UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

LUCAS MARIANO LEIPNITZ DE FRAGA

Acelerador de Hardware com Arquitetura de Módulos Especializados para Predição Intra-Quadro do Versatile Video Coding

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação

Orientador: Prof. Dr. Cláudio Machado Diniz

Porto Alegre
2023

RESUMO

Padrões de codificação de vídeo vêm sido desenvolvidos desde 1980 com o intuito de definir as formas que um vídeo deve ser comprimido e são necessários para que codificadores e decodificadores possam se comunicar, além de definirem técnicas extremamente eficientes de compressão de vídeo. Recentemente eles se provam cada vez mais essenciais para o envio de vídeos pela rede ou armazenamento de forma eficiente em tempos onde aplicações de vídeo aumentam cada vez mais, principalmente por conta dos serviços de *streaming* e vídeo conferência. Por conta deste aumento de demanda por maior eficiência de compressão, padrões de codificação estão se tornando cada vez mais complexos e custosos principalmente para execução em *software* e, cada vez mais, investimentos em aceleradores de *hardware* se tornam mais comuns e necessários, tanto para o armazenamento viável de quantidades massivas de vídeos, quanto para transmissões de vídeos de alta qualidade, como Full HD, QHD e UHD, em taxas de transmissão confortáveis para o espectador, como 30 fps ou 60. Este trabalho apresenta uma arquitetura de acelerador de *hardware* com módulos especializados para o processo de predição intra-quadro do *Versatile Video Coding*, o mais novo padrão de codificação de vídeo da JVET.

Palavras-chave: Padrões de codificação de vídeo, compressão de vídeo, VVC, Acelerador de *hardware*.

Hardware Accelerator with Specialized Modules Architecture for VVC Intra-Frame Prediction

ABSTRACT

Video coding standards have been developed since the 1980s with the purpose of defining how a video should be compressed. They are necessary for encoders and decoders to communicate effectively. Recently, they have proven to be increasingly essential for sending videos over networks or efficiently storing them. This is particularly crucial as video applications continue to grow, primarily due to streaming services and video conferencing. Due to the rising demand for greater video compression efficiency, video coding standards have become more intricate and resource-intensive, especially for *software* execution. Consequently, investments in *hardware* accelerators have become more common and necessary. These accelerators are crucial for both the viable storage of massive amounts of videos and the transmission of high-quality videos, such as Full HD, QHD, and UHD, at comfortable frame rates for viewers, like 30 fps or 60 fps.This work introduces a *hardware* accelerator architecture with specialized modules for the intra-frame prediction process of the latest video coding standard from JVET, the Versatile Video Coding (VVC).

Keywords: Video coding standards, video compression, VVC, Harware Accelerator.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

HD	*High Definition*
QHD	*Quad High Definition*
UHD	*Ultra High Definition*
ASIC	*Application-specific Integrated Circuit*
VHDL	*VHSIC hardware Description Language*
ISO	*International Organization for Standardization*
IEC	*International Electrotechnical Commission*
MCM	*Multiple Constant Multiplication*
NP	*Nondeterministic Polynomial-time*
FIR	*Finite Impulse Response*
DCT	*Discrete Cosine Transform*
SIF	*Smoothing Interpolation Filter*
DCTIF	*DCT-based interpolation filter*
AVC	*Advanced Video Coding*
HEVC	*High Efficiency Video Coding*
VVC	*Versatile Video Coding*
CTU	*Coding Tree Unit*
CU	*Coding Unit*
TB	*Transform Block*
MRL	*Multiple Reference Line*
ISP	*Intra Subpartition*

SUMÁRIO

1 INTRODUÇÃO

Nos dias de hoje, aplicações de vídeo digital têm se tornado extremamente presentes no cotidiano da população mundial. Tanto para o lazer quanto para o meio profissional, *streaming* de vídeo e videoconferência se tornaram uma das principais fontes de comunicação, de forma que a demanda por maior qualidade vem aumentando cada vez mais. A compressão de vídeo é essencial para permitir que os dispositivos armazenem vídeos de alta resolução, como 1080p, 1440p e 4K, sem ocupar muito espaço, e para garantir que as redes de comunicação possam transmitir de forma eficiente a enorme quantidade de dados desses vídeos de alta definição.

Para que computadores possam comprimir vídeos de forma eficiente, padrões de codificação de vídeo vêm sido estudados e desenvolvidos desde 1980. Em geral, eles definem como um vídeo será comprimido por um codificador na origem (onde o vídeo é capturado por uma câmera, por exemplo) e como o vídeo será descomprimido por um decodificador no destino, onde o vídeo será exibido em um *display*. Alguns dos primeiros padrões desenvolvidos e utilizados na indústria foram o H.261 (ITU-T, 1990) e o MPEG-1 (ISO, 1993), tendo como sucessores o MPEG-2/H.262 (ISO/IEC, 1994), o H.263 (ITU-T, 1995) e o MPEG-4 parte 2 (MPEG -4 Visual) (ISO/IEC, 1999). Atualmente, como podemos ver em (BITMOVIN, 2022), o padrão mais utilizado na indústria é o H.264/AVC (ITU-T; 1, 2003), com o sucessor H.265/HEVC (ITU-T; ISO/IEC, 2013) logo na sequência de utilização.

Para acompanhar o aumento da demanda por maior qualidade de vídeo, os novos padrões de codificação utilizam técnicas de compressão mais eficientes, com o intuito de diminuir o número de *bits* do vídeo comprimido mantendo a qualidade visual. O *Versatile Video Coding* (VVC), publicado pela ISO/IEC e ITU-T em 2020 (ITU-T; ISO/IEC, 2020), supera o seu antecessor H.265/HEVC em eficiência de compressão, porém, ao custo de uma maior complexidade computacional (BROSS et al., 2021). Em (SIQUEIRA; CORREA; GRELLERT, 2020), foi conduzida uma análise detalhada da complexidade computacional comparando o VVC e o HEVC através de vários testes utilizando seus respectivos *softwares* de referência. Os resultados revelaram que o VVC obteve uma economia de 44% em taxas de *bits* em relação ao HEVC, mantendo a mesma qualidade visual. No entanto, o tempo de execução do *software* de referência do codificador de vídeo compatível com o VVC foi até dez vezes maior que o *software* do seu antecessor, o HEVC.

Uma das etapas mais custosas do processo de codificação do padrão VVC é a

predição intra-quadro (PFAFF et al., 2021). Ela realiza a compressão de uma unidade de codificação (bloco de amostras retangular de uma imagem do vídeo) com base nas amostras vizinhas da mesma imagem que foram anteriormente processadas. O VVC introduz 66 modos de predição intra-quadro, tornando esta etapa muito mais complexa que os padrões anterior, como o HEVC, que possui 35 modos de predição intra-quadro. Um codificador VVC em *software* que só suporta predição intra-quadro pode ser 34 vezes mais lento que um codificador similar do padrão HEVC (MERCAT et al., 2021).

Por conta desta alta complexidade, a execução de codificadores de vídeo em software acaba se tornando ineficiente, nos fazendo depender de técnicas de otimização, como o projeto de aceleradores de *hardware*. Alguns artigos foram encontrados na literatura que apresentam arquiteturas de *hardware* para aceleradores da predição intra-quadro para o padrão VVC. (AZGIN; KALALI; HAMZAOGLU, 2019) desenvolveu uma arquitetura em FPGA (Field-programmable gate array) para o VVC, mas apresenta algoritmos não atualizados, já que foi desenvolvido em 2019, antes da finalização do VVC em 2020. Já, (BORGES et al., 2023) apresenta uma arquitetura sintetizada para ASIC (Application-Specific Integrated Circuit), que se baseia em uma heurística de modos de predição intra-quadro mais utilizados de forma a implementar em *hardware* menos modos, com foco predição da crominância de cada *pixel*.

Este trabalho tem o objetivo de propor uma arquitetura de acelerador de *hardware* para a predição intra-quadro para codificadores compatíveis com o padrão VVC, com o intuito de otimizar o tempo de execução e o custo de energia necessários para o processo. A arquitetura implementa os 18 modos mais influentes do VVC, definidos na heurística de mais utilizados citada anteriormente, dando suporte para blocos de tamanho 32×32. Os modos farão a predição para a luminância de cada *pixel*, que consiste geralmente de um filtro de quatro multiplicações utilizando duas funções diferentes, mais complexa do que a predição para crominância que consiste de um filtro de duas multiplicações com uma função única (ITU-T; ISO/IEC, 2020).

A nossa motivação é utilizar módulos especializados para cada modo, que serão resolvidos em paralelo, para realizar múltiplos filtros ao mesmo tempo. Também utilizaremos, blocos MCM, que apresentaremos na seção 2.6 para substituir as multiplicações dos filtros da predição intra por somas e deslocamentos usufruir do reuso oferecido.

A nossa contribuição é o desenvolvimento da arquitetura especializada na predição da luminância, que utiliza de filtros mais complexos que da crominância, e o desenvolvimento de um caminho de dados baseado em blocos MCM, para acompanhar o aumento

da complexidade. A nossa arquitetura será implementada para ASIC, com o objetivo de ter um custo energético menor do que em FPGA, em troca da flexibilidade.

A arquitetura foi desenvolvida em linguagens de descrição de *hardware*, com parte na linguagem VHDL (VHSIC Hardware Description Language) e parte na linguagem Verilog, e sua síntese em ASIC utilizando ferramentas de apoio ao projeto de circuitos integrados comerciais. Ao final, é realizada a análise do desempenho alcançado pelo acelerador de *hardware*.

Este trabalho foi organizado da seguinte forma. No Capítulo 2 é abordada a fundamentação teórica sobre representação e codificação de vídeos digitais. No Capítulo 3 é apresentado um detalhamento da predição intra-quadro no VVC e revisão da literatura sobre arquiteturas de aceleradores de *hardware* para predição intra-quadro. No Capítulo 4 são apresentados os conceitos teóricos para a predição intra-quadro para as diferentes amostras, bem como a proposta da arquitetura, implementação e resultados da síntese lógica. No capítulo 5 são apresentadas as conclusões e sugestões de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo tem o objetivo de revisar conceitos teóricos sobre vídeos digitais, compressão de vídeo e circuitos digitais necessários para a elaboração deste trabalho.

2.1 Representação Digital de Vídeo

Vídeos podem ser representados de forma analógica, como um sinal variante no tempo, ou de forma digital, como uma sequência de imagens digitais. As imagens digitais que formam o vídeo, conhecidas como *frames*, são exibidas sequencialmente ao longo de um determinado período (por exemplo, 30 *frames* em apenas um segundo), criando a ilusão de movimento para o sistema visual humano e, consequentemente, gerando o vídeo. Os *frames*, por sua vez, são compostos por *pixels*, estruturados em formato de uma ou multiplas matrizes $m \times n$, onde m é número de colunas e n é o número de linhas.

As dimensões da matriz formam a resolução do vídeo, por exemplo, vídeos HD (*High Definition*) possuem *frames* formados por matrizes de 1920 *pixels* na direção horizontal e 1080 *pixels* na direção vertical. Por sua vez, cada *pixel* contém uma intensidade, que é um valor inteiro definido pelo esquema de amostragem e a quantidade de valores que podem ser representados. Por exemplo, em um esquema de amostragem de tons de cinza, com 256 valores diferentes, um *pixel* pode ter valores entre 0 e 255 onde, conforme mais próximo de 255, mais intenso é o valor de cinza.

Um dos principais esquemas de amostragem, utilizado para reprodução de vídeos coloridos, é o esquema RGB (*red*, *green* e *blue*), onde cada *pixel* representa a cor a partir da combinação de três componentes de cor: vermelho, verde e azul. Cada uma das componentes que formam um *pixel* é chamado de amostra, cujo nível de intensidade é representado por um conjunto de *bit* (8 *bits* é um valor bastante usado). Neste esquema, cada *frame* é composto de três matrizes, uma para cada uma das cores, com o mesmo espaço de amostragem. Cada *frame* terá, então, as três matrizes, permitindo que as combinações de valores sejam formadas as demais cores visíveis pelo olho humano.

Diferente da reprodução do vídeo, a codificação utiliza o esquema de subamostragem YCbCr (*luma*, *chrominance blue*, *chrominance red*). Esse esquema permite separarmos a crominância (Cb e Cr) da iluminância (Y). Como o sistema visual humano é mais sensitivo a iluminância, podemos então reduzir a amostragem de crominância em relação à luminância, reduzindo o tamanho do sinal em troca de uma diferença insigni-

ficante nas cores do vídeo para o olho humano. Por exemplo, a amostragem 4:2:2, dois componentes de crominância para cada quatro componentes de luminância, geralmente a mais utilizada, requer dois terços da largura da banda de 4:4:4 YCbCr.

2.2 Codificação de Vídeo

O processo de codificação envolve a redução do tamanho de um vídeo para armazenamento ou transmissão e, posteriormente, a descompressão para a reprodução. Quando a qualidade do vídeo é preservada durante a compressão, chamamos isso de compressão sem perda de informação, sendo essencial em aplicações que exigem a manutenção exata da imagem original após a compressão. Por outro lado, quando ocorre uma diminuição na qualidade, chamamos de compressão com perda de informação. Esta última é mais comumente utilizada, pois oferece taxas de compressão muito maiores, explorando as limitações da percepção visual humana para que a diferença de qualidade entre a imagem original e a descomprimida seja praticamente imperceptível (DINIZ et al., 2009). A redução de amostragem de crominância em relação a luminância, detalhada na Seção 2.1, é um exemplo de compressão com perda de informação.

Técnicas de compressão podem ser aplicadas tanto no domínio espacial quanto no domínio frequência (DINIZ et al., 2009). Uma das principais técnicas de codificação por perdas que ocorre no domínio frequência é a etapa de quantização. Após uma imagem ser transformada em um sinal através de uma transformada, como DCT (*Discrete Cosine Transform*), a etapa de quantização será responsável por reduzir a precisão do sinal, reduzindo detalhes normalmente imperceptíveis.

No domínio espacial, uma das etapas do processo de codificação é o processo de predição de quadros, que por sua vez, é dividido em dois tipos: predição inter-quadros, que reduz a redundância temporal, e predição intra-quadro, que reduz a redundância espacial. A predição inter-quadros tem o objetivo de encontrar correlações entre os *pixels*, que chamaremos de amostras a partir de agora, distribuídos em *frames* temporalmente próximos do vídeo, enquanto que a predição intra-quadro tem o objetivo de encontrar correlações entre as amostras de um único quadro. Este tipo de predição fornece ganhos de compressão quando a predição inter-quadros não encontra um bom resultado ou quando não pode ser aplicada (DINIZ et al., 2009).

Grande parte das técnicas de predição intra-quadro envolve executar muitos filtros de interpolação em cima de amostras de um único *frame*, o que faz com que multiplicação

por constantes sejam umas das principais operações aritméticas realizadas (BROSS et al., 2021).

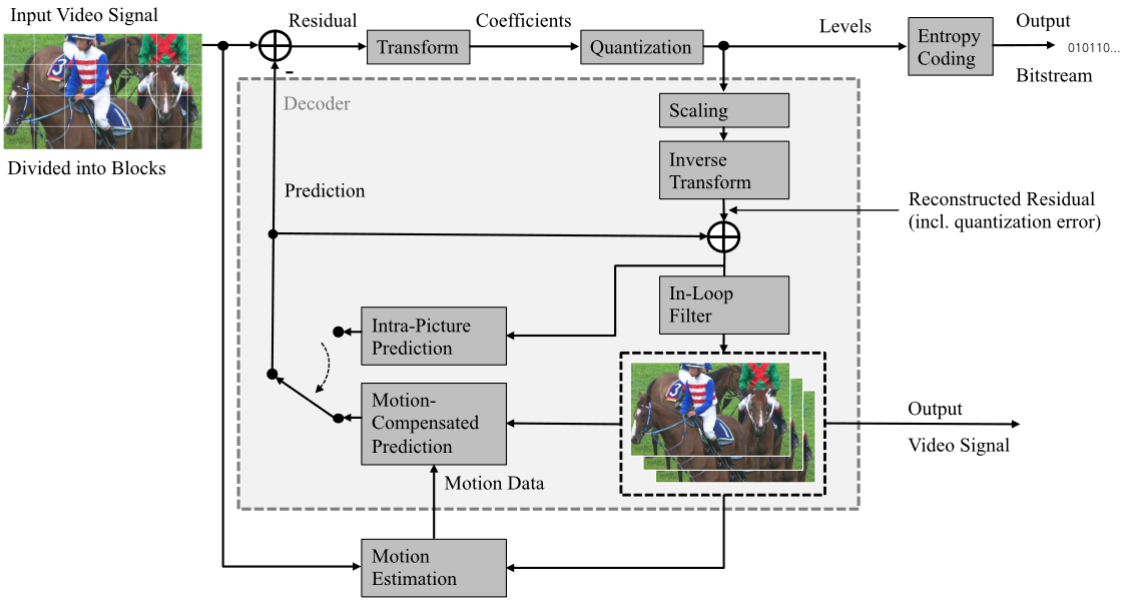
Após passar por todas as etapas, o bloco codificado e um *bitstream* com informações para que o decodificador possa fazer o processo inverso e obter o bloco original são enviados ou armazenados. Por conta dos vários modos que devem ser calculados e testados, o codificador é muito mais complexo que o decodificador, já que o último apenas apenas precisará calcular um único modo, sinalizado pelo codificador.

2.3 Etapas do Processo de Codificação no VVC

O VVC foi desenvolvido pelo *Joint Video Experts Team* (JVET), formado por especialistas do ITU-T *Video Coding Experts Group* (VCEG) e e do ISO/IEC *Moving Picture Experts Group* (MPEG) e oferece compressão elevada em relação ao seu sucessor, o HEVC (PFAFF et al., 2021). O VVC, assim como todos os padrões anteriores, é baseado no princípio de codificação híbrida (BROSS et al., 2021), que envolve a combinação do processo de predição de quadros e quantização explicados na Seção 2.2. O diagrama de blocos e das fases de codificação do VVC é ilustrado na Figura 2.1.

As etapas do processo de codificação do vídeo são: o particionamento da imagem em múltiplos blocos menores, predição inter-quadros e predição intra-quadro, transformação do domínio espacial para o domínio frequência, quantização, e então codificação por entropia, gerando o *bitstream*, que é fluxo de *bits* do vídeo comprimido. Podemos ver que a etapa de transformação atua no resíduo entre a imagem original e a imagem predita, e que, após a etapa de quantização, o *frame* já quantizado é transformado novamente para o domínio espacial, por meio de *scaling* e transformada inversa e soma com o próprio bloco predito para ser utilizado na etapa de predição intra-quadro. Na próxima Seção é abordado o processo de divisão de blocos. O Capítulo 3 aborda com mais detalhe a etapa de predição intra-quadro, que é o foco deste trabalho.

Figura 2.1: Diagrama de blocos das etapas de codificação de vídeo no VVC



Fonte: (BROSS et al., 2021)

2.4 Divisão de Blocos

A divisão de blocos é um processo importante para a codificação do vídeo, pois dependendo da forma como o *frame* é dividido, é possível encontrar diferentes resultados de compressão. Inicialmente, um *frame* é dividido em *Coding Tree Units* (CTUs) de tamanho máximo de 128×128 amostras de luminância. Cada CTU é, então, particionada recursivamente em *Coding Units* (CUs) de tamanhos menores e de dimensões retangulares. Cada CU é composta de apenas um único *Transform Block* (TB), exceto para o caso em que o bloco está no modo de subpartição intra (ISP) ou se exceder o tamanho possível de dimensão para o TB. As dimensões possíveis para os TBs, tanto em altura quanto em largura são 4, 8, 16, 32 e 64 amostras.

Particionar CTUs em tamanhos menores geralmente permite encontrar mais redundâncias e conseguir uma maior precisão, porém aumenta a quantidade de dados referentes a sinais de controle enviados ao decodificador, já que para todo bloco é preciso sinalizar para o decodificador qual modo de predição intra foi utilizado. Portanto, escolhendo entre os diferentes tamanhos de partições, o decodificador pode alternar entre eficiência de compressão e menor quantidade de dados de sinais de controle (PFAFF et al., 2021).

2.5 Application-specific Integrated Circuit

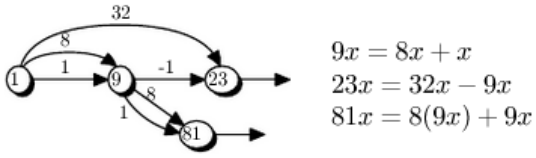
Os *Application-specific Integrated Circuits* (ASIC) são circuitos integrados projetados especificamente para implementar um único ou vários algoritmos identificados em tempo de projeto. Os circuitos ASIC abrem mão da programabilidade e generalização proporcionada pelos processadores de propósito geral em favor da especialização e desempenho. Devido a essa característica, eles são amplamente utilizados na codificação de vídeo, pois o processo de codificação geralmente envolve a repetição de padrões de algoritmos a serem executados. Além disso, devido à complexidade envolvida, esses algoritmos precisam ser implementados em circuitos de alto desempenho. Devido à especialização e desempenho, outra característica dos ASIC é que eles são muitos mais eficientes energeticamente quando comparados a processadores de propósito geral realizando a mesma função.

2.6 Multiplicações como Somas e Deslocamentos de Bits

A Seção 2.2 comenta que muitas das técnicas utilizadas em predição intra-quadro se baseiam em filtros e transformações, fazendo com que multiplicações de valores de amostras por constantes sejam as principais operações realizadas. Operações de multiplicação entre duas variáveis podem ser implementadas em ASIC, permitindo realizar as operações necessárias para o cálculo da predição intra-quadro. Porém, circuitos multiplicadores são consideravelmente complexos, e um acelerador de *hardware* para predição intra-quadro iria requerer uma quantidade grande de multiplicadores, o que tornaria o circuito bastante complexo. Contudo, as operações de predição intra-quadro são realizadas por meio de multiplicações entre variáveis e constantes, que podem ser substituídas por operações formadas por diversas combinações de somas, subtrações e deslocamentos de *bits*, que possuem um custo computacional muito menor que os multiplicadores.

Durante muito tempo, houve o interesse de reduzir as multiplicações em um filtro com o objetivo de diminuir o custo computacional em aplicações de *hardware*, o que levou ao desenvolvimento de técnicas que implementam multiplicações por constantes em formas de somas e deslocamentos (KUMM; VOLKOVA; FILIP, 2023). *Multiple Constant Multiplication* (MCM) é um problema que envolve descobrir a forma ótima de calcular um conjunto de produtos entre uma mesma variável e múltiplas constantes de ponto-fixo, utilizando apenas somas, subtrações e deslocamentos (VORONENKO; PüS-

Figura 2.2: Grafo exemplificando a heurística Hcub



Fonte: (VORONENKO; PüSCHEL, 2007)

CHEL, 2007). O MCM é um problema NP-Completo, então, encontrar a solução ótima é computacionalmente lento com os algoritmos conhecidos. Por conta disso, heurísticas foram desenvolvidas para encontrar resultados sub-ótimos de forma computacionalmente rápida.

Neste trabalho, utilizaremos a heurística Hcub, apresentada no artigo (VORONENKO; PüSCHEL, 2007). A Figura 2.2 mostra o grafo do comportamento da heurística para o exemplo de uma variável x multiplicando por duas constantes 23 e 81. A multiplicação pelos valores 8 e 32 é feita utilizando deslocamentos de bits para a esquerda por 3 e 5 respectivamente. O valor intermediário $9x$ é utilizado para calcular tanto 23 quanto 81, gerando um ganho de uma operação matemática caso as duas multiplicações fossem calculadas independentemente.

Podemos, então, definir que um bloco MCM recebe um valor de entrada, uma amostra de referência como estudaremos mais a frente, e retorna a multiplicação da referência por múltiplas constantes diferentes. Estas contantes podem variar de bloco para bloco e veremos um algoritmo para construir blocos para diferentes referências.

2.7 Caminho de dados e Controle

Um dos conceitos teóricos fundamentais de arquiteturas de circuitos digitais que abordaremos neste trabalho é o conceito de caminho de dados. O caminho de dados de uma arquitetura é um circuito composto de diversas unidades funcionais para a realização de operações necessárias para realizar o objetivo da arquitetura. Estas unidades funcionais podem ser somadores, subtratores, multiplexadores, registradores, entre outros, e são responsáveis por transformar ou armazenar dados iniciais em dados finais, armazená-los e escolher entre múltiplos caminhos diferentes para o qual ele deve enviar os dados para serem feitas operações distintas.

Porém, multiplexadores precisam saber quando é necessário escolher entre múl-

tiplos caminhos, ou quando é necessário escolher entre múltiplos dados que está recebendo. O controle é responsável por escolher quais entradas de um multiplexador devem ser selecionadas e quais endereços de um registrador devem ser lidos e escritos. Uma das estratégias mais comuns é implementar o controle como uma máquina de estados, que enviará *bits* de controle para o caminho de dados a partir do estado atual em que se encontra.

Outro conceito teórico de circuitos digitais que utilizaremos para justificar a nossa implementação é o caminho crítico. Todo circuito digital possui um atraso de propagação que está diretamente relacionado a quantidade e atraso individual das unidades funcionais e portas lógicas em seu caminho de dados. Quando existem múltiplos caminhos que podem ser escolhidos, o maior atraso é chamado de caminho crítico, e é ele quem vai definir o atraso total do circuito. O caminho crítico influencia diretamente na frequência de *clock* que podemos estabelecer para o circuito.

3 PREDIÇÃO INTRA-QUADRO DO VVC E TRABALHOS RELACIONADOS

Neste capítulo, abordaremos os aspectos teóricos da sobre o padrão VVC e a predição intra-quadro do padrão VVC, foco deste trabalho. Neste capítulo abordaremos várias equações e, para facilitar a compreensão, utilizaremos a lista de símbolos a seguir.

1. $*$ representa multiplicação aritmética.
2. $<<$ representa deslocamento lógico de *bits* para a esquerda.
3. $>>$ representa deslocamento lógico de *bits* para a direita.
4. $\&$ representa a operação *AND* lógica de *bits*.
5. $Log2(x)$ representa a operação de logaritmo com base 2 da variável x.
6. $<<$ representa deslocamento lógico de *bits* para a esquerda.
7. $x = a\ ?\ b : c$ representa o operador ternário, x recebe o valor b caso a sejá verdade, do contrário, recebe c.
8. $Round(x)$ representa a operação de arredondamento para baixo da variável x.
9. $Clip1(x)$ representa função que mantém a variável x dentro do intervalo de 0 a 255. Se $x > 255$, retorna 255, se $x < 0$, retorna 0, do contrário, retorna o valor de x.
10. $Min(x,y)$ representa função que retorna o valor mínimo entre x e y.
11. $Abs(x)$ representa função que retorna o valor absoluto de x.

Além disso, utilizaremos a notação matricial utilizada em (ITU-T; ISO/IEC, 2020) e em linguagens de programação, onde uma matriz $A_{x,\,y}$ é definida como A[x][y].

3.1 Predição Intra-Quadro do VVC

Como mencionamos na Seção 2.2, a predição intra-quadro é dividida em diferentes modos, sendo escolhido, para cada bloco de transformação, aquele que tiver o melhor desempenho. No VVC, existem quatro categorias para um total de 66 modos para blocos quadrados mais 28 para blocos retangulares. São elas: intra planar (modo 0), intra DC (modo 1), intra angular (modos 2 a 66) e WAIP (modos -1 a -14 e 66 a 80). Um dos novos serviços para predição intra quadros disponibilizados pelo VVC é o modo *Intra Subpartition* (ISP). Nele, blocos podem ser futuramente subdivididos em subpartições, em que a predição é calculada independentemente porém mantendo o modo do bloco original. Quando o modo está desativado, para a variável

`IntraSubpartitionSplitType` é atribuído o valor `ISP_NO_SPLIT` (valor zero). Neste trabalho, a implementação apresentada não utilizara o modo ISP e, portanto, a variável `IntraSubpartitionSplitType` será abstraída de futuras definições. Além disso, utilizarmos apenas modos para blocos quadrados, portanto, não abordaremos os modos WAIP, para facilitar a implementação.

3.2 Amostras de Referência

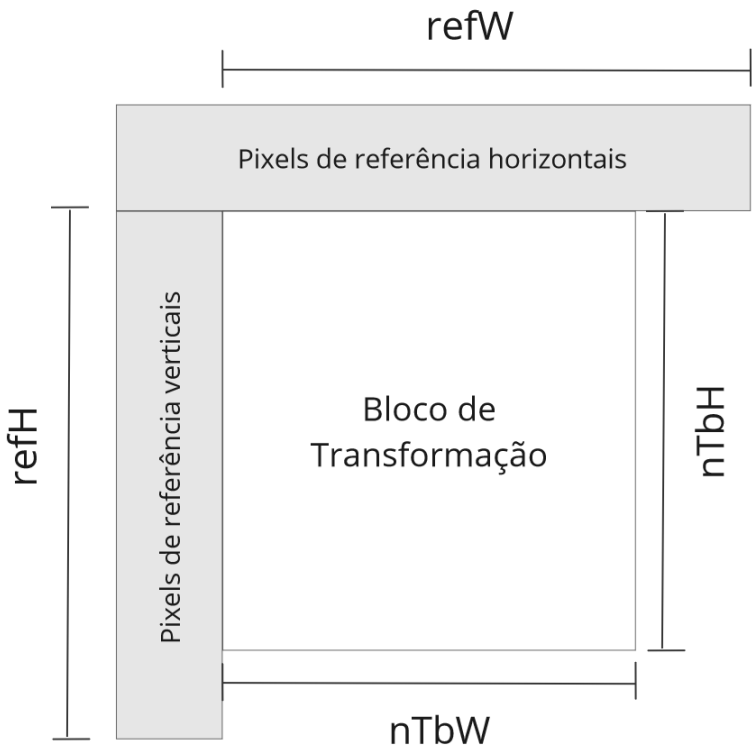
Para realizar a predição intra no bloco selecionado, o codificador contará com um vetor de amostras de referência, que passarão pelos filtros de interpolação e depois serão associadas ao bloco que será predito. Em essência, é este processo que gera a diminuição no tamanho do bloco predito em relação ao bloco original. Este vetor de referência pode mudar de modo para modo, mas para todos, as amostras fazem parte de blocos vizinhos já preditos, que se encontram acima ou à esquerda do bloco de predição atual. A Figura 3.1 mostra um bloco de transformação e suas amostras de referência, onde o vetor de referência é composto pelas amostras de referência verticais e horizontais.

As variáveis `nTbW` e `nTbH` especificam, respectivamente, o comprimento e a altura do bloco de transformação. Dependendo do modo de predição, serão utilizadas as variáveis `refW` e `refH` que especificam o comprimento das amostras de referência (quantidade de de amostras a cima do bloco) e a altura das amostras de referência (quantidade de amostras a esquerda do bloco) respectivamente. Uma das novas ferramentas disponiveis no VVC é o *Multiple Reference Line* (MRL) *Predction*, onde, para cada bloco, é possível escolher o índice da linha dos blocos adjacentes em que o vetor de referência se encontra, especificado pela variável `refIdx`.

Quando o modo MRL está desabilitado para um bloco, `refIdx` tem valor zero, fazendo com que as linhas verticais e horizontais de referencia sejam diretamente adjacente ao bloco de transformação, como mostrado na Figura 3.1. Neste trabalho, a implementação apresentada não utilizara o modo MRL e, portanto, a variável `refIdx` será abstraída de definições futuras para facilitar a compreensão.

A partir daqui, as amostras do bloco de transformação serão chamadas de amostras de predição ou *predSample*, enquanto que amostras do vetor de referência serão chamadas de amostras de referência. Elas serão representadas pela notação matricial definida em 3.1, as amostras horizontais em 3.2 e as verticais em 3.3.

Figura 3.1: Bloco de Transformação e Amostras de Referência Vizinhas



O autor

$$predSample[x][y] = \{p[x][y] \mid x \in \{0,1,2,3,...nTbW-1\}, y \in \{0,1,2,3,...nTbH-1\}\} \tag{3.1}$$

$$horRef[x] = \{p[x][y] \mid x \in \{-1,0,1,...refW-1\}, y = -1\} \tag{3.2}$$

$$verRef[y] = \{p[x][y] \mid x = -1, y \in \{0,1,...refH-1\}\} \tag{3.3}$$

Onde $p[x,y]$ é o *pixel* na posição de coluna x e linha y, $predSample$ é a matriz das amostras de predição, $horRef$ das que fazem parte da linha horizontal do vetor de referência e $verRef$ das que fazem parte da linha vertical. Esta notação será utilizada nas próximas definições.

Esta notação será utilizada nas próximas definições. A partir de agora, apresentaremos as equações dos modos de predição, apresentadas nos capítulos 8.4.5.2.11, 8.4.5.2.12 e 8.4.5.2.13 de (ITU-T; ISO/IEC, 2020), aletradas para se encaixar neste trabalho.

3.3 Modo Intra Planar

No modo 0 ou intra planar, para cada amostra de um bloco, o novo valor será obtido utilizando uma média ponderada a partir de quatro referências, a primeira se encontra na mesma coluna que a amostra que será predita e a segunda na mesma linha, já a terceira se encontra no canto superior direito em relação ao bloco de transformação e a quarta no canto inferior esquerdo. As equações para a predição planar são definidas em 3.4, 3.5 e 3.6.

$$predV[x][y] = ((nTbH - 1 - y) * p[x][-1] + (y + 1) * p[-1][nTbH]) << Log2(nTbW) \tag{3.4}$$

$$predH[x][y] = ((nTbW - 1 - x) * p[-1][y] + (x + 1) * p[nTbW][-1]) << Log2(nTbH) \tag{3.5}$$

$$\begin{aligned} predSamples[x][y] = (predV[x][y] + predH[x][y] + nTbW * nTbH) >> \\ (Log2(nTbW) + Log2(nTbH) + 1) \\ x = 0...nTbW - 1, y = 0...nTbH - 1 \end{aligned} \tag{3.6}$$

3.4 Modo Intra DC

No modo 1 ou intra DC, para blocos quadrados é feito uma média aritmética de todos as amostras de referência, este valor representará todos o bloco predito. Caso o bloco seja retangular, é feita a média apenas do conjunto de referência que tiver tamanho maior. As equações 3.7 e 3.8, definem o modo DC para blocos quadrados.

$$dcVal = (\sum_{x=0}^{nTbW-1} p[x][-1] + \sum_{y=0}^{nTbH-1} p[-1][y] + nTbW) >> (Log2(nTbW) + 1) \tag{3.7}$$

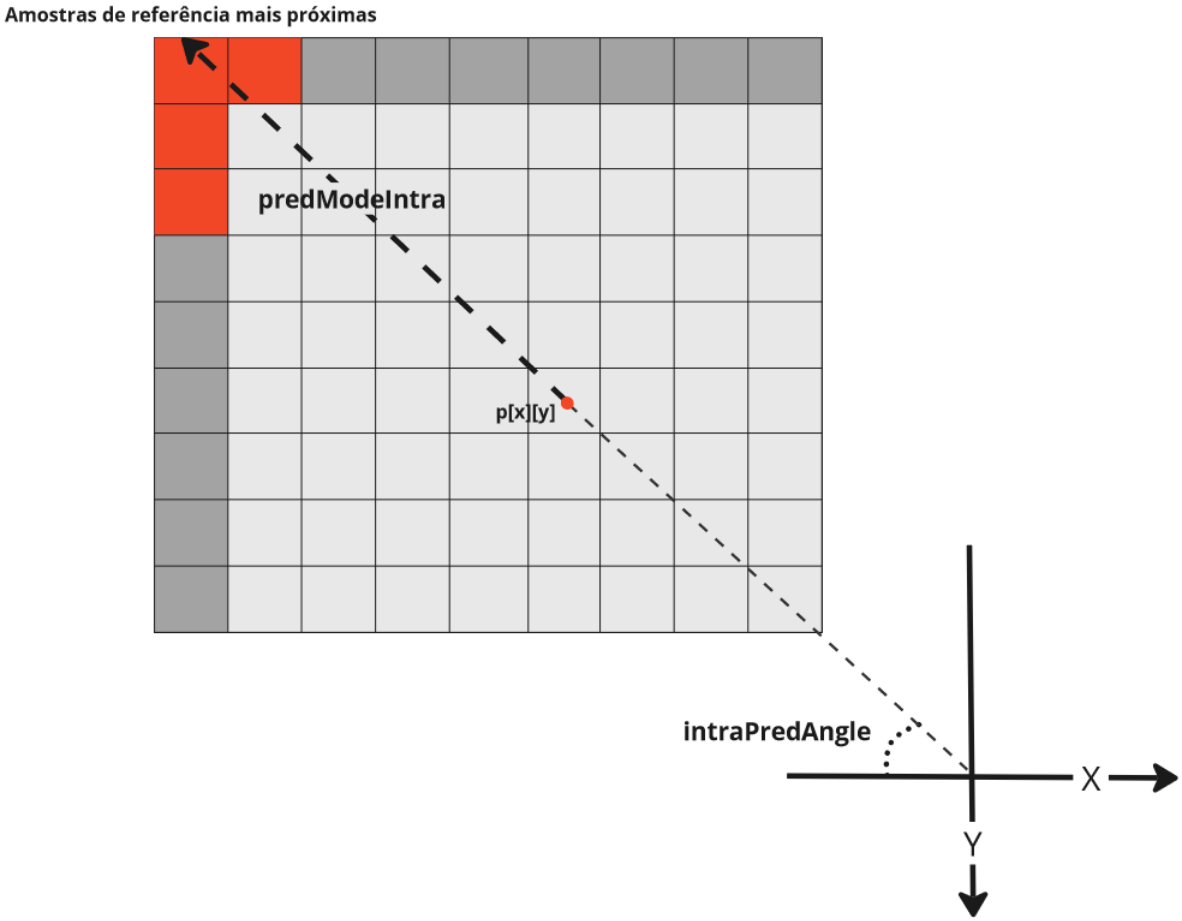
$$predSamples[x][y] = dcVal, x = 0...nTbW - 1, y = 0...nTbH - 1 \tag{3.8}$$

3.5 Modos Intra Angular

Os modos angulares são métodos de predição intra direcionais e têm como objetivo criar uma reta, em relação aos eixos x e y da imagem, a partir de uma amostra p[x][y] do bloco de transformação e calcular a equação de interpolação baseado nas amostras de referências mais próximas de onde a reta incide. A Figura 3.2 mostra um esquema da predição intra angular. A direção da reta é definida pelo angulo intraPredAngle do modo predModeIntra em relação ao eixo x, para modos menores que 34, ou em relação ao eixo y, para modos iguais ou maiores.

Os modos e ângulos estão definidos no documento oficial do VVC (ITU-T; ISO/IEC, 2020). A figura 3.3 e a Tabela 3.1 mostram a associação entre modos angulares e seus respectivos ângulos. Modos com linha pontilhada são pertencentes a modos WAIP e, como mencionado anteriormente na Seção 3.1, não serão abordados neste trabalho.

Figura 3.2: Representação de Predição intra angular

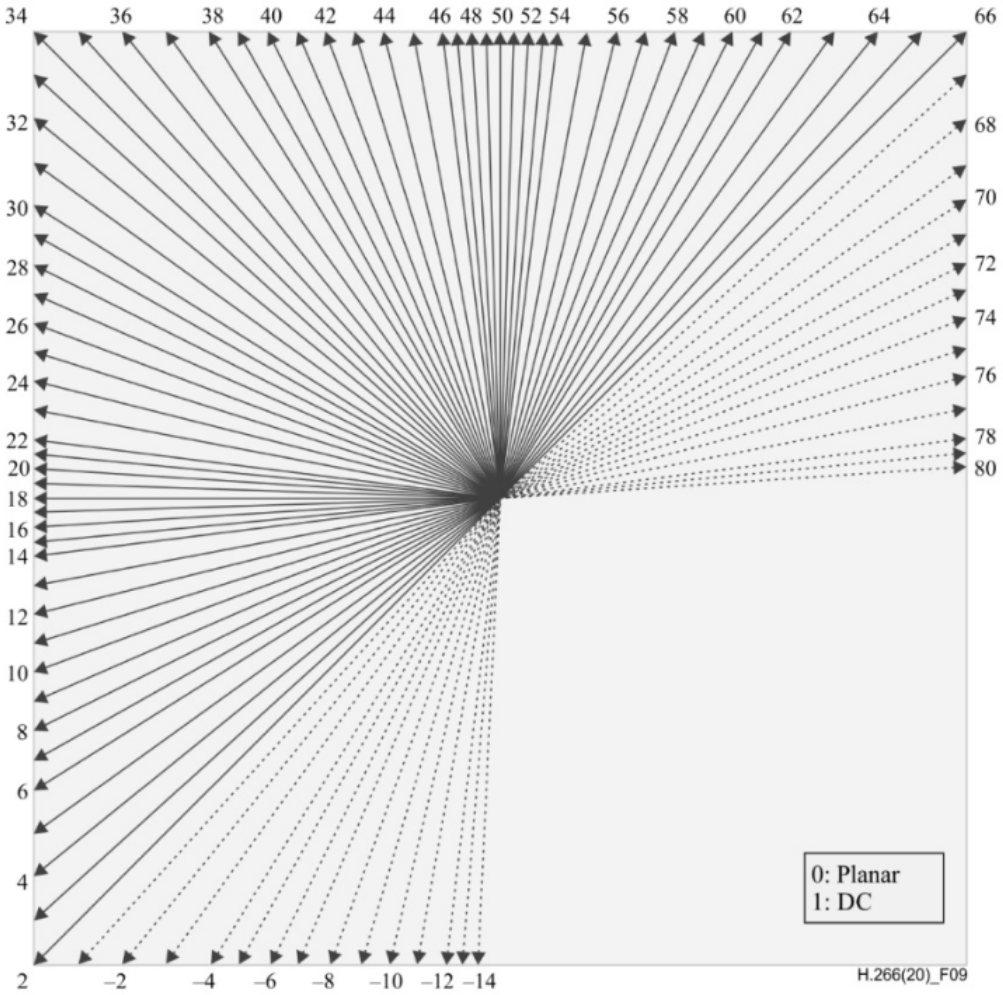


O autor

Quando um modo tem ângulo diferente de zero, o valor inverso do ângulo `invAngle` é definido pela equação 3.9.

$$invAngle = Round(\frac{512 * 32}{intraPredAngle}) \tag{3.9}$$

Figura 3.3: Modos e Ângulos da predição intra angular



(ITU-T; ISO/IEC, 2020)

Tabela 3.1: Tabela de Modos e seus respectivos Ângulos

predModeIntra	**−14**	**−13**	**−12**	**−11**	**−10**	**−9**	**−8**	**−7**	**−6**	**−5**	**−4**	**−3**	**−2**	**−1**	**2**	**3**	**4**
intraPredAngle	512	341	256	171	128	102	86	73	64	57	51	45	39	35	32	29	26
predModeIntra	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**
intraPredAngle	23	20	18	16	14	12	10	8	6	4	3	2	1	0	−1	−2	−3
predModeIntra	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**	**34**	**35**	**36**	**37**	**38**
intraPredAngle	−4	−6	−8	−10	−12	−14	−16	−18	−20	−23	−26	−29	−32	−29	−26	−23	−20
predModeIntra	**39**	**40**	**41**	**42**	**43**	**44**	**45**	**46**	**47**	**48**	**49**	**50**	**51**	**52**	**53**	**54**	**55**
intraPredAngle	−18	−16	−14	−12	−10	−8	−6	−4	−3	−2	−1	0	1	2	3	4	6
predModeIntra	**56**	**57**	**58**	**59**	**60**	**61**	**62**	**63**	**64**	**65**	**66**	**67**	**68**	**69**	**70**	**71**	**72**
intraPredAngle	8	10	12	14	16	18	20	23	26	29	32	35	39	45	51	57	64
predModeIntra	**73**	**74**	**75**	**76**	**77**	**78**	**79**	**80**									
intraPredAngle	73	86	102	128	171	256	341	512									

(ITU-T; ISO/IEC, 2020)

3.5.1 Seleção das Amostras de Referência para cada Modo

Como modos diferentes possuem direções diferentes, as amostras de referência também serão diferentes de modo para modo. Elas serão representadas pelo vetor `ref` que é definido pelas equações de 3.10 até 3.13, para modos maiores ou igual a 34, e de 3.14 até 3.17 para modos menores.

Para modos maiores ou igual a 34:

$$ref[x] = p[-1 + x][-1],\ x = 0...nTbW + 1 \tag{3.10}$$

Para intraPredAngle < 0 :

$$ref[x] = p[-1][-1 + Min((x * invAngle + 256) >> 9, nTbH)],\ x = -nTbH... - 1 \tag{3.11}$$

Do contrário :

$$ref[x] = p[-1 + x][-1],\ x = -nTbW... - 1 \tag{3.12}$$

$$ref[refW + 1] = p[-1 + refW][-1] \tag{3.13}$$

Para modos menores que 34:

$$ref[x] = p[-1][-1 + x],\ x = 0...nTbH + 1 \tag{3.14}$$

Para intraPredAngle < 0 :

$$ref[x] = p[-1 + Min((x * invAngle + 256) >> 9, nTbW)][-1],\ x = -nTbW... - 1 \tag{3.15}$$

Do contrário :

$$ref[x] = p[-1][-1 + x],\ x = -nTbH... - 1 \tag{3.16}$$

$$ref[refH + 1] = p[-1][-1 + refH] \tag{3.17}$$

Analisando as equações, podemos inferir como é feita a distribuição das amostras de referencia entre os diferentes modos. Podemos ver pelas Equações 3.11 e 3.15 que aqueles que estão no intervalo de 18 a 50, que possuem ângulo menor ou igual a zero, utilizam parte das amostras de referência horizontais e verticais, enquanto que modos maiores que 50 utilizam apenas as amostras horizontais, como mostram as Equações 3.12 e 3.13, e por fim, modos menores que 18 utilizam apenas amostras de referência verticais, como mostram as Equações 3.16 e 3.17. Para modos maiores ou igual a 34, índices positivos do vetor indicam amostras de referência horizontais, enquanto que valores negativos representam amostras verticais e, para menores que 34, o contrário.

3.5.2 Escolha do Filtro

Para uma amostra de predição, a predição angular de um modo acontecerá através de um filtro de interpolação em seu vetor de referência, que pode ser um filtro DCTIF (`fC`) ou SIF (`fG`). A escolha de qual filtro está associada a variável filterFlag, quando ela tiver o valor 0, DCTIF é escolhido, do contrário, SIF. A seleção do valor de `filterFlag` é definida no algoritmo:

Entrada: predModeIntra, intraHorVerDistThres []
Saída: filterFlag
se predModeIntra $\in \{2,34,66\}$ **então**
filterFlag $\leftarrow$ 0
senão
minDistVerHor $\leftarrow$ Min(Abs(predModeIntra - 50), Abs(predModeIntra - 18))
nTbS $\leftarrow (Log2(nTbW) + Log2(nTbH)) >> 1$
se minDistVerHor > intraHorVerDistThres[nTbS] **então**
filterFlag $\leftarrow$ 1
senão
filterFlag $\leftarrow$ 0
fim se
fim se

A Tabela 3.2 mostra os valores do vetor `intraHorVerDistThres` para diferentes valores de `ntbs`.

Tabela 3.2: Especificação intraHorVerDistThres[] para diferentes valores de nTbS

ntbs	*2*	*3*	*4*	*5*	*6*
intraHorVerDistThres[nTbS]	24	14	2	0	0

(ITU-T; ISO/IEC, 2020)

3.5.3 Predição Intra para Luminância

Por fim, as amostras da predição para a luminância são definidas pelas Equações 3.20 e 3.23. A variável de índice `iIdx`, utilizada na escolha do índice das amostras de referência multiplicadas são definidas pelas pelas Equações 3.18 e 3.21. A matriz `fT` indica o filtro de interpolação selecionado, a partir do valor de `filterFlag`, e os coeficientes do filtro, utilizando o fator de multiplicação `iFact` definido nas Equações 3.19 e 3.22. A Equação 3.24 define a relação de `fT` com `filterFlag` e `iFact` e a Tabela 3.3 mostra os valores de coeficientes para o filtro `fC`(SIF) e `fG`(DCTIF) a partir do fator de multiplicação.

Para modos maiores ou igual a 34:

$$iIdx = ((y+1) * intraPredAngle) >> 5 \tag{3.18}$$

$$iFact = ((y+1) * intraPredAngle)\&31 \tag{3.19}$$

$$predSamples[x][y] = Clip1(((\sum_{i=0}^{3} fT[i] * ref[x+iIdx+i]) + 32) >> 6) \tag{3.20}$$

Para modos menores que 34:

$$iIdx = ((x+1) * intraPredAngle) >> 5 \tag{3.21}$$

$$iFact = ((x+1) * intraPredAngle)\&31 \tag{3.22}$$

$$predSamples[x][y] = Clip1(((\sum_{i=0}^{3} fT[i] * ref[y+iIdx+i]) + 32) >> 6) \tag{3.23}$$

$$fT[j] = filterFlag ? fG[iFact][j] : fC[iFact][j],\ para\ j = 0..3 \tag{3.24}$$

Tabela 3.3: Tabela de coeficientes baseados no valor de iFact(p) para fC e fG

Fractional sample position p	fC interpolation filter coefficients				fG interpolation filter coefficients			
	$f_C[p][0]$	$f_C[p][1]$	$f_C[p][2]$	$f_C[p][3]$	$fG[p][0]$	$fG[p][1]$	$fG[p][2]$	$fG[p][3]$
0	0	64	0	0	16	32	16	0
1	−1	63	2	0	16	32	16	0
2	−2	62	4	0	15	31	17	1
3	−2	60	7	−1	15	31	17	1
4	−2	58	10	−2	14	30	18	2
5	−3	57	12	−2	14	30	18	2
6	−4	56	14	−2	13	29	19	3
7	−4	55	15	−2	13	29	19	3
8	−4	54	16	−2	12	28	20	4
9	−5	53	18	−2	12	28	20	4
10	−6	52	20	−2	11	27	21	5
11	−6	49	24	−3	11	27	21	5
12	−6	46	28	−4	10	26	22	6
13	−5	44	29	−4	10	26	22	6
14	−4	42	30	−4	9	25	23	7
15	−4	39	33	−4	9	25	23	7
16	−4	36	36	−4	8	24	24	8
17	−4	33	39	−4	8	24	24	8
18	−4	30	42	−4	7	23	25	9
19	−4	29	44	−5	7	23	25	9
20	−4	28	46	−6	6	22	26	10
21	−3	24	49	−6	6	22	26	10
22	−2	20	52	−6	5	21	27	11
23	−2	18	53	−5	5	21	27	11
24	−2	16	54	−4	4	20	28	12
25	−2	15	55	−4	4	20	28	12
26	−2	14	56	−4	3	19	29	13
27	−2	12	57	−3	3	19	29	13
28	−2	10	58	−2	2	18	30	14
29	−1	7	60	−2	2	18	30	14
30	0	4	62	−2	1	17	31	15
31	0	2	63	−1	1	17	31	15

(ITU-T; ISO/IEC, 2020)

3.6 Trabalhos Relacionados

Nesta seção apresentaremos dois trabalhos de arquiteturas para predição intra-quadro do VVC e suas principais diferenças. No final, analisaremos e classificaremos cada um deles a partir da generalização de seus módulos, e como isso influencia no desenvolvimento de cada uma das arquiteturas. Uma das classes servirá de base para o desenvolvimento da arquitetura apresentada no capitulo 4.

A primeira arquitetura estudada (AZGIN; KALALI; HAMZAOGLU, 2019) foi implementada para FPGAs da Xilinx para usufruir dos blocos DSPs integrados, pois estes podem fazer multiplicações de variáveis por diferentes constantes. Nela, blocos DSP datapath (DDP) programáveis são apresentados e esquematizados na figura 3.4. Eles utilizam dois blocos DSP e dois somadores, que implementam a equação 3.25, podendo calcular qualquer equação intra angular ao alterar as entras A, B, C, D e ExtraTerm. Desta forma, cada DDP implementa equações intra angular utilizando multiplicações ao invés de somas e deslocamentos, pois, no caso destas placas, é mais eficiente. Estes blocos DSP são organizados de modo a executar até 1040 equações, reduzindo para 405 utilizando um buffer para armazenar os resultados e reutiliza-los.

A arquitetura apresenta é relativamente simples em relação ao *datapath*, que é composto de múltiplos DDPs em paralelo. Porém, o controle se torna mais complexo, já que é ele quem deve tomar as decisões de quais equações devem ser resolvidas por cada modo e fazer a administração do buffer de reutilização.

$$P = (B1 * (D1 - A1) + C1) + (B2 * (D2 - A2) + C2) + ExtraTerm \tag{3.25}$$

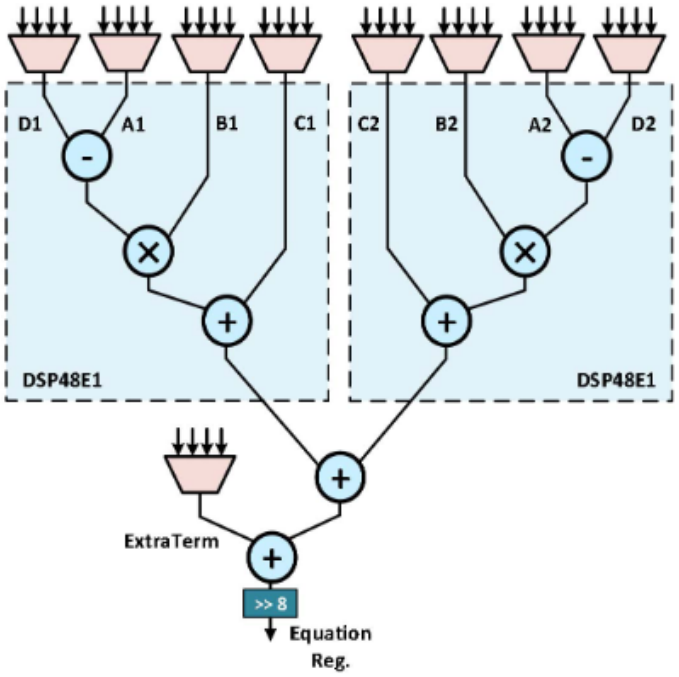
A segunda arquitetura (BORGES et al., 2023) foi desenvolvida para ASIC para a predição intra da crominância, que utiliza as Equações 3.26 e 3.27, e tem filtros diferentes em relação a luminância apresentada na Seção 3.5, mas com os valores de iIdx e iFact os mesmos.

$$\text{predSamples[x][y] = ((32 - iFact) *ref[x + iIdx + 1] + iFact * ref[x + iIdx + 2] + 16) » 5} \tag{3.26}$$

$$\text{predSamples[x][y] = ((32 - iFact) *ref[y + iIdx + 1] + iFact * ref[y + iIdx + 2] + 16) » 5} \tag{3.27}$$

No artigo (BORGES et al., 2023), é apresentada uma heurística de simplificação para a predição intra angular, com o intuito de diminuir o custo de requisitos não funci-

Figura 3.4: Bloco DDP



(AZGIN; KALALI; HAMZAOGLU, 2019)

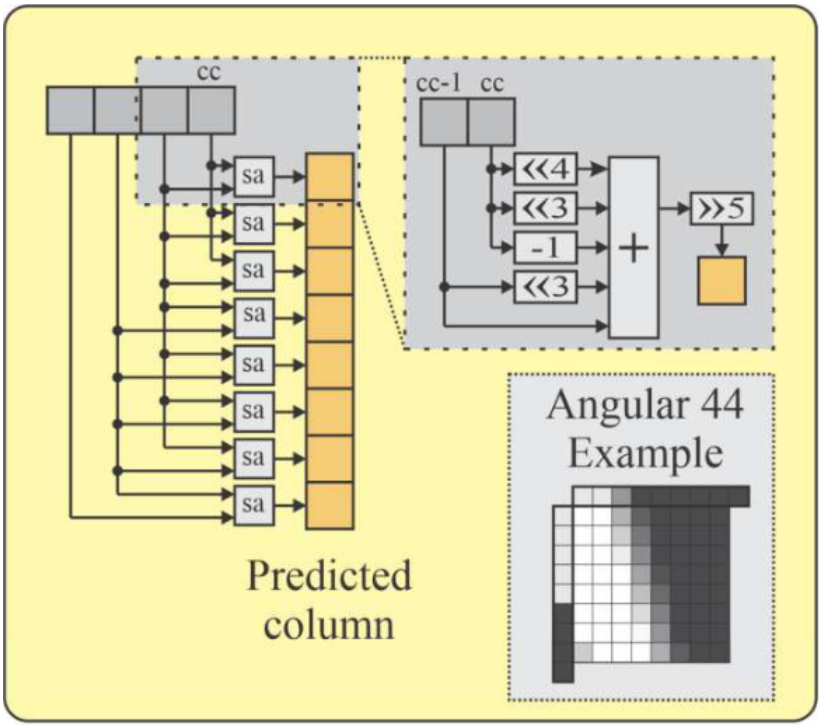
onais em *hardware*, como consumo de energia e área de circuito. Ela se baseia em um estudo estatístico que seleciona e implementa apenas os modos mais frequentemente escolhidos pelo codificador. No total, 18 modos foram escolhidos, sendo eles os modos Planar, DC, Angular 2, 3, 10, 18, 23, 26, 30, 33, 34, 35, 43, 46, 49, 50 e 54. Os resultados foram de uma diminuição de 60% no tempo de codificação para um aumento de 8.62% taxa de *bits* para manter a mesma qualidade de imagem.

A arquitetura apresentada no artigo (BORGES et al., 2023) tem módulos especializados na resolução de cada modo selecionados pela heurística e implementa os modos planar e DC, diferente do primeiro artigo estudado que implementa apenas modos angulares. A Figura 3.5 mostra a arquitetura do modo 44 angular para a predição da componente crominância, que utiliza somas e deslocamentos no lugar de multiplicadores.

O *datapath* inclui todos os modos da heurística realizados de forma paralela. Em relação a primeira arquitetura estudada, o controle é bem mais simples, enquanto que o *datapath* é mais complexo, onde os módulos são especializados na resolução das equações de um único modo de predição, implementando somas e deslocamentos específicos para aquele modo.

Das duas arquiteturas apresentadas na Seção 3.6, a que mais se enquadra para o nosso propósito é a arquitetura com módulos especializados, por já utilizar um sistema

Figura 3.5: Modo 44 com somas e deslocamentos para predição da crominância



(BORGES et al., 2023)

de somas e deslocamentos para resolver os filtros de crominância. Além disso, também utilizamos a heurística apresentada para facilitar a implementação em código. Portanto, nossa arquitetura é inspirada na arquitetura apresentada em (BORGES et al., 2023) e apresentaremos uma comparação direta com ela no capítulo de resultados.

4 ARQUITETURA PROPOSTA E IMPLEMENTAÇÃO

4.1 Análise Teórica

4.1.1 Formas de Predição

Durante o processo de predição, os módulos são responsáveis por calcular todas equações de predição em um número de ciclos. Para um bloco de transformação de dimensões `nTbH`×`nTbW`, existem três principais formas de predição que podemos escolher:

1. Calcular uma amostra por ciclo.
2. Calcular uma coluna, para modos maiores ou igual a 34, ou linha, para modos menores que 34, de tamanho N por ciclo.
3. Calcular um bloco de dimensões $N \times M$ por ciclo.

Como utilizaremos blocos MCM dentro de cada modo, o ideal é que várias equações sejam calculadas ao mesmo tempo, para que múltiplos coeficientes possam fazer parte dos blocos e aproveitar da heurística de reuso dele. Portanto, a opção 1 não será utilizada.

Entre as opção 2 e 3, a decisão se torna mais complicada, já que temos diferentes dimensões para o tamanho de linhas, colunas e blocos. Diferentes configurações podem resultar em diferentes níveis de reuso de blocos MCM. Na Tabela 4.1, podemos ver a relação entre as multiplicações das referências -1, 0, 1 e 2 pelos seus respectivos coeficientes do filtro `fG` para o modo 26.

Para calcularmos 4x4 equações em um único ciclo, o bloco MCM de ref[0] precisa multiplicar a referência pelos coeficientes fG[20][1], fG[8][1], fG[16][2], fG[20][0], fG[8][0], fG[28][1], fG[16][1]. Porém, podemos ver que estes coeficientes se repetem em outras referências quando o valor de y é incrementado. Isso ocorre porque a variável y não faz parte do cálculo de `iFact` e `iIdx`, responsáveis por variar os coeficientes e referências (ler 3.22 e 3.21), mas incrementa as variáveis de referência multiplicando (ler equação 3.23). Em outras palavras, podemos reutilizar todos o blocos MCM da linha predSample[0][0], predSample[1][0], predSample[2][0] e predSample[3][0] para calcular novas equações na linha predSample[0][1], predSample[1][1], predSample[2][1] e predSample[3][1] apenas fazendo um remapeamento das variáveis de referência.

Este fenômeno também acontece com linhas e colunas, como veremos na próxima

seção, mas é muito menos frequente do que para blocos $N \times M$. Portanto, para garantir uma quantidade maior de reuso de blocos MCM, os módulos calculam uma linha ou coluna de tamanho N por ciclo.

Tabela 4.1: Relações entre referências do modo 26

Amostra de Predição	iIdx	iFact	ref[-1]	ref[0]	ref[1]	ref[2]
predSample[0][0]	-1	20	fG[20][0]	fG[20][1]	fG[20][2]	fG[20][3]
predSample[1][0]	-1	8	fG[8][0]	fG[8][1]	fG[8][2]	fG[8][3]
predSample[2][0]	-2	28	fG[28][1]	fG[28][2]	fG[28][3]	Nulo
predSample[3][0]	-2	16	fG[16][1]	fG[16][2]	fG[16][3]	Nulo
predSample[0][1]	-1	20	Nulo	fG[20][0]	fG[20][1]	fG[20][2]
predSample[1][1]	-1	8	Nulo	fG[8][0]	fG[8][1]	fG[8][2]
predSample[2][1]	-2	28	fG[28][0]	fG[28][1]	fG[28][2]	fG[28][3]
predSample[3][1]	-2	16	fG[16][0]	fG[16][1]	fG[16][2]	fG[16][3]

O autor

4.1.2 Estados de Predição

Todo modo angular varia seus valores `iFact` e `iIdx` através das equações definidas em 3.5.3. Para `iFact`, esta variação eventualmente acaba no valor 0 e depois volta para o valor inicial (de predSample[0][0]), repetindo todos os valores anteriores, pois a operação "AND 31"faz com que qualquer valor fique no intervalo de 0 a 31. Para `iIdx`, o valor não faz parte de um intervalo específico, mas repete sua variação toda vez que `iFact` tem valor 0.

Na Tabela 4.2, podemos visualizar esta variação para os modos 26 e 12. Para o modo 12, a variação de iFact e iIdx recomeça na metade do recomeço de 26. Se escolheremos calcular uma linha de tamanho N igual a 4 em um único ciclo, precisaremos de um bloco MCM para a referência ref[0] e ref[1], no próximo ciclo, podemos reutilizar os dois blocos para a referência ref[1] e ref[2]. Este valor de N é o ideal para o modo 12, já que podemos calcular todas amostras com apenas estes dois blocos MCM. Porém, ainda precisaremos dos blocos MCM para as referências ref[-1], ref[-2] e ref[-3] para o modo 26, alternando a utilização delas em cada ciclo e gerando ociosidade. Se escolhermos calcular uma linha de tamanho N igual a 8, todas as referências serão utilizadas no modo 26 sem ter ociosidade, porém, o modo 12 precisará de um bloco MCM extra, já que agora devem ser utilizadas as referências ref[0], ref[1] e ref[2]. Neste caso, o valor de N é o

Tabela 4.2: Variação de iFact e iIdx para o modo 26 e 12

Amostra de Predição	iIdx 26	iFact 26	iIdx 12	iFact 12
predSample[0][0]	-1	20	0	8
predSample[1][0]	-1	8	0	16
predSample[2][0]	-2	28	0	24
predSample[3][0]	-2	16	1	0
predSample[4][0]	-2	4	1	8
predSample[5][0]	-3	24	1	16
predSample[6][0]	-3	12	1	24
predSample[7][0]	-3	0	2	0
predSample[8][0]	-4	20	2	8
predSample[9][0]	-4	8	2	16
predSample[10][0]	-5	28	2	24
predSample[11][0]	-5	16	3	0

O autor

ideal para o modo 26.

As Figura 4.1 e 4.2 ilustram os dois casos. Na primeira figura, teremos dois estados que fazem a seleção do multiplexador e alternam entre si, enquanto que na segunda, teremos apenas um estado que representado todo o módulo. O valor ref[0] ou ref[1], na primeira figura, é um mapeamento feito pelo *datapath* e não é um controle feito pelo modulo em si, portanto, não precisa de multiplexador e pode ser considerado apenas um único estado.

Nas Equações 4.2 e 4.3 temos definidos matematicamente os valores de `iFact` em 4.2 e 4.3, para para modos angulares menores que 34 e maiores ou igual a 34 respectivamente. O conjunto S é o nosso conjunto de estados com quantidade variada, pois dependendo do modulo, pode haver um número diferente de estados. Por exemplo, na Figura 4.1 o modulo que resolve o modo 26 tem dois estados enquanto que o modulo que resolve o 12 tem apenas um. Além disso, estados replicados são considerados como um único, já que S é um conjunto.

$$\forall y \in \{0, 1, 2, ...nTbH - 1\}, \forall x \in \{0, 1, 2, ...nTbW - 1\}, \tag{4.1}$$

$$iFact(x) \in si, i \in I \, e \, i = x/N \tag{4.2}$$

$$iFact(y) \in si, i \in I \, e \, i = y/N \tag{4.3}$$

$$S = \{s0, s1, ..., s(nTbW - 1/N)\} \tag{4.4}$$

$$\tag{4.5}$$

Onde:

1. $nTbW \in I$ sendo o comprimento do bloco de transformação
2. $nTbH \in I$ sendo a altura do bloco de transformação
3. $iFact(x)$ sendo uma função definida pela equação 3.21
4. $iFact(y)$ sendo uma função definida pela equação 3.18
5. $N \in I$ sendo o tamanho dos estados
6. $s \subset I$ sendo um estado que representa um modo angular
7. S sendo o conjunto de estados que representa um modo angular

A seguir, iremos definir um algoritmo para a implementação dos blocos MCM de um modo baseado em estados. O conjunt $blocos\_MCM$ envolve todos os blocos MCM para cada referência, $iIdx()$ é a função que calcula o valor `iIdx` de uma amostra de predição para aquele modo e $C_{k,\,ref}$ é o conjunto de constantes para uma dada referência de entrada ref de um determinado estado k.

Entrada: N, S

Saída: blocos_MCM

$blocos\_MCM \leftarrow \{\}$

$n \leftarrow 0$

$k \leftarrow 0$

para $s \in S$ **faça**

$C_{k,\,iIdx(0)} \leftarrow \{\},\ C_{k,\,iIdx(1)} \leftarrow \{\},\ ...\ C_{k,\,iIdx(N)+2} \leftarrow \{\},\ C_{k,\,iIdx(N)+3} \leftarrow \{\}$

para $coeficiente \in s,\ indice \in t$ **faça**

para $i \in \{0,1,2,3\}$ **faça**

$C_{k,\,iIdx(n)\,+\,i}\ \cup\ fT[coeficiente][i]$

fim para

$n \leftarrow n + 1$

fim para

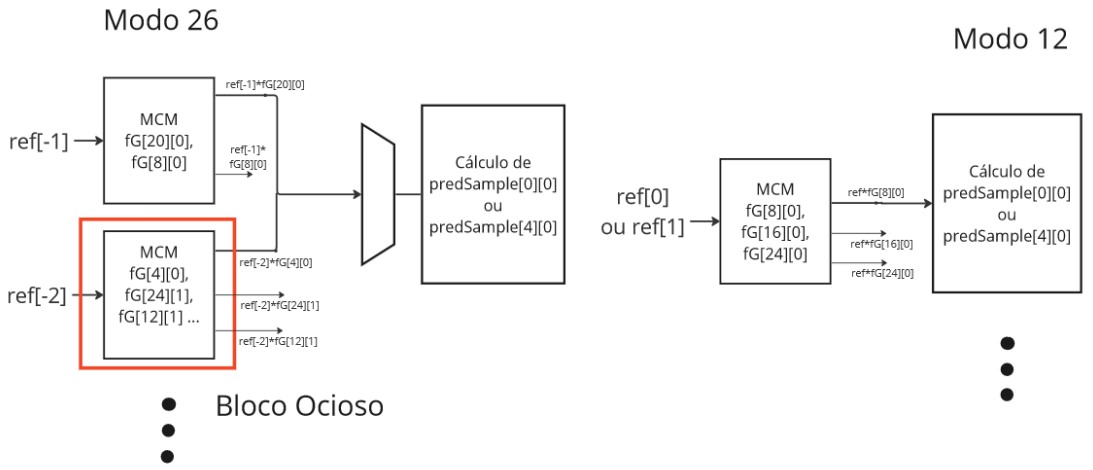
$k \leftarrow k + 1$

$blocos\_MCM\ \cup\ C_{k,\,iIdx(0)}\ \cup C_{k,\,iIdx(1)}\ \cup\ ...\ C_{k,\,iIdx(N)+2}\ \cup\ C_{k,\,iIdx(N)+3}$

fim para

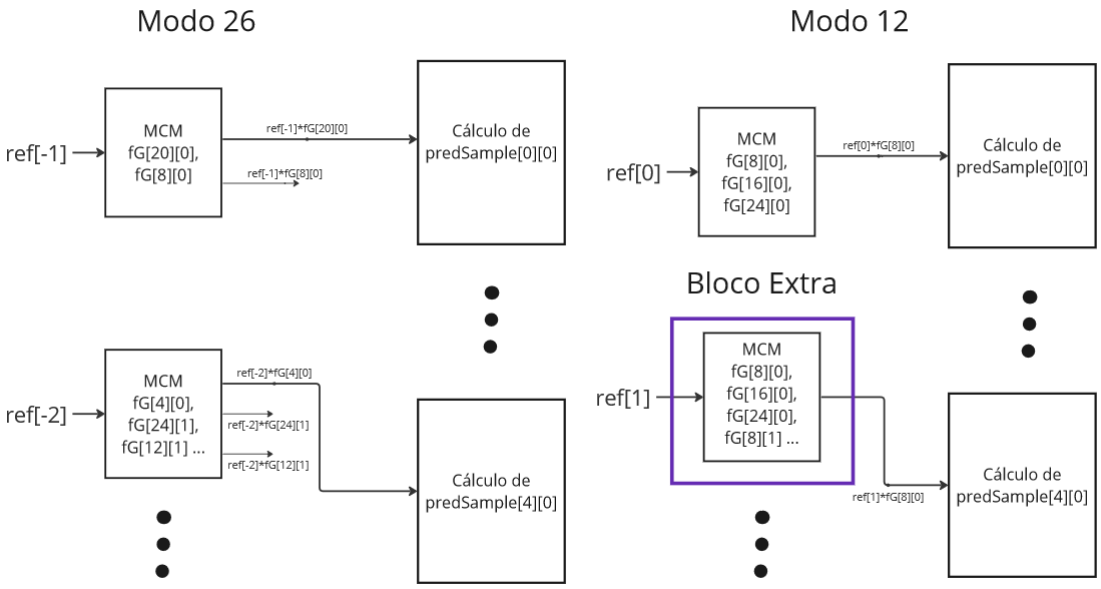
fim algoritmo

Figura 4.1: Blocos MCM para N = 4



O autor

Figura 4.2: Blocos MCM para N = 8



O autor

4.2 Valores de Variáveis Escolhidos

Com os conceitos teóricos já bem definidos, antes de irmos para a parte de arquitetura e implementação, precisamos definir valores para todas as variáveis introduzidas até agora. Primeiramente, definiremos o tamanho do bloco de transformação como sendo unicamente 32×32. Definindo o tamanho do bloco de transformação como constante, podemos converter todas as operações que eram unicamente dependentes dele como constantes também, facilitando a implementação dos modos de predição.

A segunda variável que precisamos definir é o tamanho da linha ou coluna de predição (N), introduzido na Subseção 4.1.1. Ela define a quantidade de amostras em paralelo que podemos calcular em um único ciclo, conforme mais, é mais desejável.

Como vimos na Seção 4.1.2, cada modo tem um valor ideal de N, porém, definimos que N terá o mesmo valor para todos estados, para que todos finalizem a predição de forma sincronizada.

Portanto, aumentar o valor de N aumenta o número de estados replicados, que influencia no número de blocos MCM extras, e o maior número de coeficientes em um bloco MCM, que influencia no caminho crítico, onde nenhum dos dois casos sendo desejável. Porém, aumentar o valor de N diminui a quantidade de blocos MCM ociosos por ciclo, o que é mais desejável.

Por conta de não ter um valor ótimo para todos os casos, escolhemos mantar o valor médio $N = 16$. A Tabela 4.3 mostra os modos e seus respectivos número de estados para este valor.

As últimas variáveis que precisamos definir são os filtros para cada modo (ler 3.5.2). Para os modos 2 e 34, usaremos o filtro `fC` por definição do algoritmo. Como definimos o tamanho do bloco de transformação 32×32, o valor da variável `nTbS` será 5 e, portanto, a variável `intraHorVerDistThres[nTbS]` tem valor igual a 0. Portanto, os filtros 50 e 18 também terão filtro `fC`, já que sãos os únicos que mantém valor 0 no cálculo de `mistDistVerHor`, enquanto que todos os outros modos terão o filtro `fG`. A Tabela 4.4 resume essa escolha.

Tabela 4.3: Número de estados para cada modo

Modo	Estados
2	1
3	2
7	1
10	1
18	1
23	1
26	1
30	1
33	2
34	1
35	2
43	1
46	1
49	2
50	1
54	1

O autor

Tabela 4.4: Filtros escolhidos para cada modo

Modo	Filtro	Modo $\in \{2,34\}$	minDistVerHor > 0
2	DCTIF (fC)	Verdadeiro	Verdadeiro
3	SIF (fG)	Falso	Verdadeiro
7	SIF (fG)	Falso	Verdadeiro
10	SIF (fG)	Falso	Verdadeiro
18	DCTIF (fC)	Falso	Falso
23	SIF (fG)	Falso	Verdadeiro
26	SIF (fG)	Falso	Verdadeiro
30	SIF (fG)	Falso	Verdadeiro
33	SIF (fG)	Falso	Verdadeiro
34	DCTIF (fC)	Verdadeiro	Verdadeiro
35	SIF (fG)	Falso	Verdadeiro
43	SIF (fG)	Falso	Verdadeiro
46	SIF (fG)	Falso	Verdadeiro
49	SIF (fG)	Falso	Verdadeiro
50	DCTIF (fC)	Falso	Falso
54	SIF (fG)	Falso	Verdadeiro

O autor

4.3 Arquitetura e implementação

Nesta seção, apresentaremos a arquitetura desenvolvida a partir das métricas selecionadas. A arquitetura da suporte a amostras de luminância de 8 *bits* e realiza a predição de um bloco de transformação de dimensões 32×32, como explicado na Seção 4.2. Ela precisa de um ciclo inicial de escrita, para armazenar as amostras vindas dos blocos vizinhos em um registrador próprio, mais 64 ciclos para calcular todas as amostras de predição, para todos os 18 modos escolhidos.

Com o intuito de economizar espaço, a cada ciclo, 16 amostras de predição por modo, com exceção do DC que utiliza apenas uma, serão armazenadas em seus registradores. A cada ciclo novas amostras serão armazenadas, substituindo as antigas, e cabe ao codificador recuperá-las antes da substituição ocorrer.

A Figura 4.3 apresenta o bloco principal, composto de controle, caminho de dados e registradores. O registrador de amostra de referência é responsável por armazenar as referências vindas do codificador e tem um tamanho de 105 amostras, totalizando um tamanho de 840 *bits*. O registrador de amostras de predição tem um tamanho de 273 amostras, totalizando 2184 *bits*.

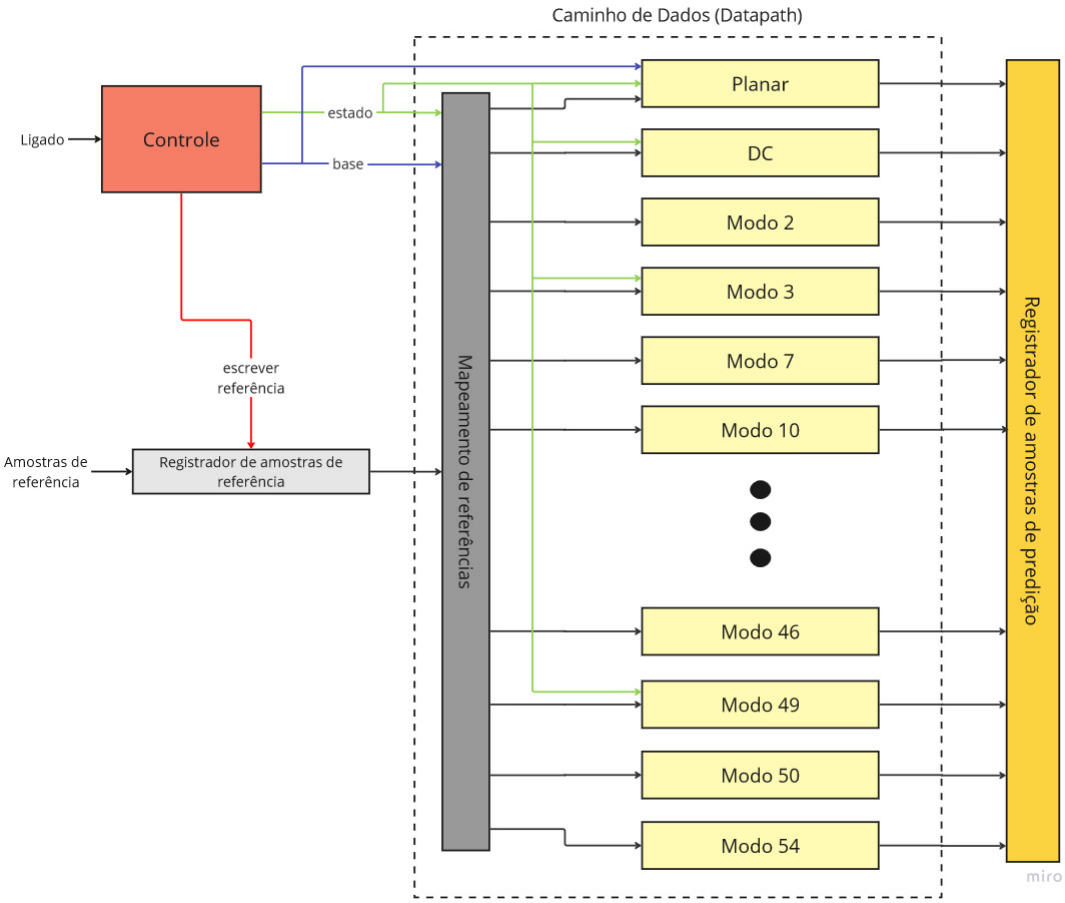
O caminho de dados tem os módulos especializados para a resolução dos 18 modos em paralelo. Cada módulo recebe seu conjunto de amostras de referência específico, vindas da unidade de mapeamento de referência, que é composta de uma série de multiplexadores que ligam amostras a cada módulo específico.

O controle é responsável por enviar os sinais de escrita de referência, para o registrador de referências armazenar as entradas apenas no primeiro ciclo, base que indica qual linha ou coluna a predição se encontra e estado, que indica para os modos de dois estados em qual estado eles se encontram. Relembrando que os modos e seus respectivos estados estão representados na Tabela 4.3.

Além das amostras de referência, nossa arquitetura recebe a *flag* `Ligado` para indicar que a predição intra-quadro foi iniciada, sendo encaminhada para a unidade de controle. A unidade é apresentada em detalhes na Figura 4.4 e é composta principalmente de quatro registradores. Quando a predição é iniciada, todos os registradores devem ser inicializados com o valor 0 utilizando a *flag* `RST`.

O registrador de estado de configuração indica aos registradores de amostras de referência que eles devem receber as amostras entradas, enviando a *flag* escrever referência com o valor de 1, no primeiro clico, mudando o valor da *flag* para 0 nos subsequentes.

Figura 4.3: Bloco principal



O autor

O registrador de Estado tem apenas 1 *bit* para a representação de dois estados, enquanto que o registrador de base tem 5 *bits* para representar as 32 linhas ou colunas em que a predição se encontra. A cada ciclo após o clico inicial de escrita, Estado é incrementado em 1, alternando entre o valor 0 e 1 a cada ciclo e incrementando o valor de Base em 1 quando ocorrer a troca de 1 para 0. Quando a Base alcança o valor 31, ocorre *overflow* e envia *carry out* para o registrador de Finalizado que indica que a predição acabou, parando o circuito.

As Figuras 4.5 e 4.6 apresentam os módulos para predição angular de modos com estado único e com dois estados respectivamente. Cada módulo recebe como entrada suas respectivas referências, que são então enviadas para seus respectivos blocos MCM implementados com o algoritmo da subsecção 4.1.2. O mapeamento de multiplicações interconecta as saídas dos blocos MCM com seus respectivos somadores. Caso seja um módulo de dois estados, este mapeamento conta com multiplexadores que selecionam quais saídas serão somadas a partir do estado atual.

A arquitetura do modo DC é apresentada na Figura 4.7. Ela consiste principalmente de, em cada ciclo, somar uma referência a um buffer que, no final dos 64 ciclos, terá o valor do somatório de todos as 32 amostras de referência horizontais e verticais. A unidade de mapeamento de referência será responsável por alternar entre referências verticais e horizontais a partir do estado atual. No final, o somatório é somado ao valor 32, deslocado para a direita por 6 (ver 3.4) e armazenado em único registrador de 8 *bits* que representa todas as amostras de predição DC.

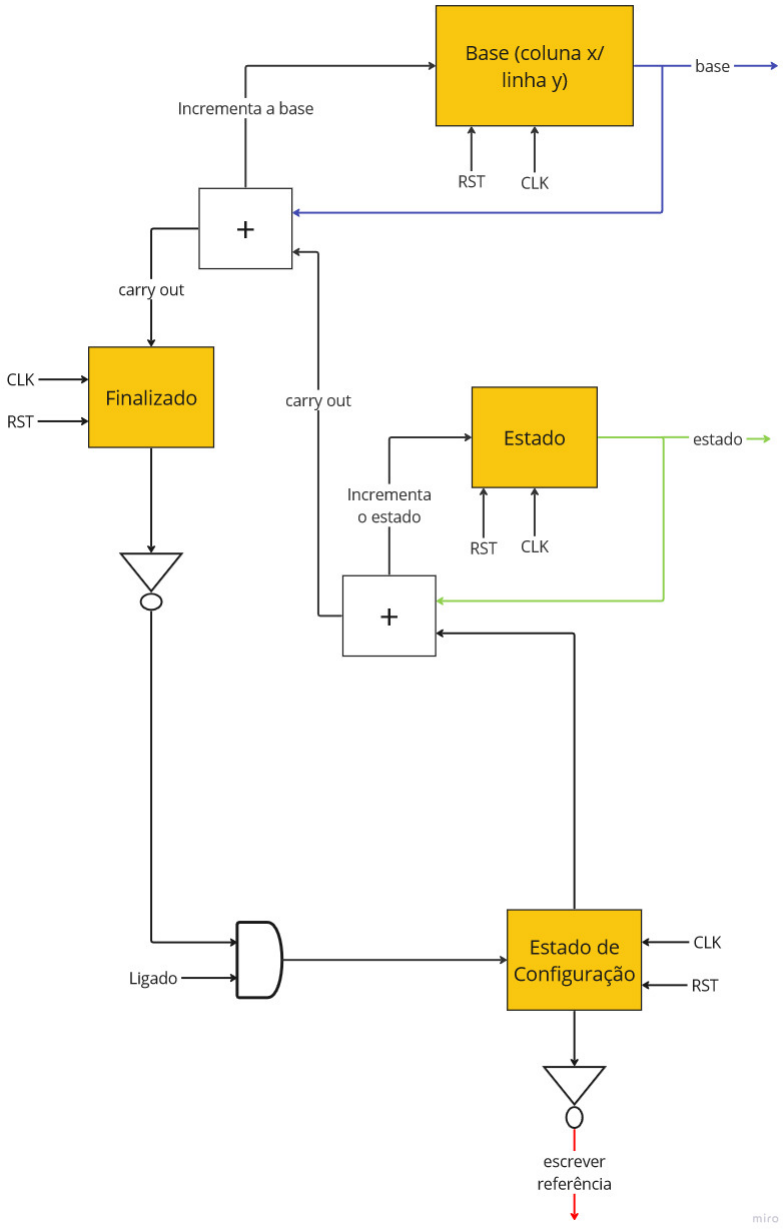
O módulo responsável por realizar o cálculo do modo planar está representado da Figura 4.8 e ele é composto principalmente das somadores e deslocadores e de outros dois módulos responsáveis por calcular os valores predH[x][y] e predV[x][y](ler 3.3), apresentados respectivamente nas Figuras 4.10 e 4.9.

O cálculo de predV consiste em dois blocos MCM, que alternam as referência de entrada conforme os estados 0 e 1 alternam. As multiplicações escolhidas também são alternadas conforme os estados, já que um realiza a predição das primeiras 16 linhas e o outro realiza para as últimas, em uma coluna x representada pela base.

O módulo predV retorna 16 valores que serão somados com seus respectivos 16 valores predH. Cada um dos valores predH é uma referência multiplicada por (31 - `base`) mais a referência p[32][-1] multiplicado por (`base` + 1).

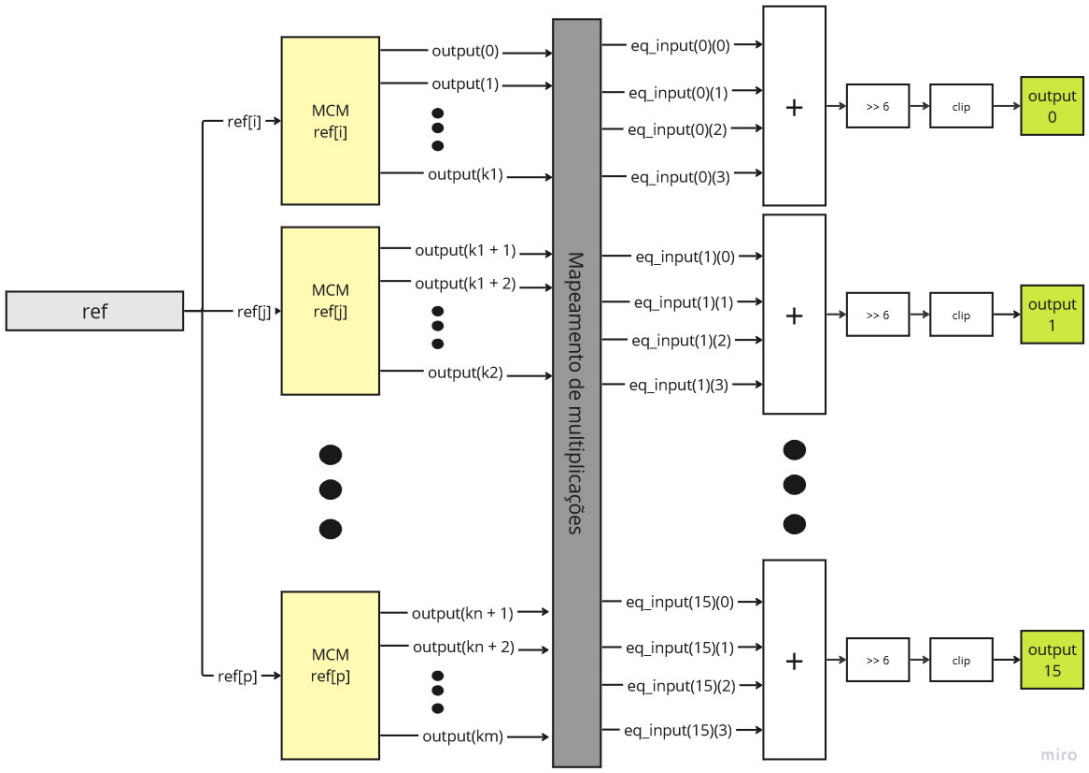
Para evitar a utilização de multiplicadores e blocos MCM, usaremos registradores que armazenam o valor das referências multiplicadas por 32, que pode ser feito como um

Figura 4.4: Controle



O autor

Figura 4.5: Angular de estado único



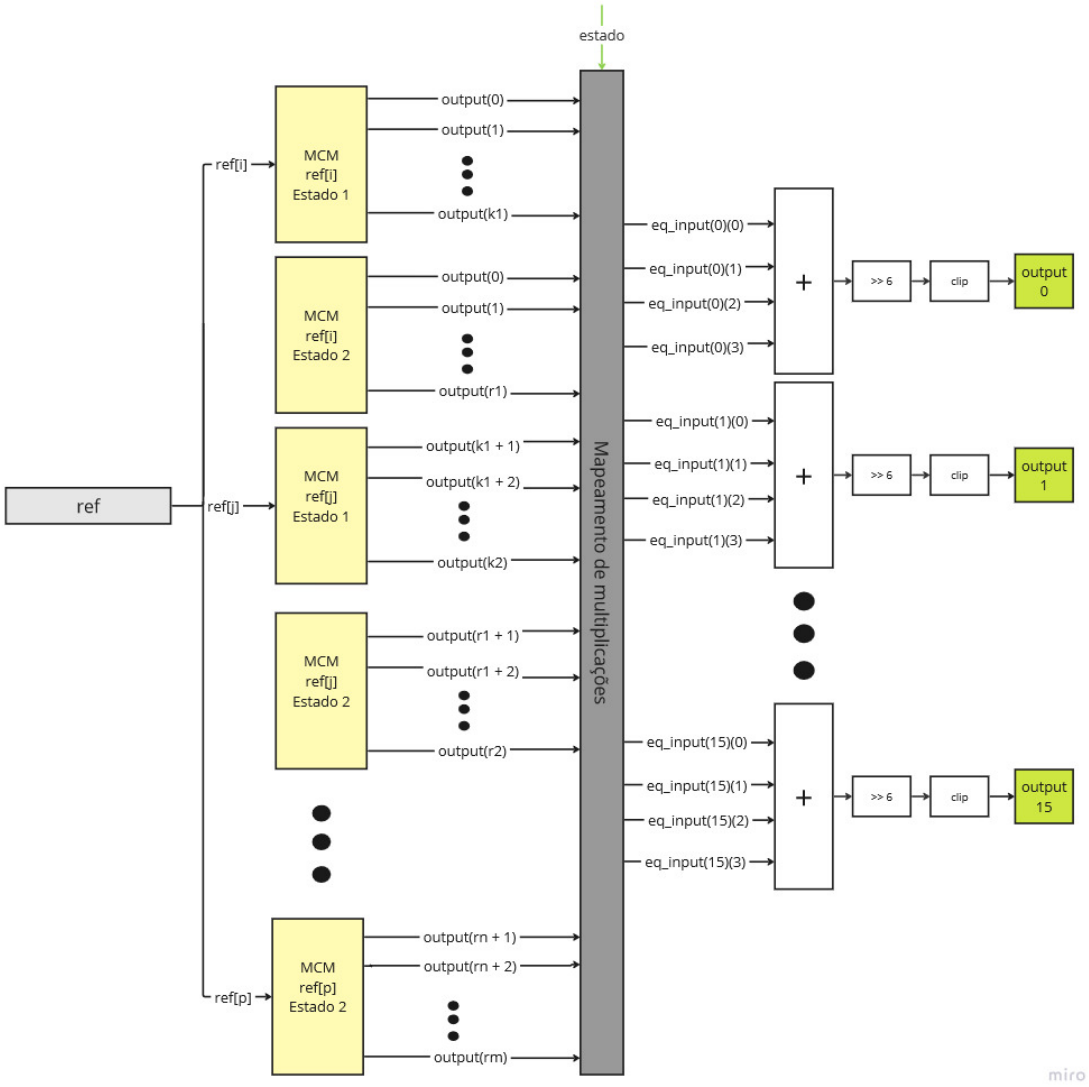
O autor

deslocamento para a esquerda por 5, que é decrementado pela própria referência a cada ciclo. Para p[32][-1] é feito o mesmo, mas inicializando o registrador com o valor da referência e depois incrementando em seu valor.

A implementação da arquitetura foi feita em VHDL com os módulos MCM gerados em verilog. Para gerar os módulos MCM, utilizamos o gerador *Spiral Multiplier Block Generator* (VORONENKO, 2006-2009), que oferece a opção de gerar o bloco como código verilog.

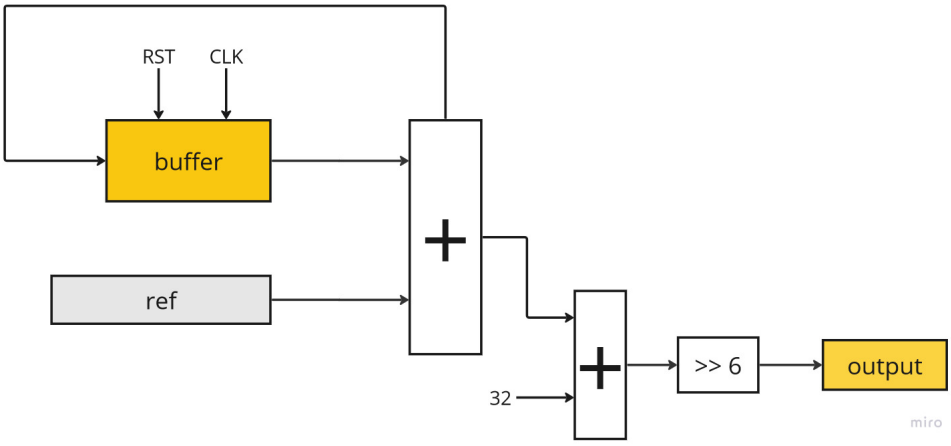
O testes foram feitos com um código de teste bench para toda a implementação da arquitetura, com as referências geradas aleatoriamente por um programa em python. Os resultados do *test bench* para cada modo foram escritos em arquivos de texto, que foram utilizados para comparar com as saídas de um programa em python que implementa todas as equações descritas no Capítulo 3.

Figura 4.6: Angular de estado duplo



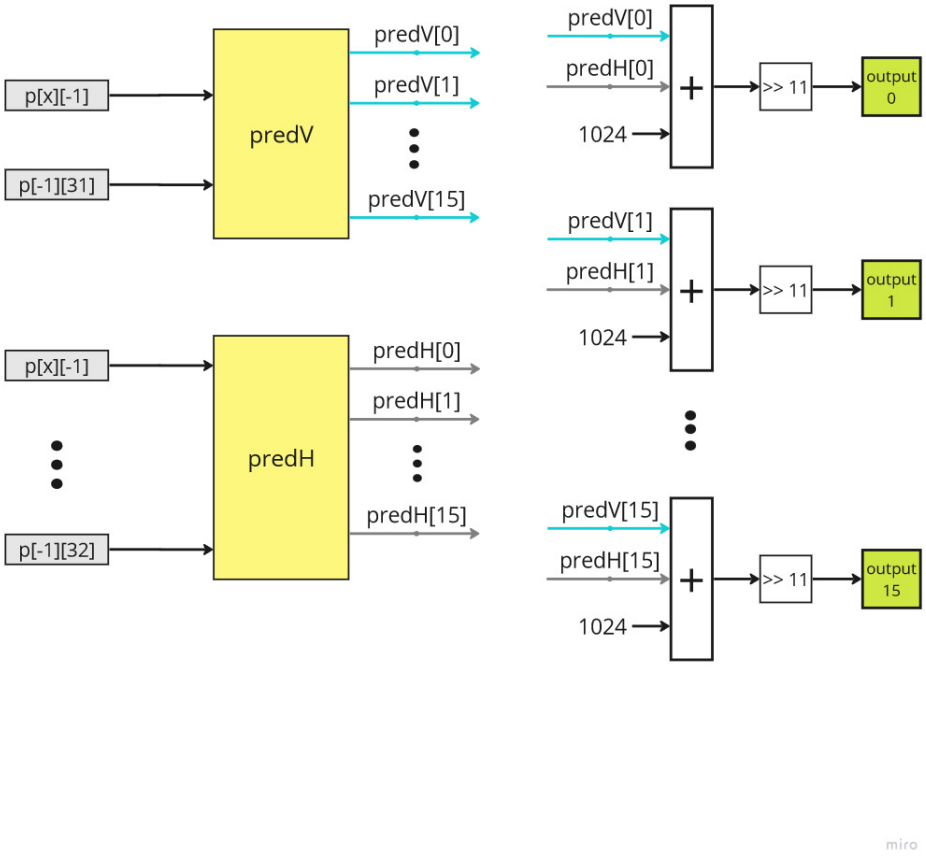
O autor

Figura 4.7: Modo DC



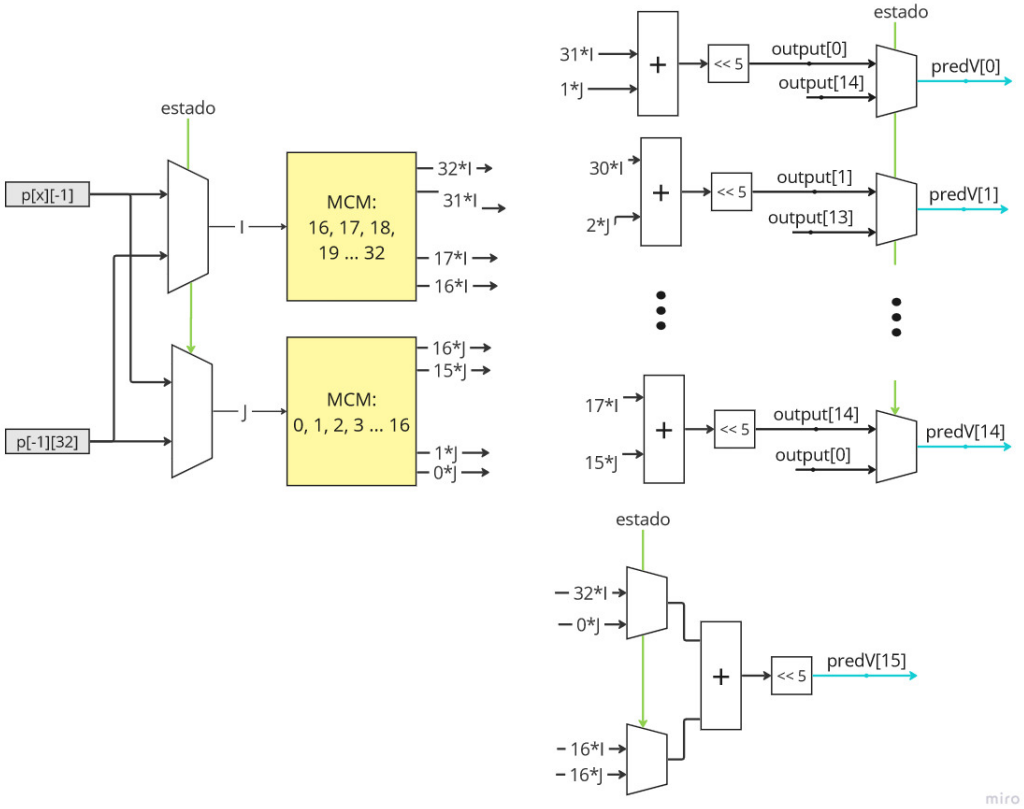
O autor

Figura 4.8: Modo planar



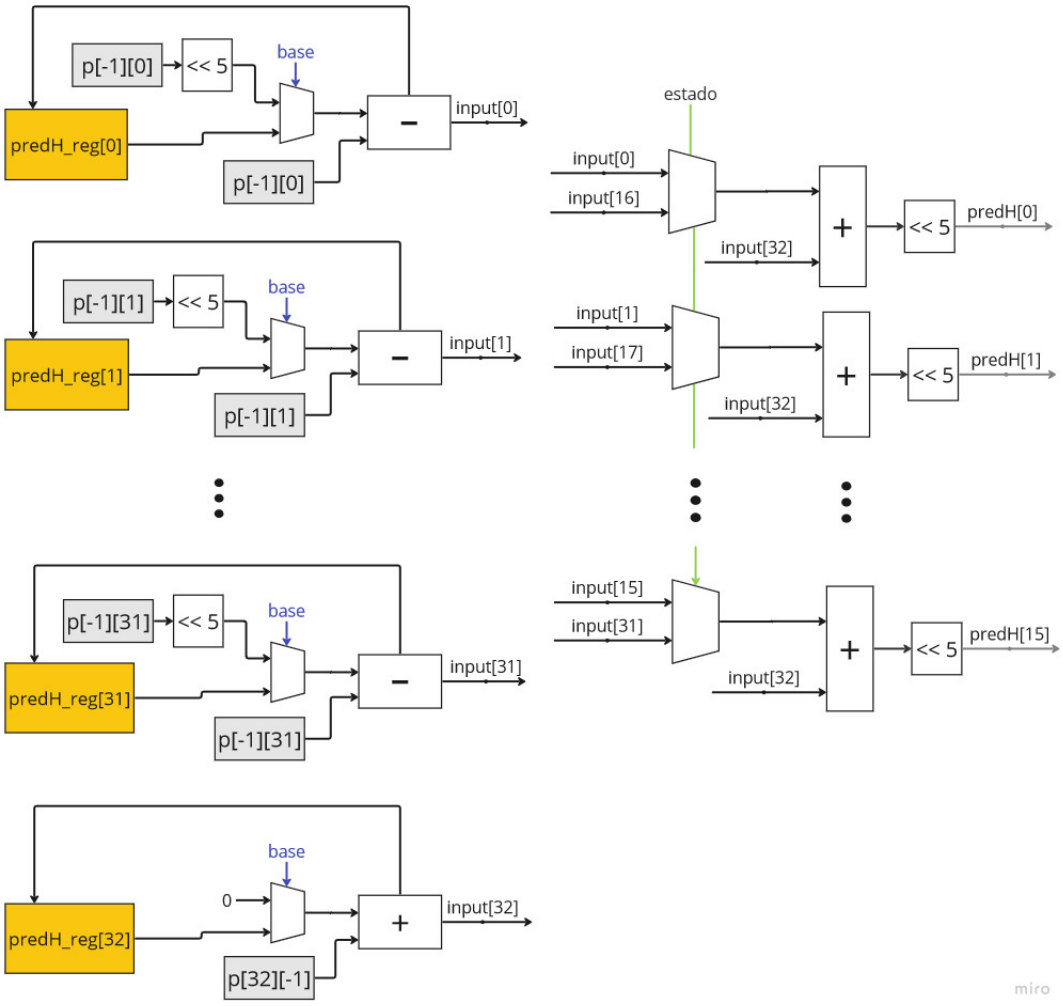
O autor

Figura 4.9: Componente para cálculo de predV



O autor

Figura 4.10: Componente para cálculo de predH



O autor

4.4 Resultados

Os resultados do *test bench* foram exatos com as saídas do programa para todos modos implementados, exceto para o modo planar que teve 939/1024 amostras corretas, todas as outras com um erro de apenas uma unidade de diferença. Nossa suspeita seja que, por conta da implementação utilizando registradores ao invés de multiplicações, um pequeno erro foi inserido no cálculo.

A síntese lógica foi feita utilizando a ferramenta Genus da Cadence, para a tecnologia XFAB 180 nm. Conseguimos alcançar uma frequência máxima de 74 MHz, uma área de 2624307 micrômetros e uma potência total de 96,8 mW. O tamanho de uma porta NAND2 para a biblioteca utilizada é de 32 micrômetros, desta forma podemos calcular a área em gate-equivalente, resultando em 82k.

Nossa arquitetura consegue fazer a predição de um bloco 32x32 em 65 ciclos. Logo, para um *frame* de qualidade Full HD (1920x1080) precisaremos de 132.600 ciclos, e para 30 *frames*, 3.978.000 ciclos. Então, para fazer uma predição de 30 frames/segundo para qualidade Full HD, precisaremos de 3.978.000 ciclos/segundo, que é aproximadamente 4 MHz. A Tabela mostra a frequência também para as qualidades QHD e UHD e para 60 fps.

Uma comparação dos resultados com as duas arquiteturas estudadas na Seção 3.6 e com uma arquitetura desenvolvida para HEVC é apresentada na Tabela 4.6.

Podemos ver que a frequência de operação para qualidade 1920x1080 a 30 fps é extremamente menor do que outros trabalhos na literatura. Isso acontece pois nossa arquitetura resolve múltiplas amostras paralelamente por ciclo mas apenas blocos de transformação de dimensão única, enquanto que outras arquiteturas resolvem amostras de forma serializada e múltiplos blocos de dimensões variadas simultaneamente.

Tabela 4.5: Frequência de operação para diferentes qualidades por fps

	Full HD	QHD	UHD
30 fps	4 MHz	7 MHz	16 MHz
60 fps	8 MHz	14 MHz	32 MHz
120 fps	16 MHz	28 MHz	64 MHz

O autor

Tabela 4.6: Comparação entre outros trabalhos na literatura

Trabalhos	Este trabalho	(BORGES et al., 2023)	(AZGIN; KALALI; HAMZAOGLU, 2019)	(KALALI; HAMZAOGLU, 2020)
Blocos suportados	32x32	4x4, 8x8, 16x16, 32x32 e 64x64	4x4, 8x8, 32x32	4x4, 8x8, 16x16, 32x32
Tipos de amostras	Luminância	Crominância	Luminância	Luminância
Quantidade de modos	18	18	65	35
Tecnologia	180 nm	40 nm	28 nm	90 nm
Área (gate-equivalente)	82k	1453k	Não especificado	3,2k
Frequência máxima	74 MHz	Não especificado	119 MHZ	333 MHZ
Frequência para 1920x1080 a 30 fps	4 MHz	76 MHz	119 MHZ	333 MHZ

O autor

5 CONCLUSÕES

A arquitetura desenvolvida consegue resolver corretamente todos os modos desenvolvidos, exceto o modo intra planar, ao qual foi introduzido um pequeno erro em alguns resultados, provavelmente fruto do cálculo de multiplicações utilizando registradores, somas e subtrações.

Ela resolve uma quantidade 16 amostras de predição por ciclo e por causa disso, junto com o fato de que ela foi desenvolvida apenas para blocos 32×32, faz com que a sua frequência para a codificação de um vídeo Full HD a 30 fps seja de apenas 4 MHz, um valor muito menor comparado com os outros apresentados na literatura, provavelmente relacionado ao fato de ser uma arquitetura especializada em apenas blocos de dimensão única. Não foi possível testar o BD-Rate da arquitetura, fazendo com que não consigamos avaliar a eficiência da arquitetura e da heurística implementada nela.

5.1 Trabalhos futuros

1. Desenvolver uma solução mais precisa para o modo planar.
2. Utilizar o *software* VTM para medirmos métricas relacionadas a compressão do vídeo.
3. Realizar a síntese física para obtermos dados mais precisos de área e potência.
4. Desenvolver variações na arquitetura e compara-las.

REFERÊNCIAS

AZGIN, H.; KALALI, E.; HAMZAOGLU, I. An efficient fpga implementation of versatile video coding intra prediction. In: **2019 22nd Euromicro Conference on Digital System Design (DSD)**. [S.l.: s.n.], 2019. p. 194–199.

BITMOVIN. **The 6th Annual Bitmovin Video Developer Report**. [S.l.: s.n.], 2022.

BORGES, V. et al. Efficient architecture for vvc angular intra prediction based on a hardware-friendly heuristic. In: **2023 IEEE 14th Latin America Symposium on Circuits and Systems (LASCAS)**. [S.l.: s.n.], 2023. p. 1–4.

BROSS, B. et al. Developments in international video coding standardization after AVC, with an overview of Versatile Video coding (VVC). **Proceedings of the IEEE**, v. 109, n. 9, p. 1463–1493, 2021.

DINIZ, C. M. et al. A real time h.264/avc intra frame prediction hardware architecture for hdtv 1080p video. In: **2009 IEEE International Conference on Multimedia and Expo**. [S.l.: s.n.], 2009. p. 1138–1141.

ISO. Coding of moving pictures and associated audio for digital storage media at up to about 1.5 mbits/s - part 2: Video. **ISO/IEC 11172-2 (MPEG-1)**, 1993.

ISO/IEC. Coding of audio-visual objects - part 2: Visual. **ISO/IEC 14496-2 (MPEG-4 Visual)**, 1999.

ISO/IEC, I.-T. e. Generic coding of moving pictures and associated audio information - part 2: Video. **ITU-T Recommendation H.262 e ISO/IEC 13818-2 (MPEG-2 Video)**, 1994.

ITU-T. Video codec for audiovisual services at 64 kbits/s. **ITU-T Recommendation H.261**, 1990.

ITU-T. Video coding for for low bit rate communication. **ITU-T Recommendation H.263 version 1**, 1995.

ITU-T; 1, I. J. Advanced video coding for generic audiovisual services. **ITU-T Recommendation H.264 and ISO/IEC 14496-10 (MPEG-4 AVC)**, 2003.

ITU-T; ISO/IEC. High Efficiency Video Coding. **ITU-T Recommendation H.265 e ISO/IEC 23008-2**, 2013.

ITU-T; ISO/IEC. Versatile Video Coding. **ITU-T Recommendation H.266 e ISO/IEC 23090-3)**, 2020.

KALALI, E.; HAMZAOGLU, I. An approximate hevc intra angular prediction hardware. **IEEE Access**, v. 8, p. 2599–2607, 2020.

KUMM, M.; VOLKOVA, A.; FILIP, S.-I. Design of optimal multiplierless fir filters with minimal number of adders. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 42, n. 2, p. 658–671, 2023.

MERCAT, A. et al. Comparative rate-distortion-complexity analysis of vvc and hevc video codecs. **IEEE Access**, v. 9, p. 67813–67828, 2021.

PFAFF, J. et al. Intra prediction and mode coding in vvc. **IEEE Transactions on Circuits and Systems for Video Technology**, v. 31, n. 10, p. 3834–3847, 2021.

SIQUEIRA, ; CORREA, G.; GRELLERT, M. Rate-distortion and complexity comparison of hevc and vvc video encoders. In: **2020 IEEE 11th Latin American Symposium on Circuits Systems (LASCAS)**. [S.l.: s.n.], 2020. p. 1–4.

VORONENKO, C. M. U. Y. **Multiplier Block Generator**. 2006–2009. <https://spiral.ece.cmu.edu/mcm/gen.html>.

VORONENKO, Y.; PüSCHEL, M. Multiplierless multiple constant multiplication. **ACM Transactions on Algorithms**, v. 3, 05 2007.