, 2001), que contém um *chip* Xilinx Virtex II Pro de 70.000 células lógicas. Entretanto, no aguardo do recebimento desta placa, os primeiros experimentos (estes apresentados no presente documento) foram realizados sobre a placa Digilent XUPV2P (Digilent, 2006), que

contém um *chip* Xilinx VP30 (Xilinx, 2006b) de 30.000 células lógicas. Ambas as plataformas possuem diversas interfaces de entrada e saída, permitindo a entrada de dados e a exibição do resultado, bem como quantidade de memória e área de FPGA que, acredita-se, seja suficiente, inclusive, para a prototipagem de um codificador H.264 (com exceção da segunda placa, que possui uma área de FPGA menor que a metade da primeira). Ambas as plataformas também apresentam processadores PowerPC que são capazes de se comunicar com os diversos periféricos e também com o *hardware* em funcionamento no FPGA.

Em um primeiro momento, o processador PowerPC é utilizado para rodar programas de monitoramento para depuração da arquitetura de *hardware*. No futuro, ele poderá ser utilizado para a execução de parte do sistema decodificador em *software*. De qualquer modo, é aplicado aqui um conceito conhecido por *hardware/software* co-design. O conteúdo de software da maioria dos sistemas eletrônicos vem crescendo nos últimos anos e, em alguns casos, chega a ser a maior parte do produto final e, conseqüentemente, do esforço de projeto (Hunt, 96). Desde que se passou a trabalhar com sistemas intrachip, a metodologia de projeto de desenvolvimento de tais chips deve responder às necessidades tanto de hardware quanto de software. Mais do que isso, conforme o tamanho e complexidade dos chips vêm crescendo, assim também é a tarefa de verificação, visto que verificar o projeto de um circuito contendo um milhão de transistores não é nada trivial. Com esta motivação, foi tirado proveito da plataforma de prototipagem disponível, naturalmente adequada ao hardware/software co-design.

Metodologia de Validação na Placa Digilent XUPV2P

Conforme citado anteriormente, esta placa apresenta um *chip* Xilinx Virtex II Pro de 30.000 células lógicas, dois PowerPC, um *slot* para memória RAM DDR, interfaces serial, ethernet e saída VGA (para monitor de vídeo), entre outros.

Tendo por objetivo colocar um módulo do decodificador em uma situação real de execução, desenvolveu-se um sistema de validação no qual o módulo é mapeado para o FPGA da placa, e o PowerPC se comporta como os seus módulos adjacentes. Isto significa que o PowerPC faz o papel do controle, enviando os comandos e *flags* para o módulo que está sendo prototipado, além de também fornecer os dados de entrada e receber os dados de saída, se comportando como os módulos que estariam antes e depois do módulo em questão na ordem de processamento. Além disto, o PowerPC deve comandar os periféricos da placa, com os quais é possível fornecer dados a partir do computador do projetista e também monitorar o que está se passando na placa.

Os dados de entrada a serem aplicados à arquitetura foram extraídos do código de referência e a saída é também comparada com aquela obtida pela execução deste código. Maiores informações sobre a extração destes dados a partir do código de referência e a geração de arquivos de teste pode ser vista em (Figueiró, 2005).

Para pôr em prática a estratégia apresentada acima, utilizou-se a ferramenta de projeto EDK (Xilinx, 2006c). Nesta ferramenta, temos um editor de texto aonde pode ser escrito tanto código VHDL (a ser mapeado para o FPGA) quanto código C (a ser executado pelo PowerPC). São gerados *scripts* que chamam ferramentas para a realização da síntese do código VHDL, mapeamento da arquitetura projetada para o *chip* em questão e *download* do projeto na placa (via interface paralela ou USB). Há

também disponível uma versão do compilador C (gcc) que gera o executável para o PowerPC a partir do código C escrito pelo projetista.

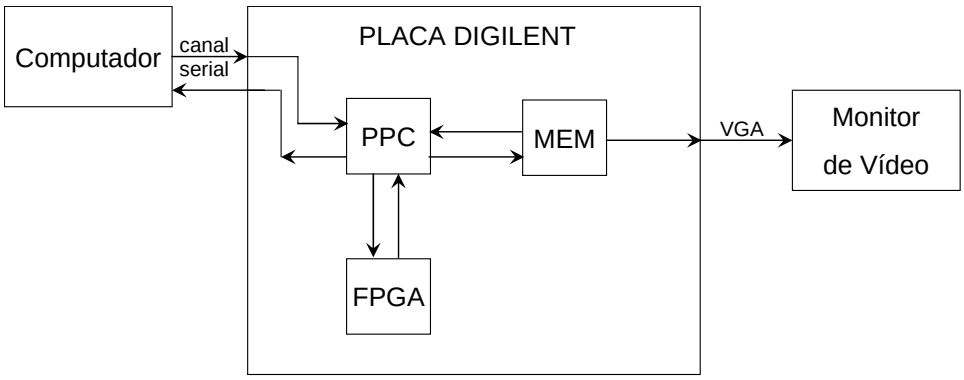

Figura 5.5: Fluxo de dados na plataforma de prototipagem

Na Figura 5.5 está descrito o fluxo de dados adotado para a prototipagem do módulo de predição intraquadro apresentado no capítulo anterior, e seguido para os demais módulos que foram concluídos. Basicamente, o computador do projetista envia os dados que servirão de entrada para a arquitetura via porta serial. Também há a possibilidade desta comunicação ser realizada através da interface Ethernet, que é muito mais rápida, porém isto ainda está em fase de desenvolvimento. Pelo fato de que a interface serial é bastante lenta haja vista a taxa de dados de entrada necessária ao funcionamento da arquitetura, o PowerPC não pode esperar que os dados cheguem via serial para, então, repassá-los à arquitetura. O que ocorre é que, em um primeiro momento, é feita a comunicação serial, o computador passa ao PowerPC todos as entradas referentes a um dado experimento e este os armazena na memória DDR presente (um pente de 512 MB). Em um segundo momento, então, o PowerPC vai lendo os dados da memória e repassando à arquitetura, que os processa e devolve o resultado. Este resultado é colhido pelo PowerPC e devolvido ao computador do projetista (via serial), além de ser armazenado na memória. Existe a necessidade de se armazenar o resultado na memória, já que o *framebuffer* que permite a exibição no monitor está implementado nesta memória DDR.

O *framebuffer* consiste em um mapa dos pixels e seus valores em memória. A partir deste *framebuffer* são feitas atualizações periódicas de todos os pixels no monitor. Sendo assim, enquanto não houver modificações nos valores dos pixels, continuará a ser exibida na tela sempre a mesma imagem, até que novos valores sejam salvos no *framebuffer*. Assim fica clara a necessidade de um *framebuffer* de tamanho proporcional à resolução desejada. Por exemplo, caso a resolução seja de 640x480, com 24 bits por pixels, temos que o *framebuffer* terá um tamanho de aproximadamente 12,6 Mbits.

No caso da plataforma em questão, existe disponível no FPGA apenas 2,5 Mbits em *block RAM*, o que impede a implementação em hardware de um *framebuffer* com resolução maior do que 352x288. A aplicação em software, rodando em PowerPC, permite a criação de um *framebuffer* que comporte uma imagem em resolução HDTV (1920x1080 pixels), utilizando para isso a memória DDR externa conectada à placa. Entretanto, os acessos a esta memória inserem uma latência muito grande fazendo com que a taxa de atualização não atinja 2 quadros por segundo. Estes dados foram obtidos perante experimentos. Obviamente, deverá ser estudada uma nova alternativa para o

processo de exibição, mas, por enquanto, será utilizada a versão do *framebuffer* em memória externa.

A memória externa também constitui um gargalo para o fornecimento de dados de entrada para o módulo que estiver rodando na placa. Apesar de ela ser uma memória rápida do tipo DDR (permite duas operações de leitura ou escrita por ciclo), a grande demanda de dados de entrada e a disputa pelo acesso para ler entradas, escrever saídas e atualizar o monitor periodicamente, tornam-na um sério gargalo. Também para o fornecimento de dados, entendemos que há a necessidade de se encontrar um meio alternativo, algo como uma carga paralela via pinos do FPGA, por exemplo.

5.1.4 Módulo de Predição Intra-Quadro

A partir dos conceitos matemáticos apresentados na descrição do padrão H.264, foi desenvolvida uma arquitetura de *hardware* para a execução das operações referentes à predição das amostras de luminância em blocos 4x4. Após a conclusão da implementação da predição 16x16 e da predição das amostras de crominância, estas unidades foram integradas em um Gerador de Predição – GP. Este é o elemento principal do módulo intra, pois é responsável efetivamente pelo cálculo da predição. No entanto, ele depende de outras ações imprescindíveis que são o armazenamento e disponilização das amostras já processadas e o tratamento dos elementos sintáticos que fornecem as informações básicas ao processamento como, por exemplo, o modo de predição escolhido. A Figura 5.6 apresenta uma visão geral do módulo de predição intra, dividido em Gerador de Predições (GP), Buffer de Amostras Vizinhas (BAV) e o Decodificador de Elementos Sintáticos (DES), e suas conexões com os módulos vizinhos (Staehler, 2006b).

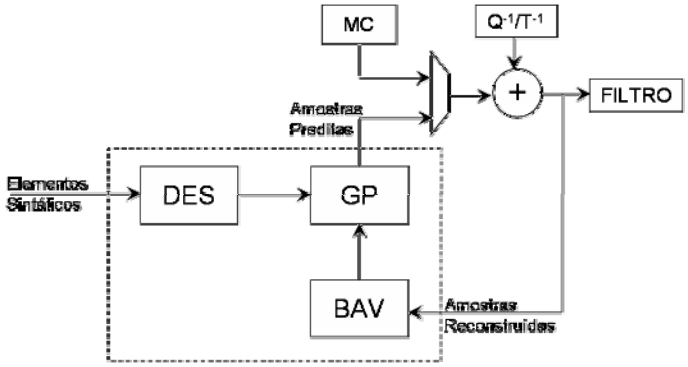

Figura 5.6: Visão Geral do Módulo de Predição Intraquadro

5.1.5 Integração com Módulos Vizinhos

A partir da conclusão do módulo intra, torna-se intuitivo o processo de integração através da incorporação dos módulos vizinhos, uma vez que também estejam prontos. O projeto do filtro de redução de efeito de bloco é apresentado em (Agostini, 2006) e a descrição da arquitetura para a transformada e quantização inversas, em (Agostini, 2006b). A partir daqui, há apenas a preocupação com a integração destes módulos uma vez que eles tenham sido projetados por outros membros da equipe.

Seguindo por esta linha, são adicionados ao módulo de predição intra-quadro o multiplexador e o somador, como pode ser visto na Figura 5.6. Deste modo, os módulos da transformada e quantização inversas (Q^{-1}/T^{-1}) e do compensador de movimento (MC)

poderão ser conectados diretamente nestas interfaces, como apresentado na Figura 5.7.

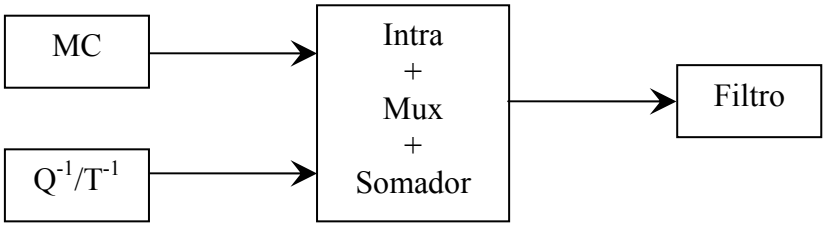

Figura 5.7: Integração Intra-Mux-Somador

Em seguida, foi adicionado ao sistema o filtro. Este possui um buffer de entrada e apenas inicia o processo de filtragem quando os dados necessários estão disponíveis, o que facilitou a sua integração com o somador por dispensar a necessidade de sincronismo. Existem dois sinais saindo do módulo intra, 'soma' e 'soma_ok', que se ligam diretamente aos sinais de entrada do filtro, 'entrada' e 'we_entrada', respectivamente. O próprio sinal que indica que a soma está pronta é o sinal que habilita o processo de escrita do dado no buffer de entrada do filtro. À medida que os dados vão sendo gerados, o filtro realiza a sua parte e repassa a informação para a exibição.

O passo seguinte foi integrar ao sistema o módulo Q^{-1}/T^{-1}. Este módulo emite dois sinais, 'residuo' e 'ok_ti', ligado ao bloco integrado como mostrado na Figura 5.8. O inconveniente apresentado foi o fato deste módulo emitir dados a uma taxa de uma amostra por ciclo enquanto no intra a vazão é de quatro amostras por ciclo. Para solucionar este problema, o clock do intra foi dividido por quatro, ficando com uma vazão de 4 amostras a cada 4 ciclos, e foi implementado um conjunto de quatro registradores de deslocamento para aguardar a emissão de quatro amostras por parte da transformada inversa e fornecê-las de 4 em 4 ao bloco integrado. Esta solução apresentou uma redução drástica da freqüência de operação do sistema ao dividir o clock do módulo intra por quatro.

Sendo assim, para a conclusão de uma primeira versão simplificada do decodificador restará ainda um módulo que faça a decodificação de entropia e o reconhecimento dos elementos sintáticos. Esta primeira versão decodificaria apenas quadros I (que utiliza apenas predição intra), já que o módulo de compensação de movimento ainda não foi integrado, e como codificação de entropia poderia utilizar apenas CAVLC, deixando o CABAC (muito mais complexo) como realização posterior.

Este módulo integrado contendo predição intra, transformada inversa e filtro, além de um somador e de um multiplexador, foi transformado por meio da ferramenta de desenvolvimento EDK, da Xilinx, em um componente CoreConnect. Na prototipagem este fato já trouxe vantagens porque possibilitou a comunicação com o PowerPC, permitindo que este rodasse uma aplicação de depuração. Este IP CoreConnect é apresentado em detalhes em (Staehler, 2006c).

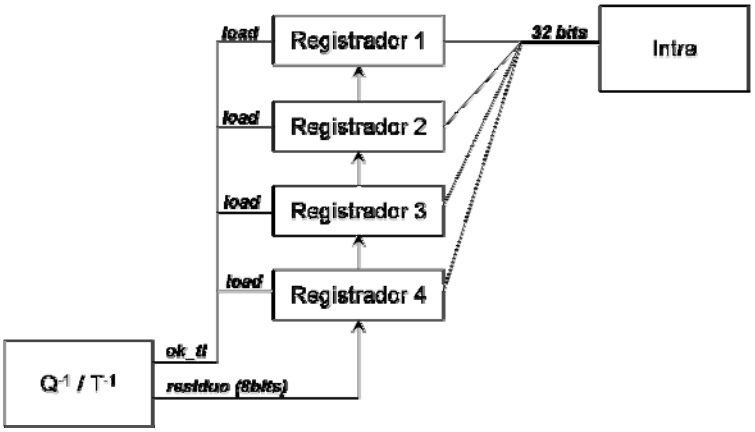

Figura 5.8: Integração do Módulo Q-1/T-1

A Tabela 5.1 apresenta os resultados da síntese da arquitetura oriunda da integração, já como um IP compatível com o modelo CoreConnect. O *chip* alvo é o Xilinx VP30 e a freqüência de operação atingida foi de 34,5 MHz, o que permite um processamento de 11 quadros HDTV por segundo e de 74 quadros SDTV por segundo.

Tabela 5.1: Resultados da Síntese

Número de Slices	9800 of 13696 (71%)
Número de Flip-Flops	10057 of 27392 (36%)
Número de LUTs	11807 of 27392 (43 %)
Freqüência de Operação	34.515 MHz
Taxa de dados	1 amostra/ciclo

5.2 Análise do Projeto do Decodificador H.264

Projetos complexos demandam uma capacidade de organização muito grande, devido à quantidade de sub-módulos e projetistas envolvidos, para que o sistema seja construído com sucesso e prazos sejam respeitados. As maiores empresas de produtos eletrônicos do mundo investem cada vez mais em metodologias de projeto que possibilitem a conclusão de projetos sempre mais complexos em prazos de mercado sempre mais curtos. E após a experiência inicial com este sistema complexo em particular, pudemos comprovar que a única forma de ter controle sobre o desenrolar do projeto, cumprindo etapas, respeitando um cronograma e culminando em um sistema que satisfaça todos os requisitos, é através de uma metodologia robusta de projeto baseado em reuso.

Na experiência descrita acima, ficam claros o atraso e dificuldade gerados pela simples ausência de uma especificação adequada nas fases iniciais do projeto. Por esta razão, pode-se declarar como aspectos fundamentais de um projeto complexo os seguintes tópicos: especificação do sistema e geração de modelos, subdivisão do sistema e especificação de suas interfaces, desenvolvimento de componentes e integração dos componentes.

5.2.1 Dificuldades no Processo de Integração

Por uma falta de experiência do grupo em grandes projetos, acabou por ser adotada uma abordagem *bottom-up* na fase inicial do projeto. Desta maneira, pretendia-se desenvolver cada módulo componente do decodificador e, futuramente, integrá-los. Isto contraria claramente a metodologia defendida nesta dissertação, segundo a qual é imprescindível como primeira etapa a especificação do sistema global, seu particionamento e definição de interfaces. Sendo assim, independentemente do resto do sistema e de como ele seria inserido no mesmo, partiu-se para o desenvolvimento do módulo de predição intra-quadro para o decodificador.

O módulo de predição intra, no decodificador H.264, tem por finalidade gerar as predições das 256 amostras de um macrobloco. Uma possível solução seria desenvolver um sistema que recebesse as entradas (vizinhos relevantes mais o modo escolhido no codificador) e, depois de um tempo de processamento, entregasse todas as predições referentes ao macrobloco. Se cada módulo utilizasse como unidade de processamento um macrobloco inteiro, o período se define pelo maior tempo entre os módulos do sistema. Durante este período todos os demais módulos teriam terminado o processamento do seu macrobloco e ficariam esperando até o próximo início de período, gerando um grande desperdício por inatividade. Reduzindo-se a unidade de processamento, este desperdício tende a ser menor porque as unidades de período são reduzidas. Deste modo, optou-se por uma vazão menor, isto é, no lugar de emitir um macrobloco de cada vez emite-se uma amostra de cada vez.

Em um primeiro momento, desenvolveu-se uma arquitetura de *hardware* que realizaria a predição de todas as amostras de um macrobloco em apenas um ciclo de relógio. Obviamente, o período do relógio obtido foi muito grande. Em seguida, sendo coerente com o raciocínio descrito acima, foi criada uma arquitetura que tivesse uma vazão de apenas uma amostra por ciclo. Imediatamente constatou-se a necessidade de *hardware* adicional para o processamento do modo DC que é o modo mais complexo e, por exceção, fornece o mesmo resultado de predição para todas as amostras do macrobloco ou bloco em questão. Utilizando-se deste *hardware* adicional decidiu-se por uma vazão de 4 amostras por ciclo no lugar de apenas uma, já que pôde ser feito um reaproveitamento da replicação e paralelização de uma unidade de processamento de forma a realizar também o modo DC em apenas um ciclo.

Finalmente, o módulo intra foi concluído (projetado e validado) atendendo aos requisitos de HDTV, e operando com uma taxa de 4 amostras por ciclo. Partindo-se dele, deve-se integrá-lo primeiramente com os módulos mais próximos e mais simples, nesta ordem: somador, filtro e transformada inversa.

O somador não ofereceu maiores problemas durante a sua integração por se tratar de um circuito puramente combinacional. Aumentou um pouco o período do clock, mas como o módulo intra operava a uma freqüência bastante alta não impediu que ainda se pudesse processar vídeos HDTV.

O filtro, por precisar de um buffer de entrada e por trabalhar com sinais de sincronismo como “saída_ok” e “escreve_entrada”, teve a sua integração bastante facilitada, bastando preencher os valores no buffer e esperar que ele detecte a possibilidade de iniciar o procedimento de filtragem.

A primeira dificuldade na fase de integração apareceu ao se constatar que a transformada inversa trabalha com vazão de uma amostra por ciclo, lembrando que no

módulo intra são 4. O resultado do intra (a predição) com o resultado da transformada (o resíduo) são somados pelo somador, aonde devem chegar devidamente sincronizados. Conseqüentemente, o módulo de predição intra deve ter sua freqüência de operação dividida por quatro para esperar o resultado da transformada na hora de somar, deixando de usufruir de seu paralelismo. Além disso, foi necessário o desenvolvimento de uma lógica de cola para a união destes módulos, um conjunto de 4 registradores para serializar a saída quádrupla do intra. Esta lógica adicional acaba por onerar o sistema com mais atraso e área. Uma decisão simples, como a escolha da taxa de amostras por ciclo a ser adotada por todos os módulos, deveria ter sido tomada ainda na etapa de especificação do sistema, otimizando a arquitetura final e facilitando a integração e o cumprimento dos prazos.

Outra dificuldade encontrada, após a conclusão da integração da transformada com o somador e o módulo intra, foi a criação de arquivos de teste para a verificação do sistema. A verificação do módulo intra foi realizada através da extração de arquivos do código de referência, antes e depois do módulo intra. Membros da equipe responsável pelo módulo intra estudaram o código de referência, isolaram a parte referente a ele e inseriram funções para a extração dos dados de entrada e de saída do mesmo (Figueiró, 2005).

Pelo fato do código de referência estar bastante intrincado, além de ser complexo e extenso, o grupo responsável pela transformada inversa encontrou dificuldades em isolar este módulo dentro do código de referência, para que servisse de modelo executável para o módulo e proporcionasse os arquivos necessários para o teste. Desta maneira, na etapa de integração foi preciso abrir um parêntesis para estudar e compreender o módulo da transformada e desenvolver seu modelo executável de modo a prover meios para a verificação deste *hardware*. Como conseqüência, houve atraso na integração devido ao fato do projetista integrador ter de se envolver na compreensão interna de um dos componentes do sistema, para que fossem gerados arquivos de teste a partir do modelo executável desenvolvido em Matlab.

O resultado final desta primeira etapa de integração é um sistema composto de predição intra-quadro, somador, transformada e quantização inversas e filtro de redução de efeito de bloco. Este sistema foi completamente integrado, validado e está de acordo com a norma H.264. Entretanto, mesmo que estes módulos atendessem individualmente com bastante folga aos requisitos necessários à decodificação de HDTV em tempo real, após a correção da vazão do módulo intra de 4 para 1, a freqüência de operação do sistema integrado caiu drasticamente, demandando alterações.

5.2.2 Especificação

Como aspecto mais importante de um projeto complexo, temos a etapa de especificação. Deve-se dedicar todo o empenho e tempo necessários à obtenção de uma especificação correta e adequada, porque é ela que irá conduzir o projeto inteiro. Erros na especificação, quanto mais avançado for detectado, poderá ser desastroso em termos de planejamento porque implicará em um recuo às fases iniciais de projeto.

A especificação também pode ser acompanhada de um modelo executável, desenvolvido em código C ou Matlab. Quando se trata de algoritmos matemáticos ou de processamento intenso, é fortemente recomendado o seu uso. Assim, pode-se verificar como o sistema funciona, de que forma que ele poderá ser subdividido em módulos de menor e similar complexidade, quais devem ser exatamente seus sinais de interface com

o mundo externo. De posse do modelo executável, as equipes de *software* e de verificação e teste também podem iniciar os seus desenvolvimentos, mesmo antes do *hardware* estar concluído, e a partir destes fluxos de projeto paralelos obtém-se uma aceleração no processo.

5.2.3 Subdivisão do Sistema e Desenvolvimento de Componentes

A partir da especificação do sistema, é importante encontrar qual o melhor particionamento do sistema em blocos de menor e similar complexidade. A complexidade ideal é aquela onde um bloco possa ser projetado por apenas uma pessoa, e onde há menos comunicação, simplificando a sua interface. Nesta etapa, também é decidido se determinado componente será efetivamente desenvolvido pela equipe ou adquirido de terceiros. Na realidade, a própria subdivisão do sistema deve levar em conta a disponibilidade de venda e custos relativos a certos componentes.

Para o caso de desenvolver o próprio componente, existe a alternativa de torná-lo um IP reutilizável. Para tanto, deve-se questionar se ele será de relevância no futuro, de maneira que valha a pena o investimento a ser feito para torná-lo reutilizável. Há um custo envolvido no processo de tornar um IP reutilizável, oferecendo diversas configurações para ele possa ser amplamente inserido em outros projetos, e ainda na obediência a padrões internacionais de interface. É sabido que um componente IP reutilizável tem um custo que vai de 3 a 7 vezes o custo do mesmo módulo projetado para uso único. Desde modo, quanto mais flexível seja o componente, e quanto mais ele for utilizado, este investimento tende a valer a pena. Um IP bem eficiente e bem documentado certamente será utilizado muito além das 7 vezes que mereçam o seu custo de projeto, e ainda irá acelerar futuros projetos ou ser comercializado com outras companhias desenvolvedoras. Pode-se ainda optar por uma simples interface ad-hoc, quando da constatação de que não haverá interesse futuro no componente.

Na aquisição de IPs de terceiros, de acordo com as restrições de desempenho e área, deve-se analisar a necessidade de um hard IP. Quando as restrições são extremas, é imprescindível a obtenção de um componente otimizado para a tecnologia de fabricação em questão, ou seja, o hard IP. Contudo, tal hard IP pode não estar disponível, ou as restrições de projeto podem não ser tão extremas; nestes casos, utilizam-se soft IPs. Soft IP, como descrito em capítulos anteriores, são apresentados em nível RTL e em código HDL e, portanto, necessitam de tempo de desenvolvimento para chegar até o nível físico. Dependendo das restrições de projeto pode-se, inclusive, optar por uma síntese automática por parte de ferramentas de projeto de circuitos integrados. Em seguida, devem ser inseridos mecanismos de verificação e teste no componente.

Componentes IP também são avaliados de acordo com o seu nível de testabilidade. Um componente bem documentado que apresente modelo executável e completo ambiente de verificação, tanto para teste *stand-alone* quanto para teste integrado ao sistema, facilitará em muito o processo de integração. A adequação a algum padrão de interface de comunicação também é essencial para a conclusão da integração satisfatoriamente. A escolha do padrão é feita ainda nas fases iniciais, e a subdivisão do sistema e escolha dos componentes deve ser coerente com o mesmo.

5.2.4 Comunicação Entre Componentes

De acordo com o projeto em questão, deve-se decidir pela forma de comunicação entre os diversos componentes do sistema. No capítulo 2, foi apresentada a forma

genérica de projeto SoC, onde é possível estabelecer a comunicação entre quaisquer componentes através dos barramentos e ainda existe a possibilidade de inserção de componentes de *software* a serem executados pelo processador de propósito geral. Existem, ainda, outras maneiras de se estabelecer a comunicação como, por exemplo, via *Network-on-chip* (NoC) ou via interface ad-hoc.

Independentemente da comunicação estipulada, será adotado um padrão de interface a ser obedecido pelos diversos componentes, visando à integração do sistema.

5.2.5 Recomendações Para a Conclusão Bem-Sucedida do Decodificador H.264

Na tentativa de aproveitar todo o esforço que já foi despendido na implementação do decodificador, nesta seção serão apresentadas recomendações baseadas na experiência e estudos realizados nos últimos meses que possam permitir a conclusão do projeto.

Inicialmente, sobre os módulos já integrados, constatou-se uma freqüência de operação muito baixa devido à diferença de vazão escolhida para o módulo intra. Este módulo deve ser alterado e reinserido no sistema para evitar este gargalo. O bloco integrado deve passar a utilizar como sinal de clock um cristal da placa de prototipagem no lugar do clock administrado via *software*, pois só assim obteremos a exibição do vídeo decodificado na taxa devida.

Para que seja possível decodificar vídeos, mesmo que inicialmente mais simples, será necessário receber o *bitstream*, e a partir de seus símbolos reconhecer e decodificar os elementos sintáticos. Os módulos do decodificador responsáveis por estas tarefas são o decodificador de entropia, que resgata os elementos sintáticos a partir dos símbolos do *bitstream*, em parceria com o analisador sintático (ou *parser*) que reconhece a ordem em que os elementos aparecem. A partir das informações colhidas, o próprio *parser* é capaz de encaminhar os dados para o devido processamento em cada um dos demais módulos do decodificador, por exemplo, os coeficientes são encaminhados para a transformada inversa, os vetores de movimento para o compensador de movimento, e assim por diante. Este tipo de tratamento de um *bitstream* por parte do decodificador de entropia e do analisador sintático pode-se ser encarado como uma tarefa descrita e desempenhada por um módulo de *software*. Como pôde ser observado na nossa plataforma de prototipagem, existem disponíveis no FPGA dois microprocessadores PowerPC para a execução de código C, e que são plenamente capazes de rodar estes módulos descritos como rotinas de *software*.

Ao se configurar o codificador do código de referência para a geração de arquivos de teste para a implementação do decodificador, pode-se configurá-lo para que utilize apenas os recursos para os quais o decodificador esteja preparado para tratar. Por exemplo, como nem o módulo de compensação de movimento nem o módulo de decodificação de entropia CABAC estão integrados, deve-se configurar o codificador para que não faça predições inter-quadro e que utilize apenas o CAVLC para a codificação de entropia. CAVLC é bastante mais simples do que CABAC, por isso deve-se inicialmente se dedicar à conclusão do primeiro para depois acrescentar o segundo.

Nesta situação, para a conclusão de uma primeira versão do decodificador, mais simplificada, recomenda-se a implementação do CAVLC e do *parser* em código C, utilizando-se o processador PowerPC presente no *chip*, além da utilização do componente CoreConnect descrito ao longo desta dissertação, composto por intra, filtro, e transformada inversa. Após o término desta versão mais simples do decodificador, os

esforços devem concentrados na integração do módulo de compensação de movimento assim como do CABAC, culminando na total conclusão do projeto e a obtenção de um decodificador completamente de acordo com o perfil Main do padrão H.264.

5.3 Conclusões

Para o caso da comunicação entre o módulo em VHDL e PowerPC, basta escrever o código C do PowerPC, e do lado do bloco de *hardware*, basta utilizar os sinais previamente estipulados pelo padrão OCP. O protocolo de comunicação já está bem definido e a integração é um processo natural e fácil.

Já no caso da integração entre os módulos Intra/Somador/Mux e Q^{-1}/T^{-1}, onde não houve a adoção de nenhum modelo de comunicação e ainda faltou uma especificação adequada da interface entre eles, surgiu a necessidade de desenvolvimento de uma lógica de cola, ou um envelope para o módulo Q^{-1}/T^{-1}, que permitisse a conexão dos seus sinais nas entradas do Intra/Somador/Mux, assim como o sincronismo entre eles. Esta lógica de cola adicionou custo ao circuito, tanto em área quanto em atraso.

É exatamente por isso que a decisão de tornar o *core* um IP deve ser tomada o mais cedo possível. Os próprios sinais do padrão adotado devem ser utilizados desde o início, sem a posterior adequação por meio da geração de um *wrapper*, o que acarreta obrigatoriamente em uma piora na qualidade do IP, assim como demais problemas relativos a sincronismo ou adequação do protocolo de comunicação ad-hoc.

6 CONCLUSÕES

O consumidor se mantém à espera de novidades cada vez mais surpreendentes no campo da eletrônica, e os sistemas projetados estão cada dia mais complexos. Os processos de fabricação evoluem rapidamente, colocando mais transistores à disposição do engenheiro. A única maneira de possibilitar a realização destes projetos no tempo desejado é mudando a forma de projetar. É necessário se atrelar a um nível de abstração mais alto, utilizando como tijolos desta construção não mais transistores ou portas lógicas, mas verdadeiros módulos funcionais. No entanto, antes de mais nada é preciso constituir uma biblioteca de módulos componentes do mesmo modo que hoje ferramentas de CAD apresentam bibliotecas de células, de acordo com a tecnologia de fabricação escolhida podendo fazer o mapeamento do circuito integrado a partir de uma descrição em código ou esquemático.

Esta mudança de paradigma exige novos hábitos dos projetistas como o aprendizado de padrões internacionais de comunicação e o conhecimento de padrões de empacotamento de módulos componentes. E ainda técnicas de projeto cada vez mais voltadas a facilitar a compreensão e teste do projeto. Verificação e teste destes sistemas complexos podem se tornar etapas bastante difíceis porque envolvem a exploração de todas as possibilidades de sinais elétricos que possam circular pelo circuito. Quanto maior o sistema muito mais difícil será a realização da verificação. É por isso que se torna imprescindível a realização de testes individuais durante o desenvolvimento dos componentes IPs, possibilitando uma verificação de mais alto nível sobre o sistema, isto é, a realização, basicamente, de uma verificação de sinais de sincronismo e de sinais de comunicação.

A complexidade atual dos projetos permite ainda que se desenvolvam soluções do tipo *hardware-software co-design*, isto é, um processador genérico rodando *software* e se comunicando com módulos de *hardware* dedicado. Esta abordagem no projeto do decodificador H.264 também trouxe outras vantagens como permitir a verificação do *hardware* com o auxílio do PowerPC, rodando um programa de depuração e monitoramento. Através de uma interface serial ligando a placa de prototipagem e o computador do projetista é possível se comunicar com o PowerPC e resgatar as informações da execução do algoritmo. Posteriormente, pretende-se a substituição do programa de depuração pelos respectivos módulos de *software* obtendo a conclusão do sistema. Isto demonstra a importância de uma solução *hardware/software co-design*, que não apenas permite a implementação do sistema como também auxilia o depuramento do projeto de *hardware*, bastante difícil devido à quantidade crescente de módulos componentes e de sinais transitantes.

Foi possível constatar durante este estudo que a fase de especificação continua sendo a mais importante. Esta etapa é constituída de uma compreensão do problema e de uma formulação de possíveis soluções baseadas nos requisitos e nos recursos disponíveis. Também se compõe de uma busca por componentes que possam suprir determinadas funcionalidades do sistema e por uma escolha do padrão de comunicação que mais se ajuste ao tipo de algoritmo.

Esta dissertação apresentou uma perspectiva de como se deve levar um projeto nos dias atuais, caracterizado por uma crescente complexidade e por prazos de entrega cada vez mais curtos. Foi mostrada uma experiência com o decodificador de vídeo H.264, analisada sob o ponto de vista do paradigma SoC. Finalmente, foi passada uma recomendação de como concluir o projeto realizando algumas alterações e gerando um IP de descompressão de vídeo que possa ser embarcado em um sistema de televisão digital.

A meta fundamental de se mudar a visão do projetista de sistemas digitais consiste na obtenção de ferramentas EDA capazes de realizar a elaboração do sistema, composto tanto de *hardware* quanto de *software*, a partir de uma descrição alto nível do mesmo. Conceito análogo ao que temos hoje em questão de mapeamento tecnológico, onde a ferramenta é capaz de gerar o conjunto de transistores em uma determinada tecnologia de fabricação a partir de uma descrição de porta lógica. Todavia, é fundamental o desenvolvimento de componentes, seja de *software* ou de *hardware*, para compor a biblioteca que vai permitir que a ferramenta implemente o sistema desejado. A forma de comunicação também deve ser previamente estipulada com a adoção de algum dos padrões internacionais de comunicação através de barramento ou ponto-a-ponto. Quanto maior a aceitação de determinado padrão maior coesão haverá entre os projetos realizados em nível mundial, e conseqüentemente maior será o intercâmbio e o reuso de componentes. Deste modo, a biblioteca tenderá a crescer rapidamente e será capaz de oferecer alternativas melhores para um maior número de sistemas.

Durante o desenvolvimento do trabalho aqui descrito, foram submetidos e aprovados três artigos de relevância em congressos científicos. Primeiro, o artigo chamado "*Real-Time 4x4 Intraframe Prediction Architecture for an H.264 Decoder*", apresentado no VI ITS – IEEE International Telecommunications Symposium, em Fortaleza – CE, descrevendo a arquitetura proposta para a decodificação intraquadro de blocos 4x4. Segundo, o artigo "*Architecture of an HDTV Intraframe Predictor for an H.264 Decoder*", apresentado no XIV IFIP International Conference on Very Large Scale Integration, em Nice, França, que descreve o projeto do módulo de predição intraquadro completo para o decodificador H.264, composto pelas partes Luma 4x4, Luma 16x16, Croma, Decodificador de Elementos Sintáticos e Buffer de Amostras Vizinhas. Finalmente, após a integração dos módulos do decodificador Predição Intraquadro, Somador, Filtro e Transformada Inversa, esta versão foi empacotada como um IP CoreConnect. Desta maneira, como apresentado na seção 5.4, será possível a realização de um SoC composto por um bloco de *hardware* e de *software* rodando no PowerPC. Este último artigo se chama "*IP Core for an H.264 Decoder SoC*", apresentado em Grenoble, França, em dezembro, no XV IP-Based SoC Design 2006.

Através da experiência apresentada esperava-se adquirir domínio sobre as metodologias de projeto baseado em reuso de componentes e também conhecer o estado-da-arte no meio industrial. Devido às dificuldades encontradas durante o início do projeto do decodificador H.264 e durante a integração dos primeiros módulos, tornou-se imperativo buscar uma nova abordagem que permitisse concluir o projeto em

um prazo menor e de uma forma mais otimizada, gerando uma solução clara, de fácil entendimento, modularizada e que fosse também otimizada em termos de área e consumo de energia. Mais do que isso, esperava-se provocar uma mudança na maneira de pensar de toda a equipe, trazendo através deste documento a prova concreta de que os projetos digitais complexos daqui para frente só obterão êxito se forem devidamente planejados segundo o paradigma *System-on-Chip*.

REFERÊNCIAS

AGOSTINI, L. et al. High throughput multitransform and multiparallelism IP directed to the H.264/AVC video compression standard. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, 2006, Ilha de Kos, Grécia. **Proceedings…** Los Alamitos: IEEE, 2006. p. 5419-5422.

AGOSTINI, L. et al. FPGA Design of a H.264/AVC Main Profile Decoder for HDTV. In: INTERNATIONAL CONFERENCE ON FIELD PROGRAMMABLE LOGIC AND APPLICATIONS, FPL, 16., 2006, Madrid, Espanha. **Proceedings…** [S.l.:s.n.], 2006.

ARM. **AMBA Home Page**. Disponível em: <http://www.arm.com/products/solutions/AMBAHomePage.html>. Acesso em: nov. 2006.

BHASKARAN, V.; KONSTANTINIDES, K. **Image and Video Compression Standards:** Algorithms and Architectures. 2nd ed. Boston: Kluwer Academic Publishers, 1997.

BRICAUD, P.J. IP Reuse Creation For System-on-a-Chip Design. In: IEEE CUSTOM INTEGRATED SYSTEMS, 1999. **Proceedings…** Los Alamitos: IEEE, 2006. p. 395-401.

CHU, C. et al. Hierarchical Global Motion Estimation/Compensation in Low Bitrate Video Coding. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 1997, Hong Kong, China. **Proceedings…** Los Alamitos: IEEE, 1997. p. 1149-1152.

DESIGN and Reuse. Disponível em: <http://www.design-reuse.com>. Acesso em: nov. 2006.

DIGILENT INC. **Digilent XUPV2P Hardware Reference Manual**. Disponível em: <www.digilentinc.com/xupv2p>. Acesso em: set. 2005.

FIGUEIRO, T. H.264 Implementation Test Using the Reference Software. In: BRAZILIAN SYMPOSIUM ON COMPUTER GRAPHICS AND IMAGE PROCESSING, SIBGRAPI, 18., 2005, Natal. **Conference proceedings**. Los Alamitos: IEEE, 2005.

GAJSKI, D.D. et al. Essential Issues For IP Reuse. In: ASIA AND SOUTH PACIFIC DESIGN AUTOMATION CONFERENCE, 2000. **Proceedings…** Los Alamitos: IEEE, 2000. p. 37-42.

GNU LGPL: Gnu Lesser General Public License. Disponível em: <http://www.gnu.org/licenses/lgpl.html>. Acesso em: dez. 2006.

HUNT, M., ROWSON, J.A. Blocking in a System-on-a-Chip. **IEEE Spectrum**, New York, v. 33, n. 11, p. 35-41, Nov. 1996.

IBM. **CoreConnect Bus Architecture**. Disponível em: <http://www.ibm.com/chips/products/coreconnect/>. Acesso em: nov. 2006.

INICORE. **Inicore System-on-Chip from Spec to Silicon**. Disponível em: <http://www.inicore.com>. Acesso em: nov. 2006.

INTERNATIONAL Organization for Standardization: ISO Home Page. Disponível em: <http://www.iso.org>. Acesso em: ago. 2005.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **ISO/IEC 11172 - MPEG-1 (11/1993)**: coding of moving pictures and associated audio for digital storage media up to about 1.5Mbit/s – part 2: video. [S.l.], 1993.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **ISO/IEC 14496-2 - MPEG-4 Part 2 (01/1999)**: coding of audio visual objects – part 2: visual. [S.l.], 1999.

INTERNATIONAL Telecommunication Union: ITU-T Home. Disponível em: <www.itu.int/ITU-T/>. Acesso em: ago. 2006.

INTERNATIONAL TELECOMMUNICATION UNION. **ITU-T Recommendation H.261 v1 (11/90):** video codec for audiovisual services at px64 kbit/s. [S.l.], 1990.

INTERNATIONAL TELECOMMUNICATION UNION. **ITU-T Recommendation H.262 (11/94):** generic coding of moving pictures and associated audio information – part 2: video. [S.l.], 1994.

INTERNATIONAL TELECOMMUNICATION UNION. **ITU-T Recommendation H.263 v3 (11/00):** video coding for low bit rate communication. [S.l.], 2000.

INTERNATIONAL TELECOMMUNICATION UNION. **ITU-T Recommendation H.264 (03/05): Advanced Video Coding for Generic Audiovisual Services**. [S.l.], 2005.

INTERNATIONAL TELECOMMUNICATION UNION. **H.264 reference code**. Disponível em: <http://iphome.hhi.de/suehring/tml/>. Acesso: dez. 2006.

ITRS – International Technology Roadmap for Semiconductors. 2001 Edition. Disponível em: <http://www.itrs.net/Links/2001ITRS/PIDS.pdf>. Acesso em: nov. 2006.

JAIN, J.; JAIN, A. Displacement Measurement and Its Application in Interframe Image Coding. **IEEE Transactions on Communications**. [S.l.], v. 29, n. 12, p. 1799-1808, Dec. 1981.

KAMACI, N.; ALTUNBASAK, Y. Performance Comparison of the Emerging H.264 Video Coding Standard with the Existing Standards. In: IEEE INTERNATIONAL CONFERENCE ON MULTIMEDIA AND EXPO, 2003, Baltimore, EUA. **Proceedings…** Los Alamitos: IEEE, 2003. p. 1345-1348.

KANNANGARA C.; RICHARDSON I. Computational Control of an H.264 Encoder through Lagrangian Cost Function Estimation. In: INTERNATIONAL WORKSHOP ON VERY LOW BITRATE VIDEO, 2005, Sardenha, Itália. **Proceedings…** [S.l.:s.n.], 2005.

KARCZEWICZ, M.; KURCEREN, R. The SP- and SI-frames design for H.264-AVC. **IEEE Transactions on Circuits and Systems for Video Technology,** [S.l.], v. 13, n. 7, p. 637-644, July 2003.

KEATING, M.; BRICAUD, P. **RMM - Reuse Methodology Manual for System-on-a-Chip Designs**. 3rd ed. Norwell: USA: Kluwer Academic Publishers, 2002.

KROUPIS, N. et al. A Modified Spiral Search Motion Estimation Algorithm and its Embedded System Implementation. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, Kobe, Grécia, 2005. **Proceedings…** Los Alamitos: IEEE, 2005. p. 3347-3350.

KUHN, P. **Algorithms, Complexity Analysis and VLSI Architectures for MPEG-4 Motion Estimation**. Boston: Kluwer Academic Publishers, 1999.

KWON, S. et al. Overview of H.264 / MPEG-4 Part 10. In: INTERNATIONAL CONFERENCE ON MULTIMEDIA, INTERNET AND VIDEO TECHNIQUES, WSEAS, 5., Corfu Island, Grécia, 2005. **Proceedings…** [S.l.:s.n.], 2005.

LI, R. et al. A New Three-Step Search Algorithm for Block Motion Estimation. **IEEE Transactions on Circuits and Systems for Video Technology,** [S.l.], v. 4, n. 4, p. 438-442, Aug. 1994.

MALVAR, H. et al. Low-Complexity Transform and Quantization in H.264/AVC. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], v. 13, n. 7, p. 598-603, July 2003.

MARPE, D. et al. Context-Based Adaptive Binary Arithmetic Coding in the H.264/AVC Video Compression Standard. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], v. 13, n. 7, p. 620-636, July 2003.

MATHWORKS Matlab. Disponível em: <http://www.mathworks.com/products/matlab/>. Acesso em: nov. 2006.

MENTOR GRAPHICS. **ModelSim Documentation**. Disponível em: <http://www.model.com>. Acesso em: set. 2005.

OCP-IP: Open Core Protocol International Partnership Home Page. Disponível em: <www.ocpip.org>. Acesso em: nov. 2006.

OCP-IP. **Open Core Protocol Specification**. Release 2.1. 2005. Disponível em: <www.ocpip.org>. Acesso em: nov. 2006.

OPENCORES: Home Page. Disponível em: <http://www.opencores.org/>. Acesso em: nov. 2006.

OPENCORES. **SoC Interconnection:** Wishbone. Disponível em: <http://www.opencores.org/projects.cgi/web/wishbone/wishbone>. Acesso em: dez. 2006.

PROJETO BrazilIP. Disponível em: <http://www.brazilip.org.br>. Acesso em: nov. 2006.

PURI, A. et al. Video Coding Using the H.264/MPEG-4 AVC Compression Standard. **Elsevier Signal Processing: Image Communication,** [S.l.], n. 19, p.793–849, 2004.

RICHARDSON, I. **H.264 and MPEG-4 Video Compression – Video Coding for Next-Generation Multimedia**. Chichester: John Wiley and Sons, 2003.

RICHARDSON, I. **Video Codec Design – Developing Image and Video Compression Systems**. Chichester: John Wiley and Sons, 2002.

SAHAFI, L. **Context-Based Complexity Reduction Applied to H.264 Video Compression**. 2005. 70 f. Thesis (Master in Applied Science) – School of Engineering Science, Simon Frase University, Canada.

SALOMON, D. **Data Compression: The Complete Reference**. 2nd ed. New York: Springer, 2000.

SARKAR, S.; SHINDE, S.; SUBASH, C.G. An Effective IP Reuse Methodology for Quality System-on-Chip Design. In: INTERNATIONAL SYMPOSIUM ON SYSTEM-ON-CHIP, 2005. **Proceedings…**, [S.l.:s.n.], 2005. p. 104-107.

SBTVD – Sistema Brasileiro de Televisão Digital. Disponível em: <http://sbtvd.cpqd.com.br/>. Acesso em: nov. 2006.

STAEHLER, W.T.; SUSIN, A.A. Real-Time 4x4 Intraframe Prediction Architecture for an H.264 Decoder. In: IEEE INTERNATIONAL TELECOMMUNICATIONS SYMPOSIUM, ITS, 6., 2006, Fortaleza. **Proceedings…** Los Alamitos: IEEE, 2006.

STAEHLER, W.T.; BERRIEL, E.A.; SUSIN, A.A.; BAMPI, S. Architecture of an HDTV Intraframe Predictor for an H.264 Decoder. In: INTERNATIONAL CONFERENCE ON VERY LARGE SCALE INTEGRATION, IFIP VLSI-SoC, 14., 2006, Nice, França. **Proceedings…** [S.l.:s.n.], 2006.

STAEHLER, W.T.; SUSIN, A.A. IP Core for an H.264 Decoder SoC. In: IP-BASED SOC DESIGN, IP-SOC, 15., 2006, Grenoble, França. **Proceedings…** [S.l.:s.n.], 2006.

SULLIVAN, G. et al. The H.264/AVC Advanced Video Coding Standard: Overview and Introduction to the Fidelity Range Extensions. In: CONFERENCE ON APPLICATIONS OF DIGITAL IMAGE PROCESSING, 27., 2004, Denver, EUA. **Proceedings…** [S.l.:s.n.], 2004.

SULLIVAN, G.; WIEGAND, T. Video Compression – From Concepts to the H.264/AVC Standard. **Proceedings of the IEEE**, [S.l.], v. 93, n. 1, p. 18-31, Jan. 2005.

SUNDANCE MULTIPROCESSOR TECHNOLOGY LTD. **SMT395 User Manual**. [S.l.], 2001. Disponível em: <www.sundance.com>. Acesso em: set. 2005.

SUNNA, P. AVC / H.264 – An Advanced Video Coding System for SD and HD Broadcasting. **European Broadcasting Union Technical Review**, [S.l.], n. 302, Apr. 2005. Disponível em: <http://www.ebu.ch/departments/technical/trev/trev_302-sunna.pdf>. Acesso em: set. 2005.

TOURAPIS, M. et al. Fast Motion Estimation Using Circular Zonal Search. In: VISUAL COMMUNICATIONS AND IMAGE PROCESSING, 1999, San Jose, EUA. **Proceedings…** [S.l.:s.n.], 1999. v. 2, p. 1496-1504.

VSIA: Virtual Socket Interface Alliance Home Page. Disponível em: <http://www.vsia.org>. Acesso em: nov. 2006.

VSIA. **VSI Alliance Architecture Document**. Version 1.0. 1997. Disponível em: <http://www.vsia.org/documents/vsiadocuments.htm>. Acesso em: nov. 2006.

VSS. **Consumer Support**: encoding guide. Disponível em: <http://www.vsofts.com/h264/h264_guide.html>. Acesso em: nov. 2005.

WIEGAND, T. et al. Rate-Constrained Coder Control and Comparison of Video Coding Standards. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], v. 13, n. 7, p. 688-703, July 2003.

WISHBONE. **Wishbone System-on-Chip Interconnection Architecture for Portable IP Cores**. Disponível em: <http://www.opencores.org/projects.cgi/web/wishbone>. Acesso em: nov. 2006.

XILINX INC. **Xilinx**: The Programmable Logic Company. Disponível em: <www.xilinx.com>. Acesso em: ago. 2006.

XILINX INC. **Virtex-II Pro and Virtex-II Pro X Platform FPGAs:** Complete Data Sheet.[S.l.], 2005. Disponível em: <www.xilinx.com>. Acesso em: set. 2005a.

XILINX INC. **Xilinx :** EDK Documentation and Supplemental Information. Disponível em: <http://www.xilinx.com/ise/embedded/edk_docs.htm>. Acesso em: dez. 2005c.

XILINX INC. **Xilinx University Program – Virtex-II Pro Development System – Hardware Reference Manual**. [S.l.], 2005. Disponível em: <www.digilentinc.com>. Acesso em: set. 2005b.

YI, X.; LING, N**.** Rapid Block-Matching Motion Estimation Using Modified Diamond Search Algorithm. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 2005, Kobe, Grécia. **Proceedings...** Los Alamitos: IEEE, 2005. p. 5489 – 5492.

APÊNDICE A PADRÃO DE COMPRESSÃO H.264

Este apêndice irá apresentar um resumo sobre o padrão H.264 (ITU-T, 2005b). Será apresentado um breve histórico do padrão, a terminologia utilizada, os perfis e os níveis do padrão. A seguir, será apresentado o núcleo de um codec H.264, com alguns detalhes sobre os seus principais blocos.

Histórico do Padrão

O padrão H.264 foi desenvolvido por um período de aproximadamente quatro anos. As raízes deste padrão estão no projeto H.26L da ITU-T, que foi iniciado pelo VCEG (*Video Coding Experts Group*) (ITU-T, 2005a), que construiu a chamada para propostas no início de 1998 e que criou o primeiro *draft* deste novo padrão em agosto de 1999. O objetivo do projeto H.26L era dobrar a eficiência de codificação atingida pelo padrão H.263 (ITU-T, 2000).

Antes do padrão H.264 surgir, alguns outros padrões já haviam sido criados e já estavam consolidados, servindo de base para o desenvolvimento do H.264. O primeiro padrão relevante para a construção do H.264 foi o H.261 da ITU-T (ITU-T, 1990). Este padrão lançou as bases do que é usado até hoje na maioria dos padrões de compressão de vídeo: DPCM com estimação de movimento na direção temporal, transformada discreta do cosseno aplicada no resíduo e quantização linear seguida de codificação por entropia. Após o padrão H.261, surgiu o padrão MPEG-1 da ISO (ISO, 1993) seguido do padrão MPEG-2 da ISO, que também foi padronizado pela ITU-T como H.262 (ITU-T,1994), que se tornou um padrão popular, muito usado até a atualidade em diversas aplicações. Apesar do grande sucesso do padrão MPEG-2, a evolução dos padrões de compressão de vídeo não parou. O padrão H.263 (ITU-T, 2000) foi lançado e incorporou alguns avanços obtidos pelos padrões MPEG-1 e MPEG-2, bem como técnicas novas que vinham sendo pesquisadas intensamente tanto pela indústria quanto pela academia.

Em 2001 o grupo MPEG (*Moving Pictures Experts Group*) da ISO (ISO, 2006) finalizou o desenvolvimento do seu mais recente padrão, conhecido como MPEG-4 Parte 2 (ISO, 1999). Então, ainda neste ano, o MPEG construiu uma nova chamada de propostas, similar à do H.26L da ITU-T, para melhorar ainda mais a eficiência de codificação atingida pelo MPEG-4. Então o VCEG, da ITU-T resolveu submeter seu *draft* em resposta à chamada de propostas do MPEG e propôs a união de esforços para completar o trabalho.

Analisando as respostas para sua chamada de propostas, o MPEG chegou a conclusões que afirmaram as escolhas de desenvolvimento realizadas pelo VCEG no H.26L:

- A estrutura de compensação de movimento com a transformada discreta do cosseno (DCT) era melhor do que as outras.
- Algumas ferramentas de codificação de vídeo que foram excluídas no passado (do MPEG-2, do H.263 ou do MPEG-4 Parte 2) por causa da sua complexidade computacional, poderiam ser reexaminadas para inclusão no próximo padrão, devido aos avanços na tecnologia de hardware.
- Para possibilitar maximizar a eficiência de codificação, a sintaxe do novo padrão não poderia ser compatível com os padrões anteriores.

Então, para permitir um avanço mais acelerado na construção do novo padrão e para evitar duplicação de esforços, o ITU-T e o ISO concordaram em unir esforços para desenvolverem, em conjunto, a próxima geração de padrão para codificação de vídeo e concordaram em usar o H.26L como ponto de partida. Então, foi criado, em dezembro de 2001, o JVT (*Joint Video Team*) (ITU-T, 2005), formado por especialistas do VCEG a do MPEG. O JVT tinha o objetivo de completar o desenvolvimento técnico do padrão até o ano de 2003. A ITU-T decidiu adotar o padrão com o nome de ITU-T H.264 e a ISO decidiu adotar o padrão com o nome de MPEG-4 parte 10 – AVC (*Advanced Video Coding* - Codificação de Vídeo Avançada). O padrão H.264 teve seu *draft* final (ITU-T, 2003) aprovado em outubro de 2003 (Sullivan, 2005).

Em julho de 2004, o JVT adicionou algumas novas funcionalidades ao padrão H.264 através de uma extensão do padrão chamada de *Fidelity Range Extensions* (FREXt) (ITU-T, 2004; ITU-T, 2005b).

Terminologia

Para o padrão H.264 a formação de uma imagem codificada pode acontecer a partir de um campo, quando o vídeo é entrelaçado, ou de um quadro, quando o vídeo é progressivo ou entrelaçado (Richardson,2003).

Um quadro codificado possui um número de quadro, que é sinalizado no *bitstream*. Este número de quadro não está, necessariamente, relacionado à ordem de decodificação deste quadro. Cada campo codificado de um quadro progressivo ou entrelaçado possui um contador que indica a sua ordem de decodificação. Os quadros codificados anteriormente podem ser utilizados como quadros de referência para a predição de outros quadros. Os quadros de referência são organizados em duas listas, chamadas de lista 0 e lista 1.

O padrão H.264 considera que o espaço de cores utilizado para representar o vídeo de entrada é do tipo luminância e crominância, como o YCbCr. A relação entre os elementos Y, Cb e Cr é dependente do perfil do padrão que está sendo considerado. Esta relação e os perfis do padrão serão apresentados ainda nesta seção.

Uma figura codificada consiste de um número de macroblocos, cada um contendo 16x16 amostras de luminância, associadas às amostras de crominância. Dependendo do perfil utilizado, o tamanho das amostras de crominância de um macrobloco pode variar, como ficará mais claro nos próximos parágrafos. Um macrobloco pode ainda ser dividido em blocos com 4x4 amostras de luma ou de croma.

Dentro de cada quadro, os macroblocos são organizados em *slices*, onde um *slice* é um grupo de macroblocos em uma ordem de varredura tipo *raster*. Um *slice* do tipo I pode conter somente macroblocos do tipo I. Um *slice* do tipo P pode conter

macroblocos do tipo P e I e um *slice* do tipo B pode conter macroblocos do tipo B e I. Além dos *slices* tipo I, P e B, o padrão também permite a existência de outros dois tipos de *slices*: SI e SP. Um quadro do vídeo pode ser codificado em um ou mais *slices*, cada um contendo um número inteiro de macroblocos. O número de macroblocos pode variar de um até o número total de macroblocos do quadro (Richardson,2003).

Os slices SP (*Switching* P) e SI (*Switching* I) são *slices* que são transmitidos para permitir o chaveamento entre fluxos de bits diferentes, mas sem causar um grande prejuízo na eficiência de codificação (Karczewicz, 2003). É comum, em codificação de vídeo, o chaveamento entre fluxos de bits diferentes, mas este chaveamento não pode ser acontecer logo antes de um quadro P ou B, pois a codificação destes quadros usa como referência quadros do fluxo de bits atual. É possível fazer o chaveamento usando um quadro do tipo I, mas o problema é que estes quadros consomem muito mais bits do que os quadros P ou B. Por isso o padrão H.264 prevê a utilização de *slices* tipo SP e SI no seu perfil *Extended*, que será apresentado na próxima seção do texto.

Os macroblocos do tipo I são codificados usando a codificação intraframe a partir das amostras do *slice* atual. A codificação pode acontecer sobre macroblocos completos ou para cada bloco. Os macroblocos do tipo I também podem ser codificados através do modo I_PCM, onde o codificador transmite os valores das amostras diretamente, sem predição ou transformação. Em alguns casos anômalos, o modo I_PCM pode ser mais eficiente do que os demais modos de codificação (RICHARDSON,2003).

Os macroblocos do tipo P são codificados usando a codificação interframe, a partir de quadros de referência previamente codificados. Um macrobloco codificado no modo inter pode ser dividido em partições de macroblocos, isto é, em blocos de 16x16, 16x8, 8x16 ou 8x8. Se o tamanho de partição escolhido for o 8x8, então cada sub-macrobloco 8x8 pode ser dividido novamente em partições de sub-macrobloco de tamanho 8x8, 8x4, 4x8 ou 4x4. Cada partição de macrobloco é codificada utilizando como referência um quadro da lista 0. Se existir uma partição de sub-macrobloco, então esta partição é codificada utilizando o mesmo quadro da lista 0 utilizado para codificar a partição de macrobloco.

Os macroblocos do tipo B também são codificados usando a codificação interframe. Cada partição de macrobloco pode ser codificada utilizando um ou dois quadros de referência, um na lista 0 e outro na lista 1. Se existir uma partição em sub-macrobloco, cada sub-macrobloco é codificado a partir dos mesmos um ou dois quadros de referência utilizados na codificação da partição de macrobloco.

Perfis e Níveis

O padrão H.264 é dividido em diferentes perfis. Cada perfil suporta um grupo particular de funções de codificação e especifica o que é necessário para cada codificador e decodificador que seguem este perfil. A primeira versão do padrão H.264, de maio de 2003 (ITU-T, 2003), define um grupo de três diferentes perfis: Baseline, Main e Extended. O perfil Baseline é direcionado a aplicações como videotelefonia, videoconferência e vídeo sem fio. O perfil Baseline suporta codificação intra e inter (usando somente *slices* I e P) e uma codificação de entropia com códigos de comprimento de palavra variável adaptativos ao contexto (CAVLC). O perfil Main é focado na transmissão de televisão e no armazenamento de vídeo. O perfil Main inclui o suporte para vídeo entrelaçado, o suporte à codificação inter utilizando *slices* do tipo B e utilizando predição ponderada e o suporte à codificação de entropia utilizando

codificação aritmética adaptativa ao contexto (CABAC). O perfil Extended é mais voltado para aplicações em *streaming* de vídeo e não suporta vídeo entrelaçado ou o CABAC, mas agrega modos para habilitar uma troca eficiente entre *bitstreams* codificados (através de *slices* do tipo SP e SI) e melhora a resiliência a erros (através do particionamento de dados).

Para os três perfis definidos na primeira versão do padrão, a relação entre os elementos de cores é fixa. Esta relação é de 4:2:0 para os elementos Y, Cb e Cr. Isso significa que os elementos de crominância Cb e Cr possuem metade da resolução horizontal e vertical dos elementos de luminância. Isso implica em uma sub-amostragem dos elementos de crominância já no vídeo original. Esta sub-amostragem contribui na redução da redundância psicovisual. Cada macrobloco, como já foi apresentado anteriormente, é formado por uma região de 16x16 pixels de um quadro do vídeo, incluindo as informações de luminância e crominância. Com a relação de 4:2:0, um macrobloco é formado por uma matriz de 16x16 amostras de luminância e por duas matrizes 8x8 amostras de crominância, uma para Cb e outra para Cr.

Os perfis Baseline, Main e Extended também possuem outra característica em comum, neste caso, relacionada à quantidade de bits utilizados por amostra. Nos três padrões, são usados 8 bits por amostra.

Estes três perfis inicialmente propostos pelo padrão H.264 foram focados em vídeos de entreterimento ou de qualidade. Mas estes perfis não incluíram suporte para vídeos com resoluções mais elevadas, como as necessárias em ambientes profissionais. Para responder às exigências deste tipo de aplicação, uma continuação do projeto JVT foi realizada para adicionar novas extensões para as capacidades do padrão original. Estas extensões foram chamadas de extensões para alcance de fidelidade (*fidelity range extensions* - FRExt). O FRExt produziu um grupo de quatro novos perfis chamados coletivamente de perfis High (Sullivan, 2004).

Os perfis High possuem algumas características comuns que são inovadoras em relação aos perfis originais: suportam um tamanho de bloco adaptativo para a transformada (4x4 ou 8x8), suportam matrizes de quantização baseadas em percepção e suportam uma representação sem perdas de regiões específicas do conteúdo do vídeo (Sullivan, 2004).

As matrizes de quantização baseadas em percepção foram usadas anteriormente no padrão MPEG-2. Com esta ferramenta, o codificador pode especificar, para cada tamanho de bloco da transformada e separadamente para codificação intra e inter, um fator de escala customizado. Isso permite um ajuste da fidelidade da quantização de acordo com o modelo de sensibilidade do sistema visual humano para diferentes tipos de erros. Tipicamente, esta ferramenta não melhora a fidelidade objetiva (PSNR), mas melhora a fidelidade subjetiva, que é o critério realmente mais importante.

O perfil High (HP) inclui vídeo com 8 bits por amostra e com relação de cor 4:2:0. O perfil High 10 (Hi10P) suporta vídeo a 10 bits por amostra, também com uma relação de cor 4:2:0. O perfil High 4:2:2 (H422P) inclui suporte à sub-amostragem de cor 4:2:2 e vídeo a 10 bits por amostra. Finalmente, o perfil High 4:4:4 (H444P) dá suporte à sub-amostragem de cor 4:4:4, suporte a vídeo com até 12 bits por amostra e, adicionalmente, suporta a codificação sem perdas de regiões do vídeo.

A Figura A.1 apresenta resumidamente a relação entre os perfis Baseline, Main, Extended e High do padrão H.264.

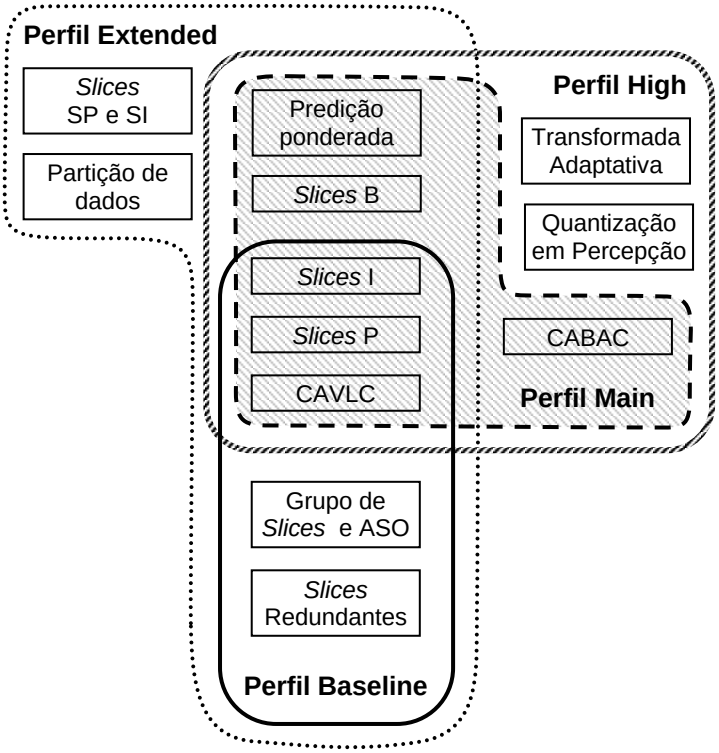

Figura A.1: Perfis Baseline, Main, Extended e High do padrão H.264

Além da divisão em diversos perfis, o padrão H.264 também define 16 diferentes níveis em função da taxa de processamento e da quantidade de memória necessária para cada implementação. Com a definição do nível utilizado, é possível deduzir o número máximo de quadros de referência e a máxima taxa de bits que podem ser utilizados (Sullivan, 2004).

Formato de Dados Codificados

Uma interessante inovação do padrão H.264 é que ele faz uma clara distinção entre a organização do vídeo codificado e da sua transmissão via rede. O padrão classifica a informação do vídeo em dois diferentes níveis de abstração, chamados de camada de vídeo codificado (VCL – *Vídeo Coding Layer*) e camada de abstração de rede (NAL – *Network Abstration Layer*) (Richardson,2003). Os dados de saída do processo de codificação estão na camada VCL, sendo formados por uma seqüência de bits que representam os dados do vídeo codificado. Estes dados são mapeados para unidades NAL antes da transmissão ou armazenamento. Uma seqüência de vídeo codificado é representada por uma seqüência de unidades NAL que podem ser transmitidas sobre uma rede baseada em pacotes, podem ser transmitidas via um link de transmissão de *bitstream* ou ainda, podem ser armazenados em um arquivo.

O objetivo de especificar separadamente a VCL e a NAL é diferenciar características específicas da codificação (na VCL) daquelas características específicas do transporte (na NAL). Neste trabalho não serão apresentados em maiores detalhes o mecanismo de transporte utilizado na NAL. Maiores detalhes podem ser encontrados na literatura (ITU-T, 2005b; Richardson,2003).

Comparação com Padrões Anteriores

A Tabela A.1 apresenta uma comparação entre os padrões H.264, MPEG-4 Parte 2 e MPEG-2 do ponto de vista das diferentes características suportadas por estes padrões (Kwon, 2005). As características do H.264 que ainda não foram discutidas nas seções anteriores serão explicadas em mais detalhes no decorrer do texto.

O padrão H.264 teve como principal objetivo, como já foi citado anteriormente, aumentar a eficiência da codificação. Nos próximos parágrafos será apresentada uma comparação entre o padrão H.264 e outros padrões, com foco na eficiência de codificação. Os dados apresentados nesta seção foram colhidos da literatura, e apresentam algumas diferenças, mas todos indicam que o padrão H.264 apresenta ganhos significativos sobre outros padrões em termos de eficiência de codificação.

Tabela A.1: Comparação das características dos padrões H.264, MPEG-4 Parte 2 e MPEG-2

	H.264	*MEPG-4 parte 2*	*MPEG-2*
Tamanho de Macrobloco	16x16	16x16	16x16 (modo quadro) 16x8 (modo campo)
Tamanho de bloco	16x16, 8x16, 16x8, 8x8, 4x8, 8x4, 4x4	16x16, 16x8, 8x8	8x8
Predição Intra	Domínio espacial	Domínio das freqüências	Não
Transformada	DCT inteira 8x8, 4x4 Hadamard 4x4, 2x2	DCT/Wavelet 8x8	DCT 8x8
Quantização	Escalar, com passo crescendo a uma taxa de 12,5%	Vetorial	Escalar, com incremento de passo constante
Codificação de Entropia	VLC, CAVLC, CABAC	VLC	VLC
Precisão de pixels	¼ pixel	¼ pixel	½ pixel
Quadros de referência	Múltiplos quadros	Um quadro	Um quadro
Modo de predição bidirecional	Para frente / para trás Para frente / para frente Para trás / para trás	Para frente / para trás	Para frente / para trás
Predição ponderada	Sim	Não	Não
Filtro de blocagem	Sim	Não	Não
Tipos de quadros	I, P, B, SI, SP	I, P, B	I, P, B
Perfis	7 perfis	8 perfis	5 perfis
Taxa de transmissão	64Kbps a 150Mbps	64Kbps a 2Mbps	2Mbps a 15Mbps
Complexidade do codificador	Elevada	Média	Média
Compatibilidade com os padrões anteriores	Não	Sim	Sim

O trabalho de (Kamaci, 2003) apresenta uma comparação entre o padrão H.264, o padrão MPEG-2 e o padrão H.263, este último nos perfis *Baseline* e *High*. Nesta

comparação, é considerada a relação sinal/ruído (PSNR) sobre vídeos nos formatos CCIR-601, CIF e QCIF. De acordo com os autores, o padrão H.264 atinge cerca 50% de ganho de codificação sobre o padrão MPEG-2, cerca de 47% de ganho sobre o padrão H.263 *Baseline* e cerca de 24% de ganho sobre o padrão H.263 *High*.

Outro trabalho (Wiegand, 2003) compara a relação sinal ruído entre os padrões MPEG-2, H.263, MPEG-4 e H.264. Foram usadas seqüências de vídeos no formato QCIF e CIF, com diferentes taxas de quadros por segundo. A Tabela A.2 apresenta os resultados comparativos nos ganhos médios na taxa de bits do padrão H.264 sobre os demais padrões.

Tabela A.2: Ganhos comparativos na taxa de bits em relação a outros padrões

Aplicação	*MPEG-4 (%)*	*H.263 (%)*	*MPEG-2 (%)*
Streaming	37,4	47,6	63,6
Vídeo Conferência	29,4	40,6	-
Entreterimento / Qualidade	-	-	45

As Figuras A.2, A.3 e A.4 apresentam exemplos de gráficos comparativos montados a partir dos diversos experimentos realizados no trabalho (Wiegand, 2003). Nestas figuras são apresentados os gráficos do ganho na taxa de bits e os gráficos da economia em relação ao MPEG-2. A Figura A.2 apresenta a comparação quando a aplicação de *streaming* é considerada. Na Figura A.3 está apresentada a comparação para aplicação de vídeo conferência. Finalmente, a Figura A.4 está a comparação para aplicações de entreterimento/qualidade.

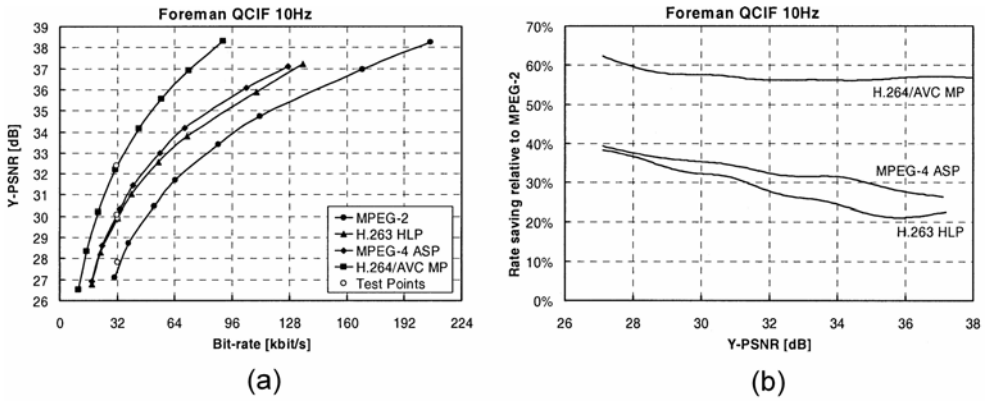

Figura A.2: Comparação (a) dos ganhos na taxa de bits e (b) na economia na taxa de bits comparada o MPEG-2, para o vídeo "Foreman" considerando resolução de vídeo para *streaming*

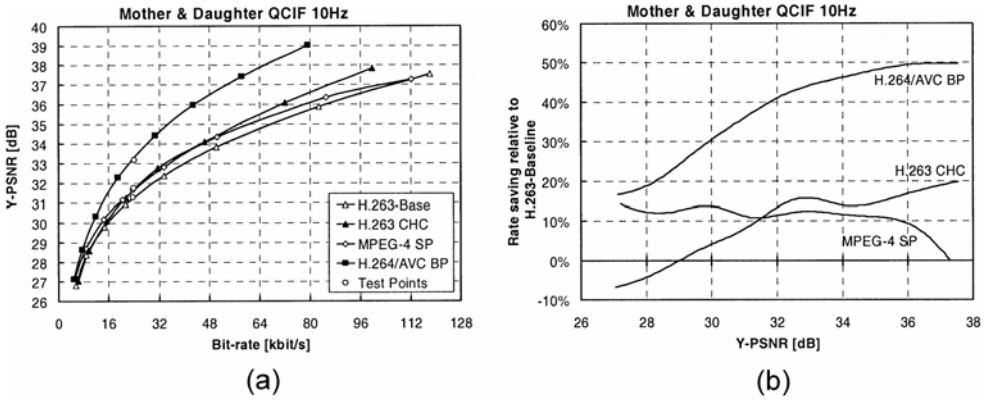

Figura A.3: Comparação (a) dos ganhos na taxa de bits e (b) na economia na taxa de bits comparada o MPEG-2, para o vídeo "Mother & Daughter" considerando resolução de vídeo para vídeo conferência.

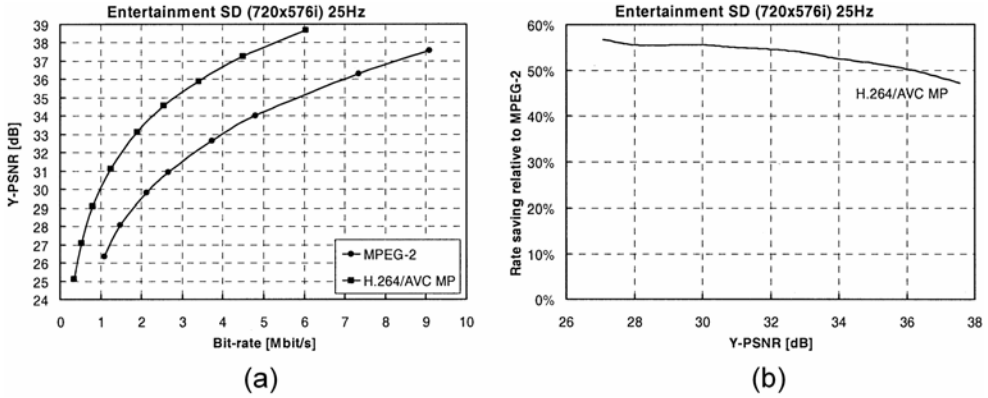

Figura A.4: Comparação (a) dos ganhos na taxa de bits e (b) na economia na taxa de bits comparada o MPEG-2, para o vídeo "Entertainment" considerando resolução de vídeo para entretenimento/qualidade.

Em todos os casos apresentados nas Figuras A.2, A.3 e A.4, o padrão H.264 apresentou desempenho superior aos demais padrões. Esta capacidade do padrão H.264 de atingir as maiores eficiências de codificação para diversos tipos de aplicação é uma das mais importantes vantagens deste padrão em relação aos demais padrões de compressão de vídeo que, normalmente, são direcionados para um tipo de aplicação específica.

Os ganhos na relação sinal ruído, vêm acompanhados de um considerável aumento da complexidade computacional dos algoritmos utilizados no H.264 em relação aos algoritmos utilizados nos padrões anteriores. Segundo o trabalho (Sunna, 2005) a relação entre o acréscimo na complexidade versus os ganhos na eficiência da codificação quando o padrão MPEG-2 é tomado como base, pode é definido de acordo com os dados apresentados na Tabela A.3. Na comparação da Tabela A.3, são considerados os três perfis presentes na primeira versão do padrão H.264 e é considerada a complexidade apenas do decodificador H.264. Os dados apresentados na Tabela A.3 são aproximados e apresentam apenas uma idéia dos ganhos de eficiência e do acréscimo em complexidade.

Tabela A.3: Relação entre o H.264 e o MPEG-2 quanto aos ganhos em eficiência de codificação e o acréscimo de complexidade.

Perfil	*Ganho em Eficiência de Codificação (vezes)*	*Acréscimo de Complexidade (vezes)*
Baseline	1,5	2,5
Extended	1,75	3,5
Main	2	4

O acréscimo na complexidade computacional de um codificador H.264 é ainda mais elevado do que o acréscimo na complexidade do decodificador quando o padrão MPEG-2 é utilizado como comparação. Considerando o codificador, o H.264 é cerca de oito vezes mais complexo do que o MPEG-2.

Núcleo do Codec H.264

Esta seção do texto irá apresentar os principais blocos de um codificador e de um decodificador H.264, bem como suas funcionalidades.

O diagrama de blocos do codificador H.264 está apresentado na Figura A.5. O bloco de Predição Interframe na Figura A.5 é formado pela estimação de movimento (ME) e pela compensação de movimento (MC), sendo que para desempenhar esta função o padrão H.264 agrega operações de grande complexidade computacional a este bloco.

Há ainda os blocos das transformadas diretas e inversas (T e T-1), os blocos da quantização direta e inversa (Q e Q-1) e o bloco da Codificação de Entropia, apresentados na Figura A.5.

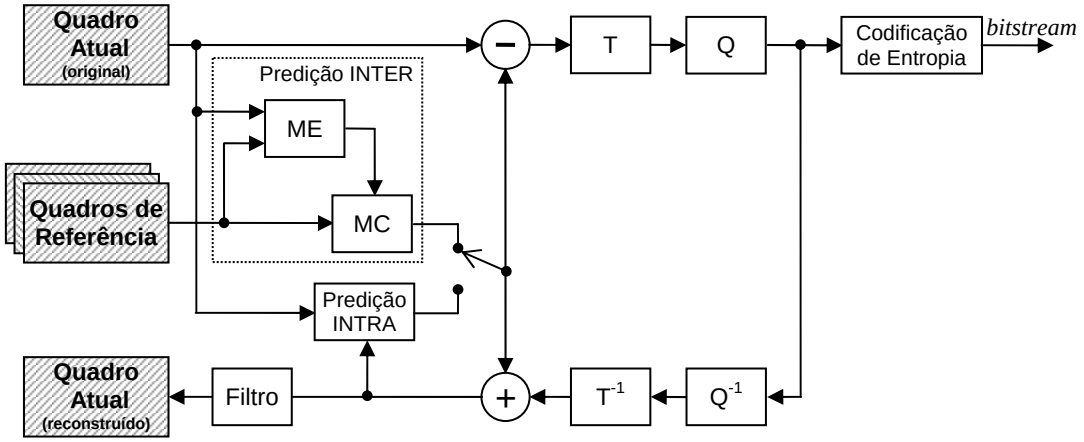

Figura A.5: Diagrama em blocos de um codificador H.264

O bloco chamado de Filtro foi inserido no padrão H.264 e é requisito obrigatório. Nos demais padrões de compressão de vídeo este filtro não é obrigatório, mas um filtro similar é usado na prática por várias implementações seguindo outros padrões. Este filtro, também chamado de filtro de deblocagem ou filtro de *loop*, é usado para minimizar o efeito de bloco. Este filtro será explicado em maiores detalhes mais adiante no texto, em seção própria.

A Figura A.6 apresenta o diagrama em blocos de um decodificador H.264. Novamente esta figura é muito parecida com a Figura 2.3 apresentada na seção 2.3. Também no decodificador H.264 está presente o filtro, que não está originalmente apresentado na Figura 2.3.

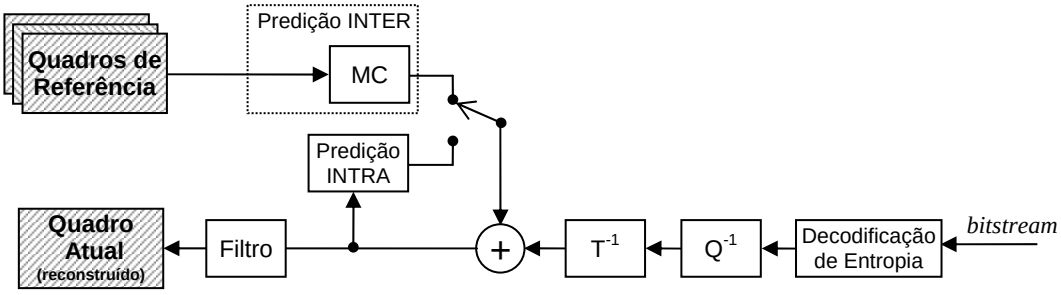

Figura A.6: Diagrama em blocos de um decodificador H.264

Cada um dos blocos do codificador e do decodificador será explicado, nas próximas seções, de acordo com sua funcionalidade dentro do padrão H.264.

Nas etapas de estimação e compensação de movimento (Kuhn, 1999) do padrão H.264 (bloco ME na Figura A.5 e bloco MC nas Figuras A.5 e A.6) se encontram a maior parte das inovações e dos ganhos obtidos pelo H.264 sobre os demais padrões. Uma importante inovação é o uso de blocos de tamanho variável, onde se pode usar uma partição do macrobloco em blocos de tamanho 16x16, 16x8, 8x16, 8x8, 4x8, 8x4 ou 4x4 para realizar a ME e a MC. Outra inovação é a precisão de ¼ de pixel, para buscar o melhor casamento e para realizar a reconstrução do quadro. O H.264 também prevê o uso de múltiplos quadros de referência, que não precisam ser somente os quadros I ou P imediatamente anteriores ou posteriores. Também é uma inovação o uso de predição bi-preditiva, ponderada e direta para os slices do tipo B. Além disso, os vetores de movimento podem apontar para fora da borda do quadro. Por fim, a predição de vetores de movimento com base nos vetores vizinhos também é uma novidade (ITU-T, 2003; Richardson, 2003; Wiegand, 2003a).

A etapa de predição INTRA (Figuras A.5 e A.6) é outra inovação introduzida pelo padrão H.264, pois mesmo nos macroblocos do tipo I é realizada uma predição antes da aplicação da transformada. Esta predição leva em conta as amostras já codificadas do *slice* atual.

No que diz respeito às transformadas direta e inversa (blocos T e T^{-1} nas Figuras A.5 e A.6), o H.264 introduziu algumas inovações. A primeira delas é que as dimensões da transformada foram definidas em 4x4 ao invés do tradicional 8x8. Outra inovação é relativa ao uso de uma aproximação inteira da transformada DCT direta e inversa, de modo a facilitar a sua implementação em hardware de ponto fixo e evitar erros de casamento entre o codificador e o decodificador (Malvar, 2003). Além disso, uma segunda transformada é aplicada sobre os elementos DC resultantes da DCT para todos os blocos de crominância (Hadamard 2x2) e para os macroblocos em que é feita a predição INTRA 16x16 (Hadamard 4x4) (Richardson, 2003).

A quantização no padrão H.264 (blocos Q e Q^{-1} nas Figuras A.5 e A.6) é uma quantização escalar (Richardson, 2002). O fator de quantização é função de um parâmetro **QP** de entrada, que é usado no codificador para controlar a qualidade da compressão e o *bit-rate* de saída.

O padrão H.264 normatiza a utilização de um filtro redutor do efeito de bloco (bloco Filtro nas Figuras A.5 e A.6). Este filtro era opcional na maioria dos demais padrões de compressão de vídeo, mas passou a ser obrigatório no H.264. Uma inovação importante do filtro definido no padrão H.264 é que este filtro é adaptativo, conseguindo distinguir entre uma aresta real da imagem de um artefato gerado por um elevado passo de quantização.

Na codificação de entropia (Figuras A.5 e A.6) o H.264 também introduz ferramentas que aumentam bastante a eficiência de codificação deste bloco. Há, basicamente, dois tipos de codificação: a codificação de comprimento variável adaptativa ao contexto (CAVLC) e a codificação aritmética adaptativa ao contexto (CABAC). A principal inovação é o uso de codificação adaptativa baseada em contextos. Nesta codificação, a maneira com que os diversos elementos sintáticos são codificados depende do elemento a ser codificado, da fase em que se encontra o algoritmo de codificação e dos elementos sintáticos que já foram codificados.

Uma característica interessante do codificador H.264 é que apenas o decodificador é normatizado pelo padrão. Então vários graus de liberdade podem ser explorados na implementação do codificador. O codificador deve gerar um *bitstream* compatível com as exigências do padrão, mas a forma como este *bitstream* é gerado não é normatizada. Então várias opções de implementação podem ser realizadas no codificador que vão deste a inserção de algoritmos mais eficientes ou mais fáceis de serem implementados em hardware, até a simples eliminação de algumas possibilidades de codificação previstas pelo padrão. É claro que estas modificações poderão gerar impactos na relação sinal/ruído, na taxa de compressão, na velocidade de processamento e/ou na utilização de recursos de hardware pelo codificador e estes impactos devem ser criteriosamente avaliados para tomar qualquer decisão nesta direção.

O Bloco da Estimação de Movimento (ME)

A predição interframe no codificador H.264, é formada pelos blocos da Estimação de Movimento (ME) e da Compensação de Movimento (MC). Esta seção focará o bloco da Estimação de Movimento, mas ambos os blocos estão estreitamente relacionados.

O bloco da estimação de movimento está presente apenas nos codificadores H.264. Este bloco é o que apresenta a maior complexidade computacional dentre todos os blocos de um codificador H.264 (Puri, 2004). Este grande custo computacional é função das inovações inseridas neste bloco do padrão, que tiveram o objetivo de atingir elevadas taxas de compressão. Reside nos blocos da ME e MC as principais fontes de ganhos do H.264 em relação aos demais padrões de compressão de vídeo (Wiegand, 2003, Richardson, 2003).

A estimação de movimento deve prover as ferramentas de codificação capazes de localizar, nos quadros de referência, qual macrobloco mais se assemelha ao macrobloco atual. Assim que é encontrado este macrobloco, a ME deve gerar um vetor indicando a posição deste macrobloco no quadro de referência. Este vetor é chamado de vetor de movimento e deve ser inserido junto com a codificação do macrobloco.

Para a realização da estimação de movimento é considerado apenas o componente de luminância do macrobloco. Cada componente de crominância possui a metade da resolução horizontal e vertical do componente de luminância, então os componentes horizontal e vertical de cada vetor de movimento são divididos por dois para serem aplicados aos blocos de croma.

A principal inovação do H.264 no ponto de vista da estimação de movimento está na possibilidade de utilização de tamanhos de blocos variáveis para realizar a estimação de movimento. Ao invés de usar um macrobloco inteiro na estimação de movimento, o padrão H.264 permite o uso de partições de macrobloco e partições de sub-macroblocos. As partições de macroblocos possuem tamanhos de 16x16, 8x16, 16x8 e 8x8, como está apresentado na Figura A.7 (Wiegand, 2003, Richardson, 2003).

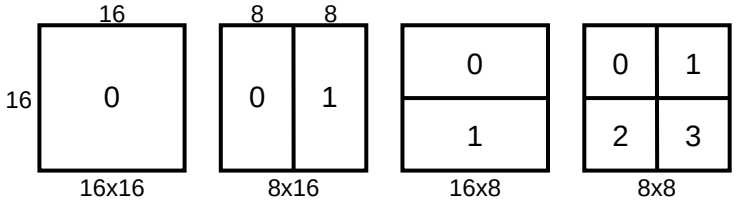

Figura A.7: Divisão do macrobloco em partições de macroblocos

As partições de sub-macroblocos são permitidas somente se a partição de macrobloco selecionada foi a de 8x8. Neste caso, as quatro partições 8x8 do macrobloco podem ser divididas em mais quatro formas, chamadas sub-macroblocos, que podem ter o tamanho 8x8, 4x8, 8x4 ou 4x4, como está apresentado na Figura A.8.

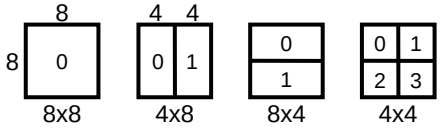

Figura A.8: Divisão de uma partição de macrobloco em partições de sub-macroblocos

Este método de particionar macroblocos em sub-blocos de tamanho variável é conhecido como compensação de movimento estruturada em árvore.

Cada bloco de croma é dividido da mesma maneira que os blocos de luma, exceto pelo tamanho da partição, que terá exatamente a metade da resolução horizontal e vertical da partição de luma. Por exemplo, uma partição de 8x16 de luma corresponde a uma partição de 4x8 de croma.

Um vetor de movimento é necessário para cada partição ou sub-macrobloco. Cada vetor de movimento precisa ser codificado e transmitido e a escolha da partição precisa ser codificada no *bitstream* comprimido. A escolha de um tamanho de partição grande (16x16, 16x8, 8x16) implica na utilização de um número menor de bits para identificar o vetor de movimento escolhido e para indicar o tipo de partição utilizada. Por outro lado, o resíduo gerado pode conter uma quantidade significativa de energia em áreas com muitos detalhes. A escolha de um tamanho de partição pequeno (8x4, 4x4) pode gerar um resíduo com quantidade de energia mínima depois da compensação de movimento, mas esta escolha requer a utilização de um número de bits maior para representar o vetor de movimento e para identificar a partição escolhida. Como pode ser percebido, a escolha do tamanho da partição possui um impacto significativo no desempenho da compressão (Richardson,2003).

Em geral, uma partição grande é apropriada para áreas mais homogêneas do quadro e uma partição menor tende a ser mais apropriada para áreas com muitos detalhes.

O tamanho de partição é escolhido a partir de alguma métrica que conduza a uma codificação mais eficiente, isto é, que analise o compromisso entre a geração de um resíduo mínimo e a geração de um número mínimo de vetores de movimento. A melhor escolha é realizada tomando por base os resultados da análise para todos os tamanhos de

partição de cada macrobloco. Então, é escolhido aquele tamanho de partição que apresentar a maior eficiência de codificação. A complexidade computacional desta operação, como pode ser percebido, é bastante grande, uma vez que a os cálculos da estimação de movimento são realizados para vários tamanhos de partição diferentes e apenas uma destas partições é escolhida. Por isso, a estimação de movimento é o bloco mais crítico na implementação de codificadores H.264.

Uma outra característica importante da estimação de movimento do padrão H.264 é que ela prevê uma precisão de ¼ de pixel para os vetores de movimento. Normalmente, os movimentos que acontecem de um quadro para o outro não estão restritos a posições inteiras de pixel. Assim, se são utilizados apenas vetores de movimento com valores inteiros, normalmente não é possível encontrar casamentos bons. Por isso, o padrão H.264 prevê a utilização de vetores de movimento com valores fracionários de ½ pixel e de ¼ de pixel. Na Figura A.9 é apresentado um exemplo da precisão do vetor de movimento com valores fracionários.

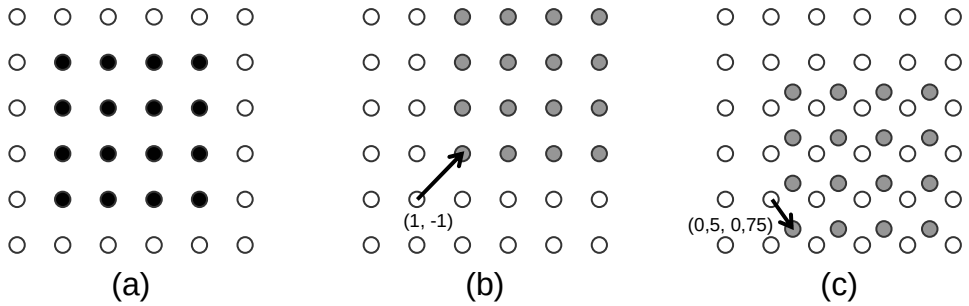

Figura A.9: Estimação de movimento com precisão de frações de pixel

Na Figura A.9 (a) está representado, com pontos pretos, um bloco 4x4 do quadro atual que será predito a partir de uma região do quadro de referência na vizinhança da posição do bloco atual. Se os componentes horizontal e vertical do vetor de movimento são inteiros, então as amostras necessárias para o casamento estão presentes no bloco de referência. Como exemplo, a Figura A.9 (b) apresenta, com pontos em cinza, um possível casamento do bloco atual no quadro de referência, onde o vetor de movimento utiliza apenas valores inteiros, neste caso, o vetor de movimento é (-1, 1). Se um ou os dois componentes do vetor de movimento apresentam valores fracionários, então as amostras preditas são geradas a partir da interpolação entre amostras adjacentes do quadro de referência. Como exemplo, a Figura A.9 (c) apresenta um casamento com um vetor de movimento que utiliza dois valores fracionários, com as amostras sendo geradas por interpolação. No caso do exemplo, o vetor é (0,5 , 0,75).

No H.264, a estimação de movimento usando ¼ de pixel é obrigatória. Na prática a estimação é realizada com precisão de ¼ de pixel para os componentes de luminância. Para os elementos de crominância a precisão é, na verdade, de 1/8 de pixel.

A interpolação para ¼ de pixel de pixel é realizada em dois passos. No primeiro passo, é realizada a interpolação dos quadros para ½ pixel usando um filtro FIR de seis *taps*, com pesos (1/32, -5/32, 5/8, 5/8, -5/32, 1/32), da forma como está descrito na Figura A.10. Na Figura A.10, as posições inteiras são representadas por letras maiúsculas e as de ½ pixel por letras minúsculas.

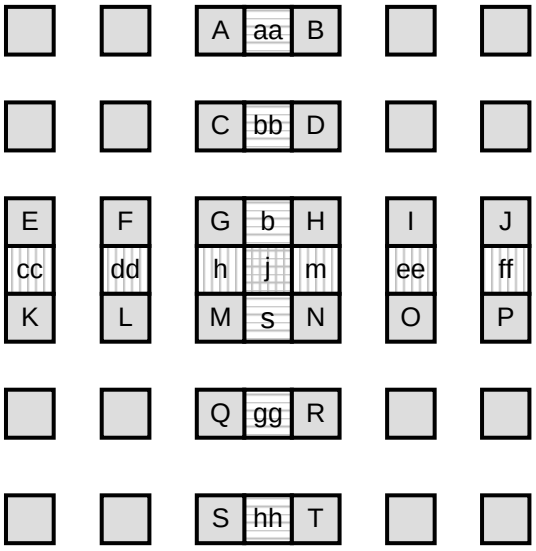

Figura A.10: Interpolação para posições de ½ pixel para o componente de luminância.

Como exemplo, a posição **b** da Figura A.10 é calculada como está indicado em (1).

$$b = (E - 5 \cdot F + 20 \cdot G + 20 \cdot H - 5 \cdot I + J)/32 \qquad (1)$$

De forma similar, **h** é interpolado através da filtragem de **A**, **C**, **G**, **M**, **Q** e **S**. Por outro lado, **j** é gerado através da filtragem de **aa**, **bb**, **b**, **s**, **gg** e **hh**, que foram gerados a partir da primeira filtragem realizada apenas sobre elementos inteiros. É importante ressaltar que o filtro pode ser aplicado horizontalmente ou verticalmente.

No segundo passo, são interpoladas as posições de ¼ de pixel a partir das amostras construídas no primeiro passo e das amostras inteiras, usando a média simples entre dois pontos, na relação que está apresentada na Figura A.11. Como exemplo, o cálculo da posição **a** na Figura A.11 é realizado como está apresentado em (2).

$$a = (G + b)/2 \qquad (2)$$

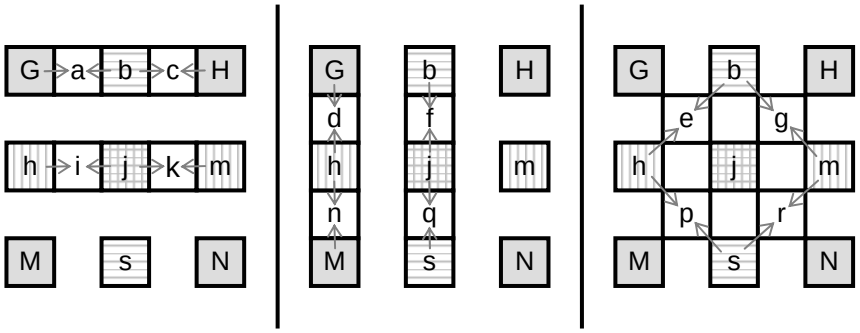

Figura A.11: Interpolação para posições de ¼ de pixel

Para os vetores de crominância, que utilizam uma resolução de $1/8$ de pixel, usa-se interpolação linear. As amostras interpoladas são geradas em intervalos de oito amostras entre amostras inteiras em cada componente de crominância. Considerando a Figura A.12 como referência, cada posição interpolada **a** é uma combinação linear das amostras inteiras da vizinhança que, no caso da Figura A.12, são as posições **A**, **B**, **C** e **D**.

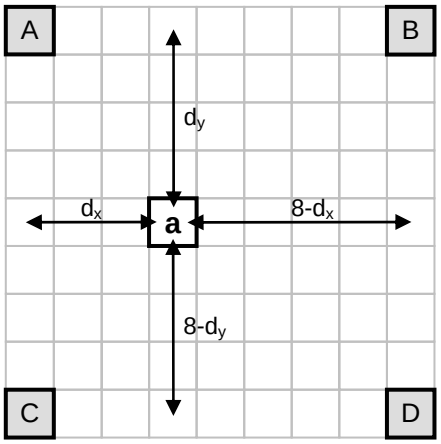

Figura A.12: Interpolação para componentes de crominância

A combinação linear das amostras inteiras é definida pela equação (3).

$$a = round\left\{\left[(8-d_x)\cdot(8-d_y)\cdot A + d_x\cdot(8-d_y)\cdot B + (8-d_x)\cdot d_y\cdot C + d_x\cdot d_y\cdot D\right]/64\right\} \qquad (3)$$

No exemplo da Figura A.13, $\mathbf{d_x}$ é igual a três e $\mathbf{d_y}$ é igual a quatro, então **a** é definido por (4) para este exemplo.

$$a = round\left[(20\cdot A + 12\cdot B + 20\cdot C + 12\cdot D)/64\right] \qquad (4)$$

Outra característica importante do padrão H.264 é o uso de múltiplos quadros de referência. No H.264 os quadros de referência não precisam ser somente os quadros I ou P imediatamente anteriores ou posteriores ao quadro atual. Há a opção de se usar como referência múltiplos quadros para frente ou para trás do quadro atual (Richardson, 2003), como no exemplo apresentado na Figura A.13.

O padrão H.264 define duas listas contendo os quadros de referência. A lista chamada de **lista 0** contém como primeiro quadro o quadro passado mais próximo, seguido de quaisquer outros quadros passados, seguidos de quaisquer outros quadros futuros. A lista chamada **lista 1** contém como primeiro quadro o quadro futuro mais próximo, seguido de quaisquer quadros futuros, seguidos de quaisquer outros quadros passados.

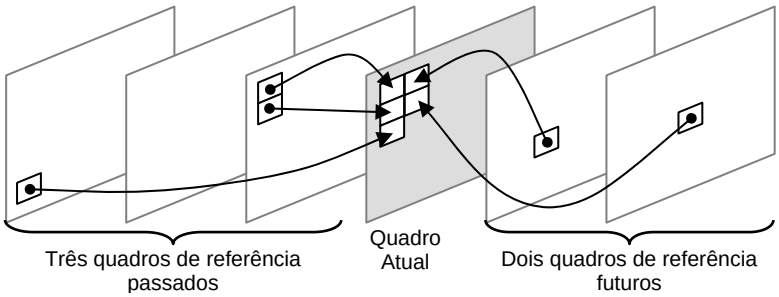

Figura A.13: Uso de múltiplos quadros de referência.

É importante destacar que o padrão H.264 permite que o codificador escolha uma ordem dos quadros para codificação completamente diferente da ordem dos quadros para apresentação do vídeo (Kwon,2005). Por isso, é possível armazenar nas listas 0 e 1, quadros futuros que, neste caso, são quadros futuros para a apresentação do vídeo, mas são quadros que já foram codificados.

As partições de macroblocos em um *slice* tipo I não utilizam estimação de movimento, uma vez que suas predições são do tipo intraframe, isto é, são realizadas apenas sobre amostras internas ao *slice* atual.

As partições de macroblocos em um *slice* tipo P podem ser codificados através de predições intraframe, de predições interframe ou através de macroblocos do tipo *skipped*. A predição interframe utiliza a estimação de movimento e pode ser realizada sobre quadros passados ou futuros que tenham sido codificados anteriormente ao quadro atual, armazenados na lista 0.

As partições de macroblocos em um *slice* B são codificadas através de predições interframe que podem ser geradas de diversas formas. É possível realizar a predição utilizando um quadro da lista 0, um quadro da lista 1 ou, ainda, utilizar um quadro de cada lista. Esta última opção é chamada de estimação bi-preditiva. Também é possível realizar a chamada estimação direta. Além destas opções, macroblocos do tipo *skip* também podem ser utilizados, como nos *slices* do tipo P.

Um macrobloco do tipo *skip* pode ser utilizado em *slices* tipo P ou B. Neste caso, nenhuma informação é transmitida para este macrobloco, nem mesmo o resíduo ou o vetor de movimento. Macroblocos do tipo *skip* são gerados quando o custo da taxa de distorção para codificar o macrobloco é mais elevado do que o custo de não enviar informação alguma sobre o macrobloco (Kannangara, 2005). Quando o decodificador encontra um macrobloco tipo *skip* em um *slice* P, o decodificador reconstrói o macrobloco a partir do primeiro quadro armazenado na lista 0, calculando seu vetor de movimento a partir dos vetores vizinhos. Se o macrobloco skip estiver em um tipo *slice* B, então o macrobloco é reconstruído no decodificador usando predição direta (Sahafi, 2005).

A estimação bi-preditiva utiliza um bloco de referência que é construído a partir de dois blocos, um na lista 0 e outro na lista 1. Neste caso, são necessários dois vetores de movimento, um para o quadro de referência da lista 0 e outro para o quadro de referência da lista 1. Então, cada amostra do bloco de predição é calculada como uma média entre as amostras dos quadros das listas 0 e 1. Este cálculo é apresentado em (5), onde **pred0(*i,j*)** e **pred1(*i,j*)** são amostras derivadas dos quadros de referência das listas 0 e 1 respectivamente e onde **pred(*i,j*)** é a amostra bi-preditiva.

$$pred(i,j) = (pred0(i,j) + pred1(i,j) + 1) \;\; >> 1 \qquad (5)$$

A equação (5) é válida se a estimação bi-preditiva não for do tipo predição ponderada. A predição ponderada utiliza pesos diferentes sobre as amostras dos macroblocos dos *slices* P ou B. Então, o vetor de movimento é construído uma combinação linear destas predições.

Quando a predição direta é usada em um quadro B, nenhum vetor de movimento é transmitido. Ao invés disso, o decodificador calcula os vetores da lista 0 e da lista 1 baseado nos vetores previamente codificados.

A estimação de movimento do padrão H.264 possui ainda características adicionais que contribuem para o aumento da eficiência de codificação. Uma destas características é a possibilidade de utilizar vetores de movimento que apontam para fora dos limites dos quadros. Neste caso é realizada uma extrapolação dos quadros nas bordas. Outra característica interessante é que quadros do tipo B podem ser usados como referência na predição, diferente do que acontece no padrão MPEG-2 (ITU-T, 1994). Por fim, para minimizar o custo da codificação dos vetores de movimento a partir da exploração da

correlação existente entre vetores de blocos vizinhos, o padrão H.264 prevê que os vetores sejam preditos a partir dos vetores calculados para as partições de macroblocos vizinhas.

Do que foi exposto até agora nesta seção é possível perceber que a estimação de movimento no H.264 é realmente muito complexa, por permitir o uso de técnicas variadas com o objetivo de atingir elevadas taxas de compressão. Uma questão importante e que deve ser considerada no projeto de codificadores H.264 em hardware é que o padrão H.264 normatiza apenas o decodificador. Por isso, a implementação do codificador pode ser realizada com vários graus de liberdade, como já foi citado. Então, várias das características da estimação de movimento previstas pelo padrão podem ser simplesmente ignoradas na construção do codificador para que a complexidade da compressão seja reduzida. Qualquer ação neste sentido irá impactuar negativamente na eficiência da codificação.

Existem diversos algoritmos utilizados para realizar a busca nos quadros de referência. Estes algoritmos variam desde a busca exaustiva (Bhaskaran, 1999) até algoritmos sub-ótimos que reduzem muito a complexidade da busca, mas que produzem impactos negativos na eficiência da codificação. Dentre estes algoritmos rápidos é possível citar a busca circular (Tourapis, 1999), a busca logarítmica (Jain, 1981), a busca em três passos (Li, 1994), a busca espiral (Kuhn, 1999; Kroupis, 2005), a busca em diamante (Yi, 2005) a busca hierárquica (Chu, 1997), entre outros.

Além dos algoritmos de busca é importante destacar a importância dos critérios de distância utilizados para decidir qual é o melhor casamento na procura sobre o quadro de referência. Existem vários critérios reportados na literatura dentre os quais cabe citar o erro quadrático médio (MSE), o erro absoluto médio (MAE), a soma de diferenças absolutas SAD, também chamada de soma de erros absolutos (SAE) (Kuhn, 1999), entre outras.

O Bloco da Compensação de Movimento (MC)

A etapa de compensação de movimento (bloco MC nas Figuras A.5 e A.6) deve adaptar-se às definições da estimação de movimento. A ME localiza o melhor casamento dentre os quadros de referência e gera um vetor de movimento. É função da compensação de movimento, a partir do vetor de movimento gerado na ME, localizar os blocos de melhor casamento na memória de quadros anteriormente codificados (quadros passados e/ou futuros) e montar o quadro predito. Este quadro será subtraído do quadro atual para gerar o quadro de resíduos que passará pela transformada.

A compensação de movimento é realizada tanto no codificador quanto no decodificador. A MC no codificador pode ser simplificada, tendo em vista as possíveis simplificações que podem ser realizadas na ME no codificador, conforme foi apresentado na seção anterior. Por outro lado, no decodificador, a compensação de movimento deve ser completa, isto é, deve estar apta a operar sobre todas as funcionalidades do padrão.

A compensação de movimento deve estar apta a atender as exigências da estimação de movimento. Deste modo, a MC deve:

- Tratar múltiplos tamanhos de partições de macroblocos;
- Utilizar múltiplos quadros de referência anteriores e posteriores;

- Interpretar corretamente os vetores construídos com base na predição de vetores;
- Tratar vetores que apontam para fora da borda do quadro;
- Reconstruir os macroblocos considerando uma precisão de ¼ de pixel para os vetores de movimento;
- Reconstruir os macroblocos que utilizam às predições bi-preditiva, ponderada e direta para *slices* do tipo B;
- Reconstruir corretamente os macroblocos do tipo *skip* para *slices* tipo P e B.

Como a ME foi abordada na seção anterior, todas as características da MC citadas acima já foram explicadas e, deste modo, não serão explicadas novamente nesta seção.

O Bloco de Predição Intra

O bloco de predição intra do padrão H.264 é responsável por realizar a predição nos macroblocos do tipo I. Esta predição é baseada nos valores previamente codificados do quadro atual dos pixels acima e à esquerda de um bloco. A predição intra para amostras de luminância pode ser utilizada sobre blocos 4x4 ou 16x16. Existem nove diferentes modos de predição intra para blocos 4x4 e quatro modos para a predição sobre blocos 16x16 (Richardson,2003). O bloco de predição intra está presente nos codificadores e nos decodificadores H.264 (Figuras A.5 e A.6).

A utilização de um bloco de predição intra é uma inovação no padrão H.264. Em função da predição intra e considerando também a predição inter, a transformada é sempre aplicada num sinal erro de predição. Como já foi mencionado anteriormente, existe um modo adicional de codificação para macroblocos do tipo I, chamado I_PCM. Neste caso, as amostras da imagem são transmitidas diretamente, sem predição, transformada e quantização (Richardson, 2003).

A parte destacada em cinza na Figura A.14 mostra um bloco 4x4 de luma a ser codificado pela predição intra. As amostras acima e a esquerda (letras A a M na Figura A.14) foram codificadas e reconstruídas antes deste bloco ser processado. Estas amostras serão utilizadas como referências para a predição intra.

M	A	B	C	D	E	F	G	H
I	a	b	c	d				
J	e	f	g	h				
K	i	j	k	l				
L	m	n	o	p				

Figura A.14: Identificação das amostras para a predição intra

A Figura A.15 apresenta os nove tipos diferentes de codificação no modo intra para blocos 4x4 de luminância. Os modos 0 e 1 fazem uma simples extrapolação (uma cópia) dos pixels das bordas verticais ou horizontais para todas as posições do bloco. O modo DC (2) faz uma média entre as amostras das bordas e copia o resultado para todas as posições do bloco. Os demais modos (3 a 8) fazem uma média ponderada das amostras das bordas, de acordo com a direção da seta na Figura A.15.

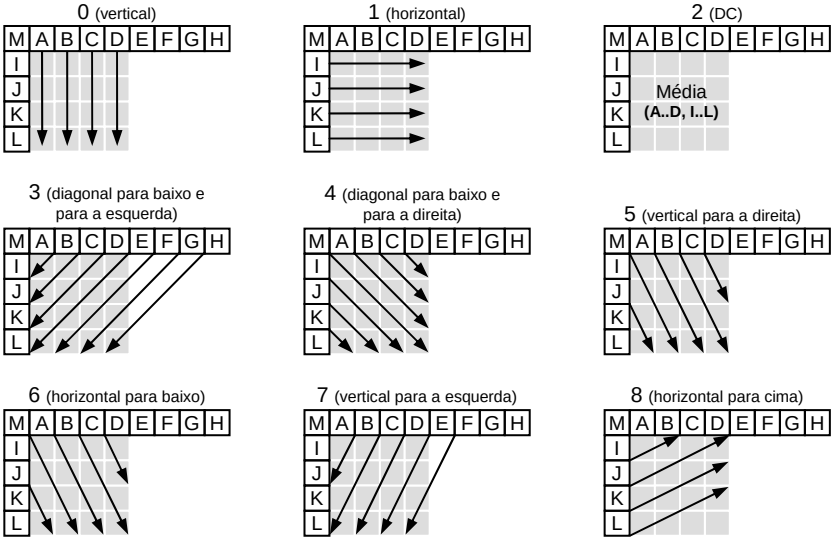

Figura A.15: Nove modos da predição intra para blocos de luma 4x4

Como exemplo, se considerarmos uma predição intra para blocos 4x4 de luminância no modo 4 da Figura A.15, então o cálculo da predição da amostra na posição **d** da Figura A.14 é dado por (6). Por outro lado, o cálculo que gera o valor das posições **a**, **f**, **k** e **p** é dado por (7).

$$d = round\left[(B + 2 \cdot C + D)/4\right] \qquad (6)$$

$$a, f, k, p = round\left[(I + 2M + A)/4\right] \qquad (7)$$

Como já foi mencionado neste texto, a predição intra também pode acontecer sobre o componente de luma completo de um macrobloco, ou seja, com blocos do tamanho 16x16. Neste caso, são quatro os modos possíveis de predição intra, conforme está apresentado na Figura A.16. Os modos 0 e 1 são simples cópias na vertical ou na horizontal das amostras reconstruídas das bordas. O modo 2 é o modo DC e é formado por uma média simples dos elementos das bordas, cujo resultado é copiado para todas as posições do bloco. O modo 3 aplica uma função linear que considera as amostras horizontais e verticais (H e V na Figura A.16) das bordas.

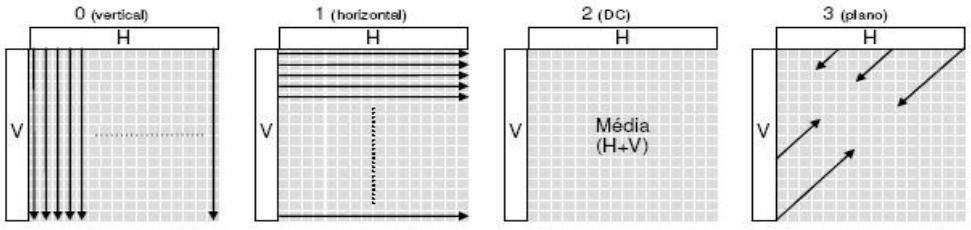

Figura A.16: Quatro modos da predição intra para blocos de luma 16x16

A predição da crominância é realizada diretamente sobre blocos 8x8 e utiliza 4 modos distintos de predição, mas os dois componentes de crominância utilizam sempre o mesmo modo. Os modos de predição para crominância são muito similares aos modos de predição de luminância para blocos 16x16 que foram apresentados na Figura A.16, exceto pela numeração dos modos. Para a predição de crominância, o modo DC é o modo zero, o modo horizontal é o modo um, o modo vertical é o modo dois e o modo plano é o modo três (Richardson, 2003).

Os diferentes modos de predição intra para luminância e crominância possibilitam a geração de uma predição para macroblocos do tipo I que conduz a uma codificação eficiente para este tipo de macrobloco. A escolha de qual modo de predição deve ser escolhido é realizada pelo codificador, que deve sinalizar o modo escolhido no cabeçalho do macrobloco. Para escolher o melhor modo, o codificador deve gerar a predição sobre todos os modos e escolher qual é o melhor do ponto de vista da eficiência de codificação. Esta tarefa possui uma complexidade computacional elevada para o codificador.

O Bloco das Transformadas Diretas (T)

O bloco T, apresentado na Figura A.5, é responsável pelas transformadas diretas e está presente apenas nos codificadores H.264. As entradas para o bloco T são blocos 4x4 de resíduos gerados pela etapa de predição. A DCT 2-D direta (FDCT 2-D) é aplicada a todas as amostras de entrada, tanto de luminância quanto de crominância. O cálculo da FDCT 2-D inteira é definido no padrão H.264 como está apresentado em (8).

$$Y = C_f X C_f^T \otimes E_f = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \begin{bmatrix} & X & \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 2 & -1 \end{bmatrix} \right) \otimes \begin{bmatrix} a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \\ a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \end{bmatrix} \quad (8)$$

Em (8), **X** é a matriz 4x4 de entrada, $\mathbf{C_f}$ é a matriz da FDCT inteira em uma dimensão, $\mathbf{C_f^T}$ é a transposta de da matriz da DCT e $\mathbf{E_f}$ é a matriz de fatores de escala. O símbolo ⊗ na equação indica uma multiplicação escalar. As letras **a** e **b** na matriz $\mathbf{E_f}$ são constantes definidas em (9).

$$a = \frac{1}{2}, \qquad b = \sqrt{\frac{2}{5}} \qquad (9)$$

O cálculo da FDCT 2-D transfere para o bloco de quantização (Q), que é o passo que segue a aplicação das transformadas diretas na compressão H.264, a tarefa de realizar a multiplicação escalar por $\mathbf{E_f}$. Esta tarefa adicional não implica em aumento na complexidade do bloco Q.

Como pode ser observado em (8), as matrizes $\mathbf{C_f}$ e $\mathbf{C_f^T}$ possuem apenas valores 1 e 2, positivos e negativos. Isso implica na realização de operações bastante simples, consistindo de somas e subtrações diretas e multiplicações por dois, que são equivalentes a um deslocamento de uma casa binária para a esquerda.

O cálculo da FDCT 2-D é aplicado sobre todos os dados de entrada, mas, no caso de amostras com informação de crominância ou com informação de luminância cuja predição tenha sido do tipo INTRA 16x16, um cálculo adicional é realizado. Nestes dois casos, é aplicada uma transformada Hadamard 4x4 sob os coeficientes DC dos blocos de luminância, enquanto que para os blocos de crominância, é aplicada uma transformada de Hadamard 2x2 sob os coeficientes DC dos blocos 4 x 4 de crominância. A transformada Hadamard explora uma correlação residual que ainda permaneça sobre os coeficientes da FDCT 2-D (Richardson, 2003).

O cálculo da Hadamard 4x4 direta definida pelo padrão H.264 está apresentado em (10). Esta transformada é aplicada apenas sobre os elementos DC dos blocos 4x4 (resultantes da aplicação da FDCT 2-D) de um macrobloco 16x16 de luminância que tenha utilizado a predição intra 16x16. Então os 16 elementos DC dos 16 blocos do macrobloco irão formar a matriz 4x4 $\mathbf{W_D}$ de entrada para a Hadamard 4x4.

$$Y_D = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} & W_D & \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \right) / 2 \qquad (10)$$

Como pode ser observado em (10), as matrizes possuem apenas valores 1, positivos e negativos. Deste modo, apenas somas e subtrações são necessárias. A divisão por dois realizada em todos os resultados da saída é um simples deslocamento de uma casa binária para a direita.

A transformada Hadamard 2x2 é aplicada apenas para amostras DC de crominância. É utilizada tanto para Cb quanto para Cr. Em função da relação de entrada de 4:2:0 para Y, Cb e Cr, cada macrobloco possui 16x16 amostras de luminância, 8x8 amostras de Cb e 8x8 amostras de Cr. As amostras de crominância passam pela FDCT 2-D em blocos de 4x4 amostras. Então, cada matriz 8x8 de entrada é formada por quatro matrizes 4x4. Os quatro elementos DC das quatro matrizes resultantes da FDCT 2-D passam pela Hadamard 2-D 2x2.

O cálculo da Hadamard 2x2 definido pelo padrão H.264 está apresentado em (11).

$$W_{QD} = \left(\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} & W_D & \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \right) \qquad (11)$$

Pode ser observado na fórmula acima que os cálculos da Hadamard 2x2 são muito simples, consistindo de poucas somas e subtrações.

O bloco T, sendo formado pelas três transformadas que estão apresentadas acima, deve sincronizar a operação entre elas, de modo a gerar o fluxo correto de dados na sua saída. A ordem de processamento está apresentada na Figura A.17. Inicialmente o macrobloco de luminância (**luma** na Figura A.17) passa pela FDCT 2-D. Se o modo é intra 16x16, então, como foi explicado, os DCs das matrizes 4x4 resultantes da FDCT 2-D passam pela Hadamard 2-D 4x4. Neste caso, primeiramente é enviado para saída o bloco **–1** na Figura A.17, com os resultados da Hadamard 2-D 4x4, e depois os elementos AC são enviados para a saída (blocos **0** a **15** na Figura A.17). Se o modo não é intra 16x16, então os blocos **0** a **15** são enviados diretamente para a saída e a Hadamard 4x4 não é aplicada. Seguindo 16 blocos de luminância vêm quatro blocos de crominância Cb e quatro blocos de crominância Cr. Os blocos de crominância passam pela FDCT 2-D e, sob os elementos DC dos resultados, é aplicada a Hadamard 2-D 2x2. Então é enviado para a saída o bloco **16** na Figura A.17, com os resultados da Hadamard 2x2 para o componente Cb, depois é enviado o bloco **17**, com os resultados da Hadamard 2x2 para Cr. Finalmente, os coeficientes AC de crominância são enviados para a saída, primeiro os de Cb (blocos **18** a **21**) e depois os de Cr (blocos **22** a **25**).

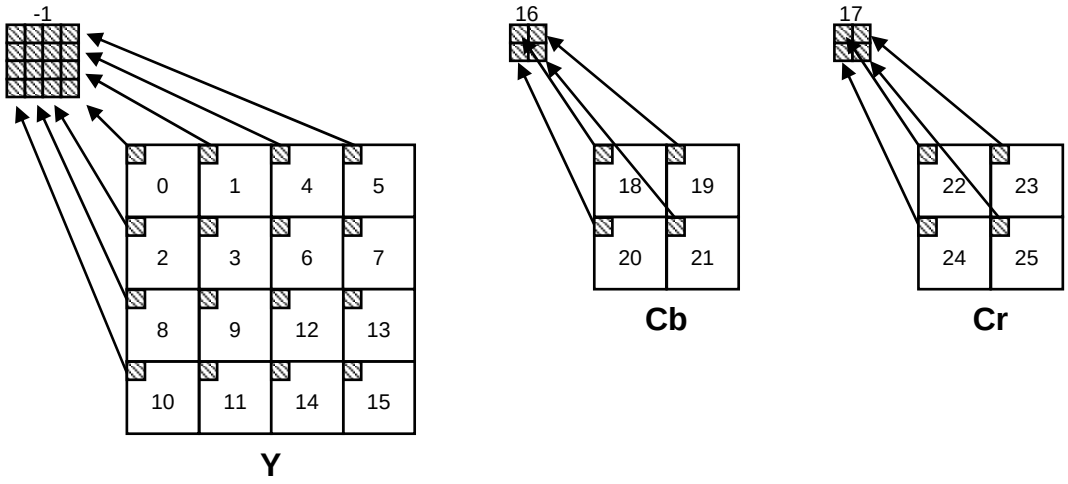

Figura A.17: Ordem de processamento de amostras pelo bloco T

O Bloco da Quantização (Q)

O bloco Q na Figura A.5 realiza a quantização direta e a correção da escala do cálculo das transformadas e está presente apenas no codificador H.264. Dependendo do modo de predição utlizado e se o elemento é de crominância ou luminância, os cálculos realizados pelo bloco Q são modificados mas, genericamente, as operações realizadas são uma multiplicação da entrada por uma constante, a soma do resultado por uma outra constante e um deslocamento no resultado da soma controlado por uma terceira constante. Estas constantes são influenciadas diretamente pelo parâmetro de quantização (QP) que é uma entrada externa que informa ao bloco Q qual é o passo de quantização (Qstep) que deve ser utilizado. QP pode variar de 0 a 51 e para cada QP existe um Qstep. Os primeiros seis valores de Qstep, relativos aos seis primeiros QP, são definidos pelo padrão como está apresentado na Tabela A.4. Os demais Qsteps podem ser derivados dos seis primeiros, pois o Qstep dobra de valor a cada variação de 6 em QP. Então o $Qstep_{(6)}$ é igual $Qstep_{(0)} \times 2$.

Tabela A.4: Relação entre QP e Qstep

QP	0	1	2	3	4	5	6	...	12
Qstep	0,625	0,6875	0,8125	0,875	1	1,125	1,25	...	2,5

Para os elementos de luminância ou crominância AC e para elementos de luminância DC que não tenham sido codificados no modo intra 16x16, ou seja, para os elementos que foram processados apenas pela FDCT 2-D no bloco T, a quantização é definida por (12), já considerando o uso de números em ponto fixo:

$$\begin{aligned} |Z_{(i,j)}| &= (|W_{(i,j)}| \cdot MF + f) >> qbits \\ sign(Z_{(i,j)}) &= sign(W_{(i,j)}) \end{aligned} \qquad (12)$$

Em (12), $\mathbf{W_{ij}}$ é o coeficiente resultante da DCT 2-D, **MF** é uma constante gerada a partir do fator de escala e do parâmetro de quantização (**QP**), **f** é uma constante definida pelo padrão em função da predição ter sido gerada pelo modo inter ou intra e do parâmetro de quantização utilizado. Por fim, **qbits** indica o deslocamento que deve ocorrer na saída antes do cálculo ser finalizado. É importante salientar que o sinal do

resultado deve ser o mesmo sinal da amostra de entrada e que o cálculo é realizado apenas considerando o módulo da amostra de entrada.

A constante **MF** é definida como está apresentado em (13).

$$MF = \frac{PF}{Qstep} >> qbits \qquad (13)$$

Em (13), **PF** é o fator de escala, **Qstep** é o passo de quantização e **qbits** é o mesmo deslocamento que estava apresentado em (12). O fator de escala **PF** depende da posição da amostra no bloco e pode ser $\mathbf{a^2}$, **ab/2** ou $\mathbf{b^2/4}$, sendo que **a** e **b** são as mesmas constantes definidas para o bloco T em (9).

O cálculo de **qbits** é função de **QP** e está apresentado em (14). É importante destacar o arredondamento para baixo do fator **QP/6** que está apresentado em (14).

$$qbits = 15 + \lfloor QP/6 \rfloor \qquad (14)$$

Por fim, a constante **f** é definida em (15).

$$\begin{aligned} f &= 2^{qbits}/3 && \text{se a predição for intra} \\ f &= 2^{qbits}/6 && \text{se a predição for inter} \end{aligned} \qquad (15)$$

A quantização para amostras **DC** de crominância ou para amostras de luminância que foram codificados segundo a predição intra no modo 16x16 é definida por (16).

$$\begin{aligned} |Z_{D(i,j)}| &= (|Y_{D(i,j)}| \cdot MF_{(0,0)} + 2f) >> (qbits + 1) \\ sign(Z_{D(i,j)}) &= sign(Y_{D(i,j)}) \end{aligned} \qquad (16)$$

Este cálculo é bastante similar ao apresentado em (12), sendo que as constantes **MF**, **qbits** e **f** são definidas de modo idêntico ao descrito (13), (14) e (15).

O Bloco das Transformadas Inversas (T^{-1})

O bloco T^{-1} do padrão H.264, apresentado nas Figuras A.5 e A.6, por fazer as operações inversas ao bloco T, possui muitas características idênticas a este bloco. O bloco T^{-1} está presente nos codificadores e nos decodificadores H.264. O curioso é que o padrão H.264 definiu que a operação do bloco T^{-1} seja fracionada em duas partes. A primeira é realizada diretamente sobre os resultados da quantização direta (Q) e envolve as transformadas Hadamard 2x2 e 4x4 inversas. Este resultado é, então, entregue para a quantização inversa (Q^{-1}). Então, os resultados da quantização inversa são entregues novamente para o bloco T^{-1}, que realiza a última etapa de seus cálculos, aplicando a DCT 2-D inversa (IDCT).

A transformada IDCT 2-D está definida em (17), de acordo com o padrão H.264, onde **X** é a matriz 4x4 de entrada, $\mathbf{C_i}$ é a matriz da IDCT inteira em uma dimensão, $\mathbf{C_i^T}$ é a transposta de da matriz da IDCT e $\mathbf{E_i}$ é a matriz de fatores de escala. O símbolo ⊗ na equação indica uma multiplicação escalar. As letras **a** e **b** na matriz $\mathbf{E_i}$ são as mesmas constantes definidas para a DCT 2-D direta.

O cálculo da IDCT 2-D transfere a tarefa de realizar a multiplicação escalar por $\mathbf{E_i}$ (do mesmo modo que para o cálculo da DCT 2-D) para o bloco de quantização inversa (Q^{-1}), que é o passo que antecede a IDCT na compressão H.264. Esta tarefa adicional não implica em aumento na complexidade do bloco Q^{-1}.

$$Y = C_i^T (X \otimes E_i) C_i = \begin{bmatrix} 1 & 1 & 1 & \frac{1}{2} \\ 1 & \frac{1}{2} & -1 & -1 \\ 1 & -\frac{1}{2} & -1 & 1 \\ 1 & -1 & 1 & -\frac{1}{2} \end{bmatrix} \left(\begin{bmatrix} X \end{bmatrix} \otimes \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix} \right) \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \frac{1}{2} & -\frac{1}{2} & -1 \\ 1 & -1 & -1 & 1 \\ \frac{1}{2} & -1 & 1 & -\frac{1}{2} \end{bmatrix} \quad (17)$$

Como pode ser observado em (17), as matrizes $\mathbf{C_i^T}$ e $\mathbf{C_i}$ possuem apenas valores 1 e ½, positivos e negativos. Isso implica na realização de operações de somas e subtrações e um deslocamento de uma casa para a direita, no caso do cálculo utilizar o valor ½.

O cálculo da IDCT 2-D é aplicado sobre todos os dados que são entregues pela quantização inversa (Q^{-1}). Para amostras com informação de crominância ou com informação de luminância cuja predição tenha sido do tipo INTRA 16x16 é aplicada a Hadamard 4x4 inversa nos coeficientes DC dos blocos de luminância, enquanto que, para os blocos de crominância, é aplicada a Hadamard 2x2 inversa nos coeficientes DC dos blocos 4 x 4 de crominância.

O cálculo da Hadamard 4x4 inversa está apresentado em (18).

$$W_{QD} = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} Z_D \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \right) \quad (18)$$

Como pode ser observado em (18), o cálculo da Hadamard 4x4 inversa é muito parecido com o cálculo da Hadamard 4x4 direta que foi apresentada em (10), na descrição do bloco T. As matrizes possuem apenas valores 1, positivos e negativos, como na Hadamard 4x4 direta. Isso implica na realização apenas de operações de somas e subtrações. A diferença entre a Hadamard 4x4 direta e inversa está na divisão por dois, que não existe na Hadamard 4x4 inversa.

A transformada Hadamard 2x2 inversa é aplicada apenas para amostras DC de crominância. É utilizada tanto para Cb quanto para Cr. O cálculo da Hadamard 2x2 inversa é idêntico ao cálculo da Hadamard 2x2 direta, que foi apresentado em (11).

O bloco T^{-1}, sendo formado pelas três transformadas inversas que estão apresentadas acima, deve sincronizar a operação entre elas, de modo a gerar o fluxo correto de dados na sua saída. A ordem de processamento é exatamente a mesma apresentada na descrição do bloco T.

O Bloco da Quantização Inversa (Q^{-1})

O bloco Q^{-1}, apresentado nas Figuras A.5 e A.6, está presente nos codificadores e nos decodificadores H.264. Este bloco realiza a quantização inversa e a correção da escala do cálculo das transformadas. Alguns autores defendem que não é correto chamar este bloco de quantização inversa, uma vez que a quantização direta é uma operação irreversível por ser uma divisão inteira (Bhaskaran, 1999). Deste modo, estes autores utilizam o termo "*rescaling*" ao invés do termo quantização inversa. Neste trabalho, adotaremos o termo quantização inversa.

Por uma definição do padrão, antes da quantização inversa são realizados partes dos cálculos relativos ao bloco T^{-1}. As tranformadas inversas de Hadamard 4x4 e 2x2, quando utilizadas (DCs de croma ou DCs de luma para o modo intra 16x16), são aplicadas antes da quantização inversa.

Dependendo do modo de predição utlizado e se o elemento é de crominância ou luminância, os cálculos realizados pelo bloco Q^{-1} são modificados, do mesmo modo que ocorre no bloco Q. Também no bloco Q^{-1} as operações realizadas são uma multiplicação da entrada por uma constante, a soma do resultado por uma outra constante e um deslocamento no resultado da soma controlado por uma terceira constante.

Como na quantização direta, estas constantes são influenciadas diretamente pelo parâmetro de quantização (QP) que é uma entrada externa que informa ao bloco Q qual é o passo de quantização (Qstep) que deve ser utilizado. Os Qsteps utilizados na quantização inversa são idênticos aos usados na quantização direta.

Para os elementos de luminância ou crominância AC e para elementos de luminância DC que não tenham sido codificados no modo intra 16x16, ou seja, para os elementos que foram processados apenas pela DCT 2-D no bloco T, a quantização inversa é definida por (19).

$$W'_{(i,j)} = Z_{(i,j)} \cdot V_{(i,j)} \cdot 2^{\lfloor QP/6 \rfloor} \qquad (19)$$

Em (19), $\mathbf{Z_{(i,j)}}$ é o coeficiente quantizado, $\mathbf{V_{(i,j)}}$ é uma constante gerada a partir do fator de escala ($\mathbf{PF_{(i,j)}}$) e do parâmetro de quantização (**QP**) e $2^{\lfloor QP/6 \rfloor}$ é um deslocamento definido por QP e que deve ocorrer na saída antes do cálculo ser finalizado.

A constante $\mathbf{V_{(i,j)}}$ está definida em (20), onde **Qstep** é o passo de quantização e $\mathbf{PF_{(i,j)}}$ é o fator de escala.

$$V_{(i,j)} = (Qstep \cdot PF_{(i,j)} \cdot 64) \qquad (20)$$

O fator de escala $\mathbf{PF_{(i,j)}}$ é diferente na quantização inversa em relação à quantização direta, mas também depende da posição da amostra no bloco. Na quantização inversa $\mathbf{PF_{(i,j)}}$ pode assumir os valores $\mathbf{a^2}$, **ab** ou $\mathbf{b^2}$ sendo que **a** e **b** são as mesmas constantes definidas para o bloco T em (9). A multiplicação por 64, que é um deslocamento de seis casas binárias para a esquerda, é utilizada para prevenir erros de arrendodamento.

A quantização inversa para elementos **DC** de luminância que foram codificados segundo a predição intra no modo 16x16 é definida em (21).

$$\begin{aligned} W'_{D(i,j)} &= W_{QD(i,j)} \cdot V_{(0,0)} \cdot 2^{floor(QP/6)} && (QP \geq 12) \\ W'_{D(i,j)} &= \left[W_{QD(i,j)} \cdot V_{(0,0)} \cdot 2^{1-floor(QP/6)}\right] >> \lfloor QP/6 \rfloor && (QP < 12) \end{aligned} \qquad (21)$$

Este cálculo é bastante similar aos anteriores, sendo que a constante $\mathbf{V_{(i,j)}}$ é definida de modo idêntico ao citado acima.

A quantização para elementos **DC** de crominância é definida por (22).

$$\begin{aligned} W'_{D(i,j)} &= W_{QD(i,j)} \cdot V_{(0,0)} \cdot 2^{\lfloor QP/6 \rfloor - 1} && (QP \geq 6) \\ W'_{D(i,j)} &= \left[W_{QD(i,j)} \cdot V_{(0,0)}\right] >> 1 && (QP < 6) \end{aligned} \qquad (22)$$

O Filtro Redutor do Efeito de Bloco

O bloco Filtro nas Figuras A.5 e A.6 são relativos ao filtro redutor de efeito de bloco que é normatizado pelo padrão H.264 e que está presente nos codificadores e nos decodificadores que seguem este padrão.

O objetivo deste filtro é suavizar o efeito de blocos do quadro reconstruído antes de ele ser usado para fazer a predição de um novo macrobloco do tipo inter. Este filtro é adaptativo, e distingue uma aresta real da imagem, que não deve ser filtrada, e um artefato gerado por um elevado passo de quantização, que deve ser filtrado.

A filtragem é aplicada para bordas verticais e horizontais dos blocos 4x4 de um macrobloco, de acordo com a ordem abaixo. Para os passos apresentados abaixo, considere a Figura A.18 como referência.

1) Filtrar as quatro bordas verticais do componente de luminância (**a**, **b**, **c** e **d** na Figura A.18);
2) Filtrar as quatro bordas horizontais do componente de luminância (**e**, **f**, **g** e **h** na Figura A.18);
3) Filtrar as duas bordas verticais de cada componente de crominância (**i** e **j** na Figura A.18);
4) Filtrar as duas bordas horizontais de cada componente de crominância (**k** e **l** na Figura A.18).

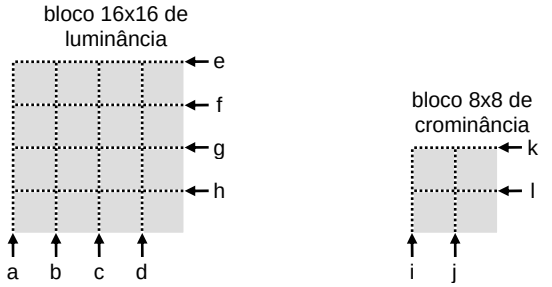

Figura A.18: Ordem de filtragem de bordas em um macrobloco

Cada operação de filtragem afeta até três amostras de cada lado da borda, como está apresentado na Figura A.19. Neste caso, são considerados dois blocos 4x4 adjacentes chamados de **p** e **q**, e apenas quatro amostras de cada bloco estão presentes em cada exemplo da Figura A.19.

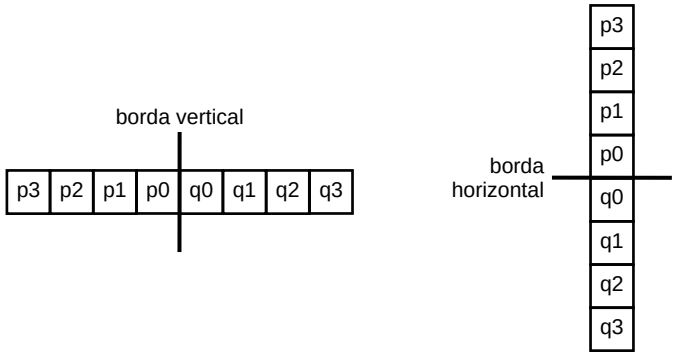

Figura A.19: Amostras adjacentes para bordas verticais e horizontais

A "força" de filtragem a ser realizada depende da quantização utilizada no bloco, do modo de codificação dos blocos vizinhos e do gradiente das amostras da imagem

através da borda. A aplicação do filtro proporciona um aumento significativo da qualidade subjetiva do vídeo reconstruído.

Existem cinco diferentes forças de filtragem, sendo definidos pelo parâmetro ***b*S** (*boundary strength*). O parâmetro ***b*S** pode variar de zero a quatro e, portanto, existem cinco diferentes forças de filtragem, onde um ***b*S**=4 define uma filtragem com força máxima e um ***b*S**=0 define que nenhuma filtragem é realizada. A definição de qual tipo de filtragem deve ser realizada segue as seguintes regras, considerando uma borda entre os blocos **q** e **p**:

- ***b*S=4** : é utilizado se os blocos **q** e/ou **p** foram codificados no modo intra e se a borda é uma borda de macrobloco.
- ***b*S=3** : é utilizado se os blocos **q** e/ou **p** foram codificados no modo intra e se a borda não é uma borda de macrobloco.
- ***b*S=2** : é utilizado se os blocos **q** e/ou **p** não foram codificados no modo intra e se **p** e **q** possuem coeficientes codificados.
- ***b*S=1** : é utilizado se os blocos **q** e/ou **p** não foram codificados no modo intra; se **p** e **q** não possuem coeficientes codificados; se **p** e **q** usam quadros de referência diferentes ou diferentes números de quadros de referência ou possuem valores de vetores de movimento que se diferenciam por uma ou mais amostras de luminância;
- ***b*S=0** : é utilizado se nenhuma das situações anteriores for verdadeira.

Mas a decisão de realizar a filtragem não depende apenas do parâmetro ***b*S**. Considerando um grupo de amostras (p2, p1, p0, q0, q1, q2), a filtragem acontece se *b*S>0 e se a condição apresentada em (23) for verdadeira.

$$|p0 - q0| < \alpha \quad \text{e} \quad |p1 - p0| < \beta \quad \text{e} \quad |q1 - q0| \leq \beta \qquad (23)$$

Em (23), α e β são definidos pelo padrão e crescem de acordo com a média do parâmetro de quantização (QP) dos blocos **p** e **q**. Se o QP tiver um valor baixo, então α e β também terão valores baixos. Neste caso, o efeito de bloco gerado tem importância menor e, por isso, o filtro permanecerá inativo durante mais tempo. Por outro lado, se QP possuir um valor elevado, então a quantização gerará mais perdas, gerando efeitos de blocos mais significativos. Neste caso, os valores de α e β serão maiores, fazendo com que o filtro fique mais tempo ativo e assim, mas amostras da borda são filtradas.

O Bloco de Codificação de Entropia

Na codificação de entropia, que está presente nos codificadores e decodificadores (Figuras A.5 e A.6) o H.264 introduz algumas ferramentas que aumentam bastante a sua eficiência de codificação. Nos níveis hierárquicos superiores (quadros, etc.), os elementos sintáticos são codificados usando códigos binários fixos ou de comprimento variável. A partir do nível de *slices* ou abaixo (macroblocos, blocos, etc.), os elementos sintáticos são codificados usando a codificação aritmética adaptativa ao contexto (CABAC) (Richardson, 2003) ou códigos de comprimento variável (VLC) (Salomon, 2000). No caso do uso de VLC, a informação residual (coeficientes quantizados das transformadas) é codificada usando a codificação de comprimento variável adaptativa ao contexto (CAVLC) (Richardson, 2003), enquanto que as demais unidades de

codificação são codificadas usando códigos Exp-Golomb (Salomon, 2000). A opção pelo CABAC não está disponível nos perfis baseline e extended.

A principal inovação introduzida no H.264, tanto no VLC quanto no CABAC é o uso de codificação adaptativa baseada em contextos. Nela, a maneira com que são codificados os diversos elementos sintáticos depende do elemento a ser codificado, da fase em que se encontra o algoritmo de codificação e dos elementos sintáticos que já foram codificados.

Os resíduos resultantes da quantização devem ser reordenados antes de serem utilizados pela codificação de entropia. Os blocos 4x4 com coeficientes da transformada, após a quantização, são reordenados em forma de ziguezague, como está apresentado na Figura A.20. Se a predição intra 16x16 foi utilizada, então os coeficientes DC de cada bloco 4x4 de luminância são extraídos do bloco 16x16, formando uma matriz 4x4 de coeficientes DC. Esta matriz é, então, reordenada do mesmo modo que está apresentado na Figura A.20. Os 15 coeficientes AC que restam em cada bloco 4x4 de luminância são, então, reorganizados do mesmo modo que está apresentado na Figura A.20, mas iniciando da segunda posição. De forma similar, os coeficientes DC dos blocos 4x4 dos dois componentes de crominância são extraídos das matrizes 8x8 de amostras de cor, formando duas matrizes 2x2 de elementos DC, que também são organizadas em ziguezague. Os 15 coeficientes AC restante nos blocos 4x4 de crominância são, então, reorganizados seguindo a ordem da Figura A.20, mas iniciando da segunda posição.

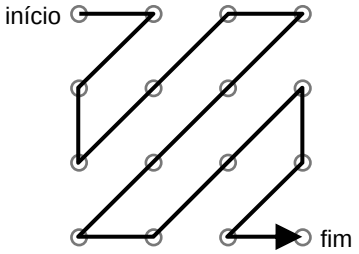

Figura A.20: Ordem de leitura dos blocos 4x4 para a codificação de entropia

As próximas subseções do texto irão apresentar resumidamente, a funcionalidade dos códigos de Exp-Golomb, a codificação de comprimento variável adaptativa ao contexto e a codificação aritmética binária adaptativa ao contexto, que são ferramentas de codificação utilizadas no codificador de entropia do padrão H.264, como já foi mencionado acima.

Códigos Exp-Golomb

Os códigos Exp-Golomb (Exponencial Golomb) (Salomon, 2000), são códigos de comprimento variável com uma construção regular. Um número inteiro não negativo *N* é codificado usando a estrutura apresentada em (24):

$$Código = [\ M\ zeros\ ][\ 1\ ][\ INFO\ ] \qquad (24)$$

Em (24), **M** representa o número de zeros que antecedem o primeiro valor 1 no código. O valor de **M** é dado por (25)

$$M = \lfloor \log_2 \cdot (num\_cod + 1) \rfloor \qquad (25)$$

A variável **num_cod** em (24) indica qual é o número do código. Códigos com maior probabilidade de ocorrência possuem um número de código mais baixo.

O campo INFO em (25) possui **M** bits e indica qual é a informação codificada. O valor do campo INFO é dado por (26).

$$INFO = num\_cod + 1 - 2^{M} \qquad (26)$$

O primeiro código de Exp-Golomb não possui os campos **M zeros** e **INFO**, sendo sempre representado apenas pelo valor '1'. A Tabela A.5 apresenta os seis primeiros códigos de Exp-Golomb.

Tabela A.5: Seis primeiros códigos de Exp-Golomb

num_cod	*Código*
0	1
1	010
2	011
3	00100
4	00101
5	00110
. . .	. . .

No padrão H.264, para cada elemento sintático **k** a ser codificado com o código Exp-Golomb há uma regra que mapeia o valor de **k** para valores inteiros não negativos, isto é, a regra indica como o valor do elemento sintático **k** deve ser mapeado para um valor de **num_cod**. São quatro as possibilidades de mapeamento, dependendo do tipo de elemento sintático codificado.

Codificação de Comprimento Variável Adaptativa ao Contexto – CAVLC

A codificação de comprimento variável adaptativa ao contexto (CAVLC) é utilizada para codificar os resíduos resultantes da quantização, já ordenados em ziguezague. A CAVLC produz códigos de comprimento variável que são dependentes do contexto da codificação, isto é, da fase em que o algoritmo de codificação se encontra e dos valores que já foram codificados. A CAVLC foi desenvolvida para explorar as características dos blocos quantizados, quais sejam (Richardson, 2003):

- O resultado da quantização produz, tipicamente, matrizes esparsas. Então a CAVLC utiliza RLE (Salomon, 2000) para representar compactamente as seqüências de zeros.
- Os coeficientes não zeros de alta freqüência, depois da leitura em ziguezague, são frequentemente seqüências de ±1. Então a CAVLC sinaliza o número dos coeficientes ±1 de alta freqüência de uma forma compacta.
- O número de coeficientes não zero em blocos vizinhos são correlacionados. O número de coeficientes é codificado usando uma tabela e a escolha de qual valor da tabela deve ser usado depende do número de coeficiente não zero nos blocos vizinhos.

- O nível (magnitude) dos coeficientes não zero tende a ser maior para os primeiros coeficientes lidos em ziguezague (baixas freqüências) e menores para as freqüências mais elevadas. Então o CAVLC tira vantagem desta característica adaptando a escolha da tabela de VLC que será utilizada para o parâmetro de nível dependendo dos níveis de magnitude dos coeficientes recentemente codificados.

A CAVLC esta disponível para todos os perfis do padrão H.264 e é uma alternativa menos complexa ao codificador aritmético adaptativo ao contexto (CABAC), que está disponível nos perfis main e high. A escolha pelo CAVLC ao invés do CABAC conduz a uma implementação de menor complexidade, mas gera um impacto negativo na eficiência de codificação.

Codificação Aritmética Binária Adaptativa ao Contexto (CABAC)

A codificação aritmética binária adaptativa ao contexto (CABAC) é uma ferramenta de codificação disponível apenas para os perfis main e high. A CABAC atinge elevadas taxas de compressão através da seleção dos modelos de probabilidade para cada elemento sintático de acordo com o contexto deste elemento, então as estimativas de probabilidade são adaptadas com base nas estatísticas locais e, finalmente, a codificação aritmética é usada para codificar o elemento sintático.

A codificação pelo CABAC envolve os seguintes estágios:

1. **Binarização**: O CABAC utiliza codificação aritmética binária, o que significa que apenas decisões binárias (0 ou 1) são codificadas. Então os símbolos que não possuem valores binários são binarizados ou convertidos em um código binário antes a codificação. Esse processo é similar ao de converter um símbolo de dados em um código de comprimento variável (VLC), mas o código binário é codificado adicionalmente pelo codificador aritmético antes de ser transmitido. Cada posição de um dígito binário é chamada de um **bin**. Os passos 2, 3, e 4 são repetidos para todos **bins**.
2. **Seleção dos modelos probabilísticos:** Um modelo de contexto é um modelo probabilístico para um ou mais **bins** do símbolo binarizado. A escolha do modelo de contexto a ser utilizado depende dos modelos disponíveis e das estatísticas dos símbolos recentemente codificados. O modelo de contexto armazena a probabilidade de cada **bin** ser 0 ou 1. No H.264 são usados 398 contextos diferentes (ITU-T, 2005b; Marpe, 2003).
3. **Codificação aritmética**: Um codificador aritmético codifica cada **bin** de acordo com o modelo probabilístico selecionado. A codificação aritmética será apresentada com mais detalhes no decorrer desta seção, mas é importante notar que, de acordo com os princípios da codificação aritmética, existem somente duas sub-faixas possíveis para cada **bin**, correspondentes a ‘0’ e ‘1’.
4. **Atualização de probabilidades:** O modelo de contexto selecionado é atualizado com base no valor atual codificado. Por exemplo, se o valor do **bin** atual é ‘0’, a contagem de ocorrências de zeros é incrementada.

Como foi apresentado no item 3 acima, a operação do CABAC utiliza a codificação aritmética para codificar os **bins**. Na codificação aritmética, uma mensagem é representada por um intervalo contido entre ‘0’ e ‘1’. Considerando que a tabela de

probabilidades de ocorrência dos símbolos já tenha sido construída, a codificação aritmética segue os seguintes passos:

1. Definir a faixa inicial de probabilidades contendo todos os símbolos. Para *N* símbolos o intervalo é dividido em *N* subintervalos tal que o intervalo correspondente a um símbolo qualquer possui largura igual à sua probabilidade;
2. Ler o próximo símbolo da entrada;
3. Encontrar a sub-faixa do símbolo atual;
4. Expandir esta sub-faixa para que seja usada como nova faixa, entre '0' e '1';
5. Repetir os passos 2 a 4 até que os o valor de um dígito pára de variar (intervalo pequeno o suficiente);
6. Transmitir algum valor que esteja na faixa de valores do último símbolo codificado.

A Figura A.21 apresenta um exemplo de codificação aritmética, considerando a seqüência de símbolos (0, -1, 0, 2). A tabela de probabilidades para este exemplo está apresentada na Tabela A.6. Na Figura A.21, os números maiores sem parênteses indicam a divisão da faixa inicial. Os números grandes entre parênteses indicam qual é o símbolo atual que está sendo codificado. Os números menores indicam a distribuição de probabilidades para cada passo da codificação.

Tabela A.6: Tabela de probabilidades e sub-faixas para os símbolos do exemplo

Símbolo	*Probabilidade*	*Sub-faixa*
-2	0,1	0 – 0,1
-1	0,2	0,1 – 0,3
0	0,4	0,3 – 0,7
1	0,2	0,7 – 0,9
2	0,1	0,9 – 1

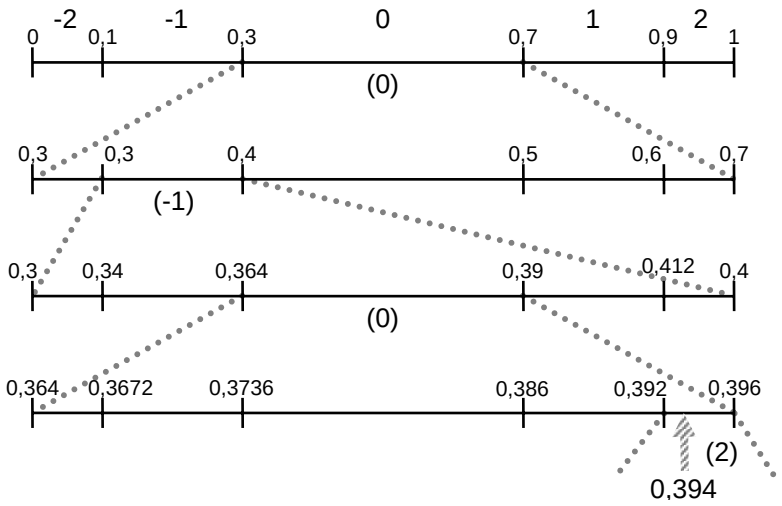

Figura A.21: Exemplo de codificação aritmética

Para o exemplo da Figura A.21, o intervalo final ficou entre 0,3928 e 0,396 e o valor 0,394 foi o escolhido para ser transmitido e representar o conjunto de valores (0, -1, 0, 2). Com este valor e com o conhecimento dos dados apresentados na Tabela A.6, o decodificador será capaz de reconstruir os símbolos.

É importante destacar que o modelo probabilístico usado é independente do codificador aritmético. Isto faz com que possam ser usados modelos probabilísticos diferentes dependendo do estado em que o codificador se encontre. Por exemplo, as probabilidades dos símbolos em cada modelo podem ser atualizadas à medida que os símbolos vão sendo codificados/decodificados.

A codificação/decodificação aritmética requer operações aritméticas para gerar as faixas, o que faz com que a codificação aritmética seja bem mais complexa computacionalmente do que os códigos de comprimento variável. Esta complexidade elevada deve ser tratada tanto pelo codificador quanto pelo decodificador H.264 e é um custo adicional associado ao uso do CABAC. Por outro lado, os ganhos em eficiência de codificação com a utilização do CABAC são significativos. Novamente o compromisso entre complexidade e eficiência de codificação deve ser avaliado para a tomada de decisão de qual tipo de codificador será utilizado na codificação de entropia dos codecs H.264.

Controle do Codificador

A especificação do padrão H.264 define apenas a sintaxe do *bitstream* e o processo de decodificação de um vídeo. O processo de codificação é deixado de fora do escopo do padrão para permitir maior flexibilidade para as implementações. Mas o controle do codificador é um problema chave para a compressão de vídeo, pois ele determina quais as decisões de codificação serão tomadas para cada vídeo processado (Wiegand, 2003).

No codificador H.264 existem muitas decisões que devem ser tomadas pelo codificador, incluindo a escolha do tipo de predição (intra 4x4, inter 4x4, intra 16x16, inter 4x8, etc.), a escolha do melhor bloco na estimação de movimento, a escolha do tamanho de bloco na ME e a escolha das imagens de referência na ME. Como são muitas as possibilidades, é muito importante que as decisões tomadas sejam as melhores possíveis, pois com escolhas sub-ótimas, alguns dos benefícios trazidos pelo H.264 podem ser perdidos. (Puri, 2004).

O problema do controle do codificador é causado porque seqüências de vídeo típicas contêm grande variação de conteúdo e movimento, sendo necessário selecionar entre diferentes opções de codificação com variação na eficiência da relação taxa-distorção para diferentes partes do vídeo. A tarefa do controle do codificador é determinar um grupo de parâmetros de codificação de modo que um certo compromisso na relação taxa-distorção seja atingida (Wiegand, 2003).

Mesmo que não sejam normatizadas pelo padrão H.264, técnicas de otimização taxa-distorção (Wiegand, 2003; Sullivan, 1998) são usadas para todas as decisões tomadas pelo codificador. Com este tipo de controle do codificador, significativos ganhos em termos de eficiência de codificação podem ser obtidos, mas o uso deste tipo de critério não evita que todas as possíveis combinações de modos de codificação tenham que ser calculadas para que se possa atingir a escolha ótima. Entretanto, ao se fazer a escolha que minimiza o custo **J** de cada decisão, esta escolha vai tender a maximizar a eficiência do codificador.

O parâmetro λ é usado para controlar a taxa obtida e está relacionado com o passo de quantização **Qstep**. É importante que sejam avaliados os compromissos entre a complexidade computacional e a eficiência de codificação introduzidos por esta técnica.

Para cada decisão, é medido o custo **J**, de acordo com (27).

$$J = D + \lambda \cdot R + \eta \cdot C \qquad (27)$$

Em (27), **R** é a taxa de bits, **D** é a distorção e **C** é a complexidade. Neste caso, para λ e η fixos, é possível gerar vários custos **J**. O plano (λ e η) pode ser varrido na busca do custo **J** que estejam mais próximos da taxa **R** ou da distorção **D** desejados.

APÊNDICE B PROJETO DO MÓDULO DE PREDIÇÃO INTRA 4X4

A predição intraquadro pode ser dividida em três grandes grupos: a predição de amostras de luminância agrupadas em blocos de 4x4 pixels, a predição de amostras de luminância agrupadas em macroblocos de 16x16 pixels, e a predição de amostras de crominância agrupadas em macroblocos de 8x8 pixels. As duas últimas são bastante semelhantes, envolvendo basicamente as mesmas operações matemáticas. Neste apêndice, será descrito passo-a-passo o projeto do módulo de predição 4x4.

Este conjunto de modos de predição, como o próprio nome diz, é aplicado sobre áreas de um quadro de 4 pixels de altura por 4 pixels de altura, isto é, envolve ao todo 16 amostras. São nove modos no total, indicados pelos vetores da Figura B.1. Os vetores indicam de onde vêm as amostras de referência, ou seja, o sentido no qual se pode dizer que ocorrem as semelhanças na imagem.

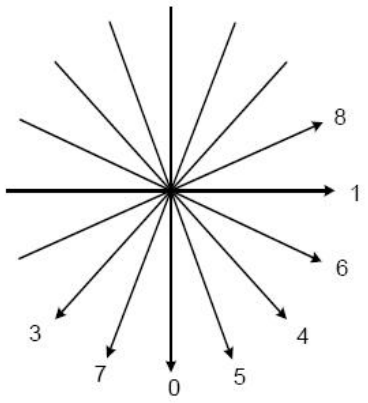

Figura B.1: Vetores indicando o sentido no qual ocorrem as semelhanças em cada um dos modos de predição 4x4

As operações matemáticas envolvidas em cada modo foram desenvolvidas estatisticamente e já estão completamente definidas. Na prática, há uma fórmula para cada modo, para cada um dos 16 pixels do bloco 4x4 em questão, baseados nos vizinhos. A Figura B.2 indica quais vizinhos podem ser utilizados como referência para determinado bloco, e apresenta a nomenclatura para indicar sua posição. O bloco é formado pelas amostras de “a” a “p”, e os vizinhos utilizados como referência estão representados de “A” a “M”. As amostras possuem coordenadas (x,y), conforme demonstrado.

M	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**
I	*a*	*b*	*c*	*d*				
J	*e*	*f*	*g*	*h*				
L	*i*	*j*	*k*	*l*				
M	*m*	*n*	*o*	*p*				

-1,-1	**0,-1**	**1,-1**	**2,-1**	**3,-1**	**4,-1**	**5,-1**	**6,-1**	**7,-1**
-1,0	***0,0***	***1,0***	***2,0***	***3,0***				
-1,1	***0,1***	***1,1***	***2,1***	***3,1***				
-1,2	***0,2***	***1,2***	***2,2***	***3,2***				
-1,3	***0,3***	***1,3***	***2,3***	***3,3***				

Figura B.2: Posição de cada uma das amostras em um bloco 4x4, e seus respectivos vizinhos a serem usados como referência

O codificador vai realizar todos os modos, e vai verificar qual gerou o menor erro (diferença) para o bloco inteiro. O decodificador, a partir da indicação do modo utilizado, é capaz de reconstruir as amostras do bloco, a predição propriamente dita, realizando as mesmas operações sobre os vizinhos. A sub-seção seguinte apresenta a descrição matemática de todos modos de predição intra 4x4.

Modos de Predição Intra 4x4

Modo 0 – Vertical

Se p(x,-1), x ∈ {0,...,3} estiverem disponíveis, $p_{PRED}(x,y) = p(x,-1)$.

Modo 1 – Horizontal

Se p(-1,y), y ∈ {0,...,3} estiverem disponíveis, $p_{PRED}(x,y) = p(-1,y)$.

Modo 2 – DC

1. Se p(x,-1), x ∈ {0,...,3} e p(-1,y), y ∈ {0,...,3} estiverem disponíveis,
$$p_{PRED}(x,y) = \frac{\sum_{y=0}^{3} p(-1,y) + \sum_{x=0}^{3} p(x,-1) + 4}{8}.$$
2. Se apenas p(x,-1), x ∈ {0,...,3} estiverem disponíveis,
$$p_{PRED}(x,y) = \frac{\sum_{x=0}^{3} p(x,-1) + 2}{4}.$$
3. Se apenas p(-1,y), y ∈ {0,...,3} estiverem disponíveis,
$$p_{PRED}(x,y) = \frac{\sum_{y=0}^{3} p(-1,y) + 2}{4}.$$
4. Se alguma das amostras de p(x,-1), x ∈ {0,...,3} não estiver disponível e o mesmo ocorrer para as amostras de p(-1,y), y ∈ {0,...,3}, então $p_{PRED}(x,y) = 2^{\text{profundidade de bits} - 1}$. Por exemplo, se as amostras são representadas em 8 bits, o valor atribuído será 128.

Modo 3 – Diagonal Down-Left

Se p(x,-1), x ∈ {0,...,7} estiverem disponíveis, e

1. Se x=y=3, então $p_{PRED}(x,y) = \frac{p(6,-1) + 3 \cdot p(7,-1) + 2}{4}$.
2. Senão $p_{PRED}(x,y) = \frac{p(x+y,-1) + 2 \cdot p(x+y+1,-1) + p(x+y+2,-1) + 2}{4}$.

Modo 4 – Diagonal Down-Right

Se p(x,-1), x ∈ {0,...,3} e p(-1,y), y ∈ {-1,...,3} estiverem disponíveis, e

1. Se x>y, $p_{PRED}(x,y) = \frac{p(x-y-2,-1) + 2 \cdot p(x-y-1,-1) + p(x-y,-1) + 2}{4}$.
2. Se x<y, $p_{PRED}(x,y) = \frac{p(-1,y-x-2) + 2 \cdot p(-1,y-x-1) + p(-1,y-x) + 2}{4}$.
3. Se x=y, $p_{PRED}(x,y) = \frac{p(0,-1) + 2 \cdot p(-1,-1) + p(0,-1) + 2}{4}$.

Modo 5 – Vertical-Right

Se p(x,-1), x ∈ {0,...,3} e p(-1,y), y ∈ {-1,...,3} estiverem disponíveis, e

1. Se 2x–y ={0,2,4,6}, $p_{PRED}(x,y) = \frac{p(x-y/2-1,-1) + p(x-y/2,-1) + 1}{2}$.
2. Se 2x–y ={1,3,5},
   $p_{PRED}(x,y) = \frac{p(x-y/2-2,-1) + 2 \cdot p(x-y/2-1,-1) + p(x-y/2,-1) + 2}{4}$.
3. Se 2x–y =-1, $p_{PRED}(x,y) = \frac{p(0,-1) + 2 \cdot p(-1,-1) + p(-1,0) + 2}{4}$.
4. Se 2x–y ={-2,-3}, $p_{PRED}(x,y) = \frac{p(-1,y-1) + 2 \cdot p(-1,y-2) + p(-1,y-3) + 2}{4}$.

Modo 6 – Horizontal-Down

Se p(x,-1), x ∈ {0,...,3} e p(-1,y), y ∈ {-1,...,3} estiverem disponíveis, e

1. Se 2y–x ={0,2,4,6}, $p_{PRED}(x,y) = \frac{p(-1,y-x/2-1) + p(-1,y-x/2) + 1}{2}$.
2. Se 2y–x ={1,3,5},
   $p_{PRED}(x,y) = \frac{p(-1,y-x/2-2) + 2 \cdot p(-1,y-x/2-1) + p(-1,y-x/2) + 2}{4}$.
3. Se 2y–x =-1, $p_{PRED}(x,y) = \frac{p(0,-1) + 2 \cdot p(-1,-1) + p(-1,0) + 2}{4}$.
4. Se 2y–x ={-2,-3}, $p_{PRED}(x,y) = \frac{p(x-1,-1) + 2 \cdot p(x-2,-1) + p(x-3,-1) + 2}{4}$.

Modo 7 – Vertical-Left

Se p(x,-1), x ∈ {0,...,7} estiverem disponíveis, e

1. Se y={0,2}, $p_{PRED}(x,y) = \frac{p(x+y/2,-1) + p(x+y/2+1,-1)+1}{2}$.
2. Se y={1,3},
   $p_{PRED}(x,y) = \frac{p(x+y/2,-1) + 2\cdot p(x+y/2+1,-1) + p(x+y/2+2,-1)+2}{4}$.

Modo 8 – Horizontal-Up

Se p(-1,y), y ∈ {0,...,3} estiverem disponíveis, e

1. Se x+2y ={0,2,4}, $p_{PRED}(x,y) = \frac{p(-1,y+x/2) + p(-1,y+x/2+1)+1}{2}$.
2. Se x+2y ={1,3},
   $p_{PRED}(x,y) = \frac{p(-1,y+x/2) + 2\cdot p(-1,y+x/2+1) + p(-1,y-x/2+2)+2}{4}$.
3. Se x+2y =5, $p_{PRED}(x,y) = \frac{p(-1,2) + 3\cdot p(-1,3)+2}{4}$.
4. Se x+2y >5, $p_{PRED}(x,y) = p(-1,3)$.

Desenvolvimento de Um Modelo Matlab da Predição Intra 4x4

A partir das fórmulas descritas na subseção anterior, que foram retiradas da norma que descreve o padrão, foi possível o desenvolvimento de um modelo da predição intra 4x4. Este modelo permite a codificação e a decodificação de quadros de teste, possibilitando a compreensão do algoritmo e um estudo que permite a elaboração de uma arquitetura de *hardware* capaz de desempenhar esta função.

Com o código Matlab desenvolvido foi possível, por exemplo, reconhecer os padrões de direção por onde ocorrem as similaridades, como mostrado na Figura B.3. Aplicando-se o algoritmo a uma imagem teste (“Lenna.jpg”, ver Figura B.4), obtém-se uma aproximação da mesma, ou seja, a imagem predita. Note que para a imagem resgatar a imagem original é enviado um valor para cada bloco de 16 pixels (4x4) no lugar de um valor para cada pixel, o que produz o ganho da codificação. Os resíduos são bastante comprimidos pela transformada e pela codificação de entropia, já que são em sua maioria valores muito baixos.

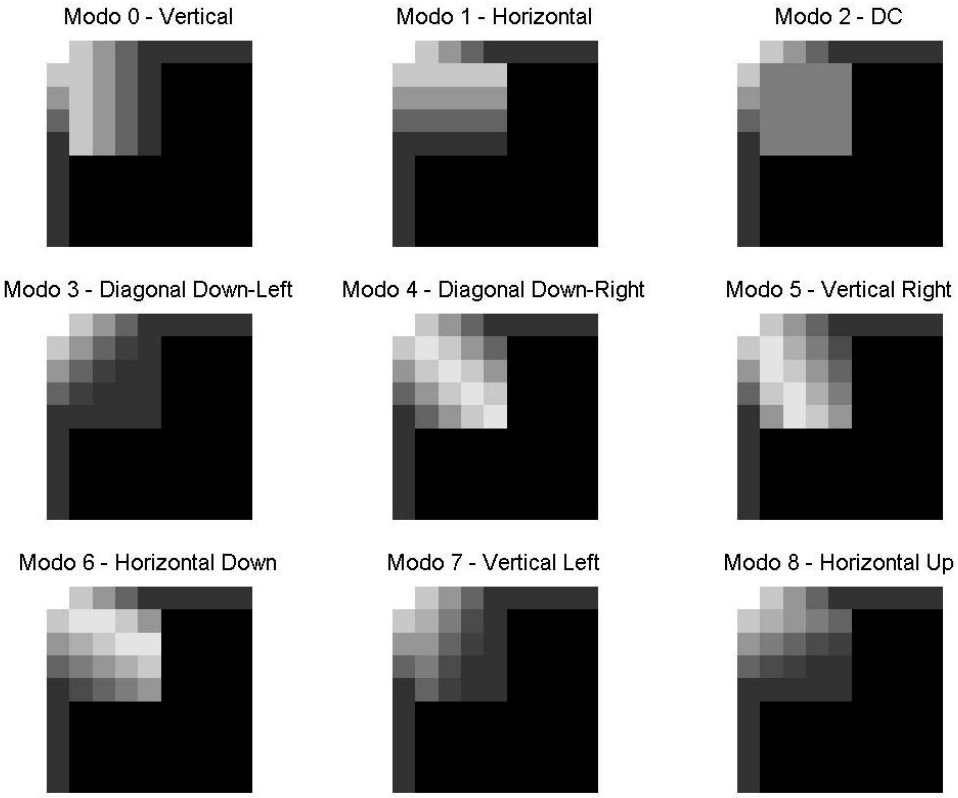

Figura B.3: Demonstração dos Modos de Predição Intra 4x4 a Partir do Modelo Matlab

Figura B.4: Predição Intra 4x4. a) Quadro Original. b) Predição. c) Resíduo.

Abaixo mostramos um trecho do código Matlab elaborado. O primeiro refere-se à descrição do modo 6 (Horizontal-Down), sendo que os índices da matriz foram ajustados porque Matlab não permite índices negativos ou iguais a zero. O segundo refere-se ao codificador, que testa cada um dos modos e verifica qual provê menor erro. Como parâmetro de decisão do modo utilizou-se o SAD (*sum of absolute differences*), isto é, faz-se a diferença entre o bloco 4x4 original e o bloco predito, e somam-se todas as diferenças absolutas dos 16 valores. O modo que fornecer o menor resultado será o escolhido. Antes de testar cada modo, o codificador confere se os vizinhos necessários à execução daquele modo estão disponíveis. No exemplo apresentado, para testar o modo 0 (Vertical) o programa confere se o grupo de vizinhos superiores está disponível. Também vale notar que há apenas uma possibilidade correta de ordem de navegação

pelos blocos de um quadro, que deve ser respeitada para o correto processamento das predições.

```
function saida = modo6(temp);
% MODO 6 - HORIZONTAL DOWN
% Predicao Intra 4x4

saida = temp;

for x=2:5
    for y=2:5
        if (2*y-x-2==0 | 2*y-x-2==2 | 2*y-x-2==4 | 2*y-x-2==6)
            saida(y,x) = (temp(y-x/2-1+1,1) + temp(y-x/2+1,1) + 1)/2;
        else
            if (2*y-x-2==1 | 2*y-x-2==3 | 2*y-x-2==5)
                saida(y,x) = (temp(floor(y-x/2-2)+2,1) + 2*temp(floor(y-x/2-1)+2,1) +
temp(floor(y-x/2)+2,1) + 2)/4;
            else
                if (2*y-x-2==-1)
                    saida(y,x) = (temp(2,1) + 2*temp(1,1) + temp(1,2) + 2)/4;
                else
                    saida(y,x) = (temp(1,x-1) + 2*temp(1,x-2) + temp(1,x-3) + 2)/4;
                end
            end
        end
    end
end
```

```
if viz_sup==1
   predicao = modo0(temp);
   erro_tmp = temp(2:5,2:5) - predicao(2:5,2:5);
   erro_tmp_abs = abs(sum(sum(erro_tmp)));
   if (erro_tmp_abs < erro_abs)
      erro = erro_tmp;
      erro_abs = erro_tmp_abs;
      quadro_predito(y+h+j+1:y+h+j+4,x+l+i+1:x+l+i+4) = predicao(2:5,2:5);
   end
end
```

Projeto de Uma Arquitetura de Hardware Para o Módulo de Predição Intra 4x4

Observando-se cada fórmula da tabela acima, nota-se que todas, com exceção do modo DC, apresentam a seguinte forma:

$$P_i = (viz1 + c1*viz2 + viz3 + c2) >> c3. \qquad (1)$$

Deste modo, pode-se desenvolver um bloco de hardware que processe estas operações, e cujas entradas possam ser chaveadas de acordo com o modo e com a posição da amostra dentro do bloco 4x4. A Figura B.5 representa o Elemento de Processamento, aonde os vizinhos viz1, viz2 e viz3, e as constantes c1, c2 e c3 devem ser corretamente chaveadas pela máquina de controle de acordo com a posição do pixel e com o modo de predição.

Figura B.5: Elemento de Processamento da Predição 4x4

O elemento de apresentado acima é capaz de realizar a soma de 4 elementos, segundo a fórmula (1). Como o modo de predição DC pode precisar da soma de até 8 elementos (caso tanto os vizinhos superiores quanto os da esquerda estejam disponíveis), utilizando-se dois Elementos de Processamento em paralelo podemos realizar a predição DC em apenas um ciclo.

De maneira a reduzir a freqüência de operação deste módulo, decidiu-se por colocar não apenas dois mas quatro Elementos de Processamento em paralelo. Assim, além de estar apto a calcular o modo de predição DC em apenas um ciclo, pode calcular quaisquer outros modos emitindo amostras a uma taxa de 4 por ciclo. A arquitetura final é apresentada na Figura B.6.

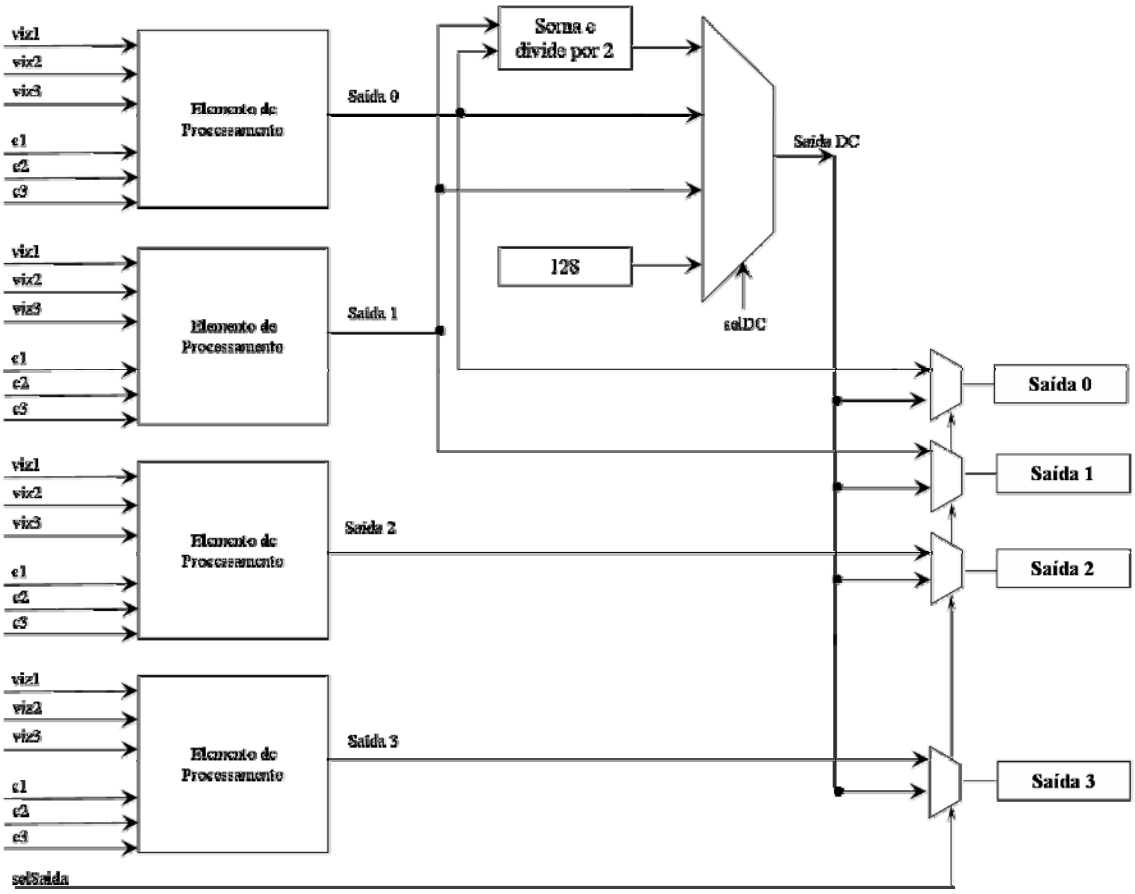

Figura B.6: Esquemático do Bloco Operacional da Predição Intra 4x4

Este sistema foi inteiramente descrito em linguagem VHDL, sintetizado com a ferramenta ISE da Xilinx, simulado com a ferramenta Modelsim, e finalmente prototipado em uma placa Digilent XUPV2P contendo um FPGA Xilinx Virtex II Pro VP30, de 30.000 células lógicas. Abaixo segue um trecho do código VHDL descrevendo o Elemento de Processamento e, em seguida, um trecho da descrição da parte de controle chaveando as entradas de maneira a executar o modo 4(Diagonal Down-Right).

```
-- Elemento de Processamento (PE)
process(coef1, viz2)
begin
   case coef1 is
      when "000" =>  -- multiplica por 0
         temp1 <= "0000000000";
      when "001" =>-- multiplica por 1
         temp1 <= ("00"&viz2);
      when "010" =>-- multiplica por 2
         temp1 <= ('0'&viz2&'0');
      when others =>-- multiplica por 3
         temp1 <= ('0'&viz2&'0') + ("00"&viz2);
   end case;
end process;
temp2 <= (("00"&viz1) + temp1) + (("00"&viz3) + ("00"&coef2));
process(coef3, temp2)
begin
   case coef3 is
      when "000" =>  -- nao divide
         saida <= temp2(7 downto 0);
      when "001" =>  -- divide por 2
         saida <= temp2(8 downto 1);
      when others => -- divide por 4
         saida <= temp2(9 downto 2);
   end case;
end process;
```

```
--modo 4
when st15=>
   selMux_PE0_1<= "1001";--J
   c1_PE0 <= "010"; --2
   selMux_PE0_2<= "1000";--I
   selMux_PE0_3 <= "1100";--M
   selMux_PE0_4<= "1111";--2
   c3_PE0 <= "010"; -- >>2
   selMux_PE1_1 <= "1000";--I
   c1_PE1 <= "010"; --2
   selMux_PE1_2 <= "1100";--M
   selMux_PE1_3 <= "0000";--A
   selMux_PE1_4<= "1111";--2
   c3_PE1 <= "010"; -- >>2
   selMux_PE2_1 <= "1100";--M
   c1_PE2 <= "010"; --2
   selMux_PE2_2 <= "0000";--A
   selMux_PE2_3 <= "0001";--B
   c2_PE2 <= "00000010";    --2
   c3_PE2 <= "010"; -- >>2
   selMux_PE3_1 <= "0000";--A
   c1_PE3 <= "010"; --2
   selMux_PE3_2 <= "0001";--B
   selMux_PE3_3 <= "0010";--C
   c2_PE3 <= "00000010"; --2
   c3_PE3 <= "010"; -- >>2
   selMux_S <= '0';
   next_state <= st16;
```