UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Vanio Rodrigues Filho

Fixed Search Patterns and VLSI Architecture for the Efficient Computation of the Versatile Video Coding Fractional Motion Estimation

Florianópolis
2022

Vanio Rodrigues Filho

Fixed Search Patterns and VLSI Architecture for the Efficient Computation of the Versatile Video Coding Fractional Motion Estimation

Dissertação submetida ao Programa de Pós-Graduação em Ciência da Computação para a obtenção do título de Mestre em Ciência da Computação.
Orientador: Prof. José Luís Almada Güntzel, Dr.

Florianópolis
2022

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Rodrigues Filho, Vanio
   Fixed Search Patterns and VLSI Architecture for the Efficient Computation of the Versatile Video Coding Fractional Motion Estimation / Vanio Rodrigues Filho ; orientador, José Luís Almada Güntzel, 2022.
   80 p.

   Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós-Graduação em Ciência da Computação, Florianópolis, 2022.

   Inclui referências.

   1. Ciência da Computação. 2. Video Coding. 3. Fractional Motion Estimation. 4. Hardware Architecture. I. Güntzel, José Luís Almada. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Ciência da Computação. III. Título.

Vanio Rodrigues Filho

Fixed Search Patterns and VLSI Architecture for the Efficient Computation of the Versatile Video Coding Fractional Motion Estimation

Florianópolis, 2022.

Aos meus pais, Tati, Duda e ao pessoal do ECL.

ACKNOWLEDGEMENTS

I would like to acknowledge my parents Vanio Rodrigues and Tania Pereira, as well as my sister Tatiani for supporting me on this indescribably long journey. I would also like to acknowledge my girlfriend Maria Eduarda for believing in me even when I did not. Third, I would like to acknowledge all my friends and colleagues at the ECL for all the unconditional help since my TCC. Especially for Ismael, Mateus and my advisor José Luís.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001

“It’s the questions we can’t answer that teach us the most. They teach us how to think. If you give a man an answer, all he gains is a little fact. But give him a question and he’ll look for his own answers”
PATRICK ROTHFUSS, ‘The Wise Man’s Fear’ (2011).

RESUMO

A codificação de vídeo é o núcleo de qualquer aplicação de vídeo, pois permite a compressão de vídeos, viabilizando seu armazenamento e transmissão através da web. A importância dos algoritmos de codificação de vídeo aumentou significativamente nestes dias de isolamento físico imposto pela pandemia do Covid-19, quando a videoconferência se tornou uma ferramenta muito poderosa para evitar a interrupção completa de atividades como trabalhos e estudos. A fim de permitir a compatibilidade entre os sistemas de codificação e decodificação, vários padrões têm sido desenvolvidos ao longo dos anos por agências internacionais de padronização, como ITU-T e ISO, com a colaboração da indústria e da academia. Finalizado em julho de 2020, o Versatile Video Coding (VVC) é o padrão de codificação de vídeo mais recente, tendo sido desenvolvido para melhorar a eficiência de codificação em mais de 40% para a mesma qualidade de imagem, quando comparado ao seu antecessor, o High Efficiency Video Coding (HEVC). No entanto, o aumento expressivo de tempo de codificação do VVC leva a uma maior dissipação de energia e dificulta a sua implementação em software para aplicações de tempo real. Portanto, é de extrema importância desenvolver técnicas para diminuir a complexidade das etapas computacionalmente mais intensivas deste novo padrão, mantendo, tanto quanto possível, suas melhorias de eficiência de codificação. Como geralmente ocorre quando um novo padrão de codificação de vídeo é lançado, as etapas mais intensivas, "e.g., a Fractional Motion Estimation (FME), são candidatas naturais a tais estudos, não apenas pela complexidade, mas também pela repetitividade intrínseca de operações. Em particular, quando se trata de dispositivos portáteis, codificadores de vídeo integrados em hardware que sejam energeticamente eficientes são indispensáveis para estender a vida útil da bateria do dispositivo. Dessa forma, este trabalho objetiva reduzir a complexidade da FME do VVC por meio da adoção de um padrão de busca fixo no projeto de uma arquitetura VLSI dedicada. Como primeiro passo, foi estimado o impacto da FME na eficiência de codificação do VVC em termos de BD-Rate executando o VVC Test Model (VTM) com a FME desabilitada. Em seguida, quatro padrões fixos de busca foram avaliados em termos de eficiência de codificação e recursos de hardware, sendo três deles propostos neste trabalho e o quarto encontrado na literatura. A eficiência de codificação foi avaliada através da implementação dos padrões dentro do VTM. Os recursos de hardware foram avaliados em termos de área e potência usando como base uma arquitetura estado da arte de hardware da FME. O padrão *Cross*, proposto neste trabalho, mostrou-se o de maior potencial de minimização de hardware com uma redução aceitável na eficiência de codificação e, portanto, foi selecionado para um projeto de hardware dedicado. A arquitetura projetada foi descrita em verilog e sintetizada usando o fluxo *standard cell* para a tecologia de 45nm. A área ocupada pela arquitetura desenvolvida é inferior a 41,4% da arquitetura base, com um potência dissipada total de apenas 28.9% em média. A arquitetura projetada também é capaz de comprimir vídeos em tempo real para resoluções de até 8K a 30 quadros por segundo, porém com um aumento de BD-Rate de 0,34% para configuração LD-P e 0,28% para configuração RA.

Palavras-chave: Codificação de vídeo. Estimação de Movimento Fracionária. Padrão de busca. Arquitetura VLSI. Versatile Video Coding.

RESUMO EXPANDIDO

Introdução

A codificação de vídeo é o núcleo de qualquer aplicação de vídeo, pois permite a compressão de vídeos, viabilizando seu armazenamento e transmissão através da web. A importância dos algoritmos de codificação de vídeo aumentou significativamente nestes dias de isolamento físico imposto pela pandemia do Covid-19, quando a videoconferência se tornou uma ferramenta muito poderosa para evitar a interrupção completa de atividades como trabalhos e estudos. A fim de permitir a compatibilidade entre os sistemas de codificação e decodificação, vários padrões têm sido desenvolvidos ao longo dos anos por agências internacionais de padronização, como ITU-T e ISO, com a colaboração da indústria e da academia. Finalizado em julho de 2020, o Versatile Video Coding (VVC) é o padrão de codificação de vídeo mais recente, tendo sido desenvolvido para melhorar a eficiência de codificação em mais de 40% para a mesma qualidade de imagem, quando comparado ao seu antecessor, o High Efficiency Video Coding (HEVC). No entanto, o aumento expressivo de tempo de codificação do VVC leva a uma maior dissipação de energia e dificulta a sua implementação em software para aplicações de tempo real. Portanto, é de extrema importância desenvolver técnicas para diminuir a complexidade das etapas computacionalmente mais intensivas deste novo padrão, mantendo, tanto quanto possível, suas melhorias de eficiência de codificação. Como geralmente ocorre quando um novo padrão de codificação de vídeo é lançado, as etapas mais intensivas, e.g., a Estimação de Movimento Fracionária (FME), são candidatas naturais a tais estudos, não apenas pela complexidade, mas também pela repetitividade intrínseca de operações.
A FME é uma ferramenta que busca refinar os resultados da Estimação de Movimento (ME) ao gerar pixels em posições fracionárias, ou seja, entre os pixels originalmente capturados. Ela é uma das ferramentas computacionalmente intensas do HEVC Test Model (HM), sendo responsável por 33,88% do tempo total de codificação, em média (BLASI et al., 2015). Quando está desabilitado em HM, a taxa de bits aumenta em média 7,01%, 6,27% e 16,33% para as sequências Ultra High Definition - 3840×2160 (UHD), Full HD - 1920×1080 (FHD) e High Definition - 1280×720 (HD), respectivamente, considerando a mesma qualidade de vídeo (BLASI et al., 2015). Soluções para mitigar este problema geralmente envolvem a simplificação dos filtros de interpolação, responsáveis por gerar artificialmente esses novos pixels, ou a limitação do número de candidatos pesquisados no processo (LV et al., 2014; HE et al., 2015; KALALI; HAMZAOGLU, 2018; LIM et al., 2016; DING; YE; WANG, 2015). O principal desafio do projeto de hardware da FME é lidar com dependências de dados durante a interpolação e a busca. Portanto, padrões alternativos de busca FME podem aliviar a quantidade de operações aritméticas necessárias, tanto na interpolação quanto na busca, ao limitar o escopo da FME a apenas alguns dos candidatos mais prováveis de serem escolhidos como referência para codificação.

Objetivos

O padrão VVC traz uma série de novas ferramentas de codificação que resultam em um aumento substancial da complexidade em relação aos padrões de codificação de vídeo predecessores. No entanto, de acordo com o conhecimento do autor, até agora nenhum estudo sobre como a FME contribui para a eficiência de codificação deste padrão recentemente concluído foi publicado e nenhum projeto de hardware dedicado foi proposto. Portanto, este trabalho traz cinco contribuições:

1. Avaliação quantitativa do impacto da eficiência de codificação da FME em vídeos codificados com o VVC.
2. Proposta de três padrões fixos (simplificados) de busca para a FME de VVC.
3. Avaliação da eficiência de codificação dos três padrões fixos propostos juntamente com trabalho correlato.
4. Estimativa da redução de hardware fornecida para cada padrão de busca quando comparado com uma arquitetura *baseline*.
5. Projeto e avaliação de uma arquitetura de hardware dedicado que implementa o padrão de busca fixo que leva (potencialmente) ao melhor *trade-off* entre minimização de consumo de energia e perda de eficiência de codificação, comparando-a com a arquitetura baseline.

Metodologia
A fim de alcançar a primeira e terceira contribuições, foram realizados experimentos com o software de referência VVC Test Model (VTM) v13.0 codificando sequências de vídeo do Common Test Conditions (CTC) (BOSSEN, 2012), de modo a avaliar a eficiência de codificação da FME . Para avaliar suas eficiências de codificação, os padrões simplificados foram implementados dentro do VTM (terceira contribuição). Para a segunda contribuição, os padrões propostos foram elaborados buscando reduzir o número de candidatos computados pela FME, analisando apenas os candidatos mais prováveis de serem escolhidos como referência. As características de uma arquitetura de hardware para FME também foram levadas em consideração na definição de cada padrão. Para chegar à quarta contribuição, foi realizada uma análise sobre a arquitetura baseline. Cada padrão foi avaliado para estimar a redução potencial de hardware. Uma implementação em hardware e avaliação do potencialmente melhor padrão de busca é a última contribuição deste trabalho. O hardware proposto é comparado a uma arquitetura baseline para se chegar às conclusões deste trabalho.

Resultados e Discussão
Os experimentos realizados mostraram que desabilitar a FME do VVC leva a um aumento de Bjøntegaard Delta Bitrate (BD-Rate) em média de 0,64% de para sequências de vídeos da classe B (sequências FHD), enquanto Blasi et al. (2015) mostrou que ao desabilitar a FME do HEVC resulta em um aumento de BD-Rate em média de 6,27% para sequências FHD. Essa diminuição do impacto da FME no VVC em relação ao HEVC pode ser devido à adição de novas ferramentas de predição inter-frame.
Os três padrões de busca propostos, denominados *Box*, *Cross* e *Diamond*, e um de um trabalho relacionado, denominado *Ding (a)*, foram implementados no VTM para medir sua eficiência de codificação em comparação com a FME padrão. Os resultados das execuções mostraram que *Diamond* e *Ding (a)* tiveram a melhor eficiência de codificação, com uma média de BD-Rate de 0,18% e 0,19% para Low Delay with P slices only (LD-P) e 0,15% e 0,16 % para acra, respectivamente. *Cross* apresentou resultados ligeiramente piores, com BD-Rates médios de 0,34% e 0,28%, para LD-P e Random Access (RA), respectivamente. *Box* teve a pior eficiência de codificação entre os padrões de busca fixos testados, com aumentos de BD-Rate de 0,57% e 0,44% para LD-P e RA, respectivamente. Com base em resultados detalhados da síntese da arquitetura baseline, foram traçadas estimativas de redução de área e potência de para uma possível implementação de cada padrão de busca com design de hardware semelhante. Dentre os quatro padrões avaliados,

o *Cross* foi estimado como tendo a maior redução de área ocupada e dissipação dinâmica de potência em relação à baseline: 49,4% e 53,0%, respectivamente. Tais reduções são consequência majoritariamente da remoção de buffers utilizados para armazenar amostras fracionárias, resultando assim em um cálculo mais eficiente da FME para uma arquitetura Very-large-scale Integration (VLSI). Além de ser o padrão mais promissor na redução de hardware, *Cross* também resulta em eficiência de codificação ligeiramente pior que *Diamond*, *Cross* e, portanto, foi escolhido para um design de hardware dedicado.

Considerações Finais
Esta dissertação de mestrado apresenta um estudo sobre a FME no padrão de compressão de vídeo VVC. Nela é apresentado o impacto em eficiência de codificação ao desabilitar a FME, assim como o impacto de realizar a FME de forma simplificada, computando apenas um subconjunto de candidatos fracionários. Quatro padrões de busca simplificados foram avaliados, três propostos neste trabalho e um retirado da literatura, em relação a eficiência de codificação e estimativas de redução de área ocupada e potência dissipada. *Cross* foi o padrão mais promissor, com reduções elevadas do hardware implementado, isso se deve ao fato desse padrão não computar as posições fracionárias que mais demandam de operações, resultando em um hardware com menos buffers e que consegue operar em uma frequência de operação menor para a mesma resolução alvo.

Palavras-chave: Codificação de vídeo. Estimação de Movimento Fracionária. Padrão de busca. Arquitetura VLSI. Versatile Video Coding.

ABSTRACT

Video coding is the core of any video application, since it allows for compressing videos, making possible their storage and transmission through the web. The importance of video coding algorithms raised significantly in these days of physical isolation imposed by the Covid-19 pandemic, when video conference became a very powerful tool to avoid the complete interruption of work and study activities. In order to allow for the compatibility between coding and decoding systems, a number of standards have been developed over the years by international standardization agencies, such as ITU-T and ISO, with the collaboration of industry and academia. Finalized in July 2020, the Versatile Video Coding (VVC) is the most recently launched video coding standard, being developed to improve the coding efficiency by more than 40% for the same image quality, when compared to its predecessor, the High Efficiency Video Coding (HEVC). However, the dramatic complexity increase of VVC leads to a higher power dissipation and hinders its implementation in software for real-time applications. Therefore, it is of utmost importance to develop techniques to lower the complexity of the most computationally intensive tasks of this new standard while keeping, as much as possible, its coding efficiency improvements. As usually occurs when a new video coding standard is released, the most intensive tasks, e.g., the Fractional Motion Estimation (FME), are natural candidates for such studies, not only due to their complexity but also to the intrinsic repetitiveness of operations. In particular, when targeting portable devices, energy-efficiency hardware embedded video encoders are indispensable to extend the device's battery life. Thereby, this work focuses on reducing the complexity of the VVC FME by employing a fixed search pattern to design a dedicated VLSI architecture. As first step, the impact of the FME on the VVC coding efficiency was estimated in terms of BD-Rate by executing the VVC Test Model (VTM) with the FME disabled. Then, four fixed search patterns were evaluated in terms of coding efficiency and hardware resources, three of them proposed in this work and the fourth one found in the literature. Coding efficiency was evaluated by implementing the patterns within VTM. Hardware resources were evaluated in terms of area and power by using as baseline a state-of-the-art FME hardware architecture that searches over all available candidates. The proposed *Cross* pattern led to the largest hardware minimization potential with acceptable reduction in coding efficiency and therefore was selected for a dedicated hardware design. The designed architecture was described in verilog and synthetize using a standard cell flow for a 45nm technology. The developed architecture occupied area is down to 41.4% of that of the baseline architecture, while dissipating just 28.9% of the total power, on average. The architecture is also capable of running real time applications on video resolutions of up to 8K@30fps. The trade-off is a BD-Rate increase of 0.34% for LD-P configuration, and 0.28% for the RA configuration.

Keywords: Video coding. Fractional Motion Estimation. Search pattern. VLSI architecture. Versatile Video Coding.

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

$\mathbf{B}^{\text{can}}$	Candidate block.
$\mathbf{B}^{\text{ori}}$	Original block.
$\mathbf{B}^{\text{ref}}$	Reference block.
$\mathbf{B}^{\text{res}}$	Residue block.
D	Differences matrix; D = $\mathbf{B}^{\text{can}}$- $\mathbf{B}^{\text{ori}}$.
$\mathbf{F}^{\text{ori}}$	Original Frame.
$\mathbf{F}^{\text{ref}}$	Reference Frame.
$\mathbf{F}^{\text{res}}$	Residual Frame.
j_{cost}	The lagrangian rate-distortion cost of selecting a given candidate as reference.
λ	The Lagrange multiplier.
S	Set of candidate blocks.

LIST OF ABBREVIATIONS AND ACRONYMS

ALF	Adaptive Loop Filter	47
AMVP	Advanced Motion Vector Prediction	47
AMVR	Advanced Motion Vector Resolution	37
BD-Rate	Bjøntegaard Delta Bitrate	9, 20, 42–44, 46
BMA	Block Matching Algorithm	30
CB	Coding Block	31
CMOS	Complementary metal–oxide–semiconductor	63
CTB	Coding Tree Block	31
CTC	Common Test Conditions	9, 21
CTU	Coding Tree Unit	31
CU	Coding Unit	12, 31
DC®	Synopsys® Design Compiler	62
DCT	Discrete Cosine Transform	30
EDA	Electronic Design Automation	69
FBMA	Fullsearch Block Matching Algorithm	12, 34
FHD	Full HD - 1920×1080	8, 21
FME	Fractional Motion Estimation	7, 11, 12, 21
FOVS	First-Order Vertical Samples	37
fps	frames per second	20
FSM	Finite State Machine	60
HD	High Definition - 1280×720	8, 21
HEVC	High Efficiency Video Coding	7, 8, 20
HM	HEVC Test Model	8, 20
HS	Horizontal Samples	37
HVS	Human Visual System	24
IME	Integer Motion Estimation	36
JVET	Joint Video Exploration Team	20
LD	Low Delay	43
LD-P	Low Delay with P slices only	9, 42, 43, 48
MC	Motion Compensation	45
MCM	Multiplierless Constant Multiplication	43
ME	Motion Estimation	34

MV	Motion Vector	34
MVD	Motion Vector Difference	46
PMD	Portable Mobile Device	20
PU	Prediction Unit	43
QP	Quantization Parameter	46
RA	Random Access	9, 43
RD	Rate-Distortion	34
RDO	Rate-Distortion Optimization	34
RTL	Register Transfer Level	45, 64
SAD	Sum of Absolute Differences	33
SATD	Sum of Absolute Transformed Differences	34
SIMD	Single Instruction/Multiple Data	46
SOVS	Second-Order Vertical Samples	37
SS	Sums and Shifts	12, 57
TB	Transpose Buffer	57
TSMC	Taiwan Semiconductor Manufacturing Company Limited	62, 68
UHD	Ultra High Definition - 3840×2160	8, 20
VCS®	Synopsys® Verilog Compiler Simulator	67
VLSI	Very-large-scale Integration	10, 49
VTM	VVC Test Model	7, 9, 20
VVC	Versatile Video Coding	7, 8, 20

CONTENTS

1 INTRODUCTION

In the last decade, the dissemination of high-speed internet connection worldwide together with the popularization of smartphones enabled an exponential growth in digital video consumption. The use of technologies such as video conferencing and streaming have become essential for the daily activities of most people. According to Cisco (2018), in 2016 73% of all internet traffic was video, and this should grow to 82% by 2022. The Covid-19 pandemic intensified this even further, as more people are relying on video services to perform their professional activities, to follow classes, or meet relatives and friends. As claimed by Koeze & Popper (2020), the Northern American people in particular have been spending more time online, with Netflix and Youtube receiving an increase in 16.0% and 15.3% in traffic, respectively, during the first three months of 2020.

To keep up with the demand of the global market, new technologies are constantly under development. Higher resolutions and frame rates are continuously demanded by those who seek a better video experience. However, this search leads to new issues that must be solved. A Ultra High Definition - 3840×2160 (UHD) video without any compression (i.e., in the "raw" format) at 30 frames per second (fps) and using 24 bits per pixel requires a bitrate of 5.971 Gbits/s, far from what current connections can provide. If this sequence is 10 minutes long, it will require 447 GB to be stored, which is not acceptable for ordinary users. Therefore, video compression is mandatory to make it possible to store and share digital videos. To compress videos, video coders rely on standards so that compatible devices can decode the information and reproduce the video.

The Versatile Video Coding (VVC) (ITU-T, 2020) is the new coding standard developed by the Joint Video Exploration Team (JVET), finalized in July of 2020. It is estimated to provide more than 40% BD-Rate[1] gain (PAKDAMAN et al., 2020), i.e., a 40% coding efficiency improvement, when compared to its predecessor, the High Efficiency Video Coding (HEVC) (ITU-T, 2013). However, this gain comes with an increase in encoding time of around 10× when comparing the VVC Test Model (VTM) and theHEVC Test Model (HM), which are the reference software models for VVC and HEVC, respectively.

Whenever a video is captured using a Portable Mobile Device (PMD), it must be compressed in real time due to storage limitations (VANNE et al., 2012). However, video compression is a computationally intensive process that consumes large amounts of energy. Therefore, video compression as a task embedded into PMD must also consider energy efficiency (BRäSCHER; SEIDEL; GüNTZEL, 2017). A common and effective approach to allow real-time compression while minimizing energy usage consists in adopting dedicated

[1] Bjøntegaard Delta Bitrate (BD-Rate) is a metric that compares the efficiency of two codifications represented in percentage, were a negative value indicates a bitrate reduction to encode the same sequence for the same quality, and a positive value indicates a bitrate increase (BJØNTEGAARD, 2001).

hardware accelerators for the video encoder.

1.1 RATIONALE, CONTRIBUTIONS AND METHODOLOGY

Fractional Motion Estimation (FME), which is a refinement search of newly interpolated candidates using pre-existing ones, is one of the most compute-intensive tools in the HM, being responsible for 33.88% of the total encoding time, on average (BLASI et al., 2015). When it is disabled in HM, the bitrate increases on average by 7.01%, 6.27% and 16.33% for UHD, Full HD - 1920×1080 (FHD) and High Definition - 1280×720 (HD) sequences, respectively, considering the same video quality (BLASI et al., 2015). To mitigate this problem, the HEVC FME solutions usually involve simplifying the interpolation filters or limiting the number of candidates searched in the process (LV et al., 2014; HE et al., 2015; KALALI; HAMZAOGLU, 2018; LIM et al., 2016; DING; YE; WANG, 2015).

The main challenge of FME hardware design is dealing with data dependencies during interpolation and search. Therefore, alternative FME search patterns can alleviate both interpolation and search by limiting the latter to a few, most probable candidates.

The VVC standard brings a number of new coding tools that result in substantial increase of complexity with respect to the previous standards. However, to the best of the author's knowledge, so far no study on how FME contributes to the coding efficiency of this recently finished standard has been published and no dedicated hardware design has been proposed. Therefore, this work brings five contributions:

1. Quantitative assessment of the coding efficiency impact of the FME on videos encoded with VVC.
2. Proposal of three (simplified) fixed search patterns for the FME of VVC.
3. Evaluation of the coding efficiency of the three proposed fixed patterns along with another one found in the literature.
4. Estimation of the hardware reduction provided by each pattern when compared to a baseline FME architecture.
5. Design and evaluation of a dedicated hardware architecture that implements the fixed search pattern leading to the best (potential) trade-off between energy minimization and coding efficiency loss, comparing it to a baseline FME architecture.

In order to achieve the first and third contributions, experiments were conducted with the VTM v13.0 by encoding video samples from the Common Test Conditions (CTC) (BOSSEN, 2012), so as to evaluate the coding efficiency. For the case stated in the first contribution, the FME function was disabled. For evaluating the coding efficiency the fixed patterns were implemented within VTM (third contribution). For the second contribution, the proposed patterns were designed searching to reduce the number of candidates computed by the FME, only analyzing the most likely candidates of being chosen as reference. Executions of the default FME showed the percentage of each candidate being chosen as reference. The characteristics of an FME hardware architecture were also taken into consideration when defining each pattern. To reach the fourth contribution, an analysis over the baseline architecture was conducted. Each pattern was evaluated to estimate potential hardware reduction. A hardware implementation and evaluation using the potentially best pattern is the last contribution of this work. The proposed hardware is compared to the baseline architecture to draw the conclusions.

1.2 ORGANIZATION OF THIS DOCUMENT

The remaining chapters are organized as follows:

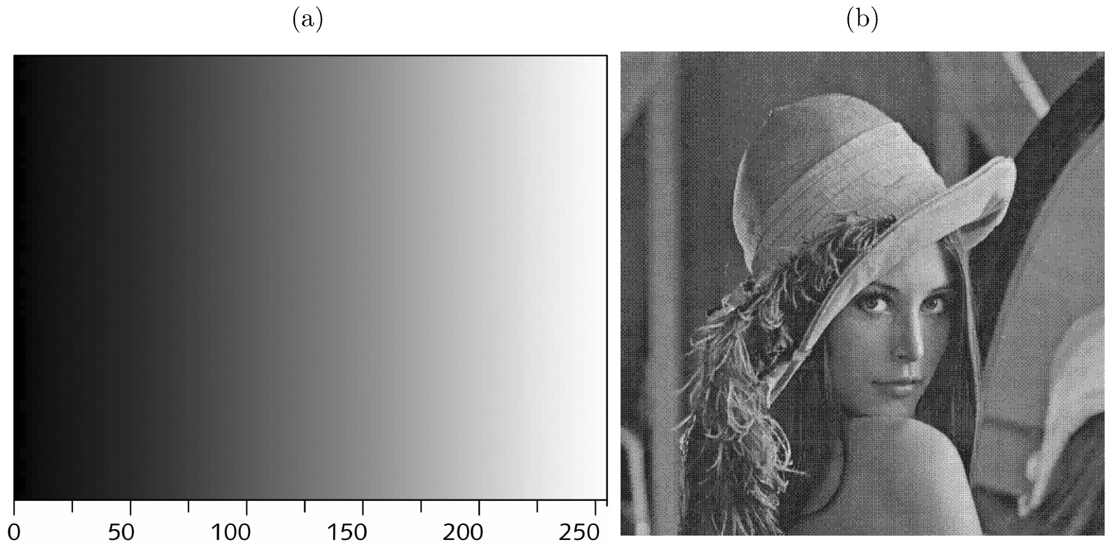
- Chapter 2 brings fundamental video coding concepts, as well as notations and definitions that are used in this manuscript. It also brings an overview of the hybrid video codec model, and reviews the most relevant concepts related to block partitioning and prediction step.
- Chapter 3 presents the most relevant aspects of FME, with special attention to the VVC FME, and presents and discusses the most relevant related work on dedicated FME hardware architectures found in the literature.
- Chapter 4 presents the quantitative assessment of the coding efficiency impact of the FME on VVC and the evaluation of the fixed FME search patterns.
- Chapter 5 presents the baseline architecture, area and power reduction estimates for the proposed FME search patterns, and a hardware architecture based on the potentially best search pattern. In addition, it presents the synthesis results of a hardware architecture.
- Chapter 6 draws the research conclusions.

2 VIDEO CODING CONCEPTS, THE HYBRID MODEL AND THE PREDICTION STEP

This chapter reviews video coding concepts that are important for the understanding of this work and presents notations and definitions that are employed all along this text. It also brings an overview of the hybrid video codec model, adopted by all contemporary video coding standards, and reviews the most relevant concepts related to block partitioning mechanism and prediction step. Particularly, the VVC standard is used to illustrate the block partitioning concepts.

A digital video is composed of a sequence of frames sampled at regular intervals in time (temporal samples), where each frame represents the image by means of a matrix of elementary picture elements (spatial samples), popularly known as pixels (RICHARDSON, 2004). In a grayscale image, each pixel holds a single value, referred to as spatial sample, representing the brightness or luminance (luma). Figure 1 illustrates the representation of luma samples represented in an 8-bit scale, where lower luma values represent visually darker pixels.

Figure 1 – (a) Grayscale ramp of luma samples represented with 8 bits, integer values columns going from 0 to 255. (b) Example of a frame represented with 8 bits in grayscale.



Source: adapted from Poynton (2012). Image “Lena” from http://www.lenna.org/lena_std.tif.

In a color system, each pixel comprises more than one component using a color model. Color models are mathematical models to represent color in tuples, usually a triple, of samples for each pixel. In the RGB color model, a color is represented by a triple associated to each pixel to represent the intensity of Red, Green and Blue, i.e., the primary colors, which can be mixed to produce a gamut of colors. In this color space all three samples are equally important and therefore, have the same resolution. This way, if each color is represented with 8 bits, each pixel represented with the RGB model will use 24 bits. Samples that hold values to represent color characteristics of the pixel are called chroma samples.

Briefly addressed in Chapter 1, everyday consumption of “raw” digital video demands resources our current technology is unable to provide. Equation 2.1 shows the required bitrate per second to compute a video with no compression. For example, an video with 8K (7680×4320) resolution running at 30 fps and 24 bits per pixel requires a $\text{bitrate}_{\text{raw}}$ of 23.89Gbps, nearly 3GB per second.

$$\text{bitrate}_{\text{raw}} = \text{bits per pixel} \times \underbrace{\text{width} \times \text{height}}_{\text{Frame Resolution}} \times \text{frame rate} \tag{2.1}$$

The total size of a digital video file is obtained by multiplying the $\text{bitrate}_{\text{raw}}$ for the duration of the video sequence, as shown in Equation 2.2. For the previous example, the total file size of the “raw” 8K video with one hour of duration would be 10.75TB, which is unreachable by current technology.

$$\text{size}_{\text{raw}} = \text{bitrate}_{\text{raw}} \times \textbf{sample duration} \tag{2.2}$$

This example shows clearly that digital video data needs to be compressed to enable its storage and transmission, which can be achieved by reducing the redundancies within its representation. Shi & Sun (2008) classify video redundancies as:

1. **Psychovisual**: associated to characteristics of the Human Visual System (HVS);
2. **Statistical**: associated to the repetitions of symbols in the video. They are subdivided in:
   a) **Coding**: the statistical redundancy associated with coding techniques, i.e., the entropy;
   b) **Interpixel**: pixels within a frame or a sequence of frames are not statistically independent. Such class can be further divided into:
      i. **Spatial**: the statistical correlation existing in pixels within the same frame, i.e., intra-frame;
      ii. **Temporal**: the statistical correlation existing in pixels within a sequence of frames, i.e., inter-frame.

The next two sub-sections discuss redundancies in more detail.

2.1 PSYCHOVISUAL REDUNDANCIES

The psychovisual redundancies are related to information which the HVS is not sensitive enough to differentiate. The HVS does not perceive visual information equally and the main reason is that the number of cone cells, responsible for differentiating the distinct light wavelengths, i.e., color, is far less numerous and less sensitive than rod cells, which are responsible for perceiving light sensitive, i.e., brightness (HUNT, 2005). Consequently, subtle variations to color are harder to be perceived than subtle variations to luminance (HUNT, 2005). Then, since color information is less important than brightness information, we may say that raw videos contain psychovisual redundancies which are explored by video compression techniques so as to reduce the data needed to represent them (SHI; SUN, 2008).

More specifically , chroma subsampling is a type of compression that explores the aforementioned characteristic of the HVS, by reducing the amount of chroma data to represent a digital image. However, to apply chroma subsampling, it is necessary to use a color model that represents color and light in separate components, e.g., YCbCr (ITU-T, 2002; ITU-T, 1995). In this color model, Y is the luma component and Cb and Cr are the blue- and red-difference chroma components. The Y component is determined as a weighted average of the RGB component, as defined in Equation 2.3.

$$Y = K_R.R + K_G.G + K_B.B \tag{2.3}$$

Where K_R, K_G and K_B are derived from the RGB space, and must satisfy the condition in Equation 2.4.

$$K_R + K_G + K_B = 1 \tag{2.4}$$

ITU-R recommendation BT.601-7 (ITU-R, 2011) defines $K_R = 0.299$, $K_G = 0.587$ and $K_B = 0.114$. The chroma components are derived from the difference between the R, G or B components and Y component, as shown in Equation 2.5 to Equation 2.7.

$$Cb = B - Y \tag{2.5}$$

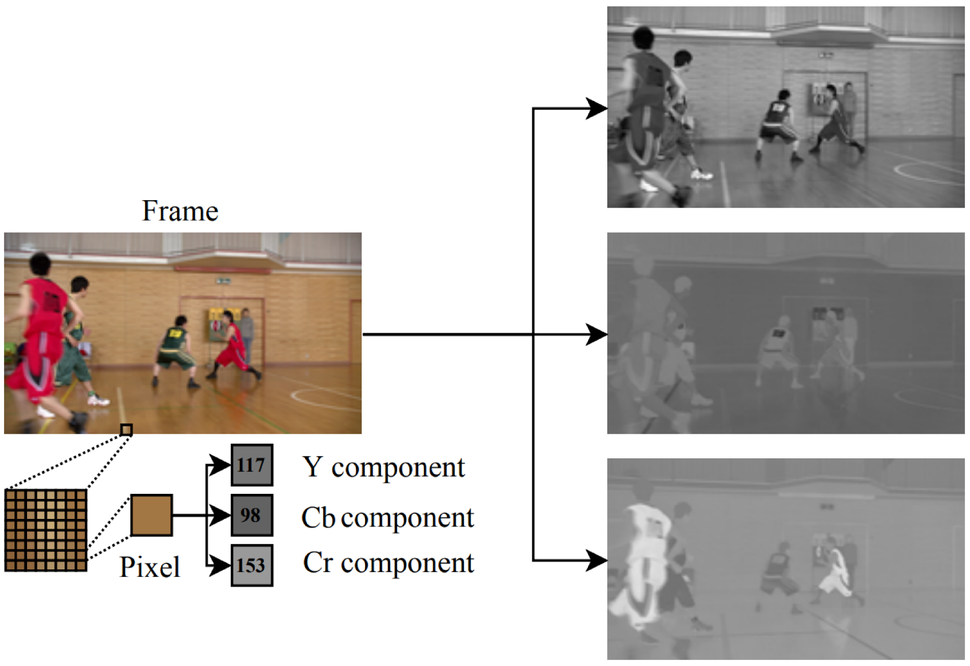
$$Cr = R - Y \tag{2.6}$$

$$Cg = G - Y \tag{2.7}$$

The complete representation of a pixel is given by the Y component and the three chroma components. However, since Equation 2.4 is always true, only two chroma components are required to store and transmit videos, as the third chroma component can be derived from the other two. In the case of YCbCr, the green-difference (Cg) is not stored or transmitted. Figure 2 shows a frame representation, where the frame is constituted from a grid of pixels, each pixel being composed of three components of the YCbCr color space. The frame representation of a component is called channel, as seen on the right side of the picture.

As previously mentioned, the amount of data to represent videos can be reduced by applying chroma subsampling. Figure 3 illustrates three possible representations for the YCbCr color model. In Figure 3a, every pixel is represented by the three components. In Figure 3b, one in every two columns of pixels is represented only by the Y component

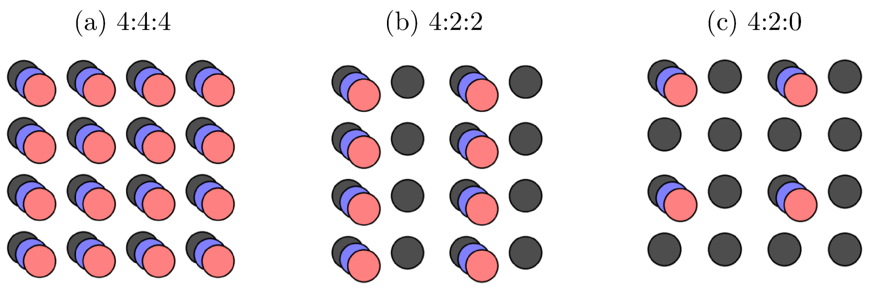
Figure 2 – A frame representation composed of a matrix of pixels, the pixels being represented in the YCbCr color model. The channels are the frame representation of each component.



Source: Cancellier (2016).

and hence, the horizontal chroma resolution is halved. In Figure 3c, both the horizontal and vertical resolutions are halved and thus, only one in every four pixels is represented by chroma and luma components whereas the others are represented only by the luma component. During the frame decoding, pixels with no chroma information receive the chroma components value of nearby pixels.

Figure 3 – Three possible representations for the YCbCr color model. Each circle represents an YCbCr color component: dark gray circles represent Y component whereas blue and red circles represent Cb and Cr components, respectively.



Source: Seidel (2014).

If each color channel has a bit depth of 8, a 4:4:4 YCbCr requires 24 bits per pixel, a 4:2:2 requires 16 bits per pixel, and a 4:2:0 requires 12 bits per pixel. Hence, the 4:2:0 chroma subsampling is able to reduce by 50% the bitrate. Going back to Equation 2.1, just applying the 4:2:0 chroma subsampling to the 8K @ 30 fps video from the previous example reduces the required bitrate from 23.89Gbps to 11.95Gbps. However, such reduction is not enough to properly store or transmit digital videos, and specific techniques must be used within an encoding model that explores the types of redundancies.

2.2 STATISTICAL REDUNDANCIES

Statistical redundancies are related to the repetitions of symbols in the video. Shi & Sun (2008) classify these redundancies into two types: interpixel and coding. Similarly to psychovisual, interpixel is related to redundancies associated with data contained in to represent the video. To reduce the bitrate, the encoder can eliminating this redundancies, e.g., chroma sampling, or utilizing correlation within pixel data to represent the image with fewer bits, as will be seen in the following sections. Coding redundancies, on the other hand, are different in the way that they are independent from the information, but instead are related to the way this information is encoded (SHI; SUN, 2008). Entropy coding can be applied to reduce these coding redundancies, compressing data by changing the representation of each symbol from a constant number of bits to a variable number, depending on the occurrence probability of each symbol.

Table 1 illustrates the entropy coding principle. The leftmost column lists six symbols to be encoded, the following columns list the occurrence probability, the fixed length base code, and the variable length new code. In the fixed base code, all symbols are encoded with three bits. In the variable new code, high probability symbols are encoded with fewer bits, whereas low probability symbols are encoded with more bits.

Table 1 – Entropy coding example.

Symbol	**Probability**	**Base code**	**New code**
a	0.5	000	0
b	0.2	001	01
c	0.1	010	001
d	0.1	011	010
e	0.05	100	0001
f	0.05	101	0010

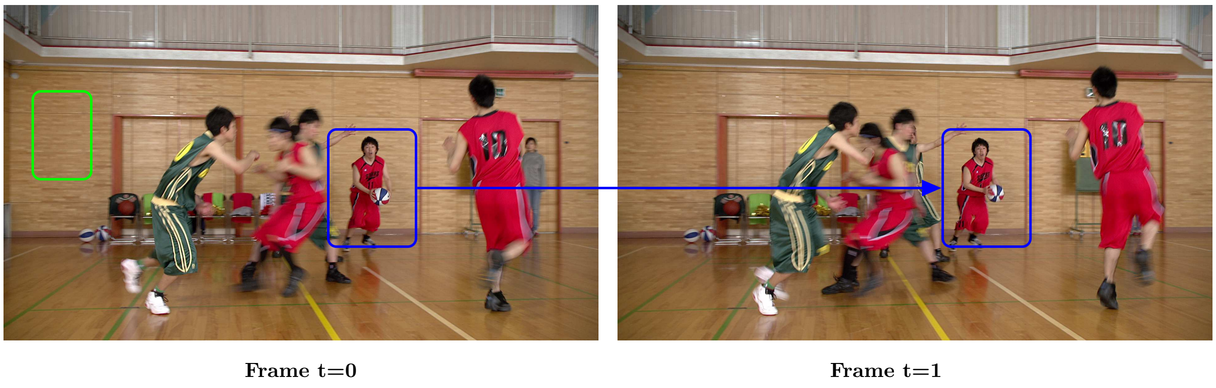
The average bit length per symbol for the fixed base code is three bits, evidently. Equation 2.8 calculates the average bit length per symbol encoded with the variable length new code, 1.9 bits per symbol, 63.3% of the base code average bit length.

$$L = 0.5 \times 1 + 0.2 \times 2 + 0.1 \times 3 + 0.1 \times 3 + 0.05 \times 4 + 0.05 \times 4 = 1.9 \tag{2.8}$$

Entropy coding is a lossless type of data compression, meaning the decoded data is the same as the original data. Thus, no information is lost. On the contrary, chroma subsampling is a lossy type of data compression, meaning that information is lost in the coding process and the reconstructed data is not the same as the original. As a consequence, this lossy compression will cause loss in video quality. The key idea is to avoid introducing losses that are visually perceptible (SEIDEL, 2019).

The second type of statistical redundancies is interpixel. The interpixel redundancy is the statistical correlation between pixels in the same frame, i.e., intra-frame, or distinct frames, i.e., inter-frame. Figure 4 illustrates the two types of interpixel redundancies. On the top left corner of **Frame t=0**, the data of the pixels representing that slice of the wall are very similar. Video coding tools can explore these data similarities across multiple pixels of the same frame to reduce the amount of bits required to encode these pixels. Likewise, there are data similarities across neighboring frames. For instance, the data needed to represent the basketball players are the very similar between **Frame t=0** and **Frame t=1**, and thus can also be explored to achieve higher compression rates.

Figure 4 – Intra- and inter-frame redundancies. Intra-frame redundancy is the correlation between adjacent pixels within a given frame, as illustrated by the wall on the top left corner of **Frame t=0**. In turn, inter-frame redundancy dwells on correlation within neighboring frames, as illustrated by the capture of the players on the pitch at **Frame t=0** and **Frame t=1**.



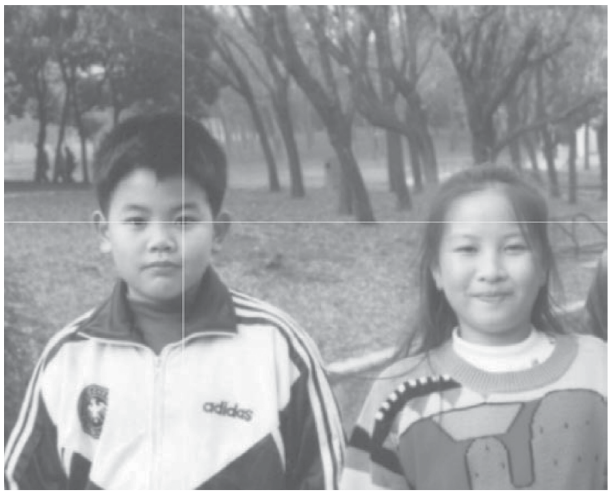
Source: adapted from Cancellier (2016).

The correlation of data within nearby samples can be better discerned when analyzing the data of a single channel. Figure 5 shows a grayscale picture with 8 bits per pixel, and the profile of the luma values for the row 318 and column 62. From Figure 5b, one may observe that samples from row 1 through row 160 hold values that are close to each other, same for samples from row 372 and 531. Smaller intervals with good value correlation are also present across the intensity profile. The column profile, shown in Figure 5c, has plenty of correlation within the samples as well: the luma values are very similar from columns 1 through 259 and from columns 560 through 689, adding to more data redundancies.

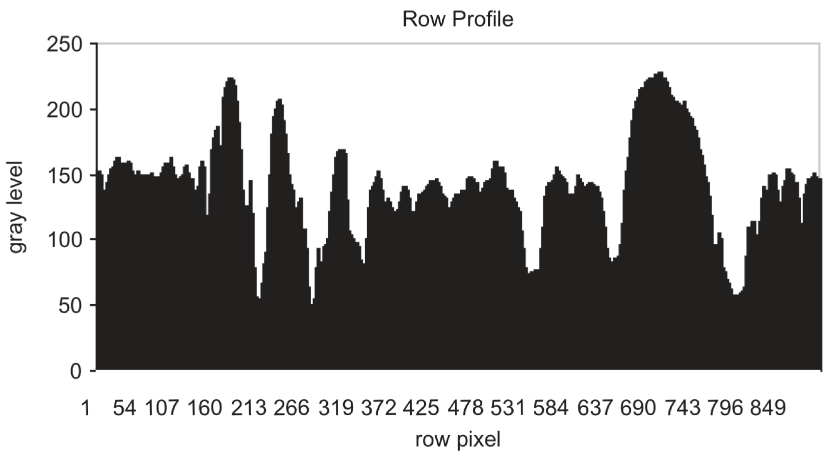
In video coding, prediction is a key technique which allows the encoding tools to explore interpixel redundancies, both intra- and inter-frame. Prediction attempts to reduce the redundancies between pixels by forming a residual frame ($\mathbf{F}^{\text{res}}$), the residual frame is the difference between the original frame ($\mathbf{F}^{\text{ori}}$) and a reference frame ($\mathbf{F}^{\text{ref}}$). Such $\mathbf{F}^{\text{res}}$ is expected to present lower entropy than its related $\mathbf{F}^{\text{ori}}$. Therefore, the $\mathbf{F}^{\text{res}}$ is encoded along with the $\mathbf{F}^{\text{ref}}$. The decoder is responsible for reconstructing the $\mathbf{F}^{\text{ori}}$ by adding the $\mathbf{F}^{\text{res}}$ to the $\mathbf{F}^{\text{ref}}$. This compression can be lossy if the $\mathbf{F}^{\text{res}}$ if quantized prior the bitstream generation, then the decoded $\mathbf{F}^{\text{ori'}}$ may not be identical to the $\mathbf{F}^{\text{ori}}$.

Figure 5 – (a) Grayscale picture of “Boy and Girl” with 318th row and 262th column demarcated in white, (b) intensity profile along 318th row, and (c) intensity profile along 62th column.

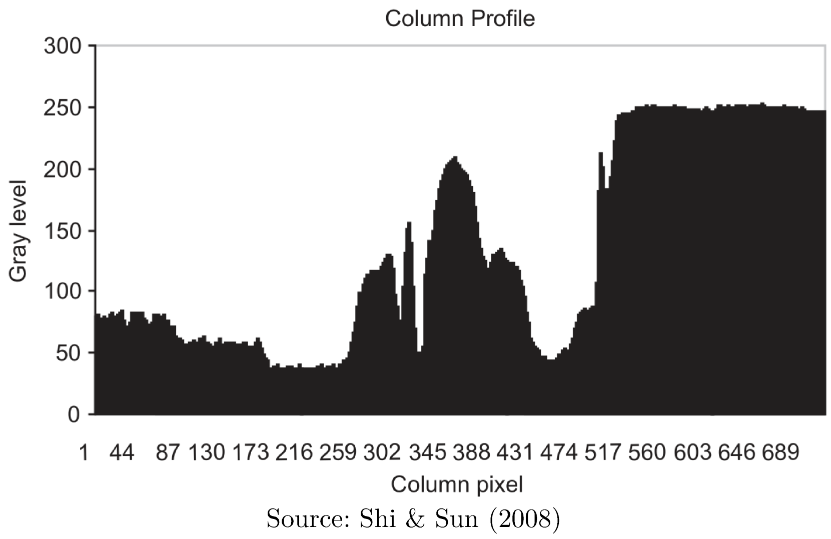
(a)



(b)



(c)



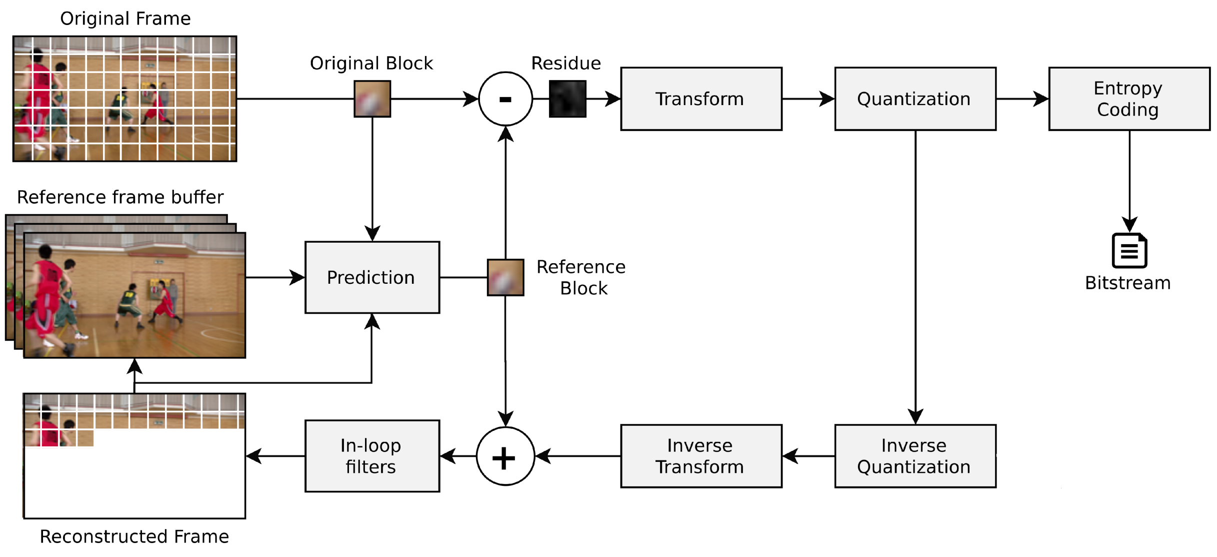
Source: Shi & Sun (2008)

Transform coding is another technique that is generally used to reduce interpixel redundancies, by converting the frame data from a spatial domain into another domain. The Discrete Cosine Transform (DCT), for example, transforms the residual data from the spatial domain to the frequency domain. By doing so, the resulting frequency spectrum concentrates the most relevant visual information of the image in a few coefficients improving data compression of techniques such as entropy encoding (RICHARDSON, 2004). During decoding, the transformed data is reversible using just a low number of arithmetic operations, therefore adding minimal complexity to the decoder.

2.3 THE HYBRID VIDEO CODEC MODEL

Figure 6 depicts the video encoder flow following the widely used video codec[1] hybrid model. The qualifier "hybrid" comes from the fact that it integrates prediction, transform and quantization stages within the same model. Since its introduction with the H.261 in 1988, hybrid coded standards follow such a generic model (RICHARDSON, 2004).

Figure 6 – Simplified video encoder hybrid model.



Source: adapted from Richardson (2004).

Besides relying on the hybrid model, modern coding standards perform a block-based prediction, meaning that prior to prediction each frame to be encoded ($\mathbf{F}^{\mathrm{ori}}$) is splitted into non-overlapping rectangular blocks of M×N pixels ($\mathbf{B}^{\mathbf{ori}}_{m\times n}$). By doing so, each block can be predicted separately. Prediction selects a reference block ($\mathbf{B}^{\mathbf{ref}}_{m\times n}$) for each $\mathbf{B}^{\mathbf{ori}}_{m\times n}$ using a Block Matching Algorithm (BMA)[2]. The residue block ($\mathbf{B}^{\mathrm{res}}$), which is

[1] The word codec refers to the encoder/decoder pair, which is responsible for compressing/reconstructing the data that represents digital video.

[2] More details on BMA in Section 2.5.

difference between $\mathbf{B}^{\text{ori}}$ and $\mathbf{B}^{\text{ref}}$, is sent to the transform step, followed by a quantization step that consists in reducing the signal representation intervals. Said intervals are set by quantization parameters and the information between this intervals is lost, making it a lossy compression. Lastly, an entropy encoding is performed to further reduce the amount of data to represent the video, thus generating the bitstream.

Part of the decoding loop consists of reversing the encoding steps by entropy decoding, inverse quantization and inverse transform. The data resulted is a reconstructed residue block which added to the reference frame is used to reconstruct the original frame.

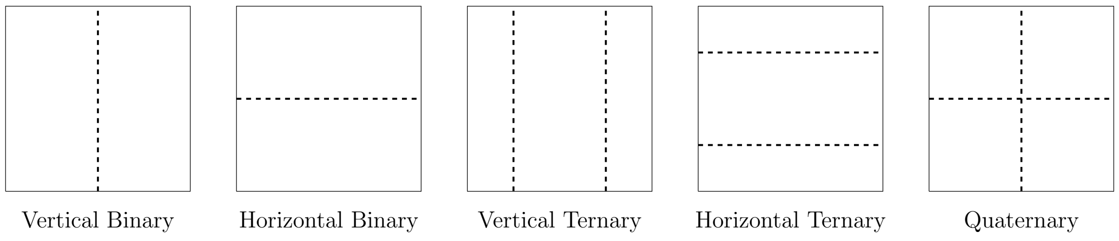
2.4 BLOCK PARTITIONING

As mentioned in Section 2.3, each frame is partitioned into blocks before undergoing the prediction, such partitioning being defined by the coding standard. This section relies on the VVC to explain the frame partitioning mechanism, since it is the most recent standard. However, it is worth mentioning that all other video coding standards adopt similar partitioning mechanisms.

In VVC, each frame is divided into the so-called Coding Tree Units (CTUs). For each frame in a video that uses the YCbCr color model, a CTU is an N×N block of luma sample and the two corresponding chroma samples, each component being called Coding Tree Block (CTB). The maximum allowed size of the luma block in a CTU is 128×128 (CHEN; YE; KIM, 2020).

Each CTU is then split into Coding Units (CUs) using a quaternary-tree structure. Afterwards, each quaternary-tree leaf node can be further split using five splitting types in a multi-type tree structure, as shown in Figure 7. A vertical binary split decomposes an N×M unit into two N/2×M units, whereas a vertical ternary split decomposes an N×M unit into N/4×M, N/2×M and N/4×M units (CHEN; YE; KIM, 2020). Horizontal binary and ternary split operate in a similar way, but in the opposite axis. The block partitioning can also choose to not split the unit. A mode decision step is responsible for the partitioning mode of each CTU. Same as the CTU, the CU consists of a luma sample block and two corresponding blocks of chroma samples, called Coding Block (CB) (CHEN; YE; KIM, 2020).

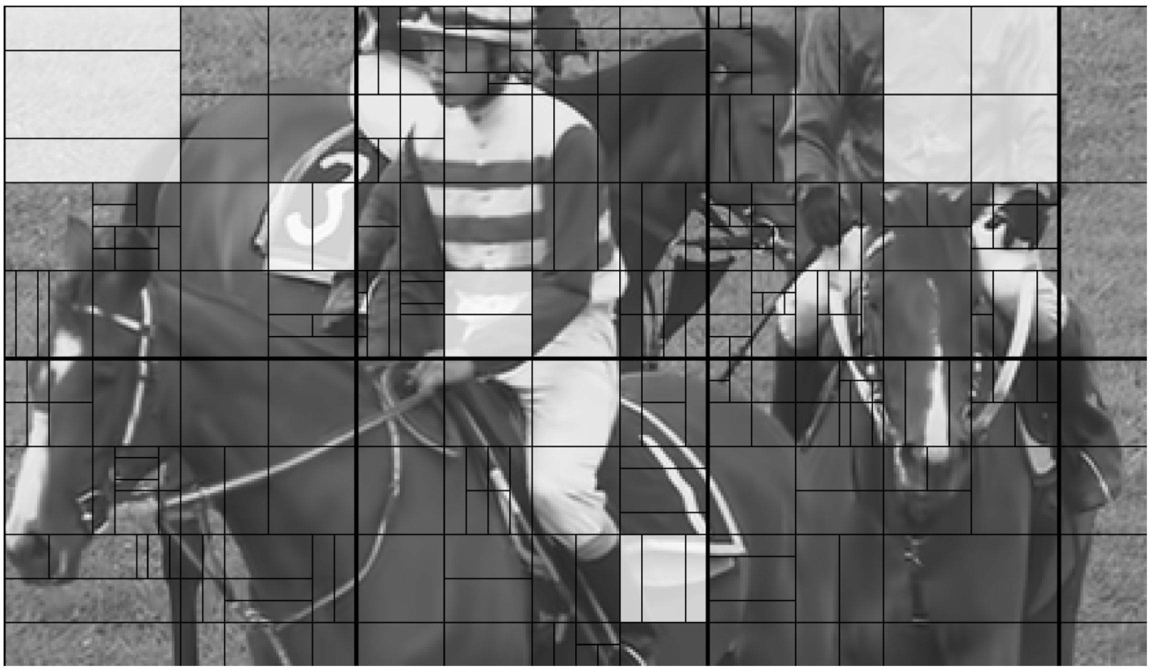
Figure 7 – Multi-type tree splitting modes supported by VVC.



Source: adapted from Chen, Ye & Kim (2020).

In VVC, the minimum supported size of a CU is 4×4 in units of luma samples, and the maximum supported size is 128×128, the same maximum size supported by the CTU (CHEN; YE; KIM, 2020). In a chroma subsampling of 4:2:0, the chroma CB minimum and maximum sizes will be, consequently, 2×2 and 64×64 (CHEN; YE; KIM, 2020), Whereas the maximum supported sizes of the transform blocks in VVC are 64×64 for luma and 32×32 for chroma. The multi-type tree leaf node CU is used for both the prediction and the transform processes, except when the CU size is larger (in width or height) than the maximum transform block size supported for the color components. In cases where the width and/or height of the CB is larger than the maximum transform, the CB is automatically split in the direction to meet the transform size restriction in that direction (CHEN; YE; KIM, 2020). Figure 8 illustrates an example of a multi-type tree block partitioning in VVC.

Figure 8 – Example of a frame partitioned into CU, where VVC's five split patterns are highlighted (quaternary, horizontal binary, vertical binary, horizontal ternary and vertical ternary). Thicker lines represent CTU boundaries.

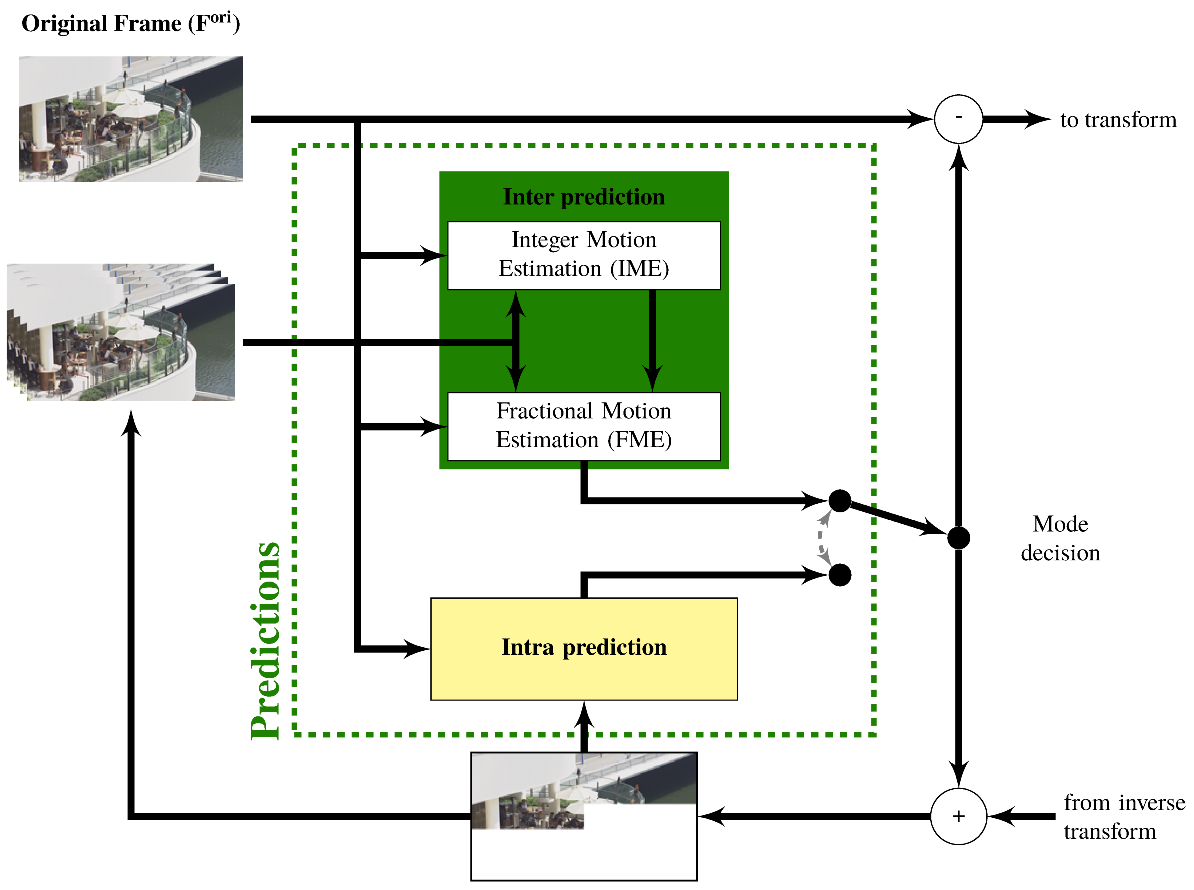


Source: Bossen et al. (2021).

2.5 PREDICTION UNIT

The goal of prediction is to reduce the entropy by exploring the interpixel redundancies. Figure 9 shows a detailed view of a prediction unit: every $\mathbf{B}^{\text{ori}}$ in $\mathbf{F}^{\text{ori}}$ undergoes an intern inter-frame prediction and intra-frame prediction step. Inter-frame prediction searches for a $\mathbf{B}^{\text{ref}}$ in other frames than $\mathbf{F}^{\text{ori}}$ by exploring the temporal redundancies, whereas intra-frame prediction searches for a $\mathbf{B}^{\text{ref}}$ within $\mathbf{F}^{\text{ori}}$ by exploring the spatial redundancies. The mode decision step is responsible for choosing, for each CU, the prediction type (inter-frame or intra-frame) and the partitioning as well. In VVC, prediction is made at a CU level.

Figure 9 – Detailed view of the prediction unit of a video encoder.


Original Frame (F^{ori})
to transform
Inter prediction
Integer Motion Estimation (IME)
Fractional Motion Estimation (FME)
Predictions
Mode decision
Intra prediction
from inverse transform

Source: Seidel (2019).

To find the $\mathbf{B}^{\mathrm{ref}}$, the encoder runs a BMA. Given $m{\times}n$ sized blocks, a BMA is executed over a set (S) of candidate blocks ($\mathbf{B}^{\mathrm{can}}$) to find the one that minimizes a cost metric for the $\mathbf{B}^{\mathrm{ori}}$ as shown in Equation 2.9 (CHAKRABARTI et al., 2015). Such $\mathbf{B}^{\mathrm{can}}$ is chosen as $\mathbf{B}^{\mathrm{ref}}$.

$$\mathbf{B}^{\mathrm{ref}}{}_{m\times n} = \underset{\mathbf{B}^{\mathrm{can}}{}_{m\times n}\in S}{\arg\min}\ cost(\mathbf{B}^{\mathrm{ori}}{}_{m\times n}, \mathbf{B}^{\mathrm{can}}{}_{m\times n}) \tag{2.9}$$

The number of $\mathbf{B}^{\mathrm{can}}$ is usually high, which requires fast and simple ways to compute the *cost* function. Among the existing *cost* functions, the Sum of Absolute Differences (SAD) is the most widely employed one thanks to its simplicity (RICHARDSON, 2004). The SAD, as the name says, is the sum of all differences between $\mathbf{B}^{\mathrm{ori}}$ and $\mathbf{B}^{\mathrm{can}}$, defined in Equation 2.10, were the difference matrix D ($\mathbf{D}_{m\times n}$ in Equation 2.11) is computed by subtracting each $\mathbf{B}^{\mathrm{ori}}$ and $\mathbf{B}^{\mathrm{can}}$ position. The $\mathbf{B}^{\mathrm{can}}$ that minimizes the metric, e.g., lowest value, is chosen as $\mathbf{B}^{\mathrm{ref}}$.

$$sa(\mathbf{D}_{m\times n}) = \sum_{i=1}^{m}\sum_{j=1}^{n} |\mathbf{D}_{i,j}| \tag{2.10}$$

$$\mathbf{D}_{m\times n} = \mathbf{B}^{\mathrm{ori}}{}_{m\times n} - \mathbf{B}^{\mathrm{can}}{}_{m\times n} \tag{2.11}$$

The $\mathbf{B}^{\mathrm{res}}$, which will encoded, is the difference sample by sample of the $\mathbf{B}^{\mathrm{ori}}$ and $\mathbf{B}^{\mathrm{can}}$ as defined in Equation 2.12.

$$\mathbf{B}^{\mathrm{res}}{}_{m\times n} = \mathbf{B}^{\mathrm{ori}}{}_{m\times n} - \mathbf{B}^{\mathrm{ref}}{}_{m\times n} \tag{2.12}$$

Another widely used metric is Sum of Absolute Transformed Differences (SATD). It computes a transform over $\mathbf{D}_{m\times n}$ anticipating the transform process of the $\mathbf{B}^{\mathrm{res}}$ that is executed after the prediction (Figure 6). Although, transforming the $\mathbf{D}_{m\times n}$ for each $\mathbf{B}^{\mathrm{can}}$ for the $\mathbf{B}^{\mathrm{ref}}$ decision is a very compute-intensive task. The trade-off is picking a $\mathbf{B}^{\mathrm{res}}$ that will produce minimal residue after the transformation step, therefore minimizing the bitrate (SEIDEL, 2019).

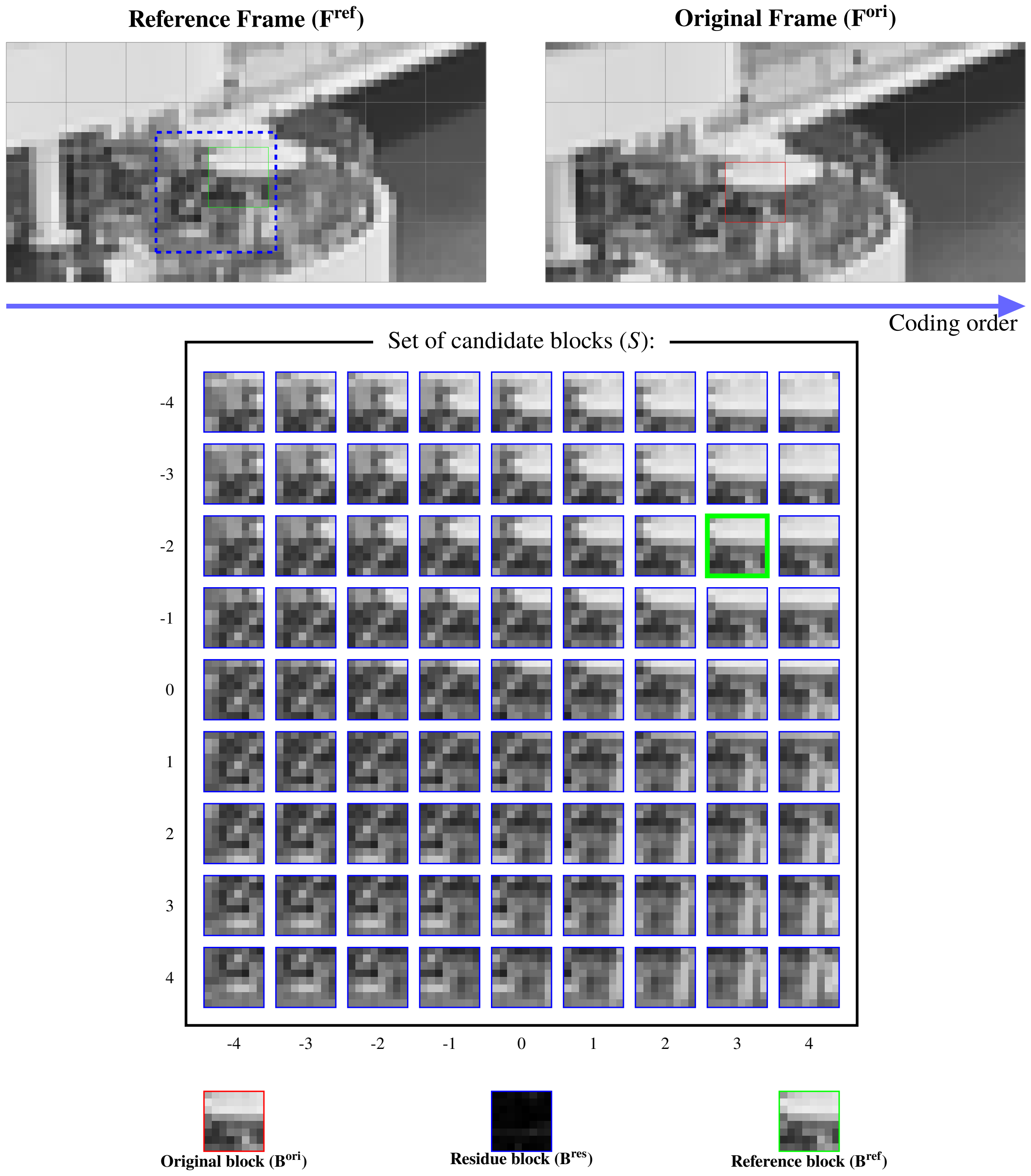
In the intra-frame prediction, the set of $\mathbf{B}^{\mathrm{can}}$s is located within the same frame as $\mathbf{B}^{\mathrm{ori}}$. They are artificially created by interpolation of samples from the borders of neighbor blocks outside the $\mathbf{B}^{\mathrm{ori}}$. Distinct intra modes define how the corresponding $\mathbf{B}^{\mathrm{can}}$s are generated. A BMA is used to select the best mode finding the $\mathbf{B}^{\mathrm{ref}}$ over a set of S that minimizes the applied cost metric. To allow decoding, the selected mode is also included in the bitstream along with the $\mathbf{B}^{\mathrm{res}}$.

The inter-frame prediction is based on Motion Estimation (ME). ME is the process of finding a $\mathbf{B}^{\mathrm{ref}}$, over a set of S inside a search area of a $\mathbf{F}^{\mathrm{ref}}$, that is the most similar to the $\mathbf{B}^{\mathrm{ori}}$ according to the applied distortion metric, as illustrated in Figure 10. A Fullsearch Block Matching Algorithm (FBMA) is performed to obtain the optimum value by including all blocks inside the search windows in the set of S. However, fast BMA algorithms, that searches only on some of the candidates, may be used to speed up the search, at a cost of a sub-optimal block matching. Besides the $\mathbf{B}^{\mathrm{res}}$, a Motion Vector (MV) that stores the coordinates of the $\mathbf{B}^{\mathrm{ref}}$ in reference of the $\mathbf{B}^{\mathrm{res}}$ is also encoded, so that the decoder can identify the $\mathbf{B}^{\mathrm{ref}}$ of each $\mathbf{B}^{\mathrm{res}}$ for future reconstruction of the encoded block. The residue MV in this example of Figure 10 is $\overrightarrow{mv}^{\mathrm{res}} = (3, -2)$.

Using only a distortion metric to compute the *cost* of each $\mathbf{B}^{\mathrm{can}}$ is a rather simplistic approach. Since the $\overrightarrow{mv}^{\mathrm{res}}$ is also encoded to the bitstream, it will also affect the candidate's *cost*. Smaller $\overrightarrow{mv}^{\mathrm{res}}$ will need fewer bits to be encoded, while larger $\overrightarrow{mv}^{\mathrm{res}}$ will require more bits to be encoded. Therefore, to improve coding efficiency, encoders often rely on an optimization to the *cost* metrics that take into consideration the distortion metric and the number of bits needed to encode such $\mathbf{B}^{\mathrm{ori}}$, called Rate-Distortion Optimization (RDO) (SULLIVAN; WIEGAND, 1998; ORTEGA; RAMCHANDRAN, 1998).

In RDO, the cost function (Equation 2.9) is usually a compound metric that considers both rate (compression) and distortion (quality), thus called Rate-Distortion (RD) cost. The Lagrangian RD cost (j_{cost}) weights both terms by means of a pre-calculated Lagrange Multiplier (λ), as defined in Equation 2.13.

Figure 10 – Example of FBMA considering a 16×16 pixel search area and 8×8 pixel block size, resulting in 9×9 $\mathbf{B}^{\mathrm{can}}$s $\in S$. The selected $\mathbf{B}^{\mathrm{ref}}$ is the $\mathbf{B}^{\mathrm{can}}$ at coordinates $(3, -2)$. After $\mathbf{B}^{\mathrm{ref}}$ is selected, the $\mathbf{B}^{\mathrm{res}}$ is computed, as defined in Equation 2.12.


Reference Frame ($\mathbf{F}^{\mathrm{ref}}$)
Original Frame ($\mathbf{F}^{\mathrm{ori}}$)
Coding order
Set of candidate blocks (S):
-4
-3
-2
-1
0
1
2
3
4
Original block ($\mathbf{B}^{\mathrm{ori}}$)
Residue block ($\mathbf{B}^{\mathrm{res}}$)
Reference block ($\mathbf{B}^{\mathrm{ref}}$)

Source: Seidel (2019).

$$j_{\mathrm{cost}}(\mathbf{B}^{\mathrm{ori}}, \mathbf{B}^{\mathrm{can}}) = distortion\left(\mathbf{B}^{\mathrm{ori}}, \mathbf{B}^{\mathrm{can}}\right) + \lambda \times rate\left(\mathbf{B}^{\mathrm{ori}}, \mathbf{B}^{\mathrm{can}}\right) \tag{2.13}$$

A simplified way to obtain a rate estimate is by considering the size $(g(a))$ of exponential Golomb codes to represent the residual MV coordinates, as shown in Equation 2.14 (Trudeau; Coulombe; Desrosiers, 2014).

$$rate(\overrightarrow{mv}) = rate(x, y) = g(x) + g(y) \tag{2.14}$$

where $g(a)$ is the number of bits to encode the given a using exponential Golomb codes (Trudeau; Coulombe; Desrosiers, 2014), such a function being defined in Equation 2.15.

$$g(a) = 2 \times \lfloor \log_2(2 \times |a| + 1) \rfloor + 1 \tag{2.15}$$

To further increase coding efficiency, ME is usually split into 2 steps, Integer Motion Estimation (IME) and Fractional Motion Estimation (FME). IME is the initial search within a search area in a $\mathbf{F}^{\text{ref}}$, as illustrated in Figure 10. FME is a refinement over IME, where new $\mathbf{B}^{\text{can}}$ are artificially generated in fractional positions, i.e., between pixels around the $\mathbf{B}^{\text{ref}}$ chosen by the IME.

3 FRACTIONAL MOTION ESTIMATION IN VVC AND RELATED WORK

This chapter begins with a description of the most relevant aspects of FME, while highlighting the particular features of the VVC FME. Then, in Section 3.3 it presents and discusses the proposals of dedicated FME hardware architectures found in the literature.

FME is further divided into two steps:

1. **Interpolation**: were samples in fractional positions around the $\overrightarrow{mv}^{\text{ime}}$ are generated to form new $\mathbf{B}^{\text{can}}$s;
2. **Block Matching**: where a BMA searches the new $\mathbf{B}^{\text{can}}$s to find a refined $\mathbf{B}^{\text{ref}}$.

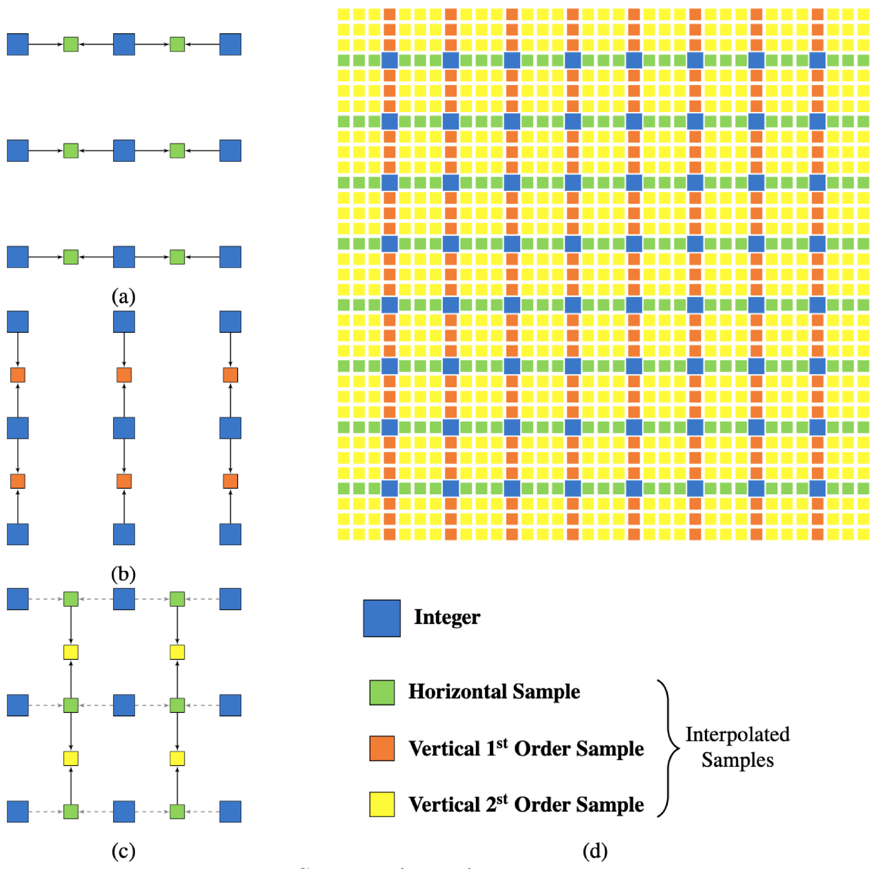
The FME is performed independently for each channel. In HEVC, the luma samples have a CU with 1/4-precision positions, i.e., the interpolation can artificially generate 4× the number of $\mathbf{B}^{\text{can}}$s, requiring two more bits to represent the $\overrightarrow{mv}^{\text{res}}$. These samples are generated using a 7/8-tap filters. The taps indicate the number of samples necessary as inputs, e.g., a 7-tap filter assembles a luma sample from an array of seven samples. VVC, in turn, introduced Advanced Motion Vector Resolution (AMVR), which enables each CU to be encoded with a different MV precision. When AMVR is set to default, the CU has 1/4-precision as in HEVC, while it can change the FME precision to 1/2-precision (CHEN; YE; KIM, 2020).

3.1 INTERPOLATION

As illustrated in Figure 11, samples in fractional positions, often referred to as fractional samples, are generated in both horizontal and vertical directions. They are classified in one of three types, according to the array of samples they are generated with: Horizontal Samples (HS), First-Order Vertical Samples (FOVS) and Second-Order Vertical Samples (SOVS). HS are generated with an horizontal array of integer samples as inputs in the interpolation filter, whereas FOVS and SOVS are generated with a vertical array of samples. However, observe that HS are used to generate SOVS and therefore, the generated HS need to be temporarily stored which in hardware implementations ,must be done in a dedicated buffering unit.

To generate the fractional samples, the inputs are processed with an interpolation filter. Table 2 shows the coefficients of the VVC interpolation filters. Each input is multiplied by a filter coefficient depending on its position with respect to the sample to be interpolated. The quarter-pel coefficients (q_i) are use to generate samples with 1/4 MV precision, i.e., half-pel samples, e.g., $\overrightarrow{mv}^{\text{res}} = (0.25, -0.5)$. The half-pel coefficients (h_i) are used for samples with 1/2 MV precision, i.e., quarter-pel samples, e.g., $\overrightarrow{mv}^{\text{res}} = (0.5, -0.5)$. The alternative h_i^{alt} are use for half-pel samples whenever the AMVR sets the FME precision to 1/2.

Figure 11 – FME interpolated samples in fractional positions. (a) Horizontal Samples (HS) interpolation. (b) First-Order Vertical Samples (FOVS) interpolation. (c) Second-Order Vertical Samples (SOVS) interpolation. (d) All pixel samples that may be interpolated at 1/4-precision positions around a 8 × 8 integer position block.



Source: Seidel (2019).

Table 2 – Coefficients defined by VVC for the interpolation of luma samples for FME, where i is the position of the closest input position to the sample being interpolated, and $i+1$ is the second closest one.

i	**-3**	**-2**	**-1**	**0**	**1**	**2**	**3**	**4**
q_i	-1	4	10	58	17	-5	1	
h_i	-1	4	-11	40	40	-11	4	-1
h_i^{alt}	0	3	9	20	20	9	3	0

Source: ITU-T (2020).

Equation 3.1 to Equation 3.4 define the VVC FME interpolation luma filters, considering samples with bit-depth b. Each input is multiplied by a filter coefficient depending on its position with respect to the sample to be interpolated. The results are summed up, divided by 64 (same value as the sum of the filter coefficients). Therefore, the filters work as a weighted arithmetic mean of the input samples, where the position of the fractional sample will dictate the weights of the input. The results are then clipped, i.e., limiting the values to match the interval of representation according to the bit-depth, thus avoiding overflow. For example, if the integer samples are represented with 8 bits, the maximum value is 255, therefore all values of fractional samples that surpass this threshold are clipped to 255 to avoid overflow. Similarly, fractional samples with negative values are clipped to 0. Filters up (Equation 3.1) and down (Equation 3.4) are applied to quarter-pel samples, whereas filters middle (Equation 3.2) and middle$^{\mathrm{alt}}$ (Equation 3.3) generate half-pel samples, for 1/4 and 1/2 MV precision, respectively.

$$\mathrm{up_k} = clip\left(0, 2^b{-}1, \sum_{i=0}^{6}(q_{i-3} \times in_{k+i-3}) >> 6\right) \tag{3.1}$$

$$\mathrm{middle_k} = clip\left(0, 2^b{-}1, \sum_{i=0}^{7}(h_{i-3} \times in_{k+i-3}) >> 6\right) \tag{3.2}$$

$$\mathrm{middle}_k^{\mathrm{alt}} = clip\left(0, 2^b{-}1, \sum_{i=0}^{5}\left(h_{i-2}^{alt} \times in_{k+i-3}\right) >> 6\right) \tag{3.3}$$

$$\mathrm{down_k} = clip\left(0, 2^b{-}1, \sum_{i=0}^{6}(q_{3-i} \times in_{k+i-3}) >> 6\right) \tag{3.4}$$

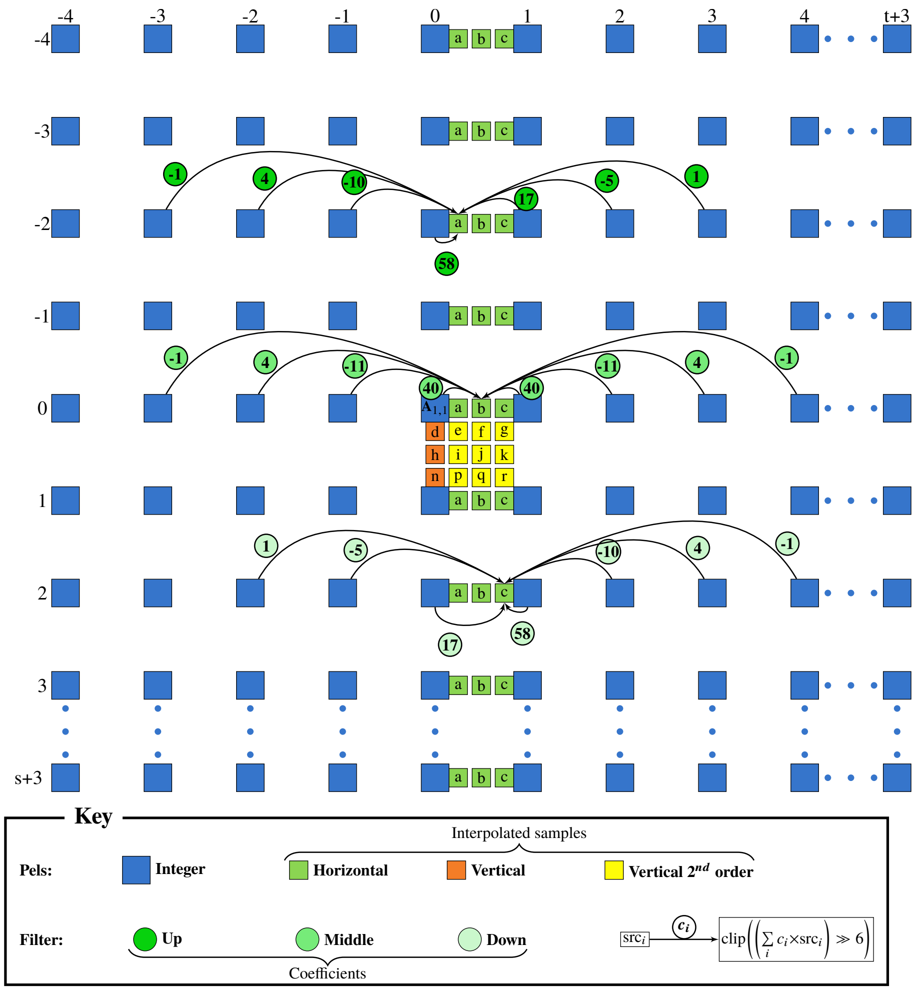
Figure 12 illustrates the interpolation process for three HS, utilizing the three 1/4-precision filters, up, middle and down. The process for FOVS and SOVS is similar, just changing the x axis by the y axis. A fractional sample requires three to four input samples in each direction. If for example the current block size is 8 × 8, as seen in Figure 11, the HS on the edge will need 4 integer samples from the right and left outside the $\mathbf{B}^{\mathrm{ref}}$, same for FOVS but on the top and bottom. For SOVS, it is required extra 4 rows of HS on the top and bottom to be interpolated prior to the interpolation of SOVS. Therefore, if the current block size is, for example, 8×8, a 16×16 block of integer sample centered around the $\mathbf{B}^{\mathrm{ref}}_{\mathrm{ime}}$ is required. For a block size of 16×16, a 24×24 integer sample block is required.

3.2 BLOCK MATCHING

In VTM, the VVC FME has 48 luma fractional $\mathbf{B}^{\mathrm{can}}$ when set to 1/4-precision, and 8 luma $\mathbf{B}^{\mathrm{can}}$ when set to the alternative 1/2-precision, as illustrated by Figure 13. As seen in Section 2.5, for the reconstruction of an inter-predicted block, it is necessary the $\mathbf{B}^{\mathrm{res}}$ and $\overrightarrow{mv}$. Furthermore, for the reconstruction of a fractional inter-predicted block, the decoder needs the $\mathbf{B}^{\mathrm{res}}$, the $\overrightarrow{mv}^{\mathrm{fme}}$, and the interpolation filter used to generate the $\mathbf{B}^{\mathrm{ref}}$. Therefore, the filters coefficients must be defined by the video coding standard, allowing in that way, that any decoder has the tools to reconstruct the fractional samples of $\mathbf{B}^{\mathrm{ref}}$ at the $\overrightarrow{mv}^{\mathrm{fme}}$ position. However, the metrics to chose this block as $\mathbf{B}^{\mathrm{ref}}$ are not necessary for its reconstruction, leading to design opportunities where a BMA is chosen to fit the requirements of certain application.

There are numerous algorithms to perform the block matching. One possible solution is applying a FBMA to the FME which, as discussed previously, computes the

Figure 12 – Interpolation of luma samples in VVC. The three types of filters are depicted generating horizontal samples. As most fractional pixels will be interpolated in the vertical direction, we named the filters which generate them considering the position of the central coefficient ($i = 0$) used for their interpolation: up (7-tap), middle (8-tap) and down (7-tap). Up is used for samples in positions a, d, e, f and g; middle for b, h, i, j and k; down for c, n, p, q and r.

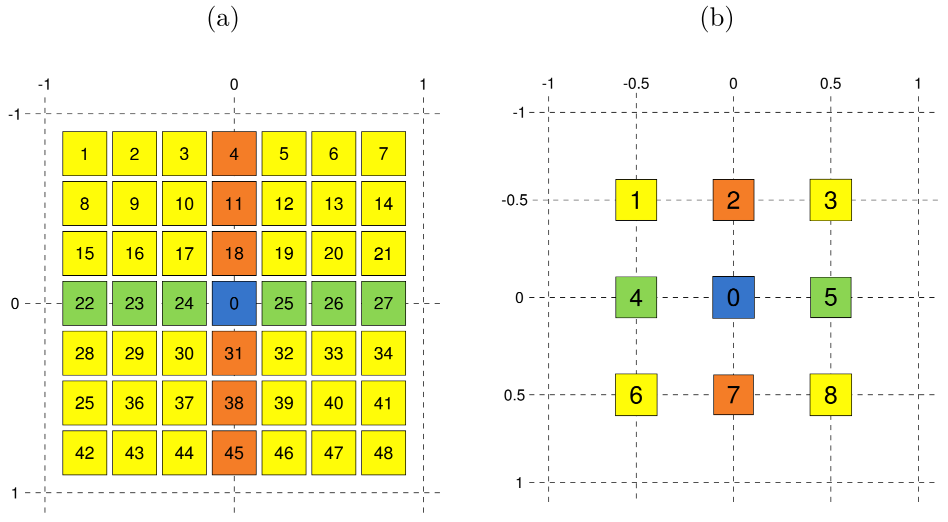


Source: adapted from Diniz et al. (2015), Afonso et al. (2015), Afonso et al. (2016).

cost for every $\mathbf{B}^{\text{can}}$ requiring a lot of computation. Another approach is applying a BMA for a smaller search window which can be a more viable option since it relies on fewer $\mathbf{B}^{\text{can}}$s. The reference software models, HM and VTM, implement a 2-step search pattern where 9 $\mathbf{B}^{\text{can}}$s are evaluated in each step, as illustrated in Figure 14. At first search, also referred to as half-pel search, the BMA computes the *cost* of the eight $\mathbf{B}^{\text{can}}$s with 1/2-precision and in the integer position, as illustrated in Figure 14a. In a second search, also referred to as quarter-pel search, the BMA evaluates the $\mathbf{B}^{\text{can}}$s around the chosen candidate in the first search, as well as the chosen candidate, as illustrated in Figure 14b. When the precision of the FME is set to 1/2, the VTM executes only the first search, which already includes all available fractional candidates for such precision.

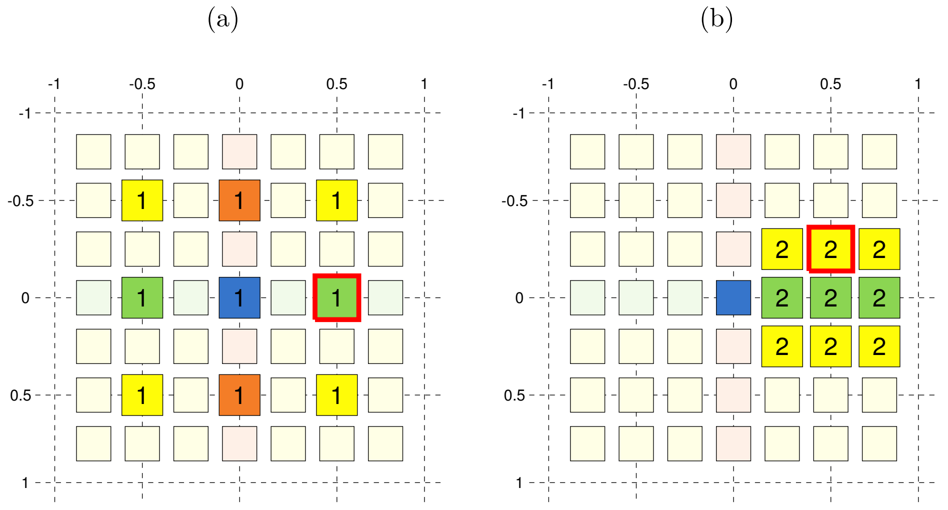
In the example of Figure 14, after executing both search steps, the $\mathbf{B}^{\text{can}}$ in the coordinates $(x_{frac}, y_{frac}) = (0.5, -0.25)$ is the best position according to the BMA, leading to the lowest value of the metric cost according to its algorithm, therefore being chosen

Figure 13 – All luma candidates available in the VVC FME, as defined in VTM, for the two modes of operation: (a) 1/4-precision and (b) 1/2-precision.



Source: the author.

Figure 14 – VTM default 2-step search pattern for the FME. (a) In the first search, eight $\mathbf{B}^{\text{can}}$ in fractional positions plus the one in integer position (0,0) are evaluated by the BMA. The eight searched positions are always the same. (b) Second search around the $\mathbf{B}^{\text{can}}$ picked in the first step (0.5,0). The nine $\mathbf{B}^{\text{can}}$ are evaluated once again by the BMA and the best one (0.5,-0.25) according to the metric cost is chosen as $\mathbf{B}^{\text{ref}}$.



Source: the author.

as $\mathbf{B}^{\text{ref}}$. The value of the $\overrightarrow{mv}$ is then updated to incorporate the fractional precision by adding the fractional coordinate to the $\overrightarrow{mv}^{\text{ime}}$, as defined by Equation 3.5.

$$\overrightarrow{mv}^{\text{fme}} = \overrightarrow{mv}^{\text{ime}} + (x_{frac}, y_{frac}). \tag{3.5}$$

Even though it is used in the reference model of both VVC and HEVC, this search algorithm has a few shortcomings. For example, for hardware implementations, when designing the FME with this search algorithm, the architecture may interpolate only the 8 $\mathbf{B}^{\text{can}}$s of the first search, and once the BMA has chosen the best, it interpolates the last 8 $\mathbf{B}^{\text{can}}$s, leading to big data stalls. Or, the architecture may interpolate all samples before executing the two-step search, in this case, leading to unnecessary calculations, since only 16 $\mathbf{B}^{\text{can}}$s of a total of 48 $\mathbf{B}^{\text{can}}$s are computed by this BMA.

Section 3.3 discusses efficient algorithms to implement the FME interpolation and/or search in dedicated hardware architectures.

3.3 RELATED WORK ON FME HARDWARE DESIGN

This section explores the main ideas behind the related work selected during the conducted systematic literature review[1]. They propose different alternatives to the FME implementation by changing the interpolation filters coefficients or proposing new search methods. It is worth mentioning that to the best of our knowledge, no work on the interpolation and/or search for the FME VVC has been published yet. Nonetheless, this work on the VVC FME can be compared to the related work applied to HEVC since it is HEVC compatible.

Badry, Shalaby & Sayed (2018) present an interpolation-free algorithm for the FME. They propose to select the $\mathbf{B}^{\text{ref}}$ based on the statistical analysis of the matching error surface around the integer sample MV location (BADRY; SHALABY; SAYED, 2018). Such strategy led to a degradation in BD-Rate of 2.2%, the highest amongst all related work, while obtaining a 96% reduction of the FME computational cost.

Lv et al. (2014) tested four different tap interpolation filters, a 4-tap, a 6-tap, an 8-tap (slightly different from the standard) and a 10-tap filter in the HM v11.0 with the LD-P configuration. The proposed method consists in using a resolution-adaptive interpolation filter, therefore selecting the best interpolation filter to be used based on the resolution of the video being coded. After the interpolation, it utilizes the standard FME search from HM which, in a hardware implementation, will lead to stalls, repetitive memory accesses and operations. Their results show that for Class A sequences of the HEVC CTC the 4-tap filter led to the best BD-Rate, with an average of -3.4% for the luma component. For the Class B, the best results came from the 6-tap, averaging a BD-Rate of 0.0%. It was also the best for the Class E, with an average of -1.0% in BD-Rate. For classes C and D, the 10-tap filter brought the best results, with averages in BD-Rate for the luma of -0.5% and -1.5%, respectively. Even though, this filter design could be used in a more efficient search, if the mentioned architectural issues were fixed. On the other hand, a hardware implementation of multiple idle filters will have a negative impact on occupied area and leakage power.

Kalali & Hamzaoglu (2018) proposed two approximate HEVC interpolation filters, each one composed of a 3-tap and a 4-tap for the samples that are interpolated with a 7-tap and an 8-tap in HM, respectively. Unlike Lv et al. (2014), Kalali & Hamzaoglu

[1] Appendix B brings the methodology employed in the systematic review of the literature.

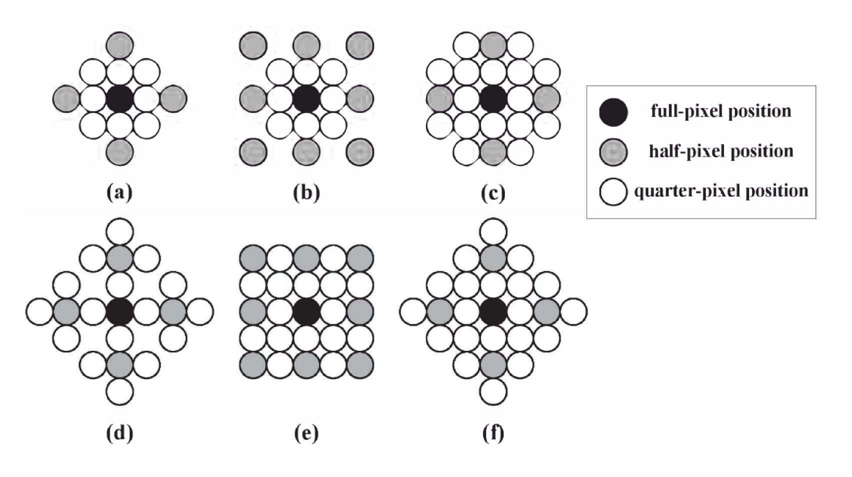
(2018) compare both filters instead of using them together. The filters are implemented with Multiplierless Constant Multiplication (MCM), which reduces the number and sizes of the adders. The interpolation architecture is implemented in an FPGA board. It interpolates all the candidate samples for an 8×8 Prediction Unit (PU) block in 44 clock cycles. The proposed filters presented a degradation in BD-Rate of 0.40% and 1.14% for the luma samples. The first ten frames of each sequence were coded with the LD-P configuration in HM 15.0. This architecture consumes up to 67.1% less energy than the original hardware proposed by the same authors. Such results are expected since the author removed more than half of the coefficients in each filter, resulting not only in smaller filters, but also in less memory reads to feed the filters.

He et al. (2015) proposed a bilinear quarter-pixel approximation together with a new search pattern to reduce the complexity of the FME. Their proposed interpolation unit generates the 24 fractional samples that are closest to the integer sample. However, only 12 candidates are searched, such selection being based on the evaluation of the neighbors integer samples costs. A SATD is performed at the search module. The output of the architecture is the fractional MV. It only supports partitions of 16×8 PU sizes and larger. Results generated with HM 10.0 and selected resolutions showed a BD-Rate degradation of 2.07% for their proposed work, while claiming to improve the power efficiency by 52%.

Lim et al. (2016) presented a fast FME algorithm by reducing the number of searched points. The authors ran simulations with HM 13.0 and observed that during the first step of the search, the half fractional candidates located in one of the x/y-axis had a 45.3% chance of being picked against 7.4% of the samples outside the axes, while integer sample is picked in the 47.3% of the remaining cases. For this reason, Lim et al. (2016) propose an algorithm that either skips the 1-step search or performs it only on the 4 candidates located in one of the x/y-axis. The 2-step, instead of searching all the 8 candidates around the candidate chosen in the 1-step (Figure 14b), searches only 3 candidates, the ones between the half position chosen and the integer position. If the 1-step is skipped, then the search module evaluates only the four closest quarter samples located at the x/y-axis. The algorithm leads to a BD-Rate of 0.93% for the Random Access (RA) configuration and 0.68% for the Low Delay (LD) configuration, when coding the first 32 frames of five selected sequences. Even though those results are impressive when compared to other works, this algorithm can mainly be applied to software implementations of the FME. An efficient hardware implementation would require a more consistent pattern of search, since a lot of quarter candidates searched would not be available, leading to big data stalls. This could be avoided by an early computation of some quarter samples, especially the horizontal ones, although they may lead to some unnecessary calculations. Moreover, for an efficient hardware implementation, the FME would ideally interpolate $\mathbf{B}^{\text{can}}$ in positions that share the inputs, at the same time searching $\mathbf{B}^{\text{can}}$ that share samples, thus avoiding repetitive operations and hardware stalls.

To avoid the two-interaction search process and the full FME computation, Ding, Ye & Wang (2015) evaluated the six patterns of candidates shown in Figure 15. Table 3 shows the coding efficiency of these patterns, evaluated in HM 16.0 following the CTC. Then, they created an adaptive pattern by merging (a), (c) and (e). According to the authors, the patterns can be selected according to the requirements of the applications.

Figure 15 – Search evaluation patterns proposed and evaluated by Ding, Ye & Wang (2015).



Source: Ding, Ye & Wang (2015).

Table 3 – Coding efficiency of the patterns presented by Ding, Ye & Wang (2015).

Pattern	BD-Rate Y (%)				Searched positions
	random access	low delay	low delay P	average	
(a)	1.4	1.3	2.1	1.6	12
(b)	1.0	0.9	2.2	1.4	16
(c)	0.6	0.6	1.5	0.9	20
(d)	1.0	1.0	2.4	1.5	20
(e)	0.5	0.5	1.2	0.7	24

Source: Ding, Ye & Wang (2015).

A number of works addressing hardware solutions for the HEVC FME can be found in the literature. In order to achieve the optimal coding efficiency possible, most of them propose hardware architectures that perform interpolation and search with all 48 candidates by implementing high levels of parallelism and avoiding repetitive memory accesses (KALALI; HAMZAOGLU, 2014; PASTUSZAK; TROCHIMIUK, 2015; AFONSO et al., 2016; FILHO et al., 2020; SEIDEL et al., 2021). However, performing a full BMA in the FME results in only 0.14% BD-Rate gain when compared to the 18-candidate search implemented in HM. Hence, discussing those works would not add much since they all have the same coding efficiency. A single exception is made for the work of Seidel et al. (2021), which is discussed in some detail in Section 5.1. The hardware architecture proposed by these authors will be discussed because it was selected to serve as baseline to this work since it was developed in the same research group, granting full access to its RTL descriptions and testbench. This made it possible to reuse some building blocks and use the same design tools, so as to establish fair comparisons. Consequently, the synthesis results mostly reflect the changes in the architecture, leading to a better conclusion on the impacts and trade-offs of the proposed algorithm.

Although no work on the interpolation and/or search for the FME VVC could be found in the literature, a few works propose the implementation of hardware accelerators for the interpolation filters of VVC, targeting the Motion Compensation (MC) Azgin, Kalali & Hamzaoglu (2020), Silva et al. (2021), Mahdavi & Hamzaoglu (2021). MC is executed during the reconstruction of the encoded frame: it processes the information of the ME to reconstruct inter-predicted frames. Therefore, the MC requires the implementation of all filters supported by the video coding standard, allowing it to always reconstruct the information correctly. MC has 15 different interpolation filters for the fractional MV precision of up to 1/16 pixel supported by VVC. Since these three works target the MC rather than the ME, they cannot be compared to the woks addressing the FME due to major differences between the two processes.

Table 4 shows a comparison between the results of related work that propose dedicated hardware architectures for the HEVC FME. It is worth mentioning that the FME VVC architecture proposed in this work can be compared to those in Table 4 because it is HEVC compatible.

Table 4 – Comparison between related work results.

Related Work	**Badry et al. 2018**	**Kalali et al. 2018**	**He et al. 2015**	**Ding et al. 2015**	**Afonso et al. 2015**	**Seidel et al. 2021**
Standard	HEVC	HEVC	HEVC	HEVC	HEVC	HEVC
Technology	65 nm	90 nm	65 nm	Xilinx FPGA	45 nm	45 nm
Maximum Frequency (MHz)	602	300	188	200	397	400
Maximum Resolution	2160p@60fps	2160p@45fps	4230p@30fps	2160p@60fps	2160p@60fps	2160p@60fps
Gate Count (K)	26.6	13.2	1183	n.a.	148	n.a.
Total Power (mW)	18.6	16.4	198.6	n.a.	15.8	27.9
BD-Rate (%)	2.2	n.a.	n.a.	2.4	4.0	1.2

4 IMPACTS OF THE FME AND PROPOSAL OF VVC FME SEARCH PATTERNS

This chapter discusses the impact of the FME on VVC. Particularly, it presents coding efficiency results obtained by disabling the FME on the VTM v13.0 (BOSSEN; LI; SUEHRING, 2020a). It also proposes fixed search patterns for the VVC FME and evaluates them in terms of coding efficiency.

4.1 IMPACTS OF THE FME

In the HM, FME is responsible for up to 33.88% of inter prediction encoding time (BLASI et al., 2015). Particularly, 46.9% of FME time is spent interpolating the half and quarter samples, while the remaining 53.1% is spent at the BMA (BLASI et al., 2015). Disabling FME entirely when encoding UHD sequences accounts for a Bjøntegaard Delta Bitrate (BD-Rate) average increase of 7.01% (BLASI et al., 2015).

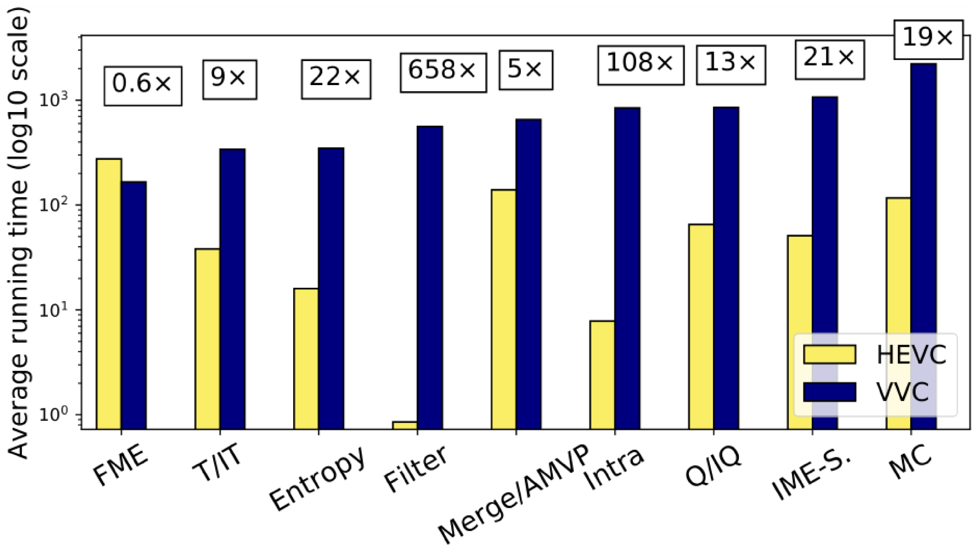
Although several works have been published on the VVC all along its development, and the number of works on the complexity and impacts of different tools is evergrowing, there is still a lack of studies that explore the complexity and impacts of the VVC FME. Siqueira, Correa & Grellert (2020) performed rate-distortion and complexity comparison of HEVC and VVC by using HM v16.9 and VTM v5.0 reference models, respectively, set for the RA profile and encoding for Quantization Parameter (QP) 22, 27, 32 and 37, in accordance with the CTC manual. The profiling results are presented in Figure 16, which shows the complexity per module for both reference models. As one may observe, from all modules presented in Figure 16, FME is the only that decreased in complexity. Siqueira, Correa & Grellert (2020) attribute the Single Instruction/Multiple Data (SIMD) optimization in VTM to cause this large decrease in the processing time of this step.

In Chapter 3, it was discussed the introduction of the AMVR in VVC, a new tool that is capable of changing the MV precision. While HEVC only computes the FME in quarter-pel precision, VVC may compute the FME in quarter-pel or in a simplified half-pel. When running in half-pel precision, the complexity of the FME is considerably lower, having only 8 $\mathbf{B}^{\text{can}}$s when compared to 48 $\mathbf{B}^{\text{can}}$s for quarter-pel precision. In such a case, VTM interpolates less candidates and executes a single search, thus lowering the overall complexity of the FME in VVC, when compared to its predecessor. Therefore, the introduction of AMVR may also decrease the processing time of the FME.

Oppositely, beyond the inter coding features in HEVC, a number of new and refined inter prediction coding tools were added to allow VVC to reach a substantial improvement in coding efficiency (CHEN; YE; KIM, 2020), namely:

- Extended merge prediction.
- Merge Mode with Motion Vector Difference (MVD).

Figure 16 – Average running time per module for VVC and HEVC.



Source: Siqueira, Correa & Grellert (2020).

- Advanced Motion Vector Prediction (AMVP).
- AMVP mode with symmetric MVD signaling.
- Affine motion compensated prediction.
- Subblock-based temporal motion vector prediction.
- Motion field storage: $1/16^{th}$ luma sample MV storage and 8x8 motion field compression.
- Bi-prediction with CU-level weight.
- Bi-directional optical flow.
- Decoder side motion vector refinement.
- Triangle partition prediction.
- Combined inter and intra prediction.
- Adaptive Loop Filter (ALF).

The results of the introduction of these inter prediction coding tools can be observed by the increase in complexity of the inter modules (Figure 16). The Filter is the module that presents the largest gap between VVC and HEVC. Siqueira, Correa & Grellert (2020) indicate that the introduction of the new ALF tool is the main cause of the said increase. Siqueira, Correa & Grellert (2020) also observed that the MC is the most complex step in VVC, mainly due to the new inter prediction coding tools, such as Affine Motion Compensated Prediction. Consequently, the addition of many new tools reduces the impacts of the FME in total encoding time and coding efficiency. As a matter of fact, the more features and complexity are introduced to many different tools, the lower will be the individual impact of each one to the overall encoding time and coding efficiency of the video standard.

4.2 EXPERIMENTAL ASSESSMENT OF THE CODING EFFICIENCY IMPACT OF THE FME ON VVC

So far, no work was found in the literature showing experimental results or even estimates of the overall impacts of the FME on the VVC coding efficiency. Hence, the first contribution of this work relies on such qualitative assessment. To accomplish that, the VTM v13.0 (BOSSEN; LI; SUEHRING, 2020a) was executed following the CTC (BOSSEN et al., 2019), for the FME disabled and for the default configuration of the VTM's FME, the latter being used as baseline for BD-Rate calculations. The configurations used were Low Delay with P slices only (LD-P) and RA. In LD-P configuration, after the first frame is encoded (using only intra-prediction, i.e., I-slice) the remaining frames are encoded using only information of past frames which were either intra (I-slice) or inter predicted (P-slice). In RA, an I-slice is encoded every N frames, while the remaining ones are B-slice or Bi-Directional. Blocks in B-slice frames can be predicted using information of frames from the past and future, switching the encoding order, resulting in a better compression performance. However, this switching of the encoding order of frames will cause decoding delays since a frame will have to wait until all frames that it references are decoded, because it may include future frames. Therefore, LD-P is optimal for applications that require extreme low delay even at the cost of worst coding efficiency, such as video conferencing or live streams. Table 5 shows the obtained results. A positive BD-Rate means that more bits are needed for the same quality level, whereas a negative value indicates improvement in coding efficiency when compared to a baseline, which in this particular analysis is VTM v13.0 with the default search pattern.

The average BD-Rates are 0.95% and 0.61% for LD-P and RA, respectively, thus indicating coding efficiency reductions, particularly for some classes, such as C and D, disabling the FME has more negative impact than for others, such as A1 and A2. However, the impact of disabling the FME in VVC is far lower from the average BD-Rate of 7.01% reported by Blasi et al. (2015) when disabling the FME in HEVC. In light of the discussion presented in Section 4.1, it is fair to speculate that the newly introduced inter prediction tools are responsible for finding new $\mathbf{B}^{\mathrm{ref}}$s for the blocks originally predicted with FME, thereby compensating, to some extend, the negative impacts of disabling the FME.

Moreover, VVC brings a multitude of new tools while also improving the old ones to further increase the coding efficiency, with bitrate savings above 40% (PAKDAMAN et al., 2020), which causes an increase in encoding time of around 10× (SIQUEIRA; CORREA; GRELLERT, 2020) when comparing VTM and HM. As previously discussed in this section, in the case of FME, VVC introduced simplifications rather than complexity. In conclusion, having assessed the impact of FME in VVC and considering the increase in complexity of this new video coding standard, the next research steps aimed to find a way of simplifying the FME envisaging a hardware implementation that could be both

Table 5 – Results of BD-Rate when disabling the FME in VTM v13.0 on LD-P and RA configurations. The (∗) indicates sequences that were encoded only for the first second of the video.

Class	Sequence	**BD-Rate (%)**	
		LD-P	RA
A1	Tango2*	0.072694	-0.026479
	Campfire*	0.500769	0.106302
	FoodMarket4*	0.236226	-0.006388
A2	CatRobot*	0.338789	0.286452
	DaylightRoad2*	0.833952	0.507905
	ParkRunning3*	0.408677	0.303408
B	MarketPlace	0.787148	0.578650
	RitualDance	0.456016	0.541155
	Cactus	0.392880	0.224483
	BasketballDrive	1.718843	0.907799
	BQTerrace	1.857627	0.942731
C	RaceHorsesC	1.132070	1.380367
	BQMall	1.226958	0.922000
	PartyScene	1.706121	1.039828
	BasketballDrill	1.445063	0.505261
D	RaceHorses	1.230089	1.832394
	BQSquare	1.777032	0.924396
	BlowingBubbles	2.334300	1.475541
	BasketballPass	1.618316	1.193067
E	FourPeople	0.773862	0.344030
	Johnny	0.698289	0.401921
	KristenAndSara	0.670167	0.347910
F	ArenaOfValor	0.337606	0.477251
	BasketballDrillText	1.563618	0.580196
	SlideEditing	0.292683	-0.016440
	SlideShow	0.268420	0.195042
Average		0.95	0.61

coding and energy efficient.

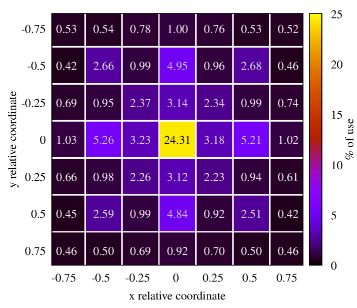
4.3 PROPOSED FIXED SEARCH PATTERNS

This section describes the process of designing fixed search patterns especially tailored to reduce the VVC FME complexity for a Very-large-scale Integration (VLSI) architecture. The devised search patterns constitute the second contribution of this work.

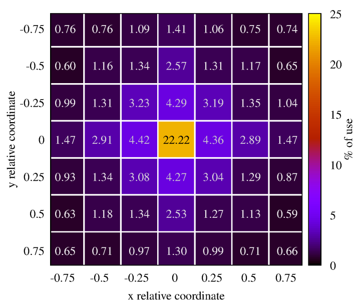
To define the course of action, the VTM v6.2 (BOSSEN; LI; SUEHRING, 2020b) was executed to measure the percentage that each $\mathbf{B}^{\mathrm{can}}$ position is chosen as $\mathbf{B}^{\mathrm{ref}}$ by the default BMA of the VTM FME. The results are illustrated in Figure 17, were each figure shows the average of the heatmap signature of the chosen $\mathbf{B}^{\mathrm{ref}}$ position by the FME, using the distortion metric SATD and following the CTC (BOSSEN et al., 2019), for configurations LD-P and RA. As commented in Section 3.1, the FME has two modes of operation in VTM: a two-step search that outputs a quarter-pel precision MV, and an alternative single-step search that outputs a half-pel precision MV. Figure 17a and Figure 17d show the results for all FME executions, Figure 17b and Figure 17e present the collected data for the cases where the FME executes the two-step search, whereas Figure 17 (c) and (f) show the results for the executions when the alternative one-step search was selected. Regarding the two search modes, for RA configuration the two-step search is selected 70.1% against 29.9% for the alternative more simplified one-step search. For LD-P the results are similar: 69.3% of the FME executions were with the two step against 30.7% for one-step search.

Figure 17 – Heatmaps of selected $\mathbf{B}^{\text{ref}}$ by the FME with SATD in summary of all sequences in the CTC (BOSSEN et al., 2019) for two configurations, LD-P and RA.

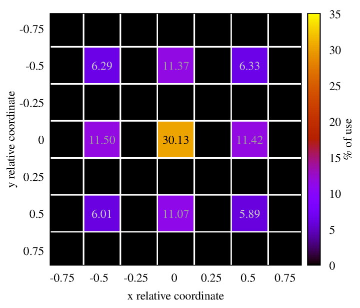
(a) LD-P complete FME.



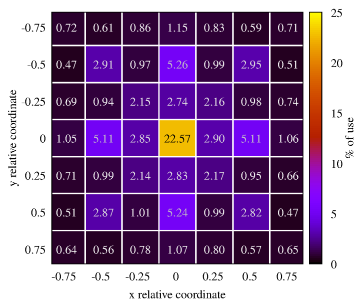
(b) LD-P 2-step.



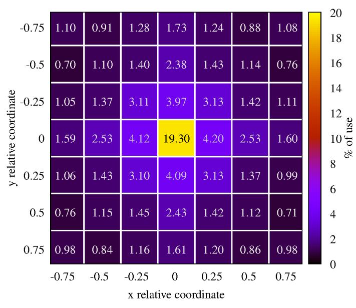
(c) LD-P alternative 1-step.



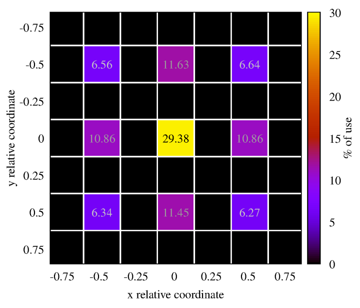
(d) RA complete FME.



(e) RA 2-step.



(f) LD-P alternative 1-step.



Source: the author.

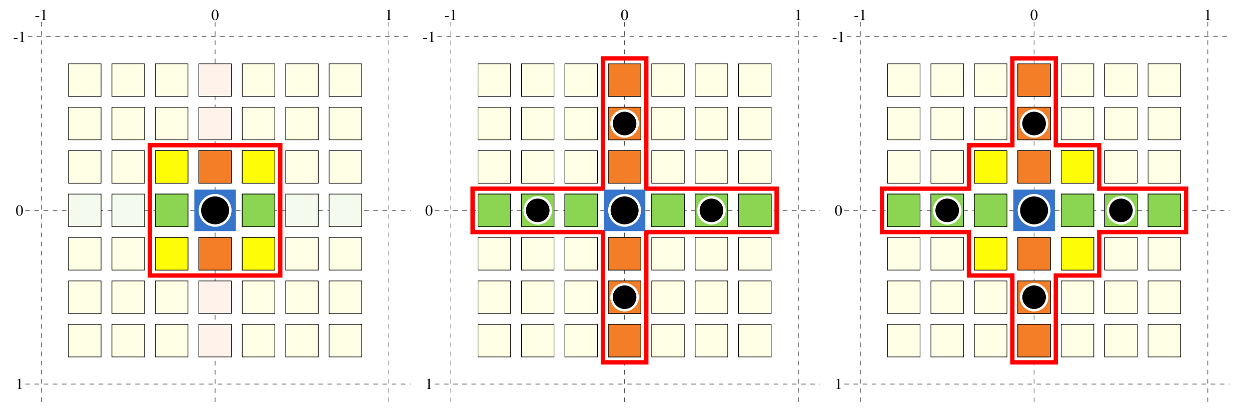
For all heatmaps, the closer the $\mathbf{B}^{\text{can}}$ is to the integer position, the higher the probability of being chosen as $\mathbf{B}^{\text{ref}}$. Furthermore, $\mathbf{B}^{\text{can}}$s located at the x axis ($y = 0$) have a higher probability than $\mathbf{B}^{\text{can}}$s located right above or below. Similarly, $\mathbf{B}^{\text{can}}$s located at the y axis ($x = 0$) have a higher probability than $\mathbf{B}^{\text{can}}$s located to their left or right. In addition, $\mathbf{B}^{\text{can}}$s on peripheral positions are rarely chosen as $\mathbf{B}^{\text{ref}}$ by the BMA.

Based upon the information provided by the heatmaps it was possible to conceive three fixed search patterns for the VVC FME to be further evaluated using VTM. Those search patterns, shown in Figure 18, consider the (central) integer position and only the most frequently chosen fractional positions.

The *Box* pattern (Figure 18a) searches the integer $\mathbf{B}^{\text{can}}$ and the positions surrounding it, which are the most chosen positions in the configurations of Figure 17b and Figure 17e. The *Cross* pattern (Figure 18b) searches all $\mathbf{B}^{\text{can}}$s composed of HS and FOVS. As seen in Section 3.1, these samples are interpolated directly from the Integer samples, thus avoiding data stalls in the case of a hardware implementation. Although the $\mathbf{B}^{\text{can}}$s in the most external positions have a very low rate of being picked by the BMA (around 1% each), they were added to *Cross* because most of the calculation for their interpolation are already executed to calculate the other included $\mathbf{B}^{\text{can}}$s. The *Diamond* pattern is built

Figure 18 – Search patterns proposed for the VVC FME: (a) Box, (b) Cross and (c) Diamond. The black circles indicate positions tested when the alternative filter is used.

(a) Box. (b) Cross. (c) Diamond.



Source: the author.

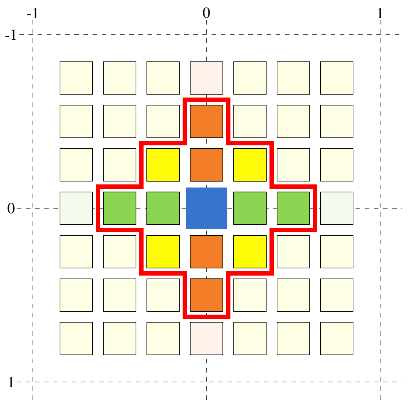
by joining all $\mathbf{B}^{\text{can}}$s searched by *Cross* and *Box*, with the purpose of getting the benefits of directly calculation most of $\mathbf{B}^{\text{can}}$s, while additionally searching some of the $\mathbf{B}^{\text{can}}$s more likely to be chosen as $\mathbf{B}^{\text{ref}}$.

Particularly, the hardware friendliness of these patterns rely on:

1. Minimizing the number of SOVS $\mathbf{B}^{\text{can}}$s, thus reducing the amount of memory required to store the HS. Seidel et al. (2018) conducted a power evaluation of their FME architecture, which showed that the buffer responsible for storing the HS interpolation corresponds to the highest consuming module of the architecture, being responsible for 34.9% of the total power dissipation. The proposed patterns only search a maximum of 4 out of a total of 36 SOVS. Consequently, a smaller portion of the HS needs to be stored and the amount of $\mathbf{B}^{\text{can}}$ search decreases considerably.
2. Interpolating and searching in fixed fractional positions, which eliminates stalls due to mode decisions during its operation. An 8×8 $\mathbf{B}^{\text{can}}$ composed of HS or FOVS takes only 8 clock cycles to be interpolated, whereas a $\mathbf{B}^{\text{can}}$ composed of SOVS takes 24 clock cycles if the necessary HS where not interpolated yet. Another way to eliminate this stall for a not fixed search pattern could be interpolating all $\mathbf{B}^{\text{can}}$s while designing a BMA that computes only a changeable group of $\mathbf{B}^{\text{can}}$s, but this would lead to numerous unnecessary calculations.

As presented in Section 3.3, Ding, Ye & Wang (2015) proposed a similar solution for the HEVC FME. They evaluated six patterns, choosing three of them to propose an adaptive hardware solution. From their six patterns (Figure 15), the one illustrated in Figure 19 shows to be the most promising one when analyzing the coverage area of the percentage its $\mathbf{B}^{\text{can}}$s and its possible hardware minimization. Therefore, their pattern was also evaluated in VTM, along with the three proposed ones. Since Ding, Ye & Wang (2015) refers to this pattern as (a), from this point onward it will be referred to as *Ding (a)*.

Figure 19 – Search pattern proposed by Ding, Ye & Wang (2015) for the HEVC FME, in this work referred to as *Ding (a)*.



Source: Adapted from Ding, Ye & Wang (2015).

From the 48 total candidates available, *Box*, *Cross*, *Diamond* and *Ding (a)* interpolate and search 8, 12, 16 and 12 candidates in fractional positions, respectively. In turn, the default FME implemented in VTM searches 16 candidates in the two step search, where only the first 8 are fixed and the others will be tested depending on the candidate chosen in the first step. It is also worth pointing out that even though these patterns were implemented and evaluated for the VVC, they are compatible with its predecessor, the HEVC.

4.4 CODING EFFICIENCY EVALUATION OF THE FIXED SEARCH PATTERNS

This section presents the coding efficiency of the four fixed patterns, the three proposed ones plus *Ding (a)*. The four search patterns were implemented in the VTM v13.0 (BOSSEN; LI; SUEHRING, 2020a) and tested to encode all video sequences from the CTC (BOSSEN et al., 2019) using profiles LD-P and RA. While the video sequences in classes B through F were completely encoded, only the first second of video sequences in classes A1 and A2 were encoded due to infrastructure limitations. It is worth mentioning that classes A1 and A2 encompass 4K videos, which encoding requires significant computing resources.

Table 6 and Table 7 present the coding efficiency results of the four search patterns for LD-P and RA profiles. *Diamond* resulted in the lowest coding efficiency reductions amongst all patterns, with average BD-Rates of 0.18% and 0.15% for LD-P and RA, respectively, closely followed by *Ding (a)*, with average BD-Rates of 0.19% and 0.16%. Such results were expected since *Diamond* and *Ding (a)* search a higher number candidates than the other two patterns. *Cross* is the pattern with the third lowest coding efficiency reductions, with average BD-Rates of 0.34% and 0.28% for LD-P and RA, respectively, whereas *Box* is the one with the highest reductions, with BD-Rates of 0.57% and 0.44%. While *Cross* lacks the inner SOVS $\mathbf{B}^{\text{can}}$s, *Box* only searches in $\mathbf{B}^{\text{can}}$s around the integer position. Additionally, it seems that changes to simplify the FME or disabling it have a more negative impact in sequences encoded with the LD-P profile than the RA.

Table 6 – BD-Rate results for the fixed patterns using VTM v13.0 for the LD-P profile. The (∗) indicates sequences that were encoded only for the first second of the video.

Class	Sequence	**BD-Rate (%)**			
		Box	*Cross*	*Diamond*	*Ding (a)*
A1	Tango2*	0.177569	0.115786	-0.074266	0.091379
	Campfire*	0.517884	0.136278	0.029105	0.082301
	FoodMarket4*	0.153291	0.122531	0.035496	0.040477
A2	CatRobot*	0.272028	0.156764	0.076500	0.139164
	DaylightRoad2*	0.711394	0.356571	0.208700	0.283688
	ParkRunning3*	0.355336	0.190854	0.113097	0.151363
B	MarketPlace	0.674109	0.245318	0.175464	0.234294
	RitualDance	0.429796	0.180560	0.127796	0.120424
	Cactus	0.444217	0.161237	-0.019382	0.004324
	BasketballDrive	1.663857	0.167225	0.101318	0.145286
	BQTerrace	1.321775	0.589345	0.303804	0.323983
C	RaceHorsesC	0.853766	0.427555	0.367078	0.282956
	BQMall	0.967253	0.336600	0.340489	0.278434
	PartyScene	1.138887	0.596778	0.313923	0.421542
	BasketballDrill	1.419108	0.328124	0.221464	0.199153
D	RaceHorses	0.788592	0.503344	0.305047	0.280567
	BQSquare	1.006846	0.811808	0.403866	0.446245
	BlowingBubbles	1.328506	1.058116	0.513607	0.562799
	BasketballPass	1.349029	0.427350	0.191083	0.154444
E	FourPeople	0.665680	0.170158	0.021398	0.051330
	Johnny	0.614567	0.285275	0.219499	-0.000518
	KristenAndSara	0.547485	0.380600	0.156978	0.056629
F	ArenaOfValor	0.283456	0.147155	0.117576	0.163965
	BasketballDrillText	1.195346	0.559634	0.313910	0.381438
	SlideEditing	0.261484	0.254875	-0.045460	0.033588
	SlideShow	0.231860	0.145683	0.033151	0.027967
Average		0.57	0.34	0.18	0.19

Figure 20 shows a plot with the coding efficiency results of the four fixed search patterns and for the FME disabled for LD-P (top) and RA (bottom), thus allowing for an overall comparative evaluation. For LD-P configuration, for most sequences, especially from classes B, C and D, disabling the FME or implementing *Box* as search pattern compromise coding efficiency the most, by a significant margin. In contrast, *Cross*, *Diamond* and *Ding (a)* have similar BD-Rate results for the majority of classes, with the exclusion of Class D sequences.

For RA configuration, although the coding efficiency is better for all analyzed implementations of FME search, the results are similar, with *Box* being the worst of the search patterns. In this configuration, *Cross* presents some BD-Rate results that are similar to those of *Box*, being a middle ground between *Box* and the *Diamond*.

The class B results illustrate how the different contents of each sequence may influence the coding efficiency drastically, especially for LD-P, even if it is just by applying changes to a single coding tool, in this case the FME. For BQTerrace, *Box* results in BD-Rate increases of 1.66%, whereas *Cross* and *Ding (a)* present BD-Rate of 0.17% and 0.15%,

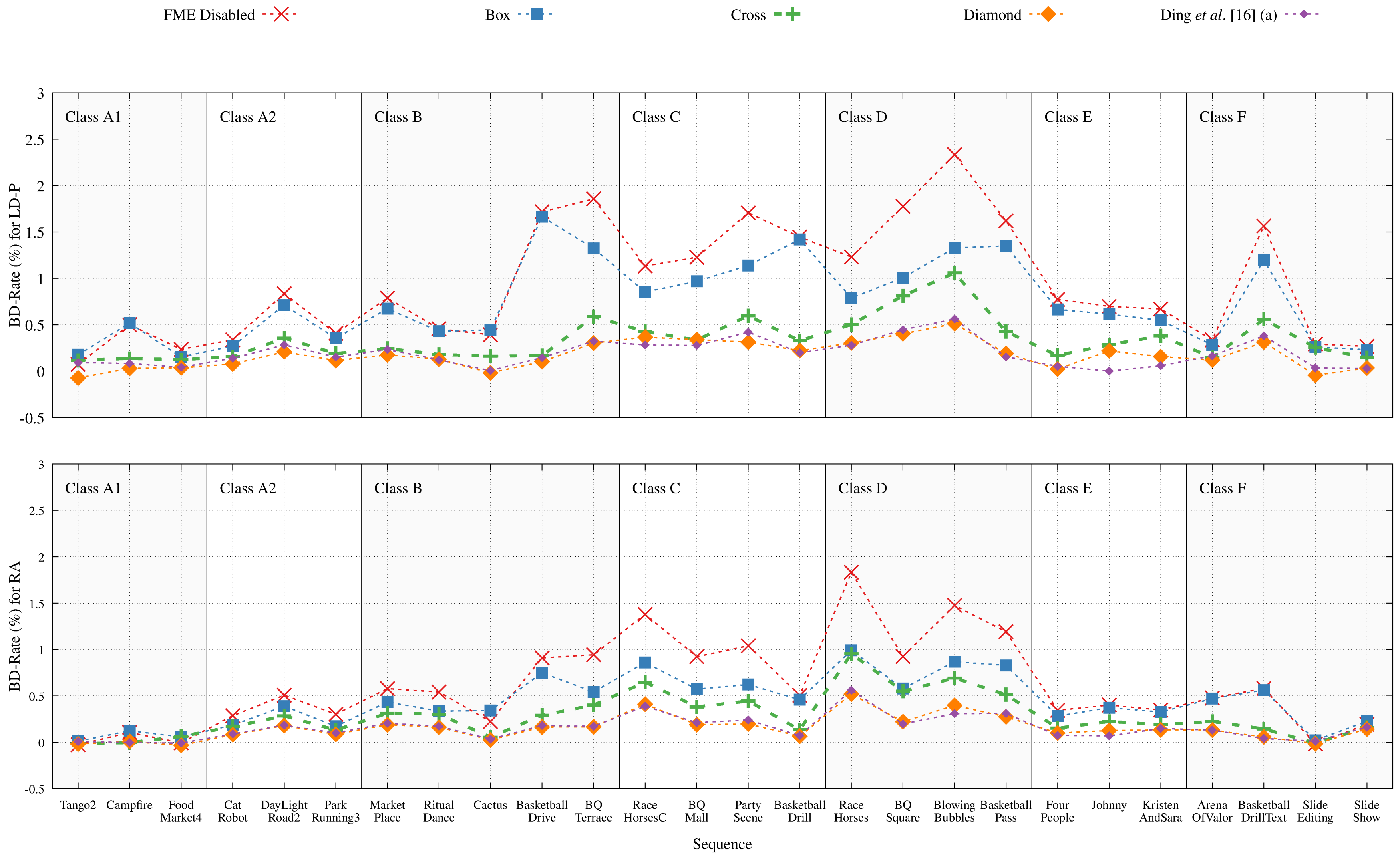
Table 7 – BD-Rate results for the fixed patterns using VTM v13.0 for the RA profile. The (∗) indicates sequences that were encoded only for the first second of the video.

Class	Sequence	**BD-Rate (%)**			
		Box	*Cross*	*Diamond*	*Ding (a)*
A1	Tango2*	0.012811	-0.013829	-0.017488	0.008907
	Campfire*	0.123548	-0.003286	0.008737	-0.001961
	FoodMarket4*	0.060345	0.063181	-0.031658	-0.008866
A2	CatRobot*	0.184295	0.169241	0.082906	0.091013
	DaylightRoad2*	0.388869	0.284623	0.180417	0.184030
	ParkRunning3*	0.173146	0.132987	0.085806	0.100589
B	MarketPlace	0.433381	0.312715	0.189935	0.205904
	RitualDance	0.336134	0.303203	0.163251	0.172462
	Cactus	0.342091	0.055867	0.026741	0.034058
	BasketballDrive	0.748574	0.290906	0.164625	0.181255
	BQTerrace	0.543398	0.401025	0.167158	0.171929
C	RaceHorsesC	0.859642	0.647990	0.410251	0.389228
	BQMall	0.572497	0.381714	0.190935	0.211516
	PartyScene	0.622009	0.445240	0.198033	0.240968
	BasketballDrill	0.458969	0.134769	0.066614	0.072337
D	RaceHorses	0.991085	0.953203	0.519853	0.561271
	BQSquare	0.578639	0.550965	0.221419	0.197478
	BlowingBubbles	0.867539	0.692329	0.397637	0.309402
	BasketballPass	0.828561	0.513310	0.274694	0.310982
E	FourPeople	0.282363	0.152837	0.100081	0.074771
	Johnny	0.373255	0.225192	0.127206	0.069411
	KristenAndSara	0.325838	0.191468	0.134193	0.148991
:	ArenaOfValor	0.470218	0.224508	0.132293	0.132043
F	BasketballDrillText	0.562219	0.144907	0.058323	0.041165
	SlideEditing	0.021667	-0.008066	-0.014123	0.011970
	SlideShow	0.226671	0.159714	0.143143	0.162381
Average		0.44	0.28	0.15	0.16

respectively. Implementing the *Diamond* search results in BD-Rate of 0.10%, increasing a tenth of a percent the number of bits necessary to encode BQTerrace for the same quality, when compared to the reference model of the VVC FME, a 1.56% gap between the worst to the best coding efficiency pattern. For MarketPlace, another class B sequence, the coding efficiency of the search pattern is more similar, with *Box* increasing the BD-Rate in 0.67%, *Cross* and *Ding (a)* in 0.24% and 0.23%, respectively, and *Diamond* with 0.17%, 0.50% separating the best coding efficiency pattern from the worst. In classes A1, A2, E and F the three patterns *Cross*, *Diamond* and *Ding (a)* still outperform *Box* and FME disabled, quite significantly for some sequences, and not so much for others.

In conclusion, *Diamond* and *Ding (a)* have the best coding efficiency out of the four analyzed fixed search patterns. It was expected little to no gap between this two patterns, since the 4 $\mathbf{B}^{\text{can}}$s search for *Diamond* but not included in the search window of *Ding (a)*, have a selection rate of around 1% each (Figure 17a and Figure 17d). Although not as coding efficient as *Diamond* and *Ding (a)*, *Cross* is a close third, keeping the BD-Rate increases close to the two for most of the CTC sequences. On the contrary, the *Box* coding efficiency is closer to disabling the FME than to the other search patterns. The main reason for the *Box* poor performance is the alternative 1/2-precision execution, where *Box* has no fractional positions, therefore the integer position is always chosen as reference. On the other hand, *Cross*, *Diamond* and *Ding (a)* search in four $\mathbf{B}^{\text{can}}$s, as illustrated by the black circles in Figure 18.

Figure 20 – Chart with the BD-Rate results for the FME disabled and the fixed patterns (the three proposed ones and *Ding (a)*) using VTM v13.0 for the LD-P (top) and RA (bottom) profiles.



Source: the author.

5 DEDICATED HARDWARE ARCHITECTURE FOR THE VVC FME

This chapter presents the baseline architecture used in this work, as well as the proposed VVC FME hardware architecture.

5.1 THE BASELINE HARDWARE

Before proceeding to dedicated hardware design, the trade-off between coding efficiency reduction of the patterns and consequent hardware minimization was investigated by assuming as baseline the hardware architecture presented by Seidel et al. (2021). Such architecture was developed to perform the HEVC FME full search with 48-candidates. Particularly, in their article Seidel et al. (2021) presented breakdowns of area and power considering each module in the proposed architecture. Hence, by relying on those breakdowns it is possible to estimate area and power reductions achieved by the four patterns considered in this work.

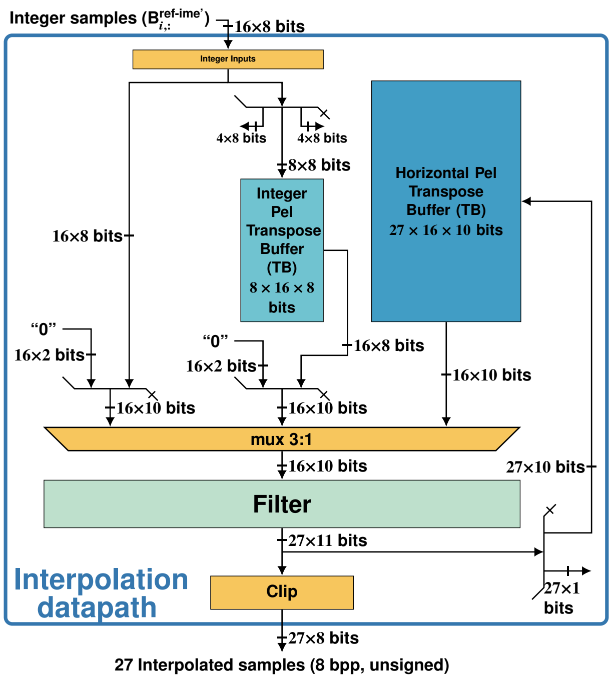
Figure 21 illustrates the hardware architecture proposed by Seidel et al. (2021). The architecture works with 8×8 blocks and takes 51 clock cycles to interpolate and search all 48 fractional candidates available in HEVC FME. The interpolation datapath takes 16 integer samples as input per cycle, a register barrier is located at the beginning of the architecture to maintain the input data stable for the entire clock cycle, as shown in Figure 21a. These inputs will generate HS directly through the interpolation filters (*Filter*). They also generate FOVS, but first these integer samples need to be transpose by the *Integer Pel Transpose Buffer (TB)*. The SOVS samples are generated from the HS, which must be previously interpolated and later stored and transposed at the *Horizontal Pel TB*.

The interpolated fractional samples are clipped by the *Clip* module before the block matching, so that their values are within the range between 0 and 255 to maintain the same 8 bits per pixel of the input samples. All values above 255 are clipped to 255, and all negative values are clipped to 0. The interpolation issues 27 interpolated samples per clock cycle, which is all the samples in fractional positions of the interpolated row or column. A register barrier was implemented at the *Clip*'s inputs to possibly shorten the critical path, besides preventing unnecessary switching activity while the *Filter*'s output (*Clip*'s input) is not stable.

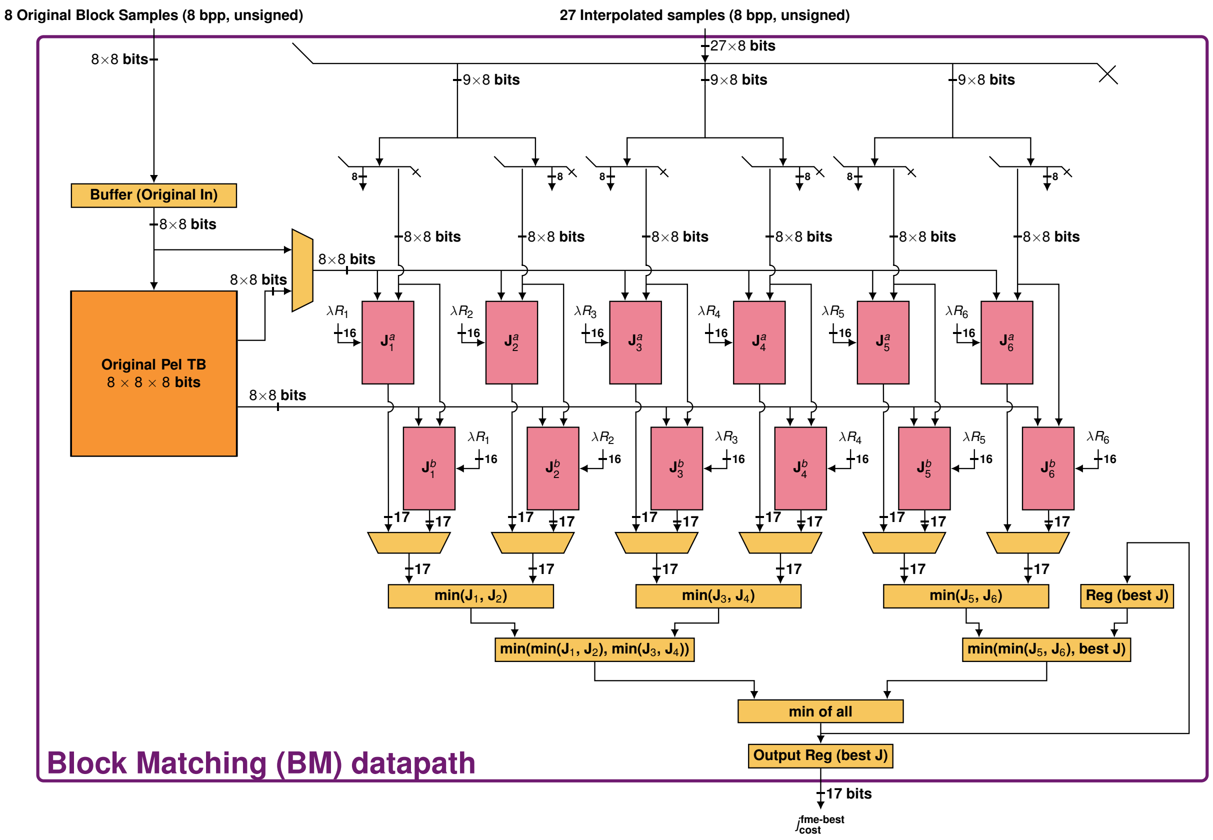
The *Filter*, which is the module responsible for the interpolation of the samples, is divided into two sub-modules, as shown in Figure 22. The *Sums and Shifts (SS)* (Figure 22a) generates all the products between the inputs and the coefficients (Table 2). It is implemented using an MCM (VORONENKO, 2017), an optimizer that explores common sub-expressions among source samples to generate every input coefficient multiplication while avoiding redundant filter operations by exploiting coefficient sharing. Equation 5.1 describes this vector function of the multiplication of input samples x by the eight distinct

Figure 21 – (a) Interpolation datapath for FME from Seidel et al. (2021), which is also responsible for transposing samples when needed, avoiding extra memory accesses. (b) Block matching datapath from Seidel et al. (2021).

(a)



(b)



Source: Seidel et al. (2021).

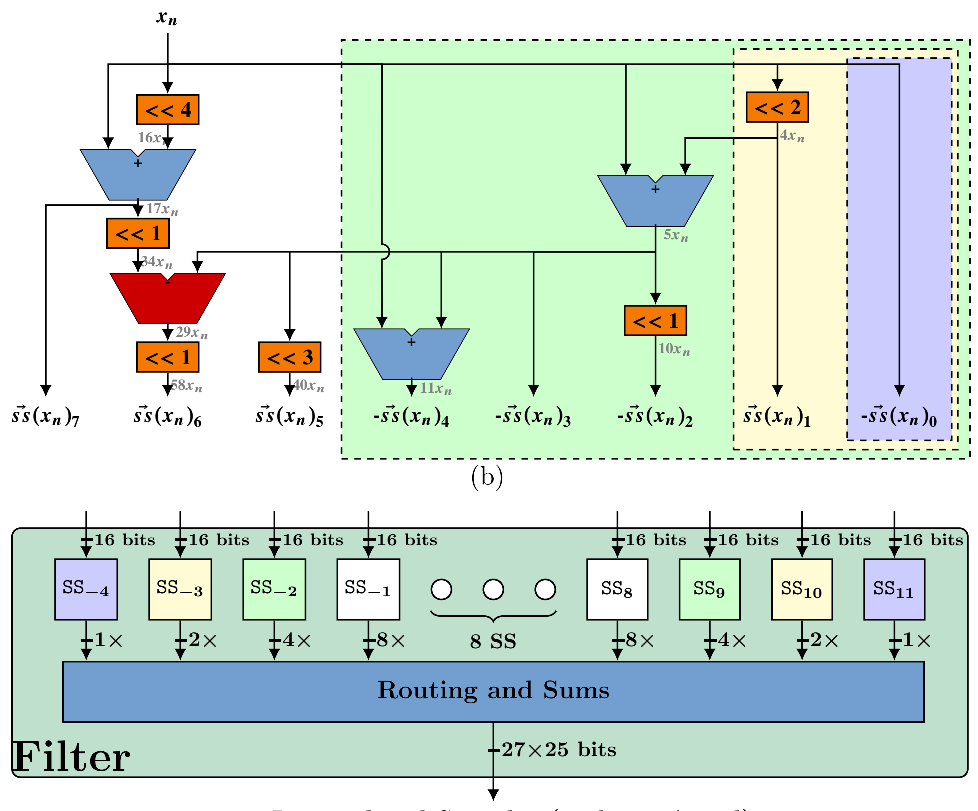
coefficients of the HEVC FME filters.

$$\overrightarrow{ss}(x) = [x; 4\times x; 10\times x; 5\times x; 11\times x; 40\times x; 58\times x; 17\times x] \tag{5.1}$$

Each input sample has its own SS module, as illustrated in Figure 22b. Each color of the SS_{index} inside the *Filter* represents the datapath of Figure 22a it contains: e.g., SS_{10} contains 2 outputs, one being $\overrightarrow{ss}(x_n)_0$ and the other is $\overrightarrow{ss}(x_n)_1$. The *Routing and Sums* is a sum tree that sums the products of the *SS* following the filters equations of each fractional position (Equation 3.1 to Equation 3.4), thus generating the interpolated samples.

Figure 22 – (a) Sums and Shifts (SS) and (b) Filter datapath for HEVC FME proposed by Seidel et al. (2021). The dotted colored boxes in (a) represent the SS modules with the same color in (b).

(a)



Source: Filho et al. (2020, apud Seidel et al. (2021)).

The block matching datapath shown in Figure 21a takes the 27 interpolated samples as inputs per cycle along with the $\mathbf{B}^{\text{ori}}$. The $\mathbf{B}^{\text{ori}}$ is provided for the architecture when the block matching datapath is computing the HS and then, stores and transposes the $\mathbf{B}^{\text{ori}}$ at the *Original Pel TB*. It is necessary to transpose this samples due to the fact that FOVS and SOVS are interpolated in columns, whereas HS, on the opposite, are interpolated HS in rows. Therefore, it is more efficient to transpose only the $\mathbf{B}^{\text{ori}}$, which is a 8×8 block, than all FOVS and SOVS.

The architecture of Seidel et al. (2021) was based on the architecture proposed by Filho et al. (2020). One of the main novelties of Filho et al. (2020) architecture lies in the computation of the rate term of Equation 2.13. Computing such a term internally improves coding efficiency by ensuring that the optimal RD candidate is selected. In addition, it also reduces the required energy by avoiding external computations. Figure 23a shows the design of Filho et al. (2020) for computing the $\lambda \times rate$ of six fractional MVs in parallel. Each rate estimate is obtained as defined in Equation 2.14 and is used to select a pre-computed value of the multiplication in $\lambda \times rate$. This approach avoids the use of multiplications and seeks to reduce the switching activity because while the set of MVs changes eight times during each FME execution, λ changes only once per frame. Moreover, as all the 32 possible rate estimates are known beforehand, such multiplications were implemented using MCM. Figure 23b shows their design for Equation 2.15, which is used to obtain each rate estimate. The output of the module in Figure 23b is selected via a series of multiplexers. The J_s modules are responsible for calculating the Langragian RD. Six J_s modules are required to match the throughput of HS and FOVS and prevent the architecture from data stalling. However, during the RD calculation of SOVS, samples from twelve different candidates will be computed per clock cycle, hence the necessity of twelve J_s modules. The outputs of the architecture are the lowest j_{cost} and the fractional position of the candidate that generate this cost.

5.1.1 Finite State Machine

The architecture proposed by Seidel et al. (2021) has two Finite State Machine (FSM)s. One to control the interpolation module and another for the block matching datapath. Figure 24 presents clock cycle diagram for the *Cross*-based version illustrating the behavior of both FSMs across a computation of an 8×8 block[1].

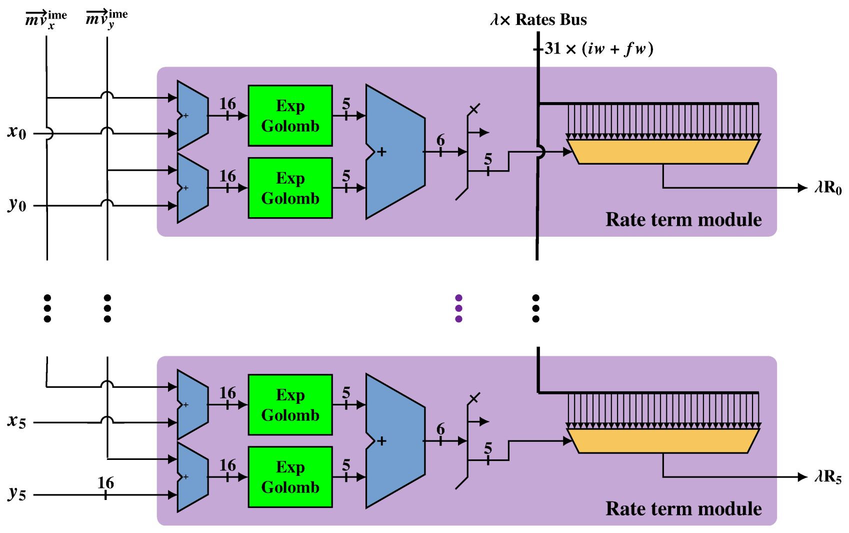
During the first 16 cycles the architecture interpolates the HS around the search area i starts. The interpolation FSM sets the enable on the *Horizontal Pel TB* and *Integer Pel TB* to store the interpolated samples, and the integer samples for the interpolation of the both types of vertical samples. At cycle 5 the interpolation FSM sets an enable control signal to the block matching FSM. This occurs when the first HS are available at the interpolation datapath output.

The block matching FSM stays idle for the first 4 cycles after the enable signal, the first 4 rows of HS available are outside the borders of the integer 8×8 block, only being computed for the interpolation of SOVS. At clock cycle 9, the first of the eight rows of HS inside the integer block are available to the block matching. The last 4 rows interpolated are also outside the borders of the integer block and so, are not computed by the block matching.

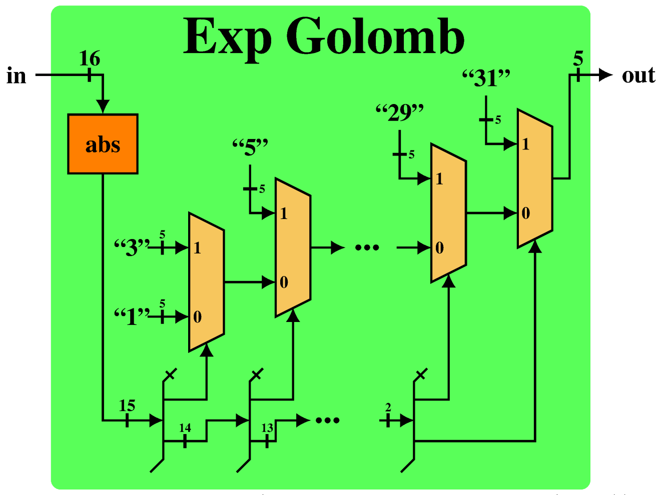
[1] With the exception of the first execution, which takes more clock cycles to fill all the register barriers.

Figure 23 – (a) Rate term ($\lambda \times r$) and (b) Exp Golomb datapath.

(a)



(b)

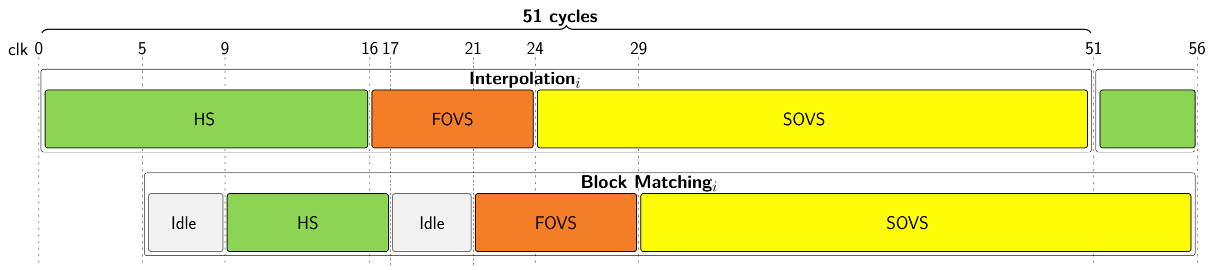


Source: Filho et al. (2020, apud Seidel et al. (2021)).

At cycle 16, the interpolation of HS ends. Then, the FSM starts feeding the *Filter* with integer samples stored in the *Integer Pel TB* for the interpolation of FOVS, taking 8 cycles to finish it, the results becoming available at the block matching 5 cycles later. Finally, the interpolation FSM takes another 27 cycles to interpolate all 27 columns of SOVS from the samples stored in the *Horizontal Pel TB*. After 51 clock cycles, all interpolation needed is finished. The block matching FSM finishes the execution of the distortion metric around a search area i five cycles later.

The architecture proposed by Seidel et al. (2021), as the other FME architectures found in the literature that compute all 48 fractional candidates for the HEVC FME, takes 51 clock cycles to compute each 8×8 integer block (FILHO et al., 2020; PASTUSZAK; TROCHIMIUK, 2016; AFONSO et al., 2016; KALALI; HAMZAOGLU, 2014), as they all need to process 51 different row/columns of fractional samples.

Figure 24 – Clock cycles diagram of the Seidel et al. (2021) architecture to process an 8x8 block.



Source: the author.

5.2 FIXED SEARCH PATTERN HARDWARE ESTIMATES

This section presents the area and power reduction estimates for the four fixed search patterns, the three proposed ones plus *Ding (a)*. The estimates are based on area and power synthesis reports of the baseline hardware architecture proposed by Seidel et al. (2021).

5.2.1 Area Reduction Estimate

Table 8 presents the area-share percentage of the main modules of the baseline architecture (SEIDEL et al., 2021) using SAD as distortion metric, as well as area reduction estimates for the four fixed patterns analyzed. The baseline architecture was synthesized using the Synopsys® Design Compiler (DC®) (SYNOPSYS, 2019a). All syntheses used a 45nm standard cell library from TSMC (TSMC, 2011). The area results of Seidel et al. (2021) are for a target throughput of 1080p@30fps (period of 20ns).

Table 8 – Area share (%) of the Seidel et al. (2021) FME architecture and estimates for the four fixed patterns.

Module	Seidel et al. (2021)	*Ding (a)*	**Box**	**Cross**	**Diamond**
Horizontal Pel TB	41.7	27.8	27.8	**0.0**	27.8
Integer Pel TB	6.5	6.5	6.5	6.5	6.5
Filter	22.0	18.0	15.0	22.0	22.0
Original Pel TB	3.8	3.8	3.8	3.8	3.8
J Tree	15.0	7.5	5.0	7.5	7.5
Rate term	6.1	4.1	2.0	6.1	6.1
Others	4.9	4.9	4.9	4.9	4.9
Total	100	72.6	65.0	**50.8**	77.0

For *Box*, *Diamond* and *Ding (a)*, one third of the *Horizontal Pel TB* can be removed since no $\mathbf{B}^{\text{can}}$ computed by these patterns is generated from b HS (Figure 12), excepting a and c. *Cross*, in turn, does not require this buffer at all, as $\mathbf{B}^{\text{can}}$ composed by SOVS are not included in its search window. For the J tree, *Ding (a)*, *Cross* and *Diamond* need only six of the original twelve J modules, since a maximum samples from six different candidates will be available for the block matching, *Box* will have a maximum of four. Since *Box* does not compute half-pel sample, it does not need the middle filter

(Equation 3.2), but some of the ALU that generate the coefficients of middle filter must be used to generate for up and down. In these situations, a more detailed analysis on the synthesis reports would be required to find a more accurate estimate. In summary, *Diamond* has the lowest estimated area reduction with only 23.0% of the total area of the baseline architecture, *Ding (a)* is estimated to reduce 27.4%, and *Box* is estimated to be 35.0%. Hence, in terms of area, *Cross* is the best pattern since its estimated area reduction is 49.2%, the largest amongst the four patterns, most of this reduction coming from the removal of the *Horizontal Pel TB*, 41.7%.

5.2.2 Power Reduction Estimate

Apart from the area advantage, the minimization of hardware may likewise bring positive impacts on power dissipation. From the area reduction estimate, the analyzed patterns will have considerably lower amount of transistors, which may indicate lower switching activity. However, it is inaccurate to estimate a possible power reduction from the area reduction, considering there is no data on switching activity, and also the utilization of synthesis tools such as clock gating can mitigate the power dissipation of inactive transistors. Nonetheless, it is possible to calculate the required operation frequency for a target throughput, as shown in Equation 5.2, where N is the number of cycles per block, W and L are the width and length of the sequence, B_W and B_L are the block width and length.

$$F = \frac{N \times W \times L \times fps}{B_W \times B_L} \tag{5.2}$$

From the frequency it is possible to calculate the switching power dissipated by a chip using static CMOS gates, as defined in Equation 5.3, where C is the capacitance being switched per clock cycle, V is the supply voltage, and F is the frequency of operation.

$$P = C \times V^2 \times F \tag{5.3}$$

Table 9 shows the required operating frequencies for the baseline architecture (SEIDEL et al., 2021) and the studied fixed patterns to process 4K (3840×2160) videos at 60 fps.

Table 9 – Frequency required by each pattern for a target throughput.

Pattern	Seidel et al. (2021)	*Ding (a)*	**Box**	**Cross**	**Diamond**
# of Candidate blocks	48	12	8	12	16
Cycles per 8×8 block	51	42	42	**24**	42
Freq.@4K 60 fps (MHz)	396	326	326	**186**	326

A 8×8 block size 48 $\mathbf{B}^{\text{can}}$s architecture takes 51 clock cycles to compute all $\mathbf{B}^{\text{can}}$s, from which 16 clock cycles are required for the HS, 8 for the FOVS and 27 for the SOVS. As detailed in Table 9, with the modifications proposed by *Ding (a)* an 8×8 block size architecture takes 42 clock cycles (17.6% less cycles than the architecture of Seidel et al. (2021)) per block to compute 12 fractional $\mathbf{B}^{\text{can}}$s. *Box* and *Diamond* take the same 42 clock cycles, but compute 8 and 16 $\mathbf{B}^{\text{can}}$s, respectively. *Cross*, however, takes just 24 clock cycles (53.0% less cycles) to compute 12 fractional $\mathbf{B}^{\text{can}}$s. It is worth remembering that VTM searches 8 or 16 $\mathbf{B}^{\text{can}}$s, depending on the MV precision.

The number of cycles to compute each block is directly proportional to the frequency of operation, as shown by Equation 5.2, and the frequency of operation is directly proportional to the switching power dissipation, as defined in Equation 5.3. Therefore, *Box*, *Diamond* and *Ding (a)* are estimated to reduce the switching power by only 17.6%, while *Cross* may reach 53% when compared to the baseline architecture. It is worth pointing out that these are very conservative estimates, once they do not take into consideration the area reduction resulting from not using a number of modules that are required by the baseline architecture. Besides the reduction in power dissipation, a lower frequency of operation for the same throughput also represents a higher maximum throughput for the same technology.

From these estimates, *Cross* is clearly the most promising pattern to drastically reduce the occupied area and power dissipation, which is trade off by a minor BD-Rate increase when compared to *Ding (a)* and *Diamond* [2]. Therefore, *Cross* is the chosen fixed pattern to be implemented in dedicated hardware in this work.

5.3 CROSS ARCHITECTURE

This section presents the datapath and FSM of the proposed architecture for this work.

5.3.1 Datapath

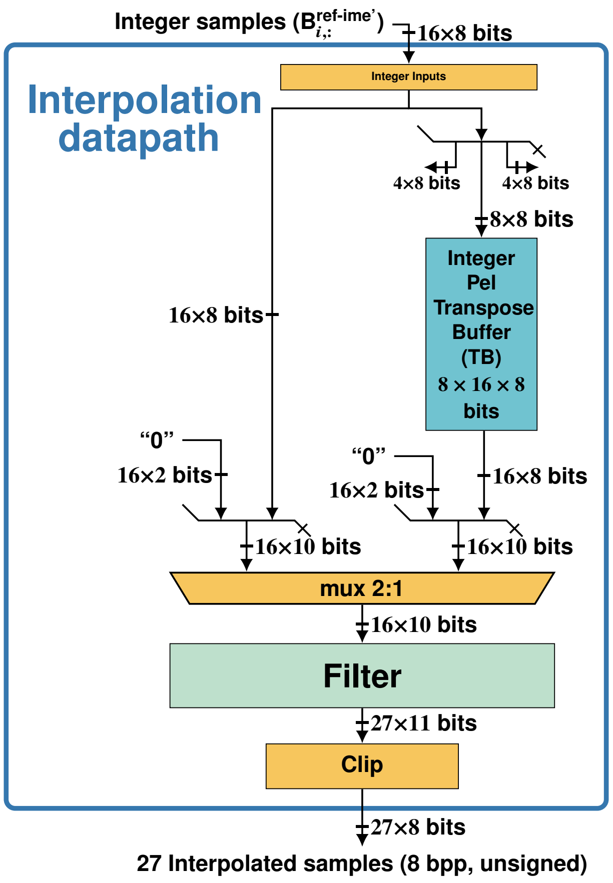
As explained in Section 3.3, Seidel et al. (2021) was selected as baseline for this work because it was developed in the same research group, what granted full access to its Register Transfer Level (RTL) descriptions and testbench, make it possible to establish a fair and accurate comparison between baseline architecture and proposed one, since their designs share the same base modules.

Figure 25 shows the interpolation datapath of the proposed *Cross*-based hardware architecture for the VVC FME hardware architecture. The same 16 samples are required at the input, following the same path of register barriers at the beginning to the now *mux 2:1* and *Integer Pel TB*. The *Cross* pattern requires all FOVS to be computed and therefore , thisTB keeps the same configuration of the baseline. As explained in Subsection 5.2.1, a *Cross* architecture does not require a TB to store and transpose the

[2] The trade-off between coding efficiency and computation complexity is very subjective, always depending on the type of application and its specifications.

HS, since no SOVS will be computed by this FME design. Consequently, the *Horizontal Pel TB* is completely removed. Also the size of the multiplexer that controls the samples to be interpolated is reduced from 3 16-samples inputs to just 2.

Figure 25 – Interpolation datapath of the proposed *Cross*-based VVC FME hardware architecture.



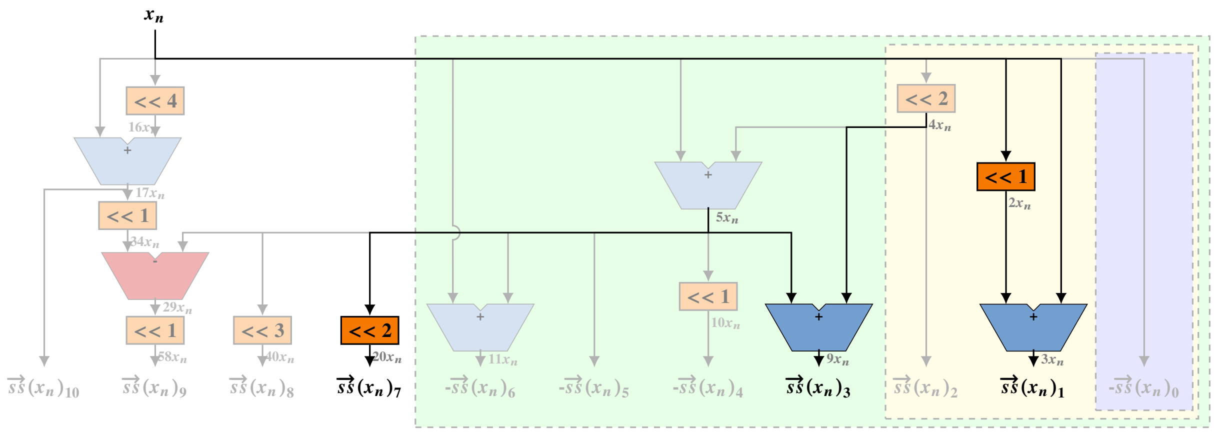
Source: the author.

The baseline architecture was designed for the HEVC FME. Therefore, modifications to the filters were required so as to implement the newly introduced three alternative half-pel coefficients (Table 2) for the VVC FME. To keep the same coefficient sharing, the vector function Equation 5.1 was modified to also generate these new coefficients, as described in Equation 5.4.

$$\overrightarrow{ss}(x) = [x; 3{\times}x; 4{\times}x; 9{\times}x; 10{\times}x; 5{\times}x; 11{\times}x; 20{\times}x; 40{\times}x; 58{\times}x; 17{\times}x] \tag{5.4}$$

Figure 26 presents the new SS modules with support for the VVC alternative half-pel precision FME filters. Two arithmetic logic units had to be added to generate the coefficients $3x_n$ and $9x_n$, increasing the maximum number of arithmetic logic units per SS from four to six. For the $20x_n$, only bit shifting was necessary since a $5x_n$ was already generated for the other filters. The *Routing and Sums* (Figure 22b) was also modified to include the required sums for the alternative filter.

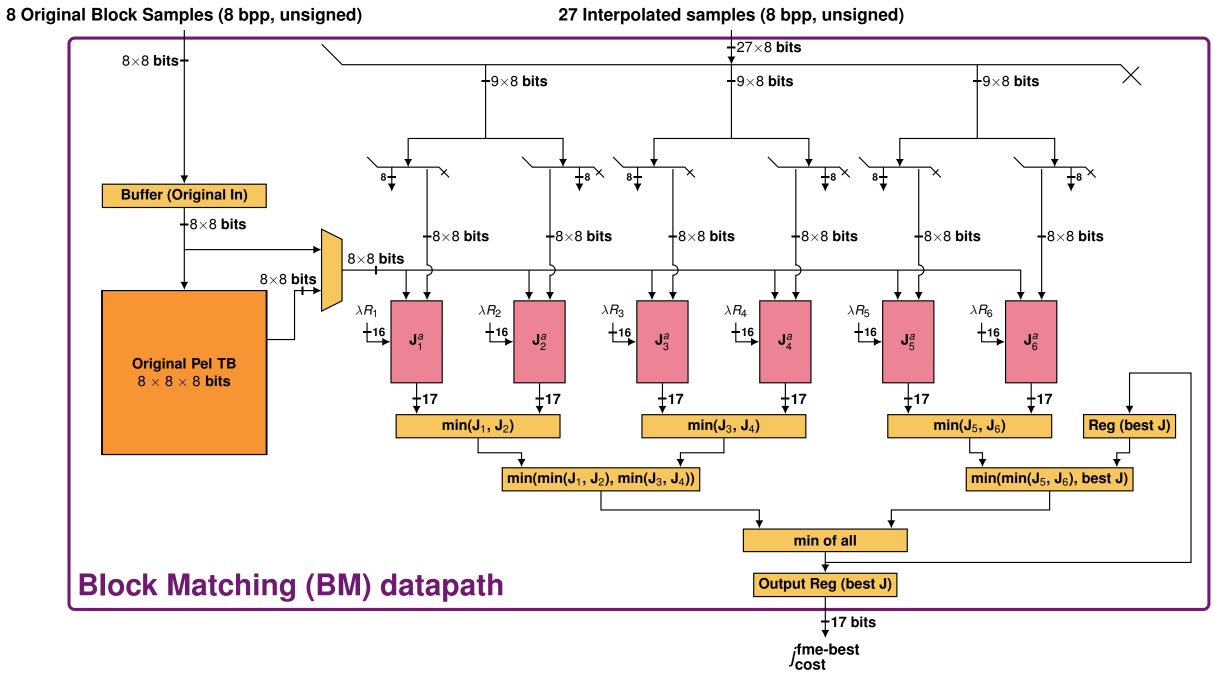
Figure 26 – Updated SS datapath to support the VVC FME alternative half-pel filter.



Source: the author.

In the block matching datapath, as illustrated in Figure 27, the main modification is the removal of six J^3 modules. Each J module performs the SAD (Equation 2.10), row-by-row, between $\mathbf{B}^{\text{ori}}$ and each interpolated $\mathbf{B}^{\text{can}}$, which is added to the appropriate $\lambda\times$rate to obtain j_{cost}, since a maximum of samples from six different candidates will be available at a given moment. Twelve J are required only during the block matching of SOVS. Other minor submodules for control and routing of the data of these modules were also removed, further reducing the hardware.

Figure 27 – Block matching datapath of the proposed *Cross*-based hardware architecture.


8 Original Block Samples (8 bpp, unsigned)
27 Interpolated samples (8 bpp, unsigned)
Buffer (Original In)
Original Pel TB 8 × 8 × 8 bits
min(J1, J2)
min(J3, J4)
min(J5, J6)
Reg (best J)
min(min(J1, J2), min(J3, J4))
min(min(J5, J6), best J)
min of all
Output Reg (best J)
Block Matching (BM) datapath
17 bits

Source: the author.

[3] These J modules use the SAD as distortion metric.

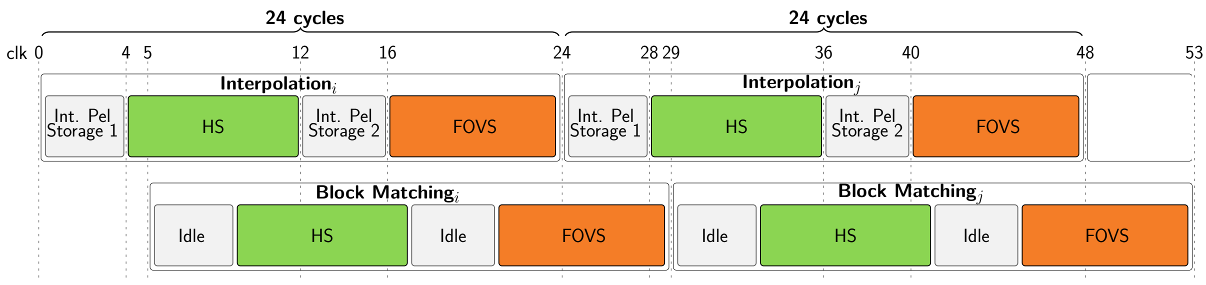
5.3.2 Finite State Machine

As presented in Section 5.1, the baseline architecture is controlled by 2 FSMs, one for the interpolation datapath and another for the block matching datapath. The *Cross*-based architecture adopts the same control unit organization.

Figure 28 presents the clock cycle diagram for the *Cross*-based FSMs. The execution of a fractional search around a search area i starts at the clock cycle 0. At the first 16 cycles, the architecture receives a row of integer samples per cycle around the search area i. The first and last four rows are outside the search window are only utilized to generate HS for the interpolation of SOVS, as explained in Section 5.1. Consequently, during these cycles, the interpolation FSM disables the *Filter* and just stores the integer samples at the *Integer Pel TB* for later interpolation of FOVS (Int. Pel Storage 1 and 2). The HS interpolation starts at cycle 4, ending 8 cycles later. The FOVS starts at the cycle 16 and ends at cycle 24, same as the baseline architecture. Since no SOVS is computed, the interpolation FSM can start the interpolation of the fractional samples of a next search area j at cycle 24.

The block matching FSM is pretty similar to the baseline FSM, the initial states work the same, only removing the states to compute the distortion rate for the SOVS. Therefore, the block matching finishes the computation of all $\mathbf{B}^{\text{can}}$s in cycle 29. At the end of its execution, the fractional coordinates of the $\mathbf{B}^{\text{ref}}$ (x_{frac},y_{frac}) and the j_{cost} are issued, and the FSM sets the control signal to start the computation of the next $\mathbf{B}^{\text{can}}$s.

Figure 28 – Clock cycle diagram of the *Cross*-based VVC FME hardware architecture to process an 8x8 block.



Source: the author.

5.4 HARDWARE EVALUATION METHOD

The baseline architecture was validated using the framework proposed by Bonotto et al. (2018). This framework allows for performing functional verification verification using Synopsys® Verilog Compiler Simulator (VCS®) (SYNOPSYS, 2012) with real stimuli from a video sequence, matching the results with HM. To validate the *Cross*-based FME hardware architecture, a reference software that executed a FBMA and a *Cross* search for the FME was developed in C++. Its correctness was verified with real stimuli from the first 100 8×8 FME executions using the default two step search, the first 100 executions of the alternative search for RaceHorsesC sequence on VTM v13.0. To perform a functional verification, the *Cross*-based FME hardware architecture was then tested with the developed reference software using Icarus Verilog (WILLIAMS, 2017). For such, random generated stimuli of over 1,000 8×8 original block and related candidates were used as inputs of the testbench.

The architecture was synthesized with DC® (SYNOPSYS, 2019b) in Topographical mode to estimate routing parasitics and thus, obtain realistic timing, area and power estimates, using a 45nm TSMC standard cell library (TSMC, 2011) and the tool's default switching activity. The input and output delays were limited to 60% of the clock period (SEIDEL, 2014). Also, the maximum primary input capacitance was set to 10× a 2-input AND gate whereas the maximum primary output capacitance was set to 30× a 2-input AND gate. Finally, to get detailed area and power reports for each module separately, DC® was restricted to keep the RTL hierarchy during syntheses.

Seidel et al. (2021) synthesized their architecture for 4 distinct periods: 20ns, 10ns, 5ns, and 2.5ns, which were chosen to meet the throughputs required for 4 distinct video configurations: 1080p@30fps, 1080p@60fps, 2160p@30fps and 2160p@60fps. They tried to increase the frequency further with a target period of 1.25ns (2160@120fps or 4320@30fps), but the tool was unable to meet such a timing constraint of the synthesis tool. Accordingly, the *Cross*-based architecture was synthesized with 4 different periods: 42.8ns, 21.4ns, 10.7ns, 5.3ns, to match the same target throughput of the baseline architecture. It was also synthesized for 2 extra periods, 2.6ns (4320p@30fps) and 1.3ns (4320p@60fps). For the 1.3ns the tool was unable to meet the timing constraint. Moreover, Seidel et al. (2021) synthesized all architectures with clock-gating, which is a common and efficient technique that disables the clock activity of components that are not operating to reduce dynamic power dissipation. To insert clock-gating they relied on the automatic clock-gating insertion option from the synthesis tool, which is the simplest strategy. For comparison purposes, the *Cross*-based VVC FME architecture was also synthesized using the clock-gating technique.

It is worth reminding that the architecture proposed by Seidel et al. (2021) was design only for the HEVC, whereas the *Cross*-based architecture is design for VVC but is compatible with HEVC as well.

5.5 SYNTHESIS RESULTS

Table 10 presents the synthesis results for both FME architectures. For all target periods syntheses with results for both architectures, the *Cross*-based architecture occupies 41.4% to 42.4% the area of the baseline, a reduction of up to 58.6% of the total occupied area, with an average of 42.2%. The reduction is even higher than that estimated in Subsection 5.2.1, what is mainly due to a conservative estimate that did not considered some minor reductions, such as those from the *Clip* module. The baseline architecture was originally synthesized with Synopsys (2019a). To check for a possible discrepancy between versions of the EDA tool, the baseline architecture was synthetized with the same version as the *Cross*-based (SYNOPSYS, 2019b), and using the same configurations. The result showed an increase in occupied area of just 0,89%.

Table 10 – Synthesis results for the Baseline (SEIDEL et al., 2021) and *Cross*-based FME architectures.

Architecture	**Baseline**	**Cross**	**Baseline**	**Cross**	**Baseline**	**Cross**	**Baseline**	**Cross**	**Baseline**	**Cross**
Target throughput	1080p@30fps		1080p@60fps		2160p@30fps		2160p@60fps		4320p@30fps	
Period (ns)	20.0	42.8	10.0	21.4	5.0	10.7	2.5	5.3	1.25	2.6
Area (μm^2)	86302.28	36646.92	86364.03	36660.33	88867.67	36815.38	96400.30	40840.83	n.a.	42969.45
Dynamic power (μW)	1392.70	312.05	2821.90	600.37	5500.00	1202.90	11767.10	2840.82	n.a.	5862.80
Static power (μW)	1344.80	614.63	1345.40	614.52	1481.00	629.76	1814.80	731.36	n.a.	802.20
Total power (μW)	2737.50	926.68	4167.30	1214.85	6981.00	1832.69	13581.90	3572.18	n.a.	6665.00

The dynamic power reduction of *Cross*-based over the baseline architecture for the different target periods was 78.0% on average, a considerable disparity over the 53.0% presented in Subsection 5.2.2. In the estimation, the dynamic power reduction only accounted the 53.0% lower frequency of operation, which is directly proportional to the dynamic power (Equation 5.3)[4]. But in reality, a lower number of nets and ports will reduce the dynamic power dissipation. The static power and occupied area stay reasonably constant for across the lower throughputs, but for higher throughputs they increase significantly, e.g., 16.1% increase in static power dissipation from 2160p@30fps to 2160p@60fps for the *Cross*-based architecture, which hints that the synthesis tool is using larger gates to meet the tighter timing constraints.

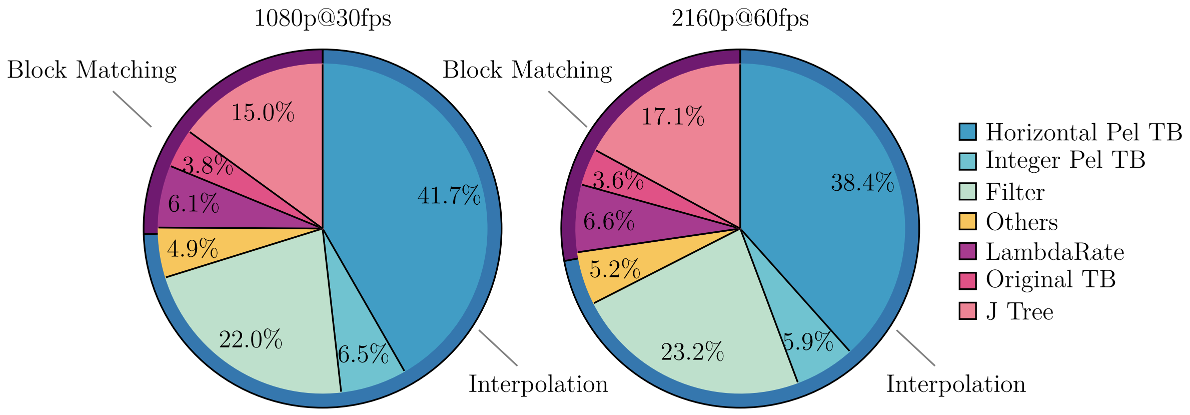
The total power of the *Cross*-based architecture is 28.9% of the baseline architecture for the same target period, on average, a 71.1% reduction. As the static power reduction is not quite significant as the dynamic power, the total power reduction does not keep up with the dynamic power reduction. Due to its lower frequency of operation, the *Cross*-based architecture, is capable of working for a higher throughput. Whereas the baseline architecture failed to meet the timing constraint for 4320@30fps, *Cross* is capable of such. After all, the target period for such, 2.6ns, is nearly the same as the target period for the baseline to run at 2160@60fps, 2.5ns. To run at such a higher frequency, the area increases in 5.2% and total power nearly doubles it for the baseline.

Figure 29 and Figure 30 present the area and power breakdowns, respectively, for the lowest and highest throughput of the baseline architecture. As explored in Section 5.1, for the baseline architecture, the *Horizontal Pel TB* is the biggest module (Figure 29a) with 41.7% and 38.4%, for the smaller and higher throughput, respectively. It is also the biggest in terms of power dissipation, over 40% of the total power total is dissipated by this module (Figure 30a). This is one of the main reasons for proposing the *Cross* search pattern, which does not require such an area and power hungry module.

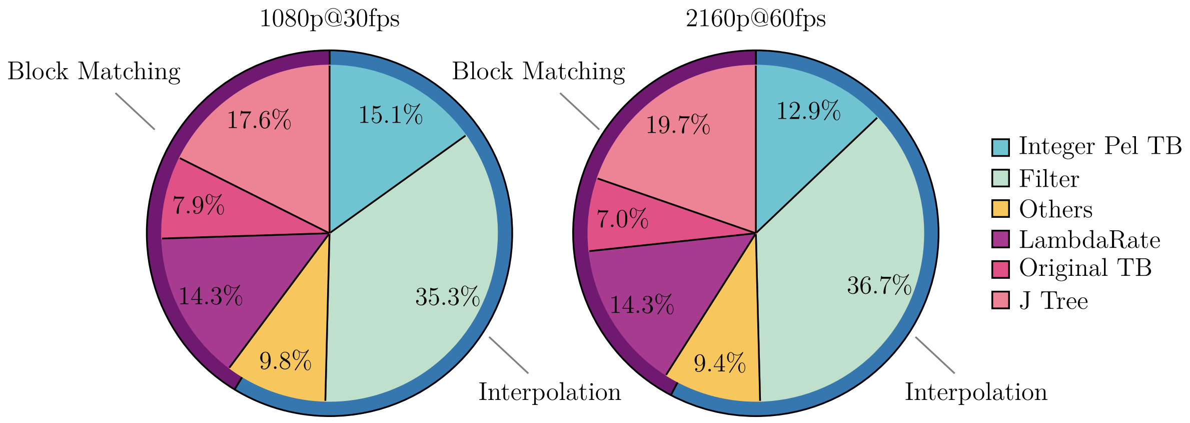
[4] This relation can be observed in Table 10: for both architectures the power dissipation nearly doubles when the period is halved.

Figure 29 – Area breakdown of the baseline and the proposed *Cross*-based FME architectures. The outer ring represents the share of Interpolation and Block matching.

(a) Baseline.



(b) Cross.

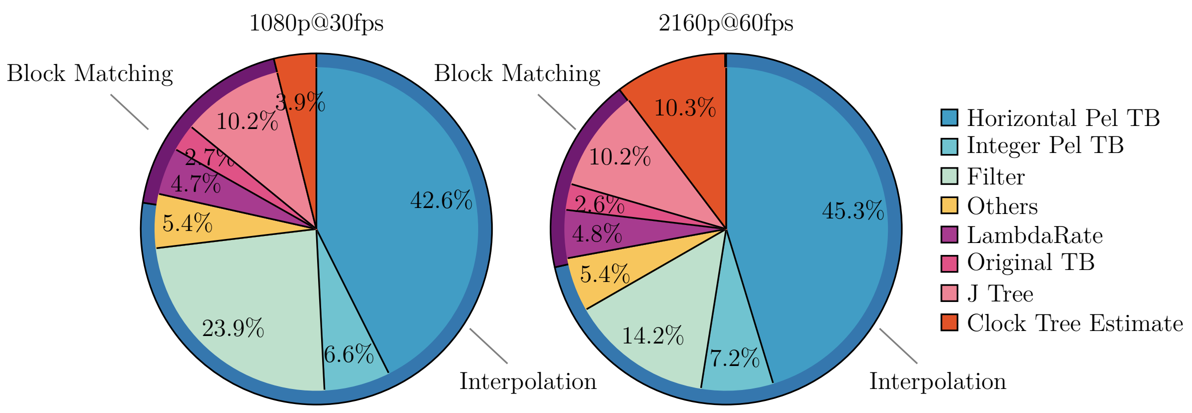


As a result of the modifications to the baseline architecture to built the *Cross*-based architecture, the biggest module is now the *Filter*, 35.3% and 36.7% of the total area for the 1080p@30fps and 2160p@60fps targets, respectively (Figure 29b). The implementation of the VVC alternative half-pel filter at the *Filter* occupies 5.4% and 6.3% of the total area, respectively, which is roughly 17% of the *Filter* area. The *Filter* is also the most power demanding module, dissipating from 44.4% to 36.3% of the total power of the architecture (Figure 30b). The submodules that implement the alternative half-pel filter in *Filter* are responsible for 8.1% to 6.1% of the total power, for the lower and higher throughput, respectively, approximately 18% of the total power dissipated by *Filter*.

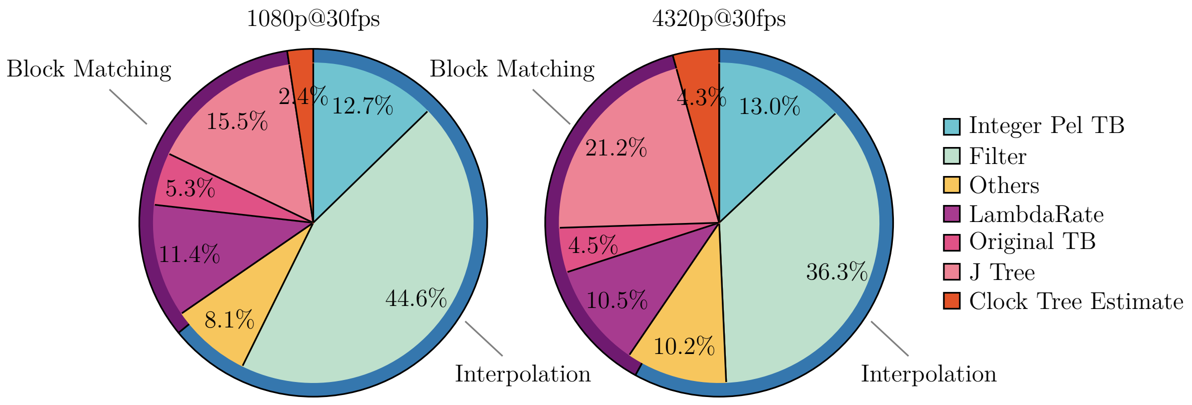
From Figure 29 and Figure 30, one may also observe the overall percentage of the *J Tree* that is the only module that swells when increasing the operation frequency for both architectures, suggesting that the tool is resizing the gates of this module to meet the more demanding timing constraints. Taking a closer look at the synthesis of the *Cross* for the target period of 1.3ns, which did not meet the tool's timing constraints, the *J Tree* occupies 23.8% of the total area. Moreover, when setting the tool to synthesize without maintaining the RTL hierarchy, the only path that could not meet the time constraints

Figure 30 – Power breakdown of the baseline and the proposed *Cross*-based FME architectures. The outer ring represents the share of Interpolation and Block matching.

(a) Baseline.



(b) Cross.



was inside the block matching passing through the *J Tree*. With a clock of 1.30ns, the tool sets a clock uncertainty requiring the data to arrive in 1.25ns, while the arrival time is 1.28ns, therefore violating the timing constraints by 0.03ns.

6 CONCLUSIONS

Finalized in July of 2020, the VVC standard brings a number of new tools to substantially improve coding efficiency, which resulted in a significant increase in complexity. Therefore, the use of such a new standard in mobile devices requires not only the use of dedicated VLSI architectures, but also the adoption of techniques that could reduce its complexity to meet the real-time and energy efficiency requirements. In this context, the most intensive encoding tools, such as the FME, are the first ones to be considered for optimization.

This work brings as first contribution, a quality assessment of the coding efficiency impact of the FME in VVC, considering the reference software VTM and the CTC guidelines. The conducted experiments shown that disabling the FME in VVC leads to an average BD-Rate of 0.64% for class B video sequences (which are FHD sequences) and the RA configuration, whereas Blasi et al. (2015) showed that disabling the FME in HEVC results in an average BD-Rate of 6.27% for FHD sequences. Such lower impact of the FME in VVC in comparison to HEVC might be due to the addition of new inter-prediction tools and a simplified 1/2-MV precision mode for the FME in the most recent standard.

Based on the analysis of the gathered data that shows the behavior of the FME illustrated with heatmaps, this work proposes three fixed (simplified) fixed search patterns for the VVC FME. They were design to reduce the complexity of computing all 48 $\mathbf{B}^{\mathrm{can}}$s that most works on FME hardware architectures apply to the BMA, therefore following the same train of thought of the VVC developers of reducing the FME complexity, but applying this idea to hardware.

The proposed fixed search patterns and one from a related work, named *Ding (a)*, were implemented in VTM to measure their coding efficiency compared to the default FME. The results showed *Diamond* and *Ding (a)* having the best coding efficiency, with an average BD-Rate of 0.18% and 0.19% for LD-P and 0.15% and 0.16 % for RA configurations, respectively. *Cross* presented slightly worse results, with 0.34% and 0.28% average BD-Rates, for LD-P and RA, respectively. *Box* had the worst coding efficiency among the tested fixed search patterns, with BD-Rate increases of 0.57% and 0.44% for LD-P and RA, respectively.

Based on detailed synthesis results of the baseline architecture found in the literature, estimates on area and power reduction of implementing each pattern in a similar hardware design were drawn. Amongst the four evaluated patterns, *Cross* was estimated to have the biggest reduction in occupied area and dynamic power dissipation with respect to the baseline: 49.4% and 53%, respectively. Such reductions are the consequence of the lack of SOVS inside the search window, thus resulting in a more efficient computation of the FME for a VLSI architecture. Although presenting slightly worse coding efficiency results than *Diamond*, *Cross* was chosen for a dedicated hardware design because it was the most promising pattern in terms of hardware reduction.

The baseline architecture was originally synthetized by Seidel et al. (2021) in four distinct periods, 20ns, 10ns, 5ns, and 2.5ns, to meet the throughput requirements of four distinct video configurations: 1080p@30fps, 1080p@60fps, 2160p@30fps and 2160p@60fps, respectively. The synthesis periods for *Cross* were chosen to match those four video configurations, plus 4320@30fps, for which the baseline was unable to meet the timing constraints in the synthesis flow. Results showed that the occupied area of the presented *Cross*-based architecture is 42.4% (57.8% of reduction) when compared to baseline architecture, on average. The total power of the *Cross*-based architecture is, on average, 28.9% (71.1% reduction) of that exhibited by the baseline architecture for the same target period. These results confirm the initial premise that the proposed fixed search patterns can considerably promote hardware minimization when designing a VLSI architecture tailored for the VVC FME, without major losses in coding efficiency. The estimates on potential reductions resulted from search patterns that partially compute SOVS, i.e., *Box*, *Diamond* and *Ding (a)* did not look so promising. In conclusion, if the application requires maximum coding efficiency, disregarding the increase in complexity, then a 48 $\mathbf{B}^{\mathrm{can}}$s FME architecture is the most appropriate. However, if the application is aimed at low-power high throughput, then a *Cross*-based VLSI is the most suitable.

6.1 FUTURE WORK

A possible future work relies on obtaining realist power estimates using switching activity from real stimuli from a video sequence for both baseline and *Cross*-based architectures, so as to provide more realist comparisons and analyses over the impact of adopting a *Cross*-based design.

As presented in Section 5.5, the synthesis for the target period of 1.3ns violated the timing constraint by 0.03ns. Thus, another future work would be to reduce the critical paths to allow this architecture to meet the timing constraints of the 1.3ns synthesis. By doing so, such an architecture would be capable of processing in real time 8K (7680×4320) videos at 60fps.

REFERENCES

AFONSO, V. et al. Hardware implementation for the HEVC fractional motion estimation targeting real-time and low-energy. **Journal of Integrated Circuits and Systems**, v. 11, n. 2, p. 106–120, 2016.

AFONSO, V. et al. Memory-aware and high-throughput hardware design for the HEVC fractional motion estimation. In: **Proceedings of the 28th Symposium on Integrated Circuits and Systems Design (SBCCI)**. [S.l.: s.n.], 2015. p. 1–6.

AZGIN, H.; KALALI, E.; HAMZAOGLU, I. An approximate versatile video coding fractional interpolation hardware. In: **2020 IEEE International Conference on Consumer Electronics (ICCE)**. [S.l.: s.n.], 2020. p. 1–4.

BADRY, E.; SHALABY, A.; SAYED, M. S. A hardware friendly fractional-pixel motion estimation algorithm based on adaptive weighted model. In: **Proceedings of the International Conference on Microelectronics, ICM**. [s.n.], 2018. v. 2017-December, p. 1–4. Cited By :1. Disponível em: www.scopus.com.

BJØNTEGAARD, G. **Calculation of average PSNR differences between RD-curves**. Austin, Texas, USA, 2001.

BLASI, S. et al. Adaptive precision motion estimation for HEVC coding. In: **Picture Coding Symposium (PCS), 2015**. [S.l.: s.n.], 2015. p. 144–148.

BONOTTO, B. et al. A named-pipe library for hardware simulation. In: **33^{o} Simpósio Sul de Microeletrônica (SIM)**. [S.l.]: SBC, 2018.

BOSSEN, F. **Common test conditions and software reference configurations**. Shanghai, 2012.

BOSSEN, F. et al. **JVET common test conditions and software reference configurations for SDR video**. Geneva, 2019.

BOSSEN, F.; LI, X.; SUEHRING, K. **VVC Test Model (VTM) v13.0**. 2020. https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_VTM/-/tags/VTM-13.0.

BOSSEN, F.; LI, X.; SUEHRING, K. **VVC Test Model (VTM) v6.2**. 2020. https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_VTM/-/tags/VTM-6.2.

BOSSEN, F. et al. Vvc complexity and software implementation analysis. **IEEE Transactions on Circuits and Systems for Video Technology**, v. 31, n. 10, p. 3765–3778, April 2021.

BRäSCHER, A. B.; SEIDEL, I.; GüNTZEL, J. L. Improving the energy efficiency of a low-area SATD hardware architecture using fine grain PDE. In: **Proceedings of the 2017 Symposium on Integrated Circuits and Systems Desing (SBCCI)**. [S.l.: s.n.], 2017. p. 155–161.

CANCELLIER, L. H. L. **Algoritmo de Eliminações Sucessivas baseado em Soma das Diferenças Transformadas Absolutas**. Monografia — UFSC, 2016.

CHAKRABARTI, I. et al. **Motion Estimation for Video Coding: Efficient Algorithms and Architectures**. [S.l.]: Springer International Publishing, 2015. (Studies in Computational Intelligence).

CHEN, J.; YE, Y.; KIM, S. H. **Meeting Report: Algorithm description for Versatile Video Coding and Test Model 8 (VTM 8)**. Brussels, BE, 2020.

CISCO. **VNI Complete Forecast Highlights: Global - 2022 Forecast Highlights**. [S.l.], 2018.

DING, D.; YE, X.; WANG, S. 1/2 and 1/4 pixel paralleled fme with a scalable search pattern for hevc ultra-hd encoding. In: **2015 IEEE 16th International Conference on Communication Technology (ICCT)**. [S.l.: s.n.], 2015. p. 278–281.

DINIZ, C. M. et al. A reconfigurable hardware architecture for fractional pixel interpolation in high efficiency video coding. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 34, n. 2, p. 238–251, 2015.

FILHO, V. R. et al. Standalone rate-distortion FME architecture. In: **2020 33rd Symposium on Integrated Circuits and Systems Design (SBCCI)**. [S.l.: s.n.], 2020. p. 1–6.

HE, G. et al. High-throughput power-efficient vlsi architecture of fractional motion estimation for ultra-hd hevc video encoding. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, v. 23, n. 12, p. 3138–3142, 2015.

HUNT, R. **The Reproduction of Colour**. Wiley, 2005. (The Wiley-IS&T Series in Imaging Science and Technology). ISBN 9780470024263. Disponível em: https://books.google.com.br/books?id=nFtW4LG24fEC.

ITU-R. **Recommendation ITU-R BT.601-7, Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios.** 2011.

ITU-T. **Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios**. Genebra, 1995.

ITU-T. **Parameter values for the HDTV standards for production and international programme exchange**. Genebra, 2002.

ITU-T. **Recommendation ITU-T H.265: High efficiency video coding**. Genebra, 2013.

ITU-T. **Versatile Video Coding, Draft 10**. 2020. Disponível em: https://jvet.hhi.fraunhofer.de/.

KALALI, E.; HAMZAOGLU, I. A low energy hevc sub-pixel interpolation hardware. In: **2014 IEEE International Conference on Image Processing (ICIP)**. [S.l.: s.n.], 2014. p. 1218–1222.

KALALI, E.; HAMZAOGLU, I. Approximate hevc fractional interpolation filters and their hardware implementations. **IEEE Transactions on Consumer Electronics**, p. 1–1, 2018. ISSN 0098-3063.

KOEZE, E.; POPPER, N. **The Virus Changed The Way We Internet**. [S.l.], 2020. Disponível em: https://www.nytimes.com/interactive/2020/04/07/technology/coronavirus-internet-use.html. Acesso em: 2020-07.

LIM, D. . et al. A fast fractional motion estimation algorithm for high efficiency video coding. In: **International Conference on Electronics, Information, and Communications, ICEIC 2016**. [S.l.: s.n.], 2016.

LV, H. et al. A resolution-adaptive interpolation filter for video codec. In: **Proceedings - IEEE International Symposium on Circuits and Systems**. [S.l.: s.n.], 2014. p. 542–545.

MAHDAVI, H.; HAMZAOGLU, I. A vvc fractional interpolation hardware using memory based constant multiplication. In: **2021 IEEE International Conference on Consumer Electronics (ICCE)**. [S.l.: s.n.], 2021. p. 1–5.

ORTEGA, A.; RAMCHANDRAN, K. Rate-distortion methods for image and video compression. **IEEE Signal Processing Magazine**, v. 15, n. 6, p. 23–50, 1998.

PAKDAMAN, F. et al. Complexity analysis of next-generation vvc encoding and decoding. In: IEEE. **IEEE International Conference on Image Processing (ICIP)**. [S.l.], 2020.

PASTUSZAK, G.; TROCHIMIUK, M. Algorithm and architecture design of the motion estimation for the H.265/HEVC 4K-UHD encoder. **Journal of Real-Time Image Processing**, Springer Berlin Heidelberg, p. 1–13, 2015.

PASTUSZAK, G.; TROCHIMIUK, M. Algorithm and architecture design of the motion estimation for the H.265/HEVC 4K-UHD encoder. **Journal of Real-Time Image Processing**, v. 12, n. 2, p. 517–529, Aug 2016.

POYNTON, C. **Digital Video and HD: Algorithms and Interfaces**. Second. [S.l.]: Morgan Kaufmann, 2012. (The Morgan Kaufmann Series in Computer Graphics). ISBN 9780123919267.

RICHARDSON, I. E. G. **H.264 and MPEG-4 video compression: video coding for next-generation multimedia**. West Sussex, England: John Wiley & Sons Ltd, 2004. 206 p.

SEIDEL, I. **Análise do impacto de pel decimation na codificação de vídeos de alta resolução**. Dissertação (mestrado) — UFSC, 2014.

SEIDEL, I. **Exploiting SATD Properties to Reduce Energy in Video Coding**. Dissertação (Ph.D. thesis) — UFSC, 2019.

SEIDEL, I. et al. Coding- and energy-efficient FME hardware design. In: **2018 IEEE International Symposium on Circuits and Systems**. [S.l.]: IEEE, 2018.

SEIDEL, I. et al. Sad or satd? how the distortion metric impacts a fractional motion estimation vlsi architecture. In: **2021 IEEE 23rd International Workshop on Multimedia Signal Processing (MMSP)**. [S.l.: s.n.], 2021. p. 1–6.

SHI, Y. Q.; SUN, H. **Image and Video Compression for Multimedia Engineering: Fundamentals, Algorithms, and Standards**. second. [S.l.]: CRC Press/Taylor & Francis, 2008. (Image Processing Series). ISBN 9780849334917.

SILVA, G. G. et al. Approximate hardware architecture for interpolation filter of versatile video coding. **Journal of Integrated Circuits and Systems**, v. 16, n. 2, p. 1–8, 2021.

SIQUEIRA, I.; 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.

SULLIVAN, G.; WIEGAND, T. Rate-distortion optimization for video compression. **Signal Processing Magazine, IEEE**, v. 15, n. 6, p. 74–90, Novembro 1998.

SYNOPSYS. **Synopsys VCS, Version G-2012.09.** 2012.

SYNOPSYS. **Synopsys Design Compiler, V.P-2019.03-SP4.** 2019.

SYNOPSYS. **Synopsys Design Compiler, V.S-2021.06-SP4.** 2019.

Trudeau, L.; Coulombe, S.; Desrosiers, C. Rate distortion-based motion estimation search ordering for rate-constrained successive elimination algorithms. In: **2014 IEEE International Conference on Image Processing (ICIP)**. [S.l.: s.n.], 2014. p. 3175–3179. ISSN 1522-4880.

TSMC. **TSMC STANDARD CELL Library TCBN45GSBWPTC**. [S.l.], 2011.

VANNE, J. et al. Comparative rate-distortion-complexity analysis of HEVC and AVC video codecs. **IEEE Journal of Selected Topics in Signal Processing**, v. 22, n. 12, p. 1885–1898, Dezembro 2012.

VORONENKO, C. M. U. Y. **Spiral Software/Hardware Generator for DSP Algorithm**. 2017. Disponível em: http://spiral.ece.cmu.edu/mcm/gen.html. Acesso em: 2017-10.

WILLIAMS, S. **Icarus Verilog**. 2017. Disponível em: http://iverilog.icarus.com. Acesso em: july, 2020.

APPENDIX A – LIST OF PUBLICATIONS AND AWARDS

A.1 PUBLICATIONS AS FIRST AUTHOR

IEEE International Workshop on Multimedia Signal Processing (MMSP2021)

- **Title:** Hardware-Friendly Search Patterns for the Versatile Video Coding Fractional Motion Estimation
- **Authors:** RODRIGUES FILHO, Vanio; MONTEIRO, Marcio; SEIDEL, Ismael; GRELLERT, Mateus; GÜNTZEL, José Luís.
- **DOI:** 10.1109/MMSP53017.2021.9733603.

IEEE Symposium on Integrated Circuits and Systems Design (SBCCI 2020)

- **Title:** Standalone Rate-Distortion FME Architecture
- **Authors:** RODRIGUES FILHO, Vanio; MONTEIRO, Marcio; SEIDEL, Ismael; GRELLERT, Mateus; GÜNTZEL, José Luís. Standalone Rate-Distortion FME.
- **DOI:** 10.1109/SBCCI50935.2020.9189898.

A.2 CONTRIBUTION TO OTHER PUBLICATIONS

IEEE International Workshop on Multimedia Signal Processing (MMSP2021)

- **Title:** SAD or SATD? How the Distortion Metric Impacts a Fractional Motion Estimation VLSI Architecture
- **Authors:** SEIDEL, Ismael; RODRIGUES FILHO, Vanio; GRELLERT, Mateus; AGOSTINI, Luciano; GÜNTZEL, José Luís.
- **DOI:** 10.1109/MMSP53017.2021.9733518.

A.3 AWARD

Best Poster (Graduate Category) of the IEEE Seasonal School on Digital Processing of Visual Signal and Applications (DPVSA 2021)

- **Title:** Search Patterns to Reduce the Complexity of the VVC FME
- **Authors:** RODRIGUES FILHO, Vanio; MONTEIRO, Marcio; SEIDEL, Ismael; GRELLERT, Mateus; GÜNTZEL, José Luís.
- **url:** https://wp.ufpel.edu.br/dpvsa2021/best-poster-awards/

APPENDIX B – SYSTEMATIC LITERATURE REVIEW

This appendix contains the search string used and the systematic review process presented culminating with the related works in 3.3.

The first step of our systematic review was to define a search string. The string cannot be very restricted, otherwise there is a potential of filtering out works that may be related but had one or two parameters that did not match with the search string. Ideally, it should not be too broad either, otherwise the amount of works would be too large to be analyzed within a realistic time frame. Since this work proposes a new algorithm for the computation of the FME, our search string returns every work that can potentially be related.

The search string used for this literature review was:

video
AND
(compression OR coding OR encoding)
AND
(hevc OR "high efficiency video coding" OR vvc OR "versatile video coding")
AND
(fme OR "fractional motion estimation" OR "interpolation filter")

There was no need to filter by date, since we filter by coding standards. HEVC was release in 2013, while its successor, VVC, was just release in July 2020. Thus, adding this terms to the string made the search to return works published no earlier than 2013.

The database used for this review was Scopus, which is the largest abstract and citation database of peer-reviewed literature. As illustrated in Figure 31, the search string returned 129 references[1], of which, 126 were journals or conference proceedings written in English.

The next step was to eliminate any work that was not related to implementations of the FME for HEVC or VVC. We also removed neural networks design since those works are out of the scope of our work.

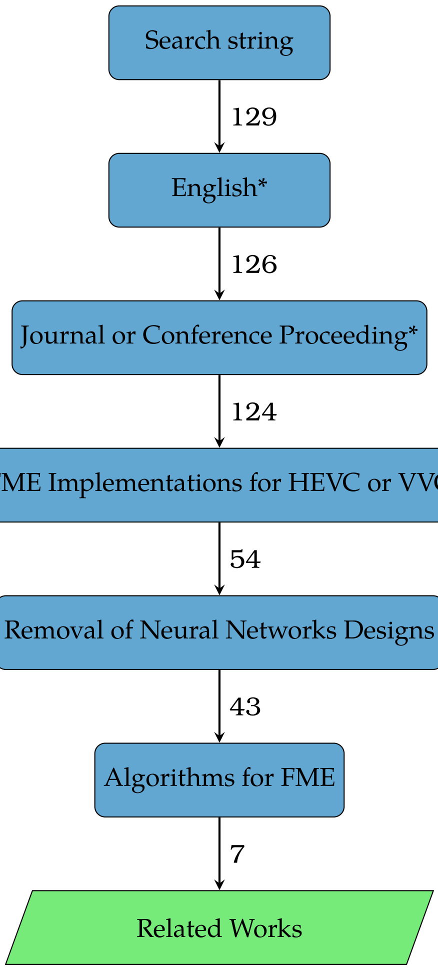
The final step was to identify works proposing alternative FME algorithms That is, we kept only works that propose modifications to the overall execution of the FME. By doing it so, removing works which main contributions were hardware implementation techniques. Since this techniques can be applied, in most cases, to any FME hardware architecture. We also eliminated multi standard architectures, mainly because they are focused on reusing hardware and calculations across different standards, while our work aims to explore the characteristics of a single standard (VVC). In addition, it would not

[1] Searched on October, 13th 2021.

be pointless to compare the designs since they have major differences in functionality and application.

Once applied the aforementioned decisions we ended up with seven related works, which are discussed in the next section.

Figure 31 – Systematic literature review process. The (∗) indicates steps that use Scopus's build-in search filters.


Search string
129
English*
126
Journal or Conference Proceeding*
124
FME Implementations for HEVC or VVC
54
Removal of Neural Networks Designs
43
Algorithms for FME
7
Related Works

Source: the author.