EFFICIENT HEVC AND VVC MOTION ESTIMATION HARDWARE

by
WAQAR AHMAD

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfillment of
the requirements for the degree of Doctor of Philosophy

Sabanci University
July 2021

EFFICIENT HEVC AND VVC MOTION ESTIMATION HARDWARE

Approved by:

Date of Approval: July 14, 2021

© WAQAR AHMAD 2021

All Rights Reserved

ABSTRACT

EFFICIENT HEVC AND VVC MOTION ESTIMATION HARDWARE

WAQAR AHMAD

Electronics Engineering PhD Thesis, 2021

Thesis Supervisor: Assoc. Prof. İlker Hamzaoğlu

Keywords: HEVC, VVC, Inter Prediction, Motion Estimation, Approximate Computing, Hardware Implementation, FPGA, Low Energy

The significant increase in digital video usage and spatial and temporal video resolutions, has led to the development of new video coding standards, which can compress more without causing quality loss. HEVC and VVC are the two latest video coding standards. VVC is the most recent standard and it is more computationally complex than HEVC. Motion estimation (ME) is used in these standards to remove temporal redundancies between successive video frames. ME accounts for at least 50% and as much as 70% of the total encoding time in both these standards. Approximate computing is an emerging technique to design efficient hardware for error-tolerant applications such as video coding.

In this thesis, an efficient HEVC ME hardware is proposed. An approximate adder, suitable for absolute difference operation, is proposed and integrated to this HEVC ME hardware. Detailed comparison of several approximate circuits including the proposed approximate adder and traditional bit truncation technique for HEVC ME is presented. The proposed approximate adder achieved up to 10% power reduction in ME hardware while providing better quality than the other approximate circuits.

An efficient hardware for translational VVC ME is also proposed. It is the first VVC ME hardware in the literature. The proposed hardware reduces the memory accesses significantly by using an efficient data access and reuse method. It uses a novel adder tree

to minimize hardware area while meeting real-time video encoding requirements. It is capable of processing up to 30 4K video frames per second.

An efficient approximate sum of absolute differences (SAD) hardware is proposed for FPGAs. It utilizes the unused LUT inputs of an FPGA to reduce area and power consumption while providing an almost accurate result. The proposed approximate SAD hardware uses up to 20% less LUTs and consumes up to 38% less power than the smallest and lowest power-consuming approximate SAD hardware in the literature, respectively. The proposed SAD hardware can be used in HEVC and VVC ME hardware.

Finally, a methodology is proposed for designing low error efficient approximate adders for FPGAs. Two approximate adders for FPGAs are designed using the proposed methodology: low error and area efficient approximate adder (LEADx), and area and power efficient approximate adder (APEx). LEADx has lower mean square error than the approximate adders in the literature. APEx is the smallest and lowest power consuming FPGA-based adder in the literature. These approximate adders are integrated to ME in HEVC software encoder. LEADx provided better quality than the other approximate adders for HEVC video coding.

ÖZET

VERIMLI HEVC VE VVC HAREKET TAHMINI DONANIMLARI

WAQAR AHMAD

Elektronik Mühendisliği, Doktora Tezi, 2021

Tez Danışmanı: Doç. Dr. İlker Hamzaoğlu

Anahtar Kelimeler: HEVC, VVC, Çerçeveler Arası Öngörü, Hareket Tahmini, Yaklaşık Hesaplama, Donanım Gerçekleme, FPGA, Düşük Enerji

Sayısal video kullanımındaki, uzamsal ve zamansal video çözünürlüklerindeki önemli artışlar nedeniyle, kalite kaybına neden olmadan daha fazla sıkıştırma yapan video kodlama standartları geliştirilmektedir. HEVC ve VVC en son geliştirilen video kodlama standartlarıdır. En yeni standart olan VVC HEVC'den daha fazla hesaplama karmaşıklığına sahiptir. Hareket Tahmini (HT) ardışık video çerçevelerindeki zamansal artıklıkları azaltmak için kullanılır. HEVC ve VVC standartlarındaki toplam kodlama süresinin en az %50'si ile en fazla %70'ini HT almaktadır. Yaklaşık hesaplama, video kodlama gibi hatalara dayanıklı uygulamalar için verimli donanım tasarlamak için kullanılan yeni bir tekniktir.

Bu tezde, verimli bir HEVC HT donanımı önerilmiştir. Mutlak fark işlemi için uygun bir yaklaşık toplayıcı önerilmiş ve HEVC HT donanımına entegre edilmiştir. Geleneksel bit kesme yöntemi ve önerilen yaklaşık toplayıcı da dahil olmak üzere, yaklaşık devrelerin HEVC HT donanımında kullanımları detaylı karşılaştırmıştır. Önerilen yaklaşık toplayıcı %10'a kadar güç azalması sağlamış ve diğer yaklaşık devrelerden daha kaliteli sonuçlar vermiştir.

Bir verimli VVC öteleme HT donanımı önerilmiştir. Bu literatürdeki ilk VVC HT donanımıdır. Önerilen donanım verimli veri erişimi ve yeniden kullanma yöntemi kullanarak bellek erişimini önemli miktarda azaltmaktadır. Önerilen donanım özgün

toplayıcı ağacı kullanarak donanım alanını azaltmasına rağmen gerçek zamanlı video kodlamaktadır. Önerilen donanım saniyede 30 tane 4K video çerçevesi işleyebilmektedir.

FPGA'lar için verimli yaklaşık mutlak farklar toplamı (MFT) donanımı önerilmiştir. Önerilen MFT donanımı, FPGA'daki LUT'ların kullanılmayan girdilerini kullanarak alan ve güç tüketimini azaltmakta ve neredeyse tam doğru sonuç vermektedir. Önerilen yaklaşık MFT donanımı, literatürdeki en küçük yaklaşık MFT donanımından %20 daha az LUT kullanmakta ve literatürdeki en az güç tüketen yaklaşık MFT donanımından %38 daha az güç tüketmektedir. Önerilen MFT donanımı HEVC ve VVC HT donanımlarında kullanılabilir.

Son olarak, FPGA'lar için düşük hatalı verimli yaklaşık toplayıcı tasarlama yöntemi önerilmiştir. Önerilen yöntem kullanılarak FPGA'lar için iki yaklaşık toplayıcı tasarlanmıştır: düşük hatalı ve alan verimliliği yüksek yaklaşık toplayıcı (LEADx), ve alan ve güç verimliliği yüksek yaklaşık toplayıcı (APEx). LEADx literatürdeki yaklaşık toplayıcılardan daha düşük ortalama kare hatasına sahiptir. APEx literatürdeki en küçük ve en az güç tüketen FPGA tabanlı toplayıcıdır. Bu yaklaşık toplayıcılar HEVC kodlayıcı yazılımındaki HT'ne entegre edilmiştir. LEADx HEVC video kodlamada diğer yaklaşık toplayıcılardan daha kaliteli sonuçlar vermiştir.

ACKNOWLEDGEMENTS

During my PhD, I had the opportunity to meet and work with some amazing people. Each one of them brought forward a unique perspective that helped me grow personally and professionally. While I truly cherish my relationship with all of them, I would like to take this opportunity to name the people without whose contribution I could not have completed this thesis.

First and foremost, I would like to extend my gratitude to my advisor, Dr. İlker Hamzaoğlu, for his invaluable guidance, help, and supervision. His technical expertise and knowledge have been instrumental in my professional development. He believed in me all along and motivated me to do more when I had lost hope. He taught me how to approach a challenging research problem and always provided constructive comments about my work.

I want to present my utmost gratitude to my thesis progress committee members, Dr. Ahmet Onat and Dr. Özgür Gürbüz. They not only provided insightful remarks to improve the quality of this thesis but also supported me in difficult times. I am also thankful to Dr. Mustafa Altun and Dr. Tuba Ayhan for accepting to be part of my thesis jury and for their valuable feedback.

I want to thank Higher Education Commission, Pakistan for supporting this thesis in part. I also want to thank Scientific and Technological Research Council of Turkey (TUBITAK) for supporting this thesis in part, under contract number 118E134.

I would like to thank Berke Ayrancioğlu for the technical discussions and informal conversations we had during my time at Sabanci University. I am also thankful to all the members of Pakistani graduate students' community at Sabanci University for creating a brotherly environment. We celebrated many events together, played friendly yet competitive sports, shared our food, and had thoughtful discussions. Their presence never let me feel alone in a foreign land without family.

I am grateful to my family for their unconditional love and never-ending support. My parents, Mian Abdul Basit and Naheed Farhat, always made sure that I get the best of everything in my life. No words can be enough to thank them for the contributions and sacrifices they made for my success. I am also thankful to my siblings for always pampering me with love and care. My brother, Suhail Basit, deserves a special mention as he always provided me the best guidance whenever needed and ensured that I have the freedom to follow my dreams. My special thanks go to my wife, Aleena, for her never-ending love and for taking care of our little bundle of joy, Maheen Basit. Last but not the least, I am grateful to my parents-in-law for being supportive and caring all the time.

TABLE OF CONTENTS

Abstract iv

Özet vi

Acknowledgements viii

List of Tables xi

List of Figures xii

List of Abbreviations xiv

1 Introduction 1

1.1 Video Coding Fundamentals 3

1.2 Video Coding Standards 4

1.2.1 High Efficiency Video Coding (HEVC) Standard 5

1.2.2 Versatile Video Coding (VVC) Standard 6

1.3 Motion Estimation 6

1.4 Thesis Contributions 7

1.5 Thesis Outline 9

2 Approximate Circuits for HEVC Motion Estimation 10

2.1 Assessment of Approximate Circuits in Absolute Difference Hardware . 12

2.2 Motion Estimation Hardware 17

2.3 Assessment of Approximate Circuits in Motion Estimation Hardware ... 18

3 An Efficient Versatile Video Coding Motion Estimation Hardware 22

3.1 VVC Motion Estimation 23

3.2 Proposed VVC Motion Estimation Hardware 25

3.2.1 Memory and Systolic PE Array 27

3.2.2 SAD Adder tree 30

3.2.3 Comparator 33
3.3 Implementation Results 33
3.3.1 Comparison With HEVC ME Hardware 35
4 An Efficient Approximate SAD Hardware for FPGAs 37
4.1 Background 38
4.1.1 SAD Hardware 38
4.1.2 Xilinx Virtex FPGA 39
4.2 Proposed Approximate SAD Hardware 41
4.3 Implementation Results 43
5 Low Error Efficient Approximate Adders for FPGAs 47
5.1 Background 49
5.1.1 Related Works 49
5.1.2 Length of carry 50
5.1.3 Xilinx Virtex FPGA 51
5.2 Proposed Design Methodology 52
5.2.1 Proposed low error and area efficient approximate adder 55
5.2.2 Proposed area and power efficient approximate adder for FPGAs ... 58
5.3 Experimental Results and Discussion 60
5.3.1 Error Metrics 61
5.3.2 Implementation Results 64
5.3.3 Comparison with Segmented and Speculative Approximate Adders 66
5.3.4 Case Study: Motion Estimation in Video Encoding 68
6 Conclusions 69
Bibliography 70

LIST OF TABLES

Table 2.1 HEVC MSE Results 19
Table 3.1 Number of possible partitions and unique motion vectors in a 64x64 CU..... 26
Table 3.2 Performance of the proposed VVC ME hardware for different configurations 34
Table 3.3 Resource usage for 128x128 CTU size 35
Table 3.4 Resource usage for 64x64 CTU size 35
Table 3.5 Comparison with HEVC ME Hardware 36
Table 4.1 Quality comparison for 4×4 approximate SAD 43
Table 4.2 Reductions achieved by proposed approximate SAD hardware 44
Table 5.1 Probability of the length of a carry being equal to L bits 51
Table 5.2 Effects of Increasing the Number of Bits (k) for Carry Prediction in a 64-Bit Approximate Adder with 12-Bits LSP 53
Table 5.3 Truth Table of Proposed 2-BIT Approximate Adder (AAd2) used for Approximation in least-significant m–2 bits of LEADx 56
Table 5.4 Error Characterization of Constant Approximate Functions for 1-Bit Addition 58
Table 5.5 Error Metrics of 64-Bit Approximate Adders 63
Table 5.6 FPGA Implementation Results of 16-Bit Adders with 8-Bit Approximation 64
Table 5.7 Comparison of 16-Bit Proposed Approximate Adders with 16-Bit Segmented and Speculative Approximate Adders 67
Table 5.8 Impact of Approximate Adders on HEVC Encoder Bitrate And PSNR 68

LIST OF FIGURES

Figure 1.1 Global consumer internet traffic - trends and forecasts 2
Figure 1.2 Country-wise estimate of daily video consumption time by device type..... 2
Figure 1.3 Workflow of a video system 3
Figure 1.4 Development history of video coding standards 4
Figure 1.5 Overview of a typical block-based encoder 5
Figure 1.6 Partitioning of a CU into a PU in HEVC 6
Figure 1.7 The motion estimation process 7
Figure 2.1 Approximate Circuits Assessment Framework 12
Figure 2.2 (a) Proposed 1-bit Approximate FA (b) n-bit Approximate Subtractor 14
Figure 2.3 Approximate Adders (a) IMPACT-1 (b) IMPACT-2 14
Figure 2.4 m-bit LOA 14
Figure 2.5 (a) GeAr Adder (N=8, R=2, P=4) (b) NAAD 0 (c) NAAD 2 15
Figure 2.6 (a) Baseline 1 AD Hardware (b) Baseline 2 AD Hardware 15
Figure 2.7 Assessment Results (a) Percentage Accuracy (b) Average Error (c) Mean Squared Error (d) Standard Deviation (e) Maximum Frequency (f) Energy Consumption 16
Figure 2.8 HEVC Motion Estimation Hardware 17
Figure 2.9 HEVC ME Processing Unit 17
Figure 2.10 (a) Power Reduction (%) (b) Slice Reduction (%) (c) LUT Reduction (%) 21
Figure 3.1 Allowed partitions in VVC. 24
Figure 3.2 An example of QTMT partitioning of 128x128 CTU and its decision tree. . 25
Figure 3.3 Examples of redundant partitions in VVC. 25
Figure 3.4 Proposed VVC ME hardware 27
Figure 3.5 Systolic processing element (PE) array and registers. 28
Figure 3.6 Processing Element (PE) 28
Figure 3.7 (a) Vertical snake scan order (b) Data re-use in downward, upward, and right directions. 29
Figure 3.8 4x4 SAD calculation. 31

Figure 3.9 SAD adder tree (a) SADs of BH, BV, Q partitions N = 4, 8, 16, 32 (b) SADs of BH_BH, BV_TH, BH_TV, BV_BV, TH, TV partitions N = 8, 16 (c) SADs of TH_TH, TH_BH, BH_TH, BV_TV, TV_BV, TV_TV partitions N = 16. 31
Figure 3.10 128x128 SAD calculation. 32
Figure 4.1 Generic SAD hardware 38
Figure 4.2 Accurate absolute difference hardware 39
Figure 4.3 Simplified architecture of a slice in Xilinx Virtex 5/6/7 FPGAs 40
Figure 4.4 Proposed adder/subtractor including complement operations..... 41
Figure 4.5 Implementation of the proposed n-bit adder/subtractor including complement operations..... 41
Figure 4.6 Proposed approximate 2×1 SAD hardware..... 42
Figure 4.7 Implementation results for 8-bit inputs 45
Figure 4.8 Implementation results for 16-bit inputs 46
Figure 5.1 Architecture of approximate full-adder based n-bit approximate adders..... 48
Figure 5.2 Proposed 2-bit approximate adder (AAd1) used in MSBs of LSP. 54
Figure 5.3 Architecture of proposed approximate adders for FPGAs. 54
Figure 5.4 Proposed n-bit low error and area efficient approximate adder (LEADx).... 57
Figure 5.5 Example of 16-bit LEADx with 8-bit approximation. 57
Figure 5.6 Proposed n-bit area and power efficient approximate adder (APEx)..... 60
Figure 5.7 Example of 16-bit APEx with 8-bit approximation. 60
Figure 5.8 Error distribution and error metrics of 16-bit approximate adders with 8-bit approximation. 62
Figure 5.9 Comparison of 32-bit approximate adders with 4-bit to 20-bit approximation (left to right). (a) LUTs vs MSE. (b) Power vs MSE..... 65
Figure 5.10 Area, Power, and Delay reduction achieved with 16-bit approximation in 64-bit approximate adders compared to 64-bit accurate adder. 66

LIST OF ABBREVIATIONS

BRAM	Block RAM
CTU	Coding Tree Unit
CU	Coding Unit
DCT	Discrete Cosine Transform
DST	Discrete Sine Transform
FPGA	Field Programmable Gate Array
HEVC	High Efficiency Video Coding
HM	HEVC Reference Software
LUT	Look Up Table
ME	Motion Estimation
MSE	Mean Squared Error
MV	Motion Vector
PSNR	Peak Signal-to-Noise Ratio
PU	Prediction Unit
QP	Quantization Parameter
QTMT	Quadtree plus Multi-type tree
SAIF	Switching Activity Interchange Format
TU	Transform Unit
VVC	Versatile Video Coding

Chapter 1

INTRODUCTION

In the last decade, the production, distribution, and consumption of digital video has grown at an extraordinary pace. The number of consumer electronics devices that can capture, process, store and transmit digital video has significantly increased. The video streaming services have become popular. The video calls have become part of daily life. The ongoing COVID-19 pandemic has further increased the digital video consumption as the demand for video conferencing and online education significantly increased.

In addition, the continuously increasing demand for higher temporal and spatial resolutions, high dynamic range (HDR) video, and immersive video further increased the amount of digital video that needs to be stored and transmitted. According to CISCO Visual Networking Index, the video content will have more than 82% share in the total internet traffic by 2022, as shown in Figure 1.1 [1].

Efficient video compression, therefore, is a critical need to enable all these applications under limited bandwidth and storage capacity. The two latest video coding standards, High Efficiency Video Coding (HEVC) and Versatile Video Coding (VVC), are developed to fulfill this need [2].

These video coding standards have very high computational complexity. Most of the video content is consumed on battery-powered devices as shown in Figure 1.2 [3]. Software implementations of these video coding standards either do not satisfy real-time performance (frames per second) or power consumption requirements for power-constrained devices. Hardware implementations, on the other hand, satisfy these requirements.

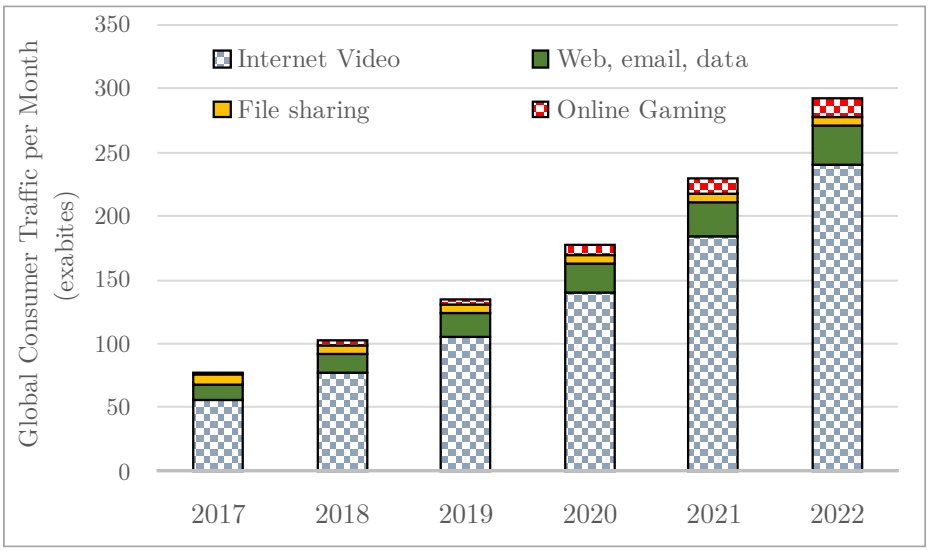


Figure 1.1 Global consumer internet traffic - trends and forecasts

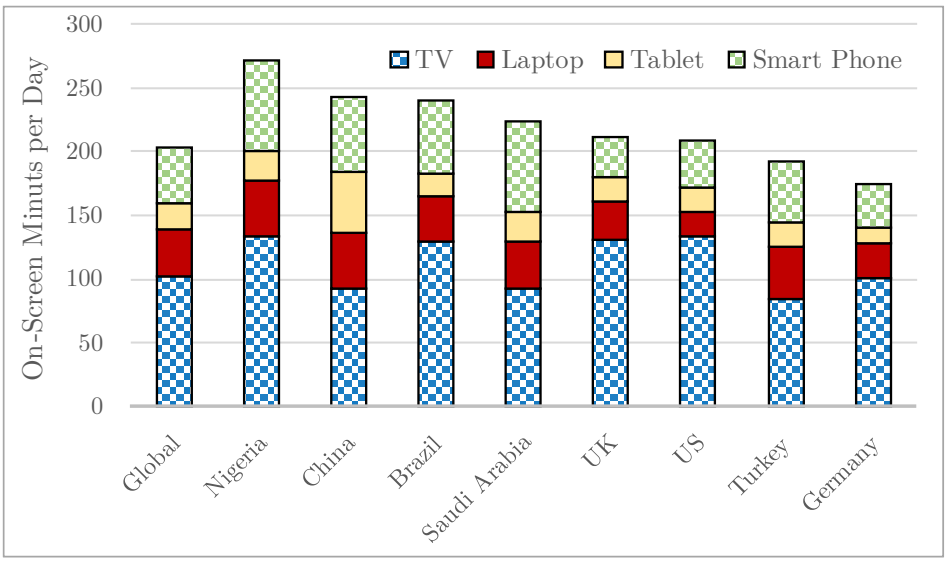


Figure 1.2 Country-wise estimate of daily video consumption time by device type.

Approximate computing is a new design technique that trades off accuracy for performance, area and/or power consumption for error-tolerant applications such as video coding. The video compression is error-tolerant in nature since the only requirement is to produce output that has sufficient quality to provide good user experience. Therefore, approximate computing has a huge potential to improve the performance, area and/or power consumption of hardware implementations of video coding standards.

1.1 Video Coding Fundamentals

A video is a sequence of images captured at high enough rate to create the visual perception of smooth motion. These images (or frames) have four types of redundancies: perceptual, spatial, temporal, and statistical. A video encoder compresses a video by removing these redundancies.

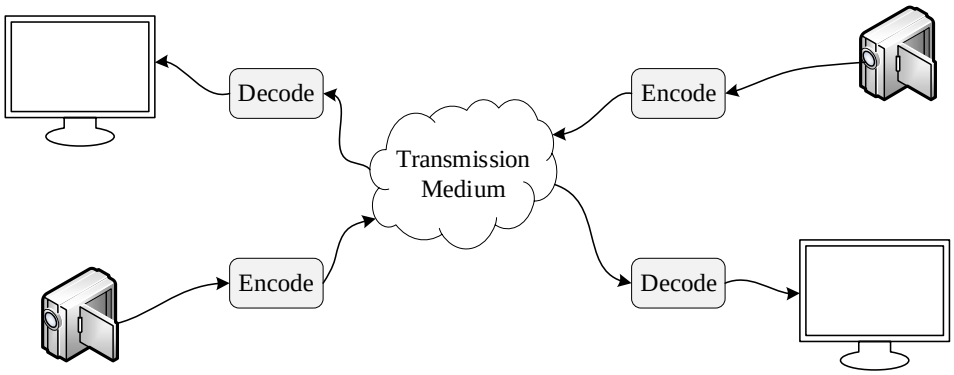


Figure 1.3 Workflow of a video system

The typical flow of a video system involves an encoder at the transmitting end and a decoder on the receiving end as shown in Figure 1.3. The video encoder produces a bit stream representing the video and the video decoder decodes that bitstream to reproduce the video. The encoder and decoder must agree on the syntax of the bitstream. A video coding standard defines the syntax of the encoded bitstream. The video encoder tries to find the best possible way to compress the video, whereas the video decoder decodes the encoded video by following the syntax of the bitstream. Therefore, video encoder has significantly higher computational complexity than video decoder.

The peak-signal-to-noise-ratio (PSNR) metric is often used to measure the quality of an encoded video. The PSNR provides a measure of relative error between original and decoded video. The average amount of bits required to encode one second of video is usually referred to as the bitrate. The efficiency of a video encoder is usually measured in terms of rate distortion (RD) performance. The RD performance is visualized by plotting the PSNR against the corresponding bitrate over a range of operating points. This graphical representation is called RD curve.

1.2 Video Coding Standards

In the last three decades, several video coding standards are developed by ITU-T and ISO standardization organizations independently or with the combined effort of their Joint Collaborative Team on Video Coding (JCT-VC). The development history of these video coding standards is shown in Figure 1.4. Each new video coding standard provides higher coding efficiency than its predecessor.

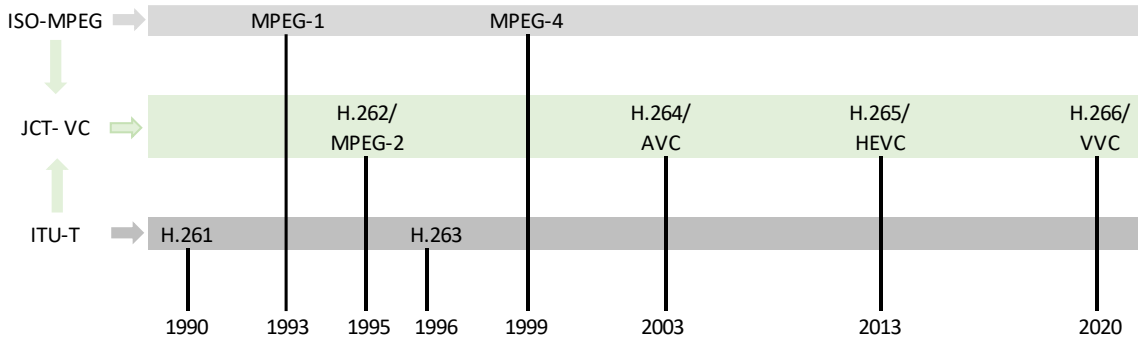


Figure 1.4 Development history of video coding standards

These video coding standards use the same block-based hybrid coding model shown in Figure 1.5 [2]. In the video encoder, there is has a forward path to generate bitstream and a reconstruction path to ensure that identical reference frames are used in both video encoder and decoder. Each video frame is divided into small blocks. Each block is coded individually in raster scan order. Each block is predicted by intra prediction and inter prediction (motion estimation). Mode decision determines the best prediction. Predicted block is subtracted from the current block, and the resulting residual block is transformed, quantized and entropy coded.

The latest video coding standards HEVC and VVC also use this block-based hybrid coding model. However, they use different algorithms for intra prediction, inter prediction and transform.

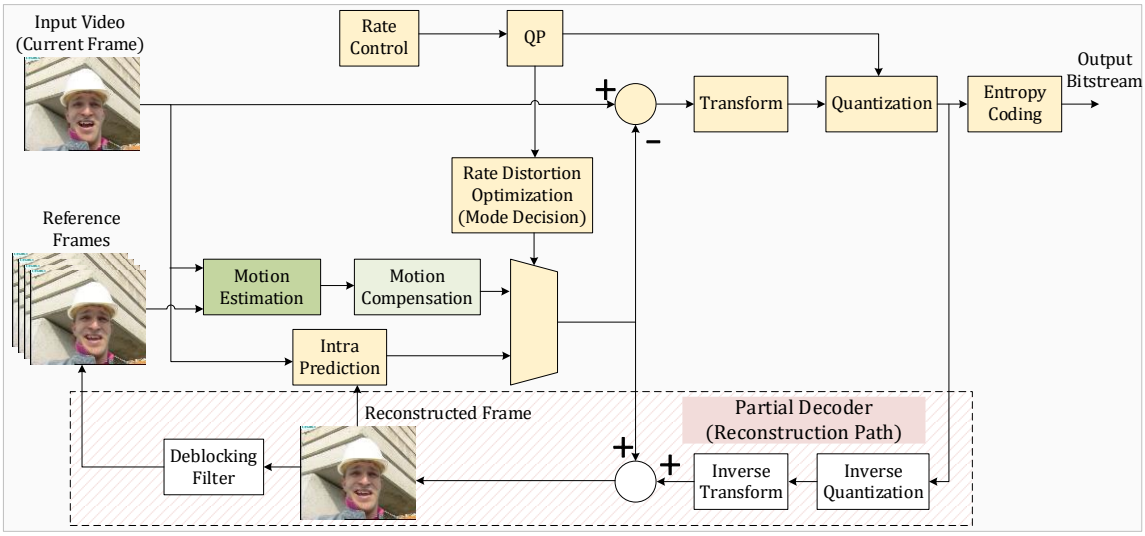


Figure 1.5 Overview of a typical block-based encoder

1.2.1 High Efficiency Video Coding (HEVC) Standard

HEVC standard was developed in 2013 [4]. HEVC provides 50% better coding efficiency than its predecessor H.264/AVC standard by using several new coding tools [5]. HEVC replaces the macroblock used in previous video coding standards with the coding tree unit (CTU). It uses a new block partitioning structure based on a quadtree. The largest CTU size in HEVC is 64x64 whereas the largest macroblock size in H.264 is 16x16. The size of a CTU in HEVC can be 16x16, 32x32, or 64x64. The CTU can be partitioned into coding units (CUs) recursively using quadtree structure. The size of a CU can be 8x8, 16x16, 32x32, or 64x64. The mode decision between intra prediction and inter prediction is done at the CU level. A CU can be further partitioned into prediction units (PU). A PU can be square, rectangular, or asymmetric as shown in Figure 1.6. The size of a PU can vary from 8x4 or 4x8 to 64x64. The asymmetric and rectangular partitions can only be used for inter-prediction and only square partitions can be used for intra prediction.

HEVC uses three types of intra prediction, DC, planar, and angular. The angular prediction supports up to 33 directions. The intra PU size can be as small as 4x4. HEVC inter prediction is computationally more complex than H.264 inter prediction because of the larger block size and larger number of block partitions.

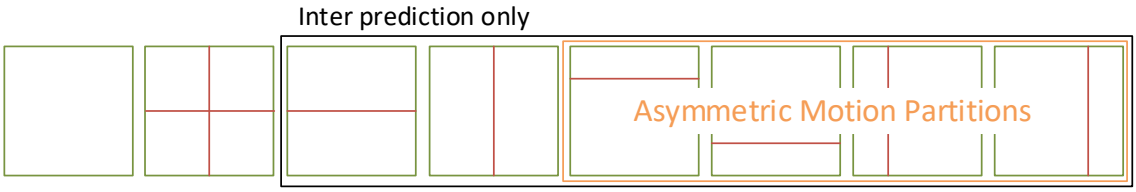


Figure 1.6 Partitioning of a CU into a PU in HEVC

1.2.2 Versatile Video Coding (VVC) Standard

VVC is the latest video coding standard developed in 2020 by the Joint Video Experts Group (JVET) of ITU and ISO standardization organizations [6]. VVC offers 50% better coding efficiency than HEVC and 75% better coding efficiency than H.264 standard at the cost of significant increase in computational complexity [7]. VVC is designed to be versatile, i.e., it supports encoding diverse type of video content such as high dynamic range, 360º video, and virtual reality.

The largest CTU size in VVC is 128x128. VVC uses a new quadtree plus multi-type tree (QTMT) structure which allows more flexible block partitions than HEVC. VVC uses translational motion estimation used in previous video coding standards. However, it also uses affine motion estimation and bi-directional optical flow to predict more complex motion. The intra prediction in VVC is also more complex than HEVC as it uses 93-direction angular prediction. VVC uses multiple primary transforms (DCT, DST) and a low frequency non-separable secondary transform.

1.3 Motion Estimation

These video coding standards use block matching for motion estimation. In block matching, current video frame is divided into blocks. As shown in Figure 1.7, for each block in the current frame, the best matching block in a search window (SW) in the reference frame and the corresponding motion vector (MV) are determined.

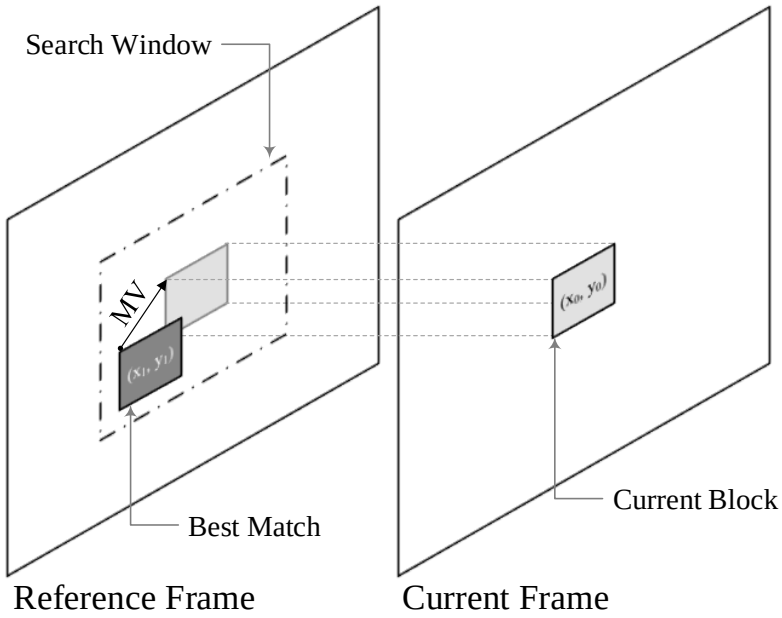


Figure 1.7 The motion estimation process

Sum of absolute differences (SAD) metric is typically used to determine the best matching block. The SAD between two blocks, A and B, of size W x H is calculated as shown in (1.1), where A(i,j) and B(i,j) are values of the pixels in ith row and jth column of A and B, respectively.

$$SAD = \sum_{i=0}^{W-1} \sum_{j=0}^{H-1} |A(i,j) - B(i,j)| \tag{1.1}$$

ME is the most computationally complex and memory intensive module in all the video coding standards. Its complexity has increased in HEVC and VVC standards. In HEVC encoder, ME accounts for up to 83% and on average 70% of the total encoder complexity [8]. In VVC encoder, ME accounts for on average 65% of the total encoder complexity [9]. Therefore, efficient HEVC and VVC ME hardware implementations are necessary to perform real-time video coding.

1.4 Thesis Contributions

This thesis makes the following technical contributions:

We propose an efficient HEVC ME hardware. An approximate adder, suitable for absolute difference operation, is proposed and integrated to this HEVC ME hardware. A framework is proposed to compare the performance of approximate circuits. Detailed

comparison of several approximate circuits including the proposed approximate adder and traditional bit truncation technique for HEVC ME is presented. The proposed approximate adder achieved up to 10% power reduction in the ME hardware while providing better quality than the other approximate circuits.

We propose an efficient translational VVC motion estimation hardware. It is the first VVC ME hardware in the literature. It supports maximum coding tree unit size of 128x128 using a 64x64 systolic processing element array and a novel memory-based SAD adder tree. It reduces memory accesses significantly by using an efficient data access and reuse method. The proposed VVC ME hardware is implemented on a Xilinx Virtex 7 FPGA. It can process up to 30 4K (3840x2160) video frames per second.

We propose an efficient approximate SAD hardware with very small maximum and average error for FPGAs. The proposed approximate SAD hardware utilizes the unused LUT inputs to reduce area and power consumption while providing an almost accurate result. The proposed approximate SAD hardware has smaller maximum and average error than the approximate SAD hardware in the literature. It uses up to 20% less LUTs than the smallest approximate SAD hardware in the literature. It consumes up to 38% less power than the lowest power consuming approximate SAD hardware in the literature.

We propose a methodology for designing low error efficient approximate adders for FPGAs. The proposed methodology utilizes FPGA resources efficiently to reduce the error of approximate adders. We propose two approximate adders for FPGAs using our methodology: low error and area efficient approximate adder (LEADx), and area and power efficient approximate adder (APEx). Both approximate adders are composed of an accurate and an approximate part. The approximate parts of these adders are designed in a systematic way to minimize the mean square error (MSE). LEADx has lower MSE than the approximate adders in the literature. The 32-bit LEADx with 16-bit approximation has 20% lower MSE than the approximate adder with the lowest MSE in the literature. The 16-bit APEx with 8-bit approximation has the same area, 60% lower MSE, and 4.5% less power consumption in Xilinx Virtex 7 FPGA than the smallest and lowest power consuming approximate adder in the literature. APEx has smaller area and lower power consumption than the other approximate adders in the literature. As a case study, the approximate adders are used in video encoding application. LEADx provided better quality than the other approximate adders for video encoding.

1.5 Thesis Outline

The rest of the thesis is organized as follows:

Chapter 2 presents a new approximate adder and an efficient HEVC ME hardware. First, a framework to compare approximate circuits is presented and the performance of approximate circuits is determined using the proposed framework. Then, HEVC ME hardware is explained. Finally, the impact of using approximate circuits in ME hardware is presented.

Chapter 3 presents an efficient VVC ME hardware. First, VVC motion estimation is explained. Then, the proposed VVC ME hardware is explained. Finally, its implementation results and comparison with HEVC ME hardware in the literature are presented.

Chapter 4 presents an approximate SAD hardware for ME. First, accurate absolute difference hardware implementations are explained, and an overview of the Xilinx Virtex FPGAs is provided. Then, the proposed approximate SAD hardware is explained. Finally, its implementation results and comparison are presented.

Chapter 5 presents low error efficient approximate adders for FPGAs. First, the concept of length of carry is presented. Then, the methodology proposed for designing low error efficient approximate adders for FPGAs is explained. Then, the proposed approximate adders and the mathematical models to compute their error metrics are explained. Then, their error analyses and implementation results are presented. Finally, the results of using approximate adders in video encoding are presented.

Chapter 6 concludes this thesis and presents potential future work.

Chapter 2

APPROXIMATE CIRCUITS FOR HEVC MOTION ESTIMATION

Video coding is a very computationally complex process and the growing demand for ultra-high-definition video has led to development of more computationally complex video coding standards. The current state-of-the-art video coding standard, High Efficiency Video Coding (HEVC), provides 50% better compression efficiency compared to H.264 video coding standard, at the expense of more computational complexity [5]. Versatile Video Coding (VVC) standard is expected to provide better compression efficiency than HEVC standard at the expense of even more computational complexity [2].

Motion estimation (ME) is the most computationally complex and power consuming module in video encoder hardware. Block matching ME is used in H.264, HEVC and VVC 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 for a current block of H x W pixels is defined as

$$SAD = \sum_{x=1}^{H} \sum_{y=1}^{W} |Cur(x,y) - Ref(x,y)| \tag{2.1}$$

where Cur(x,y) is value of the pixel in (x,y) position of the current block and Ref(x,y) is 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 search window. For example, for full search ME algorithm with 16x16 search window, 256 search locations should be searched. For full search ME algorithm with 128x128 search window, 16384 search locations should be searched. Number of arithmetic operations required for calculating SAD values for a

search location depends on size of the largest coding block, and number and sizes of its sub-blocks.

In HEVC, the largest coding block size is 64x64 and it has 593 sub-blocks. 4096 absolute difference and 4517 addition operations are required to calculate 593 SAD values of a 64x64 block in the current frame for one search location. 141 million arithmetic operations are required to calculate SAD values for 16384 search locations.

Block matching is repeated for all blocks in the current frame. There are 8100 16x16 blocks in a full high definition (1920x1080) image. There are 2025 64x64 blocks in an ultra-high definition (3840x2160) image. Block matching is then repeated for all frames in the video sequence. Video sequences typically have at least 30 frames per second. Therefore, block matching ME requires huge amount of arithmetic operations.

Approximate hardware can achieve better performance, area and power consumption than accurate hardware while providing acceptable quality for error tolerant applications [11]-[13]. Video coding can tolerate small errors [14]. 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 [15], [16]. 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 [17], [18]. 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.

Bit truncation technique is used to increase speed and reduce area and power consumption of ME hardware [19], [23] - [25]. Recently, several works analyzing impact of using approximate circuits in ME hardware are published. Impact of using approximate arithmetic in HEVC ME to reduce computational complexity is analyzed in [20]. An approximate SAD hardware using lower-part-OR adder (LOA) [16] is proposed for HEVC ME in [21]. Impact of using approximate adders in different parts of SAD tree is analyzed in [22]. However, their ME hardware does not support the asymmetric partitions defined in HEVC standard. In addition, quality of approximate adder is analyzed using only error distance and power metrics, speed and area results are not reported.

In this chapter, an approximate adder is proposed. Detailed assessment of using the proposed approximate adder, several generic approximate adders, the approximate

absolute difference hardware proposed in [26] and bit truncation in HEVC ME hardware is presented.

The proposed approximate adder 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 [18] had the worst quality, area and power consumption results.

2.1 Assessment of Approximate Circuits in Absolute Difference Hardware

We assessed impact of using several approximate circuits including the proposed approximate adder and traditional bit truncation technique in absolute difference hardware using the framework shown in Figure 2.1.

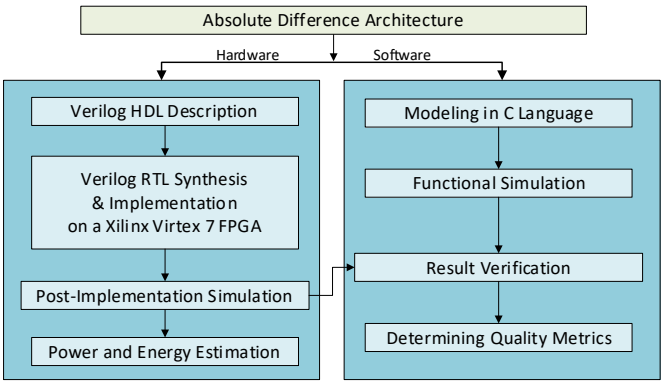


Figure 2.1 Approximate Circuits Assessment Framework

As shown in the figure, for each approximate circuit, the absolute difference hardware using this approximate circuit is described in Verilog HDL. The Verilog RTL code is synthesized and implemented on a Xilinx Virtex 7 FPGA. The FPGA implementation is verified with post-implementation timing simulations. The absolute difference hardware using this approximate circuit is also modeled in C language. Functional simulation results of the C model and post-implementation timing simulation results are compared to verify the C model and the FPGA implementation.

We used percentage accuracy, average error, mean squared error and standard deviation quality metrics as defined in equations (2.2)-(2.7) to assess the quality of using an approximate circuit in absolute difference hardware. In these equations, R is accurate result, R^' is approximate result, X is total number of results, and Y is number of

inaccurate results. We determined these quality metrics for each approximate circuit using its C model.

Absolute Error Value

$$AEV = |R - R'| \tag{2.2}$$

Percentage Accuracy

$$PA = \frac{X - Y}{X} * 100 \tag{2.3}$$

Total Error

$$TE = \sum_{i=1}^{X} AEV_i \tag{2.4}$$

Average Error

$$AE = \frac{TE}{X} \tag{2.5}$$

Mean Squared Error

$$MSE = \frac{1}{X}\sum_{i=1}^{X}(AEV_i)^2 \tag{2.6}$$

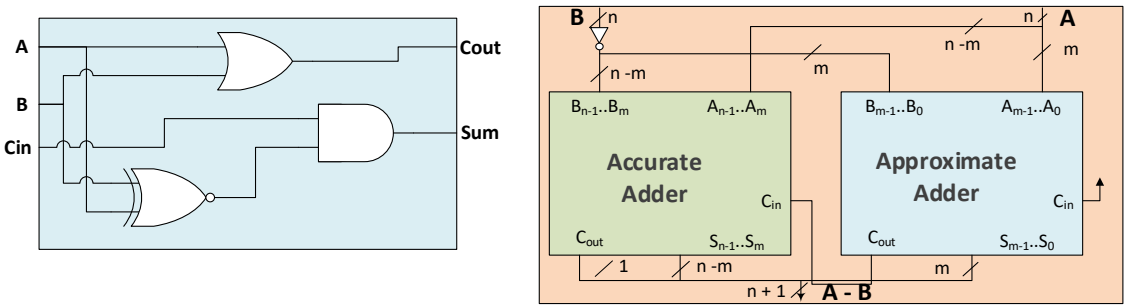
Standard Deviation

$$\sigma = \sqrt{\frac{\sum_{i=1}^{X}(AEV_i - AE)^2}{X}} \tag{2.7}$$

The proposed 1-bit approximate full adder is shown in Figure 2.2 (a). 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. The sum logic is also modified to reduce error magnitude. 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.

An approximate n-bit subtractor can be designed using the proposed 1-bit approximate full adder as shown in Figure 2.2 (b). An approximate m bit adder is used in the least significant m bits of the approximate subtractor. An exact n-m bit adder is used in the most significant n-m bits of the approximate subtractor. In the approximate m-bit

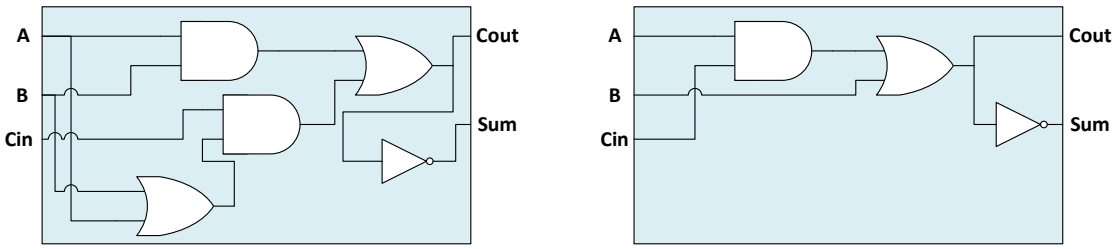
adder, m 1-bit proposed approximate full adders are used. Since carry-out output of each 1-bit full adder is generated without considering the effect of its carry-in input, carry-out outputs of all m 1-bit full adders are generated in parallel. Since the proposed approximate 1-bit full adder generates accurate outputs when carry-in input is 1, approximate n-bit subtractor using the proposed full adder has 100% accuracy when m is 1.



(a) (b)

Figure 2.2 (a) Proposed 1-bit Approximate FA (b) n-bit Approximate Subtractor

Many approximate arithmetic circuits are proposed in the literature. The approximate circuits used in this chapter are selected based on the analysis results reported in literature. In [20], two 1-bit approximate full adders from [15] are determined to give the best performance for ME. In this chapter, these 1-bit full adders are referred to as IMPACT-1 and IMPACT-2. They are shown in Figure 2.3 (a) and Figure 2.3 (b), respectively. An approximate n-bit subtractor can be designed using IMPACT-1 or IMPACT-2 as shown in Figure 2.2 (b). In the approximate m-bit adder, m 1-bit IMPACT-1 or m 1-bit IMPACT-2 full adders are used.



(a) (b)

Figure 2.3 Approximate Adders (a) IMPACT-1 (b) IMPACT-2

In [21] and [22], it is shown that lower-part-OR adder (LOA) [16] performs better than many segmented adders for ME. m-bit LOA is shown in Figure 2.4. An approximate n-bit subtractor can be designed using LOA as shown in Figure 2.2 (b). In this approximate n-bit subtractor, m-bit LOA is used as the m-bit approximate adder.

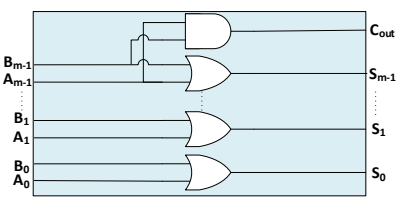


Figure 2.4 m-bit LOA

Generic accuracy reconfigurable adder (GeAr) can be configured to reproduce several other segmented adders [18]. For example, GeAr (N=8, R=1, P=3) is the same as almost correct adder (ACA-I) (N=8, Q=4) [9] and GeAr (N=8, R=2, P=2) is the same as accuracy configurable adder (ACA-II) (N=8, Q=4) [17]. Therefore, we selected GeAr among segmented adders for our analysis. GeAr is shown in Figure 2.5 (a). The novel approximate absolute difference (NAAD) hardware proposed in [26] is also included in our analysis. Two configurations of 8-bit NAAD are shown in Figure 2.5 (b) and Figure 2.5 (c).

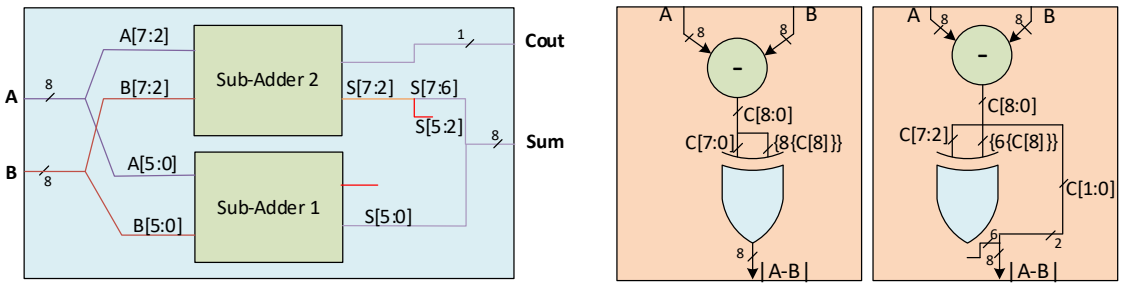
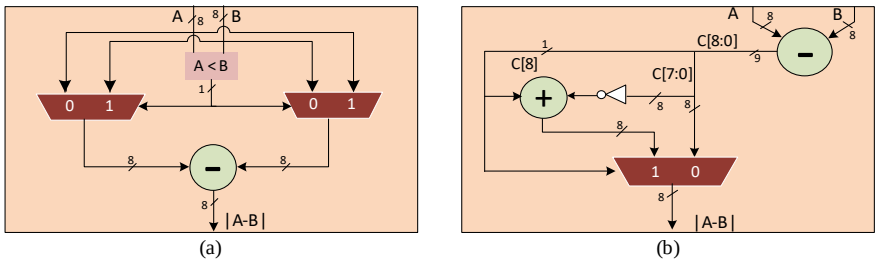


Figure 2.5 (a) GeAr Adder (N=8, R=2, P=4) (b) NAAD 0 (c) NAAD 2

We used the two accurate absolute difference (AD) hardware shown in Figure 2.6 (a) and Figure 2.6 (b) to provide baseline results for our analysis. We assessed impact of using the approximate subtractor shown in Figure 2.2 (b) 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.



(a) (b)

Figure 2.6 (a) Baseline 1 AD Hardware (b) Baseline 2 AD Hardware

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, 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.

The assessment results are shown in Figure 2.7. In the figure, percentage accuracy (PA), average error (AE), mean-squared error (MSE), and standard deviation (SD) quality metric results are shown. In addition, maximum frequencies and energy consumptions of baseline and approximate AD hardware are shown. The results can be interpreted as follows. For PA and maximum frequency, the higher the better. For all other metrics, the lower the better.

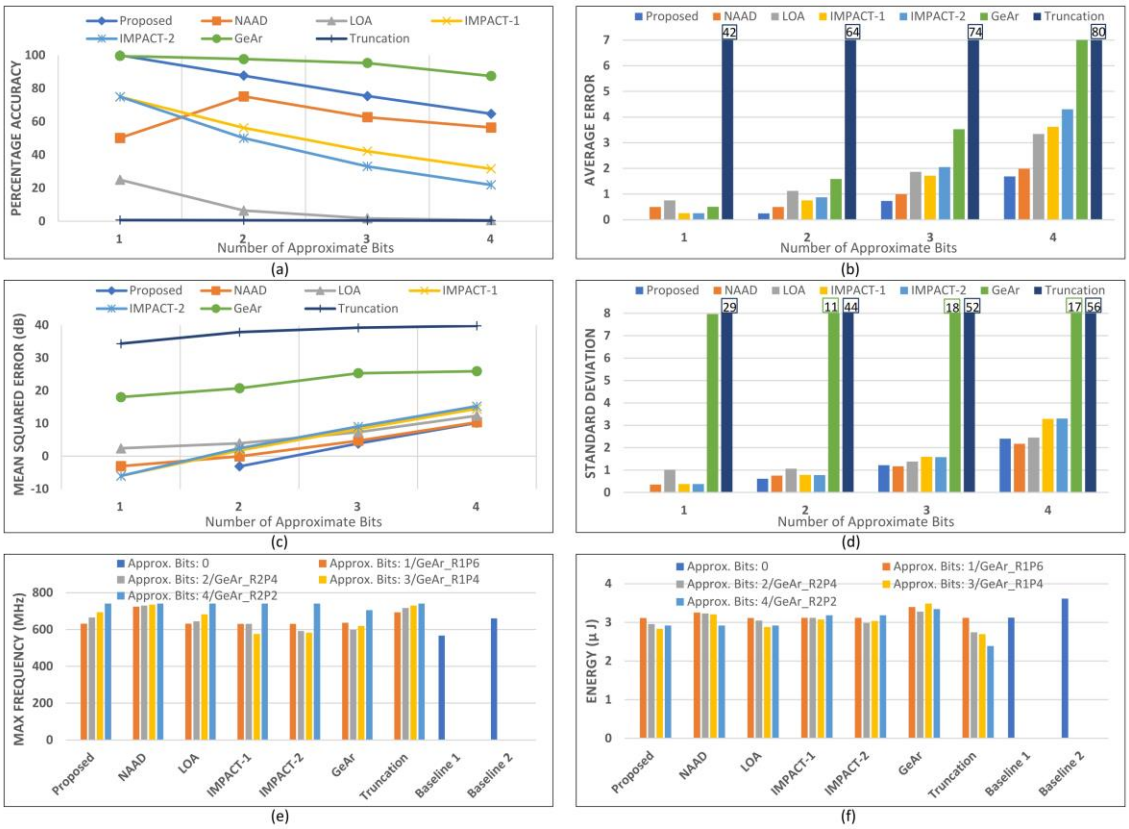


Figure 2.7 Assessment Results (a) Percentage Accuracy (b) Average Error (c) Mean Squared Error (d) Standard Deviation (e) Maximum Frequency (f) Energy Consumption

Traditional bit truncation performs the worst in terms of all quality metrics. PA of the proposed adder is the best among all approximate circuits for 1-bit approximation, and second only to GeAr for 2-bit, 3-bit and 4-bit approximations. However, the proposed adder performs much better than GeAr in other quality metrics. The proposed adder performs the best in terms of AE and MSE quality metrics. It has comparable SD results with other approximate circuits. Traditional bit truncation and NAAD achieve the fastest frequency. Traditional bit truncation, as expected, achieves the lowest energy

consumption due to reduced hardware area. The proposed adder achieves lower energy consumption than the other approximate circuits.

In summary, traditional bit truncation achieves better speed, area and energy consumption at the expense of more quality loss than the other approximate circuits. The proposed adder performs the best in terms of the quality metrics. It achieves the lowest energy consumption among the other approximate circuits. GeAr performs worse than the other approximate circuits in terms of all metrics except PA and speed.

2.2 Motion Estimation Hardware

We designed and implemented an HEVC variable block size full search ME hardware to assess impact of using approximate absolute difference hardware in an HEVC ME hardware. The proposed HEVC ME hardware supports both symmetric and asymmetric block partitions in HEVC standard. Block diagram of this ME hardware is shown in Figure 2.8. Its architecture is similar to the H.264 variable block size full search ME hardware proposed in [27].

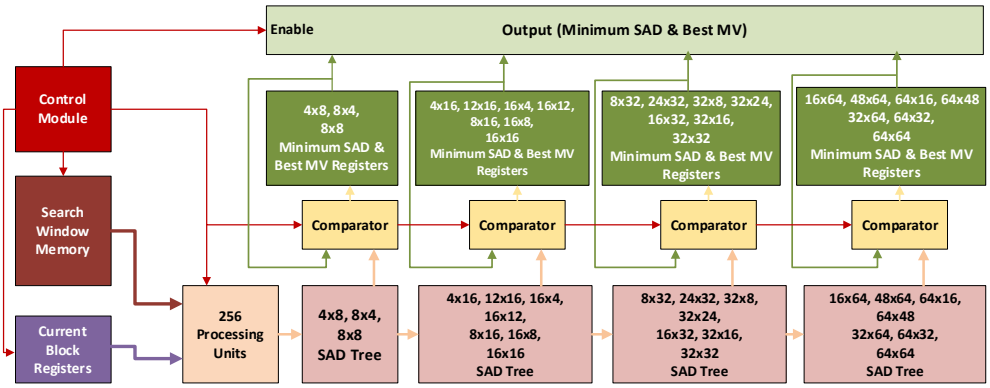


Figure 2.8 HEVC Motion Estimation Hardware

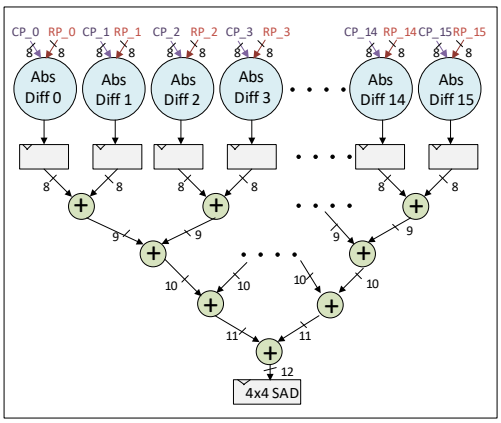


Figure 2.9 HEVC ME Processing Unit

Current block pixels are stored in the current block registers. 128x128 search window pixels are stored in sixty-five 18K Block RAMs (BRAM) in FPGA. 256 processing units are used to calculate absolute differences and 4x4 SAD values for a 64x64 block in parallel. As shown in Figure 2.9, a processing unit uses 16 absolute difference hardware and 15 adders to generate one 4x4 SAD value. The 4x4 SAD values are then used to calculate SAD values of larger block sizes.

The proposed HEVC ME hardware implements snake scan order in vertical direction. Control module keeps track of the scan direction and whenever a change in direction is required, it reconfigures the processing units to receive reference pixels from either top, right, or bottom. The proposed HEVC ME hardware has 9 clock cycles latency; 2 cycles for synchronous read from BRAM, 1 cycle for data loading/shifting, 1 cycle for absolute difference operation, 1 cycle for 4x4 SAD value generation, and 4 cycles to generate 593 SAD values for a search location. It takes 64 clock cycles to read the first 64x64 reference block from search window BRAMs. After that, 593 SAD values for a search location are generated every clock cycle.

SAD trees work in a hierarchical manner by using SAD values of smaller block sizes to calculate SAD values of larger block sizes. For example, 4x4 SAD values are used to calculate 8x4 and 4x8 SAD values, and then 4x8 SAD values are used to calculate 8x8 SAD values. This process continues until SAD value of 64x64 block is calculated. For each sub-block in a 64x64 block, its minimum SAD and corresponding best motion vector (MV) are stored in registers. In every clock cycle, comparators compare the minimum SAD values of the corresponding sub-blocks with the new SAD values calculated in SAD trees. If a new SAD value is smaller than the stored SAD value, comparator stores the new SAD value and MV to the corresponding minimum SAD and best MV registers.

2.3 Assessment of Approximate Circuits in Motion Estimation Hardware

We assessed impact of using the approximate absolute difference hardware presented in Section 2.1 in the HEVC ME hardware presented in Section 2.2. For each approximate absolute difference hardware, we replaced the 4096 exact absolute difference hardware in Verilog RTL code of the HEVC ME hardware with this approximate absolute difference hardware. In addition, we used the baseline 2 accurate absolute difference hardware shown in Figure 2.6 (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.

We also developed behavioral models of the HEVC ME hardware in C language. All Verilog RTL codes are verified with RTL simulations. All FPGA implementations are verified with post-implementation timing simulations. Simulation results matched results of the corresponding C model of the approximate HEVC ME hardware.

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

Table 2.1 presents average MSE results for all HEVC sub-block sizes for Foreman video sequence. 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.

Table 2.1 HEVC MSE Results

Approximate Circuit and its Configuration		HEVC MSE
Baseline		32.36
Proposed	1	32.36
	2	32.33
	3	32.40
	4	32.64
NAAD	0	32.32
	1	32.42
	2	32.53
	3	33.25
LOA	1	32.32
	2	32.33
	3	32.48
	4	33.05
IMPACT-1	1	32.38
	2	32.57
	3	32.85
	4	33.99
IMPACT-2	1	32.37
	2	32.88
	3	32.92
	4	34.57
GeAr	R1_P6	94.78
	R2_P4	69.45
	R1_P4	134.47
	R2_P2	71.31
Bit Truncation	1	32.38
	2	32.48
	3	33.33
	4	35.70

The other approximate circuits perform better than the proposed approximate adder in some cases. These MSE results are consistent with the quality results of approximate absolute difference hardware shown in Figure 2.7.

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.

Area and power consumption results of HEVC ME hardware are shown in Figure 2.10. In the figure, 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 achieves 5% and 3% power reductions in HEVC ME hardware respectively, without affecting quality. For 4-bit approximation, it has the smallest MSE value and the largest power reduction (10%) in HEVC ME hardware compared to other approximate circuits.

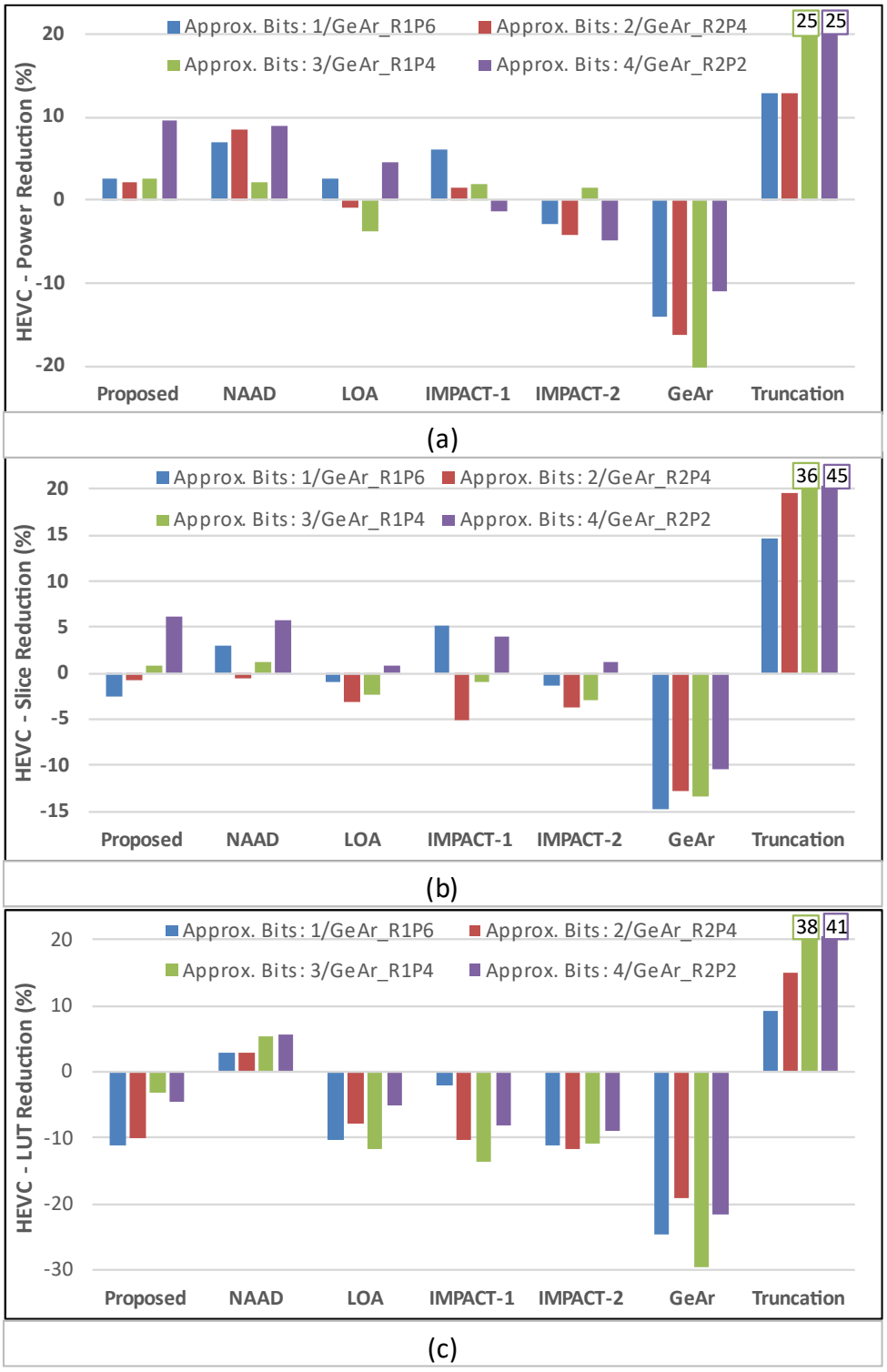


Figure 2.10 (a) Power Reduction (%) (b) Slice Reduction (%) (c) LUT Reduction (%)

Chapter 3

AN EFFICIENT VERSATILE VIDEO CODING MOTION ESTIMATION HARDWARE

As the amount of video data is increasing significantly, more efficient video compression is needed to transmit and store this video data with limited available bandwidth and storage space [1]. Therefore, Joint Video Experts Team (JVET) of ITU-T and ISO standardization organizations developed Versatile Video Coding (VVC) standard in 2020 [6]. VVC provides 50% higher compression efficiency than its predecessor High Efficiency Video Coding (HEVC) standard developed in 2013 [7, 28]. VVC is designed to encode diverse video content such as high dynamic range, 360º video and virtual reality [2].

VVC uses several new encoding tools to achieve better compression than HEVC such as new block partitioning structure called quadtree plus multi-type tree (QTMT), affine motion estimation and multiple transforms [29]. VVC divides a video frame into blocks called coding tree units (CTUs) and encodes each CTU separately. Each CTU can be further divided into coding units (CUs) using QTMT. QTMT allows more partitions than simple quadtree (QT) partitioning used in HEVC. The maximum CTU size in VVC is 128x128. The maximum CTU size in HEVC is 64x64.

VVC achieves higher compression efficiency than HEVC at the cost of significant increase in computational complexity. VVC encoder is 5 times and 31 times more complex than HEVC encoder under Low-Delay and All-Intra configurations, respectively [9]. The encoding time of VVC reference software encoder (VTM) is about 10 times more than the encoding time of HEVC reference software encoder (HM) [30]. Therefore, dedicated hardware implementations are needed for processing high resolution videos in real-time [31].

Successive frames in a video sequence have temporal redundancy. Video coding standards remove this temporal redundancy by performing motion estimation (ME). ME

is the most time consuming and memory intensive module in video encoding [32]. More than 50% of the encoding time of VVC encoder is spent for ME [9]. Up to 60% of the memory accesses of VVC encoder comes from ME module [33].

There are several HEVC ME hardware in the literature [34, 35, 36, 37, 38, 39, 40]. Several sum of absolute differences (SAD) hardware that can be used for ME are proposed in the literature [41, 42]. There are several VVC intra prediction, fractional interpolation and transform hardware in the literature [43, 44, 45, 46]. However, to the best of our knowledge, there is no VVC ME hardware in the literature.

In this chapter, we propose the first VVC ME hardware in the literature. The proposed hardware uses the full search ME algorithm to determine the best motion vector for all the QTMT partitions in a CTU, from 8x4 (4x8) to 128x128. It uses SAD metric to determine the best motion vector. The proposed hardware calculates SADs of 128x128 CTU using a 64x64 systolic processing element array and a novel memory-based SAD adder tree to achieve real-time performance with small hardware area. It reduces memory accesses significantly by using an efficient data access and reuse method.

The proposed VVC ME hardware is implemented using Verilog HDL. It works at 253 MHz on a Xilinx Virtex 7 FPGA, and it can process up to 30 4K (3840x2160) video frames per second (fps).

3.1 VVC Motion Estimation

VVC uses block matching for translational motion estimation. In block matching, current video frame is divided into blocks. As shown in Figure 1.7, for each block in the current frame, the best matching block in a search window (SW) in the reference frame and the corresponding motion vector (MV) are determined. SAD metric is typically used to determine the best matching block. SAD between blocks A and B is calculated as shown below, where WxH is the block size, A(i, j) and B(i, j) are pixels in ith row and jth column of A and B, respectively.

$$SAD = \sum_{i=0}^{W-1} \sum_{j=0}^{H-1} |A(i,j) - B(i,j)| \tag{3.1}$$

Video coding standards perform variable block size block matching motion estimation. Large block sizes achieve higher compression for smooth areas of video frames, whereas small block sizes achieve higher compression for detailed areas of video

frames. Because large block sizes can find good matches for smooth areas, and they require less MVs than small block sizes. However, large block sizes cannot find good matches for detailed areas.

Both HEVC and VVC divide a video frame into blocks called CTU. In HEVC, the maximum CTU size is 64x64. A CTU can be recursively partitioned into square-shaped CUs using QT. The size of a CU can be from 8x8 to 64x64. A CU can be partitioned only once into square, rectangular and asymmetric partitions called prediction unit (PU). The PU size can be from 4x8 or 8x4 to the CU size for motion estimation.

In VVC, the maximum CTU size is 128x128. A CTU can be recursively partitioned into CUs using QTMT [47]. QTMT achieves higher compression than QT used in HEVC by allowing more partitions than QT.

QTMT is a tree in which a node can be split using QT, binary tree (BT) or ternary tree (TT). A BT splits a node into two rectangular blocks. A TT splits a node into three rectangular blocks, two of which have the same size. BT and TT splits can be applied in horizontal or vertical direction. In Figure 3.1, five possible QTMT partitions are shown; binary horizontal (BH), binary vertical (BV), ternary horizontal (TH), ternary vertical (TV), quad (Q).

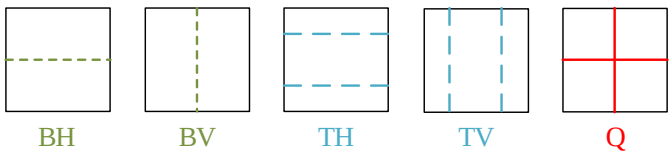


Figure 3.1 Allowed partitions in VVC.

There are some restrictions in QTMT partitioning [48]. If a node is split with QT, it can be further split with any of the five QTMT partitions. However, if a node is split with either BT or TT, it can no longer be further split with QT. An example of QTMT partitioning of 128x128 CTU is shown in Figure 3.2.

As shown in Figure 3.3, the same partitions can be achieved with different splitting patterns. If the central partition of a TT split is further split with BT in the same direction, it achieves the same partitions with BT split followed by BT split in the same direction. Similarly, QT split followed by QT split achieves the same partitions with BT split in one direction followed by BT split in the other direction. VVC does not allow these redundant partitions [48].

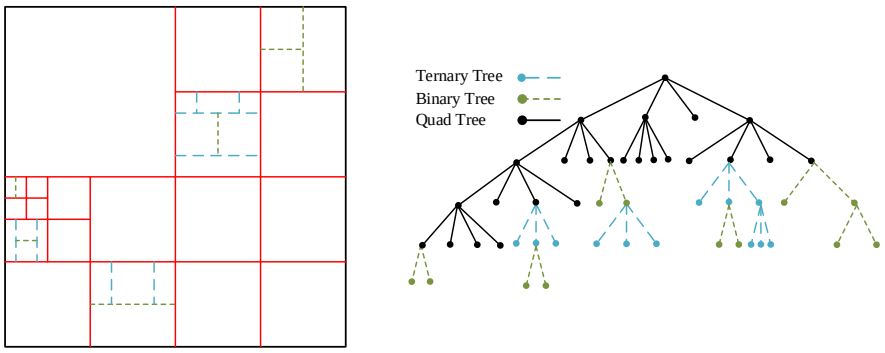


Figure 3.2 An example of QTMT partitioning of 128x128 CTU and its decision tree.

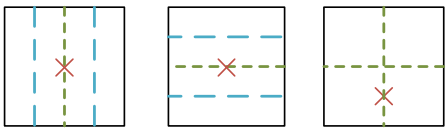


Figure 3.3 Examples of redundant partitions in VVC.

In addition to translational ME, VVC uses affine motion estimation (AME) to predict more complex motion such as rotation or scaling. Turning off AME in VVC video encoding causes 3% loss in compression efficiency but provides 20% encoding time reduction [49]. It is reported in [50] that translational motion estimation is used for more than 90% cases in VVC video encoding. Therefore, in this chapter, we propose an efficient ME hardware for VVC translational ME.

3.2 Proposed VVC Motion Estimation Hardware

VVC defines the following control parameters to adjust the computational complexity of ME by restricting the number of partitions.

- *MaxCUWidth* and *MaxCUHeight* define the maximum allowed width and height of a CU, respectively.
- *MinQTSize* defines the minimum node size that can be reached with QT split.
- *MaxBtSize* and *MaxTtSize* define the maximum node size to which BT and TT split can be applied, respectively.
- *MaxMttDepth* defines the maximum allowed depth of multi-type tree splitting after QT split.

In the proposed VVC ME hardware, MaxCUWidth and MaxCUHeight are set to 128. Therefore, the largest CU size is 128x128. MinQTSize is set to 8. Therefore, an 8x8 CU can only be further split with BT. MaxBtSize and MaxTtSize are set to 32. MaxMttDepth is set to 2. Therefore, multi-type tree split is not applied to CU sizes larger than 32x32. The maximum depth of multi-type tree split is 2, i.e., multi-type tree split can be applied at most twice.

The number of possible partitions in a 64x64 CU with these parameter values are shown in Table 3.1. Let X and Y represent one of the four possible multi-tree type partitions shown in Figure 3.1, then the partition type X_Y in Table 3.1 represents the case where first X type split then Y type split are applied after QT split. For example, BH_BH partition type represents the case where first binary horizontal split is applied after QT split, then binary horizontal split is applied to the 2 new partitions resulting in 4 partitions.

Table 3.1 Number of possible partitions and unique motion vectors in a 64x64 CU

Block Size	Partition Type	Total Partitions	Unique MVs	Block Size	Partition Type	Total Partitions	Unique MVs
64x64	No Partition	1	1	32x32	TH_TH	20	20
	Q	4	4		TV_BV	24	16
32x32	Q	16	16		TV_TV	20	20
	BH	8	8	16x16	Q	64	64
	BV	8	8		BH	32	32
	TH	12	4		BV	32	32
	TV	12	4		TH	48	16
	BH_BH	16	16		TV	48	16
	BH_TH	24	24		BH_BH	64	64
	BH_TV	24	8		BH_TV	96	32
	BV_BV	16	16		BV_BV	64	64
	BV_TH	24	8		BV_TH	96	32
	BV_TV	24	24	8x8	BH	128	128
	TH_BH	24	16		BV	128	128
Total						1077	821

The number of unique MVs is less than the number of partitions for some split types. For example, the top and bottom partitions of TH split are the same as top and bottom partitions of BH_BH split. Therefore, there is no need to calculate MVs for top and bottom partitions of TH split.

Redundant partitions, which are not allowed in VVC, are not shown in Table 3.1. For example, BH_BV split achieves the same partitions with QT split. Therefore, it is not allowed in VVC. In addition, some partitions are not allowed since they result in a partition size with height or width smaller than the minimum allowed CU size. These partitions are also not shown in Table 3.1. For example, when a 16x16 block is split with

ternary tree, its further split with ternary tree will result in a partition size of 8x2 or smaller. This is smaller than the minimum allowed CU size. Therefore, this is not allowed.

The proposed VVC ME hardware is shown in Figure 3.4. It consists of on-chip memory to store search window pixels and next block of current frame, a systolic array of processing elements (PEs) to store current and reference block pixels and calculate their absolute differences, an SAD adder tree to calculate SADs for all the supported CU sizes, a comparator unit to determine the minimum SAD and its corresponding MV for each CU size, and a control unit to perform control operations.

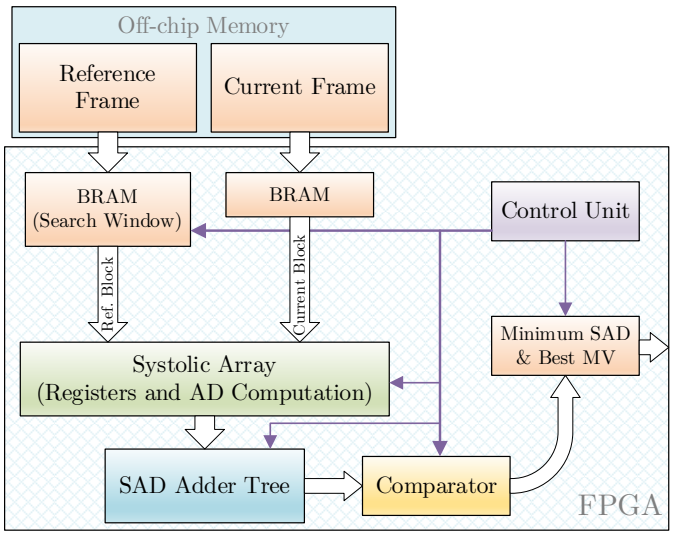


Figure 3.4 Proposed VVC ME hardware

To achieve real-time performance with small hardware area, the proposed VVC ME hardware divides 128x128 CTU into four 64x64 CUs. It uses a 64x64 systolic PE array and 64x64 SAD adder tree to determine the best 821 unique MVs for each of these 64x64 CUs sequentially. First, the best 821 unique MVs for the first 64x64 CU are determined. Then, the remaining three 64x64 CUs are processed one by one. The proposed hardware uses a novel memory-based SAD adder tree to determine the best MV for 128x128 CU. The best MV for 128x128 CU is determined together with the best MVs of last 64x64 CU.

3.2.1 Memory and Systolic PE Array

Xilinx FPGAs have fast dedicated on-chip memories called Block RAMs (BRAMs). In the proposed hardware, the current 64x64 CU and its corresponding search window are read from off-chip memories and stored in the on-chip BRAMs.

The proposed hardware has a 64x64 systolic PE array as shown in Figure 3.5. The systolic array also contains 64 registers to store an additional column of the search window. As shown in Figure 3.6, a PE consists of two registers which store a current block pixel and a reference block pixel, an absolute difference (AD) hardware, and an output register. AD hardware subtracts the reference pixel from the current pixel. If the subtraction result is negative, i.e., its sign bit is 1, it takes its 2's compliment to calculate the absolute difference.

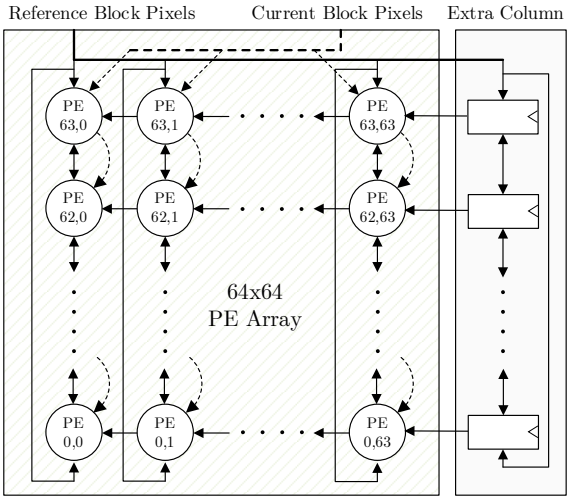


Figure 3.5 Systolic processing element (PE) array and registers.

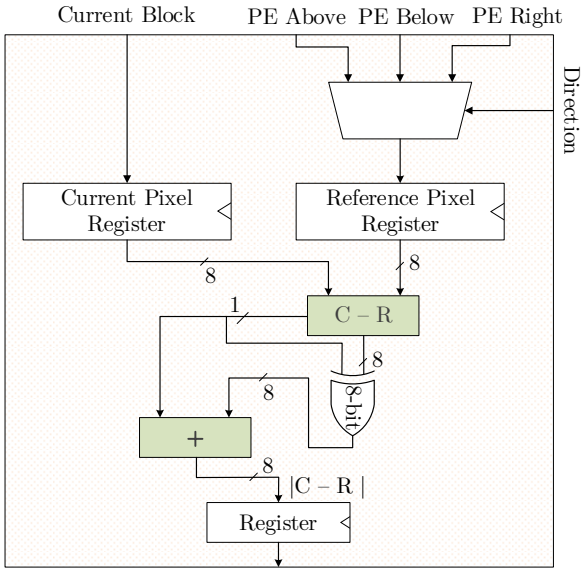


Figure 3.6 Processing Element (PE)

The systolic array receives a new row of 64 reference block pixels from BRAMs in every clock cycle. It takes 64 clock cycles to fill the systolic array with the 64x64 reference block of the first search location. At the same time, the 64x64 current block

pixels are also received from BRAMs and stored in the systolic array row by row. After that systolic array calculates 64x64 absolute differences in one clock cycle and sends them to SAD adder tree which calculates the SADs for all the partitions of 64x64 CU.

The systolic array stores the same 64x64 current block until all the search locations in the search window are searched for that current block. It can search a new search location in the search window in every clock cycle, i.e., it can process 64x64 reference block of each search location in one clock cycle.

The proposed hardware uses vertical snake scan order as shown in Figure 3.7 (a). The search starts from the top-left corner of the search window and moves downward until all the search locations in the first column are searched. Then, the search locations in the second column are searched in the upward direction. Then, the search locations in the third column are searched in the downward direction. This continues until all the search locations in the search window are searched for the current block.

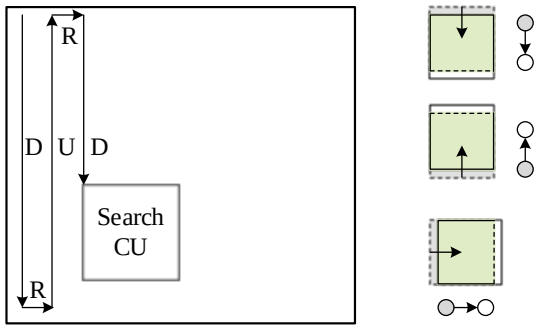


Figure 3.7 (a) Vertical snake scan order (b) Data re-use in downward, upward, and right directions.

To achieve high data reuse, each PE can shift its reference pixel up, down, or left. After a search location in a column, which is searched in the downward direction, is searched, all the PEs shift their reference pixels up, and a new row of 64 reference block pixels is read from search window memory and stored in the last row of systolic array as shown in Figure 3.7 (b). This continues until all the search locations in that column are searched.

In Figure 3.7 (b), green area represents the reused reference block pixels in the systolic array, white area represents the new row of 64 reference block pixels, and grey area represents the discarded row of 64 reference block pixels in the previous reference block.

After a search location in a column, which is searched in the upward direction, is searched, all the PEs shift their reference pixels down, and a new row of 64 reference block pixels is read from search window memory and stored in the first row of systolic array as shown in Figure 3.7 (b). This continues until all the search locations in that column are searched.

After all the search locations in a column are searched, all the PEs shift their reference pixels left, and a new column of 64 reference block pixels should be stored in the last column of systolic array. Since row aligned BRAMs are used in the proposed hardware, it would take 64 clock cycles to read a new column of 64 reference block pixels from BRAMs.

Therefore, an extra column of 64 registers is used in the systolic array. In every clock cycle, instead of 64, a new row of 65 reference block pixels is read from search window memory and stored in the systolic array. Therefore, after all the search locations in a column are searched, all the PEs shift their reference pixels left, and the PEs in the last column of systolic array receive their new reference pixels from the extra column of 64 registers. This takes only one clock cycle.

In the proposed hardware, BRAMs are configured as true dual port memories. After the current 64x64 CU is stored in the systolic array, the next 64x64 CU of the current frame is read into the BRAMs from off-chip memory. Similarly, the search window BRAMs are also updated dynamically with the search window of the next 64x64 CU of the current frame from off-chip memory.

3.2.2 SAD Adder tree

In HEVC, the maximum CTU size is 64x64, and 593 unique MVs should be calculated for a 64x64 CU [36]. In VVC, the maximum CTU size is 128x128, and 821 unique MVs should be calculated for a 64x64 CU. In addition, in VVC, there are more complex asymmetric partitions which are not used in HEVC. Therefore, SAD adder tree in VVC ME hardware is more complex than SAD adder tree in HEVC ME hardware.

In the proposed hardware, the SAD adder tree calculates the SADs of all the 821 unique partitions of a 64x64 CU by reusing the SADs of smaller partitions to calculate the SADs of larger partitions.

After the SAD adder tree receives 64x64 ADs for the first search location from the systolic array, it receives and processes 64x64 ADs of a new search location in every

clock cycle. For each 64x64 ADs, the corresponding 256 4x4 SADs are calculated in four clock cycles. One 4x4 SAD calculation including the AD calculation in PEs is shown in Figure 3.8. The red dotted lines in the figure indicate the pipeline registers.

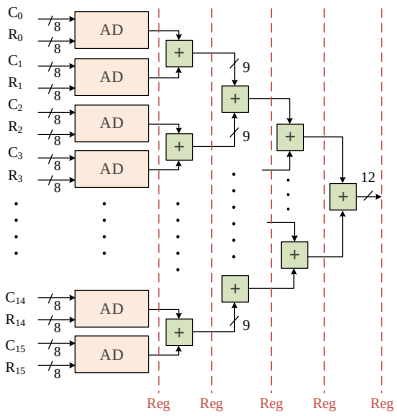


Figure 3.8 4x4 SAD calculation.

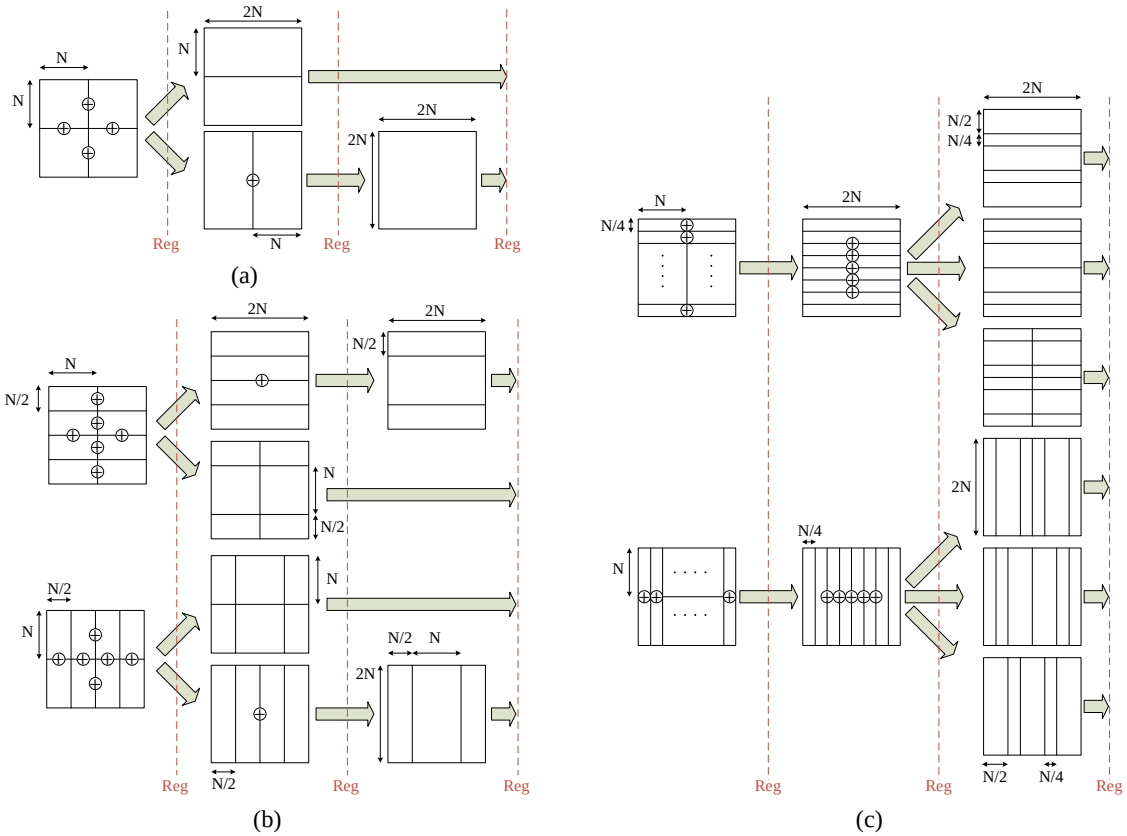


Figure 3.9 SAD adder tree (a) SADs of BH, BV, Q partitions N = 4, 8, 16, 32 (b) SADs of BH_BH, BV_TH, BH_TV, BV_BV, TH, TV partitions N = 8, 16 (c) SADs of TH_TH, TH_BH, BH_TH, BV_TV, TV_BV, TV_TV partitions N = 16.

These 4x4 SADs are then used to calculate SADs of larger partitions in a hierarchical manner. For example, 4x4 SADs are used to calculate SADs of binary partitions (BV,

BH) of 8x8 CUs. Then, the SADs of BV partitions of 8x8 CUs are used to calculate 64 SADs of 8x8 CUs. SADs of binary and quad partitions of 16x16, 32x32, and 64x64 CUs are calculated similarly as shown in Figure 3.9 (a).

Similarly, the SADs of BV and BH partitions of 8x8 CUs are used to calculate SADs of BH_BH, BV_BV, BV_TH and BH_TV partitions of 16x16 CUs. Then, the SADs of BH_BH and BV_BV partitions are used to calculate SADs of TH and TV partitions of 16x16 CUs. SADs of the same shaped partitions of 32x32 CUs are calculated similarly using BV and BH partitions of 16x16 CUs as shown in Figure 3.9 (b).

SADs of TH_TH, TV_TV, TH_BH, TV_BV, BH_TH and BV_TV partitions of 32x32 CUs are calculated using BH_BH and BV_BV partitions of 16x16 CUs as shown in Figure 3.9 (c).

The proposed hardware calculates the SADs of all the 821 unique partitions of a 64x64 CU for the first search location in the search window in 13 clock cycles and sends them to the comparator. After that, 821 new SADs are calculated in every clock cycle and sent to the comparator.

To achieve real-time performance with small hardware area, the proposed hardware divides 128x128 CU into four 64x64 CUs, processes them one by one and calculates SAD of 128x128 CU using the novel memory-based accumulator hardware shown in Figure 3.10.

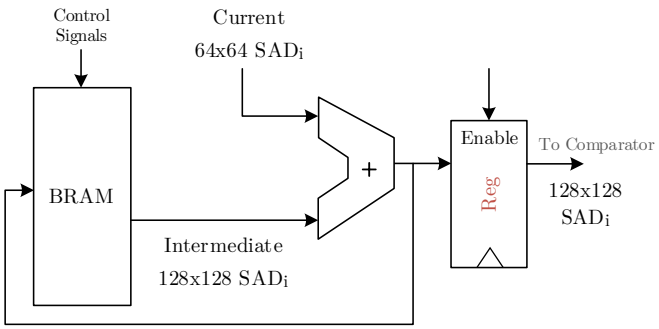


Figure 3.10 128x128 SAD calculation.

The top left 64x64 CU is processed first. For every *i*th search location in the search window, the SAD calculated for this 64x64 CU is sent to both the comparator and the memory-based accumulator where it is added to the content of *i*th location of BRAM and the result is written back to *i*th location of BRAM. The contents of the BRAM are initially set to 0. Therefore, the SADs of the top left 64x64 CU are stored in the BRAM.

Then, the top right 64x64 CU is processed. Therefore, the *i*th SAD of the top right 64x64 CU is added to the *i*th SAD of the top left 64x64 CU, and the result is written back to *i*th location of BRAM. Then, the bottom left 64x64 CU is processed similarly. Finally, the bottom right 64x64 CU is processed similarly.

When the first SAD of the bottom right 64x64 CU is added to the content of the first location of BRAM, the adder output is the first SAD of the 128x128 CU. Therefore, the output register in Figure 3.10 is enabled, and the first SAD of the 128x128 CU is sent to the comparator. After that, a new SAD of the 128x128 CU is calculated in every clock cycle and sent to the comparator. When the last SAD of the bottom right 64x64 CU is calculated, the last SAD of the 128x128 CU is also calculated after one clock cycle and sent to the comparator.

3.2.3 Comparator

The comparator unit determines the minimum SAD and its corresponding best MV for each CU size. It consists of one comparator for each of the 821 unique partitions of 64x64 CU and one additional comparator for the 128x128 CU. The sizes of these comparators vary from 13-bits for the smallest CU to 22-bits for the 128x128 CU. The latency of the comparator unit is one clock cycle. In every clock cycle, it compares all the SADs it receives from the SAD adder tree with the previous minimum SADs of the corresponding partitions and determines the minimum SAD and its corresponding best MV for each partition.

3.3 Implementation Results

The proposed VVC ME hardware is implemented using Verilog HDL. The Verilog RTL codes are implemented to a Xilinx Virtex 7 FPGA with speed grade 3 using Xilinx Vivado 2017.4 with default synthesis and performance_explore implementation strategies. The FPGA implementation is verified with post-implementation timing simulations.

The proposed hardware has 14 stages pipeline from AD calculation to comparator output. The latency for processing a 128x128 CTU can be calculated as (64 + 14 + Search Locations) x 4. The systolic array is filled in 64 clock cycles. It takes 14 clock cycles to calculate the SADs of all the CUs for the first search location in the search window and

compare them. After that, all the CUs for a new search location are processed in every clock cycle. The multiplication by 4 is necessary since a 64x64 SAD adder tree is used and a 128x128 CTU has four 64x64 CUs.

We implemented and verified the proposed VVC ME hardware in two different configurations for three different search ranges. One configuration supports 128x128 largest CTU size using a 64x64 systolic array and 64x64 SAD adder tree. The other configuration supports 64x64 largest CTU size using a 32x32 systolic array and 32x32 SAD adder tree. In each configuration, the size of the search range is set to the largest CTU size, 75% of the largest CTU size, and half of the largest CTU size.

The search range is centered around the top left pixel of current CTU. A search range of 128x128 means that the first pixel of first reference CTU is located at position (-64, -64) left of the first pixel of current CTU in the search window. Similarly, the first pixel of last reference CTU is located at position (+64, +64) right of the first pixel of current CTU in the search window.

Performance of the proposed VVC ME hardware for different configurations are shown in Table 3.2. The clock frequency (MHz), the number of clock cycles required to process a current CTU, and the throughput in frames per second (fps) for three different video resolutions (full HD, 2K, 4K) are shown in the table. The throughput in fps is calculated as shown below.

$$fps = \frac{1}{CTU\ latency \times CTUs\ per\ frame \times Clock\ period} \tag{3.2}$$

Table 3.2 Performance of the proposed VVC ME hardware for different configurations

CTU Size	128x128			64x64		
Search Locations	128x128	96x96	64x64	64x64	48x48	32x32
Frequency (MHz)	253	253	253	306	306	306
CTU Latency	65,848	37,176	16,696	16,560	9,392	4,272
FPS at 1080p	30	53	120	36	64	141
FPS at 2k	28	50	112	34	60	132
FPS at 4k	7	13	30	9	16	35

For 128x128 largest CTU size with 128x128 search range, 30 fps throughput is achieved for full HD video resolution. If the search range is reduced to 64x64, 30 fps throughput is achieved for 4K video resolution. For 64x64 largest CTU size with 32x32 search range, 35 fps throughput is achieved for 4K video resolution.

The FPGA resource usages of the proposed VVC ME hardware for 128x128 largest CTU size configuration with 128x128 search range and for 64x64 largest CTU size

configuration with 64x64 search range are shown in Table 3.3 and Table 3.4, respectively. The resource usage of 64x64 largest CTU size configuration is almost 4 times less than the resource usage of 128x128 largest CTU size configuration.

The systolic array uses the most FPGA resources. It uses 54% of the total flip-flops and 38% of the total LUTs used by the 128x128 largest CTU size configuration. The current pixel registers, reference pixel registers, and output registers in the systolic array justify the amount of flip-flop usage.

The SAD adder tree uses the second most FPGA resources. It uses 31% of the total flip-flops and 28% of the total LUTs used by the 128x128 largest CTU size configuration. Since the comparator unit uses registers to store the minimum SADs and corresponding best MVs, its flip-flop usage is higher than its LUT usage.

Table 3.3 Resource usage for 128x128 CTU size

Module	LUTs	Flip-Flops	BRAM
Systolic Array	56,321	98,816	–
SAD Adder Tree	40,970	57,312	4
Control Unit	36,582	2,806	–
Comparator	11,308	21,343	–
Memory	425	2,050	16
Total	145,606	182,327	20

Table 3.4 Resource usage for 64x64 CTU size

Module	LUTs	Flip-Flops	BRAM
Systolic Array	16,390	24,832	–
SAD Adder Tree	11,022	14,474	1
Control Unit	9,954	1,348	–
Comparator	2,818	4,948	–
Memory	210	1,026	8
Total	40,394	46,628	9

3.3.1 Comparison With HEVC ME Hardware

The proposed VVC ME hardware is the first VVC ME hardware in the literature. The proposed VVC ME hardware implementation is compared with the HEVC ME hardware implementations in the literature in Table 3.5. Although VVC ME has larger maximum CTU size and it is more computationally complex than HEVC ME, the proposed VVC ME hardware has smaller area and higher throughput than some of these HEVC ME hardware.

The HEVC ME hardware proposed in [34] and [35] use full search ME algorithm. The hardware proposed in [34] does not support the asymmetric partitions in HEVC ME.

These HEVC ME hardware use more LUTs and have lower throughput than our VVC ME hardware.

Table 3.5 Comparison with HEVC ME Hardware

	ICIP'14 [34]	IET'17 [35]	AICSP'18 [38]	JRTIP'19 [39]	JRTIP'21 [40]	This Work
Encoding Standard	HEVC	HEVC	HEVC	HEVC	HEVC	VVC
FPGA	Virtex 5	Virtex 5	Virtex 7	Virtex 7	Virtex 7	Virtex 7
CTU Size	64x64	64x64	64x64	64x64	32x32	128x128
Search Range	64x64	64x64	144x144	64x64	64x64	64x64
Working Frequency (MHz)	125	84.96	198.73	247	162	253
Throughput	4K @ 13 fps	4K @ 9 fps	FHD @ 30fps	4K @ 30 fps	8K @ 78 fps	4K @ 30 fps
LUTs	209,434	153,314	49,258	188,664	485,760	145,606
Flip-Flops	199,066	36,368	13,351	144,302	607,200	182,327

A sequential and a parallel HEVC ME hardware implementing diamond search algorithm are proposed in [38]. Since the parallel hardware has higher performance than the sequential hardware, we compare our VVC ME hardware with the parallel HEVC ME hardware. The HEVC ME hardware has smaller area and lower throughput than our VVC ME hardware.

The HEVC ME hardware proposed in [39] use full search ME algorithm. It uses more LUTs and has the same throughput as our VVC ME hardware. The HEVC ME hardware proposed in [40] use a fast hybrid pattern search algorithm. It has higher throughput and much larger area than our VVC ME hardware.

Chapter 4

AN EFFICIENT APPROXIMATE SAD HARDWARE FOR FPGAS

Approximate computing is a promising approach for designing smaller area and lower power consuming hardware than accurate hardware at the expense of quality loss [26,51]. Therefore, it can be used for error-tolerant applications. Some video processing and coding applications can tolerate inaccurate results due to limitations of human visual perception.

Sum of absolute differences (SAD) operation is a widely used metric for block matching in several error tolerant applications such as stereo vision and video coding [27, 36, 52]. SAD between two blocks, A and B, of size W x H is calculated as shown in equation (4.1), where $A(i,j)$ and $B(i,j)$ are values of the pixels in ith row and jth column of A and B, respectively.

$$SAD = \sum_{i=0}^{W-1} \sum_{j=0}^{H-1} |A(i,j) - B(i,j)| \tag{4.1}$$

SAD is the most time and power consuming operation in several error tolerant applications. For example, SAD operation is used for motion estimation in video coding, and it can consume up to 80% of the total energy consumption of a video encoder hardware [52]. Therefore, approximate computing can be used for designing efficient SAD hardware.

Several ASIC or FPGA based accurate SAD hardware implementations are proposed in the literature [27,41,53]. Several approximate SAD hardware implementations are also proposed in the literature [26,36,52,55]. None of the approximate SAD hardware has FPGA specific optimizations.

A novel approximate absolute difference hardware (NAAD) is proposed in [26]. The incrementor used for 2's complement operation in AD hardware is removed. A low-error approximate adder (LEA) and an approximate AD hardware based on LEA are proposed in [36]. An approximate SAD hardware using lower-part OR (LOA) approximate adder

is proposed in [52]. A power-efficient approximate SAD (aSAD) hardware approximating most significant bits with a single bit is proposed in [55].

In this chapter, we propose an efficient approximate SAD hardware for FPGAs. The proposed approximate SAD hardware utilizes the unused LUT inputs to reduce area and power consumption while providing an almost accurate result. The proposed approximate SAD hardware has smaller maximum and average error than the approximate SAD hardware in the literature. It uses up to 20% less LUTs than the smallest approximate SAD hardware in the literature. It consumes up to 38% less power than the lowest power consuming approximate SAD hardware in the literature.

4.1 Background

4.1.1 SAD Hardware

Block diagram of a generic SAD hardware is shown in Figure 4.1. Generally, SAD is calculated in two stages. In the first stage, absolute differences of inputs are calculated. In the second stage, these absolute difference values are added using an adder tree. Parallelism can be used in both stages.

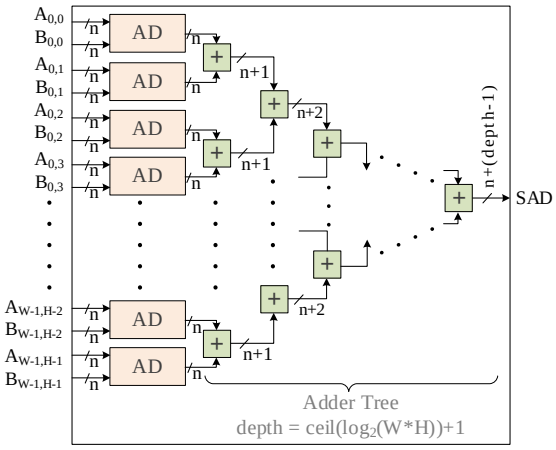
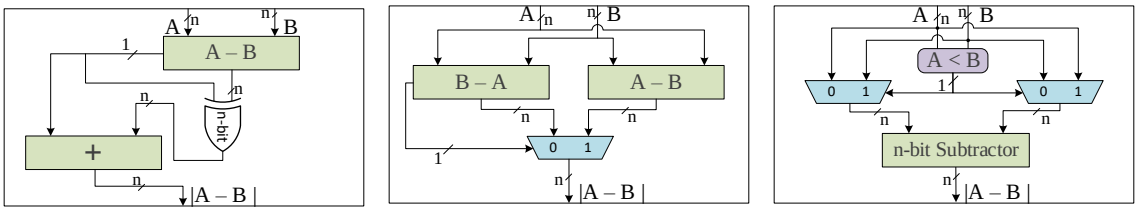


Figure 4.1 Generic SAD hardware

Three conventional accurate AD hardware are shown in Figure 4.2. Accurate AD 1 hardware, shown in Figure 4.2 (a), first subtracts the two inputs. If $A < B$, then 2's complement of the subtraction result should be calculated to obtain the AD result. Sign-bit of the subtraction result is XOR'ed with each bit of the subtraction result to perform selective inverse operation. Finally, the same sign-bit is added to the output of XOR gates

to obtain the AD result. The maximum delay of this AD hardware is $2t_{Sub} + t_{XOR}$. It uses $2n$ LUTs in a Xilinx FPGA, where n is bit length of the inputs.



(a) Accurate AD 1 (b) Accurate AD 2 (c) Accurate AD 3

Figure 4.2 Accurate absolute difference hardware

Accurate AD 2 hardware, shown in Figure 4.2 (b), trades off area for speed by using two subtractors in parallel. These two subtractors perform $(A - B)$ and $(B - A)$, respectively. The MSB of one of these subtraction results is then used to select the correct AD result. The maximum delay of this AD hardware is $t_{Sub} + t_{Mux}$. It uses $3n$ LUTs in a Xilinx FPGA.

Accurate AD 3 hardware, shown in Figure 4.2 (c), compares the two inputs for selective subtraction [27]. If $(B > A)$, then $(B - A)$ is performed, otherwise $(A - B)$ is performed. The maximum delay of this AD hardware is $t_{Comp} + t_{Sub}$. It uses $1.5n$ LUTs in a Xilinx FPGA. Since 6-input LUTs are used in Xilinx FPGAs, selection and inversion operations can be implemented in one LUT [54].

4.1.2 Xilinx Virtex FPGA

Configurable logic blocks (CLB) are the main logic resource in a Xilinx Virtex FPGA. Each CLB has two slices. Each slice in Xilinx Virtex 5/6/7 FPGAs has four copies of the hardware shown in Figure 4.3 with carry chain cascaded in series [56]. Therefore, each slice contains four 6-input LUTs, a 4-bit carry chain, and 8 output registers along with routing resources. A 6-input LUT can be used to implement one 6-input combinational logic or two 5-input combinational logics.

The efficiency of an FPGA-based hardware depends on how effectively it utilizes the logic resources available in the FPGA. This is very important for FPGA implementations that may use carry chains in FPGA. SAD hardware uses subtraction and addition operations. Therefore, it is important to understand how these arithmetic operations are implemented in Xilinx FPGA and how the unused resources can be utilized to obtain an efficient implementation.

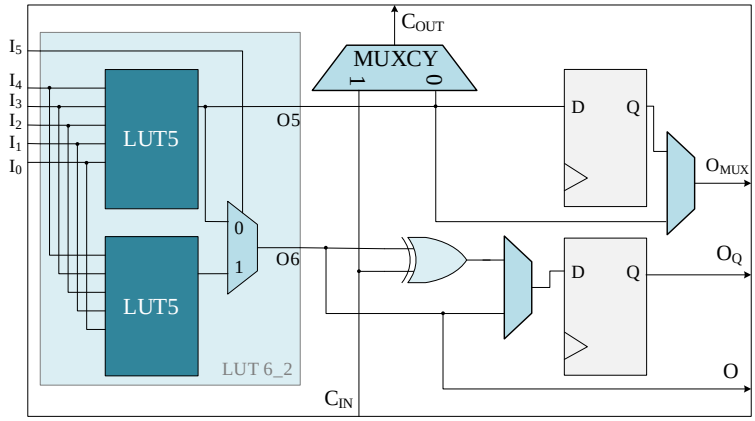


Figure 4.3 Simplified architecture of a slice in Xilinx Virtex 5/6/7 FPGAs

Generally, 1-bit addition operation is performed as shown in equation (4.2) and equation (4.3) where A and B are inputs, S is sum, and C_{IN} and C_{OUT} are carry-in and carry-out, respectively.

$$S = (A \wedge B) \wedge C_{IN} \tag{4.2}$$

$$C_{OUT} = AB \mid (A \wedge B)C_{IN} \tag{4.3}$$

However, Xilinx synthesis tools simplify equation (4.3) as equation (4.4) for a more efficient FPGA implementation.

$$C_{OUT} = \overline{(A \wedge B)}B \mid (A \wedge B)C_{IN} \tag{4.4}$$

This simplification allows the reuse of $A \wedge B$ term, hence only one output of a LUT is used. However, since only 4-bit carry chain is available in a slice, an n-bit adder uses n LUTs such that 4 inputs and one output of each LUT are not used. Therefore, additional logic can be implemented together with an adder using the same 6-input LUTs without affecting critical-path delay of the adder.

The carry chain in Xilinx Virtex FPGAs is also flexible. The carry-in for the first bit of carry chain can either be connected to carry-out of the previous slice or it can be connected to one of the user-defined inputs.

4.2 Proposed Approximate SAD Hardware

We propose to utilize the unused LUT inputs and output to implement an adder/subtractor including the complement operations shown in Figure 4. The proposed hardware computes 1's complement of both inputs without using additional resources and without additional delay. It also computes 2's complement of one of the inputs by initializing the carry chain with the selective inverse signal. The proposed hardware implements the following expression.

$$S = ((-1)^{X_n}X) + ((-1)^{Y_n}Y - Y_n) \tag{4.5}$$

X_n and Y_n can either be 0 or 1. When X_n is 1, 2's complement of X is computed. When Y_n is 1, 1's complement of Y is computed.

```
X_1's_complement = {n{X[n]}} ^ {X[n-1:0]}
Y_1's_complement = {n{Y[n]}} ^ {Y[n-1:0]}

S = X_1's_complement + Y_1's_complement + X[n]
```

Figure 4.4 Proposed adder/subtractor including complement operations

Implementation of the proposed n-bit adder/subtractor including complement operations in Xilinx Virtex 5/6/7 FPGAs is shown in Figure 4.5.

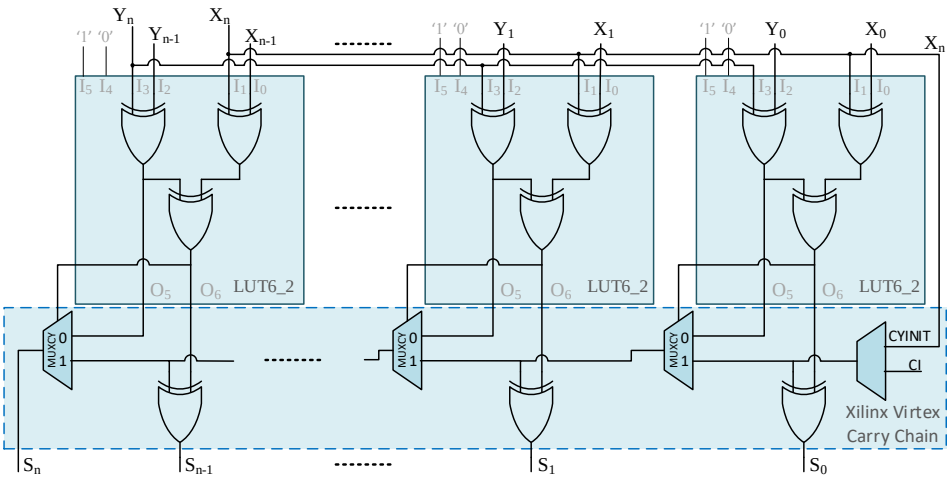


Figure 4.5 Implementation of the proposed n-bit adder/subtractor including complement operations

To perform selective bitwise inversion of X and Y, their n-bits are XOR'ed with X_n and Y_n, respectively. This is done in the first two XOR gates. When X_n is 1, X is inverted.

When Y_n is 1, Y is inverted. The third XOR gate computes first term of the sum expression shown in equation (4.2) and its output is connected to the carry chain for carry propagation and sum calculation. Finally, X_n is also connected to CYINIT (initialize carry) input of the carry chain to compute 2's complement of X, i.e. when X_n is 1, 1 is added to the result.

The proposed approximate 2×1 SAD hardware is shown in Figure 4.6. This hardware is designed by merging the XOR gates and incrementor in the Accurate AD 1 hardware shown in Figure 4.2 (a) to the first stage of adder tree, and by using the proposed adder/subtractor shown in Figure 4.5 in the first stage of adder tree.

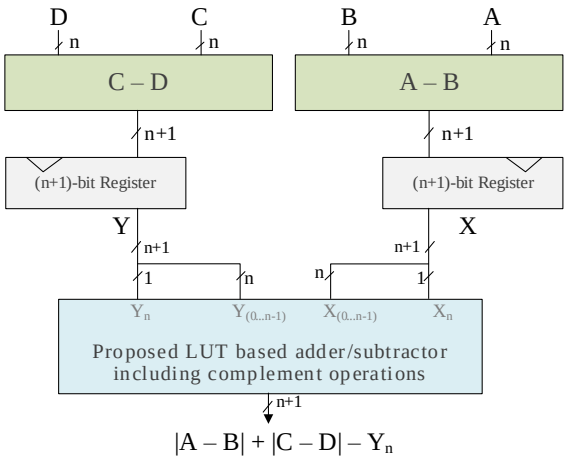


Figure 4.6 Proposed approximate 2×1 SAD hardware

In the first stage of the proposed SAD hardware only a subtractor is used. Hence, it will be mapped to only n LUTs. The subtraction result and its sign bit are passed to the next stage. In the second stage of the proposed SAD hardware the proposed adder/subtractor is used. The sign bits are connected to X_n and Y_n inputs of the proposed adder/subtractor.

The proposed adder/subtractor computes the absolute values of the two subtraction results and adds them. However, when Y_n is 1, the absolute value result of Y will be 1 less than the accurate absolute value of Y. Therefore, maximum error of the proposed approximate 2×1 SAD hardware is 1.

Several approximate 2×1 SAD hardware can be used to build larger approximate SAD hardware. In the proposed approximate SAD hardware, half of the AD hardware may have an error of 1. Therefore, maximum error of the proposed approximate SAD hardware with N AD hardware is N/2.

4.3 Implementation Results

Quality, speed, area, and power consumption of the proposed approximate SAD hardware are compared with that of several approximate SAD hardware proposed in the literature; NAAD [26], LEA based approximate SAD [36], LOA based approximate SAD [52], aSAD [55]. They are also compared with that of approximate SAD hardware designed by replacing the accurate adder in AD operation with the LUT-based fast approximate adder unit (FAU) proposed in [57]. 4-bit approximation is used for NAAD SAD hardware and for LEA, LOA and FAU based SAD hardware.

Quality comparison for 4×4 approximate SAD is shown in Table 4.1. The maximum and average error values are determined for 8-bit inputs and by applying all possible input values to an AD hardware such that the same input values are applied to all AD hardware. The results show that the proposed approximate SAD hardware has smaller maximum and average error than the other approximate SAD hardware.

Table 4.1 Quality comparison for 4×4 approximate SAD

	Maximum Error	Average Error
Proposed	8	3.98
LEA	160	27.00
FAU	256	35.81
LOA	128	53.45
NAAD	256	63.50
aSAD	3584	1021.56

To compare speed, area, and power consumption of SAD hardware, three accurate SAD hardware shown in Figure 4.2, the proposed approximate SAD hardware and five approximate SAD hardware in the literature are implemented using Verilog HDL. A common adder tree hardware, with approximations applied as and when required, is used for all SAD hardware. All implementations are highly pipelined, i.e. a pipeline register is used after each addition stage. Registers are also placed at all the inputs and outputs. Experiments are performed for 4×4, 8×8, 16×16, 24×24, and 32×32 SAD block sizes using 8-bit and 16-bit inputs.

All Verilog RTL codes are synthesized and implemented to Xilinx XC7VX485T-3FFG1761 FPGA using Vivado 2020.1. *Vivado Synthesis Defaults* and *Performance*

Explore synthesis and implementation strategies are used, respectively. FPGA implementations are verified with post-implementation timing simulations.

For power consumption estimation, switching activity interchange format (SAIF) files are generated using post-implementation timing simulations at 100 MHz for all FPGA implementations. Power consumption of each FPGA implementation is estimated with Vivado 2020.1 using the corresponding SAIF file.

Area, power consumption, and maximum clock frequency results are shown in Figure 4.7 and Figure 4.8 for 8-bit and 16-bit inputs, respectively. LUT reductions achieved by the proposed approximate SAD hardware compared to the smallest approximate SAD hardware in the literature for each block size are shown in Table 4.2. Power reductions achieved by the proposed approximate SAD hardware compared to the lowest power consuming approximate SAD hardware in the literature for each block size are also shown in Table 4.2.

Table 4.2 Reductions achieved by proposed approximate SAD hardware

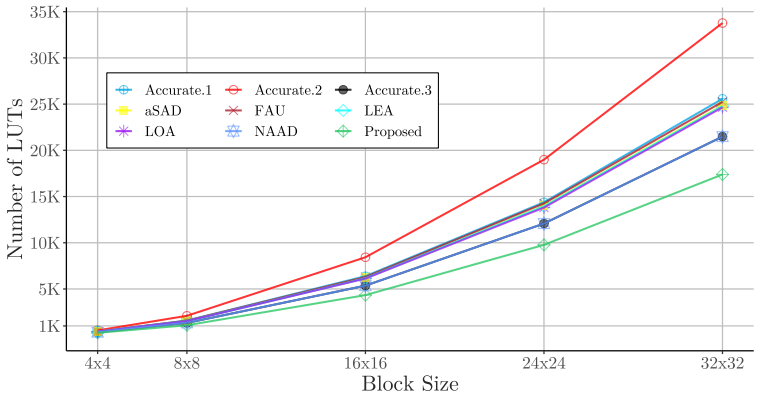
	8-bit		**16-bit**	
	LUT Reduction	**Power Reduction**	**LUT Reduction**	**Power Reduction**
4×4	19.81	0.00	20.16	20.00
8×8	19.26	11.11	19.68	20.51
16×16	19.11	16.30	19.56	32.43
24×24	19.08	24.40	19.53	34.55
32×32	19.06	27.08	15.40	38.26

The proposed approximate SAD hardware uses the smallest number of LUTs among all SAD hardware for all block sizes. It uses up to 20% less LUTs than the smallest approximate SAD hardware in the literature. However, it uses up to 3% more flip-flops for 8-bit inputs, and up to 1.5% more flip-flops for 16-bit inputs. As expected, Accurate 2 SAD hardware, which uses Accurate AD 2 hardware, has the largest area.

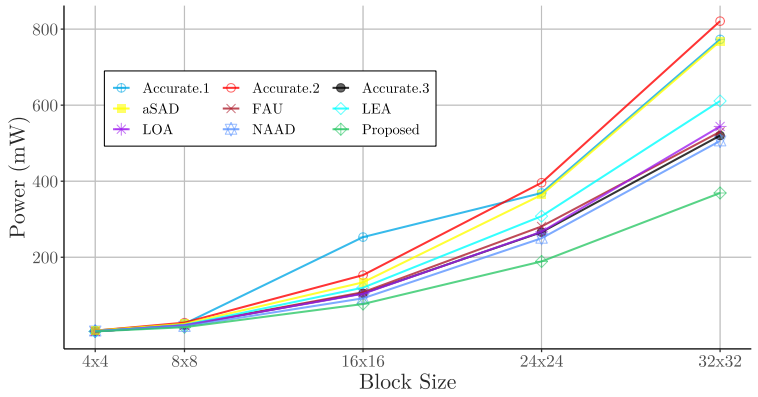
The proposed approximate SAD hardware also consumes the lowest power among all SAD hardware for all block sizes. It consumes up to 38% less power than the lowest power consuming approximate SAD hardware in the literature. The approximate SAD hardware in the literature consume less power than Accurate 1 and Accurate 2 SAD hardware. NAAD with 4-bit approximation is the most area and power-efficient among all approximate SAD hardware in the literature.

For 8-bit inputs, FAU, LEA and LOA based approximate SAD hardware are slightly faster than the proposed approximate SAD hardware. Average clock frequencies of the

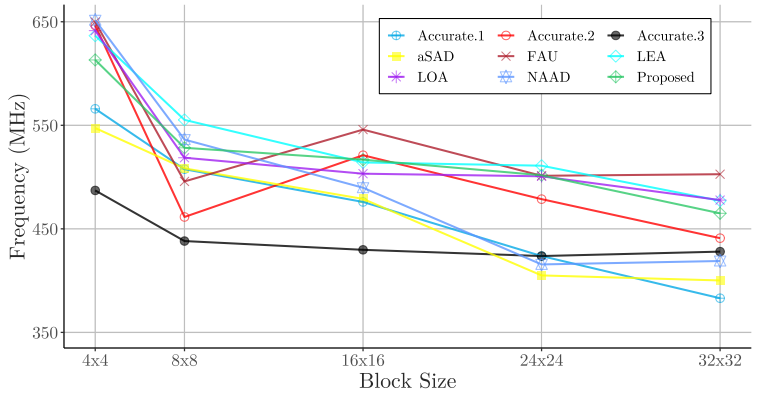
proposed, FAU, LEA, and LOA based approximate SAD hardware are 525 MHz, 539 MHz, 538 MHz, and 528 MHz, respectively. However, for 16-bit inputs, the proposed approximate SAD hardware is faster than all the other SAD hardware. The proposed approximate SAD hardware achieves an average clock frequency of 487 MHz followed by 484 MHz achieved by NAAD hardware.



(a) LUT Utilization

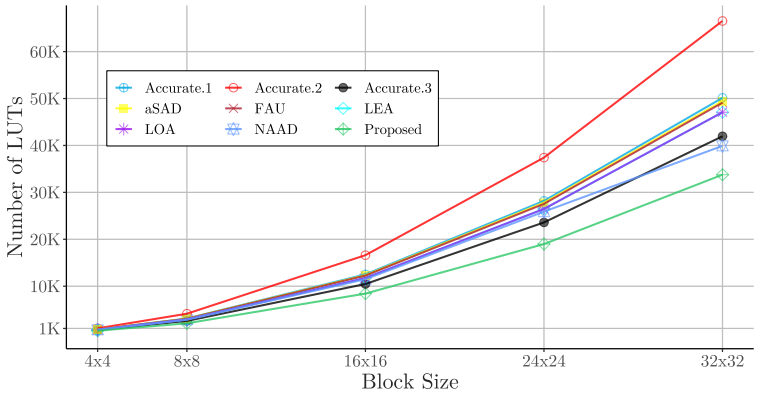


(b) Power Consumption

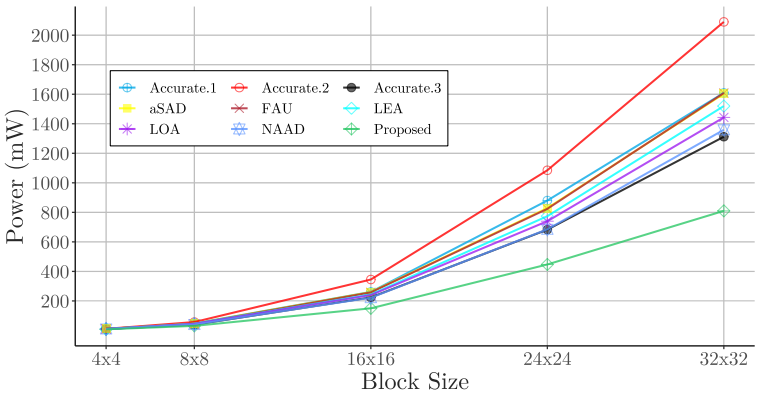


(c) Maximum Clock Frequency

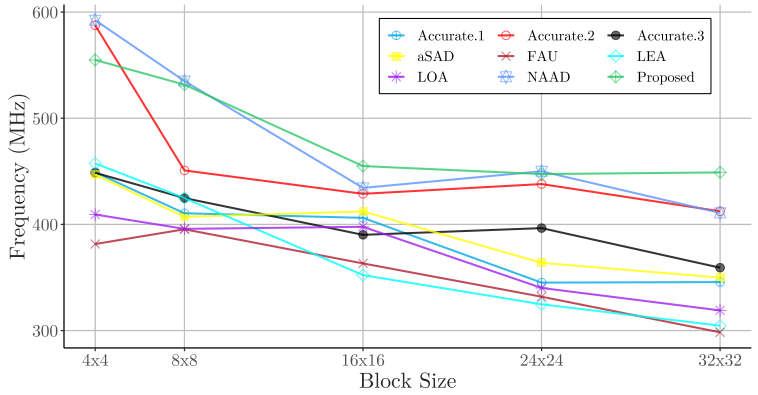
Figure 4.7 Implementation results for 8-bit inputs



(a) LUT Utilization



(b) Power Consumption



(c) Maximum Clock Frequency

Figure 4.8 Implementation results for 16-bit inputs

Chapter 5

LOW ERROR EFFICIENT APPROXIMATE ADDERS FOR FPGAS

Approximate computing trades off accuracy to improve the area, power, and speed of digital hardware. Many computationally intensive applications such as video encoding, video processing, and artificial intelligence are error resilient by nature due to the limitations of human visual perception or nonexistence of a golden answer for the given problem. Therefore, approximate computing can be used to improve the area, power, and speed of digital hardware implementations of these error tolerant applications.

A variety of approximate circuits, ranging from system level designs [28,42,58,59] to basic arithmetic circuits [17], 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 [26,60,61]. Therefore, many approximate adders have been proposed in the literature [16,36],[62]-[75]. All approximate adders exploit the fact that critical path in an adder is seldom used.

Approximate adders can be broadly classified into the following categories: segmented adders [63], which divide n-bit adder into several r-bit adders operating in parallel; speculative adders [62], which predict the carry using only the few previous bits; and approximate full-adder based adders [16,36],[65]-[68], which approximate the accurate full-adder at transistor or gate level. Segmented and speculative adders usually have higher speeds and larger areas than accurate adders [17]. Approximate full-adder based approximate n-bit adders use m-bit approximate adder in the least significant part (LSP) and $(n-m)$-bit accurate adder in the most significant part (MSP), as shown in Figure 5.1.

Most of the approximate adders in the literature have been designed for ASIC implementations. These approximate adders use gate or transistor level optimizations. Recent studies have shown that the approximate adders designed for ASIC

implementations either do not yield the same area, power, and speed improvements when implemented on FPGAs or fail to utilize FPGA resources efficiently to improve the output quality [71],[77].

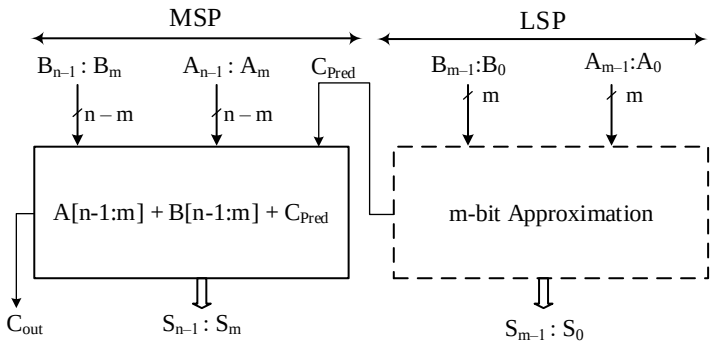


Figure 5.1 Architecture of approximate full-adder based n-bit approximate adders

This is mainly due to the difference in the way logic functions are implemented in ASICs and FPGAs. The basic element of an ASIC implementation is a logic gate, whereas FPGAs use lookup tables (LUTs) to implement logic functions. Therefore, ASIC based optimization techniques cannot be directly mapped to FPGAs.

FPGAs are widely used to implement error-tolerant applications using addition and multiplication operations. The efficiency of FPGA-based implementations of these applications can be improved through approximate computing. Only a few FPGA specific approximate adders have been proposed in the literature [70]-[74]. These approximate adders focus on improving either the efficiency or accuracy. Therefore, the design of low error efficient approximate adders for FPGAs is an important research topic.

In this chapter, we propose a methodology to reduce the error of approximate adders by efficiently utilizing FPGA resources, such as unused LUT inputs. We propose two approximate adders for FPGAs using our methodology based on the architecture shown in Figure 5.1.

We propose a low error and area efficient approximate adder (LEADx) for FPGAs. It has lower mean square error (MSE) than the approximate adders in the literature. It achieves better quality than the other approximate adders for video encoding application.

We also propose an area and power efficient approximate adder (APEx) for FPGAs. Although its MSE is higher than that of LEADx, it is lower than that of the approximate adders in the literature. It has the same area, lower MSE and less power consumption than the smallest and lowest power consuming approximate adder in the literature. It has

smaller area and lower power consumption than the other approximate adders in the literature.

We provide mathematical models to estimate the error rate (ER), MSE, and mean absolute error (MAE) of the proposed approximate adders. We compare the proposed approximate adders with the approximate adders in the literature.

5.1 Background

5.1.1 Related Works

Bit truncation in least significant bit positions is a well-known approximation technique. In truncate adder, the output of LSP is fixed to zero. Although, the truncate adder provides significant improvements in speed, area, and power consumption, it has high error rate and MSE [36],[65].

Lower-part-OR adder (LOA) is proposed in [16]. Its LSP consists of 2-input OR gates, whereas the MSP is accurate. A carry is sent to the MSP if it is generated at most significant bit position of the LSP. An approximate adder, OLOCA, is proposed in [66] by optimizing the LOA architecture. OLOCA uses only two OR gates in the LSP to compute the two most significant sum bits. Rest of the LSP is approximated to a fixed value. An approximate adder with near-normal error distribution (HOAANED) is proposed in [67]. HOAANED has similar architecture to OLOCA, however, it uses more resources to compute the two most significant sum bits of LSP. Therefore, HOAANED has better quality than OLOCA at the expense of slight increase in area.

Dutt et al. [68] proposed an approximate full adder based multibit adder (AFA). The sum of each bit of LSP is computed accurately whereas its respective carry out is equated to one of the inputs.

In recent years, a few approximate adders are proposed specifically for FPGAs. A LUT-based approximate adder (LBA) is proposed in [70]. The LSP and MSP, both perform accurate addition. A carry is passed to MSP only if it is generated at the most significant bit (MSB) of the LSP. If any other carry, that needs to be propagated to the MSP, is detected, then all bits of LSP are set to 1. LBA has high accuracy, but it does not provide performance improvement compared to the accurate adder synthesized by FPGA synthesis tool [71].

A methodology to design approximate adders (DeMAS) for FPGAs is presented in [71]. The methodology is based on an optimized truth table of approximate full-adder. Eight different variants of multibit approximate adder are presented using the optimized truth table. All these variants use same number of LUTs but differ in their error metrics.

Quaternary addition based approximate adder using the fast carry chains of FPGAs is presented in [72]. The accurate quaternary adder uses two carry inputs and generates two carry outputs. However, the authors in [72] proposed to use only one carry in the quaternary addition, hence generating an approximate result.

A single exact dual adder (SEDA) is proposed for FPGAs in [73]. The adder can either perform accurate addition of single n-bit input or approximate addition of two n-bit inputs. Carry of 2-bit addition is computed accurately, while the sum bits are equated to inverse of carry out.

High speed segmented approximate adders (xUAV) for FPGAs are proposed in [74]. Segmentation is done in 2, 3, or 5-bit groups for efficient mapping to LUTs. However, the proposed adders use more area and consume more power than accurate adder. These adders also have very large MAE and MSE as the size of adder is increased.

5.1.2 Length of carry

The key principle of approximate addition is to shorten the critical path of an adder by breaking the carry chain at one or multiple positions. This technique improves the speed of an adder at the expense of accuracy loss. In this section, we briefly explain the rationale for this technique.

The length of a carry signal in n-bit binary addition is defined as the number of bits it propagates before being killed or regenerated. For example, if a carry signal is generated at ith bit position and killed or regenerated at jth bit position ($j > i$), the length of that carry signal is defined as $j - i$ bits.

In n-bit binary addition, the outgoing carry signal at any bit position i is determined by the current and previous input bits. Bit position i is said to generate a carry if both the input bits at ith position are 1, propagate the incoming carry if both the input bits at ith position are different, and kill the incoming carry if both the input bits at ith position are 0.

In the worst case, a carry signal is generated in the least significant bit (LSB) and propagated to the most significant bit (MSB). In this case, the length of carry signal is

equal to the adder bit width. However, the worst case rarely happens, and the average length of a carry signal is usually much shorter than the adder bit width [62].

We implemented and simulated n-bit accurate adder using 10^7 independent random number pairs extracted from uniformly distributed sample space between 0 and $2^n - 1$. Based on these simulation results, probability of the length of a carry signal being equal to L bits is given in Table 5.1. As can be seen from this table, the length of a carry signal is rarely longer than 5 bits. The length of a carry signal is shorter than 5 bits with more than 90% probability.

Table 5.1 Probability of the length of a carry being equal to L bits

Adder Bit Width (n)	Probability (%)					
	L = 1	L = 2	L = 3	L = 4	L = 5	L = 6
16	53.09	25.01	11.72	5.49	2.54	1.17
32	51.59	24.97	12.10	5.86	2.84	1.37
64	50.78	25.00	12.31	6.05	2.98	1.46
128	50.38	24.99	12.41	6.16	3.05	1.51

Since the worst case of carry propagation (length of carry = n-bits) rarely happens, in most cases, the carry can be correctly predicted by considering only a few previous input bits.

5.1.3 Xilinx Virtex FPGA

The main logic resource in a Xilinx Virtex FPGA is configurable logic blocks (CLBs) [27]. Each CLB contains two slices. Simplified architecture of a slice in Xilinx Virtex 7 FPGA is shown in Figure 4.3. Each slice is composed of 4 such elements with carry-chain cascaded in series. Therefore, each slice has four 6-input LUTs. Each LUT can be used to implement two 5-input combinational logic functions or one 6-input combinational logic function. Furthermore, each slice also contains a 4-bit carry-chain and eight flip-flops.

An efficient FPGA-based implementation should be able to effectively utilize these resources. This is particularly important for implementing arithmetic functions that can utilize the fast carry-chains. Therefore, it is important to understand how the arithmetic operations are implemented on FPGAs. Particularly, we consider the mapping of a full adder to a Xilinx FPGA.

Typically, a full adder is implemented as shown in equation (5.1) and equation (5.2), where A and B represent the inputs, S is sum, and C_{IN} and C_{OUT} are carry-in and carry-out, respectively.

$$S = (A \oplus B) \oplus C_{IN} \tag{5.1}$$

$$C_{OUT} = (A \oplus B)C_{IN} + AB \tag{5.2}$$

However, when implementing a full adder on a Xilinx FPGA, the synthesis tool rewrites equation (5.2) as equation (5.3).

$$C_{OUT} = (A \oplus B)C_{IN} + \overline{(A \oplus B)}B \tag{5.3}$$

This simplification allows the reuse of A ⊕ B logic function for computing both S and C_{OUT} [78]. This term is used as input to XOR gate for sum computation and as select input of mux for selecting the appropriate signal for C_{OUT}. However, since only 4-bit carry chain is available in a slice, an n-bit adder uses n LUTs such that 4 inputs and one output of each LUT are not used. These unused resources can be utilized to implement additional logic with an adder without increasing area.

5.2 Proposed Design Methodology

The proposed design methodology uses the approximate full-adder based n-bit adder architecture shown in Figure 5.1. n-bit addition is divided into m-bit approximate adder in the LSP and (n−m)-bit accurate adder in the MSP. Breaking the carry chain at bit-position m generally introduces an error of 2^m in the final sum. The error rate and error magnitude can be reduced by predicting the carry-in to the MSP (C_{MSP}) more accurately and by modifying the logic function of LSP to compensate for the error.

The carry to the accurate part can be predicted using any k-bit input pairs from the approximate part such that $k \leq m$. Most of the existing approximate adders use $k = 1$.

As discussed in Section II, FPGA implementation of accurate adder uses only 2 inputs and 1 output of each 6-input LUT. We propose to utilize the remaining 4, available but unused, inputs of the first LUT of the MSP to predict C_{MSP}. Therefore, we propose to share the most significant 2 bits of both inputs of the LSP with the MSP for carry prediction.

Sharing more bits of LSP with MSP will increase the probability of correctly predicting C_{MSP} which will in turn reduce error rate. However, this will also increase the area and delay of the approximate adder.

To analyze the tradeoff between the accuracy and performance of an FPGA-based approximate adder with different values of k, we performed synthesis and simulation experiments on a Xilinx Virtex 7 FPGA.

The results for a 64-bit adder with 12-bits LSP using k bits to predict C_{MSP} are shown in Table 5.2. For $k > 2$, the error rate reduces slightly at the cost of increased area and delay. On the other hand, for $k < 2$, the delay improves marginally at the cost of significant increase in the error rate.

Table 5.2 Effects of Increasing the Number of Bits (k) for Carry Prediction in a 64-Bit Approximate Adder with 12-Bits LSP

k	LUT	Delay (ns)	ER (%)
0	64	1.57	50.04
1	64	1.59	24.98
2	64	1.62	12.51
3	65	1.85	6.26
4	65	1.90	3.10
5	65	2.03	1.55

Therefore, we propose using $k = 2$, as it provides good balance between accuracy and performance of approximate adders for FPGAs. In the proposed approximate adders, a carry is passed to the MSP if it is generated at bit position $m - 1$, or generated at bit position $m - 2$ and propagated at bit position $m - 1$. The C_{MSP} can be described by equation (5.4) where G_i and P_i are the generate and propagate signals of the ith bit position, respectively.

$$C_{MSP} = G_{m-1} + P_{m-1}G_{m-2} \tag{5.4}$$

The error in higher bit positions has more impact on the error magnitude of an approximate adder. As described in equation (5.4), the carry-in to MSP is predicted using two most significant bits of LSP. These 2 bits effectively implement a 3-output function $\{C_{MSP}S_{m-1}S_{m-2}\}$. An error occurs in the n-bit addition if a carry (C_{m-2}) is generated at bit position $i < (m - 2)$ and that carry should be propagated to MSP. In this case, the correct result should be $\{C_{MSP}S_{m-1}S_{m-2}\}$ = 100. However, without any error reduction mechanism the approximate result will be $\{C_{MSP}S_{m-1}S_{m-2}\}$ = 000.

To reduce the error magnitude, we propose a 2-bit approximate adder (AAd1) for computing S_{m-1} and S_{m-2}. The functionality of AAd1 is described by equation (5.5) and

equation (5.6). AAd1 is implemented using a single LUT as shown in Figure 5.2. When $C_{m-2} = 1$, $P_{m-2} = 1$, and $P_{m-1} = 1$, the approximate result will be $\{C_{MSP}S_{m-1}S_{m-2}\}$ = 011, only 1 less than the accurate result. For all other inputs, it will generate the accurate result.

$$S_{m-2} = (P_{m-2} \oplus C_{in}\ ) + (P_{m-1}C_{m-2}) \tag{5.5}$$

$$S_{m-1} = (P_{m-1} \oplus G_{m-2}) + (P_{m-2}C_{m-2}) \tag{5.6}$$

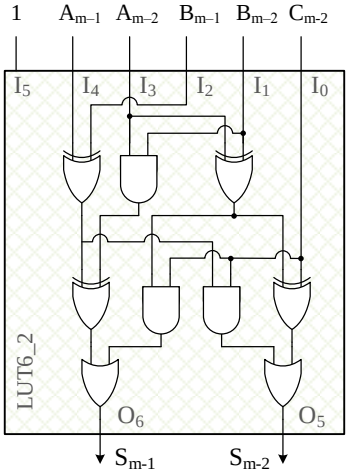


Figure 5.2 Proposed 2-bit approximate adder (AAd1) used in MSBs of LSP.

For uniformly distributed inputs, the carry-in has equal probability of being 1 or 0. The probability of inputs at bit position i propagating a carry is $P_i = 1/2$. Therefore, in the proposed n-bit approximate adders, the probability of S_{m-2} and S_{m-1} generating an error is 0.125 as shown in equation (5.7). Throughout this chapter, E_x represents the cases when hardware x generates an error.

$$\begin{aligned} Pr[E_{AAd1}] &= Pr[C_{m-2} \wedge P_{m-2} \wedge P_{m-1}] \\ &= \frac{1}{2} \times \frac{1}{2} \times \frac{1}{2} = 0.125 \end{aligned} \tag{5.7}$$

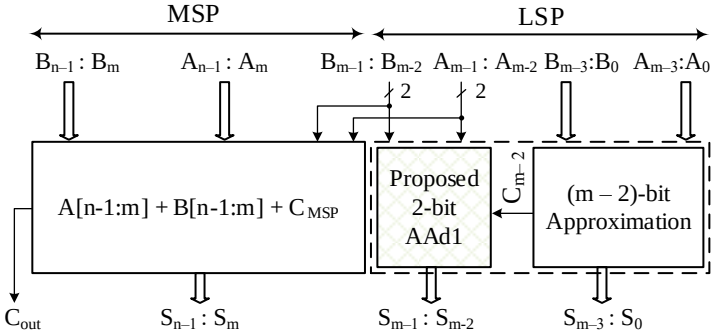


Figure 5.3 Architecture of proposed approximate adders for FPGAs.

Architecture of the proposed approximate adders is shown in Figure 5.3. It uses 2 MSBs of LSP to predict the C_{MSP}, whereas their respective sum bits are computed using AAd1. AAd1 is only suitable when the C_{out} of 2-bit inputs is predicted accurately. Accurate prediction of C_{out} requires additional resources or unused LUT inputs. Therefore, to design area efficient approximate adders for FPGAs, AAd1 is not used in the least significant $m-2$ bits of the LSP. In this chapter, we propose two n-bit approximate adders using the architecture in Figure 5.3. The two proposed n-bit approximate adders use different approximate functions for the first $m-2$ bits of the LSP.

5.2.1 Proposed low error and area efficient approximate adder

In this section, we propose a low error and area efficient approximate adder (LEADx) for FPGAs. State-of-the-art FPGAs use 6-input LUTs. These LUTs can be used to implement two 5-input functions. The complexity of the implemented logic function does not affect performance of LUT based implementation. A 2-bit adder has 5 inputs and two outputs. Therefore, a LUT can be used to implement a 2-bit approximate adder.

For an area efficient FPGA implementation, we propose to split the first $m-2$ bits of LSP into $\lceil (m-2)/2 \rceil$ groups of 2-bit inputs such that each group is mapped to a single LUT. Each group adds two 2-bit inputs with carry-in using an approximate 2-bit adder (AAd2).

To eliminate the carry chain in LSP, we propose to equate C_{out} of ith group to one of the inputs of that group (A_{i+1}). This results in error in 8 out of 32 possible cases with an absolute error magnitude of 4 in each erroneous case. To reduce the error magnitude, we propose to compute the S_i and S_{i+1} output bits as follows:

- If the C_{out} is predicted correctly, the sum outputs are also calculated accurately using standard 2-bit addition.
- If the C_{out} is predicted incorrectly and the predicted value of C_{out} is 0, both sum outputs are set to 1.
- If the C_{out} is predicted incorrectly and the predicted value of C_{out} is 1, both sum outputs are set to 0.

This modification reduces the absolute error magnitude to 2 in two cases, and to 1 in the other six cases. The resulting truth table of AAd2 is given in Table 5.3. The error cases

are shown in red. Since AAd2 produces an erroneous result in 8 out of 32 cases, the error probability of AAd2 is 0.25 as shown in equation (5.8).

Table 5.3 Truth Table of Proposed 2-BIT Approximate Adder (AAd2) used for Approximation in least-significant m–2 bits of LEADx

A_{i+1}	A_i	B_{i+1}	B_i	C_{in}	C_{i+2}	S_{i+1}	S_i
0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	1
0	0	0	1	0	0	0	1
0	0	0	1	1	0	1	0
0	0	1	0	0	0	1	0
0	0	1	0	1	0	1	1
0	0	1	1	0	0	1	1
0	0	1	1	1	0	1	1
0	1	0	0	0	0	0	1
0	1	0	0	1	0	1	0
0	1	0	1	0	0	1	0
0	1	0	1	1	0	1	1
0	1	1	0	0	0	1	1
0	1	1	0	1	0	1	1
0	1	1	1	0	0	1	1
0	1	1	1	1	0	1	1
1	0	0	0	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	0	1	0	0
1	0	0	1	1	1	0	0
1	0	1	0	0	1	0	0
1	0	1	0	1	1	0	1
1	0	1	1	0	1	0	1
1	0	1	1	1	1	1	0
1	1	0	0	0	1	0	0
1	1	0	0	1	1	0	0
1	1	0	1	0	1	0	0
1	1	0	1	1	1	0	1
1	1	1	0	0	1	0	1
1	1	1	0	1	1	1	0
1	1	1	1	0	1	1	0
1	1	1	1	1	1	1	1

$$Pr[E_{AAd2}] = 0.25 \tag{5.8}$$

The proposed LEADx approximate adder is shown in Figure 5.4. An n-bit LEADx uses $\lceil (m-2)/2 \rceil$ copies of AAd2 adder in the least significant $m-2$ bits of the approximate adder architecture shown in Figure 5.4. In LEADx, $C_{m-2} = A_{m-3}$. AAd2 implements a 5-to-2 logic function that is mapped to a single LUT. Similarly, AAd1 is also mapped to a single LUT. Therefore, $\lceil m/2 \rceil$ LUTs are used for the LSP. These LUTs work in parallel. Therefore, the delay of LSP is equal to the delay of a single LUT (t_{LUT}). The critical path of LEADx is from the input A_{m-2} to the output S_{n-1}.

Figure 5.5 shows an example of the functionality of 16-bit LEADx with 8-bit approximation. The outputs of bits enclosed in dotted lines are computed using AAd1. The outputs of the other bits of the approximate part (LSP) are computed using three copies of AAd2. The carry-in to the accurate part (C_{MSP}) is predicted from the two MSBs of LSP as shown in equation (5.4).

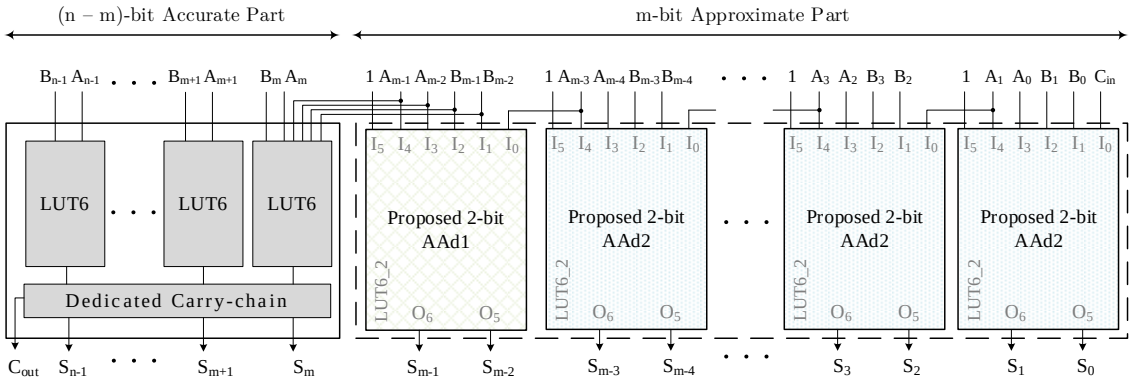


Figure 5.4 Proposed n-bit low error and area efficient approximate adder (LEADx).

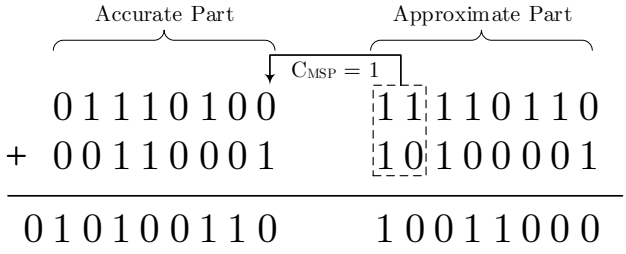


Figure 5.5 Example of 16-bit LEADx with 8-bit approximation.

The error probability of n-bit LEADx depends on the number of approximate 2-bit adders used in the approximate part. Error in any of these 2-bit adders can contribute to the error in the sum output. Therefore, the error probability of LEADx is given as the union of error probabilities of the individual 2-bit approximate adders. Let there be N copies of AAd2 in LEADx and E_{AAd2-i} represents the error in ith copy of AAd2, then the error probability of LEADx can be calculated as shown in equation (5.9).

$$Pr[E_{LEADx}] = Pr[E_{AAd1} \vee E_{AAd2-1} \vee E_{AAd2-2} \ldots \vee E_{AAd2-N}] \tag{5.9}$$

Since error in two or more of these 2-bit adders can occur concurrently, occurrence of error in these adders are not mutually exclusive. Therefore, equation (5.9) can be evaluated using inclusion-exclusion principle [79]. For example, the error probability of LEADx with 4-bits LSP, for uniformly distributed inputs, can be calculated as shown in equation (5.10).

$$\begin{aligned} Pr\left[E_{LEADx|m=4}\right] &= Pr[E_{AAd1} \vee E_{AAd2}] \\ &= Pr[E_{AAd1}] + Pr[E_{AAd2}] - Pr[E_{AAd1} \wedge E_{AAd2}] \\ &= 0.125 + 0.25 - (0.125 \times 0.25) \\ &= 0.34375 \end{aligned} \tag{5.10}$$

Similarly, the error probability of LEADx with 6-bits LSP, for uniformly distributed inputs, can be calculated as shown in equation (5.11).

$$\begin{aligned} Pr\left[E_{LEADx|m=6}\right] &= Pr[E_{AAd1} \vee E_{AAd2} \vee E_{AAd2}] \\ &= Pr[E_{AAd1}] + Pr[E_{AAd2}] + Pr[E_{AAd2}] - Pr[E_{AAd1} \wedge E_{AAd2}] \\ &- Pr[E_{AAd1} \wedge E_{AAd2}] - Pr[E_{AAd2} \wedge E_{AAd2}] + Pr[E_{AAd1} \wedge E_{AAd2} \wedge E_{AAd2}] \\ &= 0.125 + 0.25 + 0.25 - (0.125 \times 0.25) - (0.125 \times 0.25) - (0.25 \times 0.25) \\ &+ (0.125 \times 0.25 \times 0.25) \\ &= 0.50781 \end{aligned} \tag{5.11}$$

5.2.2 Proposed area and power efficient approximate adder for FPGAs

In this section, we propose an area and power efficient approximate adder (APEx) for FPGAs. APEx is also based on the approximate adder architecture shown in Figure 5.3. For the least significant $m-2$ bits of the LSP, the aim is to find an approximate function with no data dependency. Carry should neither be generated nor used for sum computation. A 1-bit input pair at any bit position $i \leq (m-2)$ should produce a 1-bit sum output only.

In general, any logic function with 1-bit output can be used as an approximate function to compute the approximate sum of 1-bit inputs at ith bit position. A constant 0 or constant 1 at the output are also valid approximate functions. Fixing the output to 0 or 1 will reduce the area and power consumption of the approximate adder because no hardware will be required for sum computation.

We evaluated error metrics of both constant functions for 1-bit addition, as shown in Table 5.4. Fixing the output to 0 introduces error in 3 out of 4 cases with an average error (AE) of -1 and MSE of ⅗ for uniformly distributed inputs. Fixing the output to 1 introduces error in 2 out of 4 cases with 0 AE and MSE of ½ for uniformly distributed inputs. Therefore, constant 1 provides a better approximation.

Table 5.4 Error Characterization of Constant Approximate Functions for 1-Bit Addition

A	B	Accurate Addition	Constant 1		Constant 0	
			Sum	Error	Sum	Error
0	0	00	1	1	0	0
0	1	01	1	0	0	-1
1	0	01	1	0	0	-1
1	1	10	1	-1	0	-2
Error Cases			2		3	
Average Error			0		-1	
Mean Square Error			½		⅗	

We further analyze the error metrics of n-bit approximate adder architecture shown in Figure 5.3 when approximate constant functions are used in the least significant $m-2$ bits of its LSP. If the least significant $m-2$ bits are fixed to 0, the maximum error (ME) occurs when the inputs A_0 to A_{m-3} and B_0 to B_{m-3} are all 1. With accurate addition, S_1 to S_{m-3} output bits are all 1 and a carry is propagated to $m-2$ bit position. Fixing S_0 to S_{m-3} to 0 and carry-in for $m-2$ bit position to 0 results in ME of $2^{m-1}-2$.

If the least significant $m-2$ bits are fixed to 1, the ME occurs when the inputs A_0 to A_{m-3} and B_0 to B_{m-3} are all 0. With accurate addition, S_0 to S_{m-3} output bits are all 0 and carry is not propagated to $m-2$ bit position. Fixing S_0 to S_{m-3} to 1 and carry-in for $m-2$ bit position to 0 results in ME of $2^{m-2}-1$. The ME of constant 1 is less than the ME of constant 0.

Furthermore, assume that a constant value V is used to approximate the function $F = A + B$. The resulting absolute error is defined as $|F - V|$. The aim is to find a constant value V such that MSE is minimized. This is a well-known problem with a well-defined solution: using mean of distribution of F as V minimizes the MSE [80,81].

Let us consider that A and B have uniform input distribution with values between 0 and 2^n-1 , then F has a symmetric triangular distribution in the range [0, $(2^{n+1}-2)$]. In the case of symmetric distribution, the mean and median are the same and located at the center of the sample space [82]. Therefore, mean and median of F are located at 2^n-1, which is the halfway point of [0, $2^{n+1}-2$]. The binary representation of 2^n-1, in $n+1$ bit sample space, is 0111…1. Therefore, using constant 1 as the sum output and 0 as carry-out minimizes the MSE of the approximate output. If i bits are fixed to 1, the probability of error in the sum output is calculated as shown in equation (5.12).

$$Pr[E_{const1}] = \frac{2^i - 1}{2^i} \tag{5.12}$$

In the proposed APEx, the S_0 to S_{m-3} outputs are fixed to 1 and the C_{m-2} is 0. This provides significant area and power consumption reduction at the expense of slight quality loss. It is important to note that this is different from bit truncation technique which fixes both the sum and carry outputs to 0. The ME of truncate adder is $2^{m+1}-2$ which is much higher than ME of APEx ($2^{m-2}-1$).

The proposed APEx approximate adder is shown in Figure 5.6. Same as LEADx, the critical path of APEx is from the input A_{m-2} to the output S_{n-1}. Similar to (9), the error probability of APEx can be calculated as shown in equation (5. 13).

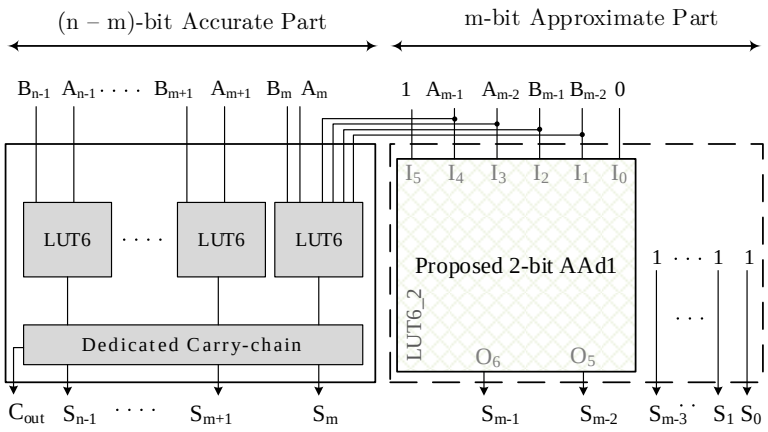


Figure 5.6 Proposed n-bit area and power efficient approximate adder (APEx).

When C_{m-2} is 0, E_{AAd1} reduces to 0 according to equation (5.7). Therefore, the error probability of APEx depends only on the number of output bits fixed to 1.

$$Pr[E_{APEx}] = Pr\left[E_{AAd1\,|\,C_{m-2}=0} \vee E_{Const1|i=m-2}\right] = \frac{2^{m-2}-1}{2^{m-2}} \tag{5.13}$$

Figure 5.7 shows an example of the functionality of 16-bit APEx with 8-bit approximation. The outputs of the bits enclosed by dotted lines are computed using AAd1. The outputs of the other bits of the approximate part (LSP) are fixed to 1. The carry-in to the accurate part (C_{MSP}) is predicted from the two MSBs of LSP as shown in equation (5.4).

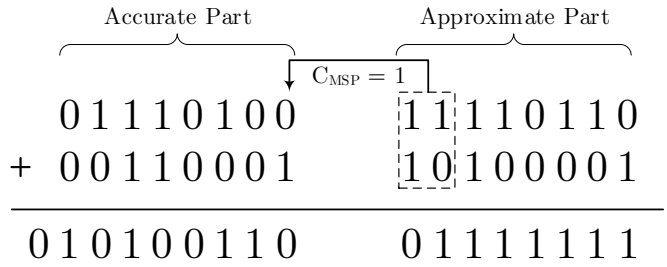


Figure 5.7 Example of 16-bit APEx with 8-bit approximation.

5.3 Experimental Results and Discussion

In this section, we present experimental results of the proposed approximate adders, LEADx and APEx. We compare LEADx and APEx with other FPGA-specific

approximate adders in the literature: LBA [70], DeMAS [71], and SEDA [73]. DeMAS can be built using different configurations. For a given number of approximate bits, each of these configurations has the same area. Therefore, we chose the configuration with the lowest average error for comparison.

We also compare LEADx and APEx with power and area efficient ASIC-based approximate adders in the literature: AFA [68], HOAANED [67], and LOA [16]. Each of these approximate adders is based on the approximate adder architecture shown in Figure 5.1, where approximation is done only in the LSP and the MSP is kept accurate. We also compare the proposed approximate adders with the segmented and speculative approximate adders in the literature.

5.3.1 Error Metrics

The functional models of these approximate adders are implemented in C++. Error metrics of these approximate adders are determined using their functional models for 16, 32, and 64-bit addition, with varying number of approximate bits, using 10^7 uniform random numbers as inputs.

The error value for each input is calculated by subtracting the accurate result from the approximate result. Error value may be positive, negative, or zero. The average error (AE) is defined as the average of all the error values. MAE, also known as mean error distance [33], is the average of the absolute values of all the error values. MAE is always positive. MSE is the average of the squares of all the error values. RMSE is the square root of MSE.

The MAE and MSE of LEADx can be calculated using equation (5.14) and equation (5.15), respectively. Similarly, the MAE and MSE of APEx can be calculated using equation (5.16) and equation (5.17), respectively. An empirical approach is used to determine these mathematical models, i.e., these formulas are determined using experimental results.

$$MAE_{LEADx} = \left(\frac{3}{16} \times 2^{m-2}\right) + 2^{m-9} \tag{14}$$

$$MSE_{LEADx} \approx 2^{2m-7} + 2^{2m-11} - 2^{1.5m-9.2} \tag{15}$$

$$MAE_{APEx} = \frac{2^{m-2}}{3} \tag{16}$$

$$MSE_{APEx} \approx \frac{5}{24} \times 4^{m-2} \tag{17}$$

As can be observed in these equations, error metrics of the proposed approximate adders depend only on the number of approximate bits (m), and they are independent of the bit width (n) of the adder.

Error metrics and the error distribution of 16-bit approximate adders with 8-bit approximation are shown in Figure 5.8. The error distribution is plotted as a function of error value and its respective percentage occurrence. As can be seen in Figure 5.8, the maximum errors of the proposed approximate adders are less than those of other approximate adders.

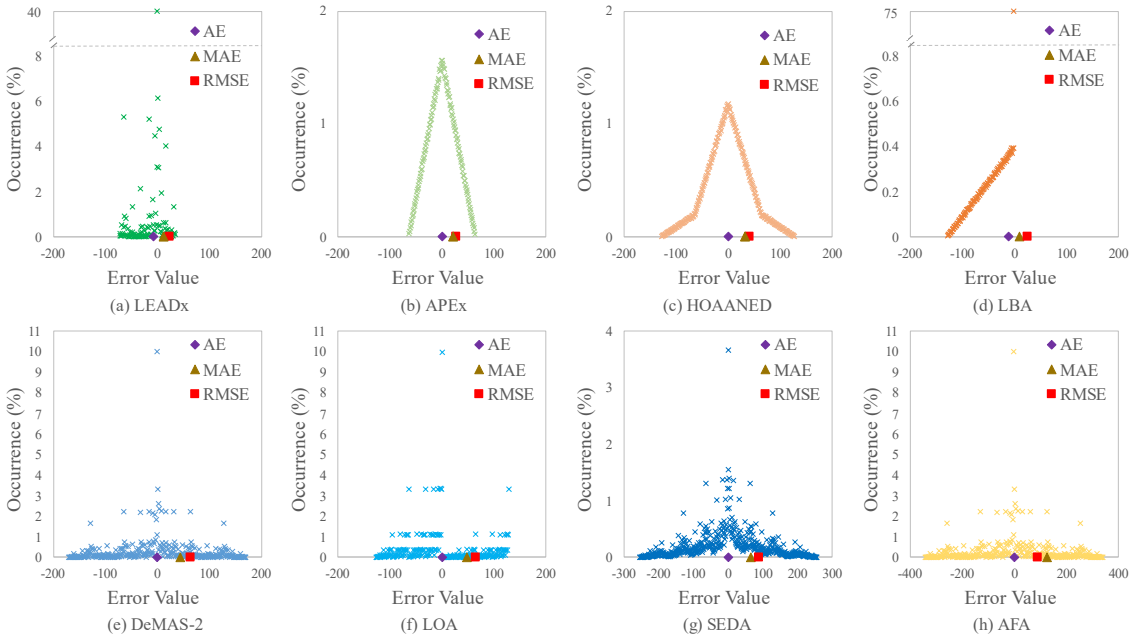


Figure 5.8 Error distribution and error metrics of 16-bit approximate adders with 8-bit approximation.

The error distribution of LEADx is skewed to the negative side. This indicates that, in most of the cases, the result of LEADx is less than the accurate result, leading to a negative AE. Whereas, plotting the error distribution of APEx results in a symmetrical triangular shape centered at zero, indicating that APEx has equal probability of negative and positive errors. Therefore, APEx has almost zero AE.

The error distribution of LBA indicates that its erroneous output is always less than the accurate result. All other approximate adders in the literature have almost symmetrical error distribution. However, their error values are spread over a wide range, resulting in much larger MAE and MSE as compared to the proposed approximate adders.

The error metrics of 64-bit adders with 4 to 12-bits of approximation are reported in Table 5.5. Our proposed approximate adders have the lowest MSE. The MSE of the LEADx is at least 20% less than that of the approximate adders in the literature.

Table 5.5 Error Metrics of 64-Bit Approximate Adders

	Adder	Approximate Bits				
		4	6	8	10	12
MSE ($x10^2$)	LEADx	0.019	0.333	5.43	87.10	1392
	APEx	0.025	0.425	6.83	109.09	1746
	AFA [68]	0.639	10.240	163.62	2620.20	41894
	DeMAS–2 [71]	0.160	2.560	40.91	655.05	10473
	HOAANED [67]	0.065	1.066	17.10	273.14	4363
	LBA [70]	0.026	0.428	6.84	110.07	1751
	LOA [16]	0.159	2.560	41.00	656.66	10494
	SEDA [73]	0.303	4.920	78.56	1257.80	20130
MAE	LEADx	0.69	3.08	12.56	50.41	201
	APEx	1.25	5.31	21.33	85.27	341
	AFA [68]	5.51	22.35	89.54	358.27	1432
	DeMAS–2 [71]	2.75	11.17	44.77	179.14	716
	HOAANED [67]	1.94	7.98	32.02	128.02	511
	LBA [70]	0.66	2.67	10.68	42.91	171
	LOA [16]	2.87	11.88	47.92	192.16	768
	SEDA [73]	4.05	16.47	65.99	264.10	1056
ER (%)	LEADx	34.44	50.83	63.10	72.31	79.26
	APEx	74.99	93.75	98.44	99.61	99.91
	AFA [68]	68.30	82.14	89.97	94.38	96.83
	DeMAS–2 [71]	68.31	82.14	89.97	94.39	96.84
	HOAANED [67]	81.26	95.34	98.82	99.71	99.93
	LBA [70]	21.88	24.31	24.85	25.04	25.04
	LOA [16]	68.35	82.21	90.01	94.37	96.82
	SEDA [73]	80.85	91.61	96.33	98.40	99.29

LBA has the lowest MAE. However, it has the worst area and power consumption results, as reported in the next section. The MAE of the proposed approximate adders is second only to that of LBA. The ER of LEADx and APEx validate the analytical error probability results given in Section II. All the approximate adders, except LBA and LEADx, have high ER.

These adders follow the fail-small approach [76]. In the fail-small approach, even if ER is high, error magnitudes are small. The rationale behind this approach is that small errors are naturally masked by algorithms, and they have less impact on MSE. Therefore, they slightly degrade the quality of applications.

The error magnitude of our proposed approximate adders is significantly reduced by accurately predicting the carry to the MSP using unused LUT inputs. AAd1 and AAd2, both fully utilize the LUT inputs to achieve low error. The LEADx is designed in a way that not only the error values are reduced but also the number of error cases are reduced.

The experimental results show that LEADx has indeed higher accuracy and lower MSE than the other approximate adders. Similarly, the logic function of the approximate part of APEx is determined to reduce the MSE. The experimental results show that the MSE of APEx is indeed less than that of the approximate adders in the literature.

5.3.2 Implementation Results

All the approximate adders are implemented using Verilog HDL. The accurate part of all the adders is identical and implemented using addition operator. Verilog RTL codes are synthesized and implemented on a Xilinx Virtex 7 FPGA with speed grade 3 using Vivado 2020.1. AreaOptimized_high strategy is used for synthesis, and default strategy is used for implementation.

The quality metrics are extracted from post-implementation timing simulations using 1 million uniform random numbers. The quality metrics are cross verified with C++ simulations. For power estimation, switching activity interchange format (SAIF) files are also generated from these post-implementation timing simulations at 100 MHz for all adders. The power consumption of each approximate adder FPGA implementation is estimated with Vivado 2020.1 using the corresponding SAIF file.

The implementation results of 16-bit adders with 8-bit approximation are given in Table 5.6. All the adders are implemented with input and output registers. SEDA and LBA are slower than the accurate adder because of carry propagation in their LSPs. All other 16-bit approximate adders have the same delay as the accurate adder. It is important to note that their delay is limited by the maximum frequency of Virtex 7 FPGA. It does not necessarily mean that the critical path of these adders is the same.

Table 5.6 FPGA Implementation Results of 16-Bit Adders with 8-Bit Approximation

Adder	LUTs		Delay (ns)	Power (mW)
	MSP	LSP		
Accurate	8	8	1.35	6.16
LEADx	8	4	1.35	6.09
APEx	8	1	1.35	4.32
AFA [68]	8	5	1.35	6.17
DeMAS–2 [71]	8	5	1.35	6.12
HOAANED [67]	8	1	1.35	4.52
LBA [70]	8	11	1.56	6.10
LOA [16]	8	4	1.35	5.66
SEDA [73]	8	6	1.35	6.16

All the approximate 16-bit adders, except LBA, use fewer LUTs than the accurate adder. Since an accurate adder is used in the MSP of all these adders, the reduction in LUTs occurs only in the LSP. Since LEADx performs 2-bit addition in a single LUT, its LSP uses 50% fewer LUTs than the accurate adder.

APEx and HOAANED use the lowest number of LUTs. For these two adders, a significant reduction in number of LUTs occurs because of the use of constant functions in their LSPs. For other approximate adders, the reduction in number of LUTs occurs because of the approximation techniques used, which allow the synthesis tool to merge two sum outputs to a single LUT.

LEADx consumes slightly less power than the accurate adder. APEx consumes the lowest power among all the approximate adders. For the 16-bit adder with 8-bit approximation, the power consumption of APEx is 29% less than that of the accurate adder and 4.5% less than that of the second lowest power consuming adder, HOAANED.

The LUTs vs MSE and power vs MSE graphs of 32-bit approximate adders are given in Figure 5.9. These results are plotted for 4-bit to 20-bit approximation in a 32-bit adder. The 32-bit accurate adder uses 32 LUTs and consumes 10.75 mW power.

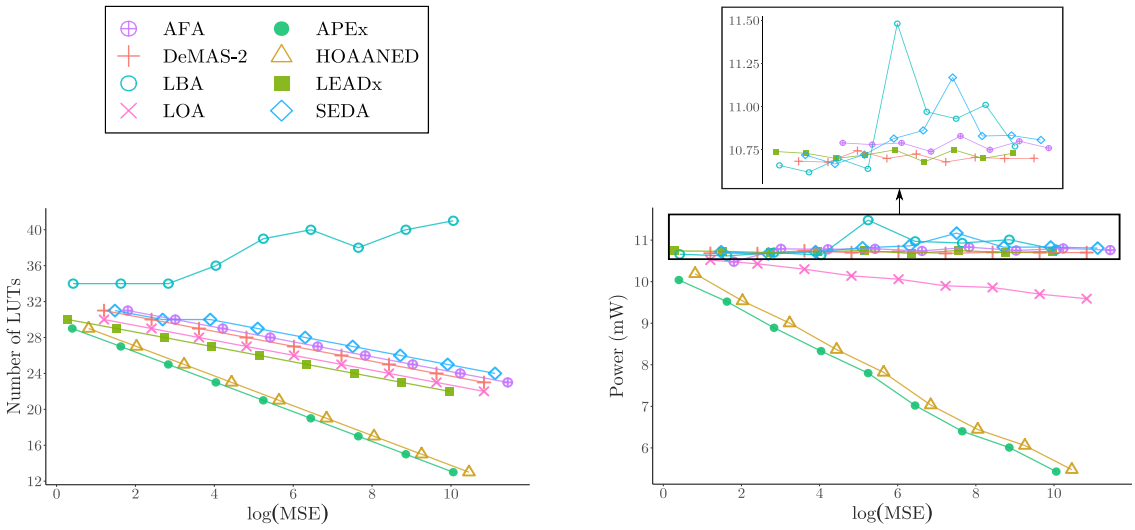


Figure 5.9 Comparison of 32-bit approximate adders with 4-bit to 20-bit approximation (left to right). (a) LUTs vs MSE. (b) Power vs MSE.

While the number of LUTs used by most of the approximate adders decreases linearly with the increase in approximation, their respective power reductions do not follow the same trend. However, APEx provides significant power reduction compared to the accurate adder at the cost of a slight loss in accuracy.

LUTs, power consumption and delay reductions achieved by 64-bit approximate adders with 16-bit approximation compared to 64-bit accurate adder are shown in Figure 5.10. LEADx reduced the LUTs by 12.5% compared to the accurate adder. APEx reduced the LUTs by 23.4% and power consumption by 21% compared to the accurate adder.

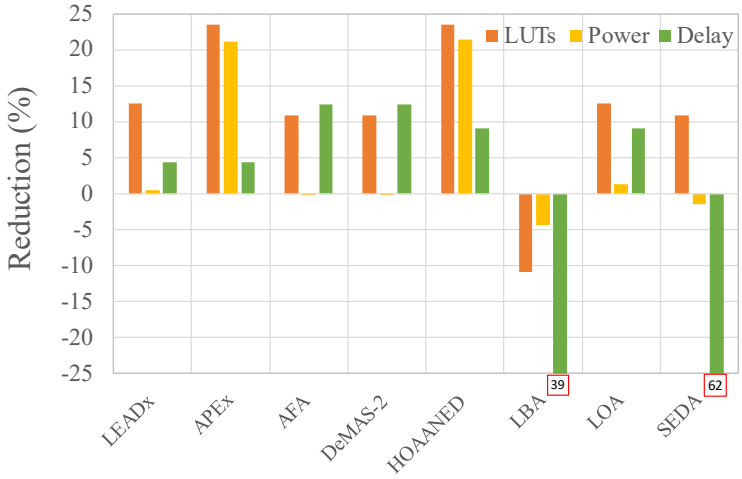


Figure 5.10 Area, Power, and Delay reduction achieved with 16-bit approximation in 64-bit approximate adders compared to 64-bit accurate adder**.**

LBA performs worse than the accurate adder in all these metrics. Among other FPGA specific adders, DeMAS provided no power reduction but reduced the LUTs by 11% compared to the accurate adder. The performance of HOAANED is compatible with APEx. However, as discussed earlier, it has lower quality than both LEADx and APEx.

These results show that our proposed LEADx has smaller area, lower power, and better quality than the FPGA specific adders in the literature. The results show that DeMAS is the most efficient FPGA specific approximate adder in the literature. With 8-bits approximation, LEADx has 7% smaller area and 86% lower MSE than DeMAS. LOA is one of the most efficient ASIC-based approximate adders in the literature [17]. LEADx has better quality than LOA at the same cost when implemented on an FPGA. With 8-bits approximation, LEADx has 87% lower MSE than LOA at the same cost. HOAANED is suitable for FPGA implementation. However, APEx has less power and better quality than HOAANED at the same cost, when implemented on an FPGA. APEx has more than 60% lower MSE than HOAANED at the same cost.

5.3.3 Comparison with Segmented and Speculative Approximate Adders

In this section, we compare the proposed approximate adders with segmented and speculative adders in the literature; Almost Correct Adder (ACA-I) [62], Accuracy

Configurable Adder (ACA-II) [64], Block-based Carry Speculative Adder (BCSA) [75], Error-tolerant adder II (ETA-II) [63], and xUAV [74].

The quality and implementation results of 16-bit adders with different approximation amounts are given in Table 5.7. These adders have same delay (1.35 ns). These adders are implemented with input and output registers. Therefore, although their critical paths are different, their speed is limited by the maximum frequency supported by Virtex 7 FPGA.

Table 5.7 Comparison of 16-Bit Proposed Approximate Adders with 16-Bit Segmented and Speculative Approximate Adders

Adder	m*	r*	LUTs	Power (mW)	ME	RMSE	ER (%)
LEADx	4	–	14	6.13	4	1.39	34.88
	8	–	12	6.09	72	23.29	63.15
APEx	4	–	13	5.45	3	1.58	74.90
	8	–	9	4.32	63	26.13	98.04
ACA-I [62]	4	1	29	6.25	34944	6702	34.13
	8	1	72	6.78	32768	1689	1.59
ACA-II [64]	4	2	22	6.29	17472	5232	47.88
	8	4	24	6.23	4096	703	5.90
BCSA [75]	4	4	24	6.42	4368	1029	6.20
	8	8	16	6.12	256	63	17.04
ETA-II [63]	4	2	22	6.30	17472	5232	47.88
	8	4	29	6.20	4096	703	5.90
xUAV [74]	3	1	16	6.24	37448	9615	61.84
	5	1	26	6.42	33824	4754	16.72

*m is the size of approximate part (LSP) of LEADx and APEx. For other adders, m is the segment size. For segmented and speculative adders, r is the number of resultant bits contributing to the final sum from each segment.

xUAV is an FPGA-specific segmented adder. Several configurations of xUAV are proposed in [74]. We used two most efficient configurations; one with the lowest error (m = 5, r = 1) and the other with low error and low area (m = 3, r = 1).

The segmented and speculative adders follow fail-rare approach [76]. They have low ER. But their error magnitudes are usually large. Therefore, these adders have high MAE and MSE. For example, ACA-I with 8-bit segmentation has only 1.5% ER. However, its ME is 2^15. Most of the errors that occur in ACA-I have large magnitude, resulting in significantly high MSE.

Among the segmented and speculative adders, BCSA with 8-bit segmentation has the best quality. ETA-II and ACA-II have similar architecture. Therefore, their error metrics are similar. However, for 8-bit segmentation, ACA-II is more area efficient than ETA-II.

LEADx and APEx have better quality, smaller area, and lower power consumption than the segmented and speculative adders. These results show that, for uniformly distributed inputs, fail-small approach gives better quality than fail-rare approach.

5.3.4 Case Study: Motion Estimation in Video Encoding

We also assessed the impact of the proposed approximate adders and the other approximate adders on video encoding quality. C++ implementations of 8-bit adders with 4-bit approximation are integrated into High Efficiency Video Coding (HEVC) reference software HM 16.14 video encoder.

The approximate adders are used for sum of absolute difference (SAD) computations for motion estimation (ME). ME accounts for approximately 70% of the computational complexity of video encoding [13]. The search strategy is set to fast test zone search (TZ). The quality results are obtained for four video sequences with different spatial resolutions.

For each approximate adder, PSNR result in dB and the percentage increase in bitrate (ΔBR) with respect to using accurate adder are shown in Table 5.8. LEADx has the least quality loss, i.e., lowest PSNR decrease and lowest bitrate increase, compared to the other approximate adders.

Table 5.8 Impact of Approximate Adders on HEVC Encoder Bitrate And PSNR

Adder	Video Sequence							
	Traffic (2560x1600)		BQ Terrace (1920x1080)		Four People (1280x720)		Party Scene (832x480)	
	ΔBR (%)	PSNR (dB)	ΔBR (%)	PSNR (dB)	ΔBR (%)	PSNR (dB)	ΔBR (%)	PSNR (dB)
Accurate	–	37.35	–	34.69	–	39.58	–	33.44
LEADx	3.93	37.05	1.49	34.51	2.07	39.38	1.86	33.29
APEx	4.33	37.03	1.98	34.50	2.08	39.38	1.98	33.28
AFA [68]	4.46	36.89	2.28	34.50	2.55	39.34	2.90	33.24
DeMAS–2 [71]	3.98	37.06	2.64	34.50	2.20	39.29	1.87	33.29
HOAANED [67]	10.70	36.76	3.54	34.44	4.68	39.28	4.37	33.18
LBA [70]	11.35	36.77	3.57	34.44	3.89	39.27	4.49	33.18
LOA [16]	11.78	36.76	3.77	34.45	3.63	39.30	3.89	33.19
SEDA [73]	11.61	36.75	3.66	34.44	2.87	39.27	4.26	33.15

Chapter 6

CONCLUSIONS

In this thesis, efficient ME hardware for HEVC and VVC standards are proposed. The proposed VVC ME hardware is the first VVC ME hardware in the literature. We proposed an approximate adder suitable for SAD calculation in ME. We analyzed the impact of approximate circuits on the performance and quality of HEVC ME. We proposed a methodology to design approximate adders for FPGAs. Two approximate adders for FPGAs, one targeting high quality and the other targeting low area and power, are designed using the proposed methodology. We also proposed a novel low error approximate SAD hardware for FPGAs. It has the lowest area and consumes the lowest power among the approximate and accurate SAD hardware in the literature.

As future work, the impact of using approximate adders in the SAD adder tree of ME hardware can be investigated. Run-time error detection and correction for the proposed approximate SAD hardware can be explored. Fast VVC ME algorithms and their efficient hardware implementations can be proposed.

BIBLIOGRAPHY

[1] Cisco Systems, "Cisco Visual Networking Index: Forecast and Trends, 2017-2022," Cisco Systems White Paper, 2018.

[2] B. Bross, J. Chen, J.-R. Ohm, G. J. Sullivan and Y.-K. Wang, "Developments in International Video Coding Standardization After AVC, With an Overview of Versatile Video Coding (VVC)," Proceedings of the IEEE, vol. 109, no. 9, pp. 1463 - 1493, 2021.

[3] "AdReaction - Video Creative in a Digital World," Millward Brown, 2016.

[4] "High Efficiency Video Coding," Recommendation ITU-T H.265 and ISO/IEC 23008-2 (HEVC), 2013.

[5] G. J. Sullivan, J.-R. Ohm, W.-J. Han and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard," IEEE Transactions on Circuits and Systems for Video Technology , vol. 22, no. 12, pp. 1649 - 1668, 2012.

[6] "Versatile Video Coding," Recommendation ITU-T H.266 and ISO/IEC 23090-3 (VVC), 2020.

[7] B. Bross, Y. Wang, Y. Ye, S. Liu, J. Chen, G. J. Sullivan and J. Ohm, "Overview of the Versatile Video Coding (VVC) Standard and its Applications," IEEE Transactions on Circuits and Systems for Video Technology, early access.

[8] Z. Pan, J. Lei, Y. Zhang, X. Sun and S. Kwong, "Fast Motion Estimation Based on Content Property for Low-Complexity H.265/HEVC Encoder," IEEE Transactions on Broadcasting, vol. 62, no. 3, pp. 675 - 684, 2016.

[9] F. Pakdaman, M. A. Adelimanesh, M. Gabbouj and M. R. Hashemi, "Complexity Analysis Of Next-Generation VVC Encoding And Decoding," in IEEE International Conference on Image Processing (ICIP), Abu Dhabi, UAE, 2020.

[10] J. Chen, M. Karczewicz, Y.-W. Huang, K. Choi, J.-R. Ohm and G. J. Sullivan, "The Joint Exploration Model (JEM) for Video Compression With Capability Beyond HEVC," IEEE Transactions on Circuits and Systems for Video Technology, vol. 30, no. 5, pp. 1208 - 1225, May 2020.

[11] Q. Xu, T. Mytkowicz and N. S. Kim, "Approximate Computing: A Survey," IEEE Design & Test, vol. 33, no. 1, pp. 8 - 22, 2015.

[12] S. Froehlich, D. Große and R. Drechsler, "Towards Reversed Approximate Hardware Design," in Euromicro Conference on Digital System Design (DSD), Prague, 2018.

[13] T. Arifeen, A. S. Hassan, H. Moradian and J. A. Lee, "Probing Approximate TMR in Error Resilient Applications for Better Design Tradeoffs," in Euromicro Conference on Digital System Design (DSD), Limassol, 2016.

[14] E. Kalali and I. Hamzaoglu, "Approximate HEVC Fractional Interpolation Filters and Their Hardware Implementations," IEEE Transactions on Consumer Electronics , vol. 64, no. 3, pp. 285 - 291, 2018.

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

[16] H. R. Mahdiani, A. Ahmadi, S. M. Fakhraie and C. Lucas, "Bio-Inspired Imprecise Computational Blocks for Efficient VLSI Implementation of Soft-Computing Applications," IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 57, no. 4, pp. 850-862, April, 2010.

[17] H. Jiang, C. Liu, L. Liu, F. Lombardi and J. Han, "A review, classification, and comparative evaluation of approximate arithmetic circuits," ACM Journal on Emerging Technologies in Computing Systems, vol. 13, no. 4, August 2017.

[18] M. Shafique, W. Ahmad, R. Hafiz and J. Henkel, "A low latency generic accuracy configurable adder," in ACM/EDAC/IEEE Design Automation Conference (DAC), San Francisco, 2015.

[19] Z.-L. He, C.-Y. Tsui, K.-K. Chan and M. Liou, "Low-power VLSI design for motion estimation using adaptive pixel truncation," IEEE Transactions on Circuits and Systems for Video Technology, vol. 10, no. 5, pp. 669 - 678, 2000.

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

[21] R. Porto, L. Agostini, B. Zatt, M. Porto, N. Roma and L. Sousa, "Energy-efficient motion estimation with approximate arithmetic," in IEEE International Workshop on Multimedia Signal Processing (MMSP), Luton, 2017.

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

[23] A. Akin, G. Sayilar and I. Hamzaoglu, "High performance hardware architectures for one bit transform based single and multiple reference frame motion estimation," IEEE Transactions on Consumer Electronics, vol. 56, no. 2, pp. 1144 - 1152, 2010.

[24] K. Singh and S. R. Ahamed, "Low Power Motion Estimation Algorithm and Architecture of HEVC/H.265 for Consumer Applications," IEEE Transactions on Consumer Electronics, vol. 64, no. 3, pp. 267 - 275, 2018.

[25] A. Akin, G. Sayilar and I. Hamzaoglu, "A reconfigurable hardware for one bit transform based multiple reference frame Motion Estimation," in Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, 2010.

[26] A. C. Mert, H. Azgin, E. Kalali and I. Hamzaoglu, "Novel Approximate Absolute Difference Hardware," in Euromicro Conference on Digital System Design (DSD), Kallithea, 2019.

[27] C. Kalaycioglu, O. C. Ulusel and I. Hamzaoglu, "Low power techniques for Motion Estimation hardware," in International Conference on Field Programmable Logic and Applications, Prague, 2009.

[28] E. Kalali and I. Hamzaoglu, "An Approximate HEVC Intra Angular Prediction Hardware," IEEE Access, vol. 8, pp. 2599 - 2607, 2020.

[29] S. Park and J. Kang, "Fast Affine Motion Estimation for Versatile Video Coding (VVC) Encoding," IEEE Access, vol. 7, pp. 158075 - 158084, 2019.

[30] J. Brandenburg, A. Wieckowski, T. Hinz, A. Henkel, V. George, I. Zupancic, C. Stoffers, B. Bross, H. Schwarz and D. Marpe, "Towards Fast and Efficient VVC Encoding," in IEEE International Workshop on Multimedia Signal Processing (MMSP), Tampere, Finland, 2020.

[31] M. Saldanha, M. C. Corrêa, D. Palomino, M. Porto, B. Zatt and L. Agostini, "An Overview of Dedicated Hardware Designs for State-of-the-Art AV1 and H.266/VVC Video Codecs," in IEEE Int. Conf. on Electronics, Circuits and Systems (ICECS), Glasgow, UK, 2020.

[32] W. Ahmad, B. Ayrancioglu and I. Hamzaoglu, "Low Error Efficient Approximate Adders for FPGAs," IEEE Access, vol. 9, pp. 117232 - 117243, 2021.

[33] A. Cerveira, L. Agostini, B. Zatt and F. Sampaio, "Memory Profiling of H.266 Versatile Video Coding Standard," in IEEE Int. Conf. on Electronics, Circuits and Systems (ICECS), Glasgow, UK, 2020.

[34] T. D'huys, S. Momcilovic, F. Pratas and L. Sousa, "Reconfigurable data flow engine for HEVC motion estimation," in IEEE Int. Conf. on Image Processing (ICIP), Paris, France, 2014.

[35] N. C. Vayalil and Y. Kong, "VLSI Architecture of Full-Search VariableBlock-Size Motion Estimation for HEVC Video," IET Circuits, Devices & Systems, vol. 11, no. 6, pp. 543-548, 2017.

[36] W. Ahmad, B. Ayrancioglu and I. Hamzaoglu, "Comparison of Approximate Circuits for H.264 and HEVC Motion Estimation," in Euromicro Conf. on Digital System Design (DSD), Kranj, Slovenia, 2020.

[37] Y. Tseng and C.-A. Shen, "The Design and Implementation of a Highly Efficient Motion Estimation Engine for HEVC Systems," in IEEE Int. Symp. on Circuits and Systems (ISCAS), Sapporo, Japan, 2019.

[38] R. Khemiri, H. Kibeya, H. Loukil, F. E. Sayadi, M. Atri and N. Masmoudi, "Real-time motion estimation diamond search algorithm for the new high efficiency video coding on FPGA," Analog Integrated Circuits and Signal Processing, vol. 94, p. 259–276, 2018.

[39] E. Alcocer, R. Gutierrez, O. LopezGranado and M. P. Malumbres, "Design and implementation of an efficient hardware integer motion estimator for an HEVC video encoder," J. of Real-Time Image Processing, vol. 16, p. 547–557, 2019.

[40] S. Gogoi and R. Peesapati, "A hybrid hardware-oriented motion estimation algorithm for HEVC/H.265," J. of Real-Time Image Processing, vol. 18, p. 953–966, 2021.

[41] B. Silveira, G. Paim, B. Abreu, M. Grellert, C. M. Diniz, E. A. C. d. Costa and S. Bampi, "Power-Efficient Sum of Absolute Differences Hardware Architecture Using Adder Compressors for Integer Motion Estimation Design," IEEE Trans. on Circuits and Systems I: Regular Papers, vol. 64, no. 12, pp. 3126 - 3137, 2017.

[42] W. Ahmad and I. Hamzaoglu, "An Efficient Approximate Sum of Absolute Differences Hardware for FPGAs," in IEEE Int. Conf. on Consumer Electronics (ICCE), Las Vegas, 2021.

[43] H. Azgin, E. Kalali and I. Hamzaoglu, "An Efficient FPGA Implementation of Versatile Video Coding Intra Prediction," in Euromicro Conf. on Digital System Design (DSD), Kallithea, Greece, 2019.

[44] H. Azgin, A. C. Mert, E. Kalali and I. Hamzaoglu, "A Reconfigurable Fractional Interpolation Hardware for VVC Motion Compensation," in Euromicro Conference on Digital System Design (DSD), Prague, Czech Republic, 2018.

[45] M. J. Garrido, F. Pescador, M. Chavarrías, P. J. Lobo, C. Sanz and P. Paz, "An FPGA-Based Architecture for the Versatile Video Coding Multiple Transform Selection Core," IEEE Access, vol. 8, pp. 81887 - 81903, 2020.

[46] A. Kammoun, W. Hamidouche, F. Belghith, J.-F. Nezan and N. Masmoudi, "Hardware Design and Implementation of Adaptive Multiple Transforms for the Versatile Video Coding Standard," IEEE Trans. on Consumer Electronics, vol. 64, no. 4, pp. 424 - 432, 2018.

[47] Y. Fan, J. Chen, H. Sun, J. Katto and M. Jing, "A Fast QTMT Partition Decision Strategy for VVC Intra Prediction," IEEE Access, vol. 8, pp. 107900 - 107911, 2020.

[48] Y. Huang, J. An, H. Huang, X. Li, S.-T. Hsiang, K. Zhang, H. Gao, J. Ma and O. Chubach, "Block Partitioning Structure in the VVC Standard," IEEE Trans. on Circuits and Systems for Video Technology, early access, doi: 10.1109/TCSVT.2021.3088134.

[49] W. J. Chien and e. al., "JVET AHG report: Tool reporting procedure (AHG13).," document JVET-S0013, Joint Video Experts Team (JVET), June 2020.

[50] S. Jung and D. Jun, "Context-Based Inter Mode Decision Method for Fast Affine Prediction in Versatile Video Coding," Electronics, vol. 10, no. 11, p. 1243, 2021.

[51] H. Azgin, E. Kalali and I. Hamzaoglu, "An Approximate Versatile Video Coding Fractional Interpolation Hardware," in IEEE International Conference on Consumer Electronics, Las Vegas, 2020.

[52] R. Porto, L. Agostini, B. Zatt, N. Roma and M. Porto, "Power-Efficient Approximate SAD Architecture with LOA Imprecise Adders," in IEEE 10th Latin American Symposium on Circuits & Systems, Armenia, Colombia, 2019.

[53] M. Kumm, M. Kleinlein and P. Zipf, "Efficient sum of absolute difference computation on FPGAs," in International Conference on Field Programmable Logic and Applications, Lausanne, 2016.

[54] S. Perri, P. Zicari and P. Corsonello, "Efficient Absolute Difference Circuits in Virtex-5 FPGAs," in IEEE Mediterranean Electrotechnical Conference, Valletta, 2010.

[55] L. D. T. Dang, N. T. M. Kieu, I. J. Chang and J. Kim, "Approximate-SAD Circuit for Power-efficient H.264 Video Encoding under Maintaining Output Quality and Compression Efficiency," Journal of Semiconductor Tech. and Science, vol. 16, no. 5, p. 605–614, 2016.

[56] Xilinx, "7 Series Configurable Logic Block," September 2016. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf.

[57] J. Echavarria, S. Wildermann, A. Becher, J. Teich and D. Ziener, "FAU: Fast and error-optimized approximate adder units on LUT-Based FPGAs," in International Conference on Field-Programmable Technology, Xian, 2017.

[58] G. A. Gillani, M. A. Hanif, B. Verstoep, S. H. Gerez, M. Shafique and A. B. J. Kokkeler, "MACISH: Designing Approximate MAC Accelerators With Internal-Self-Healing," IEEE Access, vol. 7, pp. 77142 - 77160, 2019.

[59] T. Ayhan and M. Altun, "Circuit Aware Approximate System Design With Case Studies in Image Processing and Neural Networks," IEEE Access, vol. 7, pp. 4726 - 4734, 2018.

[60] N. V. Toan and J.-G. Lee, "FPGA-Based Multi-Level Approximate Multipliers for High-Performance Error-Resilient Applications," IEEE Access, vol. 8, pp. 25481 - 25497, 2020.

[61] L. Chen, J. Han, W. Liu, P. Montuschi and F. Lombardi, "Design, Evaluation and Application of Approximate High-Radix Dividers," IEEE Trans. on Multi-Scale Computing Systems, vol. 4, no. 3, pp. 299 - 312, 2018.

[62] A. K. Verma, P. Brisk and P. Ienne, "Variable Latency Speculative Addition: A New Paradigm for Arithmetic Circuit Design," in Design, Automation and Test in Europe (DATE) Conf., Munich, 2008.

[63] N. Zhu, W. L. Goh, G. Wang and K. S. Yeo, "Enhanced low-power high-speed adder for error-tolerant application," in Int. SoC Design Conf., Incheon, 2010..

[64] A. B. Kahng and S. Kang, "Accuracy-configurable adder for approximate arithmetic designs," in Design Automation Conf. (DAC), New York, 2012.

[65] V. Gupta, D. Mohapatra, A. Raghunathan and K. Roy, "Low-Power Digital Signal Processing Using Approximate Adders," IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, vol. 32, no. 1, pp. 124-137, 2012.

[66] A. Dalloo, A. Najafi and A. Garcia-Ortiz, "Systematic Design of an Approximate Adder: The Optimized Lower Part Constant-OR Adder," IEEE Trans. on Very Large Scale Integration (VLSI) Systems, vol. 26, no. 8, pp. 1595-1599, 2018.

[67] P. Balasubramanian, R. Nayar, D. L. Maskell and N. E. Mastorakis, "An Approximate Adder With a Near-Normal Error Distribution: Design, Error Analysis and Practical Application," IEEE Access, vol. 9, pp. 4518 - 4530, 2020.

[68] S. Dutt, S. Nandi and G. Trivedi, "Analysis and Design of Adders for Approximate Computing," ACM Trans. on Embedded Computing Systems, vol. 17, no. 2, p. 40, 2017.

[69] C. Niemann, M. Rethfeldt and D. Timmermann, "Approximate Multipliers for Optimal Utilization of FPGA Resources," in Int. Symp. on Design and Diagnostics of Electronic Circuits & Systems (DDECS), Vienna, 2021.

[70] A. Becher, J. Echavarria, D. Ziener, S. Wildermann and J. Teich, "A LUT-Based Approximate Adder," in Int. Symp. on Field-Program. Custom Computing Machines (FCCM), Washington DC, 2016.

[71] B. S. Prabakaran, S. Rehman, M. A. Hanif, S. Ullah, G. Mazaheri, A. Kumar and M. Shafique, "DeMAS: An efficient design methodology for building approximate adders for FPGA-based systems," in Design, Automation & Test in Europe (DATE) Conf., Dresden, 2018.

[72] S. Boroumand, H. P. Afshar and P. Brisk, "Approximate quaternary addition with the fast carry chains of FPGAs," in Design, Automation & Test in Europe (DATE) Conf., Dresden, 2018.

[73] C. K. Jha, K. Prasad, A. S. Tomar and J. Mekie, "SEDAAF: FPGA Based Single Exact Dual Approximate Adders for Approximate Processors," in IEEE Int. Symp. on Circuits and Systems (ISCAS), Seville, 2020.

[74] T. Nomani, M. Mohsin, Z. Pervaiz and M. Shafique, "xUAVs: Towards Efficient Approximate Computing for UAVs—Low Power Approximate Adders With Single LUT Delay for FPGA-Based Aerial Imaging Optimization," IEEE Access, vol. 8, pp. 102982 - 102996, 2020.

[75] F. Ebrahimi-Azandaryani, O. Akbari, M. Kamal, A. Afzali-Kusha and M. Pedram, "Block-based carry speculative approximate adder for energy-efficient applications," IEEE Trans. on CAS II: Express Briefs, vol. 67, no. 1, pp. 137-141, 2020.

[76] V. K. Chippa, S. T. Chakradhar, K. Roy and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," in Design Automation Conf. (DAC), Austin, TX, 2013.

[77] Y. Wu, C. Shen, Y. Jia and W. Qian, "Approximate logic synthesis for FPGA by wire removal and local function change," in Asia and South Pacific Design Automation Conf. (ASP-DAC), Chiba, 2017.

[78] A. Ehliar, "Optimizing Xilinx designs through primitive instantiation," in FPGA World Conf., Copenhagen, 2010.

[79] A. Leon-Garcia, Probability, statistics, and random processes for electrical engineering, NJ, USA: Prentice-Hall, 2008.

[80] R. J. Hyndman and G. Athanasopoulos, Forecasting: principles and practice, 2nd edition, Melbourne, Australia: OTexts, 2018.

[81] E. T. Jaynes, Probability Theory: The Logic of Science, Cambridge, UK: Cambridge University Press, 2003.

[82] D. P. Doane and L. E. Seward, "Measuring Skewness: A Forgotten Statistic?," J. of Statistical Education, vol. 19, no. 2, pp. 1-18, 2011.