UNIVERSIDADE FEDERAL DE PELOTAS
Centro de Desenvolvimento Tecnológico
Programa de Pós-Graduação em Computação

Dissertação

Dedicated Hardware Architectures for the Affine Motion Estimation of Versatile Video Coding Video Standard for Real-Time UHD Videos

Marcello Morales Muñoz

Pelotas, 2025

Marcello Morales Muñoz

Dedicated Hardware Architectures for the Affine Motion Estimation of Versatile Video Coding Video Standard for Real-Time UHD Videos

Dissertação apresentada ao Programa de Pós-Graduação em Computação do Centro de Desenvolvimento Tecnológico da Universidade Federal de Pelotas, como requisito parcial à obtenção do título de Mestre em Ciência da Computação.

Advisor: Prof. Dr. Marcelo Schiavon Porto
Coadvisores: Prof. Dr. Luciano Agostini
Prof. Dr. Guilherme Corrêa

Pelotas, 2025

Universidade Federal de Pelotas / Sistema de Bibliotecas
Catalogação da Publicação

M967d Muñoz, Marcello Morales

Dedicated hardware architectures for the Affine motion estimation of versatile video coding video standard for real-time UHD videos [recurso eletrônico] / Marcello Morales Muñoz ; Marcelo Schiavon Porto, orientador ; Luciano Agostini, Guilherme Corrêa, coorientadores. — Pelotas, 2025.
81 f.

Dissertação (Mestrado) — Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, 2025.

1. Hardware. 2. Affine ME. 3. H.266/VVC. 4. Motion estimation. I. Porto, Marcelo Schiavon, orient. II. Agostini, Luciano, coorient. III. Corrêa, Guilherme, coorient. IV. Título.

CDD 005

Elaborada por Dafne Silva de Freitas CRB: 10/2175

Marcello Morales Muñoz

Dedicated Hardware Architectures for the Affine Motion Estimation of Versatile Video Coding Video Standard for Real-Time UHD Videos

Dissertação aprovada, como requisito parcial, para obtenção do grau de Mestre em Ciência da Computação, Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas.

Data da Defesa: 6 de dezembro de 2024

Banca Examinadora:

Prof. Dr. Marcelo Schiavon Porto (orientador)
Doutor em Computação pela Universidade Federal do Rio Grande do Sul.

Prof. Dr. Bruno Zatt
Doutor em Microeletrônica pela Universidade Federal do Rio Grande do Sul.

Prof. Dr. Alan Carlos Junior Rossetto
Doutor em Engenharia Elétrica pela Universidade Federal do Rio Grande do Sul.

Prof. Dr. Vladimir Afonso
Doutor em Microeletrônica pela Universidade Federal do Rio Grande do Sul.

AGRADECIMENTOS

Gostaria de agradecer as muitas pessoas que foram fundamentais para mim durante o período do mestrado. Agradeço, primeiramente, aos meus pais, Luiz Eduardo Machado Muñoz e Zoila Morales Fernandes, por todo o apoio que me deram ao longo da minha formação acadêmica, incluindo esta etapa do mestrado.

Gostaria também de expressar minha gratidão ao professor Marcelo Porto, meu orientador. Muito obrigado por ter me aceito como orientando durante a pandemia, mesmo com minha pouca experiência anterior em pesquisa e nesta área de conhecimento. Sou extremamente grato por essa oportunidade.

Agradeço igualmente aos meus co-orientadores, Luciano Agostini e Guilherme Correa, que, junto ao professor Marcelo, me ensinaram muito sobre codificação de vídeo e sobre o mundo da pesquisa acadêmica, áreas com as quais eu tinha pouco contato antes de iniciar o mestrado.

Meus sinceros agradecimentos também ao Murilo Perleberg, que me auxiliou bastante durante todo o processo do mestrado, e ao Denis Maass, cujo apoio foi igualmente essencial no desenvolvimento do meu trabalho. A contribuição de ambos foi crucial para a conclusão desta jornada.

Também quero expressar minha gratidão ao ViTech (Video Technology Research Group), do qual faço parte. Sem a infraestrutura e o suporte do grupo, este mestrado não teria sido possível.

Por fim, deixo meus agradecimentos à UFPel, onde iniciei minha trajetória em 2014 como estudante de graduação em engenharia da computação. Foi ali que aprendi tudo sobre computação e onde conheci quase todas as pessoas que, de alguma forma, contribuíram para este trabalho.

ABSTRACT

MUÑOZ, Marcello Morales. **Dedicated Hardware Architectures for the Affine Motion Estimation of Versatile Video Coding Video Standard for Real-Time UHD Videos**. Advisor: Marcelo Schiavon Porto. 2025. 81 f. Dissertation (Masters in Computer Science) – Technology Development Center, Federal University of Pelotas, Pelotas, 2025.

The demand for streaming digital videos over the internet has significantly increased in recent years. This substantial rise in video data traffic requires efficient compression techniques for storage and transmission. Moreover, these compression techniques must preserve image quality while addressing energy consumption and real-time processing constraints. In this context, the Versatile Video Coding (VVC) compression standard, also known as H.266, is the state-of-the-art video compression standard released in July 2020 by the Joint Video Experts Team (JVET).

VVC introduced several new tools to aid video compression compared to its predecessor, the H.265/HEVC standard. This results in significant gains in compression efficiency (the ratio of bit rate reduction to the encoded video quality), allowing for more efficient transmission or storage of digital videos. One of the new tools added in VVC is Affine prediction, which is used to characterize non-translational motion. This new tool has the highest computational cost in the entire inter-frame prediction stage of the VVC encoder. However, Affine prediction can provide gains of up to 2.18% in compression efficiency (BD-Rate) in exchange for a 5.26% increase in encoding time.

Affine prediction supports two versions, using four or six parameters, where two or three motion vectors are used, respectively. These vectors are inherited from the reference block to be encoded and are used for reconstructing the block. This work was designed and synthesized using ASIC hardware architectures for the Affine motion estimation in the VVC standard, focusing on low energy consumption. The architectural design emphasizes the capability to process real-time UHD 4K videos (3840×2160), aiming for its application in mobile devices. Three architectures are presented: One for Affine motion compensation with an area utilization of 189k Gates and 121mW power dissipation. Another for the architecture that calculates the Affine ΔMV with 245k Gates and 31mW of power dissipation. Also, an initial version of the gradient-based coefficient generator is presented with 6,900k Gates and 4.8W power dissipation.

Keywords: Hardware; H.266/VVC; Affine ME; Motion Estimation.

RESUMO

MUÑOZ, Marcello Morales. **Arquiteturas de Hardware Dedicada para a Estimação de Movimento Affine UHD em tempo real do Padrão de Compressão de Vídeo Versatile Video Coding**. Orientador: Marcelo Schiavon Porto. 2025. 81 f. Dissertação (Mestrado em Ciência da Computação) – Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, Pelotas, 2025.

A demanda de streaming de vídeos digitais pela internet está aumentando de forma significativa nos últimos anos. Este grande aumento no tráfego de dados de vídeo requer técnicas de compressão eficientes para armazenamento e transmissão. Além disso, estas as técnicas de compressão devem preservar a qualidade da imagem, enquanto lidam com restrições de consumo de energia e processamento em tempo real. Neste cenário, o padrão de compressão de vídeo *Versatile Video Coding* (VVC), também conhecido como H.266, é o padrão de compressão de vídeo estado-da-arte, tendo sido lançado em julho de 2020 pelo *Joint Video Experts Team* (JVET).

O VVC adicionou diversas novas ferramentas para auxiliar na compressão de vídeo, quando comparado ao seu antecessor, o padrão H.265/HEVC. Desta forma, proporcionando grandes ganhos em termos de eficiência de compressão (relação entre redução da taxa de bits e a qualidade do vídeo codificado) e permitindo a transmissão ou armazenamento de vídeos digitais de forma mais eficiente. Uma das novas ferramentas adicionadas no VVC é a predição *Affine*, que é utilizada para caracterizar movimentos não-translacionais, sendo que essa nova ferramenta possui o maior custo computacional de toda a etapa de predição inter-quadros do codificador VVC. A predição *Affine* pode trazer ganhos de até 2,18% na eficiência de compressão (BD-Rate) em troca de um aumento de 5,26% no tempo de codificação.

A predição *Affine* suporta duas versões, utilizando quatro ou seis parâmetros, onde dois ou três vetores de movimento são usados, respectivamente. Esses vetores são herdados do bloco de referência a ser codificado e são usados para reconstruir o bloco. Neste trabalho foi projetado e sintetizado arquiteturas de *hardware* ASIC para predição *Affine* do padrão VVC com foco em baixo consumo de energia. As arquiteturas são projetadas para processar vídeos UHD 4K (3840×2160) em tempo real, visando sua aplicação em dispositivos móveis. Três arquiteturas são apresentadas: uma para compensação de movimento *Affine*, com uma de área de 189k Gates e dissipação de potência de 121mW. Outra para a arquitetura que calcula o ΔMV *Affine*, com 245k Gates e dissipação de 31mW. Além disso, uma versão inicial do gerador de coeficientes baseado em gradiente do *Affine* com 6.900k Gates e dissipação de 4,8W.

Palavras-chave: Hardware; H.266/VVC; Affine ME; Estimação de movimento.

LIST OF FIGURES

Figure 1 Flowchart of the video compression 19
Figure 2 Elements present in the ME (Porto, 2012) 20
Figure 3 Search process of the ME 21
Figure 4 Affine models. (a) 4-Parameter with LT and RT MVs. (b) 6-Parameter with LT, RT, and LB MVs. 22
Figure 5 Affine AMVP Algorithm. 23
Figure 6 Affine representation of the subblock split, where 16 subblocks with 4x4 samples compose one 16x16 PU. Each subblock has its own SMV. 24
Figure 7 Representation of diagonal interpolation of a 4x4 subblock. The blue squares are the integers, the triangles are horizontal, and the stars are diagonal samples. 25
Figure 8 Representation of the Gradient-Based Iterative Algorithm. 27
Figure 9 The kernel Sobel Filter in the X- and Y-direction. 28
Figure 10 Representation of the Block-Matching Iterative Algorithm. 32
Figure 11 Representation of all the BMIA possible positions explored after three rounds for **one** CPMV. 33

Figure 12 High-level architecture of Affine MC 47
Figure 13 Architecture of the Vector Generation unit 48
Figure 14 Architecture that calculates MV differences 49
Figure 15 Architecture of the Interpolation Unit 49
Figure 16 Proposed Architecture of the Interpolation Filter F_8 50
Figure 17 Implementation of the Filter F_1 using a multiplierless approach in the Baseline architecture 50
Figure 18 Proposed Architecture of the Interpolation Filter F_1 or F_{15} (used in the PE architecture), depending on the order of the inputs. 51
Figure 19 Proposed Hardware-Efficient Architecture. 52
Figure 20 The Architecture of the Coefficient Multiplier C_1, used in HE architecture. 53
Figure 21 Time diagram of the Affine MC architecture when $LT = [4, -1332]$ and $RT = [-72, -1336]$ 54
Figure 22 High-level architecture of the GBIA 55
Figure 23 Architecture of the Sobel Engine 55
Figure 24 Sobel Unit architecture 56
Figure 25 Architecture of the Affine Parameter Generator 57
Figure 26 Architecture of the Coefficient generator 57

Figure 27 Architecture of the Affine Mode Adapter 58
Figure 28 Architecture of the Affine Mode Adapter Converter 58
Figure 29 Temporal Analysis of the Gradient-Based Coefficient Generator . . . 60
Figure 30 The Affine ΔMV Architecture. 61
Figure 31 Time diagram of the proposed Affine ΔMV architecture. 62

LIST OF TABLES

Table 1 Affine Filter Coefficients. 27
Table 2 SLE of 6-parameters. 29
Table 3 SLE of 4-parameters. 30

Table 4 Results of the Affine Motion Estimation in the VVC for **four** Reference Frames. 37
Table 5 Average Processing of the Affine Motion Estimation divided by PU size for four Reference Frames. 38
Table 6 Results of the Affine Motion Estimation in the VVC for **one** Reference Frames. 39
Table 7 Average Processing of the Affine Motion Estimation divided by PU size for **one** Reference Frames. 40
Table 8 BD-Rate of the PU constraints evaluated. 42
Table 9 BD-Rate of the Iteration constraints evaluated. 42
Table 10 BD-Rate of the proposed combined constraints evaluated. 44

Table 11 Conversions Equations of the SLE 6-parameters to 4-parameters. . 59

Table 12 Synthesis results for Interpolation Unit using **four** Filter Cores working at 808.7 MHz. 64
Table 13 Synthesis Results for the Affine MC for 4K@60fps while only processing GBIA and 16x16 PUs. 65
Table 14 Synthesis Results for the Gradient-Based Coefficient Generator for UHD 4K@60fps for both throughputs, GBIA and quadratic sizes (186MHz) and GBIA and 16x16 (48MHz). 67
Table 15 Synthesis results of the Affine ΔMV architecture when targeting 4K UHD 60 fps videos 69
Table 16 Comparison among the proposed Affine MC architecture and FPGA related works. 70
Table 17 Comparison among the proposed Affine MC architecture and ASIC related works. 71
Table 18 Comparison with related work for the Affine ΔMV architecture when targeting HD 1080p@60fps videos 71

LIST OF ABBREVIATIONS AND ACRONYMS

UHD	Ultra-High Definition
JVET	Joint Video Experts Team
VVC	Versatile Video Coding
SDR	Standard Dynamic Range
HDR	High Dynamic Range
WCG	Wide Color Gamut
ME	Motion Estimation
CME	Conventional Motion Estimation
AME	Affine Motion Estimation
FPS	Frames per Second
CU	Coding Unit
MC	Motion Compensation
MV	Motion Vectors
FME	Fractional Motion Estimation
VTM	VVC Test Model
BD-Rate	Bjontegaard Delta Rate
GBIA	Gradient-Based Iterative Algorithm
BMIA	Block Matching Iterative Algorithm
SLE	System of Linear Equations
VCEG	Video Coding Experts Group
MPEG	Moving Picture Experts Group
HEVC	High Efficiency Video Coding
NAL	Network Abstraction Layer
CTU	Coding Tree Unit
PU	Prediction Unit
RD	Rate Distortion

CPMV	Corner Point Motion Vectors
MM	Merge Mode
AMVP	Advanced Motion Vector Prediction
AMC	Affine Motion Compensation
SMV	Subbblock Motion Vectors
LT	Left-Top
RT	Right-Top
LB	Left-Bottom
PROF	Prediction Refinement with Optical Flow
FPGA	Field-Programmable Gate Array
ASIC	Application-Specific Integrated Circuit
MCM	Multiple Constant Multiplication
CTC	Common Test Conditions
QP	Quantization Parmeters
PE	Power-Efficient
HE	Hardware-Efficient
MMCM	Multiplexed Multiple Constant Multiplication
TSMC	Taiwan Semiconductor Manufacturing Company
RTL	Register-Transfer Level

CONTENTS

1 INTRODUCTION 15

2 BACKGROUND 18
2.1 Versatile Video Coding 18
2.2 Affine Motion Estimation 21
2.2.1 Interpolation Filters 24
2.2.2 Gradient-Based Iterative Algorithm 26
2.2.3 Block Matching Iterative Algorithm 31
2.3 Related Works 32

3 SOFTWARE EVALUATIONS 35
3.1 Experimental Setup 35
3.2 Hardware-Oriented Constraints Evaluation 36
3.2.1 Results and BD-Rates of Reference Frames Constraints 36
3.2.2 BD-Rate of PU Size Constraints 41
3.2.3 BD-Rate of Iteration Constraints 42
3.2.4 BD-Rate of Proposed Approach for the AME Hardware Design 43
3.3 Experiments Conclusions 44

4 PROPOSED AME HARDWARE ARCHITECTURES 46
4.1 Affine MC 46
4.1.1 Vector Generator 47
4.1.2 Interpolation Unit 47
4.1.3 Design Exploration of Filter Core 48
4.1.4 Temporal Analysis 53
4.2 Gradient-Based Coefficient Generator Architecture 54
4.2.1 Row Buffer 54
4.2.2 Sobel Engine 55
4.2.3 Parameter Generator 56
4.2.4 Coefficient Generator 57
4.2.5 Affine Mode Adapter 58
4.2.6 Temporal Analysis 59
4.3 Affine ΔMV Architecture 60
4.3.1 Gauss-Jordan Elimination Algorithm 60
4.3.2 Proposed architecture 61
4.3.3 Temporal Analysis 62

5 SYNTHESIS RESULTS AND COMPARISONS 63
5.1 Interpolation Unit Results 63
5.2 Affine MC Synthesis Results 65
5.3 Gradient-Based Coefficient Generator 66
5.4 Affine $\triangle MV$ Architecture 68
5.5 Comparison with related works 69
6 CONCLUSION 72
REFERENCES 74
APPENDIX A LIST OF PUBLICATIONS DURING THIS MASTERS DEGREE 79

1 INTRODUCTION

The COVID-19 pandemic greatly impacted global internet traffic (bitmovin, 2020). With internet usage continuing to rise, it is estimated that in 2023, 5.3 billion people have had access to the internet (Cisco, 2018), an increase from 3.9 billion in 2018. As a result, data consumption on the internet has also been increasing year after year, with video streaming representing a significant portion of this data transfer volume. Data from the first half of 2021 shows that video streaming dominated internet bandwidth, accounting for 53% of the total data transfer volume (Sandvine, 2022). In addition to the growing demand, there is an increasing evolution in video resolutions and frame rates. Projections estimated that, in 2023, two-thirds (66%) of TVs were 4K UHD (Ultra-High Definition) resolution (Cisco, 2018). In this scenario, the need for greater efficiency in digital video compression becomes evident.

There is an ongoing effort within the scientific community to advance video compressors. In 2020, the Joint Video Experts Team (JVET) released the Versatile Video Coding (VVC) standard (Chen; Ye; Kim, 2021), which represents the state-of-the-art in video coding standards. VVC is the successor to High Efficiency Video Coding (HEVC) (Jct-vc, 2013), also developed by JVET. VVC was designed to efficiently encode 4K Ultra-High-Definition (UHD) with Standard Dynamic Range (SDR) and also encode 8K UHD or even larger resolutions, High Dynamic Range (HDR), and Wide Color Gamut (WCG).

The VVC was designed to reduce bit rate and be versatile by processing different types of content such as process screen content, gaming, and 360° video for immersive and augmented reality (Bross et al., 2021) In VVC, many tools were added to assist in video compression, enabling a 44.4% reduction in the amount of data required to represent videos (also known as bitrate) (Siqueira; Correa; Grellert, 2020), with a 5-fold increase in encoding complexity in terms of encoding time, compared to the HEVC (Pakdaman et al., 2020).

One of the tools of the VVC is inter-frame prediction, which uses a reference frame to predict the current frame samples. The Motion Estimation (ME) is a step inside the inter-frame prediction that determines the motion vectors between the reference frame

and the current frame being encoded. The ME is one of the most essential steps in video encoding because two frames that are temporally near each other may have a lot of redundancies. The ME is divided into Conventional ME (CME) and Affine ME (AME): CME uses one Motion Vector (MV) to describe translation motion, which is motion in which all points move in the same direction and distance. The AME uses two or three MV to represent more complex types of motion, such as rotation, zoom in/out, and shear.

Inter-frame prediction accounts for, on average, 59% of the total encoding time in the VVC standard (Gonçalves, 2021). Inter-frame prediction can be subdivided into unilateral, bilateral, and Affine predictions. Affine prediction has the highest computational cost, representing an average of 54.75% of the total inter-frame prediction time (Park; Kang, 2019). Since AME constitutes a significant portion of the total motion estimation processing time, reducing this complexity becomes crucial for real-time video encoding, especially for high resolutions like UHD 4K and 8K and high frame rates (FPS). Along with reducing complexity, it is also necessary for AME to have low energy consumption for applications on battery-powered mobile devices, making the use of dedicated hardware essential for encoding UHD videos under the H.266/VVC standard.

Only one work in the literature proposes dedicated hardware architecture for the entire AME process, which is (Taranto, 2022). However, it simplifies the algorithm, skipping many steps to reduce the overall computational complexity, which leads to visual degradation not being calculated in the work. Besides that, many works are proposing dedicated hardware architectures that can process one of the many steps of the AME: Sheng et al. (2024) propose solver for System of Linear Equations (SLE) for the Affine ME and Maass et al. (2023) propose an SLE solver and final processing to calculate the new set of MV. Perleberg et al. (2023) proposes a hardware architecture for the Prediction Refinement with Optical Flow (PROF) algorithm. The Affine MC can be compared to the Fraction Motion Estimation (FME) because of the filtering operations, and there are many works for the FME such as Azgin et al. (2018); Canmert; Kalali; Hamzaoglu (2018); Azgin; Kalali; Hamzaoglu (2020); Mahdavi; Azgin; Hamzaoglu (2021); Silva et al. (2021) even though choosing the filter uses Affine equations, which would require an additional calculation.

To understand the throughput necessary to process the AME, this work performed an analysis of the Affine ME done using the reference software of the VVC, the VVC Test Model (VTM). This analysis may also be used to understand how often the AME is processed inside the ME. With the results from the analysis, this work presents dedicated hardware architectures with the objective of processing steps inside the AME algorithm for the Affine 4-parameter and 6-parameter. These architectures were also designed to process any Prediction Unit (PU) size ranging from 16x16 to 128x128. In

addition, this work presents a design space exploration showing different approaches to designing these architectures for different target throughput and area/power constraints.

The main contributions of this Master's dissertation work:

- An analysis of the AME algorithm. This analysis shows the Bjontegaard Delta Rate (BD-Rate) of different constraints, the number of times the AME 4-parameter and 6-parameter are processed, the number of times Gradient-Based Iteration Algorithm (GBIA) and Block-Matching Iterative Algorithm (BMIA) are processed, and these values for each PU size.

- The development of energy-efficient hardware designs for the AME of the VVC encoding standard. These architectures were developed to process all PU sizes supported in VVC AME. These architectures can process 4K Ultra High-Definition (UHD) videos at 60 frames per second.

This work presents three hardware architectures for the Affine ME. The first architecture presented is the Affine MC, which is used by both GBIA and BMIA. The synthesis of this architecture presents an area utilization of 189.2k gates with a power dissipation of 121.15mW with a frequency of 598MHz for processing UHD 4K@60fps. The second architecture presented is the Gradient-Based Coefficient Generator. The synthesis of this architecture presents an area utilization of 4849k gates with a power dissipation of 3932mW with a frequency of 48MHz for processing 4K@60fps. The results are prohibitive for real applications; however, the Gradient-Based Coefficient Generator is the first architecture for this step of the iterative algorithm in the literature. The last architecture presented is the Affine ΔMV architecture, which solves the SLE generated by the Gradient-Based Coefficient Generator and generates the Affine ΔMV, which is added to the current MV being evaluated, generating the next set of MVs to be evaluated. The synthesis of this architecture presented an area of 262k gates and 74.95mW of power dissipation.

This dissertation is organized as follows: The chapter 2 explains basic concepts of the VVC video coding standard, the AME algorithm, which is the focus of this work, and presents the related works. After, chapter 3 presents the evaluations performed to define the throughput of the developed architectures, while chapter 4 shows in detail the developed architectures. In chapter 5, the obtained syntheses are presented and compared with relevant related works. Finally, the chapter 6 concludes this work.

2 BACKGROUND

In this chapter, some background information necessary to understand this dissertation will be presented. Section 2.1 presents the Versatile Video Coding (VVC) and the basic information on how it encodes a video. Section 2.2 presents the AME algorithm, which is a new tool added in VVC and the focus of this work. Section 2.3 discusses the related works, presenting all published hardware architectures that will be compared to the synthesis results achieved in this work.

2.1 Versatile Video Coding

The VVC standard (Chen; Ye; Kim, 2021) is the state-of-the-art video compressor. The VVC was developed by the Joint Video Experts Team (JVET), which was formed by the Video Coding Experts Group (VCEG) and the Moving Picture Experts Group (MPEG). JVET issued a call for proposals in October 2017, and the first draft of the VVC was released in April 2018, with the completion of the standard in July 2020 (Jvet, 2018).

The VVC is the successor of the H.265/High Efficiency Video Coding (HEVC) (Jct-vc, 2013), and it has an improvement of 50% of bitrate reduction over its predecessor (Bross et al., 2021). The standard is named Versatile because it covers current and emerging media needs:

- Higher resolutions than its predecessor (up to 8K or larger), HDR and WCG.
- Computer-generated or screen content, as occurs in screen sharing and gaming.
- 360° video for immersive and augmented reality.
- Applications requiring ultralow delay, such as wireless displays and online gaming.

The VVC, similar to its predecessor, the HEVC, uses a block-based hybrid video coding scheme and the bitstream structure based on the network abstraction layer (NAL), and the basic processing unit is the Coding Tree Unit (CTU).

The VVC uses the same block-based partitioning scheme as its predecessor. The encoder divides the frame into Coding Tree Units (CTU). The CTU in VVC has a size of 128x128, which is the largest Coding Unit (CU) supported. Each CTU is then further divided into one or more blocks known as CU of varying sizes, and the CU is the basic processing unit of the encoder. Finally, the CU is divided into Prediction Units (PU), the units evaluated in the prediction steps of the video encoders, such as Inter-Frame or Intra-Frame prediction. The PU was chosen by evaluating its rate-distortion (RD) cost. The encoder selects the PU with the smallest RD cost because a smaller RD cost leads to fewer bits needed in the bitstream.

After the frame is split into PU, the VVC encoder uses the encoding steps of its predecessor, the core design of video data compression can be seen in Figure 1 and has five stages: prediction, transform, quantization, in-loop filters, and entropy encoding. The prediction stage can be divided into two: intra-frame prediction, which uses the similarities of the current frame to encode it, and inter-frame prediction, which searches for similarities in previously encoded frames. The transform stage changes the values to the frequency domain. In a lossy stage, the data in the frequency domain gets quantized to reduce information irrelevant to human visual perception. Finally, in the entropy encoding stage, the statistical redundancies are reduced by an entropy encoder applying arithmetic coding.

The Motion Estimation (ME) is a step inside the inter-frame prediction used to describe the motion from two frames, a reference frame and the current frame being encoded. The ME is one of the most essential steps in video encoding because two frames that are temporally near each other may have a lot of redundancies. The ME

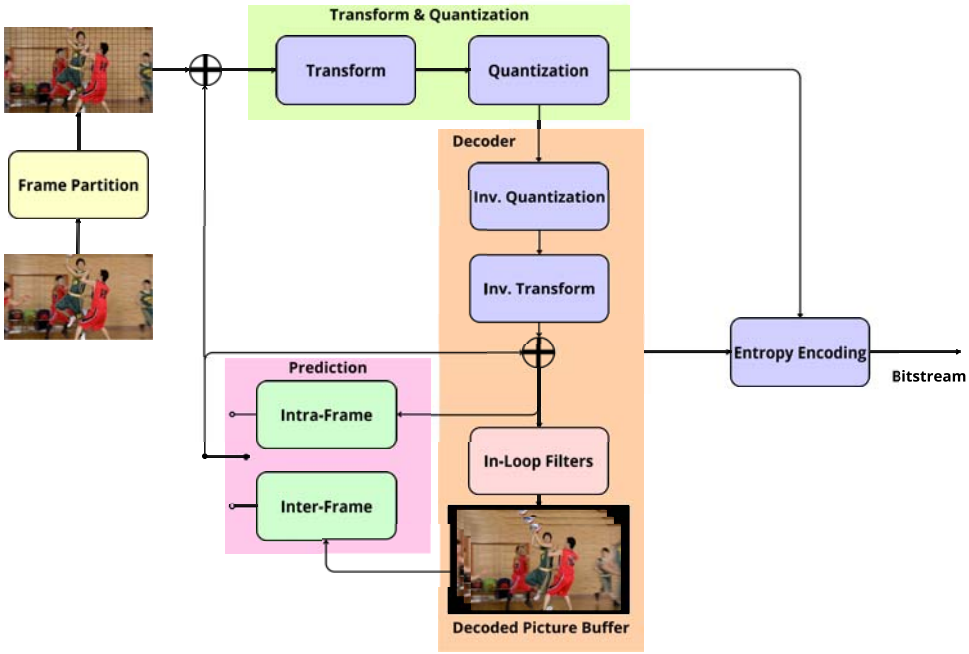

Figure 1 – Flowchart of the video compression

is divided into Conventional ME (CME) and Affine ME (AME): CME uses one Motion Vector (MV) to describe translation motion, which is motion in which all points move in the same direction and distance. The AME uses two or three MV to describe more complex types of motion, such as rotation, zoom in/out, and shear.

The Figure 2 presents the ME process. The ME searches for the best match for the current block in the reference frame; this search is not performed in all of the positions in the reference frame but only in a search area around the position of the block in the reference frame. A MV describes the motion between the block position in the current frame and the best match. This best match has its cost calculated and compared with the best matches found in the other reference frames.

The inter-frame prediction exploits similarities of the frames temporally close to reduce massively the data transferred. The process of finding a match in a reference frame in the inter-frame prediction is called ME. The ME process can be seen in Figure 3. The process applies different types of prediction in succession, however, it can be skipped early depending on the cost calculated. The ME divides the frames into two lists (L0, L1), one for frames in the past and another for frames in the future then the prediction modes are tested. With the two lists of frames, two types of prediction modes are supported: unilateral and bilateral. Uni-lateral prediction searches the frames in L0 or L1 for the best match to encode the PU. The bi-lateral prediction mode uses two frames to encode the PU, using both lists L0 and L1. These prediction modes can be further split into two types of prediction modes: Conventional and Affine. Conventional ME describes translation motion between two frames, that is, motion in which all points have the same direction. The Affine Motion Estimation is a new tool added to the VVC that uses multiple MVs to better encode more complex motions such as rotation and zoom. The Affine Motion Estimation has two different modes, which use two MVs (4-parameters) or three MVs (6-parameters) to describe the motion. Both Conventional ME and Affine ME support uni-lateral and bi-lateral modes.

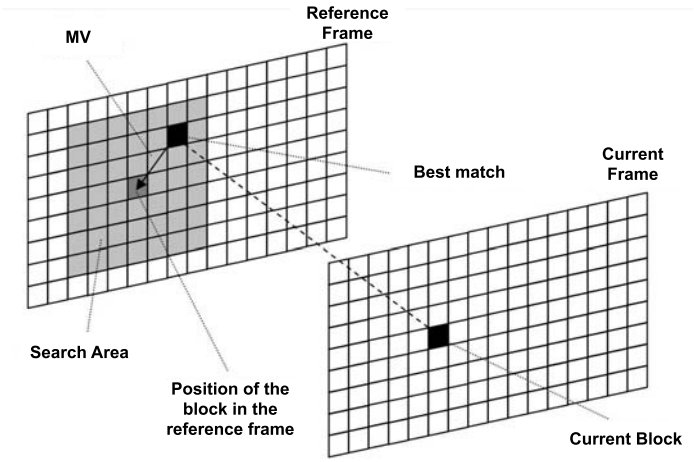

Figure 2 – Elements present in the ME (Porto, 2012)

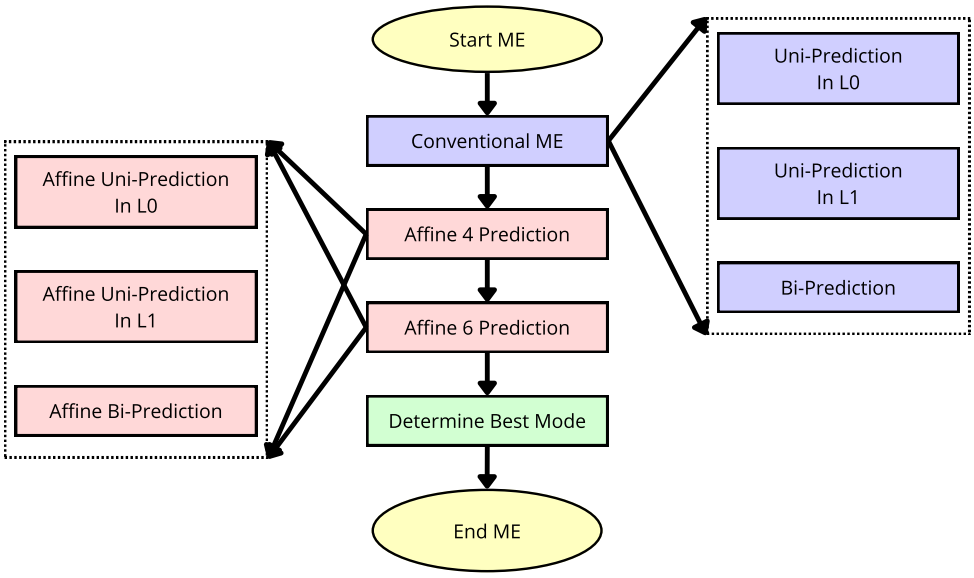

Figure 3 – Search process of the ME

The AME can describe more complex motion, leading to reduced error, reducing the bits necessary for transmission, and increasing visual fidelity. However, this reduced bandwidth/size and better visual fidelity have a higher computational complexity cost than Conventional ME.

The many new tools added to VVC enable the gain in coding efficiency. This work focuses on AME, one of the new tools added to the Inter-frame prediction.

2.2 Affine Motion Estimation

The Affine Motion Estimation (AME) is a new tool adopted in ME of VVC. The AME is used in inter-frame prediction to identify and represent more complex movements than the traditional translational movement, such as scaling, rotation, and skewing (Bross et al., 2021). As in ME, the AME performs a search for a similar block in a reference frame previously encoded, resulting in a set of two or three MVs that will be used to represent the current block with a non-translational movement (Chen; Ye; Kim, 2021).

The VVC AME supports two Affine models, with 4 or 6 parameters (Bross et al., 2020), as presented in Figure 4 (a) and (b). The 4-parameter model uses Corner Point Motion Vectors (CPMVs) in the top-right corner and the top-left corner of the PU, while the 6-parameter uses the three CPMVs in the top-left, bottom-left, and top-right corner of the PU as can be seen in Figure 4 (Bross et al., 2020). The AME is applied over 12 Prediction Unit (PU) sizes ranging from 16x16 to 128x128 (Bross et al., 2020).

The Affine MVs can be obtained in two ways: Affine Merge Mode and Affine Advanced Motion Vector Prediction (AMVP) (Chen; Ye; Kim, 2021). Affine Merge Mode inherits the MV from the neighboring PUs that were used to encode either the AME or

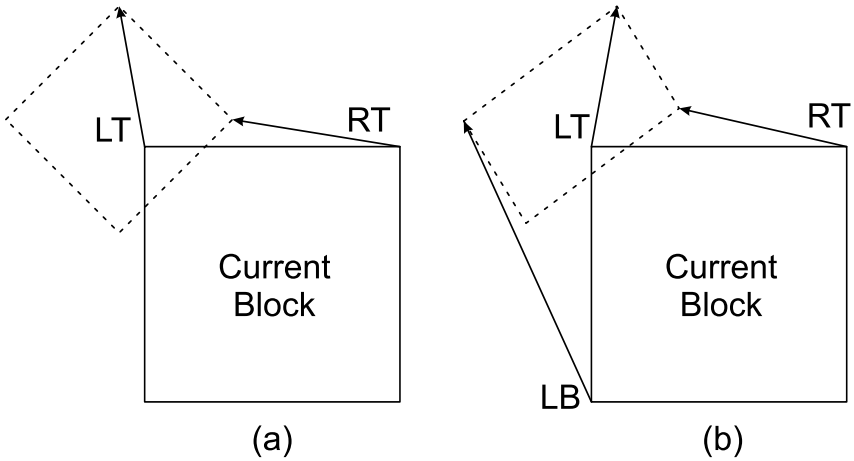

Figure 4 – Affine models. (a) 4-Parameter with LT and RT MVs. (b) 6-Parameter with LT, RT, and LB MVs.

ME. Affine AMVP is an iterative algorithm that minimizes the error in the starting MVs of a block, and this algorithm may motion compensate the block up to 82 times, leading to high complexity.

The Affine Merge Mode (MM) requires very few CPMVs to be evaluated. The list of candidate CPMVs that need to be evaluated is formed by using CPMVs from the Affine MM of neighboring PUs, candidates constructed CPMVs that are derived from the CME of neighboring PUs, and Zero MVs. This list has a maximum size of five CPMVs that are required to be evaluated. This algorithm has a very low complexity compared to Affine AMVP, but it may have a higher error, which leads to reduced encoding efficiency.

The Affine AMVP algorithm evaluates only two candidates: the first uses two CPMVs (Affine 4-parameter), and the second uses three CPMVs (Affine 6-parameter). Even though the list of candidates is smaller, it uses a more complex algorithm to minimize the error and find the best CPMVs. This algorithm is divided into two steps: A Gradient-Based Iterative Algorithm (GBIA) and a Block Matching Iterative Algorithm (BMIA). Affine AMVP candidate list of two CPMVs requires higher computational complexity than Affine Merge Mode, with up to 82 CPMVs needing to be evaluated. Also, this does not consider the steps necessary to generate a new set of MVs, which has a high computational complexity and requires a lot of memory access.

As can be seen in Figure 5, both iterative algorithms in the AMVP require Affine Motion Compensation (MC) of the PU with the starting CPMVs. In each iteration of the algorithms, the CPMVs are updated, either through the use of the error, in the case of the GBIA, or through searching neighboring positions, in the case of the BMIA. The GBIA can repeated up to five times, while the BMIA can be up to 24 times per CPMV (up to 48 times for Affine 4-parameter or 72 for Affine 6-parameter). The AMVP also contains five additional Affine MCs between GBIA and BMIA, one for each CPMV, one to keep rotation/zoom, and one to keep translation (Chen; Ye; Kim, 2021).

The Gradient-Based Iterative Algorithm (GBIA), as can be seen in Figure 5, first

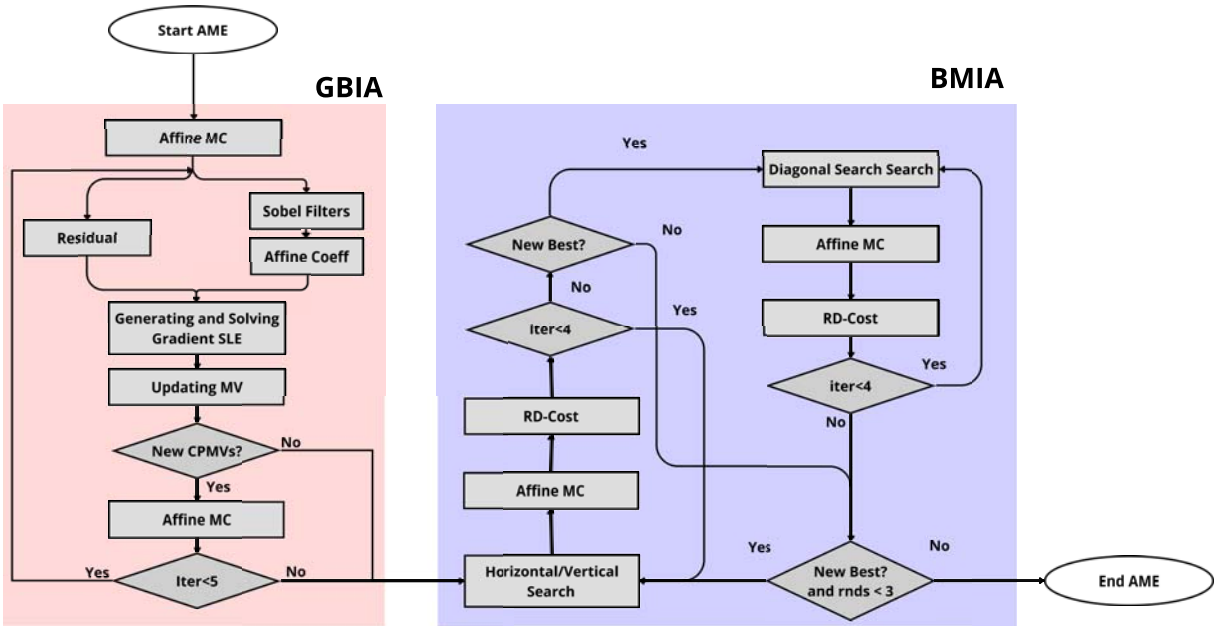

Figure 5 – Affine AMVP Algorithm.

performs the Affine MC the PU with the starting CPMVs, then uses Sobel Filters to calculate four or six Affine parameters depending on the model. These Affine parameters and the residual (error between the predicted and original samples) are then used to generate a System of Linear Equations (SLE) based on the gradient, and this SLE has a size of 4x5 or 6x7. This SLE, when solved, generates 4 or 6 ΔMV, one for each CPMV, depending on the Affine model, which is added to the respective CPMV, generating a new set of CPMVs that are evaluated in the next iteration. This repeats as long as at least one CPMV is updated or up to five times (Chen; Ye; Kim, 2021). After the GBIA, there are five motion compensations: one for each CPMV up to three, one to keep rotation/zoom, and another to keep translation.

Following the GBIA, the Block Matching Iterative Algorithm (BMIA) is used, as seen in Figure 5. The BMIA contains up to 24 Affine MC for each CPMV. First, there is a horizontal/vertical search of the nearest neighbors. If any neighbor leads to a lower cost, the current CPMV is updated as the new starting CPMV. After the last step, a search is performed in the diagonal neighbors, and the CPMV is updated if any have a lower cost. The horizontal/vertical search, followed by the diagonal search, is repeated three times. The diagonal search is only performed if a better MV is found in the horizontal/vertical search, and the horizontal/vertical is repeated up to three times, as long a better CPMV is found (Chen; Ye; Kim, 2021).

Evaluating each set of MVs requires an Affine Motion Compensation (AMC) to reconstruct the current Prediction Unit (PU) with the samples related to the set of CPMVs (Chen; Ye; Kim, 2021). In the AMC of VVC, the PU is split into subblocks of 4x4 samples, and each subblock passes by an individual 4x4 MC, similar to the conventional ME (Chen; Ye; Kim, 2021; Bross et al., 2020). This split may be seen in Figure 6,

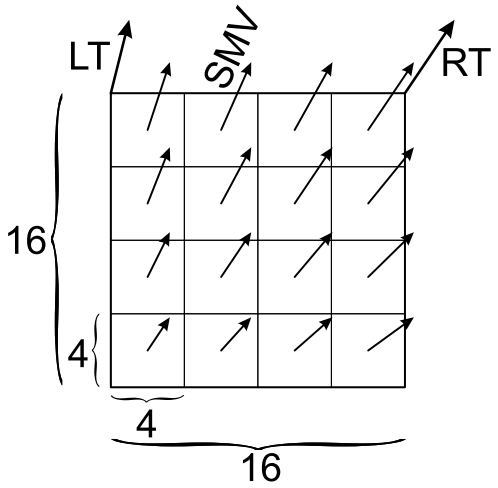

Figure 6 – Affine representation of the subblock split, where 16 subblocks with 4x4 samples compose one 16x16 PU. Each subblock has its own SMV.

where 16 subblocks of 4x4 samples are used to compose a 16x16 PU. The difference in the AMC is that each subblock receives a Subblock Motion Vector (SMV) that is obtained using the Affine equations defined in 1 and 2, according to the Affine model of 4-parameter or 6-parameter, respectively (Bross et al., 2020).

$$\begin{cases} SMV^H = \frac{RT^H - LT^H}{bW}cx + \frac{RT^V - LT^V}{bW}cy + LT^H \\ SMV^V = \frac{RT^V - LT^V}{bW}cx + \frac{RT^H - LT^H}{bW}cy + LT^V \end{cases} \tag{1}$$

$$\begin{cases} SMV^H = \frac{RT^H - LT^H}{bW}cx + \frac{LB^H - LT^H}{bH}cy + LT^H \\ SMV^V = \frac{RT^V - LT^V}{bW}cx + \frac{LB^V - LT^V}{bH}cy + LT^V \end{cases} \tag{2}$$

In (1) and (2), the LT, RT, and LB are the Left-Top, Right-Top, and Left-Bottom CPMVs inherited from the PU, where the H and V represent the horizontal and vertical parts of those CPMVs. The cx and cy are the positions of the center of the subblock, while the bW and bH represent, respectively, the width and height of the PU. The output is the horizontal and vertical parts of the SMV, which are used for the MC of each 4x4 subblock.

These SMVs generated using the equation may have a motion that does not align with the samples of the reference frame or a position between samples, making it necessary to interpolate fractional samples. The VVC has a precision of $1/16$ (Bross et al., 2021), which means that up to 15 different samples can be interpolated between two samples depending on SMV. These fractional samples are interpolated using filters.

2.2.1 Interpolation Filters

In digital videos, the motion that occurs between adjacent frames may not align in integer sample positions, resulting in SMVs having fractional parts. Therefore, VVC requires a Motion Compensation (MC) with $1/16$ sample interpolation to interpolate the

4x4 subblock in the fractional positions (Sullivan et al., 2012; Bross et al., 2020).

To reach an MC with $1/16$ sample interpolation, the VVC adopts a set of 15 interpolation filters (Bross et al., 2020, 2021). Therefore, between two horizontally neighbor samples, 15 horizontal samples may be interpolated. The horizontal interpolation occurs when the component SMV^V is equal to zero. In the same way, between two vertically neighbor integer samples, 15 new vertical fractional samples can be generated, which occurs when the component SMV^H is equal to zero. Besides, when both SMV^H and SMV^V are different from zero, the diagonal interpolation process can generate up to 225 diagonal fractional samples. To do this diagonal interpolation, it is required first to interpolate the horizontal fractional samples, and then the fractional samples can be used as input in vertical interpolation (Afonso et al., 2016).

Figure 7 presents the interpolation of fractional samples to reconstruct a 4x4 subblock for a diagonal SMV (when X and Y are different from zero). The blue squares represent the integer samples, and the colored stars represent the samples of the 4x4 subblock to be interpolated, with the triangles being the temporary samples required for the interpolation. Nine horizontal neighboring squared samples are used to interpolate the four triangle temporary samples for each line (colored triangles inside the horizontal ellipses). Then, the colored star samples are interpolated using a similar process. However, using temporary triangle neighboring samples, this process is performed for each of the four columns to generate the diagonal samples (colored stars inside the vertical ellipses). Finally, after processing nine lines and four columns, a 4x4 subblock can be obtained.

The AME adopted 6-tap filters during the interpolation process. Therefore, to interpolate one fractional sample, six input samples are necessary. The interpolation

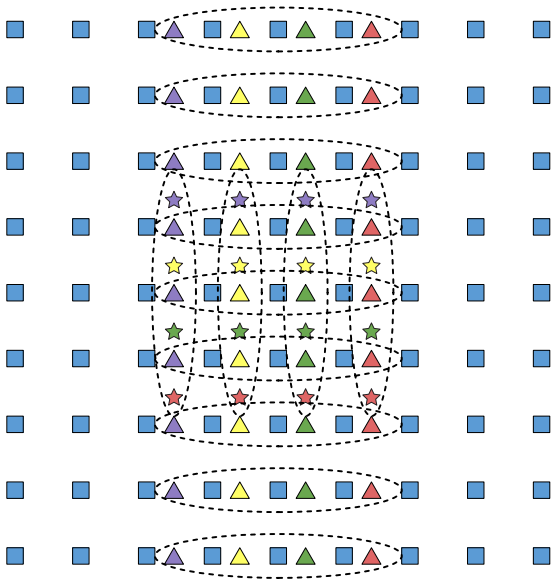

Figure 7 – Representation of diagonal interpolation of a 4x4 subblock. The blue squares are the integers, the triangles are horizontal, and the stars are diagonal samples.

process involves passing the input samples by a weighted average process, adopting different coefficients. The complete processing to interpolate one sample F_n can be seen in (3), where the six input samples are given by A_x, while the six coefficients are given by C_x. Also, besides the weighted sum of entries, offset and shift operations are performed (Chen; Ye; Kim, 2021). These operations are due to rounding properties and were performed to keep sample precision when consecutive interpolations.

$$F_n = (C_0A_{-2} + C_1A_{-1} + C_2A_0 + C_3A_1 + C_4A_2 + C_5A_3 + offset) >> shift \quad (3)$$

The fractional sample F_n is obtained by giving more weight to the input samples closer to the sample to be interpolated. In addition, the value varies according to the distance between the input and fractional samples (Chen; Ye; Kim, 2021). The Equation (4) represents an example of the use of Equation (3) for the interpolation filter F_8, which interpolates a sample in $1/2$ position. Table 1 shows the coefficients for the F_1 to F_{15} filters. Note that the F_9 to F_{15} coefficients are symmetric to the F_1 to F_7 coefficients. The F_{15} filter uses the same coefficients as the F_1 filter but inverts the input samples order. The same occurs between the F_2 and F_{14} filters, and so on.

$$F_8 = (3A_{-2} - 11A_{-1} + 40A_0 + 40A_1 - 11A_2 + 3A_3 + 32) >> 6 \quad (4)$$

Also can be seen in Table 1 the sum of the weights of each row of the filter table is equal to 64, which is used to give more weight to a sample. This is also used to calculate the shift and offset values in Equation (3). The shift value is used to divide the final sample interpolated by the row's total weight, so a division by 64 is the same as shifting by 6, and the offset value is equal to half the weight, so 32. For diagonal samples, the values of the offset and shift are not so simple to understand. However, the idea is to return the sample value to an 8-bit sample after the process. In the first interpolation, the shift value is 0, and the offset is -8192. For the second diagonal interpolation, the shift value is 12, and the offset is 526336. Both these values are generated by shifting internal precision values of the VVC.

2.2.2 Gradient-Based Iterative Algorithm

This section presents the Gradient-Based Iterative Algorithm (GBIA). Figure 8 presents the visual representation of the GBIA. As can be seen in Figure 8, GBIA starts with a set of CPMVs and generates a new set of CPMVs each iteration (up to five times) using a gradient-based algorithm which will be explained in this section, ending with a final set of CPMVs which have lower error and better encoding efficiency.

The Affine MC with the starting CPMVs will, most of the time, not generate the best block. To find the best block, VVC uses an algorithm based on gradient descent to min-

Table 1 – Affine Filter Coefficients.

	C_0	C_1	C_2	C_3	C_4	C_5
F_1	1	-3	63	4	-2	1
F_2	1	-5	62	8	-3	1
F_3	2	-8	60	13	-4	1
F_4	3	-10	58	17	-5	1
F_5	3	-11	52	26	-8	2
F_6	2	-9	47	31	-10	3
F_7	3	-11	45	34	-10	3
F_8	3	-11	40	40	-11	3
F_9	3	-10	34	45	-11	3
F_{10}	3	-10	31	47	-9	2
F_{11}	2	-8	26	52	-11	3
F_{12}	1	-5	17	58	-10	3
F_{13}	1	-4	13	60	-8	2
F_{14}	1	-3	8	62	-5	1
F_{15}	1	-2	4	63	-3	1

imize the error. This is an iterative algorithm that may be processed up to five times or until the generated CPMVs are the same as the previous iteration. The algorithm used in VVC utilizes kernel convolutions with Sobel Filters, which are utilized to generate Affine parameters. These Affine parameters are multiplied by themselves to generate an SLE that generates the ΔMV, which will be added to the current CPMVs.

Figure 9 presents the Sobel Filter in the X-direction and Y-direction, which are used in image processing for edge detection. This type of operation is called a kernel convolution. It is a process of adding each image element to its local neighbors multiplied by

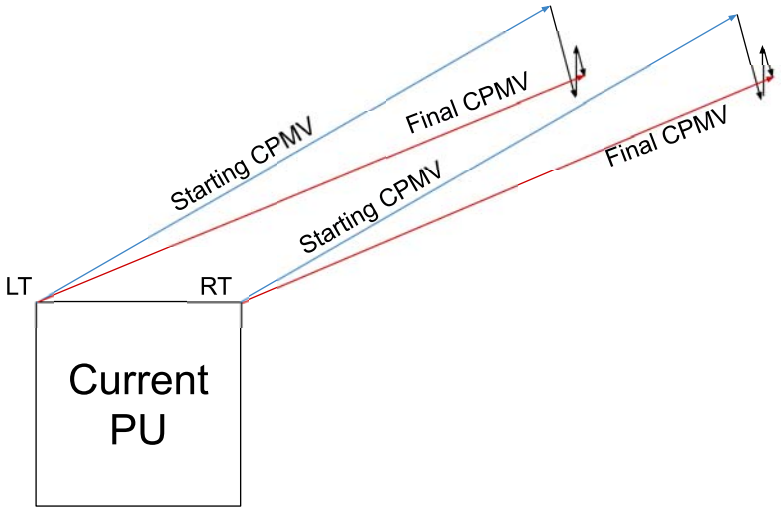

Figure 8 – Representation of the Gradient-Based Iterative Algorithm.

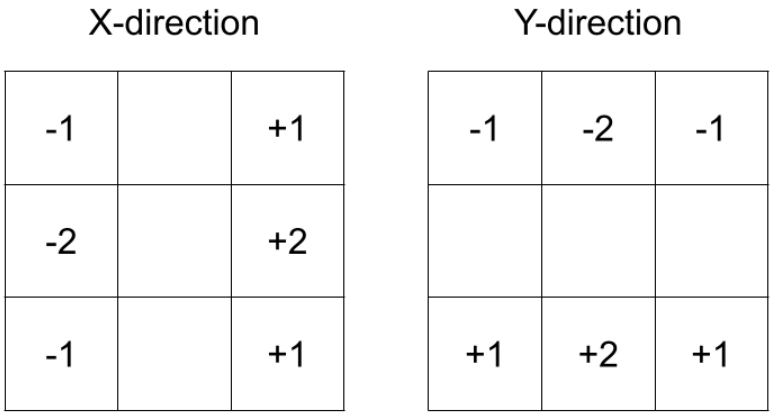

Figure 9 – The kernel Sobel Filter in the X- and Y-direction.

the filter value in that location. The Sobel Filters are not applied to the first and last row, as well as the first and last column, because the Sobel value utilizes samples around the target interpolated Sobel value. The first row is a copy of the second; the last row is a copy of the last but one; the first column is a copy of the second column, and the last column is a copy of the last but one. The Sobel filters are used to calculate the Affine coefficients. These Affine coefficients are different depending on the Affine prediction model being utilized.

The coefficients of the Affine 6-parameter prediction model can be seen in Equation (5). The SobelH and SobelV are the Sobel in X- and Y-directions, respectively. The cx and cy are the centers of the 4x4 subblock current being processed; this is calculated using the equation $cy = ((j \gg 2) \ll 2) + 2$ and $cx = ((i \gg 2) \ll 2) + 2$ where i and j are the vertical and horizontal position of the subblock inside the PU, these values may range from 2 to 126.

$$\begin{bmatrix} \mathrm{iC}_0 \\ \mathrm{iC}_1 \\ \mathrm{iC}_2 \\ \mathrm{iC}_3 \\ \mathrm{iC}_4 \\ \mathrm{iC}_5 \end{bmatrix} = \begin{bmatrix} \mathrm{SobelH} \\ cx * \mathrm{SobelH} \\ \mathrm{SobelV} \\ cx * \mathrm{SobelV} \\ cy * \mathrm{SobelH} \\ cy * \mathrm{SobelV} \end{bmatrix} \tag{5}$$

The coefficients of the Affine 4-parameter prediction model can be seen in Equation (6), and they can be seen as adding and subtracting the Affine 6-parameter coefficients. The iC_1 of Affine 4-parameter is Affine 6-parameter iC_1 added to iC_5 and The iC_3 of Affine 4-parameter is Affine 6-parameter iC_4 subtracted from iC_3.

$$\begin{bmatrix} \mathrm{iC}_0 \\ \mathrm{iC}_1 \\ \mathrm{iC}_2 \\ \mathrm{iC}_3 \end{bmatrix} = \begin{bmatrix} \mathrm{SobelH} \\ cx * \mathrm{SobelH} + cy * \mathrm{SobelV} \\ \mathrm{SobelV} \\ cy * \mathrm{SobelH} - cx * \mathrm{SobelV} \end{bmatrix} \tag{6}$$

The Affine coefficients are multiplied by each other, as seen in Equation (7). This is performed over the whole PU. The bH and bW are the PU height and width; The row and col choose the Affine coefficient and range from 0 to 5 if it is the 6-parameter mode or 0 to 3 if it is the 4-parameter model; the i and j are the current samples being processed in the PU, which ranges from 0 to $bH - 1$ for i and 0 to $bW - 1$ for j. These values are added up for the whole PU.

$$g_{[\mathrm{row,col}]} = \sum_{i=1}^{bH} \sum_{j=1}^{bW} \mathrm{iC}_{\mathrm{row}}[i][j] \times \mathrm{iC}_{\mathrm{col}}[i][j] \tag{7}$$

Table 2 shows the matrix resulting from the Equation (7) for the Affine 6-parameter prediction model where h and v are the horizontal and vertical Sobels. hx and vx are the horizontal and vertical Sobels multiplied by the position of the center of the subblock in x. hy and vy are the horizontal and vertical Sobel multiplied by the position of the center of the subblock in y. Table 3 presents the resulting matrix of the Equation (7) for the Affine 4-parameter prediction mode. The values in Table 2 can be manipulated algebraically to output the values in Table 3.

There is an additional column generated by multiplying the error by the Affine coefficients, as can be seen in Equation (8).

$$e_{[\mathrm{row}]} = \sum_{i=1}^{bH} \sum_{j=1}^{bW} (\mathrm{iC}_{\mathrm{n}}[\mathrm{row}] \times error[i,j]) \ll 3 \tag{8}$$

The error utilized is calculated using the original samples minus the predicted samples, as can be seen in Equation (9), where O is the original samples and P is the predicted samples.

Table 2 – SLE of 6-parameters.

	h	hx	v	vx	hy	vy
h	h^2	$h \cdot hx$	$h \cdot v$	$h \cdot vx$	$h \cdot hy$	$h \cdot vy$
hx		hx^2	$hx \cdot v$	$hx \cdot vx$	$hx \cdot hy$	$hx \cdot vy$
v			v^2	$v \cdot vx$	$v \cdot hy$	$v \cdot vy$
vx				vx^2	$vx \cdot hy$	$vx \cdot vy$
hy					hy^2	$hy \cdot vy$
vy						vy^2

Table 3 – SLE of 4-parameters.

	h	$hx + vy$	v	$hy - vx$
h	h^2	$h \cdot hx + h \cdot vy$	$h \cdot v$	$h \cdot hy - h \cdot vx$
$hx+vy$		$hx^2 + 2 \cdot hx \cdot vy + vy^2$	$hx \cdot v + v \cdot vy$	$hx \cdot hy - hx \cdot vx + vy \cdot hy - vy \cdot vx$
v			v^2	$v \cdot hy - v \cdot vx$
$hy-vx$				$hy^2 - 2 \cdot hy \cdot vx + vx^2$

$$error[i][j] = O[i][j] - P[i][j] \tag{9}$$

The coefficient matrix and the error column are joined, as in Equation (10). This System of Linear Equation (SLE) can be 4x5 or 6x7, depending on the Affine prediction model being calculated. After solving this SLE, 4 or 6 Affine parameters are generated, one for each row.

$$\left[\begin{array}{cccc|c} g_{[1,1]} & g_{[1,2]} & \cdots & g_{[1,n]} & e_{[1]} \\ g_{[2,1]} & g_{[2,2]} & \cdots & g_{[2,n]} & e_{[2]} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ g_{[n,1]} & g_{[n,2]} & \cdots & g_{[n,n]} & e_{[n]} \end{array}\right] \tag{10}$$

The four or six values of the solution to the SLE are the Affine parameters that go through a final step in which the new CPMVs are generated. The equations (11), (12), and (13) present this final step. The AffineP is one of the four or six Affine parameters, bW and bH are the PU width and height, ΔMV. These values are clipped to 14 bits (8.191 to -8.192).

The Equation (11) presents the ΔMV, which are common for both prediction models.

$$\begin{bmatrix} \delta \mathrm{MV}_0 \\ \delta \mathrm{MV}_2 \end{bmatrix} = \begin{bmatrix} \mathrm{AffineP}_0 \\ \mathrm{AffineP}_2 \end{bmatrix} \tag{11}$$

The Equation (12) presents the ΔMV, which is used in the Affine 4-parameter prediction model that uses two CPMVs.

$$\begin{bmatrix} \delta \mathrm{MV}_1 \\ \delta \mathrm{MV}_3 \end{bmatrix} = \begin{bmatrix} \mathrm{AffineP}_1 \times \mathrm{bW} + \mathrm{AffineP}_0 \\ -\mathrm{AffineP}_3 \times \mathrm{bW} + \mathrm{AffineP}_2 \end{bmatrix} \tag{12}$$

The Equation (13) presents the ΔMV, which is used in the Affine 6-parameter prediction model that uses three CPMVs

$$\begin{bmatrix} \delta\mathrm{MV}_1 \\ \delta\mathrm{MV}_3 \\ \delta\mathrm{MV}_4 \\ \delta\mathrm{MV}_5 \end{bmatrix} = \begin{bmatrix} \mathrm{AffineP}_1 \times \mathrm{bW} + \mathrm{AffineP}_0 \\ \mathrm{AffineP}_3 \times \mathrm{bW} + \mathrm{AffineP}_2 \\ \mathrm{AffineP}_4 \times \mathrm{bH} + \mathrm{AffineP}_0 \\ \mathrm{AffineP}_5 \times \mathrm{bH} + \mathrm{AffineP}_2 \end{bmatrix} \tag{13}$$

The δMV calculated in the previous step is then used to calculate the ΔMV that will be added up to the CPMVs currently being evaluated, generating the next set of CPMVs. This can be seen in Equation (14). Where the δMV are the values calculated in the previous step, the bW and bH are the PU width and height, and the sign returns 1 if the number is positive or -1 if the number is negative. The ΔMV calculated is clipped to $1/16$ precision and then added to the current CPMVs, generating the next set of CPMVs.

$$\begin{bmatrix} \Delta\mathrm{MV}_0 \\ \Delta\mathrm{MV}_1 \\ \Delta\mathrm{MV}_2 \\ \Delta\mathrm{MV}_3 \\ \Delta\mathrm{MV}_4 \\ \Delta\mathrm{MV}_5 \end{bmatrix} = \begin{bmatrix} (4 \times \delta\mathrm{MV}_0 + 0.5 \times sign(\delta\mathrm{MV}_0)) \ll 2 \\ (4 \times \delta\mathrm{MV}_1 + 0.5 \times sign(\delta\mathrm{MV}_1)) \ll 2 \\ (4 \times \delta\mathrm{MV}_2 + 0.5 \times sign(\delta\mathrm{MV}_2)) \ll 2 \\ (4 \times \delta\mathrm{MV}_3 + 0.5 \times sign(\delta\mathrm{MV}_3)) \ll 2 \\ (4 \times \delta\mathrm{MV}_4 + 0.5 \times sign(\delta\mathrm{MV}_4)) \ll 2 \\ (4 \times \delta\mathrm{MV}_5 + 0.5 \times sign(\delta\mathrm{MV}_5)) \ll 2 \end{bmatrix} \tag{14}$$

With the new set of the CPMVs calculated, if they are different than the set evaluated, the PU is then Affine MC again, and this process repeats. The GBIA described in this section is repeated up to five times or until all the ΔMV calculated are zero, minimizing the error of the predicted block. Then, after this step, the BMIA algorithm is applied over the PU predicted with the last CPMVs calculated in this step, which are the best CPMVs currently found.

2.2.3 Block Matching Iterative Algorithm

The BMIA is an algorithm that is applied on the PU with the best CPMVs calculated in the GBIA step. However, this algorithm is much simpler compared to the GBIA. The CPMVs in this step are evaluated by adding ± 1 to the CPMVs, the block is AMC, and the RD-Cost of the predicted block is calculated. If there are CPMVs with lower RD-Cost, they become the new origin, and the process repeats three times.

The BMIA is divided into two exploration steps in which ± 1 are added to the CPMVs, one horizontal/vertical and one diagonal. Figure 10 is a visual representation of the horizontal/vertical exploration and the diagonal exploration of the Block Matching Algorithm. As can be seen in Figure 10, BMIA starts with a set of CPMVs and evaluates all neighboring positions of the PU and the position with the smaller RD-Cost is chosen as the final set of CPMVs. Then, a diagonal exploration is done; if any diagonal position has a lower RD-Cost, it becomes the new origin. This repeats three times or

until it does not find any neighboring position with lower RD-Cost.

The BMIA has three rounds of exploration performed for each CPMV, which means that each CPMV can do three horizontal/vertical explorations and three diagonal explorations. This lets the CPMV explore up to radius six around its starting position after the three rounds of exploration. This can be seen in Figure 11, where the black square is the starting position of the CPMV and the blue are all possible ending positions to the CPMV. This heuristic updates the CPMV with the local best neighbor, which may not lead to the best CPMV. However, this algorithm lets the CPMV access 113 possible positions with only 24 iterations. There are 168 positions in a block 13x13 (without counting the origin). The CPMV could access 113 of these positions, or 67% of the search range, with only 24 iterations, or 14% of the iterations, if it were performed an exhaustive search.

For each iteration of the BMIA, an RD-Cost calculation is used to evaluate the new CPMV. If the CPMV evaluation leads to a predicted block with the RD-Cost smaller than the current best-predicted block, the CPMV with the smaller RD-Cost replaces the current CPMV. This step was unnecessary for the GBIA because the error was used to calculate the next set of CPMVs.

2.3 Related Works

There are several works in the literature proposing hardware architectures for different video coding standards. However, very few works in the literature present hardware designs for the Affine ME of the VVC standard. Taranto (2022) presents a simplified algorithm and hardware design for the Affine ME on the encoder side. Sheng et al. (2024) presents a SLE solver for the Affine ME.

To achieve a more complete comparison of the hardware designs in the literature, hardware designs for the interpolation filters of the FME were also selected. However, it should be noted that only values related to hardware results, such as area and power

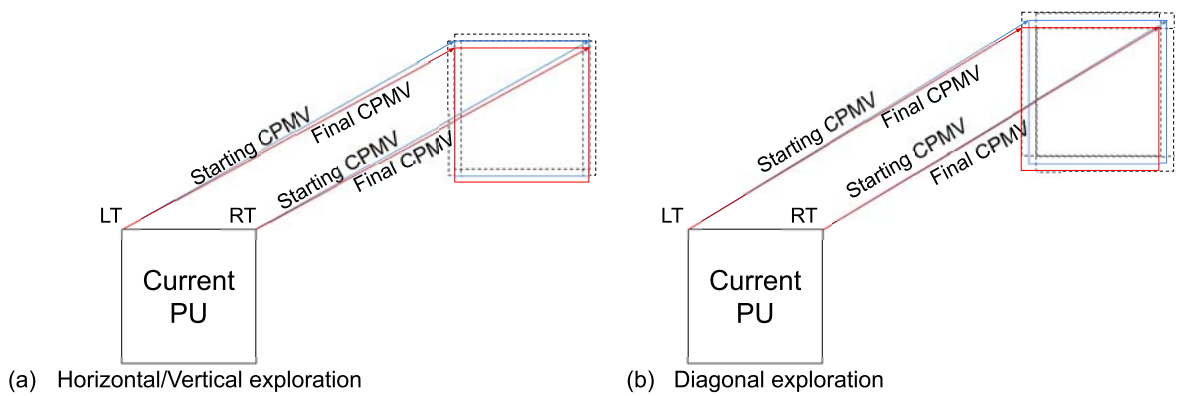

Figure 10 – Representation of the Block-Matching Iterative Algorithm.

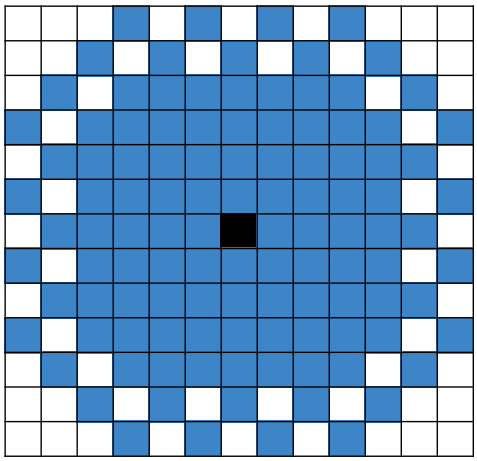

Figure 11 – Representation of all the BMIA possible positions explored after three rounds for **one** CPMV.

dissipation, are presented and will be compared as the Affine MC and the FME interpolation filters work similarly. Other results, such as coding efficiency for these works, are not taken into consideration as they are not comparable. These works are Azgin et al. (2018); Canmert; Kalali; Hamzaoglu (2018); Azgin; Kalali; Hamzaoglu (2020); Mahdavi; Azgin; Hamzaoglu (2021); Silva et al. (2021).

The work in Taranto (2022) is the only in current literature presenting a dedicated hardware design for the Affine ME of the VVC standard. This work, however, made a couple of simplifications to the algorithm. The first of the simplifications that will be discussed is the removal of fractional samples. This removes the need for interpolation filters and reduces the computational complexity because a lot of the computational complexity of the interpolation process is processed by intermediate samples. The second simplification discussed is that all PU is broken down to 16x16 blocks and only the corner 4x4 subblocks are processed, processing four subblocks every 16, this reduces the overall computational complexity by 3/4. Also, the architecture only supports PUs in sizes 16x16 up to 64x64. These simplifications are significant changes to the algorithm and lead to an increase in the BD-Rate, which is not provided in the work.

The works Azgin et al. (2018); Canmert; Kalali; Hamzaoglu (2018); Azgin; Kalali; Hamzaoglu (2020); Mahdavi; Azgin; Hamzaoglu (2021); Silva et al. (2021) presents hardware designs for the interpolations filters for the FME. Canmert; Kalali; Hamzaoglu (2018) use offsets to reduce the number of additions and multiplications. The hardware

has eight 8-tap filters, and they present results for both FPGA and ASIC 90nm. Azgin et al. (2018) uses eight 15-tap filters with reconfigurable paths to reduce the number of operations, and the results are presented for both FPGA and ASIC 90nm. Azgin; Kalali; Hamzaoglu (2020); Mahdavi; Azgin; Hamzaoglu (2021) both use 15 4-tap approximate filters, which have low area and power dissipation, and their results are only for FPGA. Silva et al. (2021) presents 15 6-tap approximate filters and MCM approach, which both lead to low area and power dissipation, their present results for ASIC 65nm.

3 SOFTWARE EVALUATIONS

This chapter presents the simulations performed on the reference software VTM (VVC Test Model) (Chen; Ye; Kim, 2021). The simulations were performed to evaluate many different constraints:

- Limiting the number of reference frames.
- Limiting PU sizes.
- Limiting the number of Affine MC iterations.
- Combination of constraints.

The number of reference frames increases the number of starting CPMVs that need to be evaluated in the GBIA algorithm. The Affine ME of the VVC standard uses PU sizes from 16x16 to 128x128, so limiting the processed sizes has a great impact because Affine MC is processed on 4x4 subblocks. Limiting the number of iterations in the GBIA and BMIA reduces the number of times the Affine MC is processed. Finally, this chapter presents an analysis of the combination of these constraints regarding the encoding efficiency losses.

3.1 Experimental Setup

The simulations presented use the VVC Test Model (VTM), which is the the VVC reference software. The version of the VTM used on all simulations is version 16.2, and the video sequences utilized are the 26 sequences presented in the Common Test Conditions (CTC) (Bossen et al., 2020). Also, the Quantization Parameters (QP) were recommended in the CTC (22, 27, 32, and 37) (Bossen et al., 2020). The simulation uses the default configuration for the sequences, and the temporal profile used is low delay P. The other temporal profiles were not simulated because this simulation is performed focusing on the use in low-power real-time devices, in which low delay is optimal because the future frames are not involved. The setup used for simulation is a Linux server with a CPU XEON Gold 5118 2.30GHz with AMD DIMM 56GB 2400MHz of RAM.

3.2 Hardware-Oriented Constraints Evaluation

This section presents the evaluations for the simulations of the constraints utilized, which are:

- Reduction of the reference frames from **four** to **one**.
- PU sizes limited to only **Quadratic** (128x128, 64x64, 32x32, and 16x16) sizes or only 16x16.
- Reduction of iterations to only **GBIA** (which Affine MC up to 5 times, see section 2.2.2) and to only **Affine MC** on the starting CPMVs (GBIA and BMIA disabled).

Then, the combination of these constraints is also evaluated:

- Limiting to only GBIA (see section 2.2.2) and Quadratic PUs
- GBIA and only 16x16 PUs

The reason that the combination only takes into consideration the GBIA is that BMIA significantly increases the overall computational complexity of the algorithm (see section 2.2.3). In contrast, the GBIA iterates very few times and can quickly converge to a minimum error because of the gradient descent (see section 2.2.2). The constraints being changed in the combined evaluation are processing only quadratic PUs and processing only PUs of 16×16.

The recommended test video sequences are divided by classes: A1 and A2 are composed of UHD 4K video sequences; B is composed of FHD 1080p; C, D, and E are composed of low-resolution videos; And F is composed of synthetic videos, of varying resolutions. Also, the results presented in this section are related to the total per frame. The reason for presenting the results per frame is that the framerate of the sequences is variable, with some sequences being 30 fps while others are 50 or 60 fps. Since the data is presented per frame calculating the necessary frequency for different throughputs is straightforward, multiply the data of the desired class by the fps.

3.2.1 Results and BD-Rates of Reference Frames Constraints

This section presents data from the Affine ME of VVC, which was extracted from the VTM using the previously mentioned configuration. This section first presents the data for the Affine with four reference frames and then for one reference frame.

The results presented in Table 4 are *ME*, which is how many times the Affine ME is tested in the ME process; *AME*, which means how many times the Affine ME, 4-parameter or 6-parameter, is processed; *Affine-4* and *Affine-6*, which are the number of each model is used; *GBIA* and *BMIA*, which are how many iterations for each algorithm.

Table 4 presents both the average number of occurrences per frame and the standard deviation results considering each class of the recommended test video sequences.

For the most demanding UHD 4K class (A2), as can be seen in Table 4, the Affine ME is to be processed around 30% of the time it the ME is executed. The Affine 6-parameter is processed 23% of the times the 4-parameter is processed. The GBIA processes the Affine MC an average of 5 times per AME. The BMIA processes the Affine MC an average of 2 times. Another important point to pay attention to, especially when calculating the frequency necessary for the processing Affine, is that the standard deviation is as big as the average.

Also, Table 4 shows that Affine computational complexity is extremely high, especially for 4K videos. Ignoring the standard deviation, which is not recommended, and adding the *GBIA* and *BMIA* columns together for the less demanding 4K Class (A1), the result is 5.2M Affine MC per frame, and at 60 fps, this gets to 312M Affine MC per second. This calculation shows that even if the Affine MC of a PU of any size is processed in one clock cycle, which would require an enormous area for parallelism and power dissipation, it would still need 312MHz of frequency.

Table 5 presents the average amount of processing performed by the AME for each PU size considering one frame of each class of the recommended test video sequences. In Table 5, *PU* presents the number of times a PU of that size is processed, *GB* is the number of iterations the GBIA, and *BM* is the number of iterations the BMIA. Table 5 shows that the bigger PU sizes, such as 128x128, use the BMIA more on average than the smaller PUs, such as 16x16.

Table 5 can also be used to show the high bandwidth of the AME. For the UHD 4K Class A1, using the Equation 15, where $N_{samples}$ are the number of samples interpo-

Table 4 – Results of the Affine Motion Estimation in the VVC for **four** Reference Frames.

Class		ME	AME	AFFINE-4	AFFINE-6	GBIA	BMIA
A1	**Avg**	2.49M	720k	589k	131k	3.73M	1.47M
	Std	2.88M	505k	407k	99.4k	2.59M	1.38M
A2	**Avg**	2.75M	811k	659k	152k	4.09M	2.3M
	Std	2.65M	474k	380k	94.2k	2.35M	1.97M
B	**Avg**	708k	190k	154k	36.1k	944k	573k
	Std	705k	108k	86.5k	21.6k	531k	441k
C	**Avg**	197k	41.5k	33.5k	7.9k	210k	134k
	Std	149k	14.1k	11.3k	2.86k	72.7k	57.3k
D	**Avg**	55.2k	8.82k	7.12k	1.69k	44.1k	34.5k
	Std	41.8k	2.56k	2.09k	482	13k	15.5k
E	**Avg**	82.4k	41.9k	33.7k	8.16k	189k	93.7k
	Std	101k	31k	24.9k	6.14k	141k	84.1k
F	**Avg**	235k	71.6k	57.9k	13.7k	330k	177k
	Std	336k	74.5k	60.1k	14.4k	350k	211k

lated per frame, N_{PU} is the number of PU evaluated, the PU_h and PU_w are PU width and height. The number of samples interpolated per frame for PU 16x16 is 46.3M. Calculating this for all PU sizes, the result is 629M samples of input. This calculation does not take into consideration that two PUs may overlap samples or even be the same PU with different MVs, which reduces the number of samples necessary to be stored. However, all PUs will generate different output samples, either because of the different CPMVs or input samples. So, without taking into consideration the GBIA and BMIA, the amount of data that needs to be processed is extremely high.

$$N_{samples} = N_{PU} * PU_h * PU_w \tag{15}$$

As previously mentioned, the computational complexity and data bandwidth necessary to process the AME in the default configurations is extremely high. So, to reduce this processing demand when targeting real-time processing, especially for battery-powered devices, the number of reference frames was reduced to only one, which is a strategy commonly used in hardware design. Table 6 presents the results considering

Table 5 – Average Processing of the Affine Motion Estimation divided by PU size for four Reference Frames.

Class		128 x 128	128 x 64	64 x 128	64 x 64	64 x 32	32 x 64	64 x 16	16 x 64	32 x 32	32 x 16	16 x 32	16 x 16
A1	PU	2.38k	3.04k	3.04k	13.6k	30.7k	31.1k	40.7k	41.8k	69.5k	151k	152k	181k
	GB	11k	14.6k	14.6k	66.4k	154k	156k	209k	215k	352k	785k	792k	957k
	BM	17.3k	14.8k	13.5k	53.6k	98.2k	95.6k	91.1k	85.2k	146k	313k	305k	239k
A2	PU	2.39k	2.25k	2.25k	12.1k	26.7k	26.9k	43.4k	43.7k	80k	177k	173k	221k
	GB	11.4k	10.8k	10.7k	58.6k	132k	132k	219k	219k	395k	895k	871k	1.14M
	BM	21.2k	13.1k	11.6k	65.1k	114k	102k	138k	111k	252k	574k	483k	416k
B	PU	598	517	517	2.83k	6.04k	5.95k	10.2k	10k	18.3k	38.9k	38.4k	57.9k
	GB	2.69k	2.36k	2.32k	13.1k	28.9k	28.3k	49.9k	48.6k	88.6k	195k	191k	293k
	BM	4.27k	2.42k	2.2k	13.3k	24.1k	21.8k	31k	25.2k	58.8k	135k	121k	134k
C	PU	89.8	22.1	22.1	471	855	836	1.74k	1.71k	3.65k	8.08k	8.12k	15.9k
	GB	433	104	102	2.28k	4.2k	4.03k	8.72k	8.33k	18.1k	41.4k	41.1k	81.5k
	BM	624	105	83.3	2.7k	4.03k	3.18k	6.63k	4.36k	13.8k	30.7k	26k	42k
D	PU	15	3.1	3.1	91.4	144	137	323	295	716	1.59k	1.53k	3.97k
	GB	73.8	14.8	14.5	447	709	665	1.62k	1.44k	3.53k	8k	7.58k	20k
	BM	124	17.6	14.6	617	812	621	1.52k	914	3.34k	7.3k	5.87k	13.3k
E	PU	249	379	381	1.21k	2.24k	2.31k	3.45k	3.55k	5.17k	6.87k	7.14k	8.91k
	GB	959	1.51k	1.5k	5.04k	9.61k	9.73k	15.6k	15.5k	22.7k	31.8k	32.7k	42.4k
	BM	668	854	688	2.87k	5.47k	4.74k	6.17k	4.58k	9.92k	21k	18.9k	17.8k
F	PU	314	285	286	1.35k	2.65k	2.53k	4.57k	4.28k	7.59k	13.7k	13k	21k
	GB	1.06k	906	898	5.09k	10.7k	10.1k	19.7k	18.1k	32.9k	65.7k	61.8k	103k
	BM	1.11k	537	526	4k	6.5k	5.89k	8.97k	7.33k	17.7k	41.1k	36.8k	46.1k

Table 6 – Results of the Affine Motion Estimation in the VVC for **one** Reference Frames.

Class		Affine-4	Affine-6	GBIA	BMIA	Hor4	Hor6	Ver4	Ver6	Diag4	Diag6
A1	**Avg**	150k	129k	1.35M	1.45M	7.59M	8.56M	5.43M	7.43M	59.3M	94.7M
	Std	102k	99.2k	946k	1.34M	3.08M	5.32M	2.37M	4.76M	28.3M	64.1M
A2	**Avg**	166k	149k	1.46M	2.23M	7.3M	10.6M	5.3M	9.08M	58.8M	124M
	Std	95.6k	94.6k	863k	1.92M	2.65M	5.19M	1.75M	4.54M	18.7M	62.2M
B	**Avg**	39.2k	36.2k	352k	570k	2.14M	2.74M	1.38M	2.29M	13.1M	28M
	Std	22k	22k	200k	430k	1.15M	1.38M	599k	1.17M	4.73M	14.3M
C	**Avg**	8.4k	7.73k	77.8k	131k	560k	654k	273k	447k	2.28M	5.09M
	Std	2.74k	2.78k	26.6k	55.2k	237k	242k	129k	163k	641k	1.68M
D	**Avg**	1.83k	1.7k	17.1k	32.2k	79.4k	114k	73.3k	123k	475k	1.14M
	Std	567	535	5.32k	15.7k	57.4k	70.3k	37.7k	44.6k	152k	535k
E	**Avg**	8.61k	8.19k	71.6k	89.8k	1.08M	682k	517k	532k	3.14M	4.87M
	Std	6.51k	6.38k	54.7k	81.3k	415k	374k	228k	290k	1.37M	2.76M
F	**Avg**	14.7k	13.5k	124k	175k	1.49M	940k	556k	745k	4.11M	7.74M
	Std	15.4k	14.5k	134k	205k	1.1M	918k	472k	749k	4.16M	8.43M

the constraint of only using one reference frame.

Table 6 had the column ME removed because the focus of this dissertation is on the AME algorithm and not on how the ME evaluates if the AME is processed. The column AME was also removed because it is the sum of the columns Affine-4 e and Affine-6, making it redundant. Table 6, however, added six new columns, and these columns are the number of 4x4 subblocks interpolated. Columns Hor4 and Hor6 are the horizontal interpolations for the Affine 4- and 6-parameter, respectively. Similarly, the columns Ver4 and Ver6 are the vertical interpolations for Affine 4- and 6-parameter. Lastly, the columns Diag4 and Diag6 are the diagonal interpolations for Affine 4- and 6-parameter.

As can be seen for Class A1, The Affine 6-parameter is used 86% of the time the Affine 4-parameter is used, a significant increase compared with four reference frames, which was 28%. The average number of GBIA iterations is 4.8 for one reference, a slight decrease when compared to one reference frame, which was 5.2. While the BMIA average increased to 5.17, in contrast to 2 when considering four reference frames, this is a significant increase.

The Affine MC, as explained in section 2.2.1, is performed on 4x4 subblocks, and if the interpolation is horizontal/vertical, the subblock MC is performed without requiring generating intermediate samples. However, if the interpolation is diagonal, 9x4 intermediate samples are required before interpolating the 4x4 subblock. These additional intermediate samples are needed to be taken into consideration when calculating the throughput.

As can be seen for Class A1, by dividing the number of interpolations per Affine model, the Affine 4-parameter has an average of 50 horizontal subblocks, 36 vertical

Table 7 – Average Processing of the Affine Motion Estimation divided by PU size for **one** Reference Frames.

Class		128 x 128	128 x 64	64 x 128	64 x 64	64 x 32	32 x 64	64 x 16	16 x 64	32 x 32	32 x 16	16 x 32	16 x 16
A1	PU	952	1.23k	1.23k	5.46k	12.3k	12.5k	15.9k	16.5k	27.3k	58.9k	60k	67.6k
	GB	3.9k	5.31k	5.31k	24.2k	56.7k	57.8k	76.1k	79.4k	129k	287k	291k	337k
	BM	16.2k	14.1k	12.8k	50.5k	94k	90.9k	88.7k	83.2k	144k	310k	301k	248k
A2	PU	960	907	908	4.83k	10.6k	10.7k	16.9k	17.2k	31.4k	69.1k	68k	83.4k
	GB	3.89k	3.92k	3.93k	20.9k	47.4k	47.7k	78.4k	79.8k	142k	320k	314k	399k
	BM	20.1k	11.8k	10.5k	61k	106k	96.5k	130k	108k	243k	557k	473k	414k
B	PU	240	206	206	1.13k	2.4k	2.37k	4.04k	4k	7.28k	15.5k	15.3k	22.8k
	GB	973	872	864	4.82k	10.7k	10.5k	18.6k	18.3k	33k	72.5k	71.5k	109k
	BM	4.12k	2.28k	2.11k	12.9k	23.2k	21.2k	30.1k	25.2k	58.2k	133k	121k	137k
C	PU	36	8.91	8.91	188	338	331	683	675	1.42k	3.1k	3.14k	6.19k
	GB	163	39.2	38.9	854	1.57k	1.51k	3.26k	3.14k	6.7k	15.1k	15.1k	30.4k
	BM	618	99.5	82.2	2.57k	3.75k	3.04k	6.12k	4.3k	13.1k	29.5k	25.4k	42.4k
D	PU	6	1.28	1.28	36.5	58.4	55.9	131	120	286	634	611	1.59k
	GB	27.9	6.03	6	171	279	262	638	567	1.36k	3.09k	2.93k	7.72k
	BM	121	14.4	12.7	570	711	564	1.32k	867	3k	6.67k	5.53k	12.8k
E	PU	100	152	153	484	897	924	1.38k	1.43k	2.08k	2.76k	2.9k	3.55k
	GB	354	576	572	1.92k	3.67k	3.74k	5.96k	6.01k	8.7k	11.9k	12.4k	15.8k
	BM	643	792	652	2.7k	5.1k	4.51k	5.75k	4.53k	9.43k	20k	18.3k	17.5k
F	PU	124	113	114	537	1.06k	1.01k	1.82k	1.72k	3k	5.39k	5.16k	8.13k
	GB	399	343	344	1.93k	4.1k	3.88k	7.52k	7k	12.5k	24.6k	23.5k	38.2k
	BM	1.08k	482	472	3.85k	6.23k	5.64k	8.59k	7.2k	17.3k	40.2k	36.6k	46.9k

subblocks, and 631 diagonal subblocks, and the Affine 6-parameter has an average of 66 horizontal subblocks, 57 vertical subblocks, and 734 diagonal subblocks. There are 7.3 diagonal subblocks per horizontal/vertical subblock interpolated for the Affine 4-parameter and 6 diagonal subblocks for every horizontal/vertical subblock for the 6-parameter.

Table 7 presents the average amount of processing performed by the AME for each PU size. In Table 7, there is a significant decrease in the number of iterations, as expected, when reducing from four to one reference frame. The results in Table 7 will be used to calculate further constraints in the following sections, such as reducing the number of iterations and PU sizes. It can also be seen in Table 7 that for UHD 4K Class A1, the reduction in the number PU evaluated for each PU is not a reduction of 4 times but a reduction 2.7 times on average, and this is because the Affine 6-parameter is evaluated relatively more for one reference frame.

The impact on encoding efficiency of the AME is evaluated using the Bjontegaard Delta Rate (BD-Rate) (BJONTEGAARD, 2001). The BD-Rate quantifies the bitrate variation required to achieve an equivalent objective image quality. Reducing from four

to one reference frame provides an average BD-Rate increase of 15.81%, considering all classes, and of 8.22% for the most demanding classes (A1, A2, and B). The increase per class is 2.36% for A1, 4.55% for A2, 13.93% for B, 21.93% for C, 31.35% for D, 14.46% for E, and 16.01% for F.

The reduction of reference frames impacts all the ME algorithms and not only the AME, and this is why the impacts on encoding efficiency in this section are so significant. However, it is a very common strategy used to reduce the overall complexity of the algorithm when targeting real-time power-constrained devices. As already presented, the computational complexity and bandwidth necessary for the AME are unachievable for real-time power-constrained devices. From this moment onward, the baseline of all comparisons will be the one reference frame.

3.2.2 BD-Rate of PU Size Constraints

This section presents an evaluation of the encoding efficiency of different PU size constraints using only one reference frame:

- Process only quadratic PUs (16x16, 32x32, 64x64, 128x128)
- Process only 16x16 PUs

Table 8 presents the average BD-Rate for all video classes and the overall average in the columns *Quad* and *16x16* for the Quadratic size constraint and the only using 16x16 constraint. The objective of these evaluations is the reduction of subblock interpolation. The Affine MC is processed on 4x4 subblocks, so removing some PU sizes has a low impact on encoding efficiency, but depending on how big the PU is, it might have a huge impact on the reduction of the number of subblock interpolations.

The number of PU tested in the AME when using only quadratic PU is reduced to 36% of the total number. The same reduction to 36% can be seen for the total number of GBIA iterations, while for the BMIA, the reduction is to 31% of the total number of iterations. This reduction is significant, especially when the increase in BD-Rate is 0.09% for Class A1, as can be seen in Table 8. This huge reduction of iterations and the number of samples processed can be seen in all classes with similar values. The average BD-Rate increase for all classes with the quadratic PU size limitation is 0.19%.

The evaluation of limiting to 16x16 PU is also presented in Table 8. The PU tested in Class A1 is reduced to 24%, and the GBIA is also reduced to 25%, while the BMIA is reduced to 17%. Similar results were obtained with the quadratic PUs, with the main difference being the BMIA, which is used less when the PU size is smaller. This constraint enormously impacts the computational complexity required and the number of samples processed with an increase in BD-Rate of 0.43% for the Class A1 and 0.76% for all classes, which can be seen in Table 8.

Table 8 – BD-Rate of the PU constraints evaluated.

Class	Quad	16x16
A1	0.0900	0.4368
A2	0.3443	2.0826
B	0.1654	0.6307
C	0.1151	0.2126
D	0.0458	0.1923
E	0.3200	0.5285
F	0.3242	1.5160
Avg.	0.1935	0.7685

When comparing the support of only quadratic PUs against only PUs of 16×16, the latter has a BD-Rate increase of four times (0.19% to 0.76%). Even though the increase in BD-Rate can be considered high, there is a significant reduction in overall data needed, and there is a significant reduction in the number of iterations required, allowing more efficient hardware design.

3.2.3 BD-Rate of Iteration Constraints

This section presents an evaluation of reducing the number of iterations in the AME algorithm using only one reference frame:

- Processing only AMC (Skipping both GBIA and BMIA)
- Processing the unrestricted GBIA (see section 2.2.2).

Table 9 presents the average BD-Rate of the video classes separated and the average for all videos. The columns *GBIA* and *AMC* present the BD-Rate increase when skipping BMIA and the BD-Rate increase when skipping both BMIA and GBIA, respectively.

Table 9 – BD-Rate of the Iteration constraints evaluated.

Class	GBIA	AMC
A1	0.0567	0.4242
A2	0.0900	1.9431
B	0.0826	0.6368
C	0.0629	0.3366
D	-0.0033	0.3699
E	0.1635	0.6123
F	0.0944	2.4687
Avg.	0.0754	0.9547

For the UHD 4K class A1, the constraint of only processing GBIA leads to a reduction of 48% of the total number of iterations. This constraint has a BD-Rate increase of 0.05% for Class A1 and 0.07% for the average of all video sequences, as can be seen in Table 9, which is small compared to its overall reduction in computational complexity and necessary data.

The limitation of only processing the AMC on the starting MV reduces the number of iterations and necessary data to 9% for the UHD 4K Class A1. When skipping both GBIA and BMIA it leads to the greatest reduction in overall complexity and data, however, this also leads to the highest increase in BD-Rate of a single limitation. This limitation has a BD-Rate increase of 0.42% for UHD 4K Class A1 and 0.95% for the average of all video sequences, as can be seen in Table 9.

Comparing both GBIA and AMC is straightforward because AMC has a significant BD-Rate increase, 0.07% to 0.95%, compared to GBIA, with a significant reduction in data and iterations, 48% of the total number of iterations to 9% of the total number of iterations.

3.2.4 BD-Rate of Proposed Approach for the AME Hardware Design

This section presents evaluation results for combined constraints: GBIA with only quadratic PU sizes; and GBIA with only PU size of 16×16. GBIA was selected for both constraints due to its minimal impact on BD-Rate while offering a notable reduction in both iterations and data requirements. The rationale behind testing quadratic PU sizes, along with 16x16, is that quadratic PU sizes, like GBIA, have a low impact on BD-Rate, contributing to a significant reduction in overall complexity and required data. The choice to support only 16x16 PU was made because it substantially reduces algorithm iterations without causing the largest increase in BD-Rate. Both of these constraints can be used to design more efficient hardware implementations.

Table 10 presents the average BD-Rate increase for all classes and for all videos. The column *GBIA+Quad* presents the BD-Rate increase for the constraint of using only quadratic sizes and only processing GBIA. The *GBIA+16x16* column presents the BD-Rate increase for the same constraint but supports only PU sizes of 16x16.

For the UHD 4K Class A1, the combined constraints of supporting only quadratic sizes and only processing GBIA leads to a reduction to 17% of the total number of iterations of the AME algorithm. Class A1 also presents a BD-Rate increase of 0.10%, and the average BD-Rate increase for all classes is 0.27%, which is slightly higher than the sum of each BD-Rate constraint separately.

For the UHD 4K Class A1, the combined constraints of supporting only PU of size 16x16 and only processing GBIA leads to a reduction in the number of iterations to 12% with a BD-Rate increase of 0.46%. The average BD-Rate increase for all classes is 0.80%, which is lower than the sum of the BD-Rate of the constraints separately.

Table 10 – BD-Rate of the proposed combined constraints evaluated.

Class	GBIA+Quad	GBIA+16x16
A1	0.1060	0.4617
A2	0.4745	2.0409
B	0.2618	0.6547
C	0.1509	0.2553
D	0.1407	0.2587
E	0.2229	0.5690
F	0.5479	1.5874
Avg.	0.2722	0.8036

Both constraints reduce the number of iterations significantly while having a low BD-Rate increase. The GBIA with only 16x16 reduces the number of iterations by a higher amount. However, it has a higher BD-Rate increase than GBIA with quadratic sizes.

3.3 Experiments Conclusions

This chapter presented, in three sections, the evaluation of different hardware-oriented constraints in the AME process, focusing on efficient hardware design for real-time processing on battery-powered devices.

When comparing the evaluation of PU size and iteration, the sizes have an impact, which is between only processing one AMC and only doing GBIA. This is expected because the GBIA uses the error to calculate new MV, which leads to a lower error, and when it is skipped, this leads to a PU with the highest error possible inside the AME algorithm. The opposite, only doing GBIA, shows that the BD-Rate increase is minimal because, as already explained, the GBIA finds the minimal error in up to 5 iterations. The BMIA algorithm has a low impact because it is much more used for the biggest PU sizes, and more importantly, the MVs generated in GBIA already have reduced error.

Because of the points presented, the GBIA was kept, and the PU size constraints were evaluated. This evaluation is done because the GBIA is processed an average of 4.8 times per PU. This makes the number of PU being processed really high, and the Affine ME algorithm is a high-complexity algorithm that needs multiple steps for the refinement of the MVs. So, the removal of PU sizes being processed by the GBIA was presented to reduce the overall iterations in the algorithm.

Only processing quadratic sizes leads to a very low impact on the BD-Rate. This is expected because any non-quadratic PU size can be represented as two quadratic PU with a higher cost. The evaluation of processing of only 16x16 is presented to show a bigger reduction in overall iterations, which is very useful when targeting real-time and

battery-powered devices. Both these constraints will be used to calculate the frequency target of the hardware architecture presented in the following chapter.

4 PROPOSED AME HARDWARE ARCHITECTURES

This section presents the three proposed hardware architectures for the Affine motion estimation of the H.266/VVC. The first architecture presented is the Affine MC, which is used in both the GBIA and BMIA algorithms to generate the predicted PU. The second architecture presented is the Gradient-Based Coefficient Generator, which generates the Affine system of linear equations in the GBIA. The third architecture is the Affine ΔMV, which solves the SLE and generates the ΔMV.

4.1 Affine MC

The Affine MC hardware architecture is presented in Figure 8. This architecture interpolates the PU evaluated in the AME algorithm, and it is used up to 82 times per PU, as discussed in section 2.2. A previous version of this architecture was published (Muñoz et al., 2023), and this version is submitted to the IEEE Design & Test (Muñoz et al., Under Review (28-Aug-2024)).

It supports all PU sizes of the VVC standard, from 16x16 to 128x128, and supports both Affine prediction models. The inputs of the high-level architecture are the three pairs of 13-bit MVs, two 8-bit values that represent the width and height of the PU, two 8-bit horizontal and two vertical positions of the subblocks, one bit to represent the Affine model being used, and two lines of nine 8-bit input samples for each Interpolation Unit.

The high-level architecture can be divided into three components: the Vector Generator for generating the Subblock MVs (SMV), the Buffers that store the SMV, and Interpolation Units to motion compensate four 4x4 subblocks in parallel. Since each Vector Generator generates one SMV per clock cycle, the Buffers implement a pipeline of SMVs being processed. Each Buffer stores the SMV by two up to five clock cycles that the Interpolation Unit requires for processing a 4x4 subblock.

4.1.1 Vector Generator

Affine MC subdivides the PU into 4x4 subblocks. Each of these subblocks requires a Subblock MV. The Vector Generator is responsible for providing these MV to the subblocks of the Affine MC. Figure 13 presents the architecture of the Vector Generator. The Vector Generator receives the three CPMVs as inputs, the Affine model that is being used (4-parameter or 6-parameter model), the PU size, and the subblock position. The output is one SMV. This SMV can be divided into the fractional part (bits on positions 3 down to 0) and the integer part (bits on positions 12 down to 4). The fractional part is used in the Interpolation Unit, while the integer part points to the sample positions in the frame.

The MV Difference unit gives the difference between the input vectors (LT, RT, and LB), presented in Figure 14, which computes the differences of the input MVs according to the CU^{Size} and the Affine model. This difference is required by all SMVs, which are multiplied by the sbp value associated with each SMV, thus resulting in the values of Equations (1) and (2) of the Affine that generate the subblock MV. This is done to simplify the equation when implementing it in hardware and remove the need for the divisor.

The Vector Generator architecture needs four multipliers. However, since it performs only multiplications by 2+4n, thus multiplying from 2 up to 126, those multiplicators can be substituted by Multiple Constant Multiplication (MCM) (Tummeltshammer; Hoe; Puschel, 2007), which leads to reduced power and area utilization.

4.1.2 Interpolation Unit

The Interpolation Unit, shown in Figure 15, interpolates the 4x4 subblock. Each Filter Core receives six samples and outputs one fractional sample that depends on the SMV generated by the Vector Generator. Depending on the SMV, these samples can be a reconstructed line or column of the subblock for horizontal and vertical interpolations. In the case of diagonal interpolation, intermediate samples must be generated

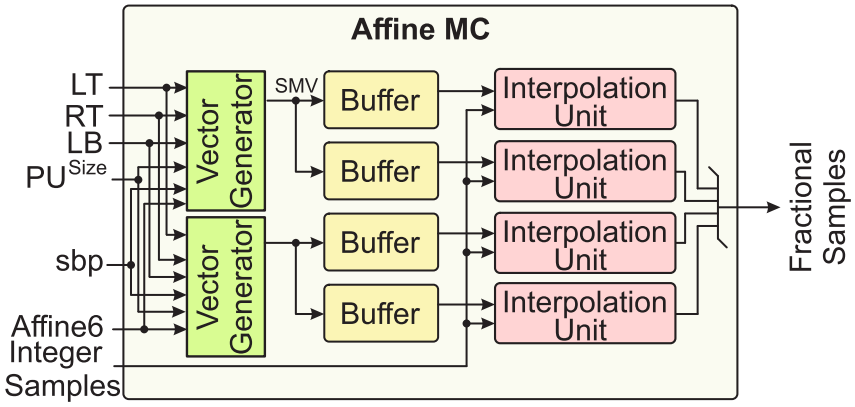

Figure 12 – High-level architecture of Affine MC

first and then stored in the Buffer, as seen in Figure 15. After, the stored samples are used as input in the Interpolation Unit. This architecture contains eight Filter Core, so it interpolates eight fractional samples per cycle, half a subblock for horizontal and vertical interpolations. Diagonal interpolation requires 9x9 fractional samples as input; this architecture can generate them in five cycles and then two additional cycles to interpolate the output subblock itself. The inputs of this architecture are the SMV and 18 (9 per row) 8-bit integer samples per cycle, while the outputs are eight (two rows of the subblock) fractional samples, each generated by a Filter Core.

Each Filter Core module supports the processing of any of the 15 filters. Six of the nine integer samples of the Affine MC are passed to each Filter Core. So, Filter Core 1 receives the inputs from 1 to 6, Filter Core 2 receives the samples from 2 to 7, and so on for the first row, and the same slicing of the samples is performed for the second row of samples (1-6 for the Filter Core 5, 2-7 Filter Core 6, etc.). Since constant values make all the multiplications on the filters, it was possible to exploit an MCM approach. Even though there are eight Filter Core, only one SMV is required to choose the filter. Finally, all the samples that compose a reconstructed line or column of the subblock are clipped to an 8-bit value.

The 15 Affine filters can be implemented in multiple ways. In the following section, a design space exploration of the filters will be presented, explaining in more depth detail the implementation of the filters hardware.

4.1.3 Design Exploration of Filter Core

The Filter Cores take six input samples and output one sample, which can be any of the 15 Affine filters. This section presents two dedicated hardware architectures for the Filter Core: The Power Efficient (PE) and the Hardware Efficient (HE), which were published in (Muñoz et al., 2023a). A baseline implementation is also presented to be used for comparison purposes (Muñoz et al., 2023b). Besides the six 8-bit Integer Samples, each Filter Core also receives the SMV as input to choose which of the 15

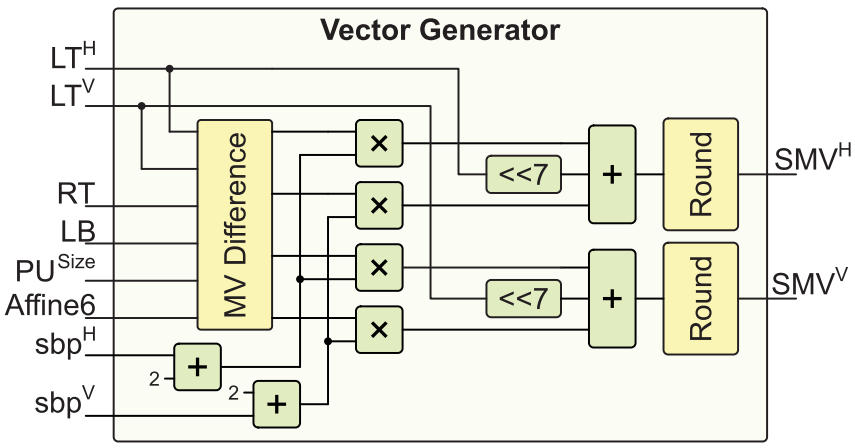

Figure 13 – Architecture of the Vector Generation unit

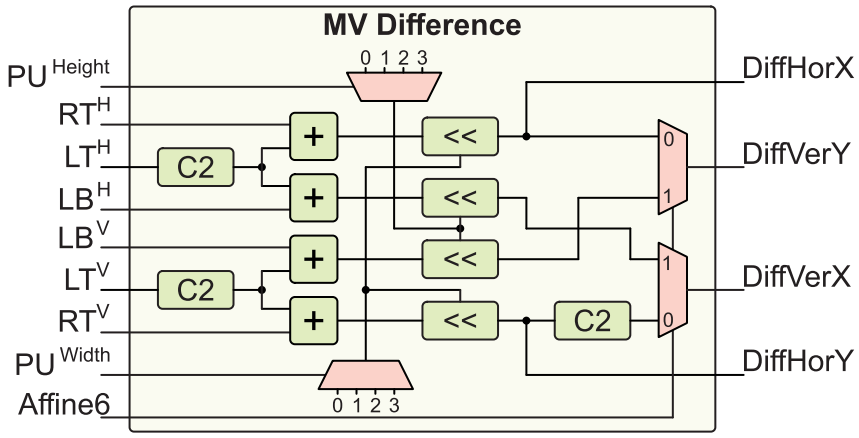

Figure 14 – Architecture that calculates MV differences

filters will be used, along with the offset and shift values used in the Equation (3).

Figure 16 presents the proposed Baseline architecture of Filter F_8, while the architectures of the other filters follow a similar approach in the baseline implementation. The input samples of each Filter Core are 16-bit since diagonal SMV requires intermediate fractional samples (which have 16-bit) as input for the interpolation filters. Each Filter has its coefficients that multiply each input sample, and those values are accumulated. Furthermore, two other values, shift and offset, are used to calculate the value of the filters, where the offset is accumulated to the output before the shift operation. For example, Equation (4) has adopted the offset and shift values as 32 and 6, respectively. All the 15 interpolation filters used in the Affine MC hardware were designed using no multiplier approach since the multiplication is performed only by constant coefficients. As a result, only shifts and adders were used in the design. This approach leads to a more efficient design, reducing area and power results since 16-bit multipliers would use multiple adders and have a 32-bit output.

The following sections will present, first, the Baseline architecture, which is the implementation of 15 Affine filters using no multiplier; second, the Power-Efficient architecture, which exploits the symmetry of the Affine filters; and last, the Hardware-

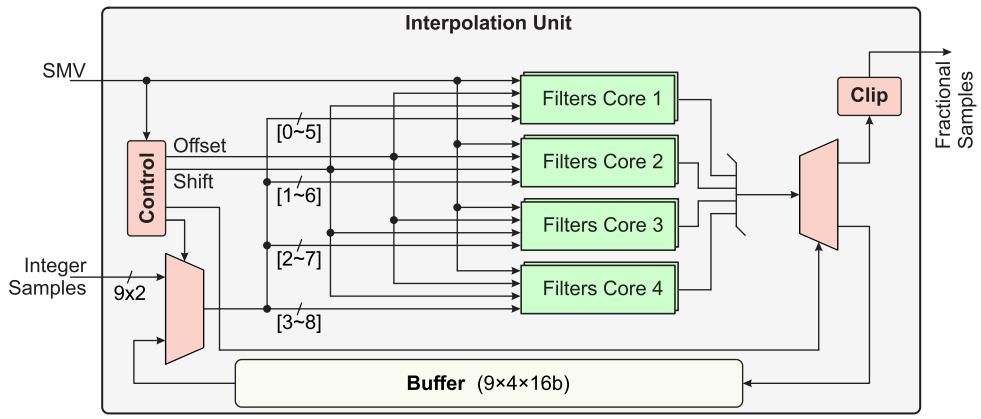

Figure 15 – Architecture of the Interpolation Unit

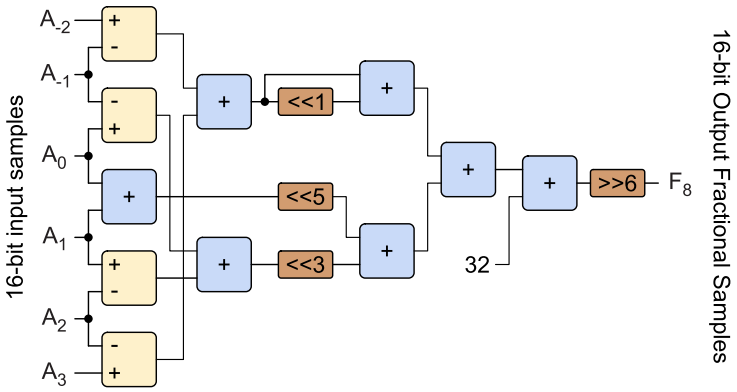

Figure 16 – Proposed Architecture of the Interpolation Filter F_8

Efficient architecture, which uses six coefficient generators instead of having to generate 15 filters.

4.1.3.1 The Baseline Architecture

In the Baseline architecture, all 15 filters in Table 1 were implemented independently. Figure 17 shows the implementation of the filter F_1 without the use of multipliers but instead using Multiple Constant Multiplication (MCM) (Tummeltshammer; Hoe; Puschel, 2007). As can be seen, inside the filter, each input sample is multiplied by its respective coefficient, and then the obtained values accumulate, generating the final interpolated sample. A similar approach was adopted to implement the other 14 filters of the VVC standard. In the Baseline, each Filter Core has 15 filters individually. The SMV defines which of those 15 filters should be used to interpolate the fractional sample. Only the filter chosen by the SMV has its inputs changed, while all other filters have their input values unchanged, so whenever a new filter is selected, only one of the 15 filters dissipates power. This version of this architecture was published in LASCAS 2023 (Muñoz et al., 2023b).

As can be noted in Table 1, in section 2.2, there are several similarities between the

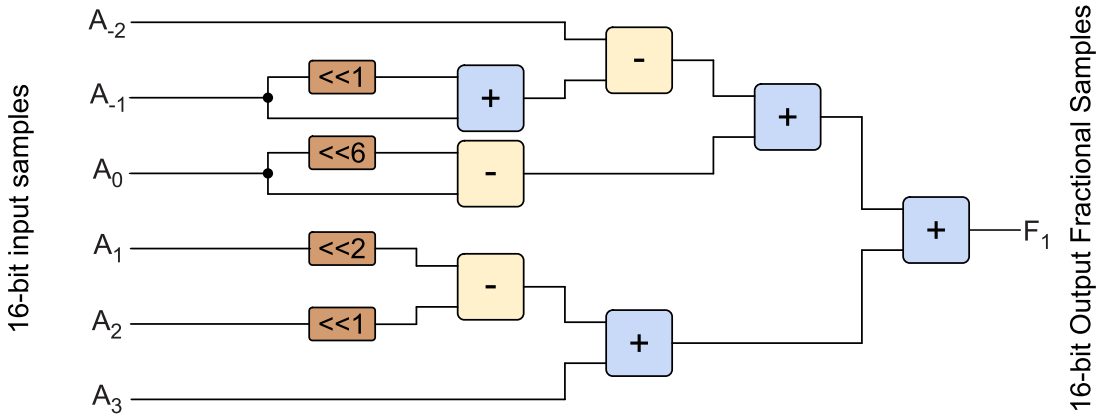

Figure 17 – Implementation of the Filter F_1 using a multiplierless approach in the Baseline architecture

filters. The Filter F_1 is symmetric with the Filter F_{15} when the order of the coefficients (or the input samples in Equation (3)) are inverted. Similarly, the Filters from F_1 to F_7 are also symmetric with the Filters from F_{15} to F_9 and, therefore, this characteristic is explored in the PE version of the Filters Core, which is presented in the next section.

4.1.3.2 Power-Efficient Architecture

In the Power-Efficient (PE) architecture, each Filters Core implements eight dedicated hardware filters to perform the interpolations, being seven hardware filters to implement the F_1 to F_7 or the F_{15} to F_9 filters, plus one dedicated hardware for the F_8 filter. For this, seven of those filters (from F_1 to F_7) have an additional logic able to invert the order of the input samples, as can be seen in Figure 18 that implements both the F_1 and F_{15} Filters. In this additional logic, a control signal controlled by the SMV defines if the order of the input samples should be inverted, thus implementing the F_1 Filter or the F_{15} Filter. Similarly to the Baseline, the PE architecture also adopts a multiplierless approach. Each coefficient of the filters is multiplied by one input sample, and then the obtained values accumulate, generating the final interpolated sample. This architecture was published in ISVLSI 2023 (Muñoz et al., 2023a).

Compared with the Baseline, the Filter Core with the PE architecture reduces from 15 to only eight dedicated filter designs. This reduction in the PE makes it almost use half the area of the baseline approach with lower power dissipation, an all-around improvement over the baseline. The results of the area and power of this architecture will be presented in the chapter 5.

Both the Baseline and the PE architectures implemented each filter separately, where only one filter was enabled according to the SMV, while all other filters were disabled. As can be noted in Table 1, each column contains only a couple of different coefficients, so instead of implementing the 15 filters (eight in the PE architecture) independently, it is possible to implement a generic filter composed of six coefficient

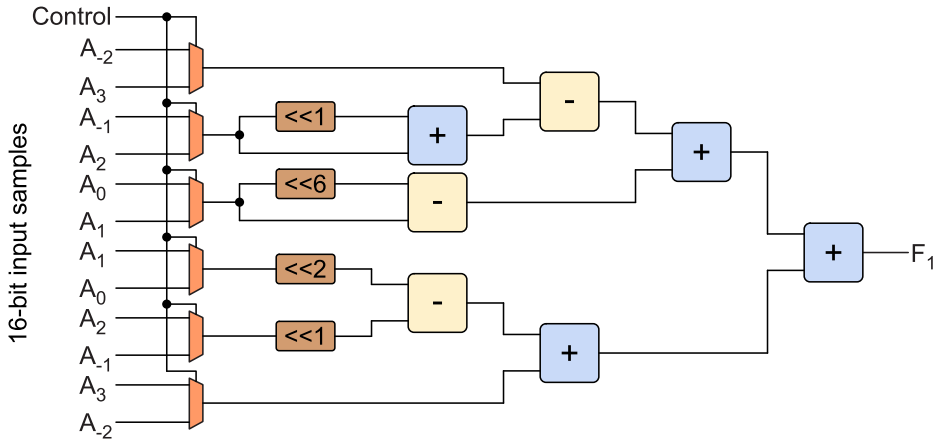

Figure 18 – Proposed Architecture of the Interpolation Filter F_1 or F_{15} (used in the PE architecture), depending on the order of the inputs.

multipliers. This approach will be used in the next section.

4.1.3.3 Hardware-Efficient Architecture

The Hardware Efficient (HE) architecture implements a generic filter, as seen in Figure 19. This generic filter can multiply the samples by the coefficient requested by the SMV before accumulating the values to compose the fractional sample. As can be seen, it receives six samples as input. Each of those samples is passed to a different Coefficient Multiplier, along with a control value, which is the filter chosen. The output of each Coefficient Multiplier is the input sample multiplied by the coefficient related to the SMV. Then, the samples are accumulated, the same as the other two approaches. This version was also published in ISVLSI 2023 (Muñoz et al., 2023a).

Figure 20 presents the proposed architecture of Coefficient Multiplier C_1, which is employed in the HE architecture. As can be noted in Figure 19, the Coefficient Multiplier C_1 receives the input sample A_{-1} or A_2, according to the control signal. So, according to the SMV, the Coefficient Multiplier C_1 can multiply the input sample by 3, 5, 8, 9, 10, or 11, the values in column C_1 from the first eight rows of Table 1. The Coefficient Multipliers only need to implement the first eight rows of coefficients from Table 1 since the HE architecture also exploits the similarities between different filters by inverting the order of the input samples. Finally, since all coefficients from C_1 and C_4 are negative, as can be noted in Table 1, the product of the Coefficient Multiplier C_1 and C_4 are turned negative inside the Filter Core, as can be seen in Figure 19.

Each one of the Coefficient Multiplier architectures, including the C_1 from Figure 20, was implemented using the Multiplexed Multiple Constant Multiplication (MMCM) (Tummeltshammer; Hoe; Puschel, 2007). The MMCM allows performing the multiplication of the input value by a given set of constant values by only using shifts, adders, and multiplexers that exploit the use of the available operators according to the control input, which leads to less area usage.

Compared with the Baseline, the Filter Core with the HE architecture reduces from

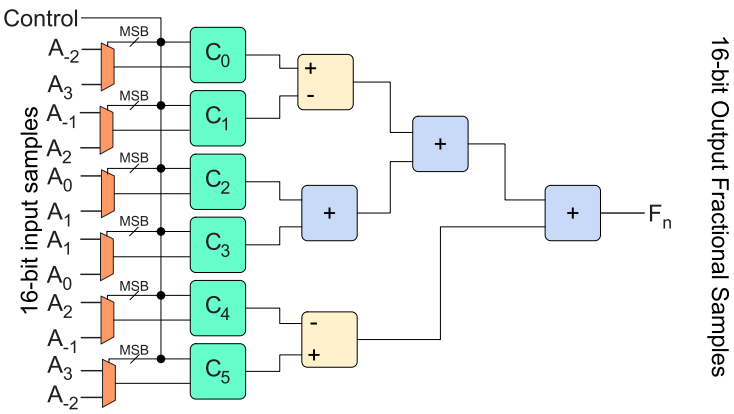

Figure 19 – Proposed Hardware-Efficient Architecture.

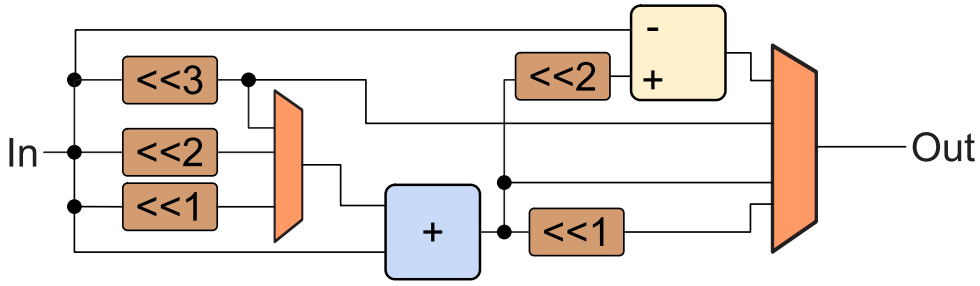

Figure 20 – The Architecture of the Coefficient Multiplier C_1, used in HE architecture.

15 to six Coefficient Multiplier. This reduction in the HE makes it use one-quarter of the area of the baseline approach; however, it has a slight increase in power dissipation, and a significant decrease in area usage is essential when this architecture is multiplied to increase parallelism. The HE architecture was chosen because of its low area utilization. Since Affine MC contains eight of the HE architecture, the overall reduction in area is significant. The results of the area and power of this architecture will be presented in Chapter 5.

4.1.4 Temporal Analysis

This section presents the temporal analysis of the Affine MC architecture. The Affine MC can interpolate and deliver up to 32 (8 per Interpolation Unit) samples at each clock cycle. However, the actual performance may be lower based on the SMVs. Each Affine MC was proposed to interpolate eight fractional samples per clock cycle. So, when the SMV is purely horizontal or vertical, one complete 4x4 subblock is interpolated by the Affine MC in two clock cycles. When the SMV is diagonal, first, the Affine MC requires five clock cycles to generate intermediate horizontal samples. These intermediate samples are buffered and used to interpolate the diagonal fractional samples in the following two clock cycles. Thus, seven clock cycles are required to compose the 4x4 subblock with a diagonal SMV. Therefore, four Affine MCs were adopted since they can all be fed by two Vector Generator when all SMVs are horizontal or vertical.

Figure 21 shows an example of the processing of the architecture for the MVs $LT = [4, -1332]$ and $RT = [-72, -1336]$. The Vector Generator takes the input MV set, which can be two (Affine 4-parameter) or three (Affine 6-parameter) pairs of MVs, and generates a pair of SMV per cycle. In the first cycle, the SMVs [7,0] and [2,0] are generated for processing by the two Interpolation Units, then in the second cycle, the SMVs [13, 0] and [8, 0] are generated for processing by the other two Interpolation Units. After that, the first two 4x4 subblocks are finished, so the process repeats. In this case, the next SMVs are [4,15] and [15, 15], so these subblocks will take seven cycles to finish processing instead of only two. This process is repeated until all subblocks are processed.

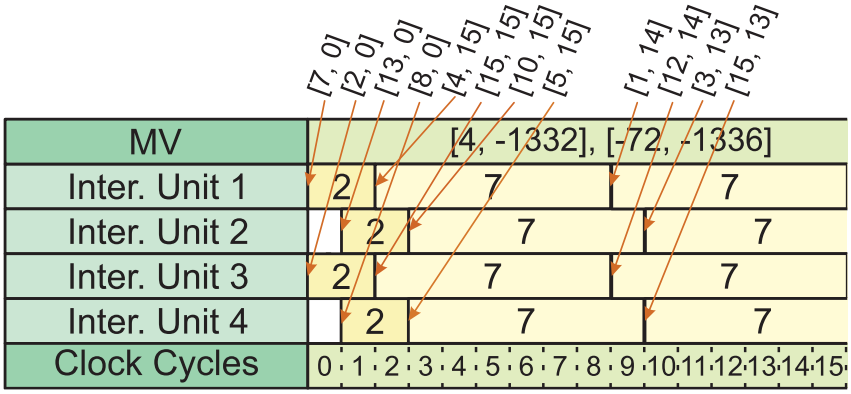

Figure 21 – Time diagram of the Affine MC architecture when $LT = [4, -1332]$ and $RT = [-72, -1336]$

4.2 Gradient-Based Coefficient Generator Architecture

This section presents the Gradient-Based Coefficient Generator Architecture used to generate the gradient of the GBIA. This architecture receives as input one line of 128 8-bit samples per cycle. This line can be one PU of width 128 or any sum of PU widths that adds up to 128. For example, two PU with a width of 64, four with a width of 32, or eight with a width of 16. This enables the architecture to process all PU sizes that the Affine ME supports, ranging from 16x16 up to 128x128, and if the width of the PU is less than 128, this lets the architecture process it in parallel with other PUs. Because of the possible parallelism, this architecture outputs one up to eight Coefficient Matrices, depending on how many PU are currently being processed.

Figure 22 presents the architecture and contains many components to calculate the final coefficients. The first of the components is the row buffer that stores three lines of 128 8-bit samples; the second component is the Sobel engine that generates 128 vertical and horizontal Sobel samples; after that, the two following components are the parameter generator used to calculate the six parameters and the error generator that calculates the error of the row currently being processed. Then, the coefficient generator generates the coefficient matrices. Finally, the last three components are an Adder Tree that adds up all the coefficient matrices, the Affine mode adapter that converts the coefficients to Affine four parameters if necessary, and an accumulator that stores the matrices for 16 up to 128 cycles, depending on the PU height.

4.2.1 Row Buffer

The Row Buffer contains three 128 8-bit buffers linked in cascade. The output of the first 128 8-bit buffer (C) is the input of the second, and the output of the second (B) is the input of the third. This reduces redundancies as the Sobel filter always uses the three most recent lines of the PU for processing, and this implementation makes the most recent line overwrite the oldest one.

The row buffer architecture is used as input for the Sobel Engine, which will be

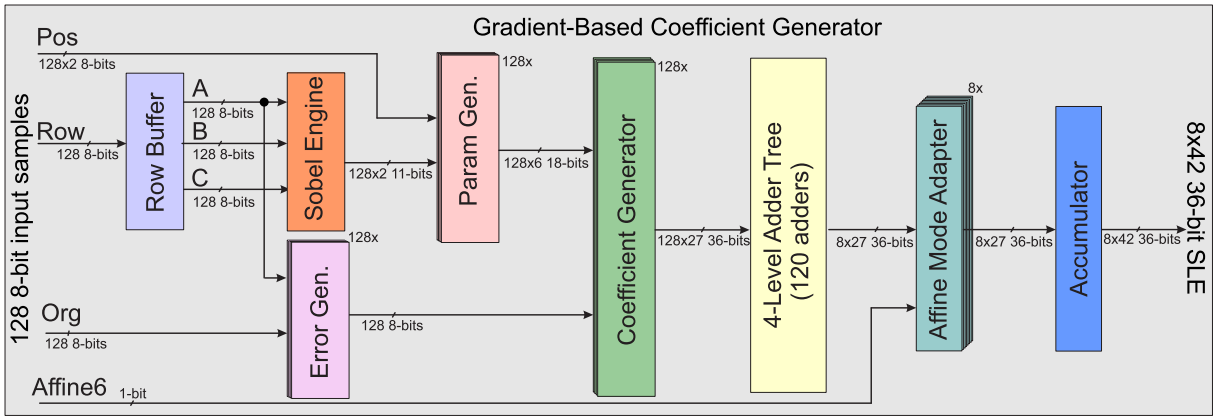

Figure 22 – High-level architecture of the GBIA

detailed in the next section.

4.2.2 Sobel Engine

This section presents the Sobel Engine, as seen in Figure 23. The Sobel Engine receives three 128 8-bit rows of the PU (A, B, C) and outputs 128 11-bit Sobel vertical and horizontal samples. Each Sobel Core receives three 16 8-bit samples and outputs 14 11-bit horizontal and vertical Sobel samples. Because the Sobel requires eight samples around the target sample being interpolated for applying both horizontal and vertical Sobel, this makes the first and last column/rows impossible to interpolate, so to solve that, the first and last Sobel samples in the columns are copies of the sample interpolated closest to it, and the same is true for the first and last rows of Sobel samples. Because of what was previously explained, this architecture contains Sobel Dividers, which are components that either copy the two adjacent horizontal and vertical Sobels between two Sobel Cores or interpolate two new horizontal and vertical Sobel samples, depending on the PU width.

Each Sobel Core contains 14 Sobel Units, while the Sobel Dividers contain two So-

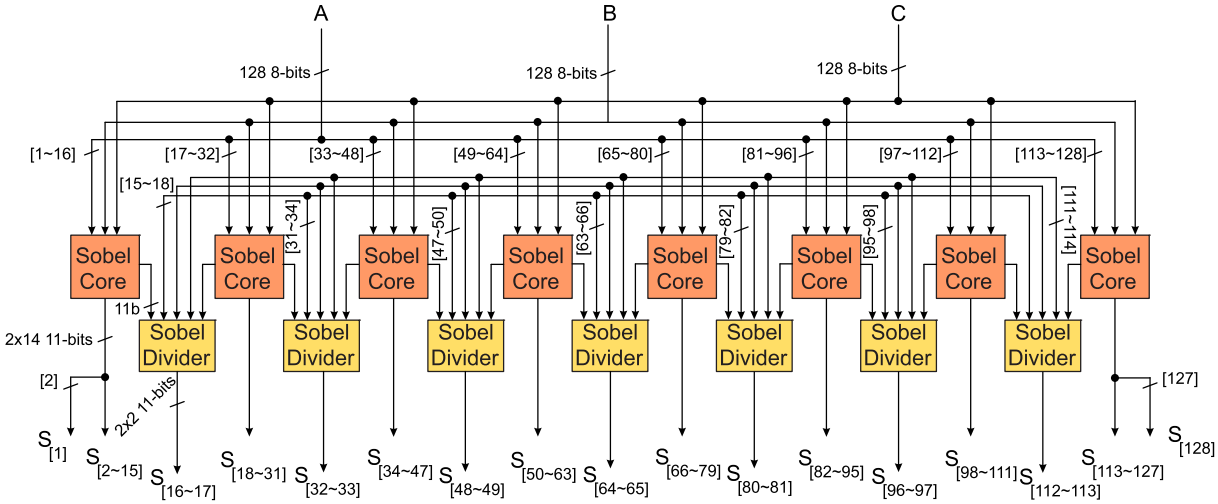

Figure 23 – Architecture of the Sobel Engine

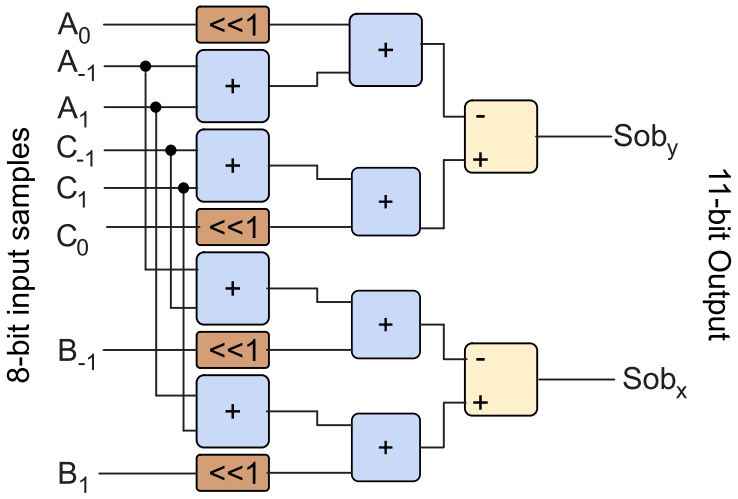

Figure 24 – Sobel Unit architecture

bel Units. The Sobel Engine contains 112 Sobel units in Sobel Cores and 14 in Sobel Dividers, totaling 126 Sobel Units. The architecture of the Sobel Units can be seen in Figure 24. It receives eight 8-bit samples, which are positioned around the target interpolated sample where A is the top row, B is the middle row, and C is the bottom row. It outputs two 11-bit samples, one for the horizontal and one for the vertical Sobel. It was performed using only adders and shifts to reduce the overall area utilization since so much of this architecture is used.

The 128 calculated horizontal and vertical Sobel samples are input for the coefficient generator. The Coefficient generator receives the error as well, which is the subtraction of the original samples and row A, as can be seen in Equation (9). The Coefficient generator will be detailed in the next section.

4.2.3 Parameter Generator

This section presents the architecture that calculates the Affine Affine 6-parameter parameters of the Equation (5), called Parameter Generator, which can be seen in Figure 25. The architecture receives 11-bit samples of the horizontal and vertical Sobel and 8-bit x and y positions of the PU. The output is six 18-bit Affine 6-parameter parameters that will be used to generate the coefficients of the Affine System of Linear Equations (SLE).

The input position is manipulated through shifts, and two are added to the position to point to the center position of the subblock currently being processed. Because only the center of each subblock is used in the multiplication, this multiplicator is substituted with MCM that can multiply the input Sobels by 32 constants (ranging from 2 to 126) that follow the series $c = 2 + 4n$ where n is a number between 0 and 31.

The high-level architecture only processes one row per cycle. However, it processes 128 columns per cycle, which requires this architecture to be replicated 128 times. With the parameters calculated for each column together with the error, the Affine SLE can

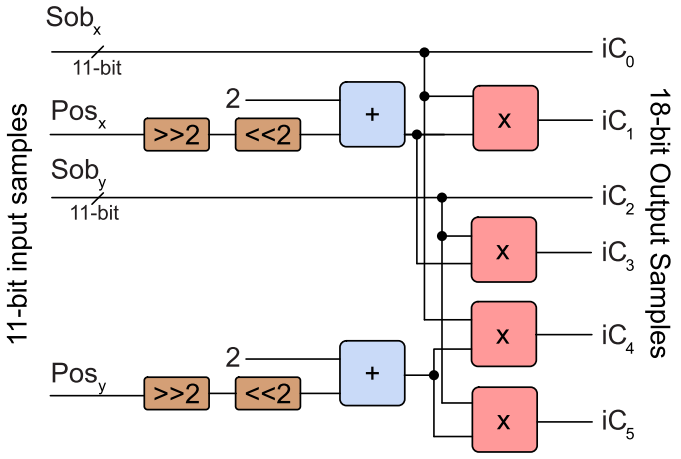

Figure 25 – Architecture of the Affine Parameter Generator

be generated.

4.2.4 Coefficient Generator

The Coefficient Generator architecture is used to calculate the coefficients of the Affine SLE, and it is presented in Figure 26. This architecture implements the equations (7) and (8). This architecture implements the Affine 6-parameter prediction model coefficients. This architecture receives six 18-bit coefficients and one 9-bit error and outputs the SLE with 21 36-bit outputs for the coefficients and six 27-bit outputs for the error.

This architecture only implements the upper triangle of the Affine SLE because it is a multiplication of six parameters by themselves, and the lower triangle is symmetric. This is important in reducing the overall area utilization by almost half, and the lower triangle can be generated without any cost in the last step of the high-level architec-

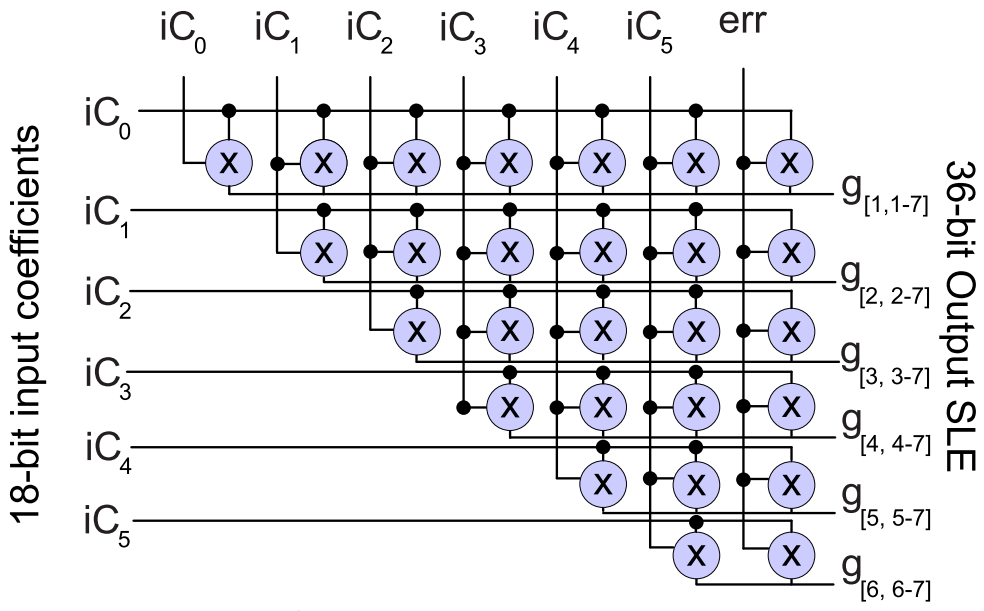

Figure 26 – Architecture of the Coefficient generator

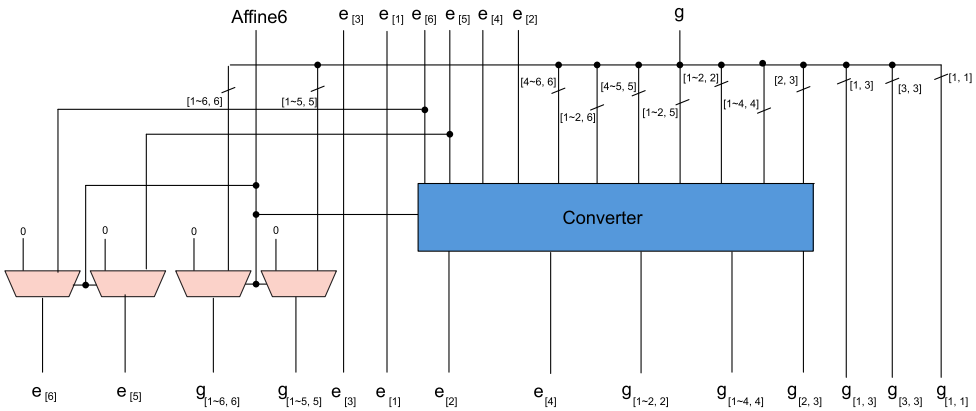

Figure 27 – Architecture of the Affine Mode Adapter

ture. Furthermore, architecture requires 27 full multiplicators because the coefficients calculated in the last step are not predictable. Full multiplicators are expensive, mainly because this architecture is replicated 128 times.

This architecture is followed by an Adder Tree of 4 levels, which reduces the Affine SLE from 128 to 8: 64 full adders in the first, 32 in the second, 16 in the third, and 8 in the fourth. The Affine SLE of the 6-parameter is not equal to the 4-parameter version. However, the 6-parameter can be converted to the 4-parameter only through adders and shifts; the next section presents the Affine Mode Adapter, which does this conversion.

4.2.5 Affine Mode Adapter

This section presents the Affine Mode Adapter architecture. This architecture converts the Affine 6-parameter SLE to the Affine 4-parameter SLE, as shown in Figure 27. It receives 27 Affine 6-parameter coefficients and a 1-bit flag Affine6 and outputs either the same SLE if the 1-bit flag is one or converts the SLE to the equivalent 4-parameter SLE if the 1-bit flag is zero.

This architecture has a converter that uses adders, shifts, and subtractors to convert the 6-parameter to the 4-parameter version. The Converter architecture is shown in the Figure 28. This architecture enables the architecture to process both Affine prediction models and removes the need for two different SLE generator architectures.

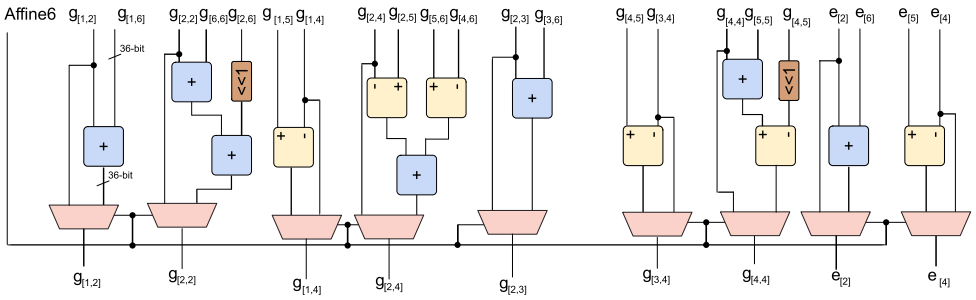

Figure 28 – Architecture of the Affine Mode Adapter Converter

Table 11 – Conversions Equations of the SLE 6-parameters to 4-parameters.

	g_1^4	g_2^4	g_3^4	g_4^4	e^4
g_1^4	$g_{[1,1]}^6$	$g_{[1,2]}^6 + g_{[1,6]}^6$	$g_{[1,3]}^6$	$g_{[1,5]}^6 - g_{[1,4]}^6$	$e_{[1]}^6$
g_2^4		$g_{[2,2]}^6 + g_{[6,6]}^6 + 2 \cdot g_{[2,6]}^6$	$g_{[2,3]}^6 + g_{[3,6]}^6$	$g_{[2,5]}^6 - g_{[2,4]}^6 + g_{[5,6]}^6 - g_{[4,6]}^6$	$e_{[2]}^6 + e_{[6]}^6$
g_3^4			$g_{[3,3]}^6$	$g_{[4,5]}^6 - g_{[3,4]}^6$	$e_{[3]}^6$
g_4^4				$g_{[5,5]}^6 + g_{[4,4]}^6 - 2 \cdot g_{[4,5]}^6$	$e_{[5]}^6 - e_{[4]}^6$

The converter implements the equations in Table 11. This table presents the SLE for the 4-parameter prediction model using the SLE for the 6-parameter as input, with g^4 being the coefficient for the Affine 4-parameter prediction model, g^6 for the six-parameter model, e^4 being the error column in Affine 4-parameter prediction model, and e^6 being the error column in Affine 6-parameter prediction model. The subscripted numbers are the position of the coefficient in the 6-parameter SLE. For example, the coefficient in the position [1, 1] is the same in both SLE; however, the coefficient in the position [1, 2] is different, but the 4-parameter coefficient [1,2] can be generated by adding the 6-parameter coefficients using the equation in the table $g_{[1,2]}^4 = g_{[1,2]}^6 + g_{[1,6]}^6$. The Affine Mode Adapter implements all these equations from Table 11 when the Affine6 flag is one. Otherwise, the output SLE is the same as the input one.

This architecture is repeated eight times because the high-level architecture can process up to eight PUs with a width of 16 simultaneously, which requires this architecture to be repeated. The output of this architecture is added up in the accumulator every cycle.

The final step in the Gradient-Based Coefficient Generator is the accumulator, which accumulates the SLE every single cycle while the current cycle is smaller than the PU height. The accumulator architecture receives 8 SLE with 27 coefficients of 36 bits and outputs 8 Affine SLE with 42 coefficients of 36 bits. Because the Affine SLE is symmetrical, these additional 15 coefficients have no cost in components or need to be stored; they can be implemented only through wiring.

4.2.6 Temporal Analysis

This section presents a temporal analysis of the Gradient-Based Coefficient Generator architecture. The Gradient-Based Coefficient Generator can process any PU size, and if the PU width is less than 128, the architecture can process multiple PUs in parallel. This architecture processes the PU in one row per cycle. This architecture takes 16 up to 128 cycles to finish a PU, depending on the height. The 16-wide processing slot that enables the parallelism of working in multiple PUs simultaneously will be called Gradient Engine in this section.

Figure 29 presents an example of the processing of the architecture. The architecture starts on the first cycle of processing two PUs of size 16x16, which takes up two of

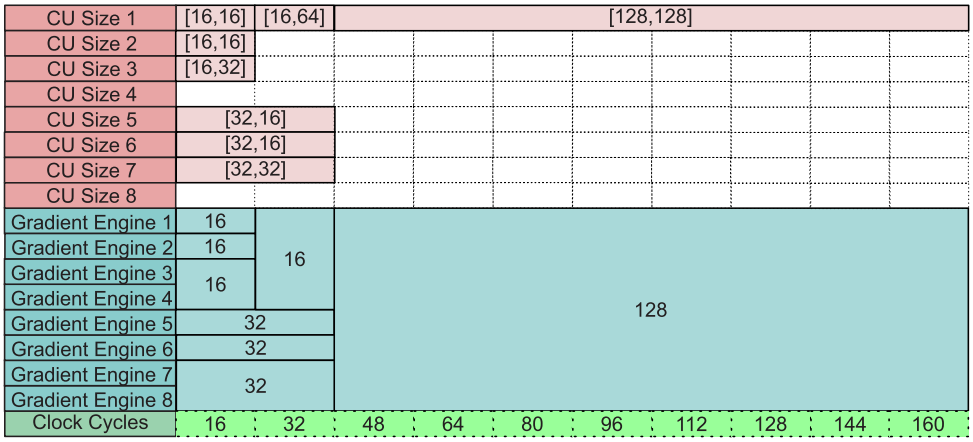

Figure 29 – Temporal Analysis of the Gradient-Based Coefficient Generator

eight possible Gradient Engines for 16 cycles. This architecture also starts processing two PUs of size 32x16, which also takes two of eight Gradient Engines, however, for 32 cycles. The last four Gradient Engines are taken by PUs of size 16x32 and 32x32. After 16 cycles have passed, a PU of size 16x64 starts processing, taking up half of the Gradient Engines. Finally, this architecture starts processing a PU size of 128x128 for 128 cycles.

Even though Gradient Engines 4 and 8 never received a PU directly, they both were never idle. This size of parallelism is chosen because processing a PU takes a long time, at least 16 up to 128, so processing multiple PUs simultaneously reduces the cost of waiting.

4.3 Affine ΔMV Architecture

This section first presents the Gauss-Jordan Elimination Algorithm and then the proposed Affine ΔMV architecture employing this algorithm. This architecture solves the SLE generated by the Gradient-Based Coefficient Generator Architecture and outputs the ΔMV of the Affine ME of the VVC standard. To generate the ΔMV, the Affine parameters, which is the solution of the SLE, need to be converted in a final processing step in which three of the four Equations are applied (11), (12), (13), and (14), depending on the Affine prediction model being used is 4-parameter or 6-parameter. A previous version of this architecture that implemented a different algorithm to solve the SLE was published in JICS (Maass et al., 2023), and this version of this architecture was published in SBCCI (Maass et al., 2024).

4.3.1 Gauss-Jordan Elimination Algorithm

The Gauss-Jordan elimination is an algorithm used to solve SLE in which operations are performed by adding and subtracting rows from each other until all values in the main diagonal are one and all other positions are zero.

To solve the SLE (10), the algorithm starts by selecting the element in the first row and column, $g_{[1,1]}$, as the pivot and normalizing its row by dividing all elements of the row by the pivot. With the first row normalized, all other rows are updated using the Equation (16), in which the R is the current row, and R_p is the pivot row, and E_p is the element in the pivot column of current row; this operation ensures that the element in the column of the pivot will always become zero. After this process is performed, only one element is non-zero in the pivot column, then the next element in the main diagonal is normalized and becomes the new pivot, and then the process is repeated. When the process is completed, the last column of the matrix contains the Affine parameters that will be used and converted into ΔMV.

It is very unlikely that the pivot values become zero during the algorithm because the SLE is generated in the last step. However, if the pivot is zero, then the process stops, and the output is set to zero.

$$R_{new} = R_c - R_p \times E_p \tag{16}$$

4.3.2 Proposed architecture

The architecture presented in this section solves the SLE generated by Gradient-Based Coefficient Generator architecture and also does the final processing. This architecture is able to process both Affine 4-parameter and Affine 6-parameter and all 12 PU sizes. The Architecture can be seen in the Figure 30. The input of architecture is 1-bit for the Affine model, 4x5 or 6x7 47-bit matrix, and two 8-bit values that represent the PU height and width, which ranges from 16x16 up to 128x128. The output of this architecture is four or six 18-bit ΔMV, depending on the Affine model.

The architecture presented in Figure 30 is divided into three modules: The first module is the Float Conversion which is responsible for converting the integer inputs into a 32-bit single-precision floating-point representation; The Calculator that implements the Equation (16) to solve the SLE, also it performs the final processing which is the implementing the Equations (11), (12), (13), and (14); the last module is the Control Unit. The Calculator comprises three dividers, 15 multipliers, and 15 subtractors, all

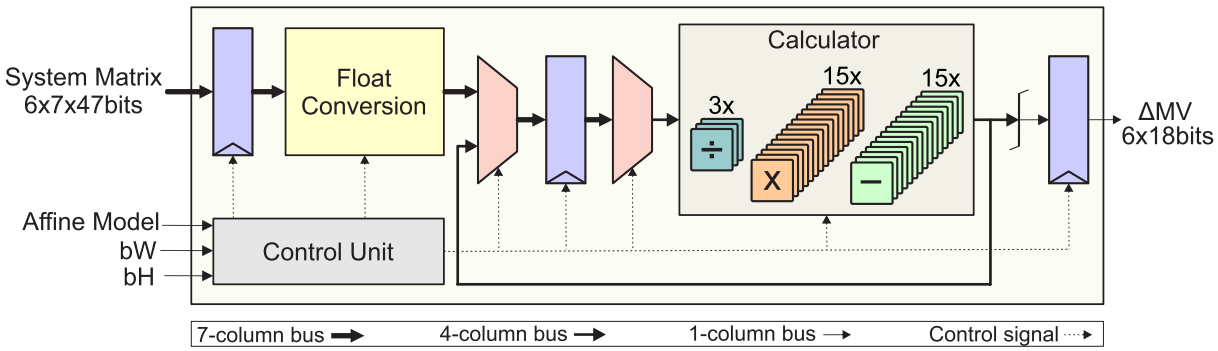

Figure 30 – The Affine ΔMV Architecture.

using 32-bit floating point values.

4.3.3 Temporal Analysis

This section presents the temporal analysis of the Affine ΔMV architecture. This architecture is able to process three columns of the matrix at each clock cycle, and this leads to this architecture taking 12 cycles when processing the Affine 6-parameter model and eight clock cycles for the 4-parameter model to generate the ΔMV. Figure 31 presents a time diagram of the architecture processing each cycle. Figure 31 presents the operations performed for both Affine 4-parameter and 6-parameter, where one is blue and the other is red, respectively. As can be seen in figure 31, the first cycle in both models is to convert the inputs into floating point. The following five or nine cycles, depending on the Affine model, are for updating each row. Then, finally, the last two cycles are for the final processing and enabling the output.

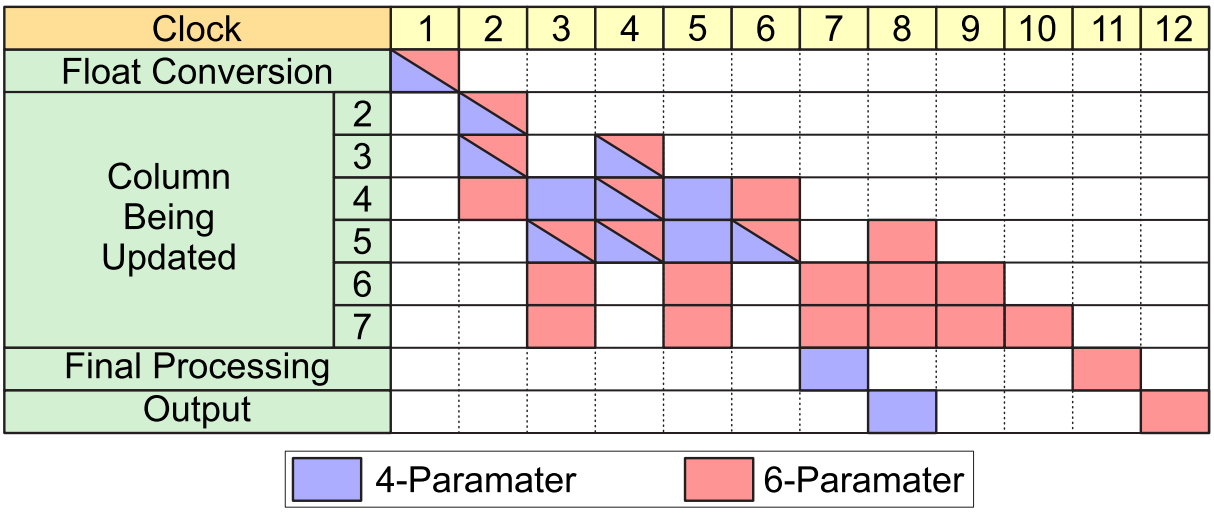

Figure 31 – Time diagram of the proposed Affine ΔMV architecture.

5 SYNTHESIS RESULTS AND COMPARISONS

This chapter presents the synthesis results for all architectures developed. The architectures presented in this master dissertation were described in VHDL and validated with the ModelSim tool (Intel, 2022). The syntheses were performed for the 40nm standard-cell technology of TSMC using the Cadence RTL Compiler tool (Cadence, 2024). The area results presented are related to the number of equivalent NAND2 gates, which are obtained according to the NAND2 size in the specific technology. For TSMC 40nm, a NAND2 occupies 0.9408mm^2, and the power dissipation results considered the switching activity for real input data obtained from the reference software implementation of the VVC standard, the VTM version 16.2 (Chen; Ye; Kim, 2021).

This section presents the synthesis of four architectures: The Filter Cores, which are used in the Interpolation Unit; the Affine MC, which is used in the Affine ME to evaluate every MV; The Gradient-Based Coefficient Generator Architecture, which is the architecture used in GBIA to generate the SLE used to calculate the next set of MV being evaluated; and the Affine ΔMV architecture, which solves the SLE generated by the Gradient-Based Coefficient Generator Architecture.

5.1 Interpolation Unit Results

This section presents the synthesis results and comparison between the three different implementations of the Filter Core (Baseline, Power-Efficient, and Hardware-Efficient) used in the Interpolation Unit in Table 12. The Baseline architecture was published in (Muñoz et al., 2023b), while the Power-efficient (PE) and Hardware-Efficient (HE) architectures were presented in (Muñoz et al., 2023a). The synthesis of each Interpolation Unit is performed using four Filter Core, enough for one line of the subblock. Also, at the frequency of 808MHz, this architecture outputs 202M horizontal/vertical subblocks or 62.2M diagonal subblocks per second. This enables the architecture to process UHD 4K video at 44 fps, the closest this architecture is able to get to 60 fps, with GBIA enabled while supporting only 16×16 PUs. The power results consider

the switching activity for 100k filtering operations of the BasketballDrive test video sequence (Bossen et al., 2020).

Table 12 presents the power dissipation and area for the three architectures divided per module. It is organized into several columns: *Filter Core*, detailing the Filter Core implementations being compared; *Buffer*, representing the buffer responsible for storing intermediate samples; *Other*, which includes any additional logic necessary for the architecture; and *Interpolation Unit*, which reflects the entire architecture. This breakdown allows for a more granular comparison of how different components contribute to the overall power dissipation and area requirements of each architecture.

It can be seen in Table 12 that the Power-Efficient architecture (PE), when compared to the Baseline Architecture, reduces the area to 44.2% and reduces the power dissipation by 4.5%. These reductions in area and power dissipation are expected since seven filters are substituted by six multiplexers to invert the input, reducing the number of filters from 15 to eight and leading to a significant area reduction. The reduction in power dissipation is also expected, with the removal of almost half of the components. Still, it is not significant because only the selected filter has switching activity. Hence, this reduction comes from the leakage power dissipation of the components used to control the filters, such as the multiplexers, that keep the input stable and without dynamic power dissipation if not selected.

Table 12 also presents the Hardware-Efficient architecture (HE) when compared to the Baseline Architecture, which reduces the area by 77.5% and increases the power dissipation by 7.5%. This is a significant reduction in area, which is expected since it only uses six components, one for each input, instead of 15 filters. The increase in power dissipation is because all components in this architecture are constantly switching, which leads to a higher dynamic power dissipation different than the Baseline, which would have 14 of the 15 filters disabled.

When comparing both the PE and HE architecture, both reduce the area utilization when compared with the Baseline, but the PE minimizes the power dissipation while the HE increases. Because the difference in power dissipation between PE and HE is 12%, this can be significant depending on how many Filter Core are used. However, since the Filter Core are used eight times per Interpolation Unit in the Affine MC, reducing

Table 12 – Synthesis results for Interpolation Unit using **four** Filter Cores working at 808.7 MHz.

		Filter Cores	Buffer	Other	Interpolation Unit
Baseline	Area (k Gates)	90.67	5.10	1.74	97.59
	Total Power (mW)	4.27	6.18	2.95	13.40
PE	Area (k Gates)	47.86	5.12	1.46	54.43
	Total Power (mW)	5.08	6.17	1.54	12.80
HE	Area (k Gates)	15.14	5.10	1.67	21.91
	Total Power (mW)	6.7	6.17	1.54	14.41

the area by almost one-quarter of the baseline, or half of the PE, is considered more important.

5.2 Affine MC Synthesis Results

This section discusses the synthesis results for the Affine MC, as presented in Table 13. Table 13 presents area and power results for the architecture when processing the required number of Affine MCs for the GBIA at UHD 4K@60fps for only PUs of size 16x16. The power results consider the switching activity for 100k filtering operations of the Tango2 test video sequence (Bossen et al., 2020). The Interpolation Unit in this version has double the throughput of our previous version (Muñoz et al., 2023), and the increase in throughput is to enable the processing of UHD 4K videos with GBIA enabled. This version of the architecture was submitted to IEEE Design & Test (Muñoz et al., Under Review (28-Aug-2024)).

Different from the Filter Cores frequency, the frequency of the Affine MC architecture was chosen by calculating the average amount of Affine MC for the PU size of 16x16 per frame while only the GBIA is enabled. The Equation (17) is used to calculate the frequency necessary to process that PU size, where AMC is the average number of PUs being interpolated per frame, CU_{Sb} is the number of 4x4 subblocks in a PU of that width and height, $througput_{hv}$ and $througput_{d}$ are how many cycles takes for horizontal/vertical and diagonal subblocks to be interpolated, $ratio_{hv}$ is $ratio_{d}$ is ratio of between horizontal/vertical and diagonal subblocks, and finally, multiply by the fps. This is done for each PU size separately, which the architecture needs to process, and then it is accumulated; for the case of only one PU size, this equation is only required once.

$$freq_{cu} = AMC \cdot PU_{Sb}(througput_{hv} \cdot ratio_{hv} + througput_{d} \cdot ratio_{d})fps \tag{17}$$

Using the Equation (17), there is an average number of 399k 16×16 Affine MC per frame, and since each 16×16 PU contains 16 4×4 subblocks, this is equal to 6.384M subblocks of 4x4. Each Interpolation Unit has eight filter cores, which enables the in-

Table 13 – Synthesis Results for the Affine MC for 4K@60fps while only processing GBIA and 16x16 PUs.

Throughput		Vector Generator	Interpolation Unit	Other	**AffineMC**
	Area (k Gates)	33.1 (17.50%)	147.7 (78.05%)	8.4 (4.45%)	**189.2 (100%)**
UHD 4K@60 16x16 (598 MHz)	Total Power (mW)	31.486 (25.99%)	77.908 (64.31%)	11.759 (9.71%)	**121.153 (100%)**

terpolation of two subblocks per cycle if the SMV is horizontal/vertical or seven cycles if the SMV is diagonal. Since this architecture has four Interpolation Units, it interpolates two subblocks per cycle, or the throughput of one subblock every 0.5 cycles. For the diagonal throughput, there is a subblock every 1.75 cycles. The ratio of horizontal/vertical is 15%, and the diagonal is 85% for UHD 4K videos. Using all the values presented here, the frequency calculated is 598MHz for UHD 4K at 60 fps.

It can be seen in Table 13 that Affine MC has 189k gates of area and a power dissipation of 121mW. Also, it can be seen in Table 13 that the two Vector Generators are 17.5% of the total area utilization while the four Interpolation Units are equal to 78% of the total area, which is expected since the Interpolation Unit has eight Filter Cores each. Similar results can be seen for power dissipation; the Interpolation Units are 64% of the total power, and the Vector Generator 25% is the total power.

It can be seen in Table 13 that the Affine MC architecture has 189k gates of area and a power dissipation of 121mW. Also, it can be seen in Table 13 that the two Vector Generators are 17.5% of the total area utilization while the four Interpolation Units are equal to 78% of the total area, which is expected since the Interpolation Unit has eight Filter Cores each. Similar results can be seen for power dissipation; the Interpolation Units are 64% of the total power, and the Vector Generator 25% is the total power.

The 4K synthesis is shown only for the GBIA and 16x16 because, by using the equation explained before, this architecture would require a frequency of 2.3GHz to process quadratic sizes. This frequency is prohibitive for 40nm TSMC without taking into consideration the power dissipation, which would be very high. The limitation of UHD 4K@60fps with GBIA and only processing 16x16 has a BD-Rate increase of 0.8%, which is good enough depending on the application.

5.3 Gradient-Based Coefficient Generator

This section presents the synthesis results for the Gradient-Based Coefficients Generator Architecture in Table 14. Two syntheses for the architecture are presented, both able to process UHD 4K (3840x2160 pixels) at 60fps videos. The frequencies 186MHz and 48MHz were calculated using the average amount of 16x16 PU processed per frame in GBIA and the average amount of quadratic PUs processed per frame in GBIA. The power dissipation results considered the switching activity for real input data from 100k PUs of the BasketballPass test video sequence (Bossen et al., 2020).

This section discusses the synthesis results for the Gradient-Based Coefficients Generator Architecture, as presented in Table 14. Two syntheses for the architecture are presented, both able to process UHD 4K (3840x2160 pixels) at 60fps videos. The frequency is calculated using the Equation (18), which is the frequency necessary to

Table 14 – Synthesis Results for the Gradient-Based Coefficient Generator for UHD 4K@60fps for both throughputs, GBIA and quadratic sizes (186MHz) and GBIA and 16x16 (48MHz).

Throughput		UHD 4K@60 GBIA Quad	UHD 4K@60 GBIA 16x16
Error Gen. & Sobel Engine	Total Power (mW)	179.05 (2.59%)	58.38(1.48%)
	Area (k Gates)	117,021 (2.41%)	
Parameter Generator	Total Power (mW)	741.34 (10.71%)	284.07(7.22%)
	Area (k Gates)	423,136 (8.73%)	
Coefficient Generator	Total Power (mW)	594.28 (8.58%)	248.07(6.31%)
	Area (k Gates)	549,878 (11.34%)	
Adder Trees	Total Power (mW)	5,347.69 (77.25%)	3,265.43(83.03%)
	Area (k Gates)	3,653,765 (75.35%)	
Other	Total Power (mW)	60.55 (0.87%)	76.97(1.96%)
	Area (k Gates)	104,955 (2.16%)	
Total	**Total Power (mW)**	**6,922.90 (100.00%)**	**3,932.92(100.00%)**
	Area (k Gates)	**4,848,754 (100.00%)**	

calculate one PU, then this is performed for all PU sizes and added up. Where AMC is the average number of Affine MC being processed per frame, the $128/CU_{width}$ is because this architecture can process multiple PU simultaneously if the width of the PU is less than 128, CU_{height} is the height of the PU and how many cycles this architecture takes to process the PU.

$$freq_{cu} = (AMC \cdot PU_{height} \cdot fps)/(128/PU_{width}) \tag{18}$$

Calculating the frequency for 16x16 using GBIA by using the Equation (18), where average AMC is equal to 399k for UHD 4K Class A2 video, the PU width and height is 16, and the fps is 60, this is equal to 48MHz which is the frequency for only 16x16. The same is done for all other quadratic sizes and added up to calculate the frequency to process quadratic sizes, which is equal to 186MHz. As can be seen in Table 14, the architecture was synthesized for both UHD 4K at 60 fps frequencies, 48MHz and 186MHz. However, this level of parallelism has a high cost, with an area of 4,848k gates and power dissipation of 6,922mW for all quadratic sizes and 3,932mW when processing only 16×16 PUs. This makes the architecture impractical for real applications.

The high area and power dissipation in the GBIA algorithm were always expected since it is the most complex part of the AME algorithm, requiring not only calculating the Sobel and error for each row of the PU being processed but also generating matrices of 27 coefficients for each sample, which are added up together in Adder Trees. This architecture contains many full adders and multiplicators that cannot be removed, followed by Adder Trees, which increase the sample size. All these lead to the area

and power dissipation being the highest of all the architectures presented.

Some strategies could be adopted to solve this area and power dissipation issues: first, the reduction of the parallelism from 128 to other PU sizes would reduce the overall area significantly since more than half of the multipliers and adders would be removed, and the Adder Trees reduced; however, this would increase the overall frequency. Another strategy is the copying of neighboring values, which is already done in the edge samples of the PU for the Sobel. This could be done for every couple of samples to reduce the number of numbers that need to be calculated; this would have much less impact on the overall area and power dissipation; however, it would not impact the frequency. Therefore, these strategies will be explored in future works.

As can be seen in Table 14, architecture can process UHD 4K videos at 60fps with a BD-Rate increase of 0.80% when targeting 16x16 with GBIA and 0.27% when targeting quadratic sizes with GBIA. This is the first architecture in literature to implement the Gradient-Based Coefficient Generation of the GBIA of the VVC standard. This architecture can be integrated into the architecture presented in the next section to solve the System of Linear Equations and generate the ΔMV.

5.4 Affine ΔMV Architecture

This section shows the results for the Affine ΔMV Architecture. The average number of SLE is used to calculate the frequency of the architecture by using the Equation (19), where AMC is the number of Affine MC being processed, $ratio_4$ and $ratio_6$ are the ratios of Affine 4-parameter and 6-parameter and $throughput_4$ and $throughput_6$ are how many cycles to generate the SLE. The architecture results (Maass et al., 2024), which implements the Gauss-Jordan Algorithm, is an optimized version of our previous work (Maass et al., 2023), which implemented Gaussian elimination with partial pivoting algorithm.

$$freq_{cu} = AMC(ratio_4 \cdot throughput_4 + ratio_6 \cdot throughput_6) \cdot fps \tag{19}$$

Using the Equation (19), the average number of Affine MC 16x16 is 399K, the ratio of Affine each parameter for one reference frame is 0.53% for 4-parameter and 0.47% for 6-parameter, and the architecture takes eight cycles for 4-parameter and 12 cycles for 6-parameter, all this data is for the for UHD 4K video class using one reference frame. This results in an operational frequency of 236MHz for processing GBIA and only 16x16 PU, and for the quadratic sizes, the necessary frequency is 334MHz.

Table 15 presents the synthesis results for UHD 4K (3840x2160 pixels) at 60fps videos for the two targets, GBIA only 16x16 and GBIA quadratic sizes. This also presents the results for each module. The power dissipation results considered the switching activity for real input data from 100k SLE of the RaceHorses test video se-

Table 15 – Synthesis results of the Affine ΔMV architecture when targeting 4K UHD 60 fps videos

	UHD 4K@60fps (236MHz) GBIA and 16x16		UHD 4K@60fps (334 MHz) GBIA and quadratic	
	Cell Area (k Gates)	Total Power (mW)	Cell Area (k Gates)	Total Power (mW)
Control	0.05	0.022	0.05	0.031
Conversion	53.3	17.260	53.3	23.682
Calculator	180.2	9.052	180.2	16.013
Total	**245.7**	**31.444**	**245.7**	**47.146**

quence (Bossen et al., 2020). As can be seen in Table 15, the float conversion module has the highest power dissipation, and the Calculator has the highest area utilization. The complete architecture requires 245.7k gates in total while dissipating 31.44mW for GBIA only 16x16 and 47.15mW for GBIA quadratic sizes.

5.5 Comparison with related works

This section presents the comparisons of the achieved results with related works. For the Affine MC architecture, there is only one work that proposes hardware (Taranto, 2022). However, to increase the number of comparisons, the proposed architecture is also compared to other works that process the Fractional ME, as the Fractional ME uses filter interpolation in a similar way to the Affine MC even though the visual degradation cannot be compared as they are different steps of the ME. For Affine ΔMV architecture, there are two compared related works, one of our previous implementations Maass et al. (2023), which implements the final processing step, and Sheng et al. (2024), which is a SLE solver without the final processing. Unfortunately, it was impossible to compare the results achieved for the proposed Gradient-Based Coefficient Generator architecture since no other published work was found in the literature.

The Affine MC and Affine ΔMV architectures were resynthesized for processing FHD 1080p (1920x1080 pixels) at 60 fps since most related works have synthesis for this target. The frequency this time was calculated using the average PUs being processed in Class B of the CTC, which is the HD 1080p class.

The operational frequencies of the Affine MC for this new synthesis are 560MHz for Quadratic GBIA and 160MHz for 16x16 GBIA for processing FHD 1080p at 60 fps. The results for the Affine MC are presented in Tables 16 and 17, where Table 16 presents results from FPGA implementations, and Table 17 presents the results of ASIC implementations. Two different throughputs are shown in Tables 17 and 16, one for AME with quadratic sizes and GBIA enabled and another for the AME with only 16×16 PUs with GBIA enabled. The BD-Rate increase for both proposed throughputs

Table 16 – Comparison among the proposed Affine MC architecture and FPGA related works.

	Azgin	Mahdavi	**Proposed**	
	FPGA		**ASIC**	
Design	FME	FME	**AME Quad**	**AME 16x16**
Technology	Virtex 7	-	**40nm**	
Area (μm^2)	-	-	**178005**	
Area (k Gates)	-	-	**189.2**	
Frequency (MHz)	227	236	**560**	**160**
Power (mW)	320.18	225	**119.099**	**42.763**
BD-Rate	-	-	**0.27%**	**0.80%**
Resolution	1080p	1080p	**1080p**	
FPS	47	49	**60**	

is 0.27% for the quadratic version and 0.80% for only 16 × 16 PUs, respectively.

Table 16 shows that Azgin; Kalali; Hamzaoglu (2020) require much more power to process FHD 1080p at 47 fps while having much lower frequency when compared to both proposed architectures. Similar to the previous related work, Mahdavi; Azgin; Hamzaoglu (2021) also requires much more power to process FHD 1080p at 49 fps than both of the proposed architectures; however, the architecture of quadratic sizes requires a higher frequency. These results can be seen even though the AME is a more complex algorithm than the FME, which does not require multiple iterative algorithms, which leads to a huge number of Affine MC to be processed. It is important to highlight that FPGA synthesis results cannot be directly compared to ASICs, especially regarding area and power since these two technologies serve different purposes. However, this comparison is made due to the limited number of related works.

As can be seen in Table 17, when compared to the FME related works, the proposed architectures have the highest number of gates. However, the 16×16 version is the second lowest power, only losing to Silva et al. (2021) while the GBIA and Quadratic version is the third, losing to also Canmert; Kalali; Hamzaoglu (2018) but being below Azgin et al. (2018), even though the AME requires much more interpolations than the FME because of the iterative algorithms. Taranto (2022) is the only comparison of the proposed hardware for the Affine ME. Both of our proposed architectures have a higher number of gates and power dissipation, however, Taranto (2022) employed many simplifications to the AME process. First, it divides each PU into 16×16 subblocks, and It only processes the four corner subblocks; the second simplification is the removal of fractional interpolation, which is one of the most expensive components regarding the area in our proposed Affine MC architectures is the Filter Core. The last simplification is the removal of the iterative algorithms GBIA and BMIA. All these simplifications significantly reduce the overall frequency necessary, but also the coding efficiency, however, the authors do not provide the coding efficiency losses its proposed solution (Taranto,

Table 17 – Comparison among the proposed Affine MC architecture and ASIC related works.

	Azgin	CanMert	Silva	Taranto	**Proposed**	
	ASIC				**ASIC**	
Design	FME	FME	FME	AME	**AME Quad**	**AME 16x16**
Technology	90 nm	90 nm	65nm	45 nm	**40nm**	
Area (μm^2)	-	-	-	23,889	**178,005**	
Area (k Gates)	11.7	37.6	68.7	29.9	**189.2**	
Frequency(MHz)	357	435	522	341	**560**	**160**
Power(mW)	77.06	467.93	23.9	1.82	**119.099**	**42.763**
BD-Rate	-	-	-	not calculated	**0.27%**	**0.80%**
Resolution	-	1080p	1600p	1080p	**1080p**	
FPS	-	88	30	50	**60**	

2022).

Table 18 presents the new synthesis results for both throughputs of the proposed Affine ΔMV architectures, with operation frequencies of 87MHz for Quadratic and GBIA and 64MHz for only 16x16 PU and GBIA. Sheng et al. (2024) presents an architecture that solves the SLE, and it does not provide results on power dissipation. However, it provides how many mega equations are solved per second, and as can be seen in the proposed architectures throughputs, Sheng et al. (2024) is not able to process the throughput necessary for GBIA using quadratic or 16x16 PU sizes. Our previous work (Maass et al., 2023), on the other hand, provides data power dissipation and area, and the throughput is enough to process 16x16 PUs and perform the GBIA. However, it cannot be used for all quadratic sizes. Also, Maass et al. (2023) requires much more power and area when compared to either throughput of the Affine ΔMV architecture.

Table 18 – Comparison with related work for the Affine ΔMV architecture when targeting HD 1080p@60fps videos

	(Sheng et al., 2024)	(Maass et al., 2023)	**Proposed Quad**	**Proposed 16x16**
Tecnology	FPGA	40nm	**40nm**	**40nm**
Frequency (MHz)	100	540	**87**	**64**
Area (mm^2)	-	-	**231178**	**231178**
Area (k Gates)	-	333.89	**245.7**	**245.7**
Total Power (mW)	-	120.37	**13.247**	**9.630**
Throughput	16.7 Meq/s	29.9Meq/s	**33.9Meq/s**	**23.9Meq/s**

6 CONCLUSION

This work presented three dedicated hardware architectures for the Affine ME of the VVC standard. The proposed architectures were described in VHDL and synthesized using the 40nm standard-cell technology of TSMC. This work also analyzed Affine ME complexity and showed that the Affine ME without restriction has too high complexity for real-time encoding, requiring prohibitive power dissipation and/or area utilization to meet necessary throughput. The analysis also evaluated different constraints and their impact on computational complexity and encoding efficiency. Two combinations of these constraints were chosen when focusing on low-power real-time devices.

All architectures presented can process both prediction models, the Affine 4-parameter and 6-parameter, and quadratic PU sizes or 16x16 PU sizes. In the Affine MC architecture, two different Filter Core implementations were presented, one focusing on lower power dissipation, the Power-Efficient, and one focusing on area reduction, the Hardware-Efficient. Both the Affine MC and the Filter Cores applied an MCM approach to substitute the multipliers for sum and shifts at the design of the Vector Generator and the Interpolation Unit, reducing the area and power requirement of the individual components. The Gradient-Based Coefficient Generator Architecture can process multiple PUs simultaneously if the width is smaller than 128, with the maximum number of PUs processed simultaneously being eight of the width of 16 samples. This architecture processes one row of the PU per cycle and takes as many cycles as the height of the PU being processed. The Affine ΔMV is able to process the Affine SLE in eight up to 12 cycles and generate the ΔMV that will be used to generate the next set of MV.

Synthesis results for the Affine MC targeting the processing of UHD 4K videos at 60fps demonstrate an area requirement of 189k Gates with 121mW of power dissipation when running at 598MHz with a BD-Rate increase of 0.80%. For the Gradient-Based Coefficient Generator, the synthesis results targeting UHD 4K videos at 60fps demonstrate an area requirement of 4,700k Gates with 3,114mW of power dissipation when running at 48MHz with a BD-Rate increase of 0.80% or when running at 186MHz the power dissipation of 4963mW with BD-Rate increase of 0.27%. This work is the first

in the literature to present an architecture for the Gradient-Based Coefficients defined by the VVC standard. However, the presented Gradient-Based Coefficient Generator results are impractical for real applications due to its high area requirements and power dissipation. Therefore, strategies for reducing hardware utilization will be explored in future works, mainly reducing the parallelism and reusing of neighboring values. Finally, the synthesis results for the Affine ΔMV targeting the processing of UHD 4K videos at 60fps show an area requirement of 245k Gates with 31mW of power dissipation when running at 236MHz with a BD-Rate increase of 0.80%.

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**, [S.l.], v.11, n.2, p.106–120, 2016.

AZGIN, H.; KALALI, E.; HAMZAOGLU, I. An Approximate Versatile Video Coding Fractional Interpolation Hardware. In: IEEE INTERNATIONAL CONFERENCE ON CONSUMER ELECTRONICS (ICCE), 2020. **Anais...** [S.l.: s.n.], 2020. p.1–4.

AZGIN, H.; MERT, A. C.; KALALI, E.; HAMZAOGLU, I. A reconfigurable fractional interpolation hardware for VVC motion compensation. In: DSD, 2018., 2018. **Anais...** [S.l.: s.n.], 2018. p.99–103.

BITMOVIN. **COVID-19's Impact on streaming video**. Acesso em: 2 de dez. de 2022. Disponível em: <https://go.bitmovin.com/hubfs/Covid-19%20OTT%20streaming_Bitmovin_Analytics_Infographic.pdf>.

BOSSEN, F. et al. **VTM common test conditions and software reference configurations for SDR video**. Teleconferência: [s.n.], 2020. JVET-T2010.

BROSS, B.; CHEN, J.; LIU, S.; WANG, Y.-K. **Versatile Video Coding Editorial Refinements on Draft 10**. JVET-T2001.

BROSS, B. et al. Overview of the versatile video coding (VVC) standard and its applications. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], v.31, n.10, p.3736–3764, 2021.

CADENCE. **RTL Compiler**. accessed in November 22 of 2024. Disponível em: <https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/synthesis.html>.

CANMERT, A.; KALALI, E.; HAMZAOGLU, I. A Low Power Versatile Video Coding (VVC) Fractional Interpolation Hardware. In: CONFERENCE ON DESIGN AND ARCHITECTURES FOR SIGNAL AND IMAGE PROCESSING (DASIP), 2018. **Anais...** [S.l.: s.n.], 2018. p.43–47.

CHEN, J.; YE, Y.; KIM, S. **Algorithm description for Versatile Video Coding and Test Model 11 (VTM 11)**. JVET-T2002.

CISCO. **Cisco Annual Internet Report (2018–2023) White Paper**. Disponível em: <https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html>. Acesso em: 31 de setembro de 2022.

GONÇALVES, P. H. R. **Um esquema rápido baseado em aprendizado de máquina para a predição interquadros do codificador de vídeo VVC**. 2021. Dissertação (Mestrado em Ciência da Computação) — Universidade Federal de Pelotas.

INTEL. **ModelSim*-Intel® FPGA Edition Software**. accessed in November 29 of 2022. Disponível em: <https://www.intel.com.br/content/www/br/pt/software/programmable/quartusprime/model-sim.html>.

JCT-VC. **High Efficiency Video Coding (HEVC)**. Disponível em: <https://hevc.hhi.fraunhofer.de/>. Acesso em: 2022-09-16.

JVET. **Versatile Video Coding (VVC)**. Disponível em: <https://jvet.hhi.fraunhofer.de/>. Acesso em: 2022-09-16.

MAASS, D. et al. High-Throughput Hardware Design for Linear Equation System Solving of VVC Affine Prediction. **Journal of Integrated Circuits and Systems**, [S.l.], v.18, n.3, p.1–11, 2023.

MAASS, D. et al. A Real-Time UHD 4K Hardware for VVC Affine Linear Equation System Solving. In: SBC/SBMICRO/IEEE SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN (SBCCI), 2024., 2024. **Anais...** [S.l.: s.n.], 2024. p.1–5.

MAHDAVI, H.; AZGIN, H.; HAMZAOGLU, I. Approximate versatile video coding fractional interpolation filters and their hardware implementations. **IEEE Trans. Consum. Electron.**, [S.l.], v.67, n.3, p.186–194, 2021.

MUÑOZ, M. M. et al. 4K UHD@ 60fps Design For The VVC Affine Motion Estimation Reconstructor. In: SBC/SBMICRO/IEEE/ACM SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN (SBCCI), 2023., 2023. **Anais...** [S.l.: s.n.], 2023. p.1–6.

MUÑOZ, M. M. et al. Efficient hardware design for the VVC affine motion compensation exploiting multiple constant multiplication. In: IEEE COMPUTER SOCIETY ANNUAL SYMPOSIUM ON VLSI (ISVLSI), 2023., 2023. **Anais...** [S.l.: s.n.], 2023. p.1–6.

MUÑOZ, M. M. et al. Hardware design for the affine motion compensation of the vvc standard. In: IEEE 14TH LATIN AMERICA SYMPOSIUM ON CIRCUITS AND SYSTEMS (LASCAS), 2023., 2023. **Anais...** [S.l.: s.n.], 2023. p.1–4.

MUÑOZ, M. M. et al. Design Space Exploration of Real-Time VVC Affine Motion Estimation Reconstructor. **Design & Test**, [S.l.], Under Review (28-Aug-2024).

PAKDAMAN, F.; ADELIMANESH, M. A.; GABBOUJ, M.; HASHEMI, M. R. Complexity Analysis Of Next-Generation VVC Encoding And Decoding. In: IEEE INTERNATIONAL CONFERENCE ON IMAGE PROCESSING (ICIP), 2020., 2020. **Anais...** [S.l.: s.n.], 2020. p.3134–3138.

PARK, S.-H.; KANG, J.-W. Fast affine motion estimation for versatile video coding (VVC) encoding. **IEEE Access**, [S.l.], v.7, p.158075–158084, 2019.

PERLEBERG, M. et al. An UHD 4K@ 120fps Hardware for the VVC Prediction Refinement with Optical Flow. In: SBC/SBMICRO/IEEE/ACM SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN (SBCCI), 2023., 2023. **Anais...** [S.l.: s.n.], 2023. p.1–6.

PORTO, M. S. **Desenvolvimento algorítmico e arquitetural para a estimação de movimento na compressão de vídeo de alta definiço**. 2012. Tese (Doutorado em Ciência da Computação) — UFRGS.

SANDVINE. **2022 Global Internet Phenomena Report**. Acesso em: 2 de dez. de 2022. Disponível em: <https://www.sandvine.com/global-internet-phenomena-report-2022>.

SHENG, Q. et al. Fast Linear Equation Solving Algorithm and its Pipelined Hardware Architecture Design for VVC Affine Motion Estimation. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], 2024.

SILVA, G. G.; SIQUEIRA, Í. G.; GRELLERT, M.; DINIZ, C. M. Approximate Hardware Architecture for Interpolation Filter of Versatile Video Coding. **JICS**, [S.l.], v.16, n.2, p.1–8, 2021.

SIQUEIRA, Í.; CORREA, G.; GRELLERT, M. Rate-distortion and complexity comparison of HEVC and VVC video encoders. In: IEEE 11TH LATIN AMERICAN SYMPOSIUM ON CIRCUITS & SYSTEMS (LASCAS), 2020., 2020. **Anais...** [S.l.: s.n.], 2020. p.1–4.

SULLIVAN, G. J.; OHM, J.-R.; HAN, W.-J.; WIEGAND, T. Overview of the high efficiency video coding (HEVC) standard. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], v.22, n.12, p.1649–1668, 2012.

TARANTO, C. **Simplified affine motion estimation algorithm and architecture for the versatile video coding standard**. 2022. Tese (Doutorado em Ciência da Computação) — Politecnico di Torino.

TUMMELTSHAMMER, P.; HOE, J. C.; PUSCHEL, M. Time-Multiplexed Multiple-Constant Multiplication. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, [S.l.], v.26, n.9, p.1551–1563, 2007.

Appendices

APPENDIX A – LIST OF PUBLICATIONS DURING THIS MASTERS DEGREE

During this master's degree, a total of seven papers were published in conferences and journals. Two of them (SBCCI 2023) were invited to extended version submissions to a journal, and these submissions are still under review. There is also a submission to ISCAS 2025 that is still under review.

Papers published in conferences:

- **Title:** A Power-Efficient Architecture for LES Solving and △MV Calculation of VVC Affine Prediction
  Authors: Denis Maass, Marcello Muñoz, Murilo Perleberg, Luciano Agostini, Marcelo Porto
  Journal: IEEE International Symposium on Circuits and Systems (ISCAS)
  Conference: Oct-2024 (Submitted, Under review)

- **Title:** A Real-Time UHD 4K Hardware for VVC Affine Linear Equation System Solving
  Authors: Denis Maass, Marcello Munoz, Murilo Perleberg, Luciano Agostini, Marcelo Porto
  Conference: 2024 37th SBC/SBMicro/IEEE Symposium on Integrated Circuits and Systems Design (SBCCI)

- **Title:** 4K UHD@ 60fps Design For The VVC Affine Motion Estimation Reconstructor
  Authors: Marcello M Muñoz, Denis Maass, Murilo Perleberg, Luciano Agostini, Guilherme Correa, Marcelo Porto
  Conference: 2023 36th SBC/SBMicro/IEEE/ACM Symposium on Integrated Circuits and Systems Design (SBCCI)

- **Title:** An UHD 4K@ 120fps Hardware for the VVC Prediction Refinement with Optical Flow
  Authors: Murilo Perleberg, Marcello M Muñoz, Denis Maass, Vladimir Afonso, Luciano Agostini, Marcelo Porto

Conference: Conferência 2023 36th SBC/SBMicro/IEEE/ACM Symposium on Integrated Circuits and Systems Design (SBCCI)

- **Title:** Efficient hardware design for the VVC Affine motion compensation exploiting multiple constant multiplications
  Authors: Marcello M Muñoz, Denis Maass, Murilo Perleberg, Luciano Agostini, Marcelo Porto
  Conference: 2023 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)

- **Title:** Hardware design for the Affine motion compensation of the VVC standard
  Authors: Marcello M Muñoz, Denis Maass, Murilo Perleberg, Luciano Agostini, Marcelo Porto
  Conference: 2023 IEEE 14th Latin America Symposium on Circuits and Systems (LASCAS)

- **Title:** Transistor Reordering for Electrical Improvement in CMOS Complex Gates
  Authors: Marcello M Muñoz, Henrique Kessler, Marcelo Porto, Vinícius V Camargo
  Conference: 2022 35th SBC/SBMicro/IEEE/ACM Symposium on Integrated Circuits and Systems Design (SBCCI)

Papers published in journals:

- **Title:** Design Space Exploration of Real-Time VVC Affine Motion Estimation Reconstructor
  Authors: Marcello M. Muñoz, Denis Maass, Murilo Perleberg, Luciano Agostini, Guilherme Corrêa, Marcelo Porto
  Journal: IEEE Design & Test
  Publication Date: Aug-2024 (Submitted, Under review)

- **Title:** An Efficient Hardware Design for the VVC Prediction Refinement with Optical Flow
  Authors: Murilo Perleberg, Marcello Muñoz, Denis Maass, Vladimir Afonso, Luciano Agostini, Marcelo Porto
  Journal: IEEE Design & Test
  Publication Date: Aug-2024 (Submitted, Under review)

- **Title:** High-Throughput Hardware Design for Linear Equation System Solving of VVC Affine Prediction
  Authors: Denis Maass, Marcello Muñoz, Murilo Perleberg, Luciano Agostini, Marcelo Porto
  Conference: Journal of Integrated Circuits and Systems (JICS)
  Publication Date: 2023/12/28