Approximate Computing based Video Compression Hardware

by

Berke Ayrancıoğlu

Submitted to the Graduate School of Engineering and Natural Sciences

in partial fulfillment of
the requirements for the degree of
Master of Sciences

Sabancı University

July 2022

To the World peace and the brighter days to come…

ACKNOWLEDGEMENTS

First of all, I would like to thank my supervisor, Dr. İlker Hamzaoğlu for all his support, honest and fair approach to every situation and his patience. I consider myself lucky for being his student since his advices will accompany me in every aspect of life in addition to academy and research. I think I wouldn't be able to keep my motivation as high without him.

I want to thank past and present members of "System-on-Chip Design and Test Lab"; Hasan Azgın, Hossein Mahdavi and Waqar Ahmad for their friendship and support. As the youngest and the least experienced member of the SoC Lab, it would be much harder for me to arrive where I am here today without their help. Working and publishing together with Waqar Ahmad has been a remarkable experience for me. I would like to thank him especially, for his modest support and mentorship.

My greatest thankfullness is to my dad, my mother and my sister. This thesis is dedicated with gratitude to my family. I will always be thankful for their support.

Finally, I want to thank Sabanci University for supporting me with scholarship throughout my studies. I also want to thank Scientific and Technological Research Council of Turkey (TUBITAK) for supporting this thesis in part under contract number 118E134.

Approximate Computing based Video Compression Hardware

Berke Ayrancıoğlu

Electronics Engineering, MS Thesis, 2022

Thesis Supervisor: Assoc. Prof. İlker HAMZAOĞLU

Keywords: Video Compression, Affine Motion Estimation, Approximate Computing, Digital Hardware, FPGA

ABSTRACT

Approximate computing trades off accuracy to improve area, power, speed of digital hardware. Many computationally intensive applications such as video encoding and video processing are error tolerant by nature due to the limitations of human visual perception. Therefore, approximate computing can be used to improve area, power, speed of digital hardware implementations of these error tolerant applications. In this thesis, a low error approximate absolute difference hardware is proposed and impact of using approximate circuits in H.264 motion estimation (ME) hardware is assessed.

There is a need for more video compression with less quality loss due to significant increase in spatial and temporal video resolutions. Therefore, versatile video coding (VVC) video compression standard is recently developed. It is more computationally complex than previous video compression standards H.264 and high efficiency video coding (HEVC). ME is the most computationally complex part of video compression standards. VVC standard uses affine ME (AME) which is not used in previous video compression standards. AME achieves higher video compression at the expense of much more computational complexity. In this thesis, a novel VVC AME hardware is proposed.

Yaklaşık Hesaplama Temelli Video Sıkıştırma Donanımları

Berke Ayrancıoğlu

Elektronik Mühendisliği, Yüksek Lisans Tezi, 2022

Tez Danışmanı: Doç. Dr. İlker HAMZAOĞLU

Anahtar Kelimeler: Video Sıkıştırma, Afin Hareket Tahmini, Yaklaşık Hesaplama, Sayısal Donanım, FPGA

Özet

Yaklaşık hesaplama sayısal donanımın alanını ve tükettiği gücü azaltmak ve hızını artırmak için işlem doğruluğunu feda eder. İnsan görüşünün sınırlı olması nedeniyle video kodlama ve video işleme gibi hesaplama karmaşıklığı yüksek bazı uygulamalar hataya toleranslıdır. Bu nedenle, bu tür uygulamalarda işlem doğruluğu alan, güç ve hız kazanımları için feda edilebilir. Bu tezde bir düşük hatalı yaklaşık mutlak fark donanımı önerildi ve H.264 hareket tahmini (HT) donanımında yaklaşık devreleri kullanmanın etkileri belirlendi.

Uzamsal ve zamansal video çözünürlüğünün artması nedeniyle daha az kalite kaybıyla daha çok video sıkıştırmaya ihtiyaç vardır. Bu nedenle, yakın zamanda Çok Yönlü Video Kodlama (VVC) standardı geliştirildi. VVC'nin hesaplama karmaşıklığı önceki video sıkıştırma standartları H.264 ve HEVC'nin hesaplama karmaşıklıklarından daha yüksektir. HT video sıkıştırma standartlarının en yüksek hesaplama karmaşıklığı olan bölümüdür. VVC standardında daha önceki video sıkıştırma standartlarında kullanılmamış olan afin hareket tahmini (AHT) kullanılmaktadır. AHT daha çok video sıkıştırma yapmaktadır ama çok daha fazla hesaplama karmaşıklığına sahiptir. Bu tezde bir özgün VVC AHT donanımı önerildi.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AD	Absolute Difference
AME	Affine Motion Estimation
BRAM	Block Ram
DSP	Digital Signal Processor
FPGA	Field Programmable Gate Array
FPS	Frame Per Second
HD	High Definition
HEVC	High Efficiency Video Coding
ME	Motion Estimation
MV	Motion Vector
PSNR	Peak Signal to Noise Ratio
SAD	Sum of Absolute Differences
VVC	Versatile Video Coding

CHAPTER I

INTRODUCTION

Approximate computing trades-off accuracy with speed, area and power consumption (Mert, Azgin, Kalali, & Hamzaoglu, 2019), (Azgin, Kalali, & Hamzaoglu, 2020). It is used for error-tolerant applications such as video processing and compression which can tolerate inaccurate results because of the limitations of human visual perception (Kalali & Hamzaoglu, Approximate HEVC Fractional Interpolation Filters and Their Hardware Implementations, 2018). Approximate hardware can achieve better performance, area and power consumption than accurate hardware while providing acceptable quality for error tolerant applications (Xu, Mytkowicz, & Kim, 2015), (Froehlich, Große, & Drechsler, 2018), (Arifeen, Hassan, Moradian, & Lee, 2016).

A variety of approximate circuits, ranging from system level designs (Gillani, et al., 2019), (Kalali & Hamzaoglu, An approximate HEVC intra angular prediction hardware, 2020), (Ayhan & Altun, 2019) (Ahmad & Hamzaoglu, An efficient approximate sum of absolute differences hardware for FPGAs, 2021) to basic arithmetic circuits (Jiang H. , Liu, Liu, Lombardi, & Han, 2017), have been proposed in the literature. Adders are used in most digital hardware, not only for binary addition but also for other binary arithmetic operations such as subtraction, multiplication, and division (Mert, Azgin, Kalali, & Hamzaoglu, 2019), (Van Toan & Lee, 2020), (Chen, Han, Liu, Montuschi, & Lombardi, 2018).

Video coding is very computationally complex and the growing demand for higher spatial and temporal video resolutions has led to development of more computationally complex video coding standards. Motion estimation (ME) is the most computationally complex module in the video encoder hardware. Block matching ME is used in H.264,

HEVC and VVC video coding standards to remove temporal redundancies in video sequences. For each block in the current frame, block matching ME determines the best matching reference block in a search window in the previous frame based on a distortion metric.

Sum of absolute differences (SAD) is the most commonly used distortion metric for block matching ME. SAD value between a current block and a reference block of size HxW pixels is calculated as

$$SAD = \sum_{x=1}^{H} \sum_{y=1}^{W} |Cur(x,y) - Ref(x,y)| \tag{1.1}$$

where Cur(x,y) is the value of the pixel in (x,y) position of the current block and Ref(x,y) is the value of the pixel in (x,y) position of the reference block. Number of search locations that should be searched for each block in the current frame depends on ME algorithm and size of the search window. For example, for full search ME algorithm with 16x16 search window, 256 search locations should be searched. For H.264, the largest coding block size is 16x16 and it has 41 sub-blocks. 256 absolute difference and 265 addition operations are required to calculate 41 SAD values of a 16x16 block in the current frame for one search location. 133376 arithmetic operations are required to calculate SAD values for 256 search locations.

Affine transformation is a type of geometric transformation in which collinearity is preserved. In other words, if a group of points abide on a line before the affine transform, they stay on the same line after the affine transform. While retaining collinearity, affine transform incorporates translation, rotation, zoom and shearing. Fig. 1.1 and 1.2 show the shear and rotation around the origin and formulae 1.2 and 1.3 show the related matrices.

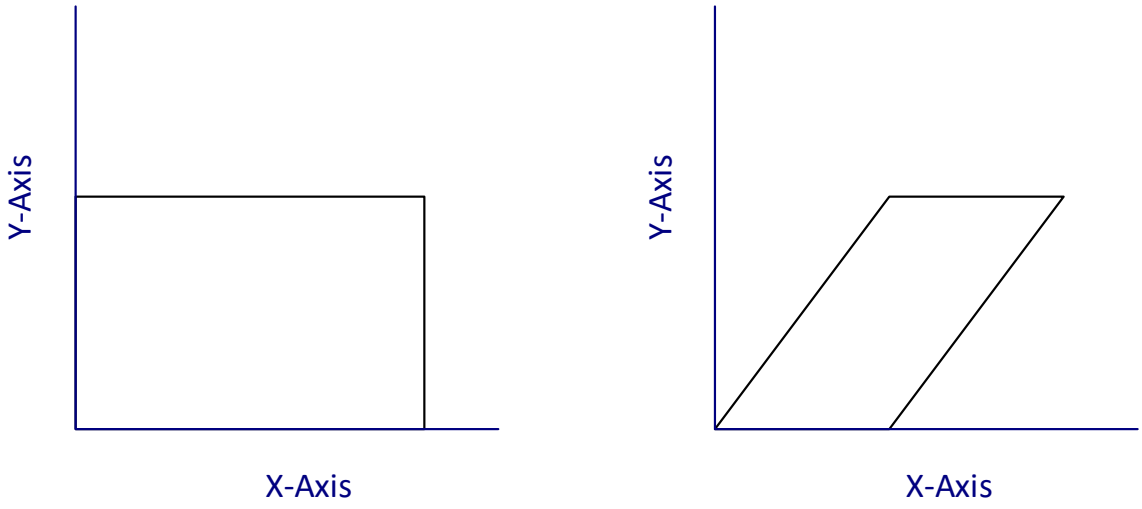


Figure 1.1 Shear

$$M_S = \begin{bmatrix} 1 & b \\ 0 & 1 \end{bmatrix} \tag{1.2}$$

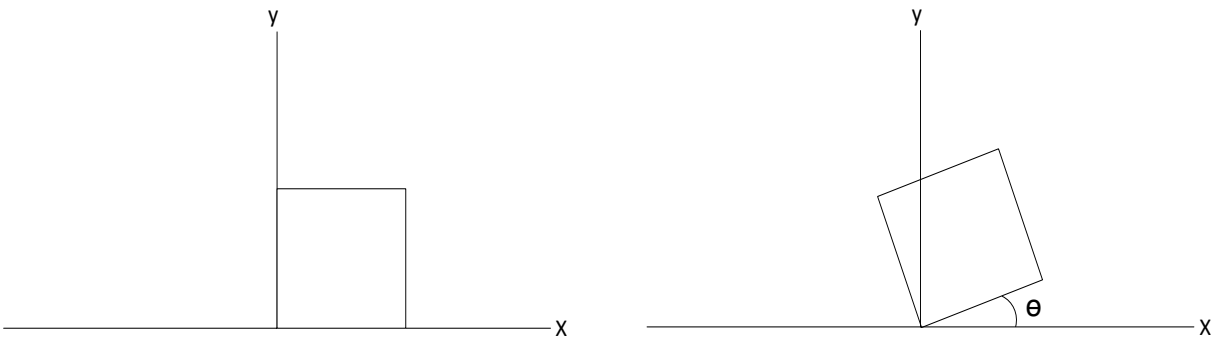


Figure 1.2 Rotation

$$M_R = R(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \tag{1.3}$$

Two-dimensional transform is performed using formula 1.4. Therefore, rotation transform is performed using formulae 1.5 and 1.6.

$$\begin{matrix} x' \\ y' \end{matrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{matrix} x \\ y \end{matrix} \tag{1.4}$$

$$x' = x\cos(\theta) - y\sin(\theta) \tag{1.5}$$

$$y' = x\sin(\theta) + y\cos(\theta) \tag{1.6}$$

H.264 and HEVC video compression standards only use translational ME. VVC video compression standard uses affine ME (AME) as well. AME takes rotation, zooming and shearing of blocks into account during block matching ME. AME achieves higher video compression than translational ME at the expense of much more computational complexity (Seferidis & Ghanbari, 1993), (Choi & Kim, 2019), (Adhuran, Fernando, Kulupana, & Blasi, 2021).

1.1 Thesis Contributions

A low error approximate absolute difference (LAD_X) hardware is proposed. The proposed hardware has lower maximum and average error values and higher accuracy than the absolute difference hardware in the literature. It has similar performance with and smaller area than them. H.264 ME hardware using LAD_X hardware performs higher quality ME than H.264 ME hardware using the approximate absolute difference hardware in the literature. It has similar performance with and smaller area than them.

Several approximate circuits from the literature are integrated to an H.264 ME hardware and their impact is assessed. The approximate adder proposed by (Ahmad, 2021) achieved up to 10% power reduction in ME hardware while providing better quality than the other approximate circuits.

A novel VVC AME hardware is proposed. To the best of our knowledge, it is the first VVC AME hardware in the literature. It implements 4-parameters affine transform and uses a novel pixel storage method which decreases the number of memory accesses. Since AME has very high computational complexity, to achieve real time performance, the proposed VVC AME hardware should be used only for parts of the current frame containing affine motion.

1.2 Thesis Organization

The rest of the thesis is organized as follows.

Chapter II explains the proposed approximate absolute difference hardware, LAD_X. It compares LAD_X and the approximate circuits in the literature. It presents the impact of using them in H.264 ME hardware.

Chapter III explains the approximate adder proposed by (Ahmad, 2021). It presents the impact of using this approximate adder and other approximate adders in the literature in H.264 ME hardware.

Chapter V first explains VVC AME algorithm. It then explains the proposed VVC AME hardware and the proposed novel pixel storage method. It finally presents the experimental results.

Chapter VI presents the conclusions and the future work.

CHAPTER II

Low Error Approximate Absolute Difference Hardware

In this thesis, we propose low error approximate absolute difference (LAD_X) hardware to increase speed and to decrease area and power consumption of absolute difference (AD) hardware at the expense of a slight quality loss. LAD_X hardware is compared with the following approximate AD hardware in the literature; novel approximate absolute difference hardware (NAAD) (Mert, Azgin, Kalali, & Hamzaoglu, 2019) and the approximate AD hardware built by using the following approximate adders for the subtraction operation in three different accurate AD hardware; lower part OR adder (LOA) (Porto R. , Agostini, Zatt, Roma, & Porto, 2019) almost correct adder (ACA1) (Verma, Brisk, & Ienne, 2008), accuracy configurable adder (ACA2) (Kahng & Kang, 2012), and generic accuracy configurable adder (GeAr) (Shafique, Ahmad, Hafiz, & Henkel, 2015). LAD_X hardware has lower maximum and average error, and higher accuracy than these approximate AD hardware in the literature. It has similar performance with and smaller area than them. The H.264 ME hardware using LAD_X hardware performs higher quality ME than the H.264 ME hardware using these approximate AD hardware in the literature. It has similar performance with and smaller area than them.

2.1 Proposed Approximate AD Hardware

LAD_2 and LAD_4 hardware are shown in Fig. 2.1 and Fig. 2.2, respectively. LAD_X hardware is composed of a subtractor, XOR gates, an adder, and OR gates. It works as follows. First, the inputs A and B are subtracted. Then, the sign bit of the subtraction result is XOR'ed with the other 8 bits of the subtraction result. Then, the sign bit is added to the least significant X bits. Therefore, instead of calculating 2's complement of the entire subtraction result, 2's complement of its least significant X bits is calculated. This breaks the carry chain in the addition operation.

Finally, the most significant bit (MSB) of the addition result is OR'ed with the other bits of the addition result. Therefore, if the MSB of the addition result is 0, the output of LAD_X hardware will be the same as the accurate AD result. If the MSB of the addition

result is 1, least significant X bits of the output of LAD_X hardware will be 1. Since the MSB of the addition result is not propagated to next bit position, the output of LAD_X hardware will be 1 less than the accurate AD result, instead of being 2^X less than the accurate AD result.

The output of LAD_X hardware is the same as the accurate AD result when the subtraction result is positive. When the subtraction result is negative, the output of LAD_X hardware is the same as the accurate AD result for $(2^X - 1)$ of every 2^X inputs, and it is 1 less than the accurate AD result for the remaining input. Therefore, the maximum error of LAD_X hardware is 1. Its accuracy and average error can be calculated as shown in (2.1) and (2.2), respectively.

$$Accuracy = 100 \times \left(1 - \frac{1}{2^{X+1}}\right) \tag{2.1}$$

$$Average\ Error = \frac{1}{2^{X+1}} \tag{2.2}$$

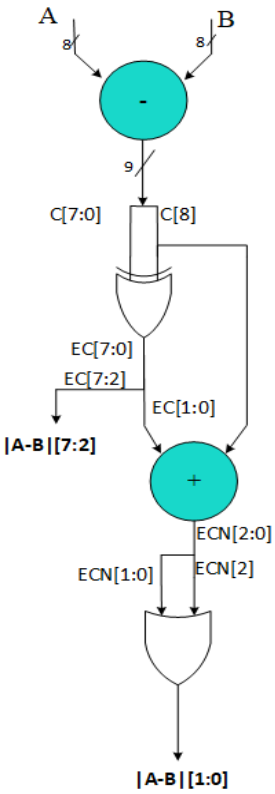


Figure 2.1 Proposed low error approximate absolute difference (LAD_2) hardware.

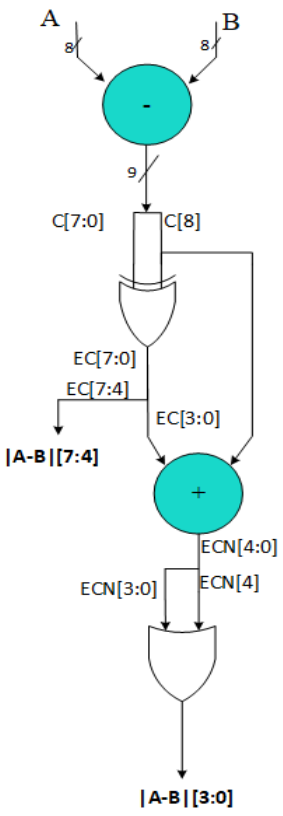


Figure 2.2 Proposed low error approximate absolute difference (LAD_4) hardware.

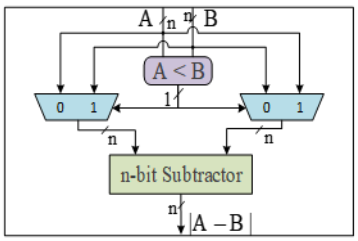


Figure 2.3 Accurate_AD_1

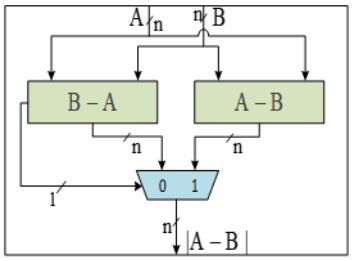


Figure 2.4 Accurate_AD_2

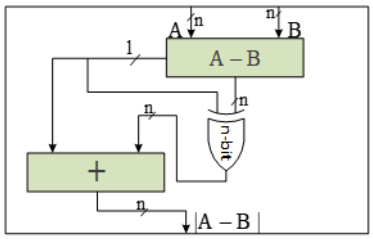


Figure 2.5 Accurate_AD_3

Accuracy is the percentage of inputs for which LAD_X hardware gives accurate AD result. Average error is the average of errors LAD_X hardware gives for all inputs. As it can be seen from (2.1) and (2.2), accuracy and average error of LAD_X hardware are independent from its bit width. Its maximum error is also independent from its bit width. In other words, the proposed N-bit LAD_X hardware has the same accuracy, average error, and maximum error for all N. In addition, as the bit width increases, ratio of the maximum input value to maximum and average error increases, and therefore, relative impact of the error decreases. Therefore, LAD_X hardware is scalable.

2.2 Quality Results

LAD_X hardware is compared with 20 approximate AD hardware in the literature. 4 approximate AD hardware are 4 configurations of novel approximate absolute difference hardware (NAAD) (Mert, Azgin, Kalali, & Hamzaoglu, 2019). 16 approximate AD hardware are built by using the following approximate adders for the subtraction operation in the 3 accurate AD hardware shown in Fig. 2.3, Fig 2.4 and Fig. 2.5; lower part OR adder (LOA) (Porto R. , Agostini, Zatt, Roma, & Porto, 2019), almost correct adder (ACA1) (Verma, Brisk, & Ienne, 2008), accuracy configurable adder (ACA2) (Kahng & Kang, 2012), and generic accuracy configurable adder (GeAr) (Shafique, Ahmad, Hafiz, & Henkel, 2015).

NAAD is configured for 0, 1, 2, and 3 approximate bits. LOA is configured for 1, 2, 3, and 4 approximate bits. ACA1 and ACA2 are configured for 4 approximate bits. GeAr is configured as GeAr_R1_P2 ($R = 1,\ P = 2$) and GeAr_R2_P4 ($R = 2,\ P = 4$). An approximate adder is used as an approximate subtractor by taking 1's complement of its second data input and setting its carry-in input to 1.

Maximum error, average error and accuracy results for 8-bit and 16-bit approximate AD hardware are shown in Table 2.I. In the table, A1_ACA1 approximate AD hardware is

built by using the ACA1 approximate adder for the subtraction operation in the Accurate_AD_1 hardware. A2_ACA1 approximate AD hardware is built by using the ACA1 approximate adder for the subtraction operation in the Accurate_AD_2 hardware. A3_ACA1 approximate AD hardware is built by using the ACA1 approximate adder for the subtraction operation in the Accurate_AD_3 hardware.

8-bit LAD_X hardware has lower maximum error and average error than the other 8-bit approximate AD hardware. 8-bit approximate AD hardware which uses GeAr_R2_P4 approximate adder has the highest accuracy. 8-bit LAD_X hardware has higher accuracy than the other 8-bit approximate AD hardware.

Since LAD_X, NAAD and approximate AD hardware which uses LOA approximate adder are scalable, their quality results for 8-bit and 16-bit AD hardware are the same. Since approximate AD hardware which use ACA1, ACA2 and GeAr approximate adders are not scalable, their quality results for 16-bit AD hardware are much worse than 8-bit AD hardware. Therefore, 16-bit LAD_X hardware has lower maximum error and average error, and higher accuracy than the other 16-bit approximate AD hardware.

Impact of using LAD_X hardware and 8 approximate AD hardware in the literature in H.264 full search motion estimation (FSME) hardware (Kalaycioglu, Ulusel, & Hamzaoglu, 2009) is analyzed. The FSME hardware proposed in (Kalaycioglu, Ulusel, & Hamzaoglu, 2009) works for 8-bit pixel values. It is modified to work with 16-bit pixel values. For every approximate AD hardware, 256 accurate AD hardware in the Verilog RTL codes of 8-bit FSME hardware and 16-bit FSME hardware are replaced with that approximate AD hardware.

Quality results of 8-bit and 16-bit H.264 FSME hardware are shown in Table 2.2. Mean square error (MSE) between the original frames and the frames reconstructed by 8-bit and 16-bit FSME hardware are calculated. Peak signal to noise ratio (PSNR) values are calculated as shown in (2.3). The maximum value of an 8-bit pixel is 255, whereas the maximum value of a 16-bit pixel is 65535. Therefore, PSNR values of 16-bit FSME hardware are much higher than PSNR values of 8-bit FSME hardware.

$$PSNR = 10 \times log_{10}\left(\frac{(Max.Pixel\,Value)^2}{MSE}\right) \tag{2.3}$$

Since LAD_X hardware has very low maximum error and average error, and very high accuracy, MSE and PSNR values of 8-bit and 16-bit H.264 FSME hardware which use LAD_X hardware are the same as the MSE and PSNR values of accurate 8-bit and 16-bit

H.264 FSME hardware. This means that LAD_X hardware did not cause any quality loss for H.264 FSME hardware.

PSNR values of 8-bit and 16-bit H.264 FSME hardware which use LAD_X hardware are higher than the PSNR values of 8-bit and 16-bit H.264 FSME hardware which use the other approximate adders. This means that LAD_X hardware gives higher quality results than the other approximate adders when they are used in H.264 FSME hardware.

2.3 Implementation Results

LAD_X hardware, 20 approximate AD hardware in the literature, and 3 accurate AD hardware are implemented using Verilog HDL. Verilog RTL codes are synthesized and implemented on Xilinx XC6VLX130T FF1156 FPGA using Xilinx ISE 14.7. FPGA implementations are verified by post-implementation timing simulations. Power consumptions of the FPGA implementations are estimated using Xilinx XPower Analyzer. Post-implementation timing simulations are performed at 100 MHz and signal activities are stored into VCD files. Power consumptions of the FPGA implementations are estimated using these VCD files.

FPGA implementation results of AD hardware are shown in Table 2.3. Number of LUTs used by 8-bit LAD_X hardware is less than the number of LUTs used by the other 8-bit AD hardware except 8-bit NAAD hardware. Frequency of 8-bit LAD_X hardware is higher than the frequency of all the accurate AD hardware and most of the approximate AD hardware. A2_GeAr_R1_P2 hardware has the highest frequency among all the 8-bit AD hardware. Power consumption of 8-bit LAD_X hardware is lower than the power consumptions of all the other 8-bit AD hardware.

As expected, as bit width increases, area and power consumptions of all AD hardware increase, and their frequencies decrease. However, as bit width increases, frequency, area, and power consumption of LAD_X hardware change less than the frequency, area, and power consumptions of the other AD hardware.

FPGA implementation results of H.264 FSME hardware are shown in Table 2.4. Numbers of LUTs used by 8-bit and 16-bit H.264 FSME hardware which use LAD_X hardware are less than the numbers of LUTs used by the other 8-bit and 16-bit H.264 FSME hardware, except the ones which use NAAD_3 hardware, respectively. Frequencies of 8-bit H.264 FSME hardware are similar. Frequencies of 16-bit H.264 FSME hardware are similar.

This is because AD hardware is not in the critical path of H.264 FSME hardware. As expected, as bit width increases, frequencies of H.264 FSME hardware slightly decrease.

Table 2.1 Quality Results of Absolute Difference Hardware

	8-bit			16-bit		
	Maximum Error	Average Error	Accuracy (%)	Maximum Error	Average Error	Accuracy (%)
LAD_1	1	0.25	75	1	0.25	75
LAD_2	1	0.125	87.5	1	0.125	87.5
LAD_3	1	0.062	93.75	1	0.062	93.75
LAD_4	1	0.031	96.87	1	0.031	96.87
NAAD_0	1	0.5	50	1	0.5	50
NAAD_1	2	0.5	75	2	0.5	75
NAAD_2	4	1	62.5	4	1	62.5
NAAD_3	8	2	56.25	8	2	56.25
A3_LOA_1	2	0.74	50	2	0.74	50
A3_LOA_2	6	1.85	31.6	6	1.85	31.6
A3_LOA_3	14	4.1	17.6	14	4.1	17.6
A3_LOA_4	30	8.6	9.5	30	8.6	9.5
A1_ACA1_Q4	128	4.88	90.7	34944	1364.	68.4
A2_ACA1_Q4	128	7.22	87.8	34944	1962.	65.95
A3_ACA1_Q4	128	7.17	87.8	34944	1962.	65.95
A1_ACA2_Q4	64	5.906	84.17	17472	1637.	54.3
A2_ACA2_Q4	64	7.168	81.25	17472	1962.	52.2
A3_ACA2_Q4	64	7.17	81.25	17472	1962.	52.2
A1_GeAr_R1_P	144	10.17	75.48	37448	2730	41.4
A2_GeAr_R1_P	144	14.16	69.92	37448	3754	38.2
A3_GeAr_R1_P	144	14.17	69.92	37448	3754	38.2
A1_GeAr_R2_P	64	1.125	98.24	16640	409	89.1
A2_GeAr_R2_P	64	1.48	97.65	16640	506	88.4
A3_GeAr_R2_P	64	1.48	97.65	16640	506	88.4

Table 2.2 Quality Results of H.264 Motion Estimation Hardware

	8-bit		16-bit	
	PSN	MS	PSNR	MS
Accurate_AD_	36.2	16.	84.47	16.3
LAD_2	36.2	16.	84.47	16.3
LAD_4	36.2	16.	84.47	16.3
NAAD_1	36.1	16.	68.26	16.6
NAAD_3	35.2	19.	83.46	19.3
A3_LOA_2	35.8	17.	84	17.6
A3_LOA_4	32.6	35.	80.8	35.6
A3_ACA1_Q4	29.8	71.	72.47	320.
A3_ACA2_Q4	31.1	50.	74	205.
A3_GeAr_R1_	30.9	54.	75.93	122.
A3_GeAr_R2_	32.2	42.	69.53	606

Table 2.3 FPGA Implementation Results of Absolute Difference

	8-bit			16-bit		
	LUTs	Frequency (MHz)	Power (mW)	LUTs	Frequency (MHz)	Power (mW)
Accurate_AD_	2	461	0.39	64	384	0.44
Accurate_AD_	2	589	0.45	50	534	0.53
Accurate_AD_	2	572	0.4	34	458	0.37
LAD_1	1	653	0.28	34	577	0.33
LAD_2	1	654	0.3	38	591	0.37
LAD_3	1	638	0.31	34	588	0.39
LAD_4	1	645	0.37	34	581	0.44
NAAD_0	1	653	0.3	38	588	0.39
NAAD_1	1	657	0.3	36	588	0.35
NAAD_2	1	657	0.34	32	589	0.33
NAAD_3	1	662	0.22	35	591	0.33
A3_LOA_1	2	543	0.37	42	431	0.36
A3_LOA_2	2	536	0.3	37	445	0.32
A3_LOA_3	2	615	0.3	34	440	0.3
A3_LOA_4	2	620	0.25	34	443	0.27
A1_ACA1_Q4	3	432	0.43	71	400	0.49
A2_ACA1_Q4	3	641	0.5	74	574	0.58
A3_ACA1_Q4	2	604	0.57	46	515	0.43
A1_ACA2_Q4	3	449	0.45	70	396	0.48
A2_ACA2_Q4	3	666	0.52	62	560	0.57
A3_ACA2_Q4	2	649	0.5	49	485	0.47
A1_GeAr_R1_	2	498	0.43	61	424	0.47
A2_GeAr_R1_	2	771	0.46	50	722	0.51
A3_GeAr_R1_	2	645	0.51	33	566	0.39
A1_GeAr_R2_	3	448	0.37	81	400	0.5
A2_GeAr_R2_	3	608	0.46	90	571	0.54
A3_GeAr_R2_	2	558	0.42	55	479	0.38

Table 2.4 FPGA Implementation Results of H.264 Motion Estimation Hardware

	8-bit		16-bit	
	LUTs	Frequency (MHz)	LUTs	Frequency (MHz)
Accurate_AD_	108	204	213	190
LAD_2	105	207	189	195
LAD_4	105	204	189	196
NAAD_1	106	205	189	187
NAAD_3	103	203	187	197
A3_LOA_2	110	200	207	185
A3_LOA_4	107	206	203	186
A3_ACA1_Q4	117	204	202	196
A3_ACA2_Q4	119	207	190	198
A3_GeAr_R1_	122	205	200	194
A3_GeAr_R2_	117	203	258	188

CHAPTER III

Comparison of Approximate Circuits for H.264 Motion Estimation

Approximate hardware can achieve better performance, area and power consumption than accurate hardware while providing acceptable quality for error tolerant applications (Xu, Mytkowicz, & Kim, 2015), (Froehlich, Große, & Drechsler, 2018), (Kalali & Hamzaoglu, Approximate HEVC Fractional Interpolation Filters and Their Hardware Implementations, 2018). Video coding can tolerate small errors (Kalali & Hamzaoglu, Approximate HEVC Fractional Interpolation Filters and Their Hardware Implementations, 2018). Therefore, approximate computing can be used for block matching ME.

Approximate adders proposed in literature can be broadly classified into two categories. (1) Approximation of 1-bit full adder (Gupta, Mohapatra, Park, Raghunathan, & Roy, 2011), (Mahdiani, Ahmadi, Fakhraie, & Lucas, April, 2010). These adders simplify 1-bit full adder logic. They divide n-bit addition into two parts, approximate part for least significant bits (LSBs) and accurate part for most significant bits (MSBs). They use the approximate 1-bit full adder in the approximate part. (2) Segmented adders (Jiang H. , Liu, Liu, Lombardi, & Han, August 2017), (Shafique, Ahmad, Hafiz, & Henkel, 2015). These adders break the carry chain by dividing n-bit addition into several smaller fixed size overlapping sub-adders working in parallel. They have higher speed than accurate adders. However, they consume more area and power than accurate adders.

In this thesis, we assessed the impact of using approximate circuits in H.264 ME hardware. Approximate circuits are used in the absolute difference operation and adder tree in H.264 ME hardware. The approximate adder proposed by (Ahmad, 2021) achieved up to 10% power reduction in ME hardware while providing better quality than the other approximate circuits. Traditional bit truncation achieved the largest area and power reductions in ME hardware at the expense of more quality loss than the proposed approximate adder. Generic accuracy reconfigurable adder (GeAr) segmented approximate adder (Shafique, Ahmad, Hafiz, & Henkel, 2015) had the worst quality, area and power consumption results.

3.1 Approximate Circuits

The proposed 1-bit approximate full adder is shown in Fig. 3.1 (a) (Ahmad, Ayrancioglu, & Hamzaoglu, Comparison of Approximate Circuits for H.264 and HEVC Motion Estimation, 2020). It generates carry-out (C_{out}) output without considering the effect of carry-in (C_{in}) input. Carry-out is 1 whenever one or both inputs A and B are 1. Error is generated in the following two cases; (A = 0, B = 1 , C_{in} = 0 → S = 0, C_{out} = 1) and (A = 1, B = 0 , C_{in} = 0 → S = 0, C_{out} = 1). An important property of the proposed approximate full adder is that it generates accurate outputs when carry-in input is 1. Since carry-in for the subtraction in absolute difference operation is always 1, this property is very useful for absolute difference operation. Maximum error magnitude of the proposed approximate full adder is 1.

Many approximate arithmetic circuits are proposed in the literature. The approximate circuits used in this thesis are selected based on the analysis results reported in the literature. In (El-Harouni, et al., 2017), two 1-bit approximate full adders from (Gupta, Mohapatra, Park, Raghunathan, & Roy, 2011) are determined to give the best performance for ME. In this thesis, these 1-bit full adders are referred to as IMPACT-1 and IMPACT-2. They are shown in Fig. 3.1 (c) and Fig. 3.1 (d), respectively. An approximate n-bit subtractor can be designed using IMPACT-1 or IMPACT-2 as shown in Fig. 3.1 (e). In the approximate m-bit adder, m 1-bit IMPACT-1 or m 1-bit IMPACT-2 full adders are used. In (Porto R. , et al., 2017) and (Paltrinieri, Peloso, Masera, Shafique, & Martina, 2018), it is shown that lower-part-OR adder (LOA) (Mahdiani, Ahmadi, Fakhraie, & Lucas, April, 2010) performs better than many segmented adders for ME. m-bit LOA is shown in Fig. 3.1 (b). An approximate n-bit subtractor can be designed using LOA as shown in Fig. 3.1 (e). In this approximate n-bit subtractor, m-bit LOA is used as the m-bit approximate adder.

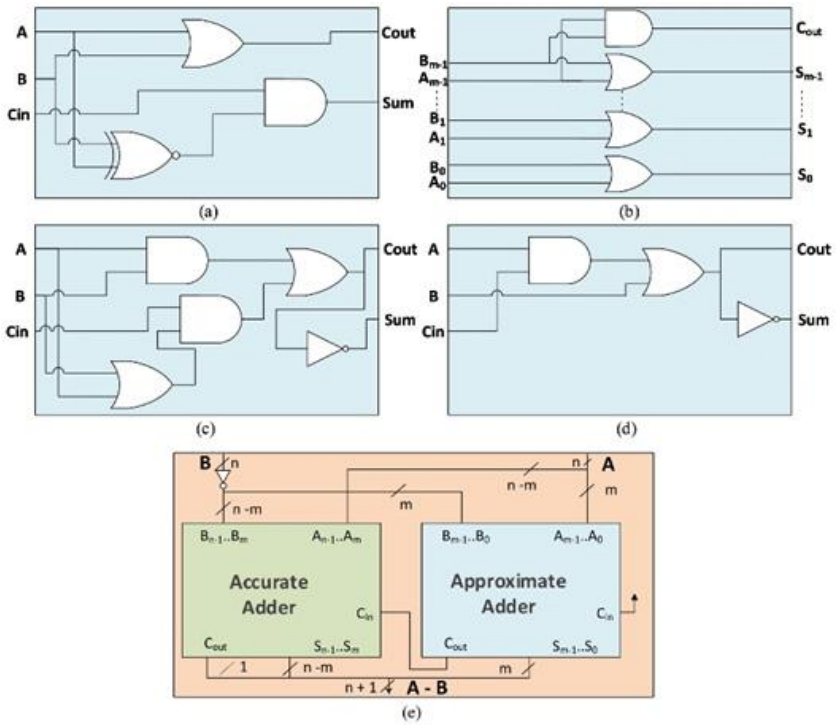


Figure 3.1 (a) Proposed 1-bit Approximate Full Adder (b) m-bit LOA (c) IMPACT-1 (d) IMPACT-2 (e) n-bit Approximate Subtractor

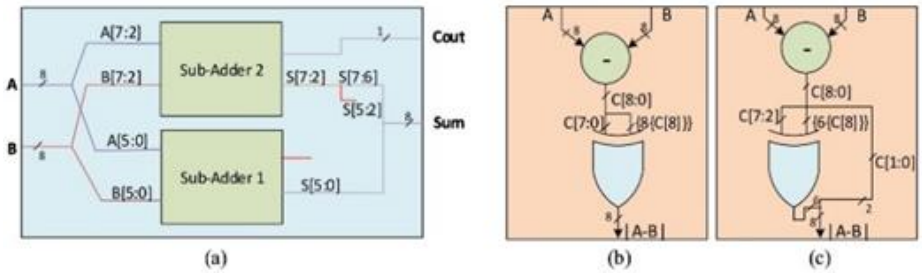


Figure 3.2 (a) GeAr Adder (N=8, R=2, P=4) (b) NAAD 0 (c) NAAD 2

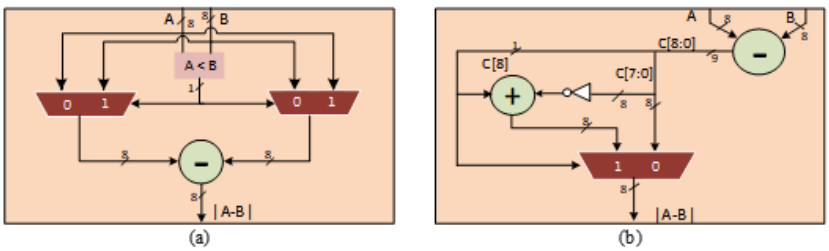


Figure 3.3 (a) Baseline 1 AD Hardware (b) Baseline 2 AD Hardware

We selected GeAr among segmented adders for our analysis. GeAr is shown in Fig. 3.2 (a). The novel approximate absolute difference (NAAD) hardware proposed in (Mert, Azgin, Kalali, & Hamzaoglu, 2019) is also included in our analysis. Two configurations of 8-bit NAAD are shown in Fig. 3.2 (b) and Fig. 3.2 (c).

We used the two accurate absolute difference (AD) hardware shown in Fig. 3.3 (a) and Fig. 3.3 (b) to provide baseline results for our analysis. We assessed impact of using the approximate subtractor shown in Fig. 3.1 (e) based on the proposed, IMPACT-1, IMPACT-2 and LOA adders for the subtraction operation in baseline 2 AD hardware. We assessed impact of using 1-bit, 2-bit, 3-bit and 4-bit approximate adder in this 8-bit approximate subtractor. We assessed impact of using the following four configurations of GeAr for the subtraction operation in baseline 2 AD hardware (a) N=8, R=1, P=6 (b) N=8, R=2, P=4 (c) N=8, R=1, P=4 (d) N=8, R=2, P=2. These configurations correspond to 1-bit, 2-bit, 3-bit and 4-bit approximations, respectively.

We analyzed using four different configurations of NAAD with 8, 7, 6, 5 XOR gates for the absolute difference operation. These configurations are referred to as NAAD 0, NAAD 1, NAAD 2, and NAAD 3, respectively. They correspond to 1-bit, 2-bit, 3-bit and 4-bit approximations, respectively. Finally, we analyzed applying traditional 1-bit, 2-bit, 3-bit, 4-bit truncation to baseline 2 AD hardware.

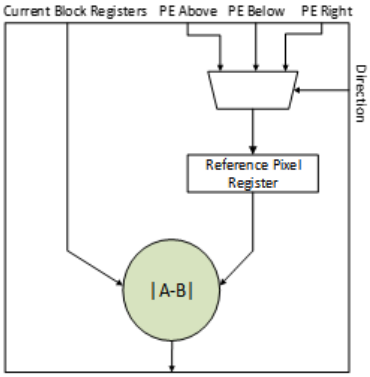


Figure 3.4 H.264 ME Processing Element

3.2 Motion Estimation Hardware

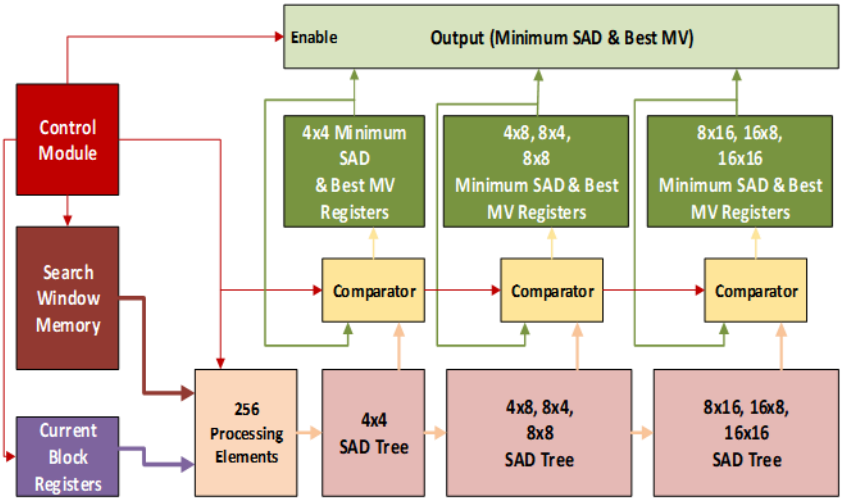


Figure 3.5 H.264 Motion Estimation Hardware

We used the H.264 variable block size full search ME hardware proposed in (Kalaycioglu, Ulusel, & Hamzaoglu, 2009) to assess impact of using approximate absolute difference hardware in an H.264 ME hardware. Block diagram of this ME hardware is shown in Fig. 3.5. 256 processing elements (PE) are used in this ME hardware to calculate absolute differences for a 16x16 block in parallel. As shown in Fig. 3.4, a PE is composed of a multiplexer to select the reference pixel, a register to store it and an absolute difference hardware. Current block pixels are stored in the current block registers. 32x32 search window pixels are stored in seventeen 18K Block RAMs (BRAM) in FPGA. In every clock cycle, 16 reference pixels are read from the search window BRAMs and stored in top or bottom PEs according to the search direction. The other PEs receive reference pixels from one of their neighboring PEs according to the search direction. It takes 16 clock cycles to fill the PE array with the first 16x16 reference block. After that, 41 SAD values for a search location are generated every clock cycle. Therefore, it takes 1040 clock cycles to generate SAD values for all search locations in a 32x32 search window.

3.3 Assessment of Using Approximate Circuits in Motion Estimation Hardware

We assessed impact of using the approximate absolute difference hardware in the H.264 ME hardware. For each approximate absolute difference hardware, we replaced the 256 exact absolute difference hardware in Verilog RTL code of the H.264 ME hardware with the approximate absolute difference hardware. In addition, we used the baseline 2 accurate absolute difference hardware shown in Fig. 3.3 (b) to provide baseline results for our analysis.

All Verilog RTL codes are synthesized and implemented using Xilinx Vivado Design Edition 2017.4 on Xilinx Virtex 7 XC7VX485TFFG1157 FPGA with speed grade 3. For all ME hardware, *Vivado Synthesis Defaults* and *Performance Explore* synthesis and implementation strategies are used, respectively.

Switching activity interchange format (SAIF) files are generated for the hardware with post-implementation timing simulations for Foreman video using Mentor Graphics QuestaSim. Power consumptions of the H.264 ME hardware are estimated with Xilinx Vivado using these SAIF files.

Table 3.1 presents average MSE results for all H.264 sub-block sizes for Foreman video. The proposed approximate adder achieves the smallest MSE results in most cases. GeAr performs the worst. It has the largest MSE results in all cases. Traditional bit truncation also performs worse than the proposed approximate adder in all cases. The other approximate circuits perform better than the proposed approximate adder in some cases.

In some cases, ME hardware using approximate absolute difference hardware has smaller MSE value than ME hardware using accurate absolute difference hardware. This is mainly because of the difference between SAD and MSE metrics. For example, for the following two sets of absolute differences A={2,2,2,2} and B = {3,3,0,0}, SAD{A} = 8 and SAD {B} = 6, whereas MSE {A}= 4 and MSE {B} = 4.5. A ME hardware using accurate absolute difference hardware will select SAD {B} as the minimum SAD. However, a ME hardware using an approximate absolute difference hardware may inaccurately select SAD {A} as the minimum SAD. Therefore, an approximate ME hardware may have smaller MSE value than accurate ME hardware.

Table 3.1 H.264 MSE Results

Approximate Circuit and its Configuration		H.264 MSE
Baseline		16.79
Proposed	1	16.79
	2	16.82
	3	16.98
	4	17.32
NAAD	0	16.80
	1	16.94
	2	17.43
	3	19.75
LOA	1	16.94
	2	17.12
	3	17.14
	4	18.81
IMPACT-1	1	16.56
	2	16.87
	3	17.87
	4	20.63
IMPACT-2	1	16.56
	2	16.87
	3	17.96
	4	21.55
GeAr	R1_P6	45.39
	R2_P4	42.73
	R1_P4	73.31
	R2_P2	50.88
Bit Truncation	1	17.20
	2	19.21
	3	22.82
	4	32.57

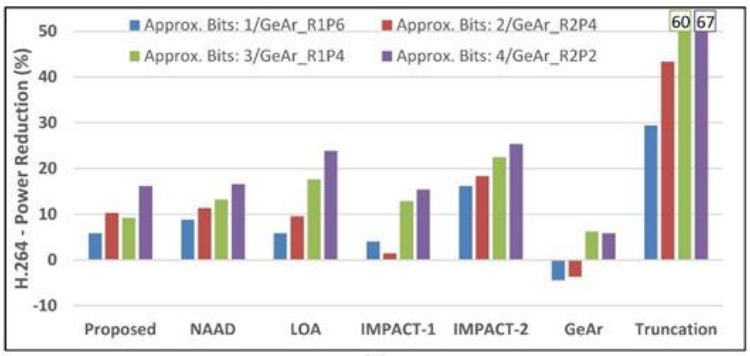


Figure 3.6 H.264 Power Reduction (%)

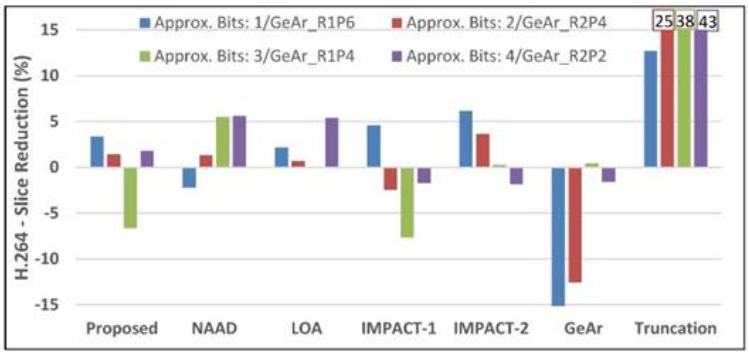


Figure 3.7 H.264 Slice Reduction (%)

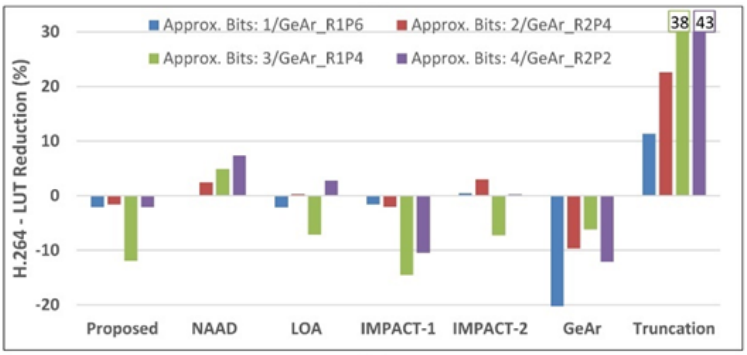


Figure 3.8 H.264 LUT Reduction (%)

Area and power consumption results of the H.264 ME hardware are shown in Fig. 3.6, Fig. 3.7 and Fig. 3.8. In the figures, percentage reductions achieved by approximate ME hardware compared to the corresponding accurate ME hardware are shown. As expected, traditional bit truncation achieves the largest area and power consumption reductions at the expense of more quality loss than the proposed approximate adder. GeAr has the worst area and power consumption results. For 1-bit approximation, the proposed approximate adder by (Ahmad, 2021) achieves 5% power reduction in H.264 ME hardware without affecting quality.

We assessed the impact of using approximate adders in the adder tree of the same H.264 ME hardware. The adder tree of the H.264 ME hardware has adders with bit widths from 8 to 15. The accurate adders in the adder tree are replaced with the previously mentioned approximate adders in the literature. PSNR and MSE results of H.264 ME hardware are shown in Table 3.2. GeAr gives the best result only for 1 approximate bit confguration. For other configurations, the proposed approximate adder by (Ahmad, 2021) gives the best results.

Table 3.2 H.264 Quality Results

Approximate Circuit and its Configuration		PSNR	MSE
Baseline		36.17	16.79
Proposed	1	36.16	16.80
	2	36.12	16.91
	3	36.03	17.14
	4	35.82	17.77
	5	35.61	18.49
LOA	1	36.13	16.89
	2	36.02	17.20
	3	35.91	17.56
	4	35.57	18.69
	5	34.91	21.53
Impact-1	1	36.17	16.78
	2	36.11	16.91
	3	35.85	17.42
	4	34.40	23.65
	5	31.16	51.21
Impact-2	1	36.15	16.85
	2	36.00	17.20
	3	35.62	18.40
	4	34.51	23.10
	5	31.77	43.58
GeAr	R1P6	19.05	813.27
	R1P4	20.00	690.56
	R2P2	19.83	696.40
	R2P4	19.49	743.74
	R1P2	19.72	714.73

All Verilog RTL codes are synthesized and implemented using Xilinx Vivado Design Edition 2020.1 on a Xilinx Virtex 7 XC7VX485TFFG1157 FPGA with speed grade 3. For all ME hardware, *Vivado Synthesis Defaults* and *Performance Explore* synthesis and implementation strategies are used, respectively. Switching activity interchange format (SAIF) files are generated for the FPGA implementations with post-implementation timing simulations for Foreman video using Mentor Graphics QuestaSim. Power consumptions of the H.264 ME hardware are estimated with Xilinx Vivado using these SAIF files.

The implementation results for H.264 ME hardware using approximate adders in the adder tree are shown in Table 3.3. H.264 ME hardware using the proposed approximate adder by

(Ahmad, 2021) has the smallest area whereas H.264 ME hardware using GeAr has the largest area. Since the adder tree is not on the critical path of the H.264 ME hardware, maximum frequency results are very close to each other. For 1 and 2 approximate bit configurations, H.264 ME hardware using the proposed approximate adder by (Ahmad, 2021) consumes less power than the ones in the literature. For configurations using 3, 4, and 5 approximate bits, H.264 ME hardware using LOA consumes less power than the others. For configurations using 3, 4 and 5 approximate bits, H.264 ME hardware using the proposed approximate adder by (Ahmad, 2021) consumes more power than LOA but less power than others.

Table 3.3 H.264 Implementation Results

Approximate Circuit and its Configuration		LUT	Max. Freq. (MHz)	Power (W)
Baseline		10001	347.22	0.266
Proposed	1	9602	350.14	0.253
	2	9407	365.50	0.240
	3	9298	371.20	0.242
	4	9516	363.64	0.242
	5	9579	367.65	0.231
LOA	1	9840	370.37	0.259
	2	9677	368.32	0.243
	3	9445	353.36	0.236
	4	9604	371.20	0.232
	5	9360	371.06	0.220
Impact-1	1	9838	370.37	0.257
	2	9898	364.30	0.256
	3	9854	363.37	0.256
	4	10110	342.70	0.261
	5	9884	347.83	0.250
Impact-2	1	9752	364.30	0.253
	2	9774	369.69	0.245
	3	9568	353.11	0.260
	4	10096	350.88	0.279
	5	10046	347.83	0.284
GeAr	R1P6	13142	350.88	0.338
	R1P4	11325	355.87	0.342
	R2P2	10643	352.36	0.317
	R2P4	11566	353.86	0.307
	R1P2	10334	355.24	0.280

CHAPTER IV

VVC AFFINE MOTION ESTIMATION HARDWARE

There are two modes of affine motion estimation (AME): 4-parameters AME which uses two motion vectors and 6-parameters AME which uses three motion vectors. Although affine transformation can be applied on various shapes, there are two important assumptions for the AME formulae to work (US Patent No. WO 2017/200771 Al, 2017). First, blocks should be squares. Second, upper left edge of the blocks (mv_0's location) should be considered as the origin of the coordinate system. These two assumptions allow the 4-parameters AME formulae to be written as shown in 4.1 and 4.2, and the 6-parameters AME formulae to be written as shown in 4.3 and 4.4. 4-parameters AME takes zoom and rotation into account, whereas 6-parameters AME takes shear into account as well.

$$mv_x = \frac{mv_{1_x} - mv_{0_x}}{w} x - \frac{mv_{1_y} - mv_{0_y}}{w} y + mv_{0_x} \tag{4.1}$$

$$mv_y = \frac{mv_{1_y} - mv_{0_y}}{w} x - \frac{mv_{1_x} - mv_{0_x}}{w} y + mv_{0_y} \tag{4.2}$$

$$mv_x = \frac{mv_{1_x} - mv_{0_x}}{w} x - \frac{mv_{2_x} - mv_{0_x}}{w} y + mv_{0_x} \tag{4.3}$$

$$mv_y = \frac{mv_{1_y} - mv_{0_y}}{w} x - \frac{mv_{2_y} - mv_{0_y}}{w} y + mv_{0_y} \tag{4.4}$$

Translation motion estimation is the most computationally intensive operation of video compression. AME has much higher computational complexity than translational motion estimation. To reduce computational complexity, VVC standard applies AME on 4x4 sub-blocks instead of pixels. As the spatial video resolutions increase, the ratio of the area occupied by a sub-block to the whole frame decreases. Therefore, significance of each sub-block decreases giving opportunity to make operations on sub-blocks instead of pixels. Fig. 4.1 shows the ratio of the area occupied by a single pixel and a 4x4 sub-block to the whole frame in various video resolutions. For example, a single pixel in an HD frame has more significance than a 4x4 sub-block in an 8K frame.

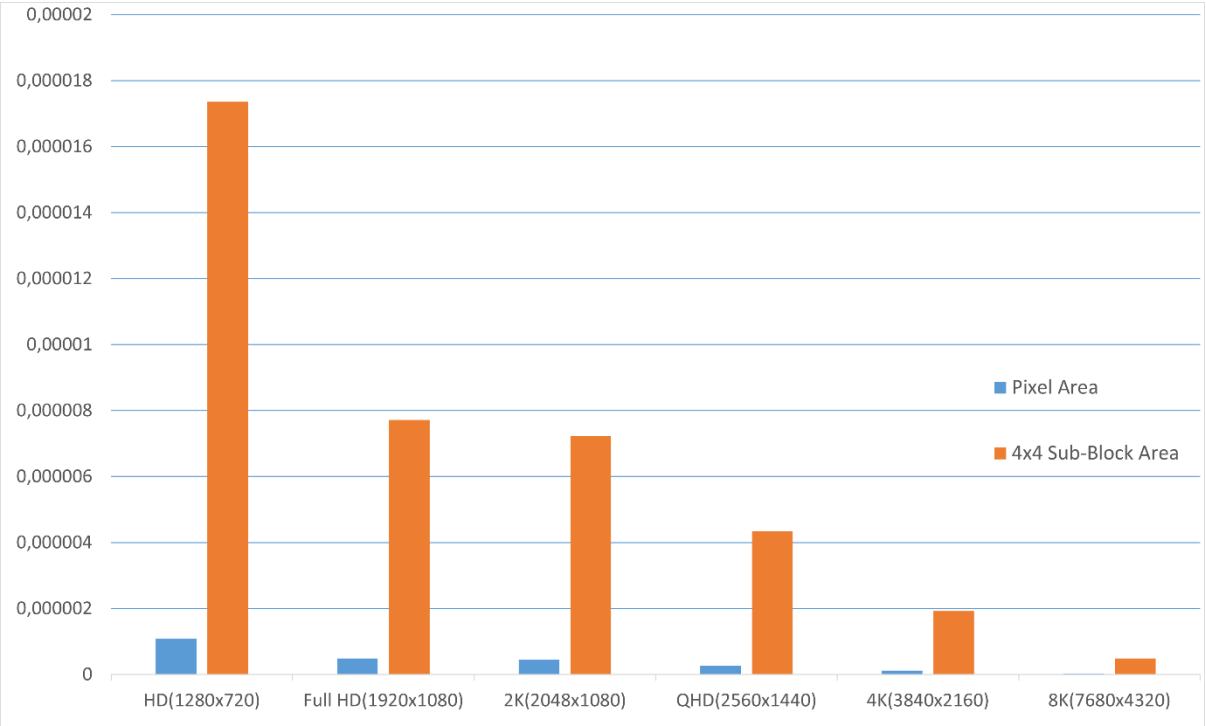


Figure 4.1 Pixel vs. 4x4 Sub-Block

For translational ME, a motion vector is calculated for each block and all the pixels in a block is moved to the location pointed by that motion vector regardless of their positions in the block. On the other hand, for AME, pixels' new locations are calculated depending on their locations in the block. For example, if AME is performed on pixels instead of 4x4 sub-blocks, decoder needs to solve the AME formulae 256 times for a 16x16 block instead of 16 times. For AME, as shown in Fig. 4.2, encoder uses affine motion vectors to calculate translational motion vectors for the center points of every sub-block.

In Fig. 4.2 and Fig. 4.3, black arrows are the affine motion vectors, and the colored arrows are four of the sixteen calculated translational motion vectors for the sub-blocks. In Fig. 4.3, only four translational motion vectors are shown to clearly point out the difference between translational motion vectors at different locations. As shown in the figures, translational motion vectors of the sub-blocks closer to the affine motion vectors are similar to the affine motion vectors. On the other hand, translational motion vectors of the sub-blocks far from the affine motion vectors are different from the affine motion vectors.

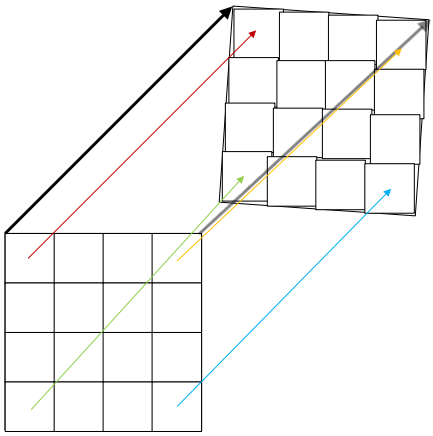


Figure 4.2 Affine Motion Vectors for Sub-Blocks

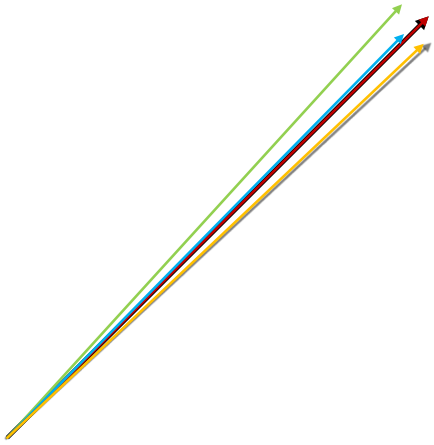


Figure 4.3 Affine Motion Vectors

Sub-block based AME introduces an approximation while decreasing the computational complexity. For example, as shown in Fig. 4.2, if the processed block is zoomed in, zoomed out or rotated, sub-blocks start to overlap and share pixels, or gaps appear between sub-blocks. Sub-block based AME keeps much less motion vectors than traditional translational motion estimation. For example, traditional translational motion estimation needs to keep 1024 motion vectors to encode a 128x128 block by using 4x4 sub-blocks whereas AME needs to keep only 2 or 3 affine motion vectors.

For 4-parameters AME for a 16x16 block with 4x4 sub-blocks, mv_0 is applied to upper left corner and mv_1 is applied to upper right corner. For 6-parameters AME, in addition to mv_0 and mv_1, mv_2 is applied to bottom left corner. X and Y components of a motion vector are shown as mv_x and mv_y. For a 16x16 block with 4x4 sub-blocks, X (mv_x) and Y (mv_y) components of 16 translational motion vectors are calculated. In the formulae 4.1 - 4.4, W stands for width and height, which are equal to 16 for a 16x16 block. x and y are the

coordinates of the location which the formula is being applied. For example, since the formulae 4.1 - 4.4 are applied to the center points of the sub-blocks, (x, y) couples for a 16x16 block are:

- (1.5, 1.5), (5.5, 1.5), (9.5, 1.5), (13.5, 1.5)
- (1.5, 5.5), (5.5, 5.5), (9.5, 5.5), (13.5, 5.5)
- (1.5, 9.5), (5.5, 9.5), (9.5, 9.5), (13.5, 9.5)
- (1.5, 13.5), (5.5, 13.5), (9.5, 13.5), (13.5, 13.5)

In the formulae 4.1 - 4.4, $+mv_{0_x}$ and $+mv_{0_y}$ ensure the translational motion where the rest is responsible for affine motion such as rotation, zoom and shear. Consider the 4-parameter formula for mv_x. $\left(mv_{1_x} - mv_{0_x}\right)$, i.e., difference of x components of two motion vectors, controls the movement on the x axis and multiplication with the x coordinate controls the distance between sub-blocks by moving the sub-blocks located on the right more than the sub-blocks located on the left. Similarly, $\left(mv_{1_y} - mv_{0_y}\right)$, i.e., difference of y components of two motion vectors, controls the movement on the x axis and multiplication with the y coordinate controls the distance between sub-blocks by moving the sub-blocks located on the bottom more than the sub-blocks located on the top. The same situation is valid for mv_y.

Block matching ME algorithms calculate a sum of absolute difference (SAD) value for every search location. The number of search locations is determined by the ME algorithm. ME algorithms searching more search locations are likely to find better matches. Therefore, they achieve more video compression at the expense of higher computational complexity. For example, traditional full search translational ME algorithm calculates SAD values for 256 search locations for a 16x16 search window and for 16384 search locations for a 128x128 search window (Ahmad, Efficient HEVC and VVC motion estimation hardware, 2021).

Since AME algorithms use two or three motion vectors for every block, AME increases the number of search locations exponentially. For example, for a 128x128 search window, there are 16384 mv_0 search locations, 16384 mv_1 search locations for every mv_0 search location and 16384 mv_2 search locations for every mv_1 search location. Therefore, if 6-parameter full search AME algorithm is performed, approximately 4.3×10^{12} SAD values should be calculated, which is practically impossible. Therefore, approximate AME algorithms should be used such as performing full search for mv_0 and searching only pre-

determined search locations for mv_1 and mv_2 (Li, Li, Lv, & Yang, 2015) (Girotra, Johar, Ghosh, & Chakrabarti, 2003).

4.1 Proposed VVC Affine Motion Estimation Hardware

In this thesis, we propose a VVC AME hardware which performs 4-parameters AME. To the best of our knowledge, it is the first VVC AME hardware in the literature. We propose a novel pixel storage method which significantly reduces the computational complexity and the number of BRAM read operations.

The proposed hardware uses 128x128 fixed search window size. It can work for 16x16, 32x32 or 64x64 block sizes. It takes the block size as input. The user decides on the trade-off between compression and speed by choosing the block size (Bross, Chen, Ohm, Sullivan, & Wang, 2021). The number of search locations searched by the proposed hardware depends on the block size. For mv_0, it searches all the search locations in the search window for which the entire current block is in the search window. In other words, it searches 4096 search locations for 64x64 block size, 9216 search locations for 32x32 block size and 12544 search locations for 16x16 block size for mv_0. For mv_1, it searches only 8 pre-determined search locations. Therefore, the proposed VVC AME hardware searches 32768, 73728 or 100352 search locations depending on the block size given by the user.

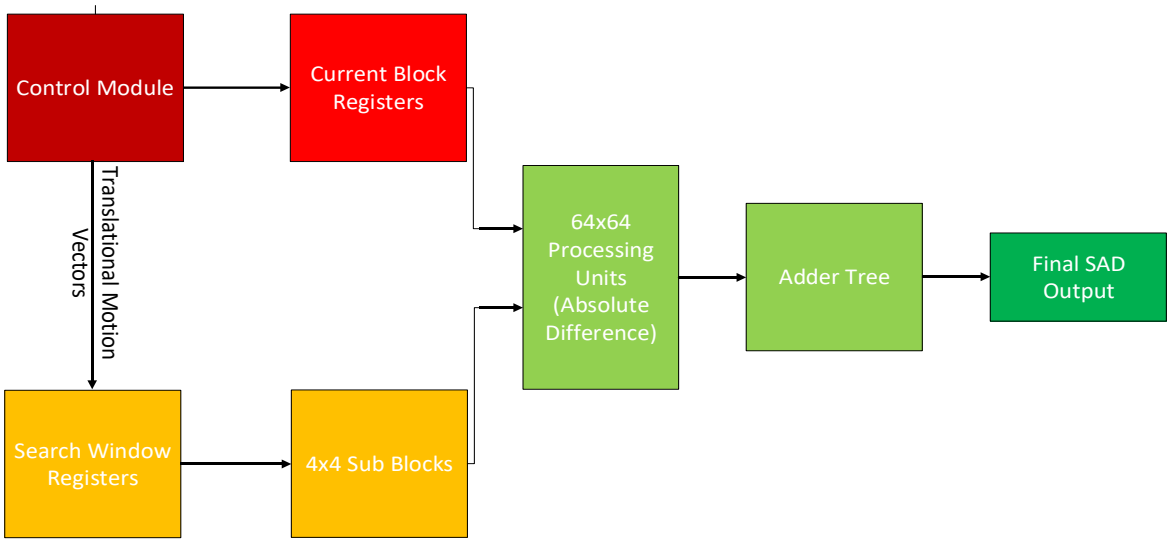


Figure 4.4 VVC Affine Motion Estimation Hardware

The proposed hardware has two copies of the VVC AME hardware shown in Fig. 4.4. Each copy is composed of a translational motion vector calculation part in the control module, multiplexers to choose pixels using the motion vectors, 64x64 processing units for absolute difference calculation and an adder tree.

After the start signal, the proposed hardware reads the search window pixels from off-chip memory and writes them to BRAMs. It uses 16 BRAMs as shown in Fig. 4.5. It reads 64 pixels in a clock cycle. The proposed hardware concatenates the 64 pixels and writes them to a single location in BRAMs. Since the search window size is 128x128, 128 rows are stored to each BRAM. Therefore, it takes 256 clock cycles to write the search window pixels to BRAMs. Then, the proposed hardware reads the current block pixels which takes 64, 16 or 4 clock cycles depending on the block size. Then, the proposed hardware reads one row of search window pixels (128 pixels) from BRAMs in one clock cycle and stores them to registers as shown in Fig 4.6. Since the affine motion vectors can point upwards and downwards, previous and next rows are needed for SAD calculations. Therefore, for example, if the block size is 16x16, the proposed hardware reads 18 rows from the BRAMs and stores them to registers. After the proposed hardware reads the necessary search window pixels from BRAMs, it starts SAD calculation.

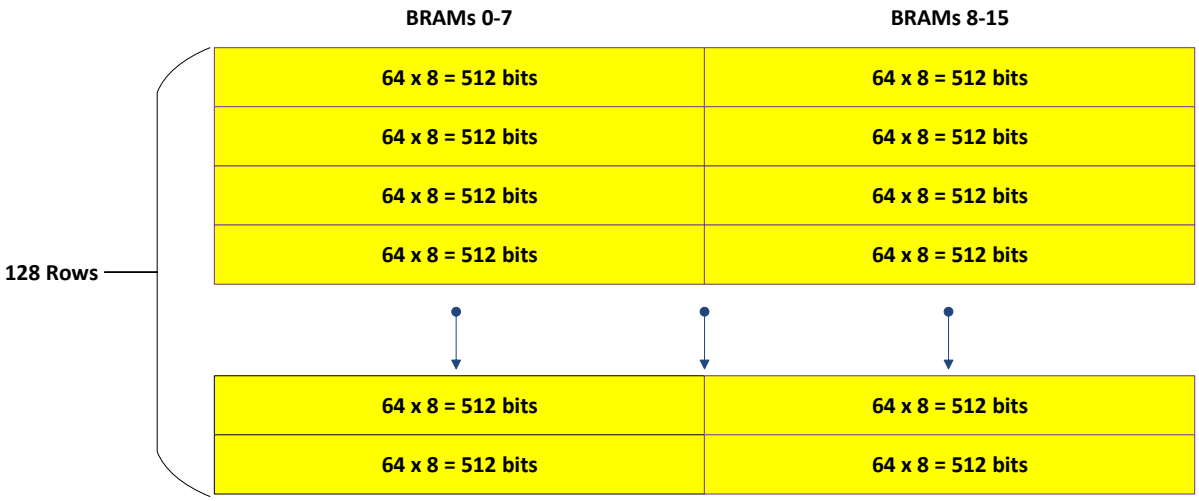


Figure 4.5 Pixel Storage in BRAMs

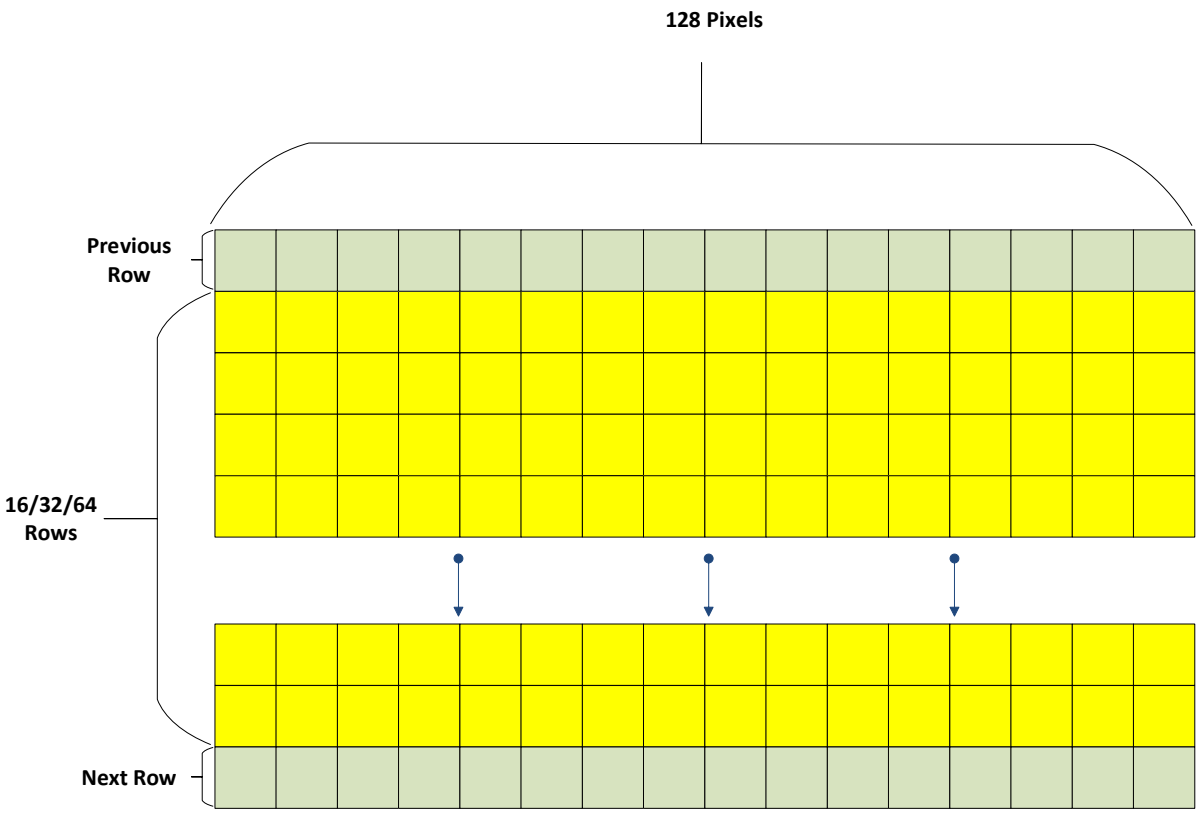


Figure 4.6 Pixel Storage in Registers

According to the translational motion vectors calculated by the control module, appropriate pixels are sent to the 64x64 processing units for absolute difference calculation and absolute difference results are added by the adder tree. The proposed hardware searches 8 mv_1 search locations for each mv_0 search location. Seven search locations are shown by arrows in Fig. 4.7, the remaining search location is the upper right edge of the block. As mentioned earlier, two copies of the VVC AME hardware shown in Fig. 4.4 work in parallel. Each copy performs motion vector and SAD calculations 4 times.

The smallest SAD value and the related motion vector information are stored in the registers for comparison with the SAD values that will be calculated. After SAD calculations for a mv_0 search location and 8 mv_1 search locations finish, the proposed hardware left shifts the search window pixels in the registers by one instead of incrementing mv_{0_x} value. Similarly, after SAD calculations for a row finish, the search window pixels in the registers are shifted up instead of incrementing mv_{0_y} value and a new row is read from the BRAMs. After all SAD calculations finish, the proposed hardware sends the smallest SAD value and the related motion vector information as the output. These steps are repeated for every block.

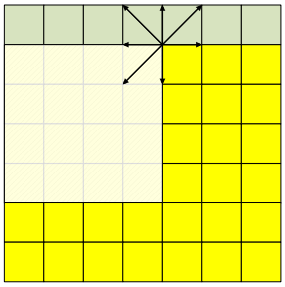


Figure 4.7 $\mathbf{mv_1}$ Locations

Since motion vector and SAD calculations are pipelined, calculations for each mv_0 search location take 4 clock cycles. Therefore, the latency is 16396, 36876 or 50188 clock cycles for every block when 12 clock cycles latency of the adder tree is added.

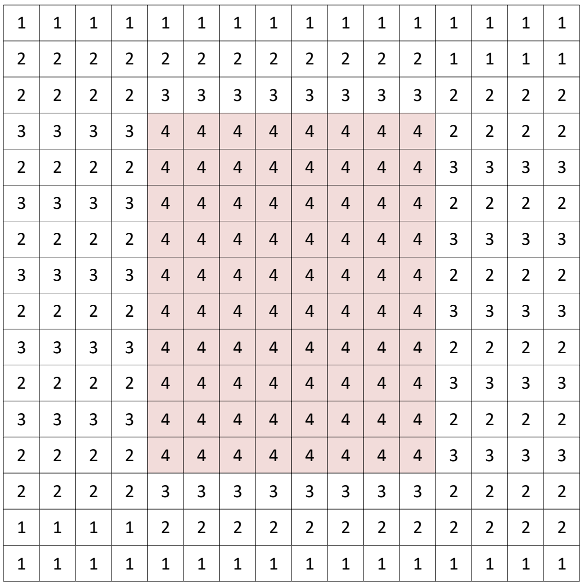
As mentioned before, the proposed hardware uses a novel pixel storage method. The proposed method uses large number of registers. However, it has three important advantages. First, it significantly reduces the number of BRAM read operations. Second, it eliminates complex address generation for BRAMs. Third, it significantly simplifies computation of translational motion vectors from affine motion vectors.

As the search window size and block size increase, the proposed hardware would read the same pixel from BRAMs repeatedly if the proposed method was not used. Fig. 4.8 and Fig. 4.9 show the number of read operations needed for each pixel for 8x8 and 16x16 search window sizes if the block size is 4x4. If the search window size is 8x8, 64 pixels are read 112 times, and if the search window size is 16x16, 256 pixels are read 688 times. If the search window size is 128x128, for 32x32 block size, 96x98 pixels will be read 32 times which adds up to 301056 pixel reads. If the search window size is 128x128, for 16x16 block size, 112x114 pixels will be read 16 times which adds up to 204288 pixel reads. Therefore, the proposed method significantly reduces the number of BRAM read operations.

For traditional translational motion estimation, reading pixels from the search window for SAD calculation is a simple task. BRAM addresses can be calculated by simply incrementing the previous addresses. For AME, BRAM address calculation is very complex. By keeping the search window pixels in registers, the proposed hardware eliminates complex address generation for BRAMs.

1	1	1	1	1	1	1	1
2	2	2	2	1	1	1	1
2	2	2	2	2	2	2	2
3	3	3	3	2	2	2	2
2	2	2	2	3	3	3	3
2	2	2	2	2	2	2	2
1	1	1	1	2	2	2	2
1	1	1	1	1	1	1	1

Figure 4.8 Number of Pixels Read for 8x8 Search Window



1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1
2	2	2	2	3	3	3	3	3	3	3	3	2	2	2	2
3	3	3	3	4	4	4	4	4	4	4	4	2	2	2	2
2	2	2	2	4	4	4	4	4	4	4	4	3	3	3	3
3	3	3	3	4	4	4	4	4	4	4	4	2	2	2	2
2	2	2	2	4	4	4	4	4	4	4	4	3	3	3	3
3	3	3	3	4	4	4	4	4	4	4	4	2	2	2	2
2	2	2	2	4	4	4	4	4	4	4	4	3	3	3	3
3	3	3	3	4	4	4	4	4	4	4	4	2	2	2	2
2	2	2	2	4	4	4	4	4	4	4	4	3	3	3	3
3	3	3	3	4	4	4	4	4	4	4	4	2	2	2	2
2	2	2	2	4	4	4	4	4	4	4	4	3	3	3	3
2	2	2	2	3	3	3	3	3	3	3	3	2	2	2	2
1	1	1	1	2	2	2	2	2	2	2	2	2	2	2	2
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 4.9 Number of Pixels Read for 16x16 Search Window

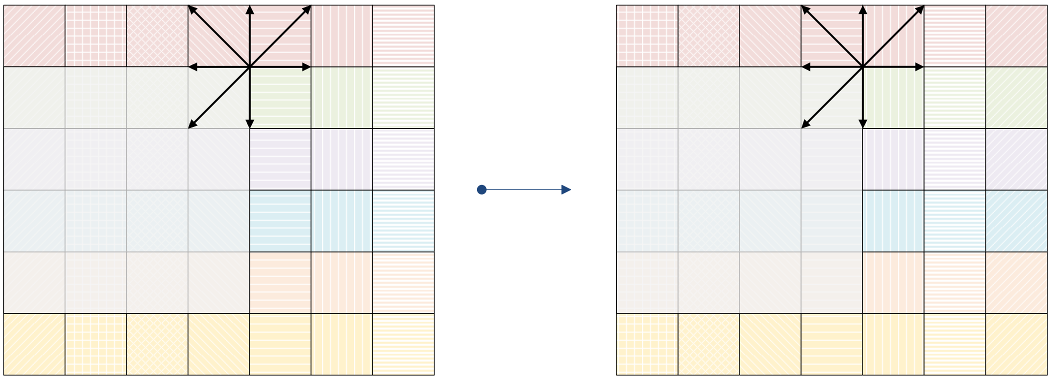


Figure 4.10 Left Shift Operation

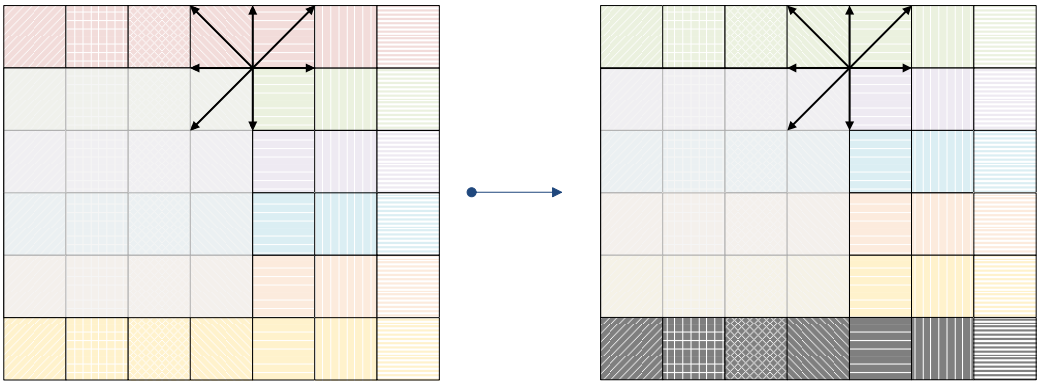


Figure 4.11 Up Shift Operation

As mentioned before, the proposed method performs left and up shifts which can be considered as moving the search window instead of the current block. Fig. 4.10 and Fig. 4.11 show the left and up shift operations on a smaller scale. Therefore, mv_0 value does not change for the vector calculation. Since the first mv_0 value is decided to be (0,0), AME formulae become as shown below.

$$mv_x = \frac{mv_{1_x}}{w}x - \frac{mv_{1_y}}{w}y \tag{4.5}$$

$$mv_y = \frac{mv_{1_y}}{w}x - \frac{mv_{1_x}}{w}y \tag{4.6}$$

$$mv_x = \frac{mv_{1_x}}{w}x - \frac{mv_{2_x}}{w}y \tag{4.7}$$

$$mv_y = \frac{mv_{1_y}}{w}x - \frac{mv_{2_y}}{w}y \tag{4.8}$$

128x128 search window requires each motion vector to be 7-bits. Since the proposed hardware eliminates mv_0 from the formulae and it searches 8 mv_1 search locations, mv_1 becomes 4-bits. Therefore, division and multiplication operations with 8-bit values are transformed to division and multiplication operations with 4-bit values. Since two copies of the VVC AME hardware shown in Fig. 4.4 work in parallel and 256 sub-blocks exist in a 64x64 block and each sub-block requires 4 division and 4 multiplication operations, 2048 divisions and 2048 multiplications are required. Therefore, reduction from 7-bits to 4-bits for vector calculation significantly reduces the hardware area.

The proposed VVC AME hardware is implemented in Verilog HDL. The Verilog RTL codes are synthesized and implemented by Vivado 2020.1 on a Xilinx Virtex UltraScale+

FPGA xcvu9p-flgb2104-3-e. Default strategy is used for synthesis and performance explore strategy is used for implementation. The maximum clock frequency is determined as 125 MHz. The FPGA implementation uses 6555741 LUTs, 252982 FFs, 576 IO PADs, 16 BRAMs and 1920 DSPs.

Table 4.1 shows the number of frames that can be processed by the proposed VVC AME hardware for various video resolutions. As mentioned before, AME for a block takes 16652, 37132 or 50444 clock cycles depending on the block size. In the table, hybrid is the case when 40% of the frame is processed with 64x64 block size, 35% of the frame is processed with 32x32 block size and 25% of the frame is processed by 16x16 block size.

Table 4.1 Number of Frames Processed per Second

Block Size	HD	Full HD
64x64	32.5 fps	15 fps
32x32	3.7 fps	1.6 fps
16x16	0.65 fps	0.29 fps
Hybrid	2 fps	0.91 fps

As shown in Table 4.1, when AME is performed for the entire video frame, the proposed hardware does not achieve real-time performance in high resolution videos. Although parallel processing can be used to increase performance, since AME hardware has large area, it is not feasible to perform AME for the entire video frame. Since affine motion does not exist in all the blocks in a video frame, the video frame should be analyzed and AME should be performed only for the blocks which possibly have affine motion.

CHAPTER V

CONCLUSIONS AND FUTURE WORK

In this thesis, we proposed a low error approximate absolute difference (LAD_X) hardware. It has lower maximum and average error, and higher accuracy than the approximate AD hardware in the literature. It has similar performance with and smaller area than them. The H.264 ME hardware using LAD_X hardware performs higher quality ME than the H.264 ME hardware using the approximate AD hardware in the literature. It has similar performance with and smaller area than them.

In this thesis, we assessed impact of using several approximate circuits from the literature in H.264 ME hardware. The approximate adder proposed by (Ahmad, 2021) achieved up to 10% power reduction in ME hardware while providing better quality than the other approximate circuits. Traditional bit truncation achieved the largest area and power reductions in ME hardware at the expense of more quality loss than the approximate adder proposed by (Ahmad, 2021). GeAr segmented approximate adder had the worst quality, area and power consumption results.

In this thesis, a novel VVC AME hardware is proposed. To the best of our knowledge, it is the first VVC AME hardware in the literature. It implements 4-parameters affine transform and uses a novel pixel storage method.

As future work, the proposed VVC AME hardware can be improved by using fast search algorithms instead of full search algorithm. Approximate circuits can be used to improve area, power, speed of the proposed VVC AME hardware. Since computational complexity of VVC AME is much more than that of H.264 ME, using approximate circuits in VVC AME hardware may have more impact than using them in H.264 ME hardware.

BIBLIOGRAPHY

Adhuran, J., Fernando, A., Kulupana, G., & Blasi, S. (2021). Affine Intra-prediction for Versatile Video Coding. *2020 28th European Signal Processing Conference (EUSIPCO)*, (s. 545-549). doi:10.23919/Eusipco47968.2020.9287579

Ahmad, W. (2021). Efficient HEVC and VVC motion estimation hardware. PhD Thesis. Sabanci University. Istanbul, Turkey.

Ahmad, W., & Hamzaoglu, I. (2021). An efficient approximate sum of absolute differences hardware for FPGAs. *Proc. IEEE Int. Conf. Consum. Electron (ICCE).* IEEE.

Ahmad, W., Ayrancioglu, B., & Hamzaoglu, I. (2020). Comparison of Approximate Circuits for H.264 and HEVC Motion Estimation. *2020 23rd Euromicro Conference on Digital System Design (DSD)*, (s. 167-173). doi:10.1109/DSD51259.2020.00036

Arifeen, T., Hassan, A. S., Moradian, H., & Lee, J. A. (2016). Probing Approximate TMR in Error Resilient Applications for Better Design Tradeoffs. *Euromicro Conference on Digital System Design (DSD).* Limassol: IEEE.

Ayhan, T., & Altun, M. (2019). Circuit aware approximate system design with case studies in image processing and neural networks. *IEEE Access*, 4726-4734.

Azgin, H., Kalali, E., & Hamzaoglu, I. (2020). An Approximate Versatile Video Coding Fractional Interpolation Hardware. *IEEE International Conference on Consumer Electronics.* Las Vegas: IEEE.

Bross, B., Chen, J., Ohm, J.-R., Sullivan, G. J., & Wang, Y.-K. (2021). Developments in International Video Coding Standardization After AVC, With an Overview of Versatile Video Coding (VVC). *Proceedings of the IEEE, 109*, 1463-1493. doi:10.1109/JPROC.2020.3043399

Chen, L., Han, J., Liu, W., Montuschi, P., & Lombardi, F. (2018, Jul.). Design evaluation and application of approximate high-radix dividers. *IEEE Access, vol. 4*(no. 3), 299-312.

Choi, Y.-J., & Kim, B.-G. (2019). A review on motion estimation and compensation for Versatile Video Coding technology (VVC). *Journal of Korea Multimedia Society, 22*, 770–779.

El-Harouni, W., Rehman, S., Prabakaran, B. S., Kumar, A., Hafiz, R., & Shafique, M. (2017). Embracing approximate computing for energy-efficient motion estimation in high efficiency video coding. *Design, Automation & Test in Europe Conference & Exhibition (DATE).* Lausanne.

Froehlich, S., Große, D., & Drechsler, R. (2018). Towards Reversed Approximate Hardware Design. *Euromicro Conference on Digital System Design (DSD).* Prague: IEEE.

Gillani, G. A., Hanif, M. A., Verstoep, B., Gerez, S. H., Shafique, M., & Kokkeler, A. J. (2019). MACISH: Designing Approximate MAC Accelerators With Internal-Self-Healing. *IEEE Access*, 77142 - 77160.

Girotra, A., Johar, S., Ghosh, D., & Chakrabarti, I. (2003). An architecture for affine motion estimation in real-time video coding. *9th Asia-Pacific Conference on Communications (IEEE Cat. No.03EX732), 1*, s. 103-107 Vol.1. doi:10.1109/APCC.2003.1274321

Gupta, V., Mohapatra, D., Park, S. P., Raghunathan, A., & Roy, K. (2011). IMPACT: IMPrecise adders for low-power approximate computing. *IEEE/ACM International Symposium on Low Power Electronics and Design.* Fukuoka.

Jiang, H., Liu, C., Liu, L., Lombardi, F., & Han, J. (2017). A review classification and comparative evaluation of approximate arithmetic circuits. *ACM J. Emerg. Technol. Comput. Syst.*, 1-34.

Jiang, H., Liu, C., Liu, L., Lombardi, F., & Han, J. (August 2017). A review, classification, and comparative evaluation of approximate arithmetic circuits. *ACM Journal on Emerging Technologies in Computing Systems (JETC), 13*(4).

Kahng, A. B., & Kang, S. (2012). Accuracy-configurable adder for approximate arithmetic designs. *DAC Design Automation Conference 2012*, (s. 820-825). doi:10.1145/2228360.2228509

Kalali, E., & Hamzaoglu, I. (2018). Approximate HEVC Fractional Interpolation Filters and Their Hardware Implementations. *IEEE Transactions on Consumer Electronics , 64*(3), 285 - 291.

Kalali, E., & Hamzaoglu, I. (2020). An approximate HEVC intra angular prediction hardware. *IEEE Access*, 2599-2607.

Kalaycioglu, C., Ulusel, O. C., & Hamzaoglu, I. (2009). Low power techniques for Motion Estimation hardware. *International Conference on Field Programmable Logic and Applications.* Prague.

Li, L., Li, H., Lv, Z., & Yang, H. (2015). An affine motion compensation framework for high efficiency video coding. *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, (s. 525-528). doi:10.1109/ISCAS.2015.7168686

Mahdiani, H. R., Ahmadi, A., Fakhraie, S. M., & Lucas, C. (April, 2010). Bio-Inspired Imprecise Computational Blocks for Efficient VLSI Implementation of Soft-Computing Applications. *IEEE Transactions on Circuits and Systems I: Regular Papers, 57*(4), 850-862.

Mert, A. C., Azgin, H., Kalali, E., & Hamzaoglu, I. (2019). Novel Approximate Absolute Difference Hardware. *22nd Euromicro Conference on Digital System Design (DSD).* Kallithea.

Paltrinieri, A., Peloso, R., Masera, G., Shafique, M., & Martina, M. (2018). Approximate-Computing Architectures for Motion Estimation in HEVC. *New Generation of CAS (NGCAS).* Valletta.

Porto, R., Agostini, L., Zatt, B., Porto, M., Roma, N., & Sousa, L. (2017). Energy-efficient motion estimation with approximate arithmetic. *IEEE 19th International Workshop on Multimedia Signal Processing (MMSP).* Luton.

Porto, R., Agostini, L., Zatt, B., Roma, N., & Porto, M. (2019). Power-Efficient Approximate SAD Architecture with LOA Imprecise Adders. *2019 IEEE 10th Latin American Symposium on Circuits & Systems (LASCAS)*, (s. 65-68). doi:10.1109/LASCAS.2019.8667554

Ren, W., He, W., & Cui, Y. (2020, July). An Improved Fast Affine Motion Estimation Based on Edge Detection Algorithm for VVC. *Symmetry, 12*, 1143. doi:10.3390/sym12071143

Seferidis, V. E., & Ghanbari, M. (1993). General approach to block-matching motion estimation. *Optical Engineering, 32*, 1464–1474.

Shafique, M., Ahmad, W., Hafiz, R., & Henkel, J. (2015). A low latency generic accuracy configurable adder. *52nd ACM/EDAC/IEEE Design Automation Conference (DAC).* San Francisco.

Van Toan, N., & Lee, J. G. (2020). FPGA-based multi-level approximate multipliers for high-performance error-resilient applications. *IEEE Access*(vol.8), 25481-25497.

Verma, A. K., Brisk, P., & Ienne, P. (2008). Variable Latency Speculative Addition: A New Paradigm for Arithmetic Circuit Design. *2008 Design, Automation and Test in Europe*, (s. 1250-1255). doi:10.1109/DATE.2008.4484850

Xu, Q., Mytkowicz, T., & Kim, N. S. (2015). Approximate Computing: A Survey. *IEEE Design & Test, vol. 33*(no. 1), 8-22.

Zou, F., Chen, J., Karczewicz, M., Xiang, L., Chuang, H.-C., & Chien, W.-J. (2017, 11 23). *US Patent No. WO 2017/200771 Al.*